Full Abstraction for PCF*
(Extended Abstract)

Samson Abramsky and Pasquale Malacaria
Imperial College

Radha Jagadeesan
Loyola University

1 Introduction

The Full Abstraction Problem for PCF [23, 20, 7, 11] is one of the longest-standing problems in the semantics of programming languages. There is quite widespread agreement that it is one of the most difficult; there is much less agreement as to what exactly the problem is, or more particularly as to the precise criteria for a solution. The usual formulation is that one wants a "semantic characterization" of the fully abstract model (by which we mean the inequationally fully abstract order-extensional model, which Milner proved to be uniquely specified up to isomorphism by these properties [20]). The problem is to understand what should be meant by a "semantic characterization".

Our view is that the essential content of the problem, what makes it important, is that it calls for a semantic characterization of *sequential, functional computation at higher types*. The phrase "sequential functional computation" deserves careful consideration. On the one hand, sequentiality refers to a computational process extended over time, not a mere function; on the other hand, we want to capture just those sequential computations in which the different parts or "modules" interact with each other in a purely functional fashion.

There have, to our knowledge, been just four models of PCF put forward as embodying some semantic analysis. Three are domain-theoretic: the "standard model" based on Scott-continuous functions [23]; Berry's bidomains model based on stable functions [5]; and the Bucciarelli-Ehrhard model based on strongly stable functions [8]. The fourth is the Berry-Curien model based on sequential algorithms [6] (Cartwright and Felleisen's model turns out to be equivalent to the sequential algorithms model [9, 12]). Of these, we can say that the standard model gives a good account of functional computation at higher types, but fails to capture sequentiality, while the sequential algorithms model gives a good analysis of sequential computation, but fails to capture functional behaviour. In each case, the failure can calibrated in terms of *definability*: the standard model includes parallel functions; the sequential algorithms model includes algorithms which compute "functionals" which are sensitive to non-functional aspects of the behaviour of their arguments. The bidomains model also contains non-sequential functions; while the strongly stable model, in the light of a recent result by Ehrhard [14], can be seen as the "extensional collapse" of the sequential algorithms model. In short, all these models are unsatisfactory because they contain "junk". On the other side of the coin, we have Milner's result that an order-extensional model is fully-abstract iff all its compact elements are definable.

* This research was supported by grants from UK SERC and ESPRIT Basic Research Action 6811 "CLICS II"

Intensional Full Abstraction

This suggests that the key step towards solving the Full Abstraction problem for PCF is to capture PCF definability. This motivates the following definition. A model $\mathcal{M}$ (not necessarily extensional) is *intensionally fully abstract* if it is algebraic, and all its compact elements are definable in PCF. In support of this terminology, we have the fact that the fully abstract model can be obtained from an intensionally fully abstract model $\mathcal{M}$ in the following canonical fashion. Firstly, define a logical relation on $\mathcal{M}$ induced by the ordering on the ground types (which are assumed *standard*, i.e. isomorphic to the usual flat domains of natural numbers and booleans). Because of the definability properties of $\mathcal{M}$, this relation is a preorder at all types. In particular, it is *reflexive* at all types. This says that all elements of the model have extensional (functional) behaviour—there is no junk.

We can now apply Theorem 7.2.2 of [24] to conclude that $\mathcal{M}$ can be collapsed by a continuous homomorphism to the fully abstract model. In short, the fully abstract model is the extensional collapse of any intensionally fully abstract model. Moreover, note that the collapsing map is a homomorphism, and in particular preserves application. This contrasts sharply with "collapses" of the standard model to obtain the fully abstract model, as in the work of Mulmuley [21] and Stoughton and Jung [19], which are only homomorphic on the "inductively reachable" subalgebra.

Thus we propose that a reasonable factorization of the full abstraction problem is to look for a semantic presentation of an intensionally fully abstract model, which embodies a semantic analysis of sequential functional computation. The construction of such a model is our first main result; it is outlined in Sections 2 and 3.

We have explained how the (order-extensional, inequationally) fully abstract model can be obtained from any intensionally fully abstract model by means of a general construction, described in [24]. However, this description of the fully abstract model leaves something to be desired. Firstly, just because the construction in [24] is very general, it is unlikely to yield any useful information about the fully abstract model. Secondly, it is not entirely syntax-free: it refers to the *type structure* of PCF.

What would the ideal form of description of the fully abstract model be? We suggest that it should comprise the specification of a cartesian closed category whose objects are certain cpo's, given together with certain additional "intensional" structure, to be used to characterize sequentiality; and whose morphisms are continuous functions between these cpo's—not *all* continuous functions, of course, but only the sequential ones, as determined by the intensional structure. The interpretation of PCF generated from this category should then be the fully abstract model. Most of the attempts at solving the full abstraction problem of which we are aware, including Berry's bidomains, Curien's bicds, and Bucciarelli and Erhard's strongly stable functions, clearly fall within this general scheme. (Thus for example the intensional structure in bidomains is the stable ordering; for domains with coherence it is the coherence.)

