Recursive Types in Games: Axiomatics and Process Representation
(Extended Abstract)

Marcelo Fiore*
<marcelo@cogs.susx.ac.uk>
COGS, University of Sussex

Kohei Honda
<kohei@dcs.ed.ac.uk>
LFCS, University of Edinburgh

Abstract

This paper presents two basic results on game-based semantics of FPC, a metalanguage with sums, products, exponentials and recursive types. First we give an axiomatic account of the category of games **G** *introduced in [15], offering a fundamental structural analysis of the category as well as a transparent way to prove computational adequacy. As a consequence we obtain an intensional full-abstraction result through a standard definability argument. Next we extend the category* **G** *by introducing a category of games* $\mathbf{G}_i$ *with optimised strategies; we show that the denotational semantics in* $\mathbf{G}_i$ *gives a compilation of FPC terms into core Pict codes (the asynchronous polyadic* π*-calculus without summation). The process representation follows a pioneering idea of Hyland and Ong [18]. However, we advance their representation by introducing semantically well-founded optimisation techniques; we also extend the setting to encompass the rich type structure of FPC. The resulting code gives basic insight on the relationship between the abstract, categorical, types and their possible implementations.*

1 Introduction

Games in semantics. Recently, the notions of games and strategies have been used for constructing mathematical models of programming languages; e.g. [4, 1, 17, 7, 18, 2, 22, 15, 3]. The basic common idea underlying these works is to construct a universe of semantic domains in which a program phrase is modelled as an object with internal structure reflecting its computational behaviour in an abstract way. In particular, semantic points are interacting processes called *strategies*, where interaction represents the behaviour of a program in all contexts.

We mention some salient features of this approach. First, game semantics offers a very precise abstract characterisation of computational behaviour leading to various full abstraction results. Second, despite its novel way of representing programs, game-theoretic models are equipped with familiar semantic structures offering traditional mathematical tools for reasoning about programs. Third, game-theoretic models are closely related to existing models of interaction; e.g. π-calculus [25] and Linear Logic [12]. We expand on this point next, focussing on the relationship to π-calculus.

Process representation of games. Following their full abstraction result for (call-by-name) PCF, Hyland and Ong [18] observed that their strategies could be faithfully represented by terms in the π-calculus. This observation suggests a close connection between the games model and π-calculus representation of programming languages (see e.g. [23, 37]). It also opens the possibility of studying dynamics of programs at a level close to implementation using game semantics.

Central themes of this paper are a further investigation in this direction, and an axiomatisation of a category of games following [9, 10]. Among other things, we aim to advance the work of Hyland and Ong [18] in a couple of points.

- Many programming languages adopt a call-by-value evaluation strategy and have more intricate type structure like sums and recursive types, cf. [26]. Can game semantics handle these features, and allow process representation in this setting?
- A close inspection of the process representation in [18] reveals that the frequent use of the "copy-cat" (or "tit-for-tat") behaviour results in quite redundant interactions. This dissociates the resulting code from any feasible implementation. Can one obtain better code from a game-theoretic model of a programming language?

*Research supported by EPSRC grant GR/J84205, Frameworks for Programming Language Semantics and Logic at LFCS, University of Edinburgh.

Results. This work addresses the above mentioned issues taking FPC [30] as an example programming language. FPC is essentially the functional core of ML; we have chosen it for its rich type structure and call-by-value operational semantics. Moreover its primitive character is useful to examine the basic features of our approach in a simple, though non-trivial, setting. Our results can be summarised as follows.

First, we present a fully abstract interpretation of FPC in the category of games $\mathbf{G}$ introduced in [15]. An important contribution here is an axiomatic approach, which offers a basic categorical analysis of the universe of games and a transparent way to prove computational adequacy.

Second, we introduce a new category of games $\mathbf{G}_i$ extending $\mathbf{G}$. Objects in $\mathbf{G}_i$ are as in $\mathbf{G}$, but the hom-sets contain "optimised" versions of the strategies in $\mathbf{G}$. The key idea is to represent the "copy-cat" behaviour by a tree isomorphism. Computational adequacy is again obtained from the axiomatisation referred to above.

Finally, we transform the interpretation of FPC in $\mathbf{G}_i$ into core Pict (the asynchronous polyadic π-calculus) [29], and show that the translation faithfully represents the original strategies. The resulting code exhibits a close link between the abstract type structures and their implementation, cast in the setting of name-passing processes.

Related work. First we comment on the relationship between the axiomatisation in this paper and that of [9, 10]. Both axiomatisations take the same conceptual standpoint based on the interplay between a category of partial maps and a subcategory of total maps there of. However, in order to accommodate the intensional nature of games, here we have to weaken the categorical structure for modelling product types (from monoidal to *binoidal*). Consequently, we have axiomatised the information-system approach to solving domain equations rather than the one based on the limit-colimit coincidence. This yields new computationally adequate models of FPC. (See Section 3 '§ Basic Axiomatisation' for details.)

