A Game Semantics of the Asynchronous π-calculus

J. Laird[*]
e-mail:jiml@sussex.ac.uk

Dept. of Informatics, University of Sussex, UK.

Abstract. This paper studies the denotational semantics of the typed asynchronous π-calculus. We describe a simple game semantics of this language, placing it within a rich hierarchy of games models for programming languages,
A key element of our account is the identification of suitable categorical structures for describing the interpretation of types and terms at an abstract level. It is based on the notion of *closed Freyd category*, establishing a connection between our semantics, and that of the λ-calculus. This structure is also used to define a *trace operator*, with which name binding is interpreted. We then show that our categorical characterization is sufficient to prove a weak soundness result.
Another theme of the paper is the correspondence between justified sequences, on which our model is based, and traces in a labelled transition system in which only bound names are passed. We show that the denotations of processes are equivalent, via this correspondence, to their sets of traces. These results are used to show that the games model is fully abstract with respect to may-equivalence.

1 Introduction

The π-calculus [23] is an elegant and powerful formalism offering a flexible description of name mobility; it can be used to give detailed descriptions of concurrent systems, whilst its conceptual and formal simplicity suggest a route to capturing an underlying "logical structure" of information flow. By investigating the semantics of π-calculus at both abstract and more concrete levels, we aim to develop a model of concurrent, mobile behaviour with both of these features.

In this paper, we describe approaches via category theory and denotational (games) semantics. These complement each other well; the former yields an abstract account of the structure of the π-calculus, whilst the latter gives more concrete representations of processes, closely linked to games models of higher-order programmming and logical systems, and also to other process models such as labelled transition systems. Moreover, their rôles in this paper are interlinked — we define the games model in terms of its categorical structure, whilst the

[*] Supported by EU FET-GC 'MyThS: Models and Types for Security in Mobile Distributed Systems' IST-2001-32617

existence of a concrete instance demonstrates the consistency and relevance of the notion of categorical model.

Game semantics already contains a strong element of concurrency: programs (or proofs) are interpreted as strategies, which are basically processes of a particular form, described via their traces. Moreover, π-calculus terms have already been proposed as an elegant formalism for describing strategies in games models of functional languages such as PCF [11, 3]. One of the objectives of this paper is to clarify and generalize this relationship between games and the π-calculus (by showing that justified sequences of moves in the former correspond to traces in a labelled transition system for the latter). This should enable the use of the π-calculus to describe and reason about games models to be extended in a methodical way — to imperative features such as state, for example.

On the other hand, given the connections between justifed sequences and traces, what is the advantage of the former to model the π-calculus itself? If we simply regard games as an abstract and mathematically precise representation of trace semantics, this is already a useful development; for example, abstract interpretation [21] and model-checking techniques [6, 7] based on HO game semantics are available. Moreover, since our games model is constructed in a purely compositional way, it can be generalized to more "truly concurrent" representations of interaction — we sketch such a pomset model in Section 3.1.

A further important characteristic of game semantics is that it imposes a higher-order, typed structure on the more chaotic world of processes, allowing us to identify a deeper "logical structure" within it. It is this structure which we aim to describe using categorical notions. In particular, our account is based on *closed Freyd categories* [24] which are categorical models of the computational λ-calculus, making a semantic connection between the π-calculus and higher-order functional computation (which is implicit in existing translations of the latter into the former). We also show that adding a simple distributivity condition yields a natural definition of a *trace operator*, which we use to interpret new-name binding, conforming to the intuition that this corresponds to a "plugging in" of input and output. More generally, our account is part of an investigation of the logical structure of higher-order imperative/concurrent computation [17].

1.1 Related Work

Hennessey [9], Stark [26] and Fiore, Moggi and Sangiorgi [20] have described domain-theoretic models of the π-calculus, which are fully abstract with respect to various notions of process equivalence. These works differ from the semantics described here in aspects of the language and equivalence studied (synchronous versus asynchronous, untyped versus typed, bisimulation versus may-testing). One may also contrast the nature of the description of processes obtained: the domain theoretic models represent name-passing quite directly, whereas the games model breaks it down into a smaller atoms. (Giving an equally natural characterization of agent mobility.)

A closer parallel is with the data-flow semantics of the π-calculus given by Jagadeesan and Jagadeesan [12], in which dynamic binding is described using notions from the Geometry of Interaction, analogous to our use of the trace operator. Viewed in the light of the correspondence between game semantics and the labelled transition system described in Section 4, our work is also related to Jeffrey and Rathke's may-testing semantics of concurrent objects [14]. The characterization of may-testing equivalence for the asynchronous π-calculus obtained via interpretatation in the fully abstract model is essentially as described (for a somewhat different LTS) by Boreale, de Nicola and Pugliese in [19].

Connections between Hyland-Ong games and the π-calculus were initially investigated by Hyland and Ong themselves, who described a translation of PCF into the π-calculus derived from the representation of innocent strategies as π-calculus terms. This work was developed by Honda, Berger and Yoshida [3, 4], who developed a typing system for the π-calculus identifying sequential processes, and showed that the translation of PCF into this fragment is fully abstract. This research has many parallels with the work described here.

2 A Simply-Typed Asynchronous π-calculus

We recall the polyadic asynchronous π-calculus [10, 5], of which the key operations are the asynchronous output $\overline{x}\langle \boldsymbol{y}\rangle$ of the tuple of names $\boldsymbol{y}$ on channel x, and bound input $x(\boldsymbol{y}).P$ — reception of a tuple of names on channel x which are then substituted for $\boldsymbol{y}$ in P. Our denotational semantics will be made clearer and simpler by a small departure from the original syntax: instead of a single name having distinct input and output capabilities, names will come in complementary pairs $(x, \overline{x})$ of an input name and the corresponding output name $\overline{x}$. New name binding $\nu x.P$ binds both x and $\overline{x}$, but abstraction binds input and output names separately.[1] By convention, we write tuples of names as pairs $(\boldsymbol{y}, \overline{\boldsymbol{z}})$ of tuples of input names $\boldsymbol{x}$ and output names $\overline{\boldsymbol{y}}$. Clearly, we can represent a tuple of names with both input and output capabilities as the pair $(\boldsymbol{x}, \overline{\boldsymbol{x}})$. However, there is no way to guarantee to the receiver of such a tuple that input and output capabilities refer to the same channels. (The difficulties inherent in doing so, in the denotational semantics, are similar to the "bad variable" problem for imperative functional languages.) For related reasons we have neither matching nor mismatching constructs (another point of difference with previous denotational models). We adopt the convention of restricting replication to input-processes, from which we may derive replication of general processes. So the terms of our calculus are given by the grammar:

$$P, Q ::= \mathbf{0} \mid \overline{x}\langle \boldsymbol{y}, \overline{\boldsymbol{z}}\rangle \mid x(\boldsymbol{y}, \overline{\boldsymbol{z}}).P \mid !x(\boldsymbol{y}, \overline{\boldsymbol{z}}).P \mid \nu x.P \mid P|Q$$

Our semantics will be given for a simply-typed version of the π-calculus. Channel types take the form $(\boldsymbol{S}, \boldsymbol{T})$, being a pair of sequences of types (possibly empty) representing the input and output capabilities of the channel (we write

[1] So, in particular, any α-conversion which replaces an input name x with y must replace $\overline{x}$ with $\overline{y}$ (and vice-versa) if x is bound by ν but need not if x is free or bound in an abstraction.