In Section 4, we will explain how the category of games described in Section 2 does indeed give rise to a category of sequential domains in exactly this sense, and hence, in our view, provides a clear-cut solution to the Full Abstraction problem for PCF.

This extended abstract gives only an outline of our constructions and results. A full paper is in preparation.

Martin Hyland and Luke Ong have independently announced the construction of an intensionally fully abstract model for PCF, based on similar but not identical ideas to ours'. A proper comparison must await full publication of the two approaches.

2 The Game Model

Our fully abstract model is based on the game semantics for Multiplicative Linear Logic [2], subsequently extended to cover the exponential types [3].

We begin by fixing some notation. If X is a set, we write $X^\star$ for the set of finite sequences (words, strings) on X and X^ω for the set of infinite sequences. If $f : X \to Y$, then $f^\star : X^\star \to Y^\star$ is the unique monoid homomorphism extending f. We write $|s|$ for the length of a finite sequence. Given a set S of sequences, we write S^{even} for the subset of even length sequences and S^{odd} for the subset of odd length sequences. If $Y \subseteq X$ and $s \in X^\star$, we write $s \upharpoonright Y$ for the result of deleting all occurrences of symbols not in Y from s. If f is a partial function, we write $fx \succeq y$ for "fx is defined and equal to y". We write $s \sqsubseteq t$ if s is a prefix of t, *i.e.* for some u, $su = t$. We always consider sequences under this prefix ordering and use order-theoretic notions [13] without further comment.

2.1 Games

The games we consider are between Player and Opponent. A *play* or *run* of the game consists of an alternating sequence of moves, which may be finite or infinite. Our plays are always with Opponent to move first.

A game is a structure $A = (M_A, \lambda_A, P_A, G_A)$, where

- M_A is the set of moves.
- $\lambda_A : M_A \to \{P, O\} \times \{Q, A\}$ is the labelling function.
  The labelling function indicates if a move is by Player(P) or Opponent(O), and if a move is a question(Q) or an answer(A). The idea is that questions correspond to requests for data, while answers correspond to data (*e.g.* integer or boolean values). In a higher-order context, where arguments may be functions which may themselves be applied to arguments, all four combinations of Player/Opponent with Question/Answer are possible.
  We write
  $$\begin{aligned} \{P,O\} \times \{Q,A\} &= \{PQ, PA, OQ, OA\} \\ \lambda_A &= \langle \lambda_A^{PO}, \lambda_A^{QA} \rangle, \\ M_A^P &= \lambda_A^{-1}(\{P\} \times \{Q,A\}), \\ M_A^O &= \lambda_A^{-1}(\{O\} \times \{Q,A\}), \\ M_A^Q &= \lambda_A^{-1}(\{P,O\} \times \{Q\}), \\ M_A^A &= \lambda_A^{-1}(\{P,O\} \times \{A\}) \end{aligned}$$
  etc., and define
  $$\overline{P} = O, \;\; \overline{O} = P,$$
  $$\overline{\lambda_A^{PO}}(a) = \overline{\lambda_A^{PO}(a)}, \;\; \overline{\lambda_A} = \langle \overline{\lambda_A^{PO}}, \lambda_A^{QA} \rangle.$$

- Let $M_A^{\circledast}$ be the set of all finite sequences s of moves satisfying:
  1. *O moves first*

$$as \in P \Longrightarrow a \in M_A^O$$

  2. *Alternation:*

$$(\forall i : 1 \leq i < |s|)\ [\lambda_A(s_{i+1}) = \overline{\lambda_A(s_i)}]$$

  3. *Bracketing condition:*

$$(\forall t \sqsubseteq s)\ (|t \upharpoonright M_A^A| \leq |t \upharpoonright M_A^Q|)$$

  Then P_A, the set of valid *positions* of the game, is a non-empty prefix closed subset of $M_A^{\circledast}$.
  The bracketing condition can be nicely visualised as follows. Write each question in a play as a left parenthesis "(", and each answer as a right parenthesis ")". Then the string must be well-formed in the usual sense, so that each answer is associated with a unique previous question—the most recently asked, as yet unanswered question. In particular, note that a question by Player must be answered by Opponent, and vice versa.
- An *automorphism* on A is a permutation $\alpha : M_A \cong M_A$ such that
  - $\lambda_A \circ \alpha = \lambda_A$
  - $(\forall s \in M_A^{\circledast})\ [\ s \in P_A \text{ iff } \alpha^\star(s) \in P_A]$

  Thus an automorphism is a permutation on the alphabet which preserves all the structure of the game tree. The set of all automorphisms on A forms a group $Aut(A)$ under composition.
  G_A is a subgroup of $Aut(A)$. G_A induces an equivalence on positions in A by

$$s \approx t \text{ iff } (\exists \alpha \in G_A)\ \alpha^\star(s) = t.$$

For example, the game for *Bool* has one possible opening move $*$ (request for data), with $\lambda_{Bool}(*) = OQ$; and two possible responses, tt or ff, with $\lambda_{Bool}(tt) = \lambda_{Bool}(ff) = PA$. $G_{Bool} = \{id_{M_{Bool}}\}$. The game for Nat is defined similarly.

