Games and Full Abstraction for the Lazy λ -calculus

Samson Abramsky Guy McCusker * Department of Computing Imperial College of Science, Technology and Medicine 180 Queen's Gate London SW7 2BZ United Kingdom

Abstract

We define a category of games \mathcal{G} , and its extensional quotient \mathcal{E} . A model of the lazy λ -calculus, a type-free functional language based on evaluation to weak head normal form, is given in \mathcal{G} , yielding an extensional model in \mathcal{E} . This model is shown to be fully abstract with respect to applicative simulation. This is, so far as we know, the first purely semantic construction of a fully abstract model for a reflexively-typed sequential language.

1 Introduction

Full Abstraction is a key concept in programming language semantics [9, 12, 23, 26]. The ingredients are as follows. We are given a language \mathcal{L} , with an 'observational preorder' \preceq on terms in \mathcal{L} such that $P \preceq Q$ means that every observable property of P is also satisfied by Q; and a denotational model $\mathcal{M}\llbracket\cdot\rrbracket$. The model \mathcal{M} is then said to be fully abstract with respect to \preceq if

$$P \preceq Q \iff \mathcal{M}\llbracket P \rrbracket \sqsubseteq \mathcal{M}\llbracket Q \rrbracket$$

for all P, Q in \mathcal{L} . (The right-to-left implication is known as soundness, the converse as completeness.) Thus a fully abstract semantics will reflect all and only the observable properties of programs. Constructing fully abstract models in a semantic fashion—rather than by term model constructions or other syntactic means—yields deep structural information about the computational concepts embodied in the programming language. When certain features are present in the language, most notably higher-order functions and recursion, the problem of achieving such a construction has proved to be very subtle and difficult; the most basic case is the well-known Full Abstraction problem for PCF, which has been studied intensively for some 20 years [9,18,23,25].

In a previous paper, game semantics was used as the basis for a solution to this problem [4]; namely, a description of the fully abstract model given completely independently of the syntax of PCF. This semantics can be seen as offering an analysis—in the light of the results obtained in [1, 4], perhaps even a definitive analysis—of sequential, functional computation at higher types.¹

Given this success, it is important to see how broad the scope of the approach initiated in [1,4] actually is. In this paper, we consider the lazy λ -calculus [2,6]. This calculus, and certain associated notions such as applicative simulation, have been quite widely influential [8, 11, 13, 14, 17, 19, 22]. In [6], a syntactic construction of a fully abstract model for the basic sequential language was given, and the canonical denotational model was shown to be fully abstract for a certain parallel extension of the language. However, the problem of giving a direct, syntax-free, purely semantic construction of a fully abstract model for the sequential language remained open.

In this paper, we present a solution to this problem. We use the theory of game semantics for recursive types as developed in [5] to give a game semantics for the lazy λ -calculus, and we show that this semantics

^{*}This research was partially supported by the UK EPSRC grant "Foundational Structures for Computing Science", and the ESPRIT Basic Research Action CLICS II. The second author is supported by an EPSRC Research Studentship.

¹Similar results, based on a somewhat different version of game semantics, were obtained independently by Hyland and Ong [15], and also by Nickau [20]. A quite different construction of the fully abstract model was subsequently obtained by O'Hearn and Riecke [21].

is fully abstract with respect to applicative simulation. This is, to our knowledge, the first such full abstraction result for a reflexively-typed sequential language. The techniques required to achieve these results significantly extend those of [1,4], while being firmly based on the work done there. It should be the case that the methods developed in the present paper will apply to a richer, typed metalanguage with recursive types, such as that described in [12]; but this remains to be seen.

2 Lazy λ -calculus

We define here the language λl_{C} [6]. The syntax is that of the type-free λ -calculus with a single constant C.

$$M ::= x \mid \lambda x \cdot M \mid MM \mid \mathsf{C}.$$

Following Barendregt [7], we refer to the terms of the language as $\Lambda(C)$, and the closed terms as $\Lambda(C)^0$. The operational semantics is based on a 'big-step' reduction relation $M \Downarrow N$ evaluating terms to weak head normal form; if a term M evaluates to some N it is said to converge and we write $M \Downarrow$. C is a constant which tests a term for convergence.

$$\frac{\overline{\lambda x.M \Downarrow \lambda x.M}}{M \Downarrow \lambda x.P} \quad \overline{C \Downarrow C} \\
\frac{M \Downarrow \lambda x.P \quad P[Q/x] \Downarrow N}{M Q \Downarrow N} \quad \frac{M \Downarrow C \quad N \Downarrow}{M N \Downarrow I}$$

where I denotes the identity combinator $\lambda x.x$. We shall also use Ω to denote the canonical divergent term $(\lambda x.xx)(\lambda x.xx)$.

The contextual preorder \sqsubseteq^C on $\Lambda(\mathsf{C})^0$ is defined by:

$$M \sqsubseteq^{C} N \stackrel{\text{def}}{=} \forall C[-] \in \Lambda(\mathsf{C})^{0}.C[M] \Downarrow \Rightarrow C[N] \Downarrow$$

where $C[-] \in \Lambda(\mathbb{C})^0$ denotes a closed context. In [6], this is shown to be equivalent to the relation \sqsubseteq^B , defined as the largest *applicative simulation*. A binary relation R on $\Lambda(\mathbb{C})^0$ is an applicative simulation if and only if for all M and N such that $(M, N) \in R$, and all $P \in \Lambda(\mathbb{C})^0$,

$$M \Downarrow P \Rightarrow \exists Q.(N \Downarrow Q \land \forall T.(PT, QT) \in R)$$

3 A games model

We now describe a category of games and (equivalence classes of) history-free strategies which is almost identical to that used in [4], and interpret the linear logic connectives \otimes (tensor), \rightarrow (linear implication) and ! (the 'of course' exponential) in \mathcal{G} . The co-Kleisli category for the comonad ! is then a Cartesian closed category. We also define a 'lifting' operation $(-)_{\perp}$ analogous to the usual domain-theoretic lift [27].

3.1 Games

A game has two participants, Player (P) and Opponent (O). A *play* of the game consists of a finite or infinite sequence of moves, alternately by O and P. In the games we consider, O always moves first.

Before defining games, we need some notation for sequences and operations on sequences. We shall use s, t, \ldots to range over sequences and a, b, \ldots to range over the elements of these sequences. We shall write asto mean the sequence whose first element is a and whose tail is s; and st for the concatenation of sequences s and t. |s| denotes the length of s, and s_i is the *i*th element of s. We use \sqsubseteq for the prefix ordering on sequences. If S is a set, $s \upharpoonright S$ is the restriction of s to elements of S, i.e. the sequence s with all elements not in S deleted. Finally, if S is a set of sequences, then S^{even} is the subset of all even length sequences in S.