() for the empty type $(\_,\_)$). Typing judgements take the form $\mathit{\Gamma} \vdash P; \mathit{\Sigma}$, where $\mathit{\Gamma}$ is a sequence of typed input names, and $\mathit{\Sigma}$ is a sequence of typed output names. Derivation rules are given in Table 1; we include explicit structural rules as this simplifies the description of the categorical semantics.

$$\begin{array}{cc}
\dfrac{\mathit{\Gamma},x{:}T,y{:}T \vdash P;\mathit{\Sigma}}{\mathit{\Gamma},z{:}T \vdash P\{z/x,z/y\};\mathit{\Sigma}} & \dfrac{\mathit{\Gamma} \vdash P;\mathit{\Sigma},\overline{x}{:}T,\overline{y}{:}T}{\mathit{\Gamma} \vdash P\{\overline{z}/\overline{x},\overline{z}/\overline{y}\};\mathit{\Sigma},\overline{z}{:}T} \\[2ex]
\dfrac{\mathit{\Gamma},x{:}S,y{:}T,\mathit{\Gamma}' \vdash P;\mathit{\Sigma}}{\mathit{\Gamma},y{:}T,x{:}s,\mathit{\Gamma}' \vdash P;\mathit{\Sigma}} & \dfrac{\mathit{\Gamma} \vdash P;\mathit{\Sigma},\overline{x}{:}S,\overline{y}{:}T,\mathit{\Sigma}'}{\mathit{\Gamma} \vdash P;\mathit{\Sigma},\overline{y}{:}T,\overline{x}{:}S,\mathit{\Sigma}'} \\[2ex]
\dfrac{}{\mathit{\Gamma} \vdash \mathbf{0};\mathit{\Sigma}} & \dfrac{\mathit{\Gamma} \vdash P;\mathit{\Sigma} \qquad \mathit{\Gamma}' \vdash Q;\mathit{\Sigma}'}{\mathit{\Gamma},\mathit{\Gamma}' \vdash P|Q;\mathit{\Sigma},\mathit{\Sigma}'} \\[2ex]
\dfrac{}{\mathit{\Gamma},\boldsymbol{y}{:}\boldsymbol{S} \vdash \overline{x}\langle \boldsymbol{y},\overline{\boldsymbol{z}}\rangle;\mathit{\Sigma},\overline{x}{:}(\boldsymbol{S},\boldsymbol{T}),\overline{\boldsymbol{z}}{:}\boldsymbol{T}} & \dfrac{\mathit{\Gamma},\boldsymbol{y}{:}\boldsymbol{S} \vdash P;\mathit{\Sigma},\overline{\boldsymbol{z}}{:}\boldsymbol{T}}{\mathit{\Gamma},x{:}(\boldsymbol{S},\boldsymbol{T}) \vdash x(\boldsymbol{y},\overline{\boldsymbol{z}}).P;\mathit{\Sigma}} \\[2ex]
\dfrac{\mathit{\Gamma},\boldsymbol{y}{:}\boldsymbol{S} \vdash P;\mathit{\Sigma},\overline{\boldsymbol{z}}{:}\boldsymbol{T}}{\mathit{\Gamma},x{:}(\boldsymbol{S},\boldsymbol{T}) \vdash !x(\boldsymbol{y},\overline{\boldsymbol{z}}).P;\mathit{\Sigma}} & \dfrac{\mathit{\Gamma},x{:}T \vdash P;\mathit{\Sigma},\overline{x}{:}T}{\mathit{\Gamma} \vdash \nu x.P;\mathit{\Sigma}}
\end{array}$$

Table 1. Typing Judgements for processes.

We adopt a reduction semantics based on the standard rules for the π-calculus, except that we require that communication only takes place over *bound* channels. (Note that this is implied by the fact that α-equivalence allows us to replace free input and output names separately: it is at the binding stage that output is "plugged in" to input.) Thus the reduction rule for communication is as follows:

$$\nu\boldsymbol{a}.\nu x.(x(\boldsymbol{y},\overline{\boldsymbol{z}}).P\rangle|\overline{x}\langle \boldsymbol{b},\overline{\boldsymbol{c}}\rangle|Q) \longrightarrow \nu\boldsymbol{a}.\nu x.(P\{\boldsymbol{b}/\boldsymbol{y},\overline{\boldsymbol{c}}/\overline{\boldsymbol{z}}\}|Q)$$

We define the reduction relation $\twoheadrightarrow$ to be the reflexive, transitive closure of (the union of) one-step reduction and structural equivalence, where the latter is defined (standardly) as the smallest congruence containing α-equivalence together with the following rules:

$$\begin{array}{ccc}
P|Q \equiv Q|P & \mathbf{0}|P \equiv P & (P|Q)|R \equiv P|(Q|R) \\
\nu x.\nu y.P \equiv \nu y.\nu x.P & (\nu x.P)|Q \equiv \nu x.(P|Q)\ (x \notin FN(Q)) & P \equiv P|!P
\end{array}$$

We will show that our semantics is fully abstract with respect to *may-testing equivalence*. Although it is rather coarse, may-equivalence is useful in describing safety properties, and in the study of *deterministic* programming languages via translation.

We test processes in the π-calculus by observing whether they may produce output on a distinguished channel. For a specified (output) name $\overline{x} : ()$, we write $P \downarrow$ if P is structurally equivalent to a process of the form $\overline{x}\langle\rangle|Q$, and $P \Downarrow$ if $P \twoheadrightarrow P'$ such that $P' \downarrow$.

Definition 1. *Assuming that our testing channel x does not occur in P or Q, we define $P \lesssim Q$ if for all contexts $C[\cdot]$, $C[P] \Downarrow$ implies $C[Q] \Downarrow$, and $P \simeq Q$ if $P \lesssim Q$ and $Q \lesssim P$.*

3 Game Semantics

An *arena* [11] is a forest in which the nodes or *moves* are labelled as belonging to either Player or Opponent. Thus an arena A is specified as a triple $(M_A, \lambda_A, \vdash_A)$ consisting of a set of moves, a labelling function $\lambda_A : M_A \rightarrow \{P, O\}$ and a set of directed edges $\vdash_A \subseteq M_A \times M_A$ or *enabling relation*[2]. The root nodes of A are called *initial* moves: an arena in which all initial moves are O-moves is said to be *negative*. The *dual* $A^\perp$ of the arena A is obtained by swapping Player and Opponent moves: $A^\perp = (M_A, \lambda_A^\perp, \vdash_A)$, where $\lambda_A^\perp(m) = P$ if $\lambda_A(m) = O$ and vice-versa.

A *justified sequence* s over an arena A is a sequence of moves of A, together with a pointer from each non-initial move in s to some preceding move which enables it. A justified sequence s may be represented as a sequence of moves s together with a (partial) justification function $j_s : \mathbb{N} \rightarrow \mathbb{N}$ such that $j_s(k) = i$ if the kth move in s is justified by the ith move. We represent processes as sets of justified sequences or *strategies*. We stipulate that strategies are closed under a preorder $\preceq$, which allows us to give a sequential representation of parallel processes by accounting for the fact that moves of the same polarity (input or output actions) are independent events; their ordering is not (directly) observable, whilst if a process responds to a sequence of actions by the environment (O-moves), then it must make at least the same response to any sequence with more or earlier O-moves. The preorder is based on that introduced in [16], but similar relations are well-established in concurrency theory, in particular we note their use in a LTS characterisation of may-equivalence for the π-calculus [19].