2.2 Strategies

A *strategy* for Player in A is a non-empty subset $\sigma \subseteq P_A^{even}$ such that $\overline{\sigma} = \sigma \cup dom(\sigma)$ is prefix-closed, where

$$dom(\sigma) = \{sa \in P_A^{odd} \mid \exists b.\ sab \in \sigma\}$$

We will be interested in a restricted class of strategies, the history-free (or history independent, or history insensitive) ones. A strategy σ is *history-free* if it satisfies

- $sab, tac \in \sigma \Longrightarrow b = c$
- $sab, t \in \sigma, ta \in P_A \Longrightarrow tab \in \sigma$ (equivalently, $ta \in dom(\sigma)$).

Note that history-free strategies are closed under directed unions.

Given any history-free strategy σ, we can define $fun(\sigma) : M_A^O \rightharpoonup M_A^P$ by

$$fun(\sigma)(a) \succeq b \text{ iff } (\exists s)\ [sab \in \sigma]$$

Conversely, given $f : M_A^O \rightharpoonup M_A^P$ we can define $traces(f) \subseteq (M_A^{\circledast})^{even}$ inductively by:

$$traces(f) = \{\epsilon\} \cup \{sab \mid s \in traces(f), sa \in P_A, f(a) \succeq b\}$$

We say that f induces a history-free strategy $\sigma_f = traces(f)$, if $traces(f) \subseteq P_A$. Note that if τ is a history-free strategy, we have

$$fun(\sigma_f) \subseteq f, \qquad \sigma_{fun(\tau)} = \tau$$

so there is always a least partial function canonically inducing a history-free strategy.

The equivalence relation on positions extends to a relation on strategies, which, by abuse of notation, we also write $\approx$.
$\sigma \approx \tau$ iff

1. $sab \in \sigma, s'a'b' \in \tau, sa \approx s'a' \Longrightarrow sab \approx s'a'b'$
2. $s \in \sigma, s' \in \tau, sa \approx s'a' \Longrightarrow [sa \in dom(\sigma) \text{ iff } s'a' \in dom(\tau)]$

Interpreting the equivalence on positions as factoring out coding conventions, $\sigma \approx \tau$ expresses the fact that σ and τ are the same modulo coding conventions. $\sigma \approx \sigma$ expresses a "representation independence" property of strategies.

Proposition 1 Properties of $\approx$.

1. *$\approx$ is a partial equivalence relation (i.e. symmetric and transitive) on strategies.*
2. *$\sigma \approx \tau$ iff :*
   - $\sigma \approx \sigma,\ \tau \approx \tau$
   - $(\forall s \in \sigma)\ (\exists t \in \tau)\ [s \approx t]$
   - $(\forall t \in \tau)\ (\exists s \in \sigma)\ [s \approx t]$

2.3 "Intensional" ordering

Firstly, some notation. We write $\hat{A}$ for the set of partial equivalence classes of strategies on a game A, which we think of as the space of "points" on A. We write $[\sigma] = \{\tau \mid \sigma \approx \tau\}$ when $\sigma \approx \sigma$. $\hat{A}$ is pre-ordered by:

$$[\sigma] \sqsubseteq [\tau] \text{ iff } (\forall \sigma' \in [\sigma])\ (\exists \tau' \in [\tau])\ [\sigma' \subseteq \tau']$$

Proposition 2. *1. $\sqsubseteq$ is a preorder*
2. *If $\sigma \subseteq \tau \approx \tau'$ then $(\exists \sigma')\ [\sigma \approx \sigma' \subseteq \tau']$*
3. *If $\sigma' \subseteq \tau \subseteq \sigma''$ and $\sigma' \approx \sigma''$ then $\sigma' \approx \tau \approx \sigma$"*
4. *if $[\sigma_1] \subseteq [\sigma_2] \subseteq \ldots$ is an ω-chain in $\widehat{A}$, then its lub $\bigsqcup_i \sigma_i$, is given by $[\bigcup \sigma_i']$ for any chain $\sigma_1' \subseteq \sigma_2' \subseteq \ldots$ with $\sigma_i' \in [\sigma_i]$*

As an immediate corollary,

Proposition 3. *$(\widehat{A}, \sqsubseteq)$ is an (ω)-cpo, with least element given by $[\{\epsilon\}]$*

2.4 Multiplicatives

Tensor The game $A \otimes B$ is defined as follows. We call the games A and B the *component* games.

- $M_{A\otimes B} = M_A + M_B$, the disjoint union of the two move sets.
- $\lambda_{A\otimes B} = [\lambda_A, \lambda_B]$, the source tupling.
- $P_{A\otimes B}$ is the set of all $s \in M^{\circledast}_{A\otimes B}$ such that:
  1. The restriction to the moves in M_A (resp. M_B) is in P_A (resp. P_B)
  2. If two successive moves are in different components, (*i.e.* one is in A and the other is in B), it is the Opponent who has switched components.
  3. Every answer in s must be in the same component game as the corresponding question.
- $G_{A\otimes B} = \{\alpha + \beta \mid \alpha \in G_A, \beta \in G_B\}$

The tensor unit is given by

$$I = (\varnothing, \varnothing, \{\epsilon\}, \{\{\epsilon \mapsto \epsilon\}\})$$

Linear Implication The game $A \multimap B$ is defined as follows. We call the games A and B the *component* games.

