

Game Semantics for Higher-Order Concurrency

J. Laird*

Department of Informatics, University of Sussex, UK
jiml@sussex.ac.uk

No Institute Given

Abstract. We describe a denotational (game) semantics for a call-by-value functional language with multiple threads of control, which may communicate values of general type on locally declared channels.

This develops previous work which interpreted freshly generated names in a category of games acted upon by the group of natural number automorphisms, by showing how names may be associated with “dependent arenas” in which interaction between strategies, corresponding to asynchronous communication on named channels, may occur.

We describe a model of the call-by-value λ -calculus (a closed Freyd category) based on these arenas, and use this as the basis for interpreting our language. We prove that the semantics is fully abstract with respect to may-testing using a correspondence between channel and function types based on the “triggering” representation of procedure-passing in terms of name-passing.

1 Introduction

Higher-order concurrency — the capacity to generate multiple threads of control and pass higher-order functions and processes as values between them — is a powerful and subtle programming paradigm. Languages and calculi with these features, such as *Concurrent ML* and the higher-order π -calculus, have been extensively studied using operational methods, but the combination of dynamic name creation and higher-order value-passing has presented a long standing challenge for denotational semantics. In this paper, we shall develop a denotational model of a call-by-value functional language with higher-order concurrency, including dynamically generated channel names, and prove that it is fully abstract with respect to may-testing.

Our model is based on *game semantics* [2, 11, 21]. This has proved successful in giving precise (fully abstract) models of higher-order programming languages with many different features, including concurrency [7, 17]. Our semantics opens the door to a range of (largely theoretical) applications: by formalizing some of the categorical and algebraic structures required to capture higher-order concurrency, it can contribute to the development of general theories, whilst also potentially being the basis for more specific forms of program analysis, already developed for various games models, such as control and information flow analysis [19], and model-checking of properties such as program equivalence [5, 6].

* This research was supported by UK EPSRC grant GR/S72181.

1.1 Related Work

Our semantics is given for a language with syntax based on Reppy’s Concurrent ML [24]: it may also be viewed as a programming language variant of the higher-order π -calculus [25]. Both languages (or fragments thereof) have been investigated using operational techniques [25, 4, 12, 13], giving, for example, labelled transition systems which are closely related to game semantics. Indeed, our semantics may be viewed as a trace semantics, defined compositionally. (See [17] for an explicit comparison between game and trace semantics of the π -calculus.)

Our model uses (and adapts) a variety of notions from game semantics, including the representation of call-by-value types introduced by Honda and Yoshida [9]. In common with earlier models of shared-variable concurrency [7] and the π -calculus [17], we represent terms up to asynchronous, may-testing observation as sets of justified sequences (traces) closed under a preorder. Particularly significant for the current work is the development of a category of “ ν -arenas and ν -strategies” acted upon by the group of natural number automorphisms [15, 16], with which the manipulation and generation of *names* can be interpreted (in the current case, channel names). A similar setting for the interpretation of the ν -calculus (“nominal games”) is described in [1]. A certain degree of parametricity is implicit in the representation of values at a range of types as names, and there are parallels in this respect between our games and the game semantics of polymorphism described by Hughes [10].

1.2 Contribution of this Paper

The main technical contribution of this paper is to show how the game semantics of freshly generated names developed in [15, 16] may be used to model the passing of values of all types on named, typed channels. The construction which makes this possible is a notion of “tree arena” which has ν -arenas as its nodes, each of which has as its children a “dependent arena” for each name which may be mentioned by each move. Using a name allows interaction to take place in its dependent arena, corresponding to message passing on the associated channel. We define a category of games in which we define “parallel composition plus hiding” of strategies to allow interaction both at top level, and within dependent arenas with names which have been made public. We then define the structure of a categorical model of the call-by-value λ -calculus (a premonoidal closed category), and interpret the key operations of our language (spawning of threads, generation of channels, sending and receiving of messages) as simple strategies. We show that our interpretation is sound and adequate with respect to may-testing.

We then prove that the bounded elements of our semantics are definable as terms of \mathcal{L} , and hence that it is fully abstract with respect to may-testing. The key to the proof of definability is the observation that justification pointers may be encoded as names using a series of definable retractions. This is a semantic counterpart of Sangiorgi’s *triggering* translation from higher-order processes into the π -calculus [25]. We may then give a simple proof that “pointer-free” sequences are definable as π -calculus-like terms.

2 A Language with Higher-Order Concurrency

The programming language \mathcal{L} which we shall interpret contains several of the key features of Reppy's CML [24]: new thread generation, new channel declaration (it omits thread identifiers, which are readily expressible using channel names, and event types). Thus it is similar both to Ferreira, Hennessey and Jeffrey's μCML [4] and to Jeffrey and Rathke's $\mu\nu\text{CML}$ [13]. Unlike these languages, communication is *asynchronous* (this leads to a model which is simpler to present, although it would seem relatively straightforward to give a synchronous version).

The types of \mathcal{L} are given by the following grammar:

$$S, T ::= B \mid S * T \mid S \Rightarrow T \mid \text{chan}[T]$$

where B is a set of basic types including at least `unit`, `bool`, and an empty type `0`, which we shall use as the type of threads which do not return a value. The syntax and typing judgements of \mathcal{L} are those of the typed λ -calculus extended with the following constants:

Pairing `pair` : $S \Rightarrow T \Rightarrow S * T$,
Projections `fst` : $S * T \Rightarrow S$ and `snd` : $S * T \Rightarrow T$,
Atomic Values `()` : `unit` and `tt`, `ff` : `bool`,
Conditional `If` : `bool` $\Rightarrow (T * T) \Rightarrow T$
Equality Testing `If eq` : $\text{chan}[T] * \text{chan}[T] \Rightarrow \text{bool}$,
Channel Declaration `newT` : `unit` $\Rightarrow \text{chan}[T]$,
Thread Creation `If spawn` : $(\text{unit} \Rightarrow 0) \Rightarrow \text{unit}$,
Send `send` : $\text{chan}[T] * T \Rightarrow 0$,
Receive `recv` : $\text{chan}[T] \Rightarrow T$.

We use the syntactic sugar (M, N) for `(pair M) N`, $\nu x.M$ for $(\lambda x.M) (\text{new } ())$, `nil` for $\nu c.\text{recv } c$, $M = N$ for `(eq (M, N))`, $M|N$ for `(spawn (\lambda x.M))`; N and `let (x, y) = M in N` for `((\lambda x.\lambda y.N) fst(M)) snd(N)`.