Definition 2. *Let $\preceq$ be the least preorder on justified sequences such that:*

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

In other words, $s \preceq t$ if s can be obtained from t by removing P-moves or migrating them forwards, and adding O-moves or migrating them backwards. The label-inversion operation $(\_)^\perp$ extends pointwise to justified sequences and is antitone with respect to $\preceq$ — $s \preceq t$ if and only $t^\perp \preceq s^\perp$.

Definition 3. *A strategy $\sigma : A$ is a non-empty subset of J_A which is prefix-closed — i.e. $s \sqsubseteq t \in \sigma$ implies $s \in \sigma$ — and $\preceq$-closed — i.e. $s \preceq t$ and $t \in \sigma$ implies $s \in \sigma$.*

Given any set $S \subseteq J_A$, we may form a strategy $\widehat{S}$ by taking the $\preceq$-closure of the prefix-closure of S: $\widehat{S} = \{t \in J_A \mid \exists s \in S, r \in J_A.t \in S.t \preceq r \sqsubseteq s\}$.

[2] Unlike games models of functional languages, we do not require that non-initial O moves are enabled by P-moves and vice-versa.

We now define a category of processes $\mathcal{P}$ in which the objects are negative arenas, and morphisms from A to B are strategies on the "function-space" $A^\perp \odot B$, where $\odot$ is the disjoint union of forests:

$$A \odot B = \langle M_A + M_B, [\lambda_A, \lambda_B], [\vdash_A, \vdash_B] \rangle$$

Composition of $\sigma : A \to B$ and $\tau : B \to C$ is by "parallel composition plus hiding" [1]:

$$\sigma; \tau = \{s \in J_{A^\perp \odot C} \mid \exists t \in J_{A \odot B \odot C}.t{\restriction}A^\perp, B \in \sigma \wedge t{\restriction}B^\perp, C \in \tau \wedge t{\restriction}A^\perp, C = s\}$$

(where $t \restriction A^\perp, B$ means t restricted to moves from A and B, with the former relabelled by swapping Player and Opponent labels). The identity strategy on A is determined by its set of $\preceq$-maximal sequences, which are the sequences which "copycat" between the two components:

$$\mathtt{id}_A = \{s \in J_{A^\perp \odot A} \mid \forall t \widehat{\sqsubseteq^{even}} .t{\restriction}A^\perp = (t{\restriction}A)^\perp\}$$

We show that $\mathcal{P}$ is a well-defined category following proofs for similar categories of games [2, 11, 16].

We observe that $\odot$ acts as a symmetric monoidal product[3] on $\mathcal{P}$ with the empty arena I as its identity element, and an action on functions taking $\sigma : A \to C$ and $\tau : B \to D$ to:

$$\sigma \odot \tau = \{s \in A^\perp \odot B^\perp \odot C \odot D \mid s{\restriction}A^\perp \odot C \in \sigma \wedge s{\restriction}B^\perp \odot D \in \tau\}$$

We also note that $\mathcal{P}$ is (pointed) cpo-enriched with the inclusion order on strategies, the least element of each hom-set being the $\preceq$-closure of the set containing only the empty sequence.

We will interpret each process $x_1 : S_1, \ldots, x_m : S_m \vdash P; \overline{y_1} : T_1, \ldots, \overline{y_n} : T_n$ as a morphism from $[\![T_1]\!] \odot \ldots \odot [\![T_n]\!]$ to $[\![S_1]\!] \odot \ldots \odot [\![S_m]\!]$ in $\mathcal{P}$. This may seem counterintuitive: why not interpret P as a morphism from inputs to outputs in the *dual* of $\mathcal{P}$? The reason is that we will interpret the type-structure of the π-calculus using structure defined on $\mathcal{P}$, rather than on its dual. We will show that we can define a *closed Freyd Category* [24] based on $(\mathcal{P}, I, \odot)$. Closed Freyd categories are models of Moggi's computational λ-calculus (in a canonical sense); thus we have the basis for a categorical analysis of the relationship between higher-order functional behaviour and name mobility. A Freyd category is determined by a symmetric premonoidal category of "computations" (in this case the SMC of processes $\mathcal{P}$), a Cartesian category of "values", and an identity-on-objects, symmetric (pre)monoidal functor from the latter to the former. The closure property operates via this functor.

We define a category of values or *abstractions* using the notion of *well-opened* strategy, adapted from [22].

Definition 4. *A legal sequence on a justified arena A is* well-opened *if it is empty, or contains precisely one initial O-move, which is the first move.*

[3] For the sake of simplicity, we shall not henceforth mention associativity and unit isomorphisms, as if in a strict monoidal category.

In other words, the set W_A of well-opened sequences of A consists of sequences of the form as, where a is an O-move and s contains no initial O-moves. A well-opened strategy σ is a non-empty and $\sqsubseteq$ and $\preceq$-closed subset of W_A.

We may think of a well-opened strategy $\sigma : A \to B$ as a process which receives a single input at B, and then produces multiple outputs at A. Thus to define composition of σ with $\tau : B \to C$, we form the "parallel composition plus hiding" of τ with the replication ad libitum of σ.

Definition 5. *Let $s|t$ denote the set of interleavings of the justified sequences s and t. Given a set X of (justified) sequences, we may define the set $!X$ consisting of interleavings of elements of X — i.e. $s \in !X$ if there exists $t_1, \ldots, t_n \in \sigma$ such that $s \in t_1| \ldots |t_n$. Note that if σ is a strategy, then $!\sigma$ is a strategy, and that $!$ is idempotent.*

Composition of σ and τ is now defined:

$$\sigma;\tau = \{s \in W_{A^\perp \odot C} \mid \exists t \in W_{A \odot B \odot C}.t{\restriction}A^\perp, B \in !\sigma \wedge t{\restriction}B^\perp, C \in \tau \wedge t{\restriction}A^\perp, C = s\}$$

Thus we may form a category of abstractions, $\mathcal{A}$ in which the objects are negative arenas, and the morphisms from A to B are well-opened strategies on $A^\perp \odot B$. The well-opened identity on A is the well-opened subset of $\mathtt{id}^{\mathcal{P}}_A$ — i.e. $\mathtt{id}^{\mathcal{A}}_A : A \to A = \mathtt{id}^{\mathcal{P}}_A \cap W_{A^\perp \odot A}$.

Lemma 1. *$\mathcal{A}$ is a well-defined category, with finite products given by $\odot$.*

Proof. We prove that $!, \mathtt{id}^{\mathcal{A}}$ have the following properties:

– If $\sigma : A \to B$ is well-opened, then $!\sigma; \mathtt{id}^{\mathcal{A}}_B = \sigma$,
– $!\mathtt{id}^{\mathcal{A}}_A = \mathtt{id}^{\mathcal{P}}_A$,
– If σ, τ are well-opened, then $!\sigma;\tau$ is well-opened, and $!(!\sigma;\tau) = !\sigma;!\tau$.

We also note that every well-opened strategy $\sigma : A \to B$ determines a unique morphism from A to B in $\mathcal{P}$ as its $\preceq$-closure. In particular, we shall write the $\preceq$-closure of the well-opened identity as $\mathtt{der}_A : A \to A$ and use the fact that $\mathtt{der}_A \subseteq \mathtt{id}_A$.

