Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 319 (2015) 271–287

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Sequential Algorithms for Unbounded Nondeterminism

J. Laird[1]

Department of Computer Science, University of Bath, UK

Abstract

We give extensional and intensional characterizations of higher-order functional programs with unbounded nondeterminism: as stable and monotone functions between the biorders of states of ordered concrete data structures, and as *sequential algorithms* (states of an exponential ocds) which compute them. Our fundamental result establishes that these representations are equivalent, by showing how to construct a unique sequential algorithm which computes a given stable and monotone function.
We illustrate by defining a denotational semantics for a functional language with countable nondeterminism ("fair PCF"), with an interpretation of fixpoints which allows this to be proved to be computationally adequate. We observe that our model contains functions which cannot be computed in fair PCF, by identifying a further property of the definable elements, and so show that it is not fully abstract.

Keywords: Sequential Algorithms, Nondeterminism, Fairness, Biorders

1 Introduction

This paper develops a model of higher-order computation with unbounded nondeterminism. In this setting we may write programs which will always return a value but may take an unbounded number of steps to do so, corresponding to the notion of *fairness* [6]. A major challenge for capturing such programs is that they do not correspond to continuous functions, in general. In domain theory, this may be resolved by weakening the continuity properties required (e.g. to ω_1-continuity [1]), although this admits many undefinable functions and leaves fewer principles with which to reason about program behaviour. A more *intensional* representation of programs (for example, as strategies in a games model) offers the possibility of studying unbounded nondeterminism in computation more directly, although traditional representations of strategies as collections of finite sequences of moves are insufficient to capture the distinction between infinite interactions and finite, but unbounded ones [7].

[1] Research supported by UK EPSRC grant EP/K037633/1

http://dx.doi.org/10.1016/j.entcs.2015.12.017

We take an approach which relates extensional and intensional representations of programs with unbounded nondeterminism: our main result is an equivalence between stable and monotone functions and sequential algorithms on ordered concrete data structures. We show that these equivalent representations may be used to interpret a simple functional programming language with unbounded nondeterminism (fair PCF). We show that this model contains elements which are not definable as terms, leading to a failure of full abstraction and suggesting how it could be further refined.

1.1 Related Work

Our model is based on an intensional description of stable and monotone functions on *biorders* generated from ordered concrete data structures. Biorders, which combine some intensional information, in the form of the stable order, with the extensional (Scott) order, were introduced by Berry [2]. In previous work, the author has shown that stable and continuous functions on biorders with a (extensionally) greatest element are (Milner-Vuillemin) sequential, and used them to construct models of sequential languages such as the lazy λ-calculus [12], as well as λ-calculi with nondeterminism [11]. However, although these models technically carry information about program behaviour, they do not do so in a transparent way.

Concrete data structures were introduced by Kahn and Plotkin [9], as part of a definition of sequentiality for higher-order functionals, but the more intensional notion of *sequential algorithm* (a state of a "function-space" CDS) introduced by Berry and Curien [3] offers an appealing model of computation in its own right. On the one hand, concrete data structures correspond to a *positional* form of games, and sequential algorithms to positional strategies (see e.g. [8]). On the other, sequential algorithms may be related to purely extensional models: in the deterministic case, Cartwright, Curien and Felleisen [4] have established that they compute, and are equivalent to "observably sequential" functions; the author has given a more abstract representation of the latter as *bistable* functions on bistable biorders [12,10].

To interpret sequential, but nondeterministic programs (corresponding to stable and monotone functions on Berry-style biorders, which are sequential, but not *strongly sequential*) as sequential algorithms, we abandon the consistency condition on states (that any cell may be filled with at most one value). However, this also requires an ordering on cells and values (corresponding to game positions), to reflect the fact that (for example) any program which may diverge in response to a given argument may still diverge in response to an argument with a wider range of behaviours. This notion of an ordered concrete data structure was introduced in [13] in which stable and continuous functions were shown to correspond to *finite-branching* sequential algorithms. Interestingly, the stable order on non-deterministic sequential algorithms had been described by Roscoe [15] on processes in CSP — the "strong order" — in work approximately contemporaneous with the discoveries of bidomains and of sequential algorithms. Here, we extend the correspondence between stable functions and sequential algorithms to unbounded nondeterminism. This requires a new notion of ordinal-indexed interaction, to distinguish computations which are

$$\frac{}{(\lambda x.M)\, N \longrightarrow M[N/x]} \qquad \frac{}{? \longrightarrow \mathtt{n}}\ n \in \mathbb{N} \qquad \frac{}{\mathbf{Y}M \longrightarrow M\,(\mathbf{Y}M)}$$

$$\frac{}{\mathtt{If0}\ \mathtt{0} \longrightarrow \lambda x.\lambda y.x} \qquad \frac{}{\mathtt{If0}\ \mathtt{suc(n)} \longrightarrow \lambda x \lambda y.(y\ \mathtt{n})}$$

$$\frac{M \longrightarrow M'}{M\ N \longrightarrow M'\ N} \qquad \frac{M \longrightarrow M'}{\mathtt{suc}\ M \longrightarrow \mathtt{suc}\ M'} \qquad \frac{M \longrightarrow M'}{\mathtt{If0}\ M \longrightarrow \mathtt{If0}\ M'}$$

Table 1
Operational Semantics for Fair PCF

infinite from those which are finite but unbounded.

2 PCF With Unbounded Nondeterminism

In order to illustrate the interaction between higher-order functions, recursion and unbounded nondeterminism, we introduce a programming language which combines them — an extension of PCF with natural number choice — for which we will give a denotational semantics. In other words, we add to the simply-typed λ-calculus over the single ground type $\mathtt{nat}$, the following constants [2]:

- Numerals $\mathtt{0} : \mathtt{nat}$ and $\mathtt{suc} : \mathtt{nat} \to \mathtt{nat}$.
- Conditional $\mathtt{If0} : \mathtt{nat} \to \mathtt{nat} \to (\mathtt{nat} \to \mathtt{nat}) \to \mathtt{nat}$. [3]
- Fixpoints: $\mathbf{Y} : (T \to T) \to T$ for each type T.
- Error and Choice: $\top : \mathtt{nat}$ and $? : \mathtt{nat}$.

2.1 Operational Semantics

The (small-step) reduction relation for closed terms is defined in Table 1. We study a must-testing semantics (unbounded choice is definable from bounded choice up to may-testing equivalence: a sequential algorithms semantics for the latter was presented in [13]). Define $\Downarrow$ (*must-convergence*) to be the least predicate on programs (closed terms of type $\mathtt{nat}$) such that for any program M, if $M' \Downarrow$ for all programs M' such that $M \longrightarrow M'$ then $M \Downarrow$. (So, in particular, every numeral is must-convergent, and if $M_1 \longrightarrow M_2 \longrightarrow \ldots \longrightarrow M_n \longrightarrow \ldots$ then none of the M_i are must-convergent.) Thus we may define must-approximation ($\lesssim$) and must-equivalence ($\simeq$) as the least precongruence and congruence on terms (respectively) such that if $M \lesssim N$ or $M \simeq N$ then $M \Downarrow$ implies $N \Downarrow$. Clearly, $\lesssim$ is a partial order on the $\simeq$-equivalence classes of closed terms at each type.

[2] Our denotational semantics can also interpret the catch operator of *observably sequential PCF*, but we omit this since our focus is not on full abstraction.

[3] If its first argument evaluates to $\mathtt{suc(n)}$, this passes n to its third argument, so we may define $\mathtt{pred} : \mathtt{nat} \to \mathtt{nat} = \lambda x.((\mathtt{If0}\ x)\ \mathtt{0})\ \lambda y.y$.