We may express recursively defined functions by passing higher-order values:

$$\mu f.\lambda x.M =_{df}$$

$$\nu c.\text{let } f = (\lambda x.\text{let } g = \text{recv } c \text{ in } (\text{send } (c, g)|g x)) \text{ in } (\text{send } (c, \lambda x.M)|f).$$

In particular, we define the replicated send operation: $!\text{send } M =_{df} \mu f.\lambda x.(\text{send } (c, M)|f ())$.

We also define non-deterministic erratic choice: $M \text{ or } N =_{df} \nu c.\text{send } (c, \text{tt})|\text{send } (c, \text{ff})|\text{If recv } c \text{ then } M \text{ e}$

2.1 Operational Semantics

Let \mathcal{L}^* be the extension of \mathcal{L} with an infinite set of channel names for each type $\text{chan}[T]$. Given a program M of \mathcal{L}^* and set of names \mathcal{N} , we write $\mathcal{N} \vdash M$ if every name in M occurs in \mathcal{N} . A *configuration* of \mathcal{L} consists of a multiset \mathcal{T} of programs or *threads* M_1, \dots, M_n (of which at most one has non-empty type), and a set \mathcal{N} of typed channel names such that $\mathcal{N} \vdash M_i$ for $1 \leq i \leq n$. The *values* of \mathcal{L}^* are given by the grammar:

$$U, V ::= v \mid n \mid C \mid \text{pair } V \mid \text{pair } V \mid \lambda x.M$$

(where v ranges over base type values, n over names and C over constants). The

evaluation contexts are:

$$E[-] ::= [-] \mid E[-] M \mid V E[-]$$

The “small step” evaluation rules for evaluating configurations are shown in Table 1. We write $M \Downarrow$ (M may converge) if $M, 0 \rightarrow (T, V), \mathcal{N}$ for some T, V, \mathcal{N} .

$T, E[(\lambda x.M) V], \mathcal{N}$	\longrightarrow	$T, E[M[V/x]], \mathcal{N}$
$T, E[\mathbf{fst}(U, V)], \mathcal{N}$	\longrightarrow	$T, E[U], \mathcal{N}$
$T, E[\mathbf{snd}(U, V)], \mathcal{N}$	\longrightarrow	$T, E[V], \mathcal{N}$
$T, E[(\mathbf{eq}(a, a))], \mathcal{N}$	\longrightarrow	$T, E[\mathbf{tt}], \mathcal{N}$
$T, E[\mathbf{eq}(a, b)], \mathcal{N}$	\longrightarrow	$T, E[\mathbf{ff}], \mathcal{N}$, if $a \neq b$
$T, E[\mathbf{If} \mathbf{tt}], \mathcal{N}$	\longrightarrow	$T, E[\mathbf{fst}], \mathcal{N}$
$T, E[\mathbf{If} \mathbf{ff}], \mathcal{N}$	\longrightarrow	$T, E[\mathbf{snd}], \mathcal{N}$
$T, E_1[\mathbf{send}(a, V)], E_2[\mathbf{recv}(a)], \mathcal{N}$	\longrightarrow	$T, E_2[V], \mathcal{N}$
$T, E[\mathbf{spawn}(V)], \mathcal{N}$	\longrightarrow	$T, V(), E[()], \mathcal{N}$
$T, E[\mathbf{new}_T()], \mathcal{N}$	\longrightarrow	$T, E[a], \mathcal{N} \cup \{(a, \mathbf{chan}[T])\}$ ($(a, \mathbf{chan}[T]) \notin \mathcal{N}$)

Table 1. Operational Semantics of \mathcal{L}

Thus we define observational approximation and equivalence with respect to may-testing, for terms $M, N : T$:

$M \lesssim N$ if $C[M] \Downarrow$ implies $C[N] \Downarrow$ for all compatible closing contexts $C[-] : \mathbf{unit}$.

$M \simeq N$ if $M \lesssim N$ and $N \lesssim M$.

3 Game Semantics

Our notion of game is based on the dialogue games of Hyland and Ong [11] (and Nickau [21]), developed in e.g. [20, 9] and extended in [15] with structure for manipulating a countable set of names, in the form of an action of the automorphism group of the natural numbers. A significant departure from these games for sequential languages arises because in the concurrent setting there are moves which may be played by either participant in a dialogue. So polarity (Player/Opponent labelling) is not intrinsic to arenas but to interactions.

An (underlying) arena A is a tuple $(M_A, \lambda_A, \vdash_A)$ consisting of a set of moves M_A , a question/answer labelling $\lambda_A : M_A \rightarrow \{Q, A\}$ and an *enabling relation* $\vdash_A \subseteq M_A \times M_A$ such that no answer enables an answer. We write M_A^I for the subset of M_A of consisting of moves with no enabling move (the *initial* moves), and say that an arena is *A-rooted* if all such moves are answers.

A justified sequence over the arena A is a sequence of moves of A together with a “justification pointer” from each non-initial move to some enabling move.

Definition 1. A (partial or total) polarization for a justified sequence s is a (partial or total) labelling of the occurrences of moves in s as belonging to Player and Opponent (concretely, a function λ^{OP} from the non-empty prefixes of s to $\{P, O\}$) such that the justifier of any Player move is an Opponent move and vice-versa. A polarized sequence is a justified sequence with a total polarization.

Given a polarized sequence s , we write s^\perp for the polarized sequence in which the labelling is reversed.

3.1 ν -Arenas

We now recall the notion of ν -arena introduced in [15, 16]. Let G be the topological group of automorphisms on \mathbb{N} with the product topology on $\mathbb{N}^{\mathbb{N}}$.

Definition 2. A ν -arena (A, π) is an underlying arena A together with an action of G on $M_A(\cdot)$ which is continuous (with respect to the discrete topology on M_A), and such that $\lambda_A(\pi \cdot m) = \lambda_A(m)$ and $m \vdash n$ iff $\pi \cdot m \vdash \pi \cdot n$.

Continuity of the group action is equivalent to requiring that the stabiliser of any move m is open in G and thus equal to the stabiliser of a finite subset $k \subseteq \mathbb{N}$, the *support* of m . This is the set of names actually mentioned by m , for which we write $\nu(m)$. By preservation of the labelling and enabling relation on moves we obtain a continuous action of G on justified sequences of each ν -arena A : $\pi \cdot (m_1 m_2 \dots m_n) = (\pi \cdot m_1)(\pi \cdot m_2) \dots (\pi \cdot m_n)$. We write \sim for the equivalence relation on legal sequences determined by the orbits of the group action — i.e. $s \sim t$ if $\exists \pi \in G. \pi \cdot s = t$.