We know of two other full abstraction results for FPC. McCusker obtained a full abstraction result for *call-by-name* FPC [22], while Riecke and Sandholm obtained full abstraction for (call-by-value) FPC via logical relations [33]. The first one differs from ours in that the target language is call-by-name, while the second one uses quite a different construction and so offers a very different semantic analysis. We also note that Abramsky and McCusker [3] presented a category of call-by-value games closely related to the one in [15]. They independently gave a fully abstract interpretation of recursive types in their universe.

Regarding process representation of strategies, we already discussed how our work builds on previous work of Hyland and Ong [18]. The incorporation of rich type structure as well as the new more viable encoding are the main additional contributions of our work. In particular, our treatment of recursive types offers new insights on the relationship between the abstract notion of recursive type (as free algebras [11]) and their implementation (via pointers). Among the work on π-calculus representation of λ-calculi, cf. [23, 6, 35, 5], a new contribution of our work is the type-theoretic (or category-theoretic) perspective it offers. For example, the difference between "eager sum" and "lazy sum" is articulated at the categorical level and consequently in the operational structure. Thus our work broadens the repertoire of π-calculus representation of functional calculi, with type-theoretic background.

Organisation of the paper. In Section 2 we review the category $\mathbf{G}$ of games introduced in [15]. In Section 3 we present new axiomatic models of FPC, and establish their computational soundness and adequacy. We apply this result to $\mathbf{G}$, and obtain intensional full abstraction for FPC. In Section 4 we introduce a new category of games $\mathbf{G}_i$ with optimised strategies. Computational adequacy for $\mathbf{G}_i$ is again obtained from the axiomatic method of Section 3. In Section 5 we give a compilation of FPC into Pict through its denotational semantics in $\mathbf{G}_i$, and establish the computational adequacy of the compilation.

2 Basic games

We outline the basic notions of games introduced in [15]. The key difference from preceding games lies in that interaction becomes data-driven (starts from the arrival of data from the environment) rather than demand-driven (starts from the inquiry of data), thus successfully modelling call-by-value computation. This means that *arenas* —the notion of type in game-based semantics— start from Answers rather than Questions, as formalised below.

Arena. A *forest* is a directed graph in which every connected component is a tree. Nodes are denoted S, T, etc. We write $S \mapsto T$ if there is a directed edge from S to T. The relation $\mapsto$ is sometimes called *enabling*. So a forest is presented by $(\Sigma, \mapsto \subseteq \Sigma^2)$ where Σ is the set of nodes. In a forest, a node S is said to be *initial* if for no T we have $T \mapsto S$.

A *pre-arena* is a forest $(\Sigma, \mapsto \subseteq \Sigma^2)$ together with labelling functions $\ell_1 : \Sigma \to \{P, O\}$ and $\ell_2 : \Sigma \to \{Q, A\}$ such that whenever $S \mapsto T$ the following are satisfied: (1) $\ell_1(T) = \overline{\ell_1(S)}$ where $\overline{P} \stackrel{\text{def}}{=} O$ and $\overline{O} \stackrel{\text{def}}{=} P$ (Opponent justifies Player, and vice versa) and (2) if $\ell_2(S) = A$ then $\ell_2(T) = Q$ (a Question justifies both Answers and Questions, while an Answer justifies only Questions). The nodes in a pre-arena are called *sorts*. We often write ℓ for the pairing $\langle \ell_1, \ell_2 \rangle$ of the labelling functions.

Let $A = (\Sigma, \mapsto, \ell)$ and $A' = (\Sigma', \mapsto', \ell')$ be pre-arenas. The *dual* pre-arena $\overline{A}$ has the same set of sorts and enabling relation as the pre-arena A, with Opponent/Player reversed: that is $\overline{\ell} \stackrel{\text{def}}{=} \langle \overline{\ell_1}, \ell_2 \rangle$ where $\overline{\ell_1}(S) \stackrel{\text{def}}{=} \overline{\ell_1(S)}$.

The *disjoint union* pre-arena $A \uplus A'$ has sorts given by the disjoint union $\Sigma \uplus \Sigma'$, with enabling relation and labelling functions as in A and A'.

The pre-arena A is said to be a *sub-pre-arena* of the pre-arena A' if (1) every initial sort in A is initial in A'; (2) $\Sigma \subseteq \Sigma'$; (3) $\mapsto \subseteq \mapsto'$; and (4) $\ell = \ell' \restriction \Sigma$.

Finally an *arena* is a pre-arena in which initial sorts are labelled PA.

Action sequence. Given arenas A and B, an *action sequence* from A to B is a finite sequence of sorts $S_0 \cdot \ldots \cdot S_n$ in the pre-arena $\overline{A} \uplus B$, where occurrences of sorts are called *actions* (often referred to as P-action, O-action, Question, and Answer according to the labels), together with a *justification* relation $\curvearrowright \subseteq [0..n-1] \times [1..n]$ (where we read $i \curvearrowright j$ as i *justifies* j) such that: (1) the sort S_0 is initial in $\overline{A}$; (2) there exists at most one $j \in [1..n]$ which is not justified and, for such j, S_j is initial in B; (3) Opponent and Player alternate; (4) no two actions justify the same action; (5) if i justifies j then S_i precedes and enables S_j; and (6) every Question justifies at most one Answer.