A game is specified by a structure $A = (M_A, \lambda_A, P_A, \approx_A)$, where

- M_A is a set (the set of moves).
- $\lambda_A : M_A \to \{P, O\} \times \{Q, A\}$ is the labelling function.

The labelling function indicates whether a move is by P or by O, and whether a move is a question (Q) or an answer (A). We write

$$\{P,O\} \times \{Q,A\} = \{PQ, PA, OQ, OA\}$$

$$\lambda_A = \langle \lambda_A^{PO}, \lambda_A^{QA} \rangle$$

and define

$$P = O, \quad O = P,$$

$$\overline{\lambda_A^{PO}}(a) = \overline{\lambda_A^{PO}}(a), \quad \overline{\lambda_A} = \langle \overline{\lambda_A^{PO}}, \lambda_A^{QA} \rangle$$

- Let M_A^{\circledast} be the set of all finite sequences s of moves satisfying:
 - $\mathbf{p1} \quad s = at \Rightarrow \lambda_A^{PO}(a) = O.$ $\mathbf{p2} \quad (\forall i : 1 \le i < |s|) \quad [\lambda_A^{PO}(s_{i+1}) = \overline{\lambda_A^{PO}(s_i)}].$
 - **p3** $(\forall t \sqsubseteq s)[\mathsf{A}(t) \leq \mathsf{Q}(t)]$ where $\mathsf{Q}(t)$ is the number of question moves (i.e. moves a such that $\lambda^{QA}(a) = Q$) in t and $\mathsf{A}(t)$ is the number of answer moves in t.

Then P_A , the set of valid positions of the game, is a non-empty prefix-closed subset of M_A^{\circledast} .

The conditions above can be thought of as global rules applying to all games. (p1) says that O always moves first, while (p2) says that O and P make moves alternately. (p3) is called the bracketing condition: it ensures that when an answer

is given, there is at least one unanswered question in the position. Questions and answers nest like a well-formed string of brackets—we associate answers to questions in the same way that ')'s are associated to '('s. A consequence of this is that a question asked by Opponent must be answered by Player and vice versa. The set P_A can be thought of as defining rules specific to the game A.

- \approx_A is an equivalence relation on P_A satisfying:
 - e1 $s \approx_A s' \Rightarrow \lambda_A^*(s) = \lambda_A^*(s')$. Here λ_A^* denotes the extension of λ_A to act on sequences; notice that this condition implies that if $s \approx_A s'$ then |s| = |s'|.

e2
$$st \approx_A s't' \land |s| = |s'| \Rightarrow s \approx_A s'.$$

e3 $s \approx_A s' \land sa \in P_A \Rightarrow (\exists a')[sa \approx_A s'a'].$

Games are to represent types. For example, a game for **Bool** has one possible opening move *, which is a request for data, and $\lambda_{Bool}(*) = OQ$; there are then two possible responses for Player, tt and ff, with $\lambda_{Bool}(tt) = \lambda_{Bool}(ff) = PA$. The equivalence relation is just the identity relation on the four possible positions of the game, namely ϵ , *, *tt and *ff. A game for Nat can be defined similarly.

The equivalence relation plays a crucial role in the definition of the exponential. We will define !A as 'infinitely many copies' of the game A, and the equivalence relation factors out 'coding tricks' based on the tagging of the different copies.

3.2 Strategies

A strategy for Player in a game A can be thought of as a rule telling Player which move to make in a given position. Since a position in which Player is about to move is always an odd-length sequence of moves, we can define a strategy as a set of even-length positions as follows.

A strategy for Player in a game A is a non-empty set $\sigma \subseteq P_A^{\text{even}}$ such that $\overline{\sigma} \stackrel{\text{def}}{=} \sigma \cup \operatorname{dom}(\sigma)$ is prefixclosed, where

$$\mathbf{dom}(\sigma) \stackrel{\text{def}}{=} \{ sa \in P_A^{\circ \text{dd}} \mid (\exists b) [sab \in \sigma] \}.$$

We are interested only in history-free strategies, i.e. those strategies whose responses depend only on the last move made, rather than on the whole position. A strategy σ is history-free if it satisfies

- $sab, tac \in \sigma \Rightarrow b = c$
- $sab, t \in \sigma, ta \in P_A \Rightarrow tab \in \sigma.$

If σ is history-free, it can also be seen a partial function from O-moves to P-moves—we write $\sigma(a) = b$ if there is some $sab \in \sigma$.

We extend \approx_A to a partial equivalence relation (i.e. a symmetric, transitive relation), which we write as \approx , on strategies for A thus:

 $\sigma\approx\tau$ iff

$$\begin{aligned} - \ sab &\in \sigma, s'a'b' \in \tau, sa \approx_A s'a' \Rightarrow sab \approx_A s'a'b' \\ - \ s &\in \sigma, s' \in \tau, sa \approx_A s'a' \Rightarrow sa \in \mathbf{dom}(\sigma) \text{ iff } s'a' \in \mathbf{dom}(\tau). \end{aligned}$$

From now on we are only interested in those historyfree strategies σ such that $\sigma \approx \sigma$; since the equivalence relation is intended to factor out 'coding tricks' in the definition of the exponential !, this condition says that σ is 'independent of coding'. If σ is a history-free strategy for a game A and $\sigma \approx \sigma$, we shall write $\sigma : A$.

3.3 Multiplicatives

Given games A and B, the game $A \multimap B$ is defined as follows:

- $-M_{A \rightarrow B} = M_A + M_B$ (where + denotes disjoint union).
- $-\lambda_{A\multimap B} = [\overline{\lambda_A}, \lambda_B].$
- $P_{A \multimap B}$ is the set of all $s \in M_{A \multimap B}^{\circledast}$ satisfying
 - 1. Projection condition: $s \upharpoonright M_A \in P_A$ and $s \upharpoonright M_B \in P_B$.
 - 2. Stack discipline: Every answer is in the same component as the corresponding question.

 $- s \approx_{A \multimap B} s'$ iff

$$s | M_A \approx_A s' | M_A, s | M_B \approx_B s' | M_B$$
 and
 $(\forall i : 1 \le i \le |s|) [s_i \in M_A \iff s'_i \in M_A].$

An immediate consequence of the projection condition described above together with the general rules (p1) and (p2) is the *switching condition*: 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, i.e. the second of the two moves is a P-move.