2.2 Examples

Evidently, we may express bounded choice (up to must-equivalence) using countable choice — e.g. defining $M \,\mathtt{or}\, N = \mathtt{If0}\,?\,\mathtt{then}\,M\,\mathtt{else}\,\lambda x.N$. But attempting to define countable choice using bounded choice — e.g. as $(\mathbf{Y}\lambda f.\lambda x.(x\,\mathtt{or}\,(f\,(\mathtt{suc}\,x))))\,0$ — will fail (evaluation may always take the right hand branch and so diverge).

To express (for example) the *fair merge* of infinite streams of objects of type T, the latter may be represented as objects of type $\mathtt{nat} \to T$ (i.e. $\mathtt{head}(M) = M\,0$, $\mathtt{tail}(M) = \lambda x.M\,(\mathtt{suc}\,x)$ and $M :: N = ((\mathtt{If0}\,y)\,M)\,N$. The fair merge function, $\mathtt{merge} : (\mathtt{nat} \to T) \to (\mathtt{nat} \to T) \to \mathtt{nat} \to T$ returns any interleaving of its arguments which includes all entries from both lists by alternately taking a non-empty initial segment of unbounded length from each stream.

$$(\mathbf{Y}\lambda f\lambda x\lambda u\lambda v.\mathtt{If0}\,x\,\mathtt{then}((f\,\mathtt{suc}(?)\,v\,u))\,\mathtt{else}\,\lambda y.\mathtt{head}(u) :: ((f\,y\,\mathtt{tail}(u)\,v)))\,?$$

Conversely, we may express countable choice in terms of fair merge — e.g. by returning the position of the first zero in a fair merge of the stream of 1s with the stream of 0s:

$$(\mathbf{Y}\lambda f.\lambda x.\lambda y.\mathtt{If0}\,\mathtt{head}(x)\,\mathtt{then}\,y\,\mathtt{else}\,(f\,\mathtt{tail}(x)\,(\mathtt{suc}(y))))\,(\mathtt{merge}\,(\lambda x.0\,)(\lambda x.1))\,0$$

For each $i \in \mathbb{N}$, define $?_i : \mathtt{nat}$ by $?_0 =?$ and $?_{i+1} = \mathtt{suc}?_i$. Then (we claim: proof via the denotational semantics is straightforward) $?_i \lesssim ?_{i+1}$ for each $i \in \mathbb{N}$, and $\top$ is a $\lesssim$-least upper bound for the chain $\langle ?_i \mid i \in \mathbb{N}\rangle$. This may be used to show that many first-order functions definable in fair PCF are not $\lesssim$-continuous. For example, $\lambda x.\mathtt{If0}\,x\,\mathtt{then}\,0\,\mathtt{else}\,\Omega : \mathtt{nat} \to \mathtt{nat}$: $\mathtt{If0}\,?_i\,\mathtt{then}\,0\,\mathtt{else}\,\Omega$ diverges for all i, but $\mathtt{If0}\,\top\,\mathtt{then}\,0\,\mathtt{else}\,\Omega$ converges.

Note that this example also shows that application is not $\lesssim$-continuous with respect to functions as well as arguments — i.e. $\lambda f.(f\,\top)$ is a $\lesssim$-least upper bound for the chain of terms $\langle M_i = \lambda f.(f\,?_i) \mid i \in \omega\rangle$ but $M_i\,\lambda x.\mathtt{If0}\,x\,\mathtt{then}\,0\,\mathtt{else}\,\Omega$ diverges for all i, whereas $(\lambda f.f\,\top)\,\lambda x.\mathtt{If0}\,x\,\mathtt{then}\,0\,\mathtt{else}\,\Omega$ converges. This creates difficulties for defining well-behaved least fixed points for functions, which we will resolve semantically by working with the stable order, for which application *is* continuous with respect to functions (although not arguments).

3 Complete Biorders

We generalize the notion of *biorder* [2,5] to infinite meets (corresponding to infinite branching under a must-testing interpretation).

Definition 3.1 A complete biorder is a complete (meet) lattice $(|D|, \sqsubseteq)$ with a second partial order $\leq_s$ on $|D|$ such that:

- If $x \leq_s y$ then $x \sqsubseteq y$.
- $\bot \leq_s x$ for all $x \in D$ (where $\bot = \bigsqcap D$).
- For any $X, Y \subseteq D$, if $X \leq_s Y$ (meaning: $\forall x \in X \exists y \in Y.x \leq_s y \wedge \forall y \in Y.\exists x \in X.x \leq_s y$) then $\bigsqcap X \leq_s \bigsqcap Y$).

We write $\uparrow X$ if $X \subseteq |D|$ is bounded above in the stable order.

Lemma 3.2 *If $\uparrow X$ then $\bigsqcap X$ is the greatest lower bound for X in the stable order.*

Proof. Supposing $x \leq_s y$ for all $x \in X$:

- For any $x \in X$, $X \leq_s \{x, y\}$ and so $\bigsqcap X \leq_s x = \bigsqcap\{x, y\}$.
- If $z \leq_s x$ for all $x \in X$, then $\{z\} \leq_s X$ and so $\bigsqcap\{z\} \leq_s \bigsqcap X$.

□

Definition 3.3 A function between biorders $f : D \to E$ is said to be *monotone stable* if it preserves both orders, and is *conditionally multiplicative* — i.e. if $\uparrow X$, then $f(\bigsqcap X) = \bigsqcap f(X)$.

Let $\mathcal{CB}$ be the category in which objects are complete biorders and morphisms from D to E are monotone stable functions from D to E.

Proposition 3.4 *$\mathcal{CB}$ is Cartesian closed.*

Proof. Products are defined pointwise. The internal hom $[D, E]$ is the lattice of monotone stable functions (i.e. $(\bigsqcap F)(x) = \bigsqcap_{f \in F} f(x)$), with the stable ordering defined:

$$f \leq_s g \text{ if for all } x, y \in D, x \leq_s y \Longrightarrow f(x) \leq_s g(y) \text{ and } g(y) = f(y) \sqcap g(x).$$

□

4 Ordered Concrete Data Structures

A concrete data structure [9,3] consists of sets of *cells*, *values* and *events* (which are pairs of cells and values), and an *enabling relation* between events and cells. The idea is that each step of a sequential computation is represented as an event (the filling of a cell with a value), which may be dependent on some combination of previous events having occurred (as specified by the enabling relation). This may be considered as a two-player game between the *environment*, which may propose an enabled cell to be filled, and the *program*, which can then fill it with a value. *Deterministic* programs correspond to deterministic strategies for this game which specify a unique value for filling enabled cells. They are represented as *states*: sets of events which satisfy two conditions: *consistency* — every cell must be filled with a unique value — and *safety* — for every filled cell there is a finite chain of enablings of filled cells within the state, back to an "initial cell" which does not depend on any prior events. In order to model unbounded nondeterministic computation with must-testing we adapt this setting in the following ways:

- Removing the consistency condition on states, so that a cell may be filled with multiple different values.
- Placing an ordering on cells and values, (and thus events) and requiring states to be upwards closed under this ordering. This reflects the fact that (for example) the response of a function to an argument which is a nondeterministic choice of x and y must include all of its responses to both x and y.

- Including a distinct element $\perp$ — representing divergence — with which any cell may be filled (cf. the representation of divergence in the game semantics of must-testing in [7]).
- Extending the safety condition to allow infinite chains of enabling events. (This captures the distinction between infinite interaction, and that which is finite but unbounded.)

Definition 4.1 A (filiform) ordered concrete data structure (ocds) A is a tuple $(C(A), V(A), \vdash_A, E(A))$ where $C(A), V(A)$ (the *cells* and *values* of A) are partial orders not containing the distinguished element $\perp$, $E(A) \subseteq C(A) \times V(A)$ is a set of *events* and $\vdash_A \subseteq (E(A) \cup \{*\}) \times C(A)$ is a relation (*enabling*) such that $(c, v) \vdash c'$ implies $c < c'$.
We write $E(A)_\perp$ for the partial order $E(A) \cup (C(A) \times \{\perp\})$, with $(c, \perp) \leq (c', v)$ if $c \leq c'$.