An action sequence represents a possible interaction sequence a process may be engaged in. A *view* [17] is part of such a sequence used by a process to decide its next action.

View. The P-*view* $\lceil s \rceil$ of an action sequence s is the sequence with justification relation inherited from s, inductively defined by (Pv1) $\lceil \varepsilon \rceil = \varepsilon$; ($P$v2) $\lceil S \rceil = S$; (Pv3) $\lceil s_0 S \rceil = \lceil s_0 \rceil \cdot S$ whenever S is a P-action; (Pv4) $\lceil s_0 \cdot S_i \cdot s_1 \cdot S_j \rceil = \lceil s_0 \rceil \cdot S_i \cdot S_j$ whenever $i \curvearrowright j$ and S_j is an O-action. The O-view of an action sequence s is denoted $\lfloor s \rfloor$ and is defined dually (that is, by exchanging the rôles of P and O actions in the previous definition). By the *view* of an action S in an action sequence $s \cdot S$ we mean the P-view $\lceil s \cdot S \rceil$ if S is a P-action, and the O-view $\lfloor s \cdot S \rfloor$ if S is an O-action.

Innocent strategy. An action sequence s is said to be *legal* when: it is *well-bracketed*, i.e. later asked questions are answered first, and it satisfies the *visibility condition*, i.e. every action is justified by an action in its view. We are now in a position to define *morphisms* between arenas. Let A and A' be arenas. An *innocent strategy* σ from A to A' (in symbols, $\sigma : A \rightharpoonup A'$) is a non-empty set of prefix-closed legal action sequences from A to A' satisfying:

(Contingency completeness) If s in σ ends with a P-action and $s' = s \cdot S$ is legal then s' is also in σ.

(Determinacy) Whenever $s \cdot S$ and $s \cdot S'$ are in σ with S and S' P-actions, $s \cdot S = s \cdot S'$.

(Innocence) For action sequences $s \cdot S$ and s' in σ with S a P-action, if $\lceil s \rceil = \lceil s' \rceil$ then $s' \cdot S$ is an action sequence in σ with $\lceil s' \cdot S \rceil = \lceil s \cdot S \rceil$.

By a *P-view from A to B* we mean the P-view of some legal action sequence from A to B. Notice that each innocent strategy σ is precisely characterised by its *innocent function*; namely, the partial function that extends every odd-length P-view by σ's reaction to it.

An innocent strategy $\sigma : A \rightharpoonup A'$ is *total* if for each initial O-answer it immediately reacts by an initial P-answer; that is, for every initial sort S in $\overline{A}$ there exists an initial sort S' in A' such that the action sequence $S \cdot S'$ (with empty justification relation) is in σ. To indicate that σ is total we either write $\sigma : A \rightarrow A'$ or $\sigma \downarrow$.

A sub-arena A of an arena A' induces a *substructure* total innocent strategy $A \rightarrow A'$ given by the strategy that copies every O-action in $\overline{A}$ to a P-action in A', and copies every O-action in A' appearing in A to a P-action in $\overline{A}$.

Composition. Given innocent strategies $\sigma : A \rightharpoonup B$ and $\tau : B \rightharpoonup C$, we define their composition $\sigma; \tau$ as follows. Consider a game board with 4 rows and ω columns whose rows are respectively called $\overline{A}$, B, $\overline{B}$ and C. Columns are counted from 1. A *configuration* consists of a game board together with a binary relation, called *justification*, on squares such that: (1) squares are filled (if ever) by sorts from $\overline{A}$, B, $\overline{B}$, C, each placed in a square of the corresponding row; (2) for some n, at each i^{th} column ($1 \leq i \leq n$) either only the square at row $\overline{A}$, or only the square at row C, or only both squares at rows B and $\overline{B}$ are filled, moreover no squares at later columns are filled; (3) only squares in the same row are related by the justification relation; (4) the projections onto B and $\overline{B}$ coincide (including the justification relation). A configuration is *legal* if, when projected onto $\overline{A}$-B (resp. $\overline{B}$-C), it induces a legal sequence from A to B (resp. from B to C). Finally a *composed play* of σ and τ is a legal configuration such that the $\overline{A}$-B (resp. the $\overline{B}$-C) projection is in σ (resp. in τ). It was shown in [15, pp.31–40] that the set of $\overline{A}$-C projections of the composed plays yields an innocent strategy from A to C, which we define to be the composition $\sigma; \tau$.

Category of Games. The category of games $\mathbf{G}$ has arenas as objects and innocent strategies as morphisms. Identities are given by the *copy-cat* strategy: that is the strategy which exactly copies the preceding O-action to the next P-action including the justification relation. We shall study these behaviours in more detail in Section 4.