The definition of $A \otimes B$ is the same that of $A \multimap B$, except that the labelling is different: $\lambda_{A \otimes B} = [\lambda_A, \lambda_B]$. A consequence of this is that the switching condition for $A \otimes B$ is the opposite of that for $A \multimap B$; this time only Opponent can switch. The unit for tensor is the empty game:

$$I \stackrel{\text{def}}{=} (\emptyset, \emptyset, \{\epsilon\}, \{(\epsilon, \epsilon)\}).$$

3.4 The category of games

First, some notation: if σ is a history-free strategy for a game A with $\sigma \approx \sigma$ write $[\sigma] = \{\tau \mid \tau \approx \sigma\}$. Let \hat{A} be the set of all such equivalence classes.

Define a category \mathcal{G} :

Objects : Games Morphisms : $[\sigma] : A \to B$ is a partial equivalence class $[\sigma] \in \widehat{A \multimap B}$.

In what follows we will frequently write $\sigma : A \to B$ to mean a strategy representing a morphism from A to B; no confusion will arise because all of the constructions we use are compatible with \approx and so lift to constructions on morphisms.

Identity For any game A, the identity morphism $[id_A]$ is the equivalence class of the 'copycat' strategy, id_A on the game $A_1 \multimap A_2$, defined by

$$\mathsf{id}_A = \{ s \in P_{A_1 \multimap A_2}^{\mathsf{even}} \mid s \upharpoonright A_1 = s \upharpoonright A_2 \}.$$

We use subscripts on the 'A's to distinguish the two occurrences.

Composition We first define the composition of strategies $\sigma : A \to B$ and $\tau : B \to C$. This construction is then lifted to equivalence classes, to give a definition of composition of morphisms.

Given $\sigma : A \to B$ and $\tau : B \to C$, define their composite $\sigma; \tau : A \to C$ by

$$\begin{split} \sigma;\tau &= \{s \upharpoonright A,C \mid s \in (M_A + M_B + M_C)^* \land \\ s \upharpoonright A,B \in \overline{\sigma},s \upharpoonright B,C \in \overline{\tau}\}^{\texttt{even}} \end{split}$$

This can be shown to be well-defined and associative.

Proposition 1 Composition is compatible with \approx : for all $\sigma, \sigma' : A \multimap B$, and $\tau, \tau' : B \multimap C$ we have

$$\sigma \approx \sigma' \wedge \tau \approx \tau' \Rightarrow \sigma; \tau \approx \sigma'; \tau'.$$

In the light of the above Proposition, we can now define composition of morphisms via composition of strategies: $[\sigma]; [\tau] \stackrel{\text{def}}{=} [\sigma; \tau]$ assuming the strategies σ and τ are of suitable types.

 \mathcal{G} as an autonomous category As in [3,4], tensor and linear implication extend to functors. For example, if $\sigma : A \to B$ and $\tau : A' \to B'$ then we define $\sigma \otimes \tau :$ $A \otimes A' \to B \otimes B'$ by

$$\{s\in P_{A\otimes A'\multimap B\otimes B'}^{\operatorname{even}}\mid s\!\upharpoonright\! A,B\in\sigma,s\!\upharpoonright\! A',B'\in\tau\}.$$

If $\sigma : A \otimes B \to C$, there is a strategy $\Lambda(\sigma) : A \to (B \to C)$ defined simply by relabelling moves in σ . (These constructions are compatible with \approx so lift to constructions on morphisms). \mathcal{G} is now an autonomous (symmetric monoidal closed) category. This also means that we can think of a strategy for A indifferently as having the type $I \to A$.

3.5 Exponential

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 countably many copies of M_A . So moves in !A have the form (i, m), where *i* is a natural number and *m* is a move of *A*.
- Labelling is by source tupling:

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

- Writing $s \upharpoonright i$ for the restriction of s to moves with index i, $P_{!A}$ is the set of all $s \in M_{!A}^{\circledast}$ such that:
 - 1. $\forall i[s \mid i \in P_A]$
 - 2. Every answer in s has the same index as the corresponding question.
- Let $S(\omega)$ be the set of permutations on ω , and π_1 and π_2 the first and second projections on the moves of !A. Then $s \approx_{!A} s'$ if and only if for some $\alpha \in S(\omega)$

$$\begin{aligned} \pi_1^*(s) &= \alpha^*(\pi_1^*(s')) \land \\ (\forall i \in \omega) [\pi_2^*(s \restriction \alpha(i)) \approx \pi_2^*(s' \restriction i)] \end{aligned}$$

There are history-free strategies weak : $|A \multimap I$ witnessing weakening, der : $|A \multimap A$ witnessing dereliction and con : $|A \multimap |A \otimes |A$ witnessing contraction. Precise definitions can be found in [4]; briefly, weak is the empty strategy, der copies moves between A and one index of |A, and con uses a bijection between $\omega + \omega$ and ω to copy moves from the two '|A's on the right into |A on the left. There is also an operation taking a strategy $\sigma : |A \rightarrow B$ to $\sigma^{\dagger} : |A \rightarrow |B$. Roughly, σ^{\dagger} works by playing ω -many versions of σ , using a bijection $\langle -, - \rangle : \omega \times \omega \rightarrow \omega$ to decide which index of |A to use: if $\sigma(b) = (j, a)$ then $\sigma^{\dagger}(i, b) = (\langle i, j \rangle, a)$. The operations der and $(-)^{\dagger}$ give ! the structure of a comonad; accordingly, ! is a functor, with action on $\sigma : A \multimap B$ given by $|\sigma \stackrel{\text{def}}{=} (\text{der}; \sigma)^{\dagger}$.

It is also possible to define the product (&) of linear logic; the co-Kleisli category for the comonad ! is then a Cartesian closed category, with $[A \Rightarrow B]$ defined as $!A \multimap B$.

3.6 Lifting

We shall also make use of a lifting construction on games. Given a game $A = (M_A, \lambda_A, P_A, \approx_A)$, define $A_{\perp} = (M_{A_{\perp}}, \lambda_{A_{\perp}}, P_{A_{\perp}}, \approx_{A_{\perp}})$ as follows:

$$\begin{split} M_{A_{\perp}} &= \{\circ, \bullet\} + M_A \\ \lambda_{A_{\perp}} &= [\{\circ \mapsto OQ, \bullet \mapsto PA\}, \lambda_A] \\ P_{A_{\perp}} &= \{\epsilon, \circ\} \cup \{\circ \bullet s \mid s \in P_A\} \\ s \approx_{A_{\perp}} s' \quad \text{iff} \quad s = s' = \epsilon \quad \text{or} \\ s = s' = \circ \quad \text{or} \\ s = \circ \bullet t \text{ and } s' = \circ \bullet t' \text{ and } t \approx_A t'. \end{split}$$

The idea is that there is an initial protocol $\circ \bullet$ determining whether or not a strategy for A_{\perp} is properly in A: if it can answer the initial question \circ (by the only available answer \bullet) then it is. After the first two moves, play continues as a play of A, so there is exactly one more strategy for A_{\perp} than for A, the 'everywhere undefined' strategy $\{\varepsilon\}$, which we write as -.