- $M_{A\multimap B} = M_A + M_B$, the disjoint union of the two move sets.
- $\lambda_{A\otimes B} = [\overline{\lambda_A}, \lambda_B]$.
- $P_{A\multimap B}$ is the set of all $s \in M^{\circledast}_{A\multimap B}$ such that:
  1. The restriction to the moves in M_A (resp. M_B) is in P_A (resp. P_B)
  2. If two successive moves are in different components, (*i.e.* one is in A and the other is in B), it is the Player who has switched components.
  3. Every answer in s must be in the same component game as the corresponding question.
- $G_{A\multimap B} = \{\alpha + \beta \mid \alpha \in G_A, \beta \in G_B\}$

We refer to the conditions requiring answers to be given in the same components as the corresponding questions as the *stack discipline*. It ensures that computations must evolve in a properly nested fashion. This abstracts out a key structural feature of functional computation, and plays an important rôle in our results.

2.5 The Category of Games

We build a category $\mathcal{G}$:

$$\begin{aligned} \text{Objects} &: \text{Games} \\ \text{Morphisms} &: [\sigma] : A \to B \text{ is a partial equivalence class } [\sigma] \in \widehat{A \multimap B} \end{aligned}$$

The identity morphism $[id_A] : A \to A$ is defined as follows. We use different subscripts on the A's to distinguish the occurrences.

$$id_A = \{s \in P^{even}_{A_1 \multimap A_2} \mid s{\upharpoonright}A_1 = s{\upharpoonright}A_2\}$$

In process terms, this is a bidirectional one place buffer [1].

Composition The composition of (history-free) strategies can be defined in either of two ways: in terms of the set representation, or via the underlying functions on moves inducing the strategies. We begin with the set representation. Firstly, a preliminary definition. Given a sequence of games $A_1, \ldots, A_n$, we define $\mathcal{L}(A_1, \ldots A_n)$, the *local* strings on $A_1, \ldots, A_n$, to be the set of all $s \in (M_{A_1} + \cdots + M_{A_n})^\star$ such that, for all i with $1 \leqslant i < |s|$, $s_i \in M_{A_j}$ and $s_{i+1} \in M_{A_k}$ implies that j is adjacent to k, *i.e.* $|j - k| \leqslant 1$. Now, given $\sigma : A \rightarrow B,\ \tau : B \rightarrow C$, define

$$\sigma;\tau = \{s{\restriction}A, C \mid\ s \in \mathcal{L}(A, B, C),\ s{\restriction}A, B \in \overline{\sigma},\ s{\restriction}B, C \in \overline{\tau}\}^{even}.$$

This definition clearly exhibits the "Cut = Parallel Composition + Hiding" paradigm proposed by the first author [1] as the correct computational interpretation of Cut in Classical Linear Logic, with respect to the CSP-style trace semantics for parallel composition and hiding [18].

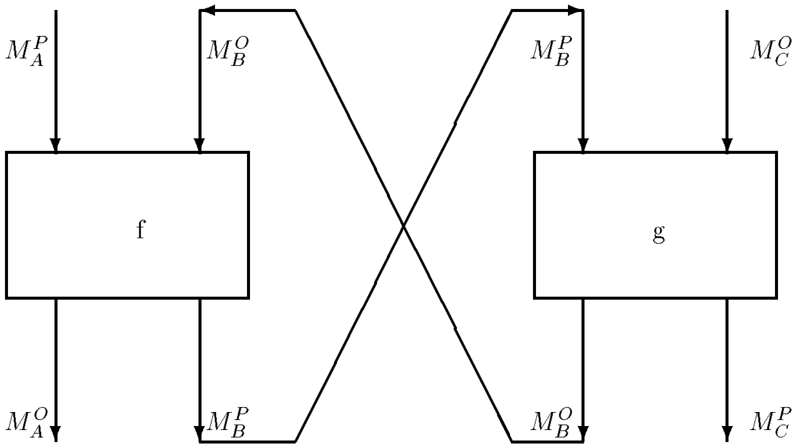
Proposition 4.

1. *Composition is compatible with* $\approx$*:*

$$\sigma, \sigma' : A \rightarrow B,\ \tau, \tau' : B \rightarrow C,\ \sigma \approx \sigma',\ \tau \approx \tau' \Longrightarrow\ \sigma;\tau \approx \sigma';\tau'$$

2. *Composition preserves the bracketing condition.*
3. *Composition is monotone and continuous with respect to* $\sqsubseteq$*.*

We now describe composition in terms of the functions inducing strategies. Say we have $\sigma_f : A \rightarrow B$, $\sigma_g : B \rightarrow C$. We want to find h such that $\sigma_f;\sigma_g = \sigma_h$. We shall compute h by the "execution formula" [15, 16, 17]. Before giving the formal definition, let us explain the idea, which is rather simple. We want to hook the strategies up so that Player's moves in B under σ get turned into Opponent's moves in B for τ, and vice versa. Consider the following picture:



Assume that the Opponent starts in C. There are two possible cases:

– The move is mapped by g to a response in C: In this case, this is the response of the function h.
– The move is mapped by g to a response in B. In this case, this response is interpreted as a move of the Opponent in B and fed as input to f. In turn, if f responds in A, this is the response of the function h. Otherwise, if f responds in B, this is fed back to g. In this way, we get an internal dialogue between the strategies f and g.