A simple example of an ocds, which we shall use to interpret the type of natural numbers, is $N = (\{c\}, \mathbb{N}, \{(*, c)\}, \{c\} \times \mathbb{N})$, which has a single initial cell which may be filled with any natural number value.

Definition 4.2 A *proof* of an event e is an *ordinal sequence* of events $\langle (c_\alpha, a_\alpha) \mid \alpha \leq \kappa \rangle$ such that $e_\kappa \leq e$ and for $\alpha \leq \kappa$:

- If $\alpha = 0$ then $* \vdash c_\alpha$ (c_0 is initial),
- If $\alpha = \beta + 1$ then $e_\beta \vdash c_\alpha$,
- If $\alpha = \bigvee_{\beta<\alpha} \beta$, then $c_\alpha = \bigvee_{\beta<\alpha} c_\beta$.

We write $x \vdash^* e$ if there is a proof of e, all of the elements of which are in x.

Note that for any proof $\langle (c_\alpha, v_\alpha) |\ \alpha \leq \kappa \rangle$, if $\beta < \alpha \leq \kappa$ then $c_\beta < c_\alpha$.

Definition 4.3 A *state* of an ocds A is a set of events $x \subseteq E(A)_\perp$ satisfying:

Upwards Closure If $e \in x$ and $e \leq e'$ then $e' \in x$.

Safety If $e \in x$ then $x \vdash^* e$.

We write $D(A)$ for the set of states of A. A state x is *total* if $x \subseteq E(A)$ (i.e. no cell is filled with $\perp$ in x).

Thus, the total states in $D(N)$ are in one-one correspondence with the subsets of $\mathbb{N}$, and there is a single divergent state $\perp = E(N)_\perp$. For a state $x \in D(A)$, we define the following sets of cells (subsets of $C(A)$):

- $F(x) = \{c \in C(A) \mid \exists a \in V(A)_\perp . (c, a) \in x\}$ — the set of *filled* cells of x.
- $En(x) = \{c \in C(A) \mid x \vdash^* (c, \perp)\}$ — the set of *enabled* cells of x.
- $A(x) = En(x) - F(x)$ — the set of *accessible* cells of x.

If $c \in En(x)$ and $(c, a) \in E(A)_\perp$, we write $x + (c, a)$ for the state $x \cup \{e \in E(A)_\perp \mid (c, a) \leq e\}$. We will also write $x + (c, V)$ for $\bigcup_{v \in V} (x + (c, v))$.

4.1 Ordered Concrete Data Structures as Complete Biorders

We now need to define extensional and stable orders on $D(A)$, to obtain a complete biorder. Roughly speaking, states x and y should be in the stable order if and only if at any point x can either behave in the same way as y (i.e. fill an enabled cell with the same value) or else diverge. In deterministic sequential data structure models, in which divergence is represented implicitly by non-filling of cells, this is inclusion of states. In our model, which has explicit divergences and is saturated under the order on events, (reverse) inclusion is an extensional order (every event in y dominates an event in x), and we define the stable order as follows:

$$x \leq_s y \text{ if and only if } y \subseteq x \text{ and if } (c,v) \in x \text{ then } (c,v) \in y \text{ or } (c,\bot) \in x.$$

Esssentially, both of these orders were introduced by Roscoe (under the name of the *definedness* and *strong* orders) for process in the failures model of CSP [15] in which, of course, divergence is represented explicitly.

Proposition 4.4 *For any OCDS A, $(D(A), \supseteq, \leq_s)$ is a complete biorder.*

Proof. Since any union of states satisfies the safety and up-closure conditions, $(D(A), \supseteq)$ is a complete (meet) lattice, (with least element $\bot_A = \bigcup D(A)$). By definition, the stable order is contained within the extensional order. $\bot_A \leq_s x$ for all x, since $(c,v) \in \bot_A$ implies $(c,\bot) \in \bot_A$. Suppose $X \leq_s Y$: then for all $(c,a) \in Y$, there exists $y \in Y$ with $(c,a) \in y$ and hence $x \in X$ with $(c,a) \in x$ and so $(c,a) \in x \subseteq \bigcup X$. Conversely, if $(c,a) \in \bigcup X$ then there exists $x \in X$ such that $(c,a) \in x$, and $y \in Y$ with $x \leq_s y$, and so either $x \in y \subseteq \bigcup Y$, or $(c,\bot) \in x \subseteq \bigcup X$. Hence $\bigcup X \leq_s \bigcup Y$ as required. □

Note that if $\uparrow X$ then for all $x, y \in X$, if $(c,v) \in x$ then either $(c,\bot) \in x$ or $(c,v) \in y$. Given $C \subseteq En(x)$, let $x_C = \bigcup_{c \in C}(x + (c,\bot))$. The stable order may be characterized as follows.

Proposition 4.5 *$y \leq_s x$ iff $y = x_C$ for some $C \subseteq En(x)$.*

Proof. Evidently, $x_C \leq_s x$, so it suffices to show that every element $y \leq_s x$ has this form. Let $C = \{c \in En(x) \mid (c,\bot) \in y\}$, so that $x_c \subseteq y$. We claim that $y \subseteq x_C$: suppose $(c,a) \in y$, but $(c,a) \notin x$. Let $\langle c_\alpha, v_\alpha \mid \alpha < \kappa \rangle$ be a proof of (c,a) in y. Then there exists a least value α such that $(c_\alpha, v_\alpha) \notin x$. So $c_\alpha \in E(x)$, and $(c,\bot) \in y$ by stability and so $c_\alpha \in C$. Since $c_\alpha \sqsubseteq c$, $(c,a) \in x_C$ as required. □

5 Sequential Algorithms

By Proposition 4.4, we may define a category $\mathcal{OC}$ in which objects are ordered concrete data structures and morphisms from A to B are monotone stable functions from $D(A)$ to $D(B)$. This has cartesian products, given by the disjoint union of ocds: $(C_1, V_1, E_1, \vdash_1) \times (C_2, V_2, E_2, \vdash_2) = (C_1 \uplus C_2, V_1 \cup V_2, \{(c.1, v) \mid (c,v) \in E_1\} \cup \{(c.2, v) \mid (c,v) \in E_2\}, \{(c.i, v), d.i) \mid ((c,v), d) \in E_i, i \in \{1,2\}\}$.

The fully faithful (identity-on-morphisms) functor $D : \mathcal{OC} \to \mathcal{CB}$ which sends each ocds A to $D(A)$ preserves products. So to establish Cartesian closure of $\mathcal{OC}$ it

suffices to define an exponent ocds $A \Rightarrow B$ for each A, B, such that $D(A \Rightarrow B) \cong [D(A), D(B)]$ in $\mathcal{CB}$. This is a key result, since it establishes that every monotone stable function between the biorder of states of an ocds is computed by a unique state of $A \Rightarrow B$ or *sequential algorithm*.

We define the ordered concrete data structure $A \Rightarrow B$ (cf. the analogous definition of unordered concrete data structure [3]) as follows :

Cells A cell of $A \Rightarrow B$ is given by a pair of a total state of A and a cell of B: $C(A \Rightarrow B) = (D(A) \cap \mathcal{P}(E(A))) \times C(B)$ — with $(x, c) \leq (x', c')$ if $x \subseteq x'$ and $c \leq c'$.

Values A *value* of $A \Rightarrow B$ is either a cell from A or a value from B — the order being determined pointwise from that of $V(B)$ and the dual of $C(A)$: $V(A \Rightarrow B) = C(A)^c \uplus V(B)$ [4].