Lifting can be made into a functor as follows. If $\sigma: A \to B$ then $\sigma_{\perp}: A_{\perp} \to B_{\perp}$ is defined to be

$$\{\varepsilon, \circ_B \circ_A\} \cup \{\circ_B \circ_A \bullet_A \bullet_B s \mid s \in \sigma\}.$$

If we define \mathcal{G}_{\perp} to be the category whose objects are games with a unique first move and whose morphisms $A \to B$ are (equivalence classes of) those strategies which respond to the initial move in B with the initial move in A, then $(-)_{\perp}$ is left adjoint to the forgetful functor $U : \mathcal{G}_{\perp} \to \mathcal{G}$. The unit and co-unit of this adjunction yield, for any game A, maps $up_A : A \to A_{\perp}$ and $dn_A : A_{\perp} \to A$ such that $up_A; dn_A = id_A$. This will be important for us later.

3.7 Recursive types

Games admit a treatment of recursive types very similar to that of information systems [27]. We define an ordering \leq on games as follows. Given two games A and B, $A \leq B$ iff

$$- M_A \subseteq M_B$$
$$- \lambda_A = \lambda_B \upharpoonright M_A$$
$$- P_A = P_B \cap M_A^{\circledast}$$
$$- s \approx_A s' \quad \text{iff} \quad s \approx_B s' \text{ and } s \in P_A.$$

This is a (large) dcpo with least element I and least upper bounds of directed sets given by taking the union of each component, just as for information systems. If a type constructor F is continuous with respect to \trianglelefteq , then we can construct a fixed point D = F(D)

as $\bigsqcup_{\exists} F^n(I)$. In fact we can generalise this to obtain minimal invariants [10] for a large class of functors $F: \mathcal{G}^{\circ p} \times \mathcal{G} \to \mathcal{G}$, including all the type constructors described in this paper, so we obtain canonical solutions of recursive type equations. Details of this are presented in [5].

Another useful fact about \trianglelefteq is that if $A \trianglelefteq B$ then a strategy $\sigma : A$ can be considered as a strategy for B, and a strategy $\tau : B$ can be projected onto A by simply throwing away the moves of B which aren't moves of A.

3.8 A model of λl_{C}

For the remainder of this paper we will be concerned with the game which is the canonical solution of the equation $D = (!D \rightarrow D)_{\perp}$. In the co-Kleisli category, we have maps up : $[D \Rightarrow D] \rightarrow D$ and dn : $D \rightarrow$ $[D \Rightarrow D]$ such that up; dn = id_{[D \Rightarrow D]}. Given this, it is standard that we can obtain a λ -algebra [7] and hence a model of the (untyped) λ -calculus. A term M with n free variables will be interpreted as a morphism [M] : $D^n \rightarrow D$; in \mathcal{G} , this will be a map $!D \otimes \ldots \otimes !D \rightarrow D$, where there are n occurrences of !D. Note that in this model, substitution corresponds to co-Kleisli composition, so that if M has one free variable x, and N is closed, $[M[N/x]] = [N]^{\dagger}; [M]$.

To extend this to a model of λl_{C} we just need to interpret the constant C; this interpretation should clearly be a map which, when applied to - returns -, and when applied to a non-bottom morphism returns the identity. Currying the identity morphism gives a map $\Lambda(\mathsf{id}_D): I \to [D \Rightarrow D]$, which we can consider as having type $[D \Rightarrow D] \to [D \Rightarrow D]$ (by our comments on \trianglelefteq) to give a map α . Then $\alpha_{\perp}: D \to D$ is such that

$$\begin{array}{rcl} -; \alpha_{\perp} &=& -: I \to D \\ \sigma; \alpha_{\perp} &=& \Lambda(\texttt{id}); \texttt{up} : I \to D \text{ if } \sigma \neq - . \end{array}$$

So we can interpret C as $\llbracket C \rrbracket \stackrel{\text{def}}{=} \Lambda(\alpha_{\perp}); up : I \to D$. This gives a model of λl_{C} :

Proposition 2 For any M and $N \in \Lambda(\mathsf{C})^0$,

$$M \Downarrow N \Rightarrow \llbracket M \rrbracket = \llbracket N \rrbracket \neq -$$

In [24], Pitts develops a theory of 'invariant relations' for minimal invariant solutions of recursive equations in the category of dcpos and strict functions, and shows how to use it to prove computational adequacy of a denotational semantics. These techniques can be adapted to \mathcal{G} without difficulty, and yield:

Theorem 3 (Adequacy) If $M \in \Lambda(\mathsf{C})^0$ is such that $\llbracket M \rrbracket \neq -$ then $M \Downarrow$.

3.9 The extensional category

We now describe the extensional quotient \mathcal{E} of the category \mathcal{G} . We make extensive use of the game I_{\perp} , so some observations will be useful at this point. There are only two strategies for I_{\perp} : the empty strategy, which we denote by -, and the strategy which can answer the initial question, which we denote by \top . If $\alpha : A \to I_{\perp}$ immediately answers the initial question with its corresponding answer, rather than switching to A, we also denote α by \top . Given a game A, we define the *intrinsic preorder* \leq_A on the strategies for A (considered as having type $I \to A$) by

$$\sigma \lesssim_A \tau \Leftrightarrow (\forall \alpha : A \to I_{\perp}[\sigma; \alpha = \top \Rightarrow \tau; \alpha = \top]).$$

The morphism $\alpha : A \to I_{\perp}$ can be thought of as a test of σ and τ ; when $\sigma \leq_A \tau$, τ will pass any test that σ passes. If $\sigma; \alpha = \top$ we write $\sigma; \alpha \downarrow$. We denote by \simeq_A the equivalence relation associated with the preorder \leq_A .