A key example is the ν -arena of names N , in which the set of moves is the set of natural numbers (all of which are initial moves), with the canonical action of G upon \mathbb{N} — i.e. $M_N = M_N^I = \mathbb{N}$, $\lambda(i) = A$ for all i , and $\pi \cdot i = \pi(i)$.

For each ν arena, we define functions P_ν, O_ν from the set of polarized sequences over A to $\mathcal{P}_{fin}(\mathbb{N})$ which identify the sets of new names introduced by Player and, respectively, Opponent.

- $P_\nu(\varepsilon) = \emptyset$,
- $P_\nu(sa) = P_\nu(s) \cup (\nu(sa) - \nu(s))$ if a is Player move,
- $P_\nu(sa) = P_\nu(s)$ otherwise.
- $O_\nu(s) = \nu(s) - P_\nu(s)$.

3.2 Tree Arenas

In order to use names to represent channel types, we introduce a notion of “tree arena”, in which each node is an arena, from which there are branches for each name occurring in the support of each move. Essentially, playing a move which mentions a name with a given “dependent arena” allows both Player and Opponent to commence play in that arena, corresponding to sending (if Player starts) or receiving (if Opponent starts) a value on the associated channel.

Definition 3. A (finite-depth) tree arena is a ν -arena A , together with an indexed set $\{\alpha(m)_i \mid i \in \nu(m)\}$ of tree arenas for each move $m \in M_A$, with the properties:

- Invariance with respect to G -action: for any $\pi \in G$, $m \in M_A$ and $i \in \nu(m)$, $\alpha(m)_i = \alpha(\pi \cdot m)_{\pi(i)}$.

- *Finite Depth*: there is no infinite chain of arenas $A \ll A_1 \ll A_2 \ll \dots$, where $B \ll C$ if there exists $m \in M_B$ and $i \in \nu(m)$ such that $\alpha(m)_i = C$.

Thus, for example, for any tree arena A , we may form the tree arena $Ch(A)$ which has as its root node the arena N , and as its children, copies of the arena A — i.e. $ch(A) = (N, \{\{\alpha(i)_i \mid i \in \mathbb{N}\})$, where $\alpha(i)_i = A$ for all i .

Definition 4. We refer to the set of nodes of a tree arena as its dependent arenas. Formally, this is defined by induction on tree depth as follows:

$$|A| = \{\alpha(m)_i \mid m \in M_A \wedge i \in \nu(m)\} \cup \bigcup \{|\alpha(m)_i| \mid m \in M_A \wedge i \in \nu(m)\}.$$

We obtain a tree arena \hat{A} — the expansion of A — by explicitly adding \mathbb{N} -indexed copies of the dependent arenas of A to the top node. More precisely, we define:

- $M_{\hat{A}} = M_A + \{\langle i, A, m \rangle \in \mathbb{N} \times |A| \times \bigcup \{M_B \mid B \in |A|\} \mid m \in M_B \wedge i \notin \nu(m)\}$,
- $m \vdash_{\hat{A}} \text{in}_1(n)$ if $m = \text{in}_1(m')$ and $m' \vdash_A n$.
- $m \vdash_{\hat{A}} \text{in}_r(\langle i, B, n \rangle)$, if $m = \text{in}_r(\langle i, B, m' \rangle)$, where $m' \vdash_B n$,
- $\lambda_{\hat{A}}(\text{in}_1(m)) = \lambda_A(m)$,
- $\lambda_{\hat{A}}(\text{in}_r(\langle i, B, m \rangle)) = \lambda_B(m)$,
- $\pi \cdot \text{in}_1(m) = \text{in}_1(\pi \cdot_A m)$,
- $\pi \cdot \text{in}_r(\langle i, B, m \rangle) = \text{in}_r(\pi(i), \pi \cdot_B m)$
- $\alpha^{\hat{A}}(\text{in}_r(m))_i = \alpha^A(m)_i$
- $\alpha^{\hat{A}}(\text{in}_1(\langle i, B, m \rangle))_i = B$, $\alpha^{\hat{A}}(\text{in}_1(\langle i, B, m \rangle))_j = \alpha^B(m)_j$ if $j \neq i$.

We refer to moves of the form $\langle i, m \rangle$ as dependent moves. The name of $\langle i, m \rangle$ is i .

Definition 5. A legal sequence over the tree arena A is a justified sequence s over \hat{A} , satisfying the conditions:

Uniformity Every occurrence of a name refers to the same arena: if $ta, t'a' \sqsubseteq s$ and $i \in \nu(a) \cap \nu(a')$ then $\alpha(a)_i = \alpha(a')_i$.

Dependency The name of any dependent move has already occurred in s : if $t\langle i, a \rangle \sqsubseteq s$ then $i \in \nu(t)$.

Well-openedness s contains at most one initial and non-dependent move.

Well-answering Every question in s justifies at most one answer.

A negative sequence is a polarized legal sequence in which the unique initial move is an Opponent move. We write L_A for the set of legal sequences, and L_A^- for the set of negative sequences over A .

To represent the behaviour of strategies “up to asynchronous observation” requires saturation under a preorder \preceq on polarized sequences as in e.g. [3, 14]. This is defined to be the least preorder on polarized sequences such that:

- If $\lambda(a) = O$ then $sabt \preceq sbat$ and if $\lambda^{OP}(a) = P$ then $sbat \preceq sbat$.
- If $\lambda(a) = O$ then $sat \preceq st$, and if $\lambda(a) = P$, then $t \preceq sat$.

Note that $s \preceq t$ if and only if $t^\perp \preceq s^\perp$.

Definition 6. Let A be a tree arena. A strategy $\sigma : A$ is a non-empty set of negative sequences over A satisfying:

Prefix closure If $s \in \sigma$ and $t \sqsubseteq s$ then $t \in \sigma$.

\sim -closure If $s \in \sigma$ and $s \sim t$ then $t \in \sigma$.

\preceq -closure If $s \in \sigma$ and $t \preceq s$ then $t \in \sigma$.

We write $\ker(\sigma)$ for the set of \preceq -minimal sequences of σ — i.e. $\{s \in \sigma \mid \forall t \in \sigma. s \preceq t \implies t \preceq s\}$.

4 Denotational Semantics

We will now construct a *premonoidal* closed category [23] of tree arenas and strategies in which to model the call-by-value λ -calculus. This follows the constructions of Honda and Yoshida [9] or variants described by Laurent [18], and (for ν -arenas) [15]. In each case the group action and dependent arena structure on compound arenas is defined pointwise. Play in the function-space arena $A_1 \rightarrow A_2$ starts on the left (by labelling the initial moves of A_1 as questions which enable the initial moves of A_2).