It remains to give a formula for computing h according to these ideas. This is the execution formula:

$$h = EX(f, g) = \bigvee_{k \in \omega} m_k$$

The join in the definition of h can be interpreted concretely as union of graphs. It is well-defined because it is being applied to a family of partial functions with pairwise disjoint domains of definition. The functions $m_k : M_A^+ + M_C^- \rightharpoonup M_A^- + M_C^+$ are defined by

$$m_k = \pi^\star \circ ((f + g) \circ \mu)^k \circ (f + g) \circ \pi$$

The idea is that m_k is the function which, when defined, feeds an input from M_A^+ or M_C^- exactly k times around the channels of the internal feedback loop and then exits from M_A^- or M_C^+. The retraction

$$\pi : M_A + M_C \lhd M_A + M_B + M_B + M_C : \pi^\star$$

is defined by

$$\pi^\star = [inl, 0, 0, inr] \qquad \pi = [in_1, in_4]$$

and the "message exchange" function $\mu : M_A^- + M_B^+ + M_B^- + M_C^+ \rightharpoonup M_A^+ + M_B^- + M_B^+ + M_C^-$ is defined by

$$\mu = 0 + [inr, inl] + 0$$

Here, 0 is the everywhere undefined partial function.

$\mathcal{G}$ as an autonomous category We have already defined the object part of the tensor product $A \otimes B$, and linear implication $A \multimap B$. We need to extend the definitions to morphisms. We present a couple of cases below. Further details can be found in [2].

Let $\sigma : A \rightarrow B$, $\tau : A' \rightarrow B'$. Then

$$\sigma \otimes \tau = \{s \in P^{even}_{A \otimes A' \multimap B \otimes B'} \mid s \upharpoonright A, B \in \sigma,\ s \upharpoonright A', B' \in \tau\}$$

Let $\sigma : A \otimes B \rightarrow C$. Then $\Lambda(\sigma) : A \rightarrow (B \multimap C)$ is defined as follows.

$$\Lambda(\sigma) = \{assoc^\star(s) \mid s \in \sigma\}$$

where

$$assoc : (M_A + M_B) + M_C \cong M_A + (M_B + M_C)$$

All the above constructions are compatible with $\approx$; for example

$$\sigma, \sigma' : A \rightarrow B,\ \tau, \tau' : A' \rightarrow B',\ \sigma \approx \sigma',\ \tau \approx \tau' \Longrightarrow \sigma \otimes \tau \approx \sigma' \otimes \tau'.$$

All the above constructions are also monotone and continuous with respect to $\sqsubseteq$.

2.6 Products

The game $A\&B$ is defined as follows.

$$\begin{aligned} M_{A\&B} &= M_A + M_B \\ \lambda_{A\&B} &= [\lambda_A, \lambda_B] \\ P_{A\&B} &= P_A + P_B \\ G_{A\&B} &= \{\alpha + \beta \mid \alpha \in G_A, \beta \in G_B\} \end{aligned}$$

The projections are defined as follows. As before, we use different subscripts on the A, B's to distinguish the occurrences.

$$\begin{aligned} fst : A_1\&B \multimap A_2 &= \{s \in P^{even}_{A_1\&B \multimap A_2} \mid s{\restriction}A_1 = s{\restriction}A_2\} \\ snd : A\&B_1 \multimap B_2 &= \{s \in P^{even}_{A\&B_1 \multimap B_2} \mid s{\restriction}B_1 = s{\restriction}B_2\} \end{aligned}$$

fst, snd are compatible with $\approx$.

Pairing *cannot* be defined on history-free strategies in $\mathcal{G}$; however, it can be defined on arrows in the co-Kleisli category for the comonad !, as we will see.

2.7 Exponentials

The game $!A$ is defined as the "infinite tensor power" of A.

- $M_{!A} = \omega \times M_A = \sum_{i \in \omega} M_A$, the disjoint union of countable copies of the moves of A. So, moves of $!A$ have the form (i, m), where i is a natural number, called the index, and m is a move of A.
- Labelling is by source tupling:

$$\lambda_{!A}(i, a) = (i, \lambda_A(a)).$$

- We write $s{\restriction}i$ to indicate the restriction to moves with index i. $P_{!A}$ is the set of all $s \in M^{\circledast}_{!A}$ such that:
  1. $(\forall i)\ [s{\restriction}i \in P_A]$.
  2. If two successive moves are in different indices, it is the Opponent who has switched components.
  3. If $\cdots(i,a)(i,b)\cdots(j,a)(j,c)\cdots$ occurs in s and $\lambda^{PO}_A(a) = O$, then $b = c$
  4. Every answer in s is in the same index as the corresponding question.
- Let $S(\omega)$ be the set of permutations on ω.