The proof of Lemma 1 also establishes that $!$ acts as a (identity-on-objects) functor from $\mathcal{A}$ to $\mathcal{P}$ such that $!(\sigma \odot \tau) = !\sigma \odot !\tau$ (and $!I = I$) and so $!$. The Cartesian structure of $\mathcal{A}$ gives projection maps $\pi_l : A_1 \odot A_2 \to A_1$ and $\pi_r : A_1 \odot A_2 \to A_2$, and a diagonal map $\Delta_A : A \to A \odot A$. We use the fact that each hom-set of $\mathcal{A}$ is a join semilattice with respect to the inclusion order to define $\nabla_A : A \odot A \to A = \pi_l \cup \pi_r$. This has the defining properties $!\Delta_A; !\nabla_A = !((\Delta;\pi_l) \cup (\Delta;\pi_r)) = !\mathtt{id}^{\mathcal{A}}_A = \mathtt{id}^{\mathcal{P}}_A$ and $\mathtt{id}^{\mathcal{P}}_{A \odot A} \subseteq !\nabla_A; !\Delta_A$.

We have established that the SMC $(\mathcal{P}, I, \odot)$, the Cartesian category $\mathcal{A}$, and the functor $!$ form a Freyd Category in which the product is monoidal rather than premonoidal. Moreoever, it is a *closed* Freyd Category — i.e. the functor $A \odot !\_ : \mathcal{A} \to \mathcal{P}$ has a right-adjoint $A \rightharpoonup \_$. We define $A \rightharpoonup B$ to be $\uparrow (A^\perp \odot B)$, where $\uparrow \_$ is the *lifting* operation which converts a forest to a tree by adding an edge into each of its roots from a single, new (O-labelled) root.

$$\uparrow A = (M_A + \{*\}, [\lambda_A, \{\langle *, O \rangle\}], [\vdash_A, \varnothing] \cup \{\langle *, m \rangle \mid \ \vdash_A m\})$$

Proposition 1. *For any arena B, $B \multimap \_$ is right-adjoint to $!\_ \odot B : \mathcal{A} \to \mathcal{P}$.*

Proof. There is a simple bijection taking justified sequences in $(A \odot B)^{\perp} \odot C$ to well-opened sequences in $A^{\perp} \odot \uparrow (B^{\perp} \odot C)$: prefix a single initial move in $\uparrow (B^{\perp}, C)$, with justification pointers from the initial moves in $(B^{\perp}, C)$. This acts on strategies to yield a natural isomorphism $\Lambda : \mathcal{P}(!A \odot B, C) \to \mathcal{A}(!A, B \multimap C)$

We use the following additional property of the adjunction: for any A, B, $(\mathsf{der}_{A \multimap B} \odot \mathtt{id}_A); \mathsf{app}_{A,B} = \mathsf{app}_{A,B}$, where $\mathsf{app}_{A,B} : (A \multimap B) \odot A \to B$ is the co-unit. (Since $\mathsf{app}_{A,B}$ only makes the initial move in $A \multimap B$ once.)

We will interpret output as app, and input as the operation Λ, composed with a natural transformation expressing a distributivity property for the exponential over the tensor.

Definition 6. *A Freyd category may be said to be* distributive-closed *if it is closed, and there is a natural transformation: $\varrho_{A,B,C} : !(A \multimap (B \odot C)) \to B \odot !(A \multimap C)$ satisfying the following properties:*

– $(\varrho_{A,B,C} \otimes \mathtt{id}_A); (\mathtt{id}_B \odot \mathsf{app}_{A,C}) = \mathsf{app}_{A,B \odot C}$,
– $\varrho_{A,B,C \otimes D}; (\mathtt{id}_B \otimes \varrho_{A,C,D}) = \varrho_{A,B \otimes C,D}$

In $\mathcal{P}$, ϱ is induced by the map from justified sequences on $A \multimap (B \odot C)$ to justified sequences on $(A \multimap B) \odot C$ which relabels initial moves and removes the justification pointers from the initial moves in C.

In any distributive-closed Freyd category, including $\mathcal{P}$, we may define a *trace operator* $\mathsf{Tr}^{B}_{A,C} : \mathcal{P}(A \odot B, C \odot B) \to \mathcal{P}(B, C)$ making $\mathcal{P}$ a traced monoidal category [15]. This provides a natural notion of “feedback” connecting input to output, with which we interpret new-name binding. We define the trace of $f : A \odot B \to C \odot B$:

$$\mathsf{Tr}^{B}_{A,C}(f) = !\Lambda(f; \theta_{B,C}); \varrho_{B,B,C}; \theta_{B,B \multimap C}; \mathsf{app}_{B,C}$$

Using naturality of the constituent operations, together with the axioms for ϱ, we prove the following lemma.

Lemma 2. Tr *is a trace operator for $\mathcal{P}$ in the sense of [15].*

We may now give the interpretation of the π-calculus in $\mathcal{P}$. The type $(\boldsymbol{S}, \boldsymbol{T})$ is interpreted as the negative arena $[\![\boldsymbol{T}]\!] \multimap [\![\boldsymbol{S}]\!]$ — i.e. $\uparrow ([\![T_1]\!]^{\perp} \odot \ldots [\![T_n]\!]^{\perp} \odot [\![S_1]\!] \odot \ldots \odot [\![S_m]\!])$. Terms-in-context $\Gamma \vdash P; \Sigma$ are interpreted as morphisms $[\![\Gamma \vdash P; \Sigma]\!] : [\![\Sigma]\!] \to [\![\Gamma]\!]$ using the structure of a symmetric monoidal, distributive-closed Freyd category, according to the rules in Table 2. (More precisely, Table 2 gives rules for interpreting each typing-judgement *derivation* as a morphism; we show that every derivation of the same term receives the same denotation.)

We will now use the categorical structure of our model to establish a weak soundness result with respect to the reduction semantics: if M may reduce to N, then $[\![N]\!]$ is included in $[\![M]\!]$. We first show soundness with respect to structural equivalence.