Definition 7. Given tree arenas A_1, A_2 we define the (Q -rooted) call-by-value function-space tree-arena $A_1 \rightarrow A_2$ as follows:

- $M_{A_1 \rightarrow A_2} = M_{A_1} + M_{A_2}$,
- $\lambda_{A_1 \rightarrow A_2}(\text{in}_i(m)) = Q$, if $i = 1$ and $m \in M_{A_2}^I$,
- $\lambda_{A_1 \rightarrow A_2}(\text{in}_i(m)) = \lambda_{A_i}(m)$, otherwise,
- $m \vdash_{A_1 \rightarrow A_2} \text{in}_i(n)$ if $m = \text{in}_i(m')$ and $m' \vdash_{A_i} n$ or $i = 2$, $n \in M_{A_2}^I$ and $m = \text{in}_1(m')$, where $m' \in M_{A_1}^I$.
- $\pi \cdot \text{in}_i(m) = \text{in}_i(\pi \cdot m)$.
- $\alpha(\text{in}_i(m))_j = \alpha^{A_i}(m)_j$.

Examples:

New Channel Generation For each arena A , we have a strategy $\text{new} : I \rightarrow \text{ch}(A)$ (where I is the arena I with a single initial answer move). This responds to Opponent’s initial question by generating a fresh name and making it public (i.e. playing an arbitrary move in N) — thus its set of \preceq -minimal sequences is $\{\varepsilon, q\} \cup \{qi \mid i \in \mathbb{N}\}$.

Message Receiving For each A -rooted arena A , we have a strategy $\text{recv} : \text{ch}(A) \rightarrow A$ which responds to Opponent’s initial question — which supplies the a channel name i — by playing copycat between A and the dependent arena of i . (See Fig. 1.)

To define the composition of strategies $\sigma : A \rightarrow B$ and $\tau : B \rightarrow C$ we need to allow interaction both in the shared “public arena” B and in the dependent arenas. In addition, we impose the “freshness conditions” introduced in [15, 16] to ensure that the new names introduced by σ are disjoint from those introduced by τ , and that both are disjoint from those introduced by Opponent.

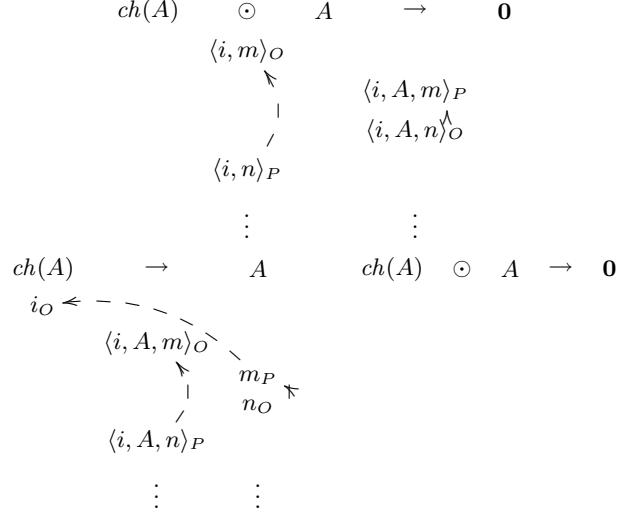


Fig. 1. Typical plays of `snd` and `recv`

Definition 8. An interaction sequence is a justified sequence t with two partial polarizations λ^L and λ^R such that:

- Every move has at least one polarity: for all $s \sqsubseteq t$, $\lambda^L(s) \downarrow$ or $\lambda^R(s) \downarrow$.
- If a move has two polarities then they are opposite: there is no $s \sqsubseteq t$ such that $\lambda^L(s) = \lambda^R(s)$.

and satisfying the following “freshness conditions” for name introduction [15]:

- i** $P_\nu(s \upharpoonright L) \cap P_\nu(s \upharpoonright R) = \emptyset$,
- ii** $(P_\nu(s \upharpoonright L) \cup P_\nu(s \upharpoonright R)) \cap O_\nu(s \upharpoonright L \Delta R) = \emptyset$.

(Where we write $s \upharpoonright L$ (resp. $s \upharpoonright R$) for the polarized sequence obtained by restricting to moves for which λ^L is defined, and $s \upharpoonright L \Delta R$ for the restriction to moves for which only one of λ^L, λ^R is defined.)

Let $I_{A,B,C}$ be the set of interaction sequences which are legal sequences of $(A \rightarrow B) \rightarrow C$. Given $\sigma : A \rightarrow B, \tau : B \rightarrow C$, we may now define: $\sigma; \tau : A \rightarrow C = \{s \in L_{A \rightarrow C}^- \mid \exists t \in I_{A,B,C}. t \upharpoonright L \in \sigma \wedge t \upharpoonright R \in \tau \wedge s = t \upharpoonright L \Delta R\}$.

We prove that composition is well-defined and associative following the standard arguments used in [11, 20, 9], extended to ν -strategies in [16]. The identity strategy (for A -rooted arenas) $\text{id}_A : A \rightarrow A$ consists of the \preceq closure of the set of copycat sequences: $\ker(\text{id}_A) = \{s \in (L_{A \rightarrow A}^-)^E \mid \forall t \sqsubseteq^E s. t \upharpoonright A^+ = t \upharpoonright A^-\}$ (where $s \sqsubseteq^E t$ if s is an even-length prefix of t). Thus we may form a category \mathcal{G} in which objects are A -rooted tree arenas and morphisms from A to B are strategies on $A \rightarrow B$.

Proposition 1. \mathcal{G} is a well-defined category.

\mathcal{G} has all small coproducts, given by the “disjoint union” of arenas, and an initial object, the empty arena $\mathbf{0}$, containing no moves. We define *premonoidal* structure on \mathcal{G} based on that described in [9, 18, 15], in which initial moves of $A \odot B$ are pairs of initial moves from A and B , but non-initial moves are either from A or from B .

Definition 9. *From A -rooted tree arenas A_1, A_2 , we form the tree arena $A_1 \odot A_2$:*

- $M_{A_1 \odot A_2} = \{(m, n) \in (M_{A_1} \times M_{A_2}^I) \cup (M_{A_1}^I \times M_{A_2}) \mid i \in \nu(m) \cap \nu(n) \implies \alpha(m)_i = \alpha(n)_i\}$,
- $\lambda_{A_1 \odot A_2}(\langle m_1, m_2 \rangle) = \lambda_{A_2}(m_2)$, if $m_1 \in M_{A_1}^I$,
- $\lambda_{A_1 \odot A_2}(\langle m_1, m_2 \rangle) = \lambda_{A_1}(m_1)$, otherwise,
- $(m_1, m_2) \vdash_{A_1 \odot A_2} (n_1, n_2)$ if $m_1 = n_1 \in M_{A_1}^I$ and $m_2 \vdash_{A_2} n_2$ or $m_2 = n_2 \in M_{A_2}^I$ and $m_1 \vdash_{A_1} n_1$,
- $\pi \cdot \langle m, n \rangle = \langle \pi \cdot m, \pi \cdot n \rangle$,
- $\alpha(m, n)_i = \alpha^{A_1}(m)_i$, if $i \in \nu(m)$,
- $\alpha(m, n)_i = \alpha^{A_2}(n)_i$, otherwise.