$$G_{!A} = \{\alpha \times \beta \mid \alpha \in S(\omega), \beta \in G_A\}.$$

We now describe the strategies $weak :!A_1 \rightarrow I$ witnessing *weakening*, $der :!A_1 \rightarrow A_2$ witnessing *dereliction*, and $con :!A_1 \rightarrow !A_2 \otimes !A_3$ witnessing *contraction*. As before, we use different subscripts on the A's to distinguish the occurrences, and use π_1, π_2

for the projections acting on the moves of $!A$. In the following, k is some fixed natural number.

$$\begin{aligned} weak &= \{\epsilon\} \\ der &= \{s \in P^{even}_{!A_1 \multimap A_2} \mid \pi_1^\star(s{\restriction}A_1) \subseteq \{k\}, \pi_2^\star(s{\restriction}A_1) = s{\restriction}A_2\} \\ con &= \{s \in P^{even}_{!A_1 \multimap !A_2 \otimes !A_3} \mid \\ &\qquad (\forall i)(\forall u = 0,1)\,[\,\pi_2^\star((s{\restriction}!A_1){\restriction}2i+u) = \pi_2^\star((s{\restriction}!A_{2+u}){\restriction}i)]\} \end{aligned}$$

$weak, der, con$ are compatible with $\approx$.

Let $\sigma :!A \rightarrow B$. Then $\sigma^\dagger :!A \rightarrow !B$ is defined as follows. We need some notation first. Let $p(\ ,\ )$ be a bijection between $\omega \times \omega$ and ω. Let $S_i = \{(i,b) \in M_{!B}\} \cup \{(p(i,j), m) \in M_{!A} \mid j \in \omega\}$.

$$\sigma^\dagger = \{s \in P_{!A \multimap !B} \mid (\forall i)\ [\pi_2^\star(s{\restriction}S_i) \in \sigma]\}.$$

This construction is compatible with $\approx$, and is monotone and continuous with respect to $\sqsubseteq$. The operations $(\ )^\dagger$ and der give ! the structure of a comonad. In particular, ! is a functor, with action on $\sigma : A \rightarrow B$ given by $!\sigma = (der;\sigma)^\dagger$.

We can then define

$$\alpha\ = !(A\&B) \xrightarrow{con} !(A\&B) \otimes !(A\&B) \xrightarrow{!fst \otimes !snd} !A \otimes !B$$

and verify that it is an isomorphism. Moreover, given $\sigma :!C \rightarrow A$, $\tau :!C \rightarrow B$, we can define their pairing by:

$$\langle \sigma, \tau \rangle\ = !C \xrightarrow{con} !C \otimes !C \xrightarrow{\sigma^\dagger \otimes \tau^\dagger} !A \otimes !B \cong !(A\&B) \xrightarrow{der} A\&B.$$

2.8 $\mathcal{G}$ as a model of PCF

Proposition 5. *$\mathcal{G}$ is a cpo-enriched model of IMELL (Intuitionistic Multiplicative-Exponential Logic). The co-Kleisli category $K_!(\mathcal{G})$ is a cpo-enriched cartesian closed category, and, fixing the interpretation of ground types as in the definitions of Bool and Nat at the end of Section 2.1, we get a model for PCF.*

PCF types are thus interpreted by defining $A \Rightarrow B$ as $!A \multimap B$. The game interpreting a PCF type A will have a set of moves M_A which is the disjoint union of a family $\{M_i\}_{i \in I}$, where I is the set of (copies of) occurrences of ground types in A. The subgames A_i obtained by restricting plays in A to M_i are called the *constituents* of A. These subgames will each be either a ground type game or its dual. Thus moves will be of the form $(i, *)$ or (i, d), where d is a datum (a boolean or natural number value). The questions of A are exactly the moves of the form $(i, *)$, the answers those of the form (i, d). Moreover, for any play of A, if we write questions $(i, *)$ as $(_i$, and answers (i, d) as $)_i$, then when we pair up parentheses, the indices match. That is, the answer to a question must be given in the *same* constituent.

3 Intensional Full Abstraction

In order to obtain our intensional full abstraction result, it turns out that we need to consider an extension of PCF. This extension is quite "tame", and does not change the character of the language. It consists of extending PCF with call-by-value at ground types. That is, we introduce a new term-forming operation $\lambda^v x^\gamma.M$ for $\gamma \in \{Bool, Nat\}$, with reduction rules

$$\frac{N \longrightarrow N'}{(\lambda^v x.M)N \longrightarrow (\lambda^v x.M)N'} \qquad (\lambda^v x.M)c \longrightarrow M[c/x] \ \ (c \text{ a ground constant}).$$

Alternatively, since we only require definability for compact elements, it would suffice to add a family of constants

$$case_k^T : Nat \Rightarrow \underbrace{T \Rightarrow \cdots \Rightarrow T}_{k} \Rightarrow T$$

for each type T and $k \in \omega$, with the evident behaviour. These constants can easily be defined using λ^v.