Proposition 4 $\sigma \lesssim_{A \multimap B} \tau \Leftrightarrow$ $\forall \alpha : I \to A, \beta : B \to I_{\perp}[\alpha; \sigma; \beta \downarrow \Rightarrow \alpha; \tau; \beta \downarrow].$

As a consequence of this proposition we can define a new category \mathcal{E} whose morphisms from $A \to B$ are equivalence classes of strategies for $A \to B$ under \simeq . Notice that if $\sigma \approx \tau$ then $\sigma \simeq \tau$, so we could just as well have taken equivalence classes of morphisms of \mathcal{G} . Identity and composition are defined from the constructions on strategies, and of course \leq gives rise to a partial order \leq on each hom-set of \mathcal{E} . It is also the case that minimal invariant solutions of recursive type equations in \mathcal{G} transfer to minimal invariants in \mathcal{E} , and we have a computationally adequate model of λl_c in \mathcal{E} just as before. In this category, however, we have the following stronger result.

Theorem 5 (Soundness) Let $M, N \in \Lambda(C)$. Then

$$\llbracket M \rrbracket \leqslant \llbracket N \rrbracket \Rightarrow M \sqsubseteq^B N.$$

The (routine) proof consists of showing that the relation $\llbracket M \rrbracket \leqslant \llbracket N \rrbracket$ between terms $M, N \in \Lambda(\mathsf{C})^0$ is an applicative simulation; then since \sqsubseteq^B is the largest such relation, the result holds.

To illustrate the sequential nature of our model D, consider the "parallel convergence" combinator defined as in [2,6] by the rules

$$\frac{M\Downarrow}{\mathsf{P}MN\Downarrow \mathsf{I}} = \frac{N\Downarrow}{\mathsf{P}MN\Downarrow \mathsf{I}}$$

Thus if M is a canonical term, $\mathsf{P}M\Omega\Downarrow$ and $\mathsf{P}\OmegaM\Downarrow$, but $\mathsf{P}\Omega\Omega\Uparrow$. It is easy to see that no strategy for D can implement P, since any strategy must begin by exploring one of its arguments, and will then diverge if that argument diverges. By contrast, solving the equation $D = (D \Rightarrow D)_{\perp}$ over a category of domains, as in [2,6], will yield a model in which the parallel convergence combinator does live. Indeed, it is proved in [2] that if C is replaced by P in $\lambda l_{\rm C}$, all compact elements of the domain-theoretic model are definable, and it is therefore fully abstract. All this of course parallels the situation for PCF with respect to the parallel or.

4 Definability

4.1 Decomposition

We describe a decomposition of the morphisms $\sigma : I \to D$ which reveals the structure of the terms they represent. The idea is that each σ unfolds into a tree of substrategies; each substrategy has a type of the form $!D_1 \otimes \ldots \otimes !D_n \otimes A_1 \otimes \ldots \otimes A_m \to D$ where each $A_i = !D_{i,1} \otimes \ldots \otimes !D_{i,L_i} \multimap D$ for some L_i . The $!D_i$ components correspond to free variables of the term, while the A_i components correspond to instances of these free variables on which which some computation has been performed 'further up' the decomposition tree. As such, each A_i is associated with one of the $!D_j$. We abbreviate the components $!D_1 \otimes \ldots \otimes !D_n \otimes A_1 \otimes \ldots \otimes A_m$ by T_n , and always assume that an association of A_i to $!D_j$ is specified.

Given $\sigma: T_n \to D$, we decompose by cases according to σ 's response to the initial question in D, and in each case obtain substrategies with types of this form, so that they too can be decomposed. We omit the verification that the substrategies are well-defined, which is simple. There are four possibilities:

- σ has no response. Then $\sigma = \{\varepsilon\}$; no further decomposition is possible, and we write $\sigma = -T_n \rightarrow D$.
- $\sigma(\circ) = \bullet$; so the strategy 'converges' immediately, corresponding to a λ -abstraction. In this case, let $\sigma' = \{s \mid \circ \bullet s \in \sigma\}$. Uncurrying gives this strategy the type $T_n \otimes !D \to D$, which is of the correct form. We write $\sigma = \lambda \sigma'$.
- $\sigma(\circ) = (i, \circ)$ in some $!D_j$. This corresponds to interrogating an argument, i.e. testing it for convergence. We can relabel the *i*th index of $!D_j$ to be a separate D, so the type is $T_n \otimes D \to D$. Then letting $\sigma' = \{\varepsilon\} \cup \{\circ s \mid \circ \circ \bullet s \in \sigma\}$ gives a substrategy $\sigma' : T_n \otimes (!D \multimap D) \to D$. We associate the new 'A' component (namely $!D \multimap D$) with $!D_j$ in this new type, and write $\sigma = (\mathcal{C}_j)\sigma'$.

 $-\sigma(\circ) = \circ$ in some A_i . This corresponds to further testing of a variable which has already been interrogated; it is a test of convergence of a variable applied to some arguments, so the substrategies will represent the arguments and the 'branch' or 'continuation' term. The branch substrategy is easy to extract: let $\sigma_{br} = \{\varepsilon\} \cup \{\circ s \mid \circ \circ \bullet s \in \sigma\}$. If we think of T_n as $T'_n \otimes A_i$, then the type of σ_{br} is $T'_n \otimes A'_i \to D$, where $A'_i = !D_{i,1} \otimes \ldots \otimes !D_{i,L_i} \otimes$ $!D \multimap D.$ The argument strategies require a little more manipulation. First, it can be shown that $\{s \mid o \circ s \in \sigma, \bullet \notin s\}$ (i.e. the possible play after the convergence test has started but before it 'succeeds') is a strategy for $T'_n \to !D_{i,1} \otimes \ldots \otimes !D_{i,L_i}$. Applying derelictions to the A_i components gives a strategy which can be shown to be equivalent to one of the form

$$\operatorname{con}; \sigma_1^{\dagger} \otimes \ldots \otimes \sigma_L^{\dagger} : T_n'' \to !D_{i,1} \otimes \ldots \otimes !D_{i,L_i}$$

where T''_n is the same as T'_n but has each A_j replaced by $!A_j$, and each $\sigma_i : T''_n \to D$. But at most one index of each A_j component is used in any given play, so we can recover the type $T'_n \to D$. We write $\sigma = (\mathcal{C}_i \sigma_1 \dots \sigma_L) \sigma_{\mathtt{br}}$.

On the basis of this decomposition, a strategy can be thought of as representing an 'infinite term' of λl_{C} ; this is the reason for our suggestive notation.