Examples:

Equality Testing For each object A , we have a strategy $\text{eq} : ch(A) \odot ch(A) \rightarrow I + I$ which is supplied by Opponent with a pair of names and responds with $\text{in}_l(*)$ if they are equal and $\text{in}_r(*)$ otherwise: $\text{eq} = \{\langle i, i \rangle \text{in}_l(*) \mid i \in \mathbb{N}\} \cup \{\langle i, j \rangle \text{in}_r(*) \mid i, j \in \mathbb{N} \wedge i \neq j\}$

Message Receiving For each object A , we have a strategy $\text{send} : ch(A) \odot A \rightarrow \mathbf{0}$ which is supplied with a channel name i and an initial move m in A , and plays it as an initial move i, m in the dependent arena for A , and henceforth plays copycat between the explicit and dependent occurrences of A . (See Fig. ??.)

We now define premonoidal structure on \mathcal{G} , following [9]. For each object A we define an endofunctor $_{-} \odot A : \mathcal{G} \rightarrow \mathcal{G}$: given $\sigma : B \rightarrow C$, $\sigma \odot A : B \odot A \rightarrow C \odot A = \{s \in L_{B \odot A \rightarrow C \odot A} \mid s \downarrow A, A \in \sigma \wedge s \uparrow A, A \in \text{id}_A \wedge P_\nu(s \downarrow A, A) = P_\nu(s)\}$ where $s \downarrow A, A$ is obtained from $s \in L_{B \odot A \rightarrow C \odot A}$ by taking the left projection from each (non-dependent) move and erasing all initial moves of B, C except the opening move and its answer from the result, and $s \uparrow A, A$ is obtained by erasing all non-dependent moves, taking the right projection from each move and erasing all initial moves of A, A except the opening move and its answer from the result.

Proposition 2. *(\mathcal{G}, I, \odot) is a symmetric premonoidal category.*

We now identify, via conditions on strategies, a category of arenas for which the premonoidal product is Cartesian. The first condition is *totality*, as in [9].

Definition 10. *A morphism $f : A \rightarrow B$ is total if $g; f = \perp$ implies $g = \perp$ (where \perp is the \preceq -closure of $\{\varepsilon\}$). So $\sigma : A \rightarrow B$ is total if it may respond to each initial question in A with an answer in B .*

A sequence $qas \in L_{A \rightarrow B}$ is total if a is a (initial) move in B and single-threaded if:

- Player does not introduce any new names with the move a — i.e. $P_\nu(qa) = \emptyset$.
- There is at most one move justified by a in s .

A total strategy σ is single-threaded if every sequence of at least two moves in $\ker(\sigma)$ is single-threaded and $qa, qa' \in \ker(\sigma)$ implies $a = a'$.

To define the composition of single-threaded strategies we apply a “promotion” operation $(_)^\dagger$.

Definition 11. Given a strategy $\sigma : A$, let σ^\dagger be the least subset of L_A such that for any interaction sequence s , if $qa(s\upharpoonright L) \in \sigma^\dagger$, $qa(s\upharpoonright R) \in \sigma$ and $qa(s\upharpoonright L\Delta R) \in L_A$ then $qa(s\upharpoonright L\Delta R) \in \sigma^\dagger$.

We define a category \mathcal{G}_t with tree arenas as objects and single-threaded total strategies on $A \rightarrow B$ as morphisms from A to B . Composition of $\sigma : A \rightarrow B$ and $\tau : B \rightarrow C$ is defined $\sigma^\dagger; \tau$, and the identity on A is the restriction of $\text{id}_A^{\mathcal{G}}$ to single-threaded sequences. To prove that \mathcal{G}_t is a category, we establish the following properties for any σ, τ (following e.g. [20]):

- $\text{der}_A^\dagger = \text{id}_A$,
- $\sigma^\dagger; \text{der}_A = \sigma$,
- $\sigma^\dagger; \tau^\dagger = (\sigma^\dagger; \tau)^\dagger$.

We also note that $_ \odot _$ is a Cartesian product on \mathcal{G}_t , and by the above properties, $(_)^\dagger$ is a functor from \mathcal{G}_t to \mathcal{G} .

Proposition 3. $(\mathcal{G}, \mathcal{G}_t, (_)^\dagger)$ is a Freyd category [22] (a symmetric premonoidal category \mathcal{G} , a Cartesian category \mathcal{G}_t , and an identity-on-objects strict symmetric premonoidal functor from \mathcal{G}_t to \mathcal{G}).

Moreover, it is a closed Freyd category: the functor $(_)^\dagger \odot A : \mathcal{G}_t \rightarrow \mathcal{G}$ has a right adjoint $A \multimap _ : \mathcal{G} \rightarrow \mathcal{G}_t$.

Definition 12. For any Q -rooted tree arena B , let $\uparrow B$ be the arena obtained by adding to B a single initial answer (invariant under G action) which enables all of the initial moves of B — i.e.

- $M_{\uparrow A} = \{a\} + M_A$,
- $\lambda_{\uparrow A} = [\{a, A\}, \lambda_A]$,
- $\vdash_{\uparrow A} = \{(\text{in}_l(m), \text{in}_r(n)) \mid (m = a \wedge n \in M_A^I) \vee \langle m, n \rangle \in \vdash_A\}$,
- $\pi \cdot \text{in}_l(a) = \text{in}_l(a), \pi \cdot \text{in}_r(m) = \text{in}_r(\pi \cdot m)$.
- $\alpha(\text{in}_r(m))_i = \alpha^A(m)_i$, if $i \in \nu(m)$.

We then define $A \multimap B = \uparrow (A \rightarrow B)$.

Thus, for example, the arena $I \multimap \mathbf{0}$ consists of an initial answer which enables a single question. So we have a strategy $\text{spawn} : (I \multimap \mathbf{0}) \rightarrow I$ which responds to the initial Opponent question by concurrently answering it and playing the (unique) initial question in $I \rightarrow \mathbf{0}$. (See Fig. 2.)

Proposition 4. $A \multimap _$ is right adjoint to $(_)^\dagger \odot A : \mathcal{G}_t \rightarrow \mathcal{G}$.