Events A cell (x, c) of $A \Rightarrow B$ may be *filled* with either a cell accessible from x in A or a value in B which can fill c: $E(A \Rightarrow B) =$
$\{((x, c), c') \mid (x, c) \in C(A \Rightarrow B) \wedge c' \in A(x)\}$
$\cup \{((x, c), v) \mid (x, c) \in C(A \Rightarrow B) \wedge (c, v) \in E(B)\}$.

Enabling The event $((x, c), c')$ enables the cell (x', c) if x' is obtained from x by filling c':
The event $((x, c), v)$ enables the cell (x, c') if (c, v) enables c' in B: $\vdash_{A \Rightarrow B}$
$= \{(((x, c), c'), (x', c)) \in E(A \Rightarrow B) \times C(A \Rightarrow B) \mid \exists V \subseteq V(A).x' = x + (c', V)\}$
$\cup \{(((x, c), v), (x, c')) \in E(A \Rightarrow B) \times C(A \Rightarrow B) \mid (c, v) \vdash_B c'\}$.

A *sequential algorithm* from A to B is a state of $A \Rightarrow B$.

5.1 Stable Functions from Sequential Algorithms

We need to establish that $D(A \Rightarrow B)$ and $[D(A), D(B)]$ are isomorphic in $\mathcal{CB}$. We first show that every sequential algorithm $\sigma \in D(A \Rightarrow B)$ computes a monotone stable function $\mathsf{fun}(\sigma)$ from $D(A)$ to $D(B)$. Given a state x, let $\mathsf{fun}(\sigma)(x) =$

$$\{(c, a) \in E(B)_\perp \mid \exists x' \subseteq x.((x', c), a) \in \sigma \vee \exists c'.(c', \perp) \in x \wedge ((x', c), c') \in \sigma\}$$

We need to show that $\mathsf{fun}(\sigma)$ is a well-defined, monotone stable function.

Lemma 5.1 *For any x, $\mathsf{fun}(\sigma)(x)$ is a state.*

Proof.

Upwards closure Suppose $(c', a') \geq (c, a) \in \mathsf{fun}(\sigma)(x)$. If there exists $x' \subseteq x$ with $((x', c), a) \in \sigma$ then $((x', c'), a') \geq ((x', c), a)$ and so $((x', c'), a') \in \sigma$ and $(c', a') \in \mathsf{fun}(\sigma)(x)$.
If there exists $(c'', \perp) \in x$ such that $((x', c), c'') \in \sigma$ then $((x, c'), c'') \geq ((x', c), c'')$ so $((x', c'), c'') \in \sigma$ and hence $(c', a') \in \mathsf{fun}(\sigma)(x)$.

Safety Suppose $(c, a) \in \mathsf{fun}(\sigma)(x)$. Then then there exists $((y, c), b) \in \sigma$ with $y \subseteq x$, and a proof of (y, c) in σ which therefore restricts to a proof of c in $f(y)$.

[4] We will elide any explicit tagging, assuming that the sets of cells of A and values of B are disjoint.

□

Lemma 5.2 *If* $\uparrow X$ *and* $(c,a) \in \mathsf{fun}(\sigma)(\bigcup X)$ *then either* $(c,a) \in X$ *for all* $x \in X$ *or* $(c,\bot) \in \mathsf{fun}(\sigma)(x')$ *for some* $x' \in X$.

Proof. If $(c,a) \in \mathsf{fun}(\sigma)(\bigcup X)$ then there exists an event $e \in \sigma$ such that either $e = ((w,c),a)$, where $w \subseteq \bigcup X$ or $e = ((w,c),c')$, where $w + (c',\bot) \subseteq \bigcup X$. If $w \subseteq x$ for every $x \in X$ then in the first case $(c,a) \in \bigcup \mathsf{fun}(\sigma)(X)$, and in the second case there exists $x' \in X$ with $w + (c',\bot) \subseteq x'$ and so $(c,\bot) \in \mathsf{fun}(\sigma)(x')$.

So suppose $w \not\subseteq x$ for some $x \in X$. Fixing a proof of e in σ, let $e' = ((w',c'),a')$ be the first element in this proof such that $w' \not\subseteq x$ for some $x \in X$. Then there is an immediately preceding event $((w'',c'),c'')$ such that $w' = w'' + (c'',V)$ for some set of values V, including a value u such that $(c'',u) \notin x$. Because $w' \subseteq \bigcup X$, there exists $x' \in X$ with $(c'',u) \in x'$. Since $x \uparrow x'$, therefore $(c'',\bot) \in x'$, and hence $(c',\bot) \in \mathsf{fun}(\sigma)(y)$. Since $c' \sqsubseteq c$, we have $(c,\bot) \in \mathsf{fun}(\sigma)(x')$ as required. □

Proposition 5.3 *For any sequential algorithm* σ, $\mathsf{fun}(\sigma)$ *is monotone stable.*

Proof. Evidently, if $x \supseteq y$ then $\mathsf{fun}(\sigma)(x) \supseteq \mathsf{fun}(\sigma)(y)$. If $x \leq_s y$, then $f(x) \leq_s f(y)$: suppose $(c,v) \in f(x) = f(x \cup y)$ and so by Lemma 5.2, $(c,v) \in f(y)$ or $(c,\bot) \in f(y)$. If $\uparrow X$, then $\mathsf{fun}(\sigma)(\bigcup X) = \bigcup \mathsf{fun}(\sigma)(X)$: if $(c,a) \in \mathsf{fun}(\sigma)(\bigcup X)$ then by Lemma 5.2, either $(c,a) \in \mathsf{fun}(\sigma)(x)$ for all $x \in X$, and so $(c,a) \in \bigcup \mathsf{fun}(\sigma)(X)$, or $(c,a) \geq (c,\bot) \in \bigcup \mathsf{fun}(\sigma)(X)$. □

We now show that fun itself is a monotone stable function.

Lemma 5.4 *If* $\sigma \leq_s \tau$ *then* $\mathsf{fun}(\sigma) \leq_s \mathsf{fun}(\tau)$.

Proof.

- For all x, $\mathsf{fun}(\sigma)(x) \leq_s \mathsf{fun}(\tau)(x)$. Suppose $(c,v) \in \mathsf{fun}(\sigma)(x)$ but $(c,\bot) \notin \mathsf{fun}(\sigma)(x)$. Then there exists $x' \subseteq x$ with $((x',c),v) \in \sigma$ and $((x',c),\bot) \notin \sigma$ and so $((x',c),v) \in \tau$ and $(c,v) \in \mathsf{fun}(\tau)(x)$ as required.
- For all $x \leq_s y$, $\mathsf{fun}(\sigma)(x) = \mathsf{fun}(\tau)(x) \cup \mathsf{fun}(\sigma)(y)$. Suppose $(c,a) \in \mathsf{fun}(\sigma)(x)$, we need to show that $(c,a) \in \mathsf{fun}(\tau)(x)$ or $(c,a) \in \mathsf{fun}(\sigma)(y)$. By Proposition 5.3, if $(c,a) \notin \mathsf{fun}(\sigma)(y)$, then $(c,\bot) \in \mathsf{fun}(\sigma)(x)$, and so we may assume that $a = \bot$.

  So either there exists an event $((z,c),\bot) \in \sigma$ with $z \subseteq x$ or else there exists $z + (c',\bot) \subseteq x$ such that $((z,c),c') \in \sigma$. But this latter case reduces to the first one, since either $((z,c),c') \in \tau$ (and so $(c,\bot) \in \mathsf{fun}(\tau)(x)$ and we are done), or else $((z,c),\bot) \in \sigma$.