Lemma 3. *If $M \equiv N$, then $[\![M]\!] = [\![N]\!]$.*

$$\begin{gathered}
\llbracket \Gamma, y : T, x : S, \Gamma' \vdash P; \Sigma \rrbracket = \llbracket \Gamma, x : S, y : T, \Gamma' \vdash P; \Sigma \rrbracket; (\mathtt{id}_{\llbracket \Gamma \rrbracket} \odot \theta_{\llbracket T \rrbracket, \llbracket S \rrbracket} \odot \mathtt{id}_{\llbracket \Gamma' \rrbracket}) \\
\llbracket \Gamma \vdash P; \Sigma, \overline{y} : T, \overline{x} : S, \Sigma' \rrbracket = (\mathtt{id}_{\llbracket \Sigma \rrbracket} \odot \theta_{\llbracket S \rrbracket, \llbracket T \rrbracket} \odot \mathtt{id}_{\llbracket \Sigma' \rrbracket}); \llbracket \Gamma \vdash P; \Sigma, \overline{x} : S, \overline{y} : T, \Sigma' \rrbracket \\
\llbracket \Gamma, z : T \vdash P\{z/x, z/y\}; \Sigma \rrbracket = \llbracket \Gamma, x : T, y : T \vdash P; \Sigma \rrbracket; (\mathtt{id}_{\llbracket \Gamma \rrbracket} \odot ! \Delta_{\llbracket T \rrbracket}) \\
\llbracket \Gamma \vdash P\{\overline{z}/\overline{x}, \overline{z}/\overline{y}\}; \Sigma, \overline{z} : T \rrbracket = (\mathtt{id}_{\llbracket \Sigma \rrbracket} \odot ! \nabla_{\llbracket T \rrbracket}); \llbracket \Gamma \vdash P; \Sigma, \overline{x} : S, \overline{y} : T \rrbracket \\
\Gamma \vdash \mathbf{0}; \Sigma = \bot_{\Sigma, \Gamma} \\
\llbracket \Gamma, \Gamma' \vdash P|Q; \Sigma, \Sigma' \rrbracket = \llbracket \Gamma \vdash P, \Sigma \rrbracket \odot \llbracket \Gamma' \vdash Q; \Sigma' \rrbracket \\
\llbracket \Gamma, x : (\boldsymbol{S}, \boldsymbol{T}) \vdash x(\boldsymbol{y}, \overline{\boldsymbol{z}}).P; \Sigma \rrbracket = ! \Lambda(\llbracket \Gamma, \boldsymbol{y} : \boldsymbol{S} \vdash P; \Sigma, \overline{\boldsymbol{z}} : \boldsymbol{T} \rrbracket); \varrho_{\llbracket \boldsymbol{S} \rrbracket, \llbracket \Gamma \rrbracket, \llbracket \boldsymbol{T} \rrbracket}; (\mathsf{der}_{\llbracket (\boldsymbol{S}, \boldsymbol{T}) \rrbracket} \odot \mathtt{id}_{\llbracket \Gamma \rrbracket}) \\
\llbracket \Gamma, x : (\boldsymbol{S}, \boldsymbol{T}) \vdash ! x(\boldsymbol{y}, \overline{\boldsymbol{z}}).P; \Sigma \rrbracket = ! \Lambda(\llbracket \Gamma, \boldsymbol{y} : \boldsymbol{S} \vdash P; \Sigma, \overline{\boldsymbol{z}} : \boldsymbol{T} \rrbracket); \varrho_{\llbracket \boldsymbol{S} \rrbracket, \llbracket \Gamma \rrbracket, \llbracket \boldsymbol{T} \rrbracket} \\
\llbracket \Gamma, \boldsymbol{y} : \boldsymbol{S} \vdash \overline{x}\langle \boldsymbol{y}, \overline{\boldsymbol{z}} \rangle; \Sigma, \overline{x} : (\boldsymbol{S}, \boldsymbol{T}), \overline{\boldsymbol{z}} : \boldsymbol{T} \rrbracket = \bot_{\Sigma, \Gamma} \odot \mathsf{app}_{\llbracket \boldsymbol{S} \rrbracket, \llbracket \boldsymbol{T} \rrbracket} \\
\llbracket \Gamma \vdash \nu x.P; \Sigma \rrbracket = \mathsf{Tr}^{\llbracket T \rrbracket}_{\llbracket \Sigma \rrbracket, \llbracket \Gamma \rrbracket}(\llbracket \Gamma, x : T \vdash P; \Sigma, \overline{x} : T \rrbracket)
\end{gathered}$$

Table 2. Interpretation of processes

Proof. The equivalences for parallel composition follow directly from the analogous properties of $\odot$. Those for new-name binding follow from its interpretation as a trace operator — e.g. scope extrusion follows from "tightening"; naturality of $tr^X_{A,B}$ with respect to A, B. For replication, we use the fact that $!f = !\Delta; ((!f; \mathsf{der}) \odot !f); \pi_r \subseteq !\Delta; ((!f; \mathsf{der}) \odot !f); !\nabla \subseteq !\Delta; (!f \odot !f); !\nabla$.

Proposition 2. *If $M \longrightarrow N$, then $\llbracket N \rrbracket \subseteq \llbracket M \rrbracket$.*

Proof. To show soundness of the reduction rule, we first observe that for any process $\Gamma, x : B, y : B \vdash P; \Sigma, \overline{x} : B, \overline{y} : B$, $\llbracket \nu x.\nu y.P \rrbracket \subseteq \llbracket \nu x.P\{x/y, \overline{x}/\overline{y}\} \rrbracket$:
By sliding (naturality of the trace operator in B) we have $\llbracket \nu x.P\{x/y, \overline{x}/\overline{y}\} \rrbracket$
$= \mathsf{Tr}^B_{\llbracket \Sigma, \Gamma \rrbracket}((\mathtt{id}_{\llbracket \Sigma \rrbracket} \odot !\Delta_B); \llbracket P \rrbracket; (\mathtt{id}_{\llbracket \Gamma \rrbracket} \odot !\nabla_B)) = \mathsf{Tr}^{B \odot B}_{\llbracket \Sigma \rrbracket, \llbracket \Gamma \rrbracket}((\mathtt{id}_{\llbracket \Sigma \rrbracket} \odot (!\nabla; !\Delta)); \llbracket P \rrbracket)$.
Since $\mathtt{id}_B \subseteq !\nabla_B; !\Delta_B$, we have $\llbracket \nu x.\nu y.P \rrbracket = \mathsf{Tr}^B_{\llbracket \Sigma \rrbracket, \llbracket \Gamma \rrbracket}(\mathsf{Tr}^B_{\llbracket \Sigma \rrbracket \odot B, \llbracket \Gamma \rrbracket \odot B}(\llbracket P \rrbracket)) = \mathsf{Tr}^{B \odot B}_{\llbracket \Sigma \rrbracket, \llbracket \Gamma \rrbracket}(\llbracket P \rrbracket) \subseteq \llbracket \nu x.P\{x/y, \overline{x}/\overline{y}\} \rrbracket$ as required.

We then show that if c is not free in P, Q, then $\llbracket \nu c.(\overline{c}\langle \boldsymbol{a}, \overline{\boldsymbol{b}} \rangle | c(\boldsymbol{y}, \overline{\boldsymbol{z}}).P \rrbracket = \llbracket P\{\boldsymbol{a}/\boldsymbol{y}, \overline{\boldsymbol{b}}/\overline{\boldsymbol{z}}\} \rrbracket$, since $\llbracket \nu c.(\overline{c}\langle \boldsymbol{a}, \overline{\boldsymbol{b}} \rangle | c(\boldsymbol{y}, \overline{\boldsymbol{z}}).P \rrbracket = \mathsf{Tr}^B_{\llbracket \Sigma \rrbracket, \llbracket \Gamma \rrbracket}(\mathsf{app} \odot (!\Lambda(\llbracket P \rrbracket); \mathrm{dist}; (\mathsf{der} \odot \mathtt{id}))); \theta) = (!\Lambda(\llbracket P \rrbracket) \odot \mathtt{id}); \mathrm{dist}; (\mathtt{id} \odot \mathtt{id})(\mathtt{id}_\Gamma \odot \mathsf{app}) = (!\Lambda(\llbracket \mathrm{P} \rrbracket) \odot \mathtt{id}); \mathsf{app}) = \llbracket \mathrm{P}\{\boldsymbol{a}/\boldsymbol{y}, \overline{\boldsymbol{b}}/\overline{\boldsymbol{z}}\} \rrbracket$ (by the "generalized yanking" property for trace operators).
So $\llbracket \nu x.P\{\boldsymbol{a}/\boldsymbol{y}, \overline{\boldsymbol{b}}/\overline{\boldsymbol{z}}\} | Q \rrbracket = \llbracket \nu x.(\nu c.\overline{c}\langle \boldsymbol{a}, \overline{\boldsymbol{b}} \rangle | c(\boldsymbol{y}, \overline{\boldsymbol{z}}).P) | Q \rrbracket \subseteq \llbracket \nu x.\overline{x}\langle \boldsymbol{a}, \overline{\boldsymbol{b}} \rangle | x(\boldsymbol{y}, \overline{\boldsymbol{z}}).P | Q \rrbracket$ as required.