To see how harmless this extension—call it PCFv—is, note that each term in PCFv is observationally equivalent to one in PCF. Specifically, if

$$EQ : Nat \Rightarrow Nat \Rightarrow Bool$$

is the monotone equality function on Nat, which is of course definable in PCF, then

$$\lambda^v x^{Nat}.M \approx^{obs} \lambda x^{Nat}.\, cond(EQxx)M\Omega.$$

The point is that our intensional model is sufficiently fine-grained to distinguish between these observationally equivalent terms. However, note that our results in Section 4 apply directly to PCF.

We can now state our first main result.

Theorem 6. *$K_!(\mathcal{G})$ is intensionally fully abstract for PCFv: that is, for each PCF type T interpreted by a game A, and each compact strategy σ on A, there exists a PCFv term M such that $[\![M]\!] = \sigma$.*

In fact, there is an order-isomorphism between the strategies on a PCF type A and an algebraic cpo of "evaluation trees" of type A. These trees are the ideal completion of a certain set of PCFv terms under the Ω-match ordering. The map from trees to strategies is the continuous extension of the semantic function. This immediately yields the definability result.

An example Let us consider the type $(B \Rightarrow B \Rightarrow B) \Rightarrow B$ (where we write B for $Bool$). This is a key test case, where most of the pathologies of previous models show up. Note first that the opening move by Opponent must (by the polarities of the underlying Linear connectives) be in the rightmost B. Player may have no defined response (everywhere undefined function), *or* imediately return tt or ff(constant function), *or* interrogate its argument, by asking a question in the rightmost B in the argument. Now Opponent may interrogate *it's* first or second argument. At this point in the games model without the stack discipline (or in the sequential algorithms model) we could "jump out" of the argument interrogation, and immediately return tt or ff, depending on whether the function requested its first or second argument (i.e. on the "sequentiality index"). This strategy would correspond to a variant of the Cartwright-Felleisen *catch* construct. However, the stack discipline prevents this; a valid strategy can only reply to the original question when it has received an answer to its subsidiary question.

A further, more subtle problem still exists. Even if we cannot return the result of the function until our interrogation of the argument is complete, it would not be precluded that the result would depend on non-functional information we had gleaned about the argument $f : B \Rightarrow B \Rightarrow B$. For example, we could return tt if f had asked for its first argument first, and ff if it had asked for the second argument first. This kind of non-functional behaviour is the basis of the final counter-example in [10].

It is at this point that the restriction to history-free strategies plays a vital rôle. Because the strategy is history-free, its response to the value returned by the input, i.e. the move of type Answer played by the Opponent, can depend only on that move, and not on the previous behaviour of the Opponent. Thus the stack discipline and history-freeness combine to enforce exactly the constraint of functional interactions between modules that we were after.

4 Extensional Full Abstraction

We will now explain how the category of games described above does indeed give rise directly to a category of sequential domains in the sense explained in the Introduction. The key idea is one which has played a significant rôle in work on Synthetic Domain theory and PER models; namely, an *intrinsic preorder*.

Firstly, some notation. We define the *Sierpinski game* $\mathbb{O}$ to be the game with one initial question by Opponent, and one possible response by Player. Note that $\hat{\mathbb{O}}$ is indeed the usual Sierpinski space, i.e. the two point lattice.

Now for any game A we define the intrinsic preorder $\lesssim_A$ on $\hat{A}$ by:

$$x \lesssim_A y \iff \forall \alpha : A \to \mathbb{O}.\ \alpha \circ x \sqsubseteq \alpha \circ y.$$

The following is immediate.

Proposition 7.

1. *$\lesssim_A$ is a preorder.*
2. *$\sqsubseteq\ \subseteq\ \lesssim_A$*
3. *If $(x_n)_{n \in \omega}$ is a $\sqsubseteq$-chain, and for all n $x_n \lesssim_A x$, then $\bigsqcup_{n \in \omega} x_n \lesssim_A x$.*
4. *If $\sigma : A \to B$ is any strategy, then σ is monotone with respect to the intrinsic preorder:*
$$x \lesssim_A y \implies \sigma \circ x \lesssim_B \sigma \circ y.$$

Let $\simeq_A$ be the equivalence induced by $\lesssim_A$. Then we can define a poset

$$\mathcal{D}(A) = (\hat{A}/{\simeq_A}, {\lesssim_A}/{\simeq_A});$$

and every strategy $\sigma : A \rightarrow B$ induces a monotone map $\mathcal{D}(\sigma) : \mathcal{D}(A) \rightarrow \mathcal{D}(B)$ by $\mathcal{D}(\sigma)([x]_A) = [\sigma \circ x]_B$.

To go further, we need an additional assumption on games. We say that a game A is *ω-bifinite* if $\hat{A}$ is an ω-bifinite cpo ("SFP domain") in the usual sense.