It is worth noting two points about the last case above. First, because it is clearly possible for σ_{br} to 'reuse' A_i (in the form of A'_i), it is not in general possible to combine strategies $\sigma_1, \ldots, \sigma_L, \sigma_{br}$ of suitable types to form $(\mathcal{C}_i \sigma_1 \dots \sigma_L) \sigma_{br}$: history freeness requires that the strategies be in some way compatible. This problem will be addressed by the combinators we introduce shortly. There is also the possibility that the argument strategies make use of some of the A_i components which σ_{br} also uses, again leading to compatibility issues. However, this can be overcome by the (somewhat surprising) observation that if an argument strategy σ_k passes a test $(\beta, \sigma_k, \alpha)$ then it makes no use of the A_i in doing so. The proof of this hinges on the fact that σ_{k}^{\dagger} uses at most one index of A_{i} in any given play; we omit the details. The important point is that any moves σ_k might make in A_i have no effect on its extensional behaviour and can therefore be ignored without consequence.

It should also be pointed out that although the decomposition described here is compatible with \approx , and so well-defined on morphisms in \mathcal{G} , it is not compatible with \simeq : it is perfectly possible for some $\tau \simeq \sigma$ to begin a new convergence test rather than re-using an old one, and so fall under the third case above rather than the fourth. (In this case the decomposition tree for τ would look something like $(\mathcal{C})(\mathcal{C}\sigma_1)\ldots(\mathcal{C}\sigma_1\ldots\sigma_L)\sigma_{br}$ instead). Therefore, although we seek a definability result for the model in \mathcal{E} , we study the model in \mathcal{G} , only passing to \mathcal{E} at the last moment.

4.2 Tests

It is clear that because the A_i are so closely related to the $!D_j$, when applying test strategies to the substrategies in our decomposition we need to constrain their behaviour so that each A_i is treated as an index of its associated $!D_j$. Using repeated applications of dn and uncurrying, we can define strategies

$$\mathsf{dn}_L: D \to (!D_1 \otimes \ldots \otimes !D_L \multimap D).$$

From these we can easily build, for each type T_n , a map

unf :
$$!D_1 \otimes \ldots \otimes !D_n \to A_1 \otimes \ldots \otimes A_m$$

which 'unfolds' each A_i out of the corresponding $!D_j$. Then a suitable test of a strategy $\sigma : T_n \to D$ consists of $\beta : I \to (!D)^n$ and $\alpha : D \to I_{\perp}$; we apply the test using unf, saying that the test succeeds if and only if $(\beta \otimes \beta; \text{unf}); \sigma; \alpha \neq -: I \to I_{\perp}$. In this case we write $(\beta, \sigma, \alpha) \downarrow$. Otherwise, $(\beta, \sigma, \alpha) \uparrow$.

We can now state the definability result we wish to prove:

Theorem 6 (Definability) If $\sigma : T_n \to D$ and $\beta : I \to (!D)^n, \alpha : D \to I_{\perp}$ are test strategies such that $(\beta, \sigma, \alpha) \downarrow$ then there is some $M \in \Lambda(\mathbb{C})$ such that $\llbracket M \rrbracket : (!D)^n \to D$ satisfies:

$$\begin{array}{l} - (\beta, \llbracket M \rrbracket, \alpha) \downarrow. \\ - \forall \beta', \alpha' \quad (\text{of suitable types}) \quad (\beta', \llbracket M \rrbracket, \alpha') \downarrow \quad \Rightarrow \\ (\beta', \sigma, \alpha') \downarrow. \end{array}$$

Notice that in the case when T_n contains no A_j components, the second condition above reduces to $[\![M]\!] \leq \sigma$.

We now describe the various lemmas necessary for the proof of this Theorem. First, we characterise the successful tests of a strategy. As a preliminary, note that any $\beta : I \to (!D)^n$ can be written as $\beta_1^{\dagger} \otimes \ldots \otimes \beta_n^{\dagger}$ where each $\beta_i : I \to D$. Also, if $- \neq \alpha : D \to I_{\perp}$ then $\alpha = dn; (\alpha_1 \multimap \alpha_2)$ for some $\alpha_1 : I \to !D, \alpha_2 : D \to I_{\perp}$. We use these notations in the following lemma.

Lemma 7 (Success Lemma) Suppose $\beta : I \to (!D)^n$ and $\alpha : D \to I_{\perp} \neq -, \top$.

- 1. $(\beta, -, \alpha)$ [†].
- 2. $(\beta, \lambda \sigma, \alpha) \downarrow \Leftrightarrow (\beta \otimes \alpha_1, \sigma, \alpha_2) \downarrow$.

3. $(\beta, (\mathcal{C}_i \sigma_1 \dots \sigma_L) \sigma_{\mathbf{br}}, \alpha) \downarrow \Leftrightarrow (\beta, \sigma_{\mathbf{br}}, \alpha) \downarrow \land \beta; \operatorname{con}; (\sigma_1^{\dagger} \otimes \dots \otimes \sigma_L^{\dagger}); \Lambda^{\perp 1}(\beta_j; \operatorname{dn}_L) \neq -$ where $!D_j$ is associated with A_i , and con denotes the L-fold contraction map $!D \to (!D)^L$.

The proofs of these facts are by considering the play witnessing the success of a test; the last case is the only non-trivial one. Note also that we leave the case of $(\mathcal{C})\sigma$ as a special case of the last one.

This lemma shows that the success of a test, if not immediate (i.e. if $\alpha \neq \top$), depends on the success of tests of the substrategies; furthermore, the plays witnessing the success of these sub-tests are shorter than that of the main test, so a successful test can only search a strategy to finite depth. Armed with this intuition, we seek a method for truncating a strategy which passes a test, so that it still passes that test but its decomposition terminates i.e. it denotes a (finite) term of $\lambda l_{\rm C}$.

4.3 Combinators

Here we attempt to perform the opposite of the decomposition: we build strategies from substrategies. The cases of the undefined strategy and abstraction are simple: we can simply undo the decomposition, and for this reason we use the same notation. However, as remarked previously, the case of a convergence test is not so simple, because the branch strategy may re-use the component in which the convergence test was carried out. The idea here is that any such re-use of A_j which σ might make is forced to occur instead in a fresh index of $!D_i$ (which has no effect on extensional behaviour), thus overcoming the first problem of the decomposition.