  Assuming $(c,\bot) \notin \mathsf{fun}(\sigma)(y)$, let P be a proof of $((z,c),\bot)$ in σ, and let $((z',c'),a)$ be the least element of P such that $z' \not\subseteq y$. Then there must be an immediately preceding event in P of the form $((z'',c'),c'')$, where $z' = z'' + (c'',V)$ for some V, and hence $z'' + (c'',\bot) \subseteq x$, as $\uparrow \{x,y\}$. If $((z'',c'),c'') \in \tau$ then $(c,\bot) \in \mathsf{fun}(\tau)(x)$. Otherwise $((z'',c'),\bot) \in \sigma$ and so $(c',\bot) \in \mathsf{fun}(\sigma)(y)$, and so $(c,\bot) \in \mathsf{fun}(\sigma)(y)$ as required.

□

Noting that for any set of states $S \subseteq D(A \Rightarrow B)$, $\mathsf{fun}(\bigcup S) = \bigcup_{\sigma \in S} \mathsf{fun}(\sigma)$ by construction — i.e. fun is *additive* — we have shown that:

Proposition 5.5 $\mathsf{fun} : D(A \Rightarrow B) \to [D(A), D(B)]$ *is a monotone stable function.*

5.2 Stable Functions and Sequentiality

Concrete data structures were introduced in order to give a description of sequentiality for higher-order functionals [9]. Essentially, a function between (the sets of states of) concrete data structures A and B is Kahn-Plotkin sequential if any argument (state) x of A, and cell c of B which is filled in $f(y)$ for some y which extends x, can be associated with a cell, accessible from x, which must be filled in any state z (which extends x) such that c is filled in $f(z)$. However, in this original setting, divergence is represented *implicitly*, by not filling an enabled cell (rather than as an explicit divergence by filling a cell with $\perp$), and inclusion of states corresponds to the *stable* order. Thus we translate this original definition of Kahn-Plotkin sequentiality to the current setting by (essentially) replacing the role of "accessible cell" with that of "cell filled with $\perp$", and "filled cell" with "enabled cell not filled with $\perp$". We define a partial order ($\preceq$) on total states (which plays the role of the stable order in the original definition of Kahn-Plotkin sequentiality): $x \preceq y$ if $x \subseteq y$ and if $c \in F(x)$ then $(c, v) \in y$ implies $(c, v) \in x$.

Definition 5.6 A function $f : D(A) \to D(B)$ is *explicitly sequential* if whenever x, y are total states such that $x \preceq y$ then for any event $(c, v) \in f(y)$, either $(c, v) \in f(x)$, or there exists $c' \in A(x) \cap F(y)$ such that if $\uparrow \{x, z\}$ and $(c', \perp) \in z$ then $(c, \perp) \in f(z)$.

We will now show that all monotone stable functions are explicitly sequential.

Lemma 5.7 *If $x \preceq y$ then $y \subseteq x_{A(x) \cap F(y)}$.*

Proof. Suppose $(c, v) \in y$ but $(c, v) \notin x$. Let P be a proof of c in y, and (c', v') the least element of P which is not in x. Then c' is accessible (enabled but not filled) in x, as $x \preceq y$ — i.e. $c' \in A(x) \cap F(y)$, so $(c, v) \geq (c', \perp) \in x_{A(x) \cap F(y)}$. □

Proposition 5.8 *Any monotone stable function from $D(A)$ to $D(B)$ is explicitly sequential.*

Proof. Suppose $x \preceq y$ and $(c, v) \in f(y)$ but $(c, v) \notin f(x)$. Since $y \subseteq x_{A(x) \cap F(y)}$, we have $(c, v) \in f(x_{A(x) \cap F(y)})$ and since $f(x_{A(x) \cap F(y)}) \leq_s f(x)$, we have $(c, \perp) \in f(x_{A(x) \cap F(y)})$.

By conditional multiplicativity, $f(x_{A(x) \cap F(y)}) = \bigcup_{c' \in A(x) \cap F(y)} f(x + (c', \perp))$ and so there is a cell $c' \in A(x) \cap F(y)$ such that $(c, \perp) \in f(x + (c', \perp))$ as required. □

We establish the converse — that every explicitly sequential function is monotone stable — by showing below that every explicitly sequential function from $D(A)$ to $D(B)$ is computed by a state of $A \Rightarrow B$, which corresponds via fun to a monotone stable function from $D(A)$ to $D(B)$.

5.3 Sequential Algorithms from Stable and Monotone Functions

We will use the sequentiality property to establish that $\mathsf{fun} : D(A \Rightarrow B) \to [D(A), D(B)]$ is an isomorphism by defining its inverse, $\mathsf{strat} : [D(A), D(B)] \to D(A \Rightarrow B)$. Given a monotone stable function $f : D(A) \to D(B)$, define $\mathsf{strat}(f) \in D(A \Rightarrow B)$ to be the set of events:

$$\{((x,c),a) \in C(A \Rightarrow B) \times V(B)_\perp \mid (c,a) \in f(x)\}$$
$$\cup\{((x,c),c') \in C(A \Rightarrow B) \times C(A) \mid (c,\perp) \in f(x + (c',\perp))\}$$

Lemma 5.9 $\mathsf{strat}(f)$ *is an upper set.*

Proof. Suppose $((x',c'),a') \geq ((x,c),a) \in \mathsf{strat}(f)$. If $a, a' \in V(B)_\perp$ — i.e. $(c,a) \in f(x)$ — then $(c',a') \geq (c,a) \in f(x') \supseteq f(x)$, and so $((x',c'),a') \in \mathsf{strat}(f)$. If $a, a' \in C(A)$, so that $a' \leq a$ and hence $x + (a,\perp) \subseteq x' + (a',\perp)$, then $(c,\perp) \in f(x' + (a',\perp))$ and so $((x',c'),a') \in \mathsf{strat}(f)$ as required. □

Lemma 5.10 $\mathsf{strat}(f)$ *satisfies the safety property.*

Proof. We construct a proof of each event $((x,c),a)$ in $\mathsf{strat}(f)$ using the explicit sequentiality property for f. Suppose $a \in V(B)_\perp$ and fix a proof $\langle (c_\beta, v_\beta) \mid \beta \leq \lambda\rangle$ of (c,a) in $f(x)$.
For each ordinal α, we define:

- A state $x_\alpha \preceq x$.
- An ordinal $\kappa(\alpha) \leq \alpha$
- An event $e_\alpha \in E(A \Rightarrow B)$ such that $e_\alpha = ((x_\alpha, c_{\kappa(\alpha)}, v_{\kappa(\alpha)})$ or $e_\alpha = ((x_\alpha, c_{\kappa(\alpha)}), c')$ for some $c' \in F(x)$.

such that if $\kappa(\alpha) < \lambda$, then $\langle e_\gamma \mid \gamma \leq \alpha\rangle$ is a proof of e_α.

- Let $x_0 = \{\}$ and $\kappa(0) = 0$,
- For all α, if $(c_{\kappa(\alpha)}, v_{\kappa(\alpha)}) \in f(x_\alpha)$ then let $e_\alpha = ((x_\alpha, c_{\kappa(\alpha)}), v_{\kappa(\alpha)})$ and $x_{\alpha+1} = x_\alpha$ and $\kappa(\alpha+1) = \mathtt{min}\{\lambda, \kappa(\alpha)+1\}$.
  Otherwise, by the explicit sequentiality of f (Proposition 5.8) there is a cell $c' \in A(x_j) \cap F(x)$ such that $(c_{\kappa(\alpha)}, \perp) \in f(x_\alpha + (c',\perp))$ and so we may set $e_\alpha = ((x_\alpha, c_{\kappa(\alpha)}), c')$, and $x_{\alpha+1} = \bigcup\{x_\alpha + (c', v) \mid (c', v) \in x\}$ and $\kappa(\alpha+1) = \kappa(\alpha)$.
- If $\alpha = \bigcup\{\beta < \alpha\}$ then $x_\alpha = \bigcup_{\beta<\alpha} x_\beta$ and $\kappa(\alpha) = \bigcup\{\kappa(\beta) \mid \beta < \alpha\}$.