We write $\mathbf{G}_t$ for the subcategory of $\mathbf{G}$ consisting of the total innocent strategies, and $\mathbf{G}_s$ for the subcategory of $\mathbf{G}_t$ consisting of the sub-structure total innocent strategies. $\mathbf{G}_s$ is actually a (large) cpo with lubs given by unions.

We order innocent strategies by inclusion of innocent functions, then each hom-set $\mathbf{G}(A, A')$ becomes a cpo. With respect to this order, henceforth denoted $\sqsubseteq$, the composition operation $\mathbf{G}(A', A'') \times \mathbf{G}(A, A') \rightarrow \mathbf{G}(A, A')$ becomes continuous [15]. Thus, $\mathbf{G}$ equipped with $\sqsubseteq$ is a $\mathbf{Cpo}$-category (see [9, Chapter 2] for the basic concepts of enriched category theory). The restriction of $\precsim$ to total innocent strategies is denoted $\sqsubseteq$; it makes $\mathbf{G}_t$ a $\mathbf{Cpo}$-category.

Type structure. We outline the basic construction of types in $\mathbf{G}$. First $\mathbf{0}$ is the empty arena (no sorts), while $\mathbf{1}$ is the arena with a (distinguished) single sort. The *sum* of arenas $+$ is simply given by their disjoint union. The *pretensor* arena $A \otimes A'$ of the arenas A and A' has forest consisting of the trees obtained as the coalesced sum of every tree in the forest of A with every tree in the forest of A'; the labelling function is inherited from each component. The *higher-order* arena $A \rightharpoonup A'$ of the arenas A and A' has forest consisting of one new initial sort enabling (every tree in) the forest of A and every initial sort of A enabling a copy of (every tree in) the forest of A'; the labelling function labels the new initial sort with PA, each initial sort in A with OQ, each non-initial sort S in A with its dual $\overline{\ell}(S)$, and each sort in a copy of A' with the same label.

3 Denotational semantics of FPC

FPC is a metalanguage with sums, products, exponentials and recursive types, equipped with a call-by-value evaluation. The definition of FPC can be found in [13, 38, 9]. Here we only provide the syntax of types and expressions.

Types

$$\alpha \quad ::= \quad \mathtt{T} \mid \alpha_1 + \alpha_2 \mid \alpha_1 \times \alpha_2 \mid \alpha_1 \rightharpoonup \alpha_2 \mid \mu\mathtt{T}.\alpha,$$

Expressions

$$\begin{aligned} M \quad ::= \quad & \mathtt{x} \mid \mathtt{inl}_{\alpha_1,\alpha_2}(M) \mid \mathtt{inr}_{\alpha_1,\alpha_2}(M) \mid \\ & \mathtt{case}\ M\ \mathtt{of}\ \mathtt{inl}(\mathtt{x}_1).M_1\ \mathtt{or}\ \mathtt{inr}(\mathtt{x}_2).M_2 \mid \\ & \langle M_1, M_2\rangle \mid \mathtt{fst}(M) \mid \mathtt{snd}(M) \mid \lambda\mathtt{x} : \alpha.M \mid \\ & M_1(M_2) \mid \mathtt{intro}_{\mu\mathtt{T}.\alpha}(M) \mid \mathtt{elim}(e). \end{aligned}$$

Our aim is to show that the category $\mathbf{G}$ allows computationally adequate interpretations of FPC via an axiomatic method (cf. [9]). The axiomatics elucidates the type structure of the category $\mathbf{G}$, providing an algebraic method for reasoning about strategies. It also plays an important rôle in later sections.

Basic axiomatisation. We start by axiomatising a notion of *basic model* giving the data for interpreting FPC. These basic models are computationally sound. The notion of *model*, will be introduced in “§ Axiomatisation” below, by requiring further axioms on basic models.

The axiomatisation of basic models (see Definition 3.1) can be understood as consisting of two parts: (1) the data (BMa)–(BMb) subject to (BM1)–(BM2), providing the data for interpreting the type constructors $+$, $\times$, and $\rightharpoonup$; and (2) the datum (BMc) subject to (BM3), allowing the interpretation of the recursive-type constructor μ.

The categorical structure for interpreting the sum type constructor $+$ is standard; that is, given by binary coproducts. It then follows that the interpretation of the product type constructor $\times$ cannot be standard, viz. given by products (cf. [20, 16]). However, the category of games $\mathbf{G}$ admits a pretensor constructor $\otimes$ (see Section 2, § Type structure) which is a product in the subcategory, $\mathbf{G}_t$, of total innocent strategies [15]. This situation, which is typical in models of FPC [9, 10], is the one that we axiomatise. The structure for interpreting the exponential type constructor $\rightharpoonup$ arises as in partial cartesian closed categories [21, 27, 34].