Proposition 8. *If A is ω-bifinite, then $\mathcal{D}(A)$ is an ω-bifinite cpo, with compact elements of the form $[a]_A$ with a compact in $\hat{A}$. If $\sigma : A \rightarrow B$ is any strategy between ω-bifinite games, then $\mathcal{D}(\sigma)$ is a continuous map.*

Thus we can define a category sD of sequential domains, with objects ω-bifinite games, and morphisms $f : A \rightarrow B$ given by continuous functions $f : \mathcal{D}(A) \rightarrow \mathcal{D}(B)$ such that $f = \mathcal{D}(\sigma)$ for some $\sigma : A \rightarrow B$. Thus the strategies act as "sequential realizers" for functions. (Note that our realizers are *typed*, and that *all* "intensional objects" automatically have extensional behaviour. As far as we know, these features are unprecedented among the many extant notions of realizability.)

The next step is to consider the type structure on sD. This will be induced from that on the category of games, so that product and function space objects are defined exactly as on games, and the accompanying morphisms are realized by the corresponding strategies witnessing the cartesian closed structure on the category of games. A key issue which immediately arises is how the intrinsic preorder on products and function spaces compares with the pointwise order.

Proposition 9. *The intrinsic and pointwise preorders coincide for products and function spaces.*

The following is then easily established:

Proposition 10. *sD is a cpo-enriched cartesian closed category. Moreover, it is well-pointed in the order-enriched sense: for all $f, g : A \rightarrow B$,*

$$f \leq g \iff \forall x : 1 \rightarrow A.\, f \circ x \leq g \circ x.$$

As an immediate consequence of this Proposition, we obtain an order-extensional model $\mathcal{M}$ of PCF from sD. By the previous results on the category of games, we know that every compact strategy at a PCF type is PCFv-definable. Since the compact elements in $\mathcal{M}$ are the observational equivalence classes of the compact strategies, we know that every compact element of $\mathcal{M}$ is PCF-definable. It follows by Milner's Definability Theorem that $\mathcal{M}$ is the fully abstract model.

References

1. S. Abramsky. Proofs as processes. Unpublished Lecture, 1991.
2. S. Abramsky and R. Jagadeesan, *Games and Full Completeness for Multiplicative Linear Logic*, 1992. To appear in *Journal of Symbolic Logic*.
3. S. Abramsky and R. Jagadeesan, *Game semantics for Exponentials*, 1993. Announcement on the types mailing list.
4. M. Abadi and G. Plotkin, *A* PER *model of polymorphism and recursive types*, Proc. of 5th Annual Symposium on Logic in Computer Science, 1990.

5. G. Berry, *Modeles completement adequats et stable de lambda-calculs types.* These de Doctorat d'Etat, Universite Paris VII, 1979.
6. G. Berry and P.-L. Curien, *Sequential algorithms on concrete data structures.* Theoretical Computer Science 1982.
7. G. Berry, P.-L. Curien and J.-J. Levy, *Full Abstraction for Sequential Languages: the State of the Art*, in M. Nivat and J. Reynolds, eds., Algebraic Methods in Semantics, Cambridge University Press 1985.
8. A. Bucciarelli and T. Ehrhard, *Extensional embedding of a strongly stable model of PCF*, Proc. ICALP 1991, Springer LNCS 510 1991.
9. R. Cartwright and M. Felleisen, *Observable Sequentiality and Full Abstraction*, Proc. POPL 1992.
10. P.-L. Curien, *Categorical Combinators, Sequential Algorithms and Functional Programming*, Pitman 1986.
11. P.L. Curien, *Sequentiality and Full Abstraction*, in M. Fourman, P. T. Johnstone and A. M. Pitts, eds., Applications of Categories in Computer Science, Cambridge University Press 1992.
12. P.-L. Curien, *Observable sequential algorithms on concrete data structures*, Proc. LICS 1992.
13. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.
14. T. Ehrhard, *Projecting sequential algorithms on the strongly stable functions*, Annals of Pure and Applied Logic, 1993.
15. J.-Y. Girard. Towards a geometry of interaction. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 69–108. American Mathematical Society, 1989.
16. J.-Y. Girard. Geometry of interaction 1: Interpretation of System F. In R. Ferro et al., editor, *Logic Colloquium 88.* North Holland, 1989.
17. J.-Y. Girard. Geometry of interaction 2: Deadlock-free algorithms. In P. Martin-Löf and G. Mints, editors, *International Conference on Computer Logic, COLOG 88*, pages 76–93. Springer-Verlag, 1988. Lecture Notes in Computer Science 417.
18. C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.
19. A. Jung and A. Stoughton, *Studying the Fully Abstract Model of PCF within its Continuous Function Model*, Proc. Typed Lambda Calculus, 1993.
20. R. Milner, *Fully Abstract Models for Typed Lambda Calculi*, Theoretical Computer Science 1977.
21. K. Mulmuley, *Full Abstraction and Semantic Equivalence*, MIT Press 1987.
22. W. Phoa, *Effective domains and intrinsic structure*, Proc. of 5th Annual Symposium on Logic in Computer Science, 1990.
23. G. Plotkin, *LCF considered as a programming language*, Theoretical Computer Science 1977.
24. A. Stoughton, *Fully Abstract Models of Programming Languages*, Pitman 1988.