Since $\kappa(\alpha) \neq \lambda$ implies $e_\alpha \neq e_\beta$ for all $\beta < \alpha$, there must be some (least) α (no greater than the cardinality of $E(A \Rightarrow B)$) such that $\kappa(\alpha) = \lambda$ and so $\langle e_j \mid \beta \leq \alpha\rangle$ is a proof of $((x,c),a)$ in $\mathsf{strat}(f)$. The case in which a is a cell in $C(A)$ is similar.□

Proposition 5.11 *For any monotone stable function* $f : D(A) \to D(B)$, $\mathsf{strat}(f)$ *is a well-defined state of* $A \Rightarrow B$.

We now show that $(\mathsf{fun}, \mathsf{strat})$ are an isomorphism between $D(A \Rightarrow B)$ and $[D(A), D(B)]$. Given a state $x \in D(A)$, let $x^\top$ be the upper closure of $\{(c,v) \in x \mid (c,\perp) \notin x\}$.

Lemma 5.12 $x^\top$ *is a well-defined (total) state.*

Proof. $x^\top$ is by definition an upper set. For safety, suppose $(c, v) \in x$: for any event (c', v') in any proof of c, $c' \leq c$ and so if $(c', v) \notin x^\top$ then $(c, v) \notin x^\top$ — i.e. if $(c, v) \in x$ then any proof of c in x is a proof of c in $x^\top$. □

By definition, if $(c, v) \in x$ then either $(c, \bot) \in x$ or $(c, v) \in x^\top$ — i.e. $x \leq_s x^\top$.

Lemma 5.13 *For any monotone stable function $f : D(A) \to D(B)$ and state $x \in D(A)$: $\mathsf{fun}(\mathsf{strat}(f))(x) = f(x)$.*

Proof. Suppose $(c, a) \in \mathsf{fun}(\mathsf{strat}(f))(x)$. Then either:

- there exists a total $y \subseteq x$ with $((y, c), a) \in \mathsf{strat}(f)$, and so $(c, a) \in f(y) \subseteq f(x)$.
- or there exists $y \subseteq x$ with $((y, c), c') \in \mathsf{strat}(f)$ and $(c', \bot) \in x$, and so $(c, \bot) \in f(y + (c', \bot)) \subseteq f(x)$.

For the converse, suppose $(c, a) \in f(x)$. If $(c, a) \in f(x^\top)$ then $((x^\top, c), a) \in \mathsf{strat}(f)$, and so $(c, a) \in \mathsf{fun}(\mathsf{strat}(f))(x)$ as required.

Otherwise $(c, \bot) \in f(x)$, and by the explicit sequentiality of f (Proposition 5.8), there exists $c' \in A(x^\top) \cap F(y)$ such that $(c', \bot) \in x$ and $(c, \bot) \in f(x^\top + (c', \bot))$ and so $((x^\top, c), c') \in \mathsf{strat}(f)$ and $(c, a) \in \mathsf{fun}(\mathsf{strat}(f))(x)$ as required. □

Lemma 5.14 *For all sequential algorithms $\sigma \in D(A \Rightarrow B)$, $\mathsf{strat}(\mathsf{fun}(\sigma)) = \sigma$.*

Proof. Suppose $((x, c), a) \in E(A \Rightarrow B)_\bot$. Then:

- If $a \in V(B)_\bot$ then $((x, c), a) \in \sigma$ if and only if $(c, a) \in \mathsf{fun}(\sigma)(x)$ if and only if $((x, c), a) \in \mathsf{strat}(\mathsf{fun}(\sigma))$.
- If $a \in C(A)$ then $((x, c), a) \in \sigma$ if and only if $(c, \bot) \in \mathsf{fun}(\sigma)(x + (c', \bot))$ if and only if $((x, c), a) \in \mathsf{strat}(\mathsf{fun}(\sigma))$.

□

Theorem 5.15 $\mathsf{fun} : [D(A), D(B)] \to D(A \Rightarrow B)$ *is an isomorphism, with inverse* $\mathsf{strat} : D(A \Rightarrow B) \to [D(A), D(B)]$.

6 Denotational Semantics for Fair PCF

Types are interpreted as ordered concrete data structures: the type `nat` denotes the OCDS N of natural numbers and $S \to T$ the exponential OCDS $[\![S]\!] \Rightarrow [\![T]\!]$. Terms $x_1 : S_1, \ldots, x_n : S_n \vdash M : T$ denote monotone stable functions from $[\![S_1]\!] \times \ldots \times [\![S_n]\!]$ to $[\![T]\!]$: $\top$ denotes the (constant function returning the) empty state and ? the state $\{(c, i) \mid i \in \mathbb{N}\}$ in which the single cell is filled with every value (but not $\bot$). There are evident functions denoted by `suc` and `If0`, and so it remains to show that the constant $\mathbf{Y}$ may be interpreted as a fixed point. By the Tarski-Knaster theorem, any monotone function $f : D(A) \to D(A)$ has a $\sqsubseteq$-least fixed point, since $D(A)$ is a lattice under the extensional order. However, because function application is not continuous in the extensional order, we do not know how to prove that this fixed point delivers a *computationally adequate* model. Instead, we will show that fixed points may be constructed as *stable* least upper bounds of chains of approximants.

Definition 6.1 Say that an ocds is *stably complete* if for any stably directed set of states X (i.e. for all $x, y \in X$ there exists $z \in X$ with $x, y \leq_s z$), $\bigcap X$ is a stable least upper bound for X.

There are pathological examples of ocds in which this property fails, but we can show that it holds for every object denoting a PCF type (which is evident in the case of `nat`).

Proposition 6.2 *If B is stably complete, then $A \Rightarrow B$ is stably complete.*

Proof. We show that the internal hom $[D(A), D(B)]$ has suprema of stably directed sets, and hence so does $D(A \Rightarrow B)$, and then observe that these are given by the intersection operation. Suppose $F \subseteq [D(A), D(B)]$ is stably directed. For any $x \in D(A)$, $\{f(x) \mid f \in F\}$ is stably directed, so we may define $(\bigvee F)(x) = \bigcap\{f(x) \mid f \in F\}$. This is evidently monotone with respect to the extensional and stable orders. To establish conditional multiplicativity, we need to show that if $\uparrow X$ then $(\bigvee F)(\bigsqcap X) \subseteq \bigsqcap(\bigvee F)(X)$. Suppose $e \notin \bigcup_{x \in X}(\bigvee F)(x)$ and choose any $x \in X$: there exists $f \in F$ with $e \notin f(x)$. Now consider any $y \in X$: there exists $g \in F$ such that $e \notin g(y)$, and $h \in F$ such that $f, g \leq h$ and so $e \notin h(x \cup y) = h(x) \cup h(y)$. Then $f(x) \cup f(y) = f(x \cup y) = f(x) \cup h(x \cup y)$, and so $e \notin f(y)$. So $e \notin \bigcup_{x \in X} f(x) = f(\bigsqcap X)$ and hence $e \notin (\bigvee F)(\bigsqcap X)$ as required.

To show that $\bigvee F$ is a stable upper bound for F, suppose $f \in F$. Then for all x, $f(x) \leq \bigcap\{f(x) \mid f \in F\} = (\bigvee F)(x)$, and if $x \leq y$, $f(x) = \bigcap\{g(x) \cup f(y) \mid g \in F \sqcap g \geq_s f\} = \bigcap\{g(x) \mid g \in F \wedge g \geq_s f\} \cup f(y) = (\bigvee F)(x) \sqcap f(y)$.