We have already mentioned that the pretensor constructor $\otimes$ on the category of games $\mathbf{G}$ is not a product. However, for every pair of arenas A and B, we have endofunctors $A\otimes(\_)$ and $(\_)\otimes B$ on $\mathbf{G}$ such that $A\otimes(B) = (A)\otimes B$ [15]. That is, in the terminology of [31], $\mathbf{G}$ equipped with the family of endofunctors $A \otimes \_$ and $\_ \otimes B$ ($A, B \in \mathbf{G}$) is a *binoidal category*.

Now, consider the type judgement $\mathtt{A}, \mathtt{T} \vdash \mathtt{1} + \mathtt{T} \times \mathtt{A} \times \mathtt{T}$. It induces a function on arenas mapping a pair of arenas (A, T) to the arena $\mathbf{1} + T \otimes A \otimes T$, which by the previous considerations does not extend to a functor $\mathbf{G}^2 \to \mathbf{G}$. Thus one cannot interpret type judgements $\mathtt{T}_1, \ldots, \mathtt{T}_n \vdash \alpha$ as functors $\mathbf{G}^n \to \mathbf{G}$, and consequently one cannot interpret recursive types as *parameterised free algebras* [11, 9, 30, 32]. Our approach to solving this problem is to restrict attention to a subcategory of $\mathbf{G}$ in which type judgements can be interpreted as functors, and find the recursive types there. This idea is traditional in domain theory. For example, it occurs when solving domain equations of functors of mixed variance when one may restrict a $\mathbf{Cpo}$-category to its subcategory of embeddings [36], and it occurs when solving domain equations with information systems when one restricts from the category of information systems to a pointed cpo of them [19, 38]. There are various ways in which one may restrict $\mathbf{G}$ so that the action of $\otimes$ on arenas extends to a functor. One possibility is to consider the *centre* of $\mathbf{G}$ (see [31] for this concept) which in our case coincides with $\mathbf{G}_t$. Here, rather than doing so, we will axiomatise the information-system approach (as has been done in [8, 22]). In our example, this will amount to interpreting type judgements $\mathtt{T}_1, \ldots, \mathtt{T}_n \vdash \alpha$ as continuous functions ${\mathbf{G}_s}^n \to \mathbf{G}_s$ and finding recursive types with Scott's least fixed-point operator (or rather with Bekič's parameterised version of it). Thus recursive types will be interpreted up to equality; this slightly simplifies the proof of computational adequacy (cf. [9, Chapters 9 and 10]).

An extra ingredient of basic models is a $\mathbf{Cpo}$-enrichment axiomatising the information order. This structure is not needed to obtain computational soundness, but plays a central rôle when proving computational adequacy.

We now introduce basic models; an explanation of the definition in elementary terms will follow. It will be helpful to think of $\mathcal{G}$ as $\mathbf{G}$, of $\mathcal{T}$ as $\mathbf{G}_t$, and of $\mathcal{S}$ as $\mathbf{G}_s$.

Definition 3.1 (Basic models) A basic model *consists of*

(BMa) *a* $\mathbf{Cpo}$*-category,* $\mathcal{G}$*, equipped with a* $\mathbf{Cpo}$*-binoidal structure* $(X \otimes (\_), (\_) \otimes Y)$*,*

(BMb) *a full on objects* $\mathbf{Cpo}$*-subcategory of* $\mathcal{G}$*,* $\mathcal{T}$*, equipped with finite* $\mathbf{Cpo}$*-coproducts* $(0, +)$ *and finite* $\mathbf{Cpo}$*-products* $(1, \times)$*,*

(BMc) *a full on objects subcategory of* $\mathcal{T}$*,* $\mathcal{S}$*,*

such that

(BM1) *the inclusion* $\mathcal{T} \to \mathcal{G}$ *is strict binoidal,*

(BM2) *for each* X*, the* $\mathbf{Cpo}$*-functor* $\_ \otimes X : \mathcal{T} \to \mathcal{G}$ *has a right* $\mathbf{Cpo}$*-adjoint* $X \rightharpoondown \_ : \mathcal{G} \to \mathcal{T}$*,*

(BM3) $\mathcal{S}$ *is a pointed cpo, and for* $\star = +, \otimes, \rightharpoondown$*, the function* $\star : \mathcal{S} \times \mathcal{S} \to \mathcal{S} : (X, Y) \mapsto X \star Y$ *is continuous.*

The requirement (BM1) amounts to asking for $\mathbf{Cpo}$-functors $X \otimes (\_) : \mathcal{G} \to \mathcal{G}$ and $(\_) \otimes Y : \mathcal{G} \to \mathcal{G}$ for all $X, Y \in |\,\mathcal{G}\,|$ such that (i) $X \otimes (Y) = (X) \otimes Y = X \times Y$; (ii) for all f in $\mathcal{T}$, $X \otimes (f) = X \times f$, $(f) \otimes Y = f \times Y$; and (iii) every map in $\mathcal{T}$ is *central [31]* in $\mathcal{G}$; that is, every $f : A \to B$ in $\mathcal{T}$ has the property that, for every $u : X \rightharpoonup Y$ in $\mathcal{G}$, $(A \otimes u); (f \otimes Y) = (f \otimes X); (B \otimes u)$ and $(u \otimes A); (Y \otimes f) = (X \otimes f); (u \otimes B)$.