3.1 More Categorical Models

Any semantics based on a distributive-closed Freyd category satisfying the properties used in the proof will also satisfy weak soundness: we have instances reflecting both finer and weaker notions of process equivalence.

Justified Pomsets By moving to a representation of interaction in terms of pomsets rather than sequences, we may define a finer "true-concurrency-style" version of our games model. Its relationship to the pomset semantics of the synchronous π-calculus in [12] is still under investigation.
A *justified* pomset over an arena A is a finite pomset p for which the labels are elements of M_A, and for each event $e \in p$ with a non-initial label, a pointer

to an event e' such that $e' < e$ and $\mathsf{label}(e') \vdash \mathsf{label}(e)$. A pomset over A is alternating, if whenever e' is a maximal element of the set $\{d \in p \mid d <_p e\}$ then $\lambda_A(\mathsf{label}(e)) = \lambda_A^{\perp}(\mathsf{label}(e'))$.
Note that each justified sequence $s \in J_A$ may be viewed as a (non-alternating), justified *total* pomset: we may say that the justified sequence s is a *sequentialization* of the justified pomset p if there is a order, pointer and label-preserving bijection from p to s. The sequentialization of a set X of pomsets is the set of justified sequences which are the sequentialization of some $p \in \sigma$.
Let O_p and P_p be the restrictions of P to Opponent and player moves, respectively. We define the saturation order on justified, alternatin pomsets thus: $p \preceq q$ if there exist injective functions $f : O_q \to O_p$ and $g : P_p \to P_q$ whicch preserve and reflect order and labelling, and such that if $f(a)$ justifies b, then a justifies $g(b)$, if a justifies $f(b)$ then $g(a)$ justifies b, and if $f(a) < b$ then $a < g(b)$.
Using the constructions $(\_)^{\perp}$, $\odot$ and $\uparrow (\_)$, we may construct a distributive-closed Freyd category of arenas and pomset-strategies (sets of justified, alternating pomsets, closed under $\preceq$), and thus a semantics of the π-calculus. Denotational equivalence in this model is strictly finer than the interleaving semantics: the justified sequence denotation of any term is the sequentialization of its denotation in the justified pomset model.

Unjustified Sequences On the other hand, we may construct models which are coarser (not being *adequate* with respect to may-testing) but do have somewhat simpler structure and might therefore be used as *abstract interpretations*. One example is obtained by simply forgetting the justification-pointers in the games model. Given an arena A, an "unjustified sequence" over A is simply a sequence in M_A^* such that every non-initial move is preceded by at least one move which enables it.We may define strategies and their composition exactly as for the justified model, yielding a symmetric monoidal distributive-closed Freyd category. With limited forms of recursion (iteration rather than general replication) we may describe unjustified strategies as regular grammars [6].
We have further instances of "event-structure" like models in which morphisms represent reachable positions, and compositional is wholly relational. In this case we also lose some information about the sequential ordering of events.

4 Full Abstraction

We will now show that our game semantics is fully abstract with respect to may-equivalence. The difficult part of the proof is to show that it is *adequate*: any process $\_ \vdash P; \overline{x} : ()$ which has a non-empty denotation will produce a corresponding output on x. To show this we will relate our game semantics to a *labelled transition system* for the asynchonous π-calculus. We show that traces in our LTS are in bijective correspondence with justified sequences over the associated arenas, and that this extends to relate traces of terms to their denotations as strategies. Rather than the standard LTS for the π-calculus, we use one in which only bound names may be passed. This corresponds to labelled transition systems for HOπ [25, 13], in which messages are fresh names used as "triggers".

Actions α are either silent (τ) or take the form $x\langle \boldsymbol{k}, \overline{\boldsymbol{l}}\rangle$ (input) or $\overline{x}\langle \overline{\boldsymbol{k}}, \boldsymbol{l}\rangle$ (the complementary output) where $\boldsymbol{k}, \boldsymbol{l}$ are distinct names such that if $x : (\boldsymbol{S}, \boldsymbol{T})$, then $\boldsymbol{k} : \boldsymbol{T}$ and $\boldsymbol{l} : \boldsymbol{S}$. We refer to x (resp. $\overline{x}$) as the *channel* of α, or $\kappa(\alpha)$, and to $\boldsymbol{k}, \overline{\boldsymbol{l}}$ (resp. $\overline{\boldsymbol{k}}, \boldsymbol{l}$) as the *contents* of α or $\epsilon(\alpha)$. We require that the channel of any action performed by P *must* occur free in P, and that its contents, and their complements, *must not* occur free in P — i.e. the rules of Table 3 all have as an implicit side condition that $P \xrightarrow{\alpha} Q$ only if $\kappa(\alpha) \in FN(P)$ and $(\epsilon(\alpha) \cup \epsilon(\overline{\alpha})) \cap FN(P) = \varnothing$. We write $[x \mapsto \overline{y}]$ for the "persistent forwarder" $!x(\boldsymbol{a}, \overline{\boldsymbol{b}}).\overline{y}\langle \boldsymbol{a}, \overline{\boldsymbol{b}}\rangle$.

$$x(\boldsymbol{y}, \overline{\boldsymbol{z}}).P \xrightarrow{x\langle \boldsymbol{k}, \overline{\boldsymbol{l}}\rangle} P\{\boldsymbol{k}/\boldsymbol{y}, \overline{\boldsymbol{l}}/\overline{\boldsymbol{z}}\} \qquad \overline{x}\langle \boldsymbol{y}, \overline{\boldsymbol{z}}\rangle \xrightarrow{\overline{x}\langle \overline{\boldsymbol{k}}, \boldsymbol{l}\rangle} [\boldsymbol{y} \mapsto \overline{\boldsymbol{k}}] | [\boldsymbol{l} \mapsto \overline{\boldsymbol{z}}]$$

$$\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \frac{P \xrightarrow{\alpha} P'}{\nu x.P \xrightarrow{\alpha} \nu x.P'}$$

$$\frac{P \xrightarrow{\alpha} Q \qquad P \equiv P'}{P' \xrightarrow{\alpha} Q} \qquad \frac{P \xrightarrow{x\langle \boldsymbol{k}, \overline{\boldsymbol{l}}\rangle} P' \qquad Q \xrightarrow{\overline{x}\langle \overline{\boldsymbol{k}}, \boldsymbol{l}\rangle} Q'}{\nu x.(P|Q) \xrightarrow{\tau} \nu x.\nu \boldsymbol{k}.\nu \boldsymbol{l}.(P'|Q')}$$

Table 3. LTS for bound name passing

We write $\texttt{trace}(P)$ for the set of traces of the process P, with the τ-actions erased. If P is typable as $\Gamma \vdash P; \Sigma$, then every member of $\texttt{trace}(P)$ is a (τ-free) *well-formed trace* over (Γ, Σ) — a sequence of actions α such that:

- The channel of α either occurs as a free name in P — in which case we shall say that α is *initial* — or in the contents of some unique previous action in the trace, which we may call the *justifier* of α.
- The contents of α and their complements do not appear in Γ, Σ, or previously in the trace.