- 1. Undefined strategy For each type T_n , define $-T_n \rightarrow D \stackrel{\text{def}}{=} \{\varepsilon\} : T_n \rightarrow D.$
- 2. Abstraction Given $\sigma : T_n \otimes !D \to D$ where no A_j is associated to !D, we can form

$$\lambda \sigma \stackrel{\text{def}}{=} \Lambda(\sigma); \mathsf{up}: T_n \to D$$

3. Convergence test Given $\sigma : T_n \otimes A_j \to D$ with $A_j = (!D_{j,1} \otimes \ldots \otimes !D_{j,L} \otimes !D \to D)$ associated to $!D_i$, and $\sigma_1, \ldots, \sigma_L : (!D)^n \to D$, define first unf $: T_n \to T_n \otimes A_j$ which 'unfolds' A_j out of $!D_i$. Now let $\sigma_{br} = unf; \sigma : T \to D$. Let $\sigma_{test} = con; \sigma_1^{\dagger} \otimes \ldots \otimes \sigma_L^{\dagger} : (!D)^n \to (!D)^L$. Now we can define

$$(\mathsf{C}_j \sigma_1 \dots \sigma_L) \sigma = \{\varepsilon, \circ \circ\} \cup \{\circ \circ s \mid s \in \sigma_{\texttt{test}} \} \\ \cup \{\circ \circ s \bullet t \mid s \in \sigma_{\texttt{test}}, \circ t \in \sigma_{\texttt{br}} \}.$$

Motivated by our earlier comments, we extend this combinator to operate on $\sigma_i : T_n \to D$ by simply taking the projections of the σ_i onto the type $(!D)^n \to D$.

It is clear that all the combinators are monotone with respect to \subseteq (at the level of strategies).

Of course, all of this would be no help if the combinators and decomposition did not correspond closely to each other. In fact they do, as demonstrated by the following result, which follows from the Success Lemma.

Lemma 8 (Correspondence Lemma) Suppose τ : $T_n \to D$ is such that $\tau = (\mathcal{C}_i \sigma_1 \dots \sigma_L) \sigma$, and let β and α be suitable test strategies. Then

$$(\beta, \tau, \alpha) \downarrow \Leftrightarrow (\beta, (\mathsf{C}_i \sigma_1, \dots, \sigma_L) \sigma, \alpha) \downarrow.$$

Similar results are true for the other cases, for trivial reasons.

We have seen that a successful test interrogates the tree of substrategies to some finite depth, until the ' α ' part of the test is reduced to \top ; the test then succeeds and no further information is asked of the strategy. This motivates the following definition.

Definition (Truncations) Given $\sigma : T_n \to D$, define a sequence of strategies $\phi_k(\sigma) : T_n \to D$ as follows:

$$\begin{split} \phi_0(\sigma) &\stackrel{\text{def}}{=} & -\\ \phi_{k+1}(-) &\stackrel{\text{def}}{=} & -\\ \phi_{k+1}(\lambda\sigma) &\stackrel{\text{def}}{=} & \lambda\phi_k(\sigma)\\ \phi_{k+1}((\mathcal{C}_i\sigma_1\ldots\sigma_L)\sigma) &\stackrel{\text{def}}{=} & (\mathcal{C}_i\phi_k(\sigma_1)\ldots\phi_k(\sigma_L))\phi_k(\sigma) \end{split}$$

Note that while $\phi_k(\sigma) \subseteq \phi_{k+1}(\sigma)$ for any σ and k, it is not the case that $\phi_k(\sigma) \subseteq \sigma$. However, we can find an strategy which passes the same test as σ and contains σ , as follows:

Definition (Normalisations) Given $\sigma : T_n \to D$, define a sequence of strategies $\theta_k(\sigma) : T_n \to D$ as follows:

$$\begin{array}{rcl} \theta_0(\sigma) & \stackrel{\text{def}}{=} & \sigma \\ \\ \theta_{k+1}(-) & \stackrel{\text{def}}{=} & - \\ \\ \theta_{k+1}(\lambda\sigma) & \stackrel{\text{def}}{=} & \lambda\theta_k(\sigma) \\ \\ \theta_{k+1}((\mathcal{C}_i\sigma_1\ldots\sigma_L)\sigma) & \stackrel{\text{def}}{=} & (\mathsf{C}_i\theta_k(\sigma_1)\ldots\theta_k(\sigma_L))\theta_k(\sigma) \end{array}$$

It is clear that $\phi_k(\sigma) \subseteq \theta_k(\sigma)$, and the Success Lemma and Correspondence Lemma show that $\theta_k(\sigma)$ passes the same tests as σ does, for each k.

We can now prove a vital result.

Proposition 9 (Truncation Lemma) Let $\sigma : T_n \to D$ and let β and α be suitable test strategies such that $(\beta, \sigma, \alpha) \downarrow$. Then $\exists k \in \omega[(\beta, \phi_k(\sigma), \alpha) \downarrow]$.

The proof of this is by induction on the number of moves taken for the test to succeed. Once more it involves simultaneously proving a similar result for tests of the form $(\beta; \sigma_1^{\dagger} \otimes \ldots \otimes \beta; \sigma_L^{\dagger}); \gamma$ —this part makes essential use of the normalisation of strategies defined above.

Proof of Definability Theorem In the light of the Truncation Lemma, since it is also clear that $(\beta', \phi_k(\sigma), \alpha') \downarrow \Rightarrow (\beta', \sigma, \alpha') \downarrow$, we just need to show that the $\phi_k(\sigma)$ are definable. We can define a term $\Phi_k(\sigma)$ corresponding to $\phi_k(\sigma)$ by induction on k. First we need to decorate the type T_n with some variable names: each $!D_i$ has a unique variable associated with it, and correspondingly the A_i are associated with variables. Then we define $\Phi_k(\sigma)$ by induction on k. $\Phi_0(\sigma) \stackrel{\text{def}}{=} \Omega$ and $\Phi_{k+1}(-) \stackrel{\text{def}}{=} \Omega$. For $\lambda \sigma'$, pick a fresh variable name x not associated with any of the $!D_i$, and associate this to the new !D in the type of σ' . Then $\Phi_{k+1}(\lambda \sigma') \stackrel{\text{def}}{=}$ $\lambda x \Phi_k(\sigma')$. For $(\mathcal{C}_i)\sigma'$, suppose x is the variable associated with $!D_i$. Then $\Phi_{k+1}((\mathcal{C}_i)\sigma') \stackrel{\text{def}}{=} (\mathsf{C}x)\Phi_k\sigma'$. Finally, for $(\mathcal{C}_i \sigma_1 \dots \sigma_L) \sigma_{br}$, suppose A_i is associated with the variable x. Then $\Phi_{k+1}((\mathcal{C}_i\sigma_1\ldots\sigma_L)\sigma_{\mathbf{br}}) \stackrel{\text{def}}{=}$ $(\mathsf{C}x\Phi_k(\sigma_1)\ldots\Phi_k(\sigma_L))\Phi_k(\sigma_{\mathtt{br}})$. It just remains to show that the terms $\Phi_k(\sigma)$ indeed realise the $\phi_k(\sigma)$, i.e. that for any test strategies β and α ,

$$(\beta, \llbracket \Phi_k(\sigma) \rrbracket, \alpha) \downarrow \Leftrightarrow (\beta, \phi_k(\sigma), \alpha) \downarrow.$$

This is quite routine, using the definition of [-], the Success Lemma and the Correspondence Lemma.