As a consequence of (BM2), the functor $\_ \otimes X$ preserves $\mathbf{Cpo}$-colimits. Thus, $\mathcal{G}$ has finite $\mathbf{Cpo}$-coproducts given as in $\mathcal{T}$. Moreover, there is a canonical isomorphism

$$(1) \quad \begin{aligned} \delta_{X,Y,Z} : X \otimes (Y + Z) &\cong (Y + Z) \otimes X \\ &\cong (X \otimes Y) + (X \otimes Z) \\ &\cong (Y \otimes X) + (Z \otimes X) \end{aligned}$$

natural with respect to maps in $\mathcal{T}$. This map together with the canonical map $\Delta_X \stackrel{\text{def}}{=} \langle \mathrm{id}, \mathrm{id} \rangle : X \to X \otimes X$ will be useful when interpreting the `case` statement.

Interpretation. We sketch the denotational semantics of FPC in basic models.

Throughout this section we fix a basic model $(\mathcal{G}, \mathcal{T}, \mathcal{S})$. We interpret well-formed types $\Theta \vdash \alpha$, where Θ is a list of distinct type variables containing those that are free in α, as continuous functions $[\![\Theta \vdash \alpha]\!] : \mathcal{S}^{|\Theta|} \to \mathcal{S}$. The definition is by induction on types, interpreting the sum type constructor as $+$, the product type constructor as $\otimes$, the exponential type constructor as $\rightharpoondown$, and the recursive-type constructor as Bekič's parameterised least fixed-point operator $(\_)^\dagger$ (see Figure 1). The interpretation of well-formed types induces the interpretation of well-formed expression contexts $\Theta \vdash \Gamma$ as continuous functions $[\![\Theta \vdash \Gamma]\!] : \mathcal{S}^{|\Theta|} \to \mathcal{S}$ in the usual way.

Lemma 3.2 (Substitution lemma) *For* $\Theta, \mathtt{T} \vdash \alpha_1$ *and* $\Theta \vdash \alpha_2$*, we have that*

$$[\![\Theta \vdash \alpha_1[\mathtt{T} \mapsto \alpha_2]]\!] \quad = \quad \langle \mathrm{Id}, [\![\Theta \vdash \alpha_2]\!] \rangle ; [\![\Theta, \mathtt{T} \vdash \alpha_1]\!] \ .$$

Hence, for $\Theta \vdash \mu\mathtt{T}.\alpha$*, the equality* $[\![\Theta \vdash \alpha[\mathtt{T} \mapsto \mu\mathtt{T}.\alpha]]\!] = [\![\Theta \vdash \mu\mathtt{T}.\alpha]\!]$ *holds.*

Convention. Maps in $\mathcal{G}$ will be indicated with $\rightharpoonup$. For a map $u : X \rightharpoonup Y$, we write $u : X \to Y$ or $u \downarrow$ to further indicate that it is in the subcategory $\mathcal{T}$.

The interpretation of a well-formed expression $\Theta, \Gamma \vdash M : \alpha$ is a family $[\![\Theta, \Gamma \vdash M : \alpha]\!] = \{[\![\Theta, \Gamma \vdash M : \alpha]\!]_X \mid X \in \mathcal{S}^{|\Theta|}\}$ where $[\![\Theta, \Gamma \vdash M : \alpha]\!]_X$ is a map $[\![\Theta \vdash \Gamma]\!](X) \rightharpoonup [\![\Theta \vdash \alpha]\!](X)$. The definition of the interpretation function is essentially that of [9, 10] and is given in Figure 2.

We write $\Theta, \Gamma \vdash M \rightsquigarrow V : \alpha$ to indicate that both $\Theta, \Gamma \vdash M : \alpha$ and $\Theta, \Gamma \vdash V : \alpha$ are derivable and that M evaluates to V according to call-by-value evaluation rules. We also write $\Theta, \Gamma \vdash M \checkmark : \alpha$ when $\Theta, \Gamma \vdash M \rightsquigarrow V : \alpha$ for some value V.

Lemma 3.3 *In every basic model, (1)* $[\![\Theta, \Gamma \vdash V : \alpha]\!] \downarrow$ *for every value* V*, and (2) if* $\Theta, \Gamma \vdash M \rightsquigarrow V : \alpha$ *then* $[\![\Theta, \Gamma \vdash M : \alpha]\!] = [\![\Theta, \Gamma \vdash V : \alpha]\!]$.

Corollary 3.4 (Computational soundness) *In every basic model, if* $\Theta, \Gamma \vdash M \checkmark : \alpha$ *then* $[\![\Theta, \Gamma \vdash M : \alpha]\!] \downarrow$.

Axiomatisation. The $\mathbf{Cpo}$-enrichment on $\mathcal{G}$ will be denoted $\underset{\sim}{\sqsubseteq}$. For maps $u \underset{\sim}{\sqsubseteq} v$ in $\mathcal{G}$, we write $u \sqsubseteq v$ to indicate that u and v are in $\mathcal{T}$. Thus, $\sqsubseteq$ denotes the $\mathbf{Cpo}$-enrichment on $\mathcal{T}$.