To show that $\bigvee F$ is a stably least upper bound, suppose h is a stable upper bound for F. If $x \leq_s y$ then for all $f \in F$, $f(x) = h(x) \sqcap f(y)$ and so $(\bigvee F)(x) = \bigcap\{h(x) \cup f(y) \mid f \in F\} = h(x) \sqcap (\bigvee F)(y)$

Finally, observe that $\mathsf{strat}(\bigvee F) = \bigcap\{\mathsf{strat}(f) \mid f \in F\}$, since $((x,c),v) \in \bigcap\{\mathsf{strat}(f) \mid f \in F\}$ if and only if for all $f \in F$, $(c,v) \in f(x)$, if and only if $((x,c),v) \in \bigvee F$, and similarly $((x,c),c') \in \bigcap\{\mathsf{strat}(f) \mid f \in F\}$ if and only if for all $f \in F$, $(c,\bot) \in f(x + (c',\bot))$, if and only if $((x,c),c') \in \bigvee F$. □

Proposition 6.3 *If A is stably complete and $f : D(A) \to D(A)$ is monotone stable, then f has a $\leq_s$-least fixedpoint.*

Proof. Define the chain of stable approximants $f^\lambda \in D(A)$ for each ordinal λ by:

- $f^\lambda = \bot$, if $\lambda = 0$,
- $f^\lambda = f(f^\kappa)$, if $\lambda = \kappa + 1$
- $f^\lambda = \bigvee_{\kappa<\lambda} f^\kappa$ if $\lambda = \bigcup_{\kappa<\lambda} \kappa$.

By the bounded cardinality of $D(A)$ this has a stationary point, which is a $\leq_s$-least fixed point for f. □

Thus, we may define the denotation of $\mathbf{Y} : (T \to T) \to T$ to be the $\leq_s$-least fixpoint of $h : (T \Rightarrow T) \Rightarrow T \to (T \Rightarrow T) \Rightarrow T$ such that $h(x)(y) = \mathsf{fun}(y)(\mathsf{fun}(x)(y))$.

6.1 Soundness

Straightforward analysis of the reduction rules establishes that:

Lemma 6.4 *For any reducible program M, $[\![M]\!] \subseteq \bigcup\{[\![N]\!] \mid M \longrightarrow N\}$.*

Proposition 6.5 *If $M \Downarrow$ then $[\![M]\!] \neq \bot$.*

Proof. By Lemma 6.4, if $(c, \bot) \notin [\![N]\!]$ for all N such that $M \longrightarrow N$ then $(c, \bot) \notin [\![M]\!]$. So by definition, the set of convergent programs is contained in $\{[\![M]\!] \mid (c, \bot) \notin [\![M]\!]\}$. □

To prove the converse (computational adequacy), we define "approximation relations" in the style of Plotkin [14]: for each type T we define a relation $\lhd_T$ between elements of $[\![T]\!]$ and closed terms of type T:

- $x \lhd_{\mathtt{nat}} M$ if $M \not\Downarrow$ implies $(c, \bot) \in x$ and $M \longrightarrow^* \mathtt{n}$ implies $(c, n) \in x$.
- $f \lhd_{S \to T} M$ if $x \lhd_S N$ implies $f(x) \lhd_T M\, N$.

Note that:

- If $M \longrightarrow N$ for some unique N such that $e \lhd_{\mathtt{nat}} N$ then $e \lhd_{\mathtt{nat}} M$
- If $\langle f_\alpha \mid \alpha < \lambda\rangle$ is a stable chain of functions such that $f_\alpha \lhd_{S\to T} M$ for all $\alpha < \lambda$ then $\bigvee_{\alpha<\lambda} f_\alpha \lhd_{S\to T} M$.

Lemma 6.6 $[\![\mathbf{Y}]\!] \lhd_{(T\to T)\to T} \mathbf{Y}$

Proof. Suppose $T = T_1 \to \ldots \to T_k \to \mathtt{nat}$. We show by ordinal induction that $h^\lambda \lhd_{(T\Rightarrow T)\Rightarrow T} \mathbf{Y}$ for each λ — i.e. if, $g \lhd_{T\Rightarrow T} M$ and $e_i \lhd_{T_i} N_i$ for $1 \leq i \leq k$ then $(h^\lambda g)\, e_1 \ldots e_k \lhd_{\mathtt{nat}} (\mathbf{Y}M)\, N_1 \ldots N_k$.

- For $\lambda = 0$, we have $(h^\lambda g)\, e_1 \ldots e_k = \bot \lhd_{\mathtt{nat}} M\, N_1 \ldots N_k$.
- For $\lambda = \kappa + 1$,: by hypothesis $h^\kappa \trianglelefteq \mathbf{Y}$, and so $(h^\lambda\, g)\, e_1 \ldots e_k = g\,(h^\kappa\, g)\, e_1 \ldots e_n = g\,(h^\kappa g)\, e_1 \ldots e_n \trianglelefteq M\,(\mathbf{Y}M)\, N_1 \ldots N_k \longrightarrow M\,(\mathbf{Y}M)\, N_1 \ldots N_k$ and so $h^\lambda \trianglelefteq \mathbf{Y}$ as required.
- For $\lambda = \bigcup_{\kappa<\lambda} \kappa$, we have $h^\lambda = \bigvee_{\kappa<\lambda} h^\kappa \trianglelefteq [\![M]\!]$ by stable chain closure.

□

We then define $f : [\![\Gamma]\!] \to [\![T]\!] \lhd_{\Gamma,T} \Gamma \vdash M : T$ if $\Gamma = x_1 : S_1, \ldots, x_n : S_n$ and $e_1 \lhd_{S_1} N_1, \ldots, e_n \lhd_{S_n} N_n$ implies $f(e_1, \ldots, e_n) \lhd_T M[N_1/x_1, \ldots, N_n/x_n]$.

Proposition 6.7 (Adequacy) $[\![M]\!] \neq \bot$ *implies* $M \Downarrow$.

Proof. We prove that if $\Gamma \vdash M : T$ then $[\![M]\!] \lhd_{\Gamma,T} M$ by structural induction. □

Hence, by a standard argument, our model is inequationally sound: if $[\![M]\!] \sqsubseteq [\![N]\!]$ then $M \lesssim N$.

7 Conclusions

We conclude by considering the completeness problem for our model. By adding *catch* (a simple, non-local control operator which can distinguish between different

sequentializations of a function) to PCF with *bounded nondeterminism* [13] we can show that every *finite branching* nondeterministic sequential algorithm is the least upper bound of a chain of definable approximants and so prove that this model is *fully abstract*. The situation in fair PCF is more complicated: since continuity fails, we cannot reduce full abstraction to the *finite definability* property. Moreover, our model does not accurately reflect sequential testing of arguments with unbounded nondeterminism, as we can show by giving an example of a stable and monotone function which is not definable in fair PCF. Consider the function $k : D(N) \to D(N)$ such that:

- $k(x) = \top$, if x is finite,
- $k(x) = \bot$ if $x = \bot$,
- $k(x) = 0$, otherwise

This is monotone and stable, since $x \leq_s y$ in $D(N)$ implies $x = y$ or $x = \bot$. However, it cannot be computed in fair PCF, since verifying that x contains infinitely many values requires infinitely many computation steps. (The function $g : D(N) \to D(N)$ such that $g(x) = \top$ if x is finite, and $g(x) = \bot$, otherwise, *is* definable as $\lambda x.(\mathbf{Y}\lambda f.\lambda y.\mathtt{If0}\,(x < y)\,\mathtt{then}\,\top\,\mathtt{else}\,(f\,x)\,\mathtt{suc}(y))\,0.$) We may prove that k is not definable in fair PCF by showing that all definable functions have the following property:

Definition 7.1 A stable function $f : D \to E$ is *weakly co-continuous* if for any downwards $\sqsubseteq$-directed set X, $f(\bigsqcap X) \leq_s \bigsqcap f(X)$.