We note also that the set of well-formed traces is closed under α-equivalence, and that if $M \equiv_\alpha N$, then $\forall t \in \texttt{trace}(M).\exists t' \in N$ such that $t \equiv_\alpha t'$, and vice-versa.

We may now observe that there is a simple correspondence between the traces over (Γ, Σ) and the justified sequences on $[\![\Sigma]\!]^\perp \odot [\![\Gamma]\!]$: we replace each output action with a Player move and each input action with an Opponent move, and retain the justification structure. To determine which move replaces a given action, we note that the justification history of each action determines a unique path through the syntax forest of (Γ, Σ) (which is isomorphic to the forest of moves $[\![\Sigma]\!]^\perp \odot [\![\Gamma]\!]$), and this path determines a unique move in the arena. More precisely, for each action α (with $\kappa(\alpha) : T$) in a trace s over (Γ, Σ) we define a

map $\psi_{s,\alpha}$ from $M_{[\![T]\!]}$ to $M_{[\![\Sigma]\!]^{\perp}\odot[\![\Gamma]\!]}$:
If α is initial then if $\kappa(\alpha) = \overline{x_i} : S_i$, then $\psi_{s,\alpha}(m) = (\mathrm{in_l}(\mathrm{in_i}(m))$, and if $\kappa(\alpha) = y_i : T_i$, then $\psi_{s,\alpha}(m) = \mathrm{in_r}(\mathrm{in_i}(m))$.
If α is justified by β, then if $\kappa(\alpha) = \overline{x_i} : S_i$, then $\psi_{s\alpha}(m) = \psi_{s,\beta}(\mathrm{in_l}(\mathrm{in_i}(m))$, and if $\kappa(\alpha) = y_i : T_i$, then $\psi_{s,\alpha}(m) = \psi_{s,\beta}(\mathrm{in_r}(\mathrm{in_i}(m)))$.
Thus we may define a map ϕ from traces over (Γ, Σ) to justified sequences over $([\![\Sigma]\!]^{\perp} \odot [\![\Gamma]\!])$ by replacing each action α in s with $\psi_{s,\alpha}(*)$, and defining a justification function so that $j_{\phi(s)}(k) = i$ if the kth action in s is the justifier of the ith action.

It is easy to see that ϕ sends α-equivalent traces to the same justified sequence: the following lemma is then straightforward to prove.

Lemma 4. *For any context (Γ, Σ), (α-equivalence classes of) traces over (Γ, Σ), and justified sequences over the arena $[\![\Gamma]\!]^{\perp} \odot [\![\Sigma]\!]$ are in bijective correspondence.*

What is the relationship between $\phi(\texttt{trace}(P))$ and the denotation of P in the games model? They are not equal in general; for example, the nil process has no non-empty traces, but is not represented by the empty strategy. However, by inductive characterization of $\texttt{trace}(P)$ we may prove the following.

Proposition 3. *For any process P, $[\![P]\!] = \phi(\widehat{\texttt{trace}(P)})$.*

We now use our weak soundness result, and the correspondence between traces and justified sequences to prove that the denotational semantics is sound and adequate with respect to may-testing in the the reduction semantics. To complete the proof, we need to show that if a process may perform an input action in the LTS, then it may perform the same action in the reduction semantics. First, we prove the following equivalences by showing that the smallest precongruence containing them is preserved by reduction.

Lemma 5. *For any process $\Gamma \vdash P; \Sigma$, if $k \notin \Gamma$ then $P\{k/x\} \lesssim \nu x.(P|[k \mapsto \overline{x}])$, and if $\overline{k} \notin \Sigma$, then $P\{\overline{k}/\overline{x}\} \lesssim \nu x.(P|[x \mapsto \overline{k}])$.*

Lemma 6. *For any process $\_ \vdash P; \overline{x} : ()$, if $P \xrightarrow{\tau^*} P' \xrightarrow{\overline{x}\langle\rangle} Q$ then $P \Downarrow$.*

Proof. By induction on the number of silent actions in $P \xrightarrow{\tau^*} P'$. If $P = P'$, then we show by induction on the derivation of $P \xrightarrow{x\langle\rangle} Q$ that $P \downarrow$.
Otherwise, $P \xrightarrow{\tau} P'' \xrightarrow{\tau^*} P' \xrightarrow{x\langle\rangle} Q$. We prove by induction on the derivation of $P \xrightarrow{\tau} P''$ that there are terms R_1, R_2 such that $P \equiv \nu\boldsymbol{y}.(\overline{y_i}\langle\boldsymbol{c}, \overline{\boldsymbol{d}}\rangle|y_i(\boldsymbol{a}, \overline{\boldsymbol{b}}).R_1|R_2)$, and $P'' \equiv \nu\boldsymbol{y}.\nu\boldsymbol{k}.\nu\boldsymbol{l}.([\boldsymbol{c} \mapsto \overline{\boldsymbol{k}}]|[\boldsymbol{l} \mapsto \overline{\boldsymbol{d}}]|R_1\{\boldsymbol{k}/\boldsymbol{a}, \overline{\boldsymbol{l}}/\overline{\boldsymbol{b}}\}|R_2)$. Hence P reduces to $\nu\boldsymbol{y}.(R_1\{\boldsymbol{c}/\boldsymbol{a}, \overline{\boldsymbol{d}}/\overline{\boldsymbol{b}}\}|R_2)$, and by Lemma 5, $P'' \lesssim \nu\boldsymbol{y}.(R_1\{\boldsymbol{c}/\boldsymbol{a}, \overline{\boldsymbol{d}}/\overline{\boldsymbol{b}}\}|R_2)$. By induction hypothesis, $P'' \Downarrow$, and hence $P \Downarrow$ as required.

Proposition 4. *For any process $\_ \vdash P; \overline{x} : ()$, $P \Downarrow$ if and only if $[\![P]\!] \neq \bot$.*

Proof. From left-to-right, this follows from weak soundness by induction on derivation length. From right-to left, this follows from Proposition 3 and Lemma 6: if $[\![P]\!] \neq \bot$, then $\texttt{trace}(P) \neq \varnothing$, and hence $P \Downarrow$ as required.

To prove full abstraction, we now show that for any terms with distinct denotations, we may define a distinguishing context. It is sufficient to show that strategies which test for the existence of a given trace are definable, for which we need to prove that for any sequence $s \in [\![\Sigma]\!]^\perp \odot [\![\Gamma]\!]$, the strategy $\widehat{\{s\}}$ is the denotation of a term $\Gamma \vdash P; \Sigma$. In contrast to previous such "definability results" for games models, this is straightforward to prove, since from any trace we may extract a ("minimal") term which generates it (following proofs of similar results for the higher-order π-calculus [25]).