Roughly speaking, a model (see Definition 3.5) is a basic model that (1) is not trivial; and in which (2) composition is strict (in accordance with call-by-value evaluation); (3) definedness is an observable (or semi-decidable) property; (4) the information order on $\mathcal{T}(X, A_1 + A_2)$ for maps of the form $X \to A_i \xrightarrow{\amalg_i} A_1 + A_2$ is determined by the information order on $\mathcal{T}(X, A_i)$; (5) the approximation structure provided by $\mathcal{S}$ corresponds to standard approximation structure in $\mathcal{G}$ (viz. embeddings [36]), and moreover is compositional with respect to the type structure.

Definition 3.5 (Model) A model *is a basic model that further satisfies:*

(M1) $0 \not\cong 1$.

(M2) *For* $u : X \rightharpoonup Y$ *and* $v : Y \rightharpoonup Z$*, if* $u; v \downarrow$ *then* $u \downarrow$.

(M3) *For every* X*, the hom-set* $\mathcal{T}(1, X)$ *is Scott-open in the hom-cpo* $\mathcal{G}(1, X)$.

$$[\![\Theta \vdash \Theta_i]\!] \stackrel{\text{def}}{=} \amalg_i \qquad [\![\Theta \vdash \alpha_1 + \alpha_2]\!] \stackrel{\text{def}}{=} \langle [\![\Theta \vdash \alpha_1]\!], [\![\Theta \vdash \alpha_2]\!] \rangle ; +$$

$$[\![\Theta \vdash \alpha_1 \times \alpha_2]\!] \stackrel{\text{def}}{=} \langle [\![\Theta \vdash \alpha_1]\!], [\![\Theta \vdash \alpha_2]\!] \rangle ; \otimes \qquad [\![\Theta \vdash \alpha_1 \rightharpoonup \alpha_2]\!] \stackrel{\text{def}}{=} \langle [\![\Theta \vdash \alpha_1]\!], [\![\Theta \vdash \alpha_2]\!] \rangle ; \rightharpoonup$$

$$[\![\Theta \vdash \mu \mathtt{T}.\alpha]\!] \stackrel{\text{def}}{=} [\![\Theta, \mathtt{T} \vdash \alpha]\!]^{\dagger}$$

Figure 1. Interpretation of well-formed types

(M4) The coproduct injections $\amalg_i$ reflect the order in $\mathcal{T}$. That is, if $f; \amalg_i \sqsubseteq g; \amalg_i : X \to A_1 + A_2$ then $f \sqsubseteq g : X \to A_i$.

(M5) Every map in $\mathcal{S}$ is an embedding in $\mathcal{G}$ (and hence it is in $\mathcal{T}$), and the inclusion functor $\mathcal{S} \to \mathcal{G}$ preserves the initial object and colimits of ω-chains.

Moreover, writing $\iota_{X,Y}$ for the unique arrow $X \to Y$ in $\mathcal{S}$ whenever it exists and writing $\jmath_{Y,X}$ for its associated projection $Y \rightharpoonup X$, the following equalities hold: $\iota_{X+Y,X'+Y'} = \iota_{X,X'} + \iota_{Y,Y'}$, $\iota_{X \otimes Y, X' \otimes Y'} = \iota_{X,X'} \times \iota_{Y,Y'}$, and $\iota_{X \rightharpoonup Y, X' \rightharpoonup Y'} = \jmath_{X',X} \rightharpoonup \iota_{Y,Y'}$.

As expected, $(\mathbf{G}, \mathbf{G}_t, \mathbf{G}_s)$ provides a model. For instance, to see that (M5) holds one uses the fact that the inclusion functor $\mathcal{S} \to \mathcal{G}$ preserves colimits of ω-chains if and only if, for every ω-chain $\langle X_i \rangle$ in $\mathcal{S}$ with lub X, it happens that $\bigsqcup_i \jmath_{X,X_i} ; \iota_{X_i,X} = \mathrm{id}_X$ (see [36]).

Relation to operational semantics. We have the following main results.

Theorem 3.6 (Computational adequacy) *In every model, if $[\![\vdash P : \alpha]\!] \downarrow$ then $\vdash P \checkmark : \alpha$.*

The proof follows the structure of the one in [9, Chapter 9] (see also [10]).

Corollary 3.7 *In the category of games* **G***, the innocent strategy $[\![\vdash P : \alpha]\!]$ is total iff $\vdash P \checkmark : \alpha$.*

Theorem 3.8 (Full abstraction) *The standard quotient of* **G** *(cf. [15]) gives an inequationally fully abstract model of FPC.*

The proof is along the lines of [15, Theorems 5.9 and 5.10] via a definability result in **G** for *finite canonical forms* of *finite type* (cf. [22]).