The function h is not weakly co-continuous: let X be the set of finite states of N — this is downwards $\sqsubseteq$-directed, but $\bigsqcap X$ is infinite and so $h(\bigsqcap X) = 0 \not\leq_s \top = \bigsqcap h(X)$. But we can show that all terms denote weakly co-continuous functions.

Lemma 7.2 *Let F be a (upwards) $\leq_s$-directed set of weakly co-continuous functions from D to E (which is stably complete). Then $\bigvee F$ is weakly co-continuous.*

Proof. $(\bigvee F)(\bigsqcap X) = \bigvee\{f(\bigsqcap X) \mid f \in F\} \leq_s \bigvee\{\bigsqcap f(X) \mid f \in F\}$. □

Proposition 7.3 *Every term of fair PCF denotes a weakly co-continuous function.*

Proof. For each type T, we define a predicate (hereditary weak co-continuity) on the states of $[\![T]\!]$, by induction, as follows:

- Every $x \in D([\![N]\!])$ is hereditarily weakly co-continuous.
- $\sigma \in D([\![S \to T]\!])$ is hereditarily weakly co-continuous if $\mathsf{fun}(\sigma)$ is weakly co-continuous and for any hereditarily weakly co-continuous $x \in D([\![S]\!])$, $\mathsf{fun}(\sigma)(x) \in D([\![T]\!])$ is hereditarily weakly co-continuous.

We now show by structural induction that for any term $x_1 : S_1, \ldots, x_n : S_n \vdash M : T$, $\lambda x_1 \ldots x_n.M : S_1 \to \ldots \to S_n \to T$ is hereditarily weakly co-continuous (using Lemma 7.2 for the fixpoint combinator). □

This is sufficient to show that our semantics is not fully abstract, independently of the absence of the *catch* operators (which are weakly co-continuous).

Proposition 7.4 *The sequential algorithm semantics of fair PCF is not fully abstract.*

Proof. Define $\mathtt{bchoice} : \mathtt{nat} \to \mathtt{nat} = \mathbf{Y}\lambda f.\lambda x.\mathtt{If0}\, x\, \mathtt{then}\, \top\, \mathtt{else}\, \lambda y.(y\, \mathtt{or}\, f\, y)$ — this evaluates its argument and nondeterministically returns a bounded choice over all smaller values. Define $\mathtt{let}\, x =?\, \mathtt{in}\, M$ to be $\mathtt{If0}\, ?\, \mathtt{then}\, M[0/x]\, \mathtt{else}\, \lambda y.M[\mathtt{suc}(y)/x]$ — i.e. nondeterministically choose a value and bind it to x in M. Now define $M = \lambda f.\mathtt{If0}\,(f\,?)\,\mathtt{then}\,(\mathtt{let}\, x\, =?\, \mathtt{in}\, f\, \mathtt{bchoice}(x))\,\mathtt{else}\,(\mathtt{let}\, x\, =?\, \mathtt{in}\, f\, \mathtt{bchoice}(x))$ and $N = \lambda f.f\,?$. These terms (of type $(\mathtt{nat} \to \mathtt{nat}) \to \mathtt{nat}$) are not equivalent in our model — they may be distinguished by application to $\mathsf{strat}(k)$: $[\![M]\!](\mathsf{strat}(k)) = \top$, but $[\![N]\!](\mathsf{strat}(k)) = 0$ (applying k to $[\![?]\!]$ returns 0, but applying k to $[\![\mathtt{bchoice}(\mathtt{n})]\!]$ returns $\top$ and so $[\![M]\!](\mathsf{strat}(k))$ returns $\bigsqcap_{n\in\mathbb{N}} k(\{i < n\}) = \top$.

However, M and N are observationally equivalent: Let $L : \mathtt{nat} \to \mathtt{nat}$ be any closed term of fair PCF. Then by Proposition 7.3 L denotes (the sequential algorithm of) a weakly co-continuous function and so $[\![L\,?]\!] \leq_s [\![\mathtt{let}\, x\, =?\, \mathtt{in}\, L\, \mathtt{bchoice}(x)]\!]$. Thus either $[\![L\,?]\!] = \bot$, in which case $[\![M\,L]\!] = \bot = [\![N\,L]\!]$ or $[\![L\,?]\!] = [\![\mathtt{let}\, x\, =\, ?\, \mathtt{in}\, L\, \mathtt{bchoice}(x)]\!]$, in which case $[\![M\,L]\!] = [\![\mathtt{If0}\, L\,?\, \mathtt{then}\, L\,?\, \mathtt{else}\, L\,?]\!] = [\![L\,?]\!] = [\![N\,L]\!]$. Therefore by adequacy of our semantics, and the Context Lemma for PCF (which extends straightforwardly to fair PCF), M and N are observationally equivalent. □

7.1 Further Directions

We have established a fundamental relationship between extensional and intensional representations of higher-order functional computation with unbounded nondeterminism. Further study of this model may shed light on the debate over the relevance of the concept of *fairness* [6]. Questions posed more directly by our semantics include:

- Can the notion of weak co-continuity be completed to give a characterization of the sequential algorithms which are definable in fair PCF (with *catch*) ? (This will also require a characterization of the effectively computable nondeterministic sequential algorithms.)
- Can we build a graph games model of unbounded non-determinism based on ordered concrete data structures ?

References

[1] Apt, K. R. and G. D. Plotkin, *Countable nondeterminism and random assignment*, Journal of the ACM **33** (1986), pp. 724–767.

[2] Berry, G., *Stable models of typed λ-calculi*, in: *Proceedings of the 5th International Colloquium on Automata, Languages and Programming*, number 62 in LNCS (1978), pp. 72–89.

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

[4] Cartwright, R., P.-L. Curien and M. Felleisen, *Fully abstract semantics for observably sequential languages*, Information and Computation (1994).

[5] Curien, P.-L., G. Winskel and G. Plotkin, *Bistructures, bidomains and linear logic*, in: *Milner Festschrift* (1997).

[6] Djikstra, E. W., "A Discipline of Programming," Prentice-Hall, 1976.

[7] Harmer, R. and G. McCusker, *A fully abstract games semantics for finite non-determinism*, in: *Proceedings of the Fourteenth Annual Symposium on Logic in Computer Science, LICS '99* (1998).

[8] Hyland, M. and A. Schalk, *Games on graphs and sequentially realizable functionals*, in: *Proceedings of LICS '02* (2002).

[9] Kahn, G. and G. Plotkin, *Concrete domains*, Theoretical Computer Science **Böhm Festschrift special issue** (1993), first appeared as technical report 338 of INRIA-LABORIA, 1978.

[10] Laird, J., *Locally Boolean domains*, Theoretical Computer Science **342** (2005), pp. 132–148.

[11] Laird, J., *Bidomains and full abstraction for countable non-determinism*, in: *Proceedings of FoSSaCS'06*, number 3921 in LNCS (2006), pp. 352–366.

[12] Laird, J., *Bistability: A sequential domain theory*, Logical Methods in Computer Science **3** (2007).

[13] Laird, J., *Nondeterministic sequential algorithms*, in: *Proceedings of CSL '09*, number 5771 in LNCS (2009).

[14] Plotkin, G., *Lectures on predomains and partial functions* (1985), notes for a course given at the Center for the study of Language and Information, Stanford.

[15] Roscoe, A. W., *An Alternative order for the failures model*, in *Journal of Logic and Computation*, 2(5):557–557, 1992.