Proposition 5. *For every justified sequence $s \in [\![\Sigma]\!]^\perp \odot [\![\Gamma]\!]$, there exists a process $\Gamma \vdash P; \Sigma$ such that $[\![P]\!] = \widehat{\{s\}}$.*

Proof. By induction on the length of s. Suppose the first move in s is an Opponent move. We suppose without loss of generality that this move is the initial move in the final conjunct of $\Gamma = \Gamma', x : (\boldsymbol{S}, \boldsymbol{T})$. By removing it, and relabelling moves hereditarily justified by it as moves in $[\![\boldsymbol{S}]\!]$ and $[\![\boldsymbol{T}]\!]^\perp$, we may define a justified sequence t on the arena $[\![\Sigma, \boldsymbol{S}]\!] \odot [\![\Gamma, \boldsymbol{T}]\!]^\perp$. By induction hypothesis, the strategy $\widehat{\{t\}}$ is definable as a term $\Gamma, \boldsymbol{y} : \boldsymbol{S} \vdash P; \Sigma, \overline{\boldsymbol{z}} : \boldsymbol{T}$ and hence $\widehat{\{s\}}$ is definable as $x(\boldsymbol{y}, \overline{\boldsymbol{z}}).P$.

If the first move is a Player move (initial in the final component of $\Sigma = \Sigma', \overline{x} : (\boldsymbol{S}, \boldsymbol{T})$), then by removing it and relabelling as above, we obtain a sequence t such that $\widehat{\{t\}}$ is definable as a process $\Gamma, \boldsymbol{y} : \boldsymbol{T} \vdash P; \Sigma, \overline{\boldsymbol{z}} : \boldsymbol{T}$. Then $\widehat{\{s\}} = [\![\nu \boldsymbol{y}.\nu \boldsymbol{z}.(P | \overline{x}\langle \boldsymbol{z}, \overline{\boldsymbol{y}} \rangle)]\!]$.

Theorem 1. *For any processes $\Gamma \vdash P, Q; \Sigma$, $P \lesssim Q$ if and only if $[\![P]\!] \subseteq [\![Q]\!]$.*

Proof. From right-to left (inequational soundness) this follows from soundness and adequacy (Proposition 4). We prove the converse for processes P, Q with a single (input) name y, which implies the general case as $P(\boldsymbol{a}, \overline{\boldsymbol{b}})$ may be recovered from $y(\boldsymbol{a}, \overline{\boldsymbol{b}}).P$. So suppose $[\![y : T \vdash P]\!] \not\subseteq [\![y : T \vdash Q]\!]$. Then there exists $s \in [\![T]\!]$ such that $s \in [\![P]\!]$ and $s \notin [\![Q]\!]$. By Proposition 5, the strategy $\widehat{\{s^\perp *\}}$ on $[\![T]\!]^\perp \odot (\uparrow I)^\perp$ (where $*$ is the unique move in $\uparrow I$) is definable as a process $\vdash R; \overline{y} : T, \overline{x} : ()$. Then $[\![\nu y.(P|R)]\!] = \{*\}$ and hence by adequacy, $\nu y.(P|R) \Downarrow$. But $[\![\nu y.(Q|R)]\!] = \bot$, since for all $t* \in \lceil s^\perp * \rceil$ we have $t \preceq s^\perp$ and hence $s \preceq t^\perp$ and so by assumption $t^\perp \notin [\![Q]\!]$. Hence $\nu y.(Q|R) \not\Downarrow$, and $P \not\lesssim Q$ as required.

5 Conclusions and Further Directions

For the sake of simplicity, we have restricted our semantics to simple types, but it is possible to extend it with recursive types, using the methodology developed by McCusker for solving recursive domain equations in a functional setting [22]. In particular, we may construct a model of the standard, untyped π-calculus, based on the type $\mu X.(X, X)$.

Our semantics represents names implicitly, via information flow. So, for instance, it does not provide a natural way to interpret matching and mismatching constructs, or the association of input and output capabilities to a single name.

We could, however introduce the capacity to represent names explicitly into our model using the techniques described in [18] based on a category of games acted on by the group of natural number permutations.

The representation of processes via their traces, is probably suitable only for characterizing testing equivalences. A natural extension of the current research would be to construct a model of must-testing, by recording traces resulting in divergence, as in [8, 16] and deadlock. However, there are many technical complications. We have also sketched "true concurrency" and "abstract interpretation" examples of our categorical semantics which require further investigation. Alternatively, we may use π-calculus terms themselves to represent strategies, in which case we may ask which notions of equivalence yield a distributive-closed Freyd category. We may also attempt to develop domain-theoretic instances of our categorical constructions and relate them to other such models of the π-calculus.

References

1. S. Abramsky, R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59:543–574, 1994.
2. S. Abramsky, R. Jagadeesan and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163:409–470, 2000.
3. M. Berger, K. Honda, and N. Yoshida. Sequentiality and the π-calculus. In *Proceedings of TLCA 2001*, volume 2044 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.
4. M. Berger, K. Honda, and N. Yoshida. Strong normalization in the π-calculus. In *Proceedings of LICS 2001*. IEEE Press, 2001.
5. G. Boudol. Asynchrony in the pi-calculus. Technical Report 1702, INRIA, 1992.
6. D. Ghica and G. McCusker. The regular language semantics of second-order Idealised Algol. *Theoretical Computer Science (To appear)*, 2003.
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 nondeterminism. In *Proceedings of the Fourteenth Annual Symposium on Logic in Computer Science, LICS '99*. IEEE Computer Society Press, 1998.
9. M. Hennessy. A fully abstract denotational semantics for the π-calculus. Technical Report 041996, University of Sussex (COGS), 2996.
10. K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proceedings of ECOOP '91*, number 512 in LNCS, pages 133–147, 1991.
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. L. J. Jagadeesan and R. Jagadeesan. Causality and true concurrency: A data-flow analysis of the pi-calculus. In *Proceedings of AMAST '95*, 1995.
13. A. S. A. Jeffrey and J. Rathke. Contextual equivalence for higher-order pi-calculus revisited. Technical Report 0402, University of Sussex (COGS), 2002.
14. 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.
15. A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Math. Proc. Camb. Phil. Soc.*, 119:447 – 468, 1996.

16. J. Laird. A game semantics of ICSP. In *Proceedings of MFPS XVII*, number 45 in Electronic notes in Theoretical Computer Science. Elsevier, 2001.
17. J. Laird. A categorical semantics of higher-order store. In *Proceedings of CTCS '02*, number 69 in ENTCS. Elsevier, 2002.
18. 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.
19. R. de Nicola M. Boreale and R. Pugliese. Trace and testing equivalence on asynchronous processes. *Information and Computation*, 172(2):139–164, 2002.
20. E. Moggi M. Fiore and D. Sangiorgi. A fully abstract model for the π-calculus. In *Proceedings of LICS '96*, 2996.
21. P. Malacaria and C. Hankin. Generalised flowcharts and games. In *Proceedings of the 25th International Colloquium on Automata, Langugages and Programming*, 1998.
22. G. McCusker. *Games and full abstraction for a functional metalanguage with recursive types*. PhD thesis, Imperial College London, 1996. Published by Cambridge University Press.
23. R. Milner. Polyadic π-calculus: a tutorial. In *Proceedings of the Marktoberdorf Summer School on Logic and Algebra of Specification*, 1992.
24. J. Power and H. Thielecke. Environments in Freyd categories and κ-categories. In *Proceedings of ICALP '99*, number 1644 in LNCS. Springer, 1999.
25. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1993.
26. I. Stark. A fully abstract domain model for the π-calculus. In *Proceedings of LICS '96*, 1996.