5 Full abstraction

With the Definability Theorem in place, we can prove the following completeness result.

Theorem 10 (Completeness) For $M, N \in \Lambda(\mathsf{C})^0$, $M \sqsubseteq^B N \Rightarrow \llbracket M \rrbracket \leqslant \llbracket N \rrbracket$.

Proof In fact we prove the contrapositive of the above. Suppose for some $M, N \in \Lambda(\mathbb{C})^0$, $\llbracket M \rrbracket \notin \llbracket N \rrbracket$. Let $m = \llbracket M \rrbracket$ and $n = \llbracket N \rrbracket$, so that for some $\alpha : D \to I_{\perp}, m; \alpha \neq -$ and $n; \alpha = -$. Then we also have $m^{\dagger}; \operatorname{der}; \alpha \neq -$ and $n^{\dagger}; \operatorname{der}; \alpha = -$. Consider the strategy $\beta : !D \to D$ defined by treating $\operatorname{der}; \alpha : !D \to I_{\perp}$ as having this type. By the Definability Theorem, there is a term $P \in \Lambda(\mathbb{C})$ with one free variable such that m^{\dagger} ; $\llbracket P \rrbracket \neq -$ and n^{\dagger} ; $\llbracket P \rrbracket = -$. Suppose the one free variable of P is x. Then by the definition of $\llbracket - \rrbracket$,

$$\begin{bmatrix} P[M/x] \end{bmatrix} = m^{\dagger}; \begin{bmatrix} P \end{bmatrix} \neq - \\ \begin{bmatrix} P[N/x] \end{bmatrix} = n^{\dagger}; \begin{bmatrix} P \end{bmatrix} = -$$

So by computational adequacy, $P[M/x] \Downarrow$ and $P[N/x] \Uparrow$, so $M \not\sqsubseteq^B N$.

Finally, putting the Soundness Theorem and Completeness Theorem together gives:

Theorem 11 (Full abstraction) For all M and $N \in \Lambda(\mathsf{C})^0$,

$$M \sqsubseteq^B N \Leftrightarrow \llbracket M \rrbracket \leqslant \llbracket N \rrbracket.$$

References

- [1] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF, 1995. To appear.
- [2] S. Abramsky. The lazy λ -calculus. In D. A. Turner, editor, *Research Topics in Functional Programming*, chapter 4, pages 65–117. Addison Wesley, 1990.
- [3] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. Journal of Symbolic Logic, 59(2):543 - 574, June 1994. Also appeared as Technical Report 92/24 of the Department of Computing, Imperial College of Science, Technology and Medicine.
- [4] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF (extended abstract). In M. Hagiya and J. C. Mitchell, editors, *Theoretical* Aspects of Computer Software. International Symposium TACS'94, number 789 in Lecture Notes in Computer Science, pages 1-15, Sendai, Japan, April 1994. Springer-Verlag.
- [5] S. Abramsky and G. McCusker. Games for recursive types. In C. L. Hankin, I. C. Mackie, and R. Nagarajan, editors, *Theory and Formal Methods of Computing 1994: Proceedings of the Second Imperial College Department of Computing Workshop on Theory and Formal Methods.* Imperial College Press, October 1995. Also available by anonymous ftp from theory.doc.ic.ac.ukin directory papers/McCusker.
- [6] S. Abramsky and C.-H. L. Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 105(2):159-267, August 1993.
- [7] H. P. Barendregt. The Lambda Calculus: Its Syntax and Semantics. North-Holland, revised edition, 1984.
- [8] G. Boudol. A lambda calculus for (strict) parallel functions. Information and Computation, 108:51-127, 1994.
- [9] P.-L. Curien. Categorical Combinators, Sequential Algorithms and Functional Programming. Progress in Theoretical Computer Science. Birkhauser, 1993.

- [10] P. J. Freyd. Algebraically complete categories. In A. Carboni et al., editors, Proc. 1990 Como Category Theory Conference, pages 95-104, Berlin, 1991. Springer-Verlag. Lecture Notes in Mathematics Vol. 1488.
- [11] A. D. Gordon. Functional programming and Input/Output. Distinguished Dissertations in Computer Science. Cambridge University Press, 1994.
- [12] C. A. Gunter. Semantics of Programming Languages: Structures and Techniques. Foundations of Computing. MIT Press, 1992.
- [13] M. Hennessy. A fully abstract denotational model for higher-order processes. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science* [16], pages 397-408.
- [14] D. J. Howe. Equality in lazy computation systems. In Proceedings, Fourth Annual Symposium on Logic in Computer Science, pages 198-203. IEEE Computer Society Press, 1989.
- [15] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. 130 pages, ftp-able at theory.doc.ic.ac.uk in directory papers/Ong, 1994.
- [16] IEEE Computer Society Press. Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science, 1993.
- [17] A. Jeffrey. A fully abstract semantics for concurrent graph reduction. In *Proceedings, Ninth Annual IEEE* Symposium on Logic in Computer Science, pages 82-91. IEEE Computer Society Press, 1994.
- [18] R. Milner. Fully abstract models of typed lambdacalculi. Theoretical Computer Science, 4:1-22, 1977.
- [19] R. Milner. Functions as processes. In Proceedings of ICALP 90, volume 443 of Lecture Notes in Computer Science, pages 167-180. Springer-Verlag, 1990.
- [20] H. Nickau. Hereditarily sequential functionals. In Proceedings of the Symposium on Logical Foundations of Computer Science: Logic at St. Petersburg, Lecture notes in Computer Science. Springer, 1994.
- [21] P. W. O'Hearn and J. G. Riecke. Kripke logical relations and PCF. Information and Computation, 120(1):107-116, 1995.
- [22] C. H. L. Ong. Non-determinism in a functional setting. In Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science [16], pages 275-286.
- [23] C.-H. L. Ong. Correspondence between operational and denotational semantics. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic* in Computer Science, Vol 4, pages 269-356. Oxford University Press, 1995.
- [24] A. M. Pitts. Relational properties of domains. Technical Report 321, Cambridge Univ. Computer Laboratory, December 1993. 37 pages.

- [25] G. Plotkin. LCF considered as a programming language. Theoretical Computer Science, 5:223-255, 1977.
- [26] A. Stoughton. Fully abstract models of programming languages. Pitman, 1988.
- [27] G. Winskel. The Formal Semantics of Programming Languages. Foundations of Computing. The MIT Press, Cambridge, Massachusetts, 1993.