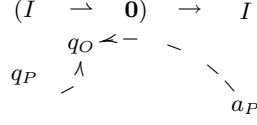


Fig. 2. A typical play of spawn

Proof. There is an obvious bijection from (non-empty) legal sequences on $A \odot B \rightarrow C$ to single-threaded sequences on $A \rightarrow (B \rightarrow C)$, sending $\langle m, n \rangle$ s to $mans$. Thus for each morphism $\sigma : A \odot B \rightarrow C$, we define a unique single-threaded strategy $\Lambda(\sigma) : A \rightarrow (B \rightarrow C) = \{\varepsilon\} \cup \{ma \mid m \in M_A^I\} \cup \{mans \mid \langle m, n \rangle \in \sigma\}$ such that $(\Lambda(\sigma))^\dagger \odot B; \mathbf{app}_{B,C} = \sigma$, where the co-unit $\mathbf{app}_{B,C} : (B \rightarrow C) \odot B \rightarrow C$ is derived from $\mathbf{id}_{B \rightarrow C}$ by the above bijection.

Thus we have a model of the call-by-value λ -calculus, and so we may interpret terms $x_1 : S_1, \dots, x_n : S_n \vdash M : T$ of \mathcal{L} as morphisms from $\llbracket S_1 \rrbracket \odot \dots \odot \llbracket S_n \rrbracket$ to $\llbracket T \rrbracket$ in \mathcal{G} . by fixing the interpretation of the types:

Base types $\llbracket 0 \rrbracket = \mathbf{0}$, $\llbracket \mathbf{unit} \rrbracket = I$ and $\llbracket \mathbf{bool} \rrbracket = I + I$,

Function types $\llbracket S \Rightarrow T \rrbracket = \llbracket S \rrbracket \rightarrow \llbracket T \rrbracket$,

Channel Types $\llbracket \mathbf{chan}[T] \rrbracket = \mathbf{ch}(\llbracket T \rrbracket)$.

and of the constants as the key strategies already defined for channel-generation, equality testing, thread-spawning and sending and receiving values.

4.1 Soundness and Adequacy

In order to prove soundness and adequacy with respect to may-testing (i.e. $M \Downarrow$ if and only if $M \neq \perp$) we define the interpretation of configurations $(\mathcal{T}, \mathcal{N})$.

Given strategies $\sigma : A \rightarrow \mathbf{0}$ and $\tau : A \rightarrow B$, we define the asymmetric interleaving $\sigma \parallel \tau : A \rightarrow B$ to consist of sequences $qs \in L_{A \rightarrow B}$ such that there exists an interaction sequence t with $q(t \uparrow L) \in \sigma$, $q(t \uparrow R) \in \tau$ and $q(t \uparrow L \Delta R) = qs$. Then $f \parallel (g; h) = (f \parallel g); h$, and we also have the following key lemma for **spawn**.

Lemma 1. *For any $f : A \rightarrow \mathbf{0}$, and $g : I \rightarrow B$, $\Lambda(f); \mathbf{spawn}; g = f \parallel t_A; g$, (where $t_A : A \rightarrow I$ is the terminal map in the category of single threaded strategies).*

We interpret the configuration $M_1 : \mathbf{0}, \dots, M_n : \mathbf{0}, N : S, \{a_1 : \mathbf{chan}[T_1], \dots, a_m : \mathbf{chan}[T_m]\}$ as $\mathbf{new}_{[T_1]} \odot \dots \odot \mathbf{new}_{[T_m]}; (\llbracket M_1 \rrbracket \parallel \dots \parallel (\llbracket M_n \rrbracket \parallel \llbracket N \rrbracket))$.

Lemma 2. *If $C \longrightarrow C'$ then $\llbracket C \rrbracket \subseteq \llbracket C' \rrbracket$.*

Proof. We show that we may interpret evaluation contexts $a_1 : T_1, \dots, a_n : T_n \vdash E[\cdot : S] : T$ as morphisms $\llbracket E[\cdot] \rrbracket : \llbracket S \rrbracket \odot \llbracket T_1 \rrbracket \odot \dots \odot \llbracket T_n \rrbracket \rightarrow \llbracket T \rrbracket$ so that $\llbracket E[M] \rrbracket = \delta_{\llbracket T_1 \rrbracket \odot \dots \odot \llbracket T_n \rrbracket}; (\llbracket M \rrbracket \odot (\llbracket T_1 \rrbracket \odot \dots \odot \llbracket T_n \rrbracket))$; $\llbracket E[\cdot] \rrbracket$ and verify soundness for each reduction of the operational semantics using the categorical structure of \mathcal{G} and the following (in)equations:

Communication $\pi_r \subseteq \text{send}(\pi_l; \text{recv})$,
Equality $\text{new}_A; \langle \text{id}_{ch(A)}, \text{id}_{ch(A)}, \text{id}_{ch(A)} \rangle^\dagger; \text{eq} \odot ch(A) = \text{new}_A; (\text{in}_l \odot ch(A))$,
Inequality $(\text{new}_A \odot ch(A)); \langle \pi_l, \pi_r, \pi_l, \pi_r \rangle^\dagger; (\text{eq} \odot (ch(A) \odot ch(A))) = (\text{new}_A \odot ch(A)); \text{in}_r \odot (ch(A) \odot ch(A))$.

Thus $M \Downarrow$ implies $M \neq \perp$. To prove computational adequacy — if $\llbracket \mathcal{M} \rrbracket \neq \perp$ then $M \Downarrow$, we first observe that \mathcal{G} is cpo-enriched with the inclusion order on strategies. We then establish the following by analysis of the semantics of configurations in which every thread is either sending or receiving a value.

Lemma 3. $\llbracket C \rrbracket = \bigcup \{ \llbracket C' \rrbracket \mid C \longrightarrow C' \}$.

We then define a translation which allows us to count internal reductions of M as **recv** actions, and hence allows adequacy to be proved by a simple induction on this measure.

Definition 13. Fix a variable $c : \text{chan}[\text{unit}]$, we define a translation from each term $\Gamma \vdash M : T$ not containing c to a term $\Gamma, c \vdash M^c : T$:

- $x^c = x$ for each variable $x \neq c$,
- $(MN)^c = (\text{recv } c); (M^c N^c)$
- $(\lambda x.M)^c = \lambda x.M^c$
- $C^c = C$ for each constant C .

We may then prove by structural induction that:

Lemma 4. For every term $\Gamma \vdash M : T$, $\llbracket M \rrbracket = \llbracket \nu c. !\text{send}(c, ()) \mid M^c \rrbracket$.

Defining $\text{send}^1 M = \text{send } M$ and $\text{send}^{i+1} M = \text{send } M \mid \text{send}^i M$, we have $\llbracket !\text{send } V \rrbracket = \bigcup_{i \in \mathbb{N}} \llbracket \text{send}^i V \rrbracket$.

Lemma 5. If $\llbracket \nu c. \text{send}^n(c, ()) \mid M^c \rrbracket \neq \perp$ then $M \Downarrow$.

Proof. By induction on n , using Lemma 3 and analysis of possible configurations.

Proposition 5. If $\llbracket C \rrbracket \neq \perp$ then $M \Downarrow$.

Proof. If $\llbracket M \rrbracket \neq \perp$ then $\llbracket \text{newc}. !\text{send}(c, ()) \mid M^c \rrbracket \neq \perp$. By continuity, there exists n such that $\llbracket \text{newc}. \text{send}^n(c, ()) \mid M^c \rrbracket$. So by Lemma 5, $M \Downarrow$ as required.

5 Definability and Full Abstraction

We prove full abstraction by establishing that for any legal sequence s over a type-object, the least strategy containing s (the closure of $\{s\}$ under the relations \sqsubseteq , \sim and \preceq) is the denotation of a term. This is sufficient to define “tests” to distinguish any pair of distinct strategies. In fact, since we may define the union of definable strategies as a non-deterministic choice between the corresponding terms, we may establish definability for all strategies σ for which the length of sequences in $\ker(\sigma)$ is bounded.

Proposition 6. For any type T there is a type \overline{T} and a definable retraction $(\text{inj}_T, \text{proj}_T) : T \trianglelefteq \overline{T}$ such that for all $s \in \llbracket T \rrbracket \rightarrow \mathbf{0}$ there exists a pointer-free $\overline{s} \in \llbracket x \vdash \text{proj}_T x \rrbracket; \lceil s \rceil$ such that $\llbracket x \vdash \text{inj}_T x \rrbracket; \lceil \overline{s} \rceil = \lceil s \rceil$.

Proof. We define \overline{T} by structural induction. For each base type B , $\overline{B} = B$; $\overline{S * T} = \overline{S} * \overline{T}$.

Using the retraction (in, out) , we define $\overline{S \Rightarrow T} = \text{chan}[\overline{S} * \text{chan}[\overline{T}]]$:
 $\text{inj}_{S \Rightarrow T} = \lambda f. \text{in}(\lambda x. \text{inj}_T(f(\text{proj}_S x)))$ and $\text{proj}_{S \Rightarrow T} = \lambda y. \lambda x. \text{proj}_T((\text{out } y)(\text{inj}_T x))$.

Because $\text{chan}[_]$ is not functorial, it is not the case that $T \trianglelefteq \overline{T}$ implies $\text{chan}[T] \trianglelefteq \text{chan}[\overline{T}]$. Instead, we define $\text{chan}[T] = \text{chan}[T] * \text{chan}[\overline{T}]$, $\text{inj}_{\text{chan}[T]} = \lambda x. \nu c. !\text{send}(c, \text{inj}(\text{rec } v x)) !\text{send}(x, \text{proj}_{\text{chan}[T]}) = \lambda y. !\text{send}(\text{fst}(y), \text{inj}(\text{rec } v \text{snd}(y))) !\text{send}(\text{snd}(y), \text{proj}(\text{rec } v \text{fst}(y))) \text{fst}(y)$.

So given a sequence s in $\llbracket T \rrbracket \rightarrow \mathbf{0}$, if $\lceil \overline{s} \rceil$ is definable as a term $x : \overline{T} \vdash M : \mathbf{0}$ then $\lceil s \rceil$ is definable as $M(\text{inj}_T x)$.

We now show that each such pointer-free strategy is definable, via a decomposition which successively erases dependent moves.

Proposition 7. For any pointer-free $s \in \llbracket T_1 \rrbracket \odot \dots \odot \llbracket T_n \rrbracket \rightarrow \mathbf{0}$, $\lceil s \rceil$ is definable as a term $x_1 : T_1, \dots, x_n : T_n \vdash M : \mathbf{0}$ such that $\llbracket M \rrbracket = \lceil s \rceil$.

Proof. We assume T_1, \dots, T_n are pointed (i.e. base, function or channel types) and define M by induction on the length of s . If this is less than 2, then $\lceil s \rceil = \perp = \llbracket \text{nil} \rrbracket$.

If s has length greater than 2 then $s = \langle a_1, \dots, a_n \rangle \langle i, B, b \rangle s'$, where $i \in \nu(\langle a_1, \dots, a_n \rangle)$ and thus $i = a_j$ for some $1 \leq j \leq n$, and so $B = \llbracket T_j \rrbracket$. So if $T_j = \text{chan}[S_1 * \dots * S_m]$ (where each S_k is pointed) then $b = \langle b_1, \dots, b_m \rangle$ where $b_k \in M_{\llbracket S_k \rrbracket}^I$ for each k . So we may form a legal sequence $s'' = \langle a_1, \dots, a_n, b_1, \dots, b_m \rangle s'$ on $\llbracket T_n \rrbracket \odot \dots \odot \llbracket T_1 \rrbracket \odot \llbracket S_1 \rrbracket \odot \dots \odot \llbracket S_m \rrbracket$. By induction hypothesis, $\lceil s'' \rceil$ is definable as a term $x_1 : T_1, \dots, x_n : T_n, y_1 : S_1, \dots, y_m : S_m \vdash M : \mathbf{0}$

If $\langle i, B, b \rangle$ is an Opponent move then we have $\lceil s \rceil = \llbracket \text{let}(y_1, \dots, y_m) = \text{rec } v x_i \text{ in } M \rrbracket$.

If $\langle i, B, b \rangle$ is a Player move then for each $k \leq m$ we define a term $x_1 : T_1, \dots, x_n : T_n \vdash N_k : S_k$:

- If $S_k = \text{unit}$ then $N_k =_{df} ()$,
- If $S_k = \text{bool}$ then $N_k =_{df} \text{tt}$ if $b_k = \text{in}_1(*)$ and $N_k =_{df} \text{ff}$, otherwise.
- If $S_k = U \Rightarrow V$ then let $N_k =_{df} \lambda x. \text{nil}$.
- If $S_k = \text{chan}[U]$ then $N_k =_{df} x_i$, if $b_k = a_i$ and $b_k \neq x_j$ for $j < i$,
 $N_k =_{df} \text{new}_u()$, if $b_k \neq x_j$ for all $j \leq n$.

For each $j \leq n$ we define a test term $B_j : \text{bool}$:

- If $T_j = \text{bool}$, we define $B_j =_{df} x_i$ if $a_j = \text{in}_1(*)$ and $B_j =_{df} \neg x_i$, otherwise.
- If $T_j = U \Rightarrow V$ or $T_j = \text{unit}$, then $B_j =_{df} \text{tt}$,
- If $T_l = \text{chan}[U]$ then $B_l = \bigwedge_{k \leq n} E_k$, where:
 $E_k =_{df} x_i = x_j$ if $T_j = T_k$ and $a_j = a_k$,
 $E_k =_{df} \neg x_j = x_k$ if $T_j = T_k$ and $a_j \neq a_k$,
 $E_k =_{df} \text{tt}$, otherwise.

Then $\lceil s \rceil$ is definable as: $\text{let } (y_1, \dots, y_m) = (N_1, \dots, N_m) \text{ in If } \bigwedge_{j \leq n} B_j \text{ then } (\text{send } (x_i, y_1, \dots, y_m) | M) \text{ e.}$

Corollary 1. *For any type T , if $ms* \in L_{\llbracket T \rrbracket \rightarrow I}$ then there is a term $x : T \vdash M : \text{unit}$ such that $\llbracket M \rrbracket = \lceil ms* \rceil$.*

Proof. We translate $ms*$ to a sequence $\langle m, i \rangle s \langle i, I, * \rangle_P$ over the arena $\llbracket T \rrbracket \odot \text{ch}(I) \rightarrow \mathbf{0}$, which is definable as a term $x : T, y : \text{chan}[\text{unit}] \vdash M' : \mathbf{0}$. Hence $\lceil ms* \rceil$ is definable as $\nu y. (M' | \text{recv } y)$.

Theorem 1. $\llbracket M \rrbracket \subseteq \llbracket N \rrbracket$ if and only if $M \lesssim N$.

Proof. From right-to left (inequational soundness) this follows from soundness and adequacy. We prove the converse for closed values U, V , which implies the general case. So suppose $\llbracket U \rrbracket \not\subseteq \llbracket V \rrbracket$. Then there exists a sequence $qs \in I \rightarrow \llbracket T \rrbracket$ such that $qs \in \llbracket U \rrbracket$ and $qs \notin \llbracket V \rrbracket$. By Corollary 1, the strategy $\lceil s^\perp * \rceil$ on $\llbracket T \rrbracket \rightarrow \text{unit}$ (where $*$ is the unique move in I) is definable as a term $x : T \vdash M : \text{unit}$. Then $\llbracket (\lambda x. M) U \rrbracket = \{*\}$ and hence by adequacy, $(\lambda x. M) U \Downarrow$. But $\llbracket (\lambda x. M) V \rrbracket = \perp$, since for all $t* \in \lceil s^\perp * \rceil$ there exists $r \sim s^\perp$ such that $t \preceq r$ and hence $s \sim r^\perp \preceq t^\perp$ and so by assumption $qt^\perp \notin \llbracket V \rrbracket$. Hence $(\lambda x. M) V \not\Downarrow$, and $U \not\lesssim V$ as required.

References

1. S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, and I. Stark. Nominal games and full abstraction for the nu-calculus. In *Proceeds of LICS '04*. IEEE Press, 2004.
2. S. Abramsky, R. Jagadeesan and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163:409–470, 2000.
3. M. Boreale, R. de Nicola, and R. Pugliese. Trace and testing equivalence on asynchronous processes. *Information and Computation*, 172(2):139–164, 2002.
4. W. Ferreira, M. Hennessy, and A. S. A. Jeffrey. A theory of weak bisimulation for core CML. *J. Functional Programming*, 8(5):447–491, 1998.
5. D. Ghica and G. McCusker. The regular language semantics of second-order Idealised Algol. *Theoretical Computer Science*, 309:469 – 502, 2003.
6. D. Ghica, A. S. Murawaki, and C.-H. L. Ong. Syntactic control of concurrency. In *Proceedings of ICALP '04*, number 3142 in LNCS. Springer, 2004.
7. D. Ghica and A. Murawski. Angelic semantics of fine-grained concurrency. In *Proceedings of FOSSACS '04*, number 2987 in LNCS, pages 211–225. Springer, 2004.
8. R. Harmer 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*. IEEE Computer Society Press, 1998.
9. K. Honda and N. Yoshida. Game theoretic analysis of call-by-value computation. In *Proceedings of ICALP '97*, volume 1256 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.
10. D. Hughes. Games and definability for System F. In *Proceedings of the twelfth International symposium on Logic in Computer Science, LICS '97*. IEEE Computer Society Press, 1997.

11. J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163:285–408, 2000.
12. A. S. A. Jeffrey and J. Rathke. Contextual equivalence for higher-order pi-calculus revisited. In *Logical Methods in Computer Science* 1(1:4), 2002.
13. A. S. A. Jeffrey and J. Rathke. A fully abstract may-testing semantics for concurrent objects. In *Proceedings of LICS '02*, pages 101–112, 2002.
14. J. Laird. A game semantics of ICSP. In *Proceedings of MFPS XVII*, number 45 in Electronic notes in Theoretical Computer Science. Elsevier, 2001.
15. J. Laird. A game semantics of local names and good variables. In *Proceedings of FOSSACS '04*, number 2987 in LNCS, pages 289–303. Springer, 2004.
16. J. Laird. A game semantics of names and pointers. To appear in *Annals of Pure and Applied Logic*, available from <http://www.cogs.susx.ac.uk/users/jiml>, 2005.
17. J. Laird. A game semantics of the asynchronous pi-calculus. In *Proceedings of CONCUR '05*, number 3653 in LNCS, pages 51–65. Springer, 2005.
18. O. Laurent. Polarized games. In *Proceedings of the Seventeenth International Symposium on Logic In Computer Science, LICS '02*, 2002.
19. P. Malacaria and C. Hankin. Generalised flowcharts and games. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, 1998.
20. G. McCusker. *Games and full abstraction for a functional metalanguage with recursive types*. PhD thesis, Imperial College London, 1996. Published by Cambridge University Press.
21. H. Nickau. Hereditarily sequential functionals. In *Proceedings of the Symposium on Logical Foundations of Computer Science: Logic at St. Petersburg*, LNCS. Springer-Verlag, 1994.
22. J. Power and H. Thielecke. Environments in Freyd categories and κ -categories. In *Proceedings of ICALP '99*, number 1644 in LNCS. Springer, 1999.
23. J. Power and E. Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 1997.
24. J. Reppy. CML: A higher-order concurrent language. In *Proceedings of PLDI '91*, pages 293–305. ACM Press, 1991.
25. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1993.