4 Games with optimised strategies

Two issues in process representation of strategies. This section and the next exploit the intensional nature of game semantics to obtain a semantically directed compilation of FPC terms into Pict codes [29]. Our aim is to deepen the understanding of the operational content of FPC programs through a process representation of their game-theoretic interpretation. From this viewpoint, we may observe the following two basic issues in the original process representation of PCF strategies by Hyland and Ong [18].

(1) (Problem of Sums) When infinitely branching arenas are involved (as, for example, in lists) we should either use an infinite sequence of names (which is infeasible) or use an index from an infinite set of constants as is done in [18] (which is again infeasible when complex data structures are present, as in FPC).

(2) (Problem of Copy-Cat) While copy-cat-like behaviours are prevalent (e.g. in the interpretation of projections and injections), their process representation in [18] reveals significant redundancy. This makes the resulting code hard to understand and far from any imaginable implementation scheme. For example, if the process representation of a closed term $M(N)$ needs n steps to converge, then that of $M(IN)$ (where $I \stackrel{\text{def}}{=} \lambda \mathtt{x}.\mathtt{x}$) needs, essentially, $2n$ steps to converge. This is an omnipresent problem. For example, when a term involves unbounded behaviour (like a fixed-point combinator) it is impossible to statically erase the redundancy.

To obtain feasible and concise code without sacrificing the structure of the interpretation of FPC in **G**, we address these issues at the categorical level. The first problem is solved by introducing a *lazy* interpretation of sums, as will be presented in Section 5. A solution to the second problem is given in this section by extending **G** via a slight change in the operational structure of games. The key idea for the new notion of games is to use certain tree isomorphisms to represent copy-cats. As we shall see in Section 5, this new relation on actions represents the *communication of free names* in π-calculus. The construction is interesting in its own right, since it gives an analysis on the notion of copy-cat, a fundamental idea in game semantics.

V-strategies. We use the same arenas as in Section 2. Fix arenas A and B and consider the pre-arena $\overline{A} \uplus B$. We write $\widehat{S}$ for the tree with root S in this pre-arena, and introduce the notation $\Psi : \widehat{S_1} \bowtie \widehat{S_2}$ to indicate that Ψ is an isomorphism between the trees $\widehat{S_1}$ and $\widehat{S_2}$ that preserves QA-labels but reverses OP-labels. Further, we write $\mathsf{source}(\Psi)$ for S_1 and $\mathsf{target}(\Psi)$ for S_2. A *V-sequence from A to B* is an action sequence in the sense of Section 2 (§ Action Sequence) together with a set of *V-pointers* on it. A V-pointer on an action sequence $S_0 \ldots S_{n-1}$ is a triple $i \overset{\Psi}{\Rightarrow} j$, satisfying: (1) $0 \le i < j \le n-1$; (2) S_i is by Opponent and S_j is by Player; (3) $\Psi : T' \bowtie T''$ for T' and T'' such that $S_i \mapsto T'$ and $S_j \mapsto T''$; (4) for no $j < k \le n-1$ such that $S_k = \mathsf{target}(\Psi)$ we have that $j \curvearrowright k$ ("free names do not bind"); and (5) if $l \overset{\Psi'}{\Rightarrow} j$ for some l and Ψ' such that $\mathsf{target}(\Psi) = \mathsf{target}(\Psi')$, then $l = i$ and $\Psi' = \Psi$ ("non-ambiguity in binding"). Prefix, equality, etc. on V-sequences are defined considering both justification and V-pointers. A V-sequence satisfies *the visibility condition* if it does so as an action sequence and, moreover, if for each V-pointer $i \overset{\Psi}{\Rightarrow} j$, i is in the view of j. A V-sequence is *legal* if it is both well-bracketed as an action sequence and satisfies the visibility condition. A *V-strategy* is an innocent strategy defined as in Section 2 by replacing "action sequences" with "V-sequences". A V-strategy is *total* when, as before, it immediately reacts at the codomain for each initial O-answer.

Note that all strategies in $\mathbf{G}$ are trivially V-strategies. As a non-trivial example, let $\mathrm{id}'_A : A \rightharpoonup A$ be the following V-strategy: letting Ψ be the evident isomorphism mapping $\overline{A}$ to A in $\overline{A} \uplus A$, for each initial O-answer S, the strategy answers by $\Psi(S)$, carrying one V-pointer for each S' such that $S \mapsto S'$, with the associated isomorphism $\Psi \upharpoonright \widehat{S'}$. No more actions are possible by (4) above.

Composition of V-strategies. An *extended V-sequence* is an action sequence with V-pointers which satisfies (1), (2), (3), and (5) of the preceding subsection. Legality and other notions for extended V-sequences are inherited from V-sequences.

Let $\sigma : A \rightharpoonup B$ be a V-strategy and let f_σ be its innocent function. We define f^+_σ to be the smallest partial function on P-views of extended V-sequences from A to B, satisfying:

- $f^+_\sigma \supseteq f_\sigma$;
- if s is in the range of f^+_σ with the underlying sequence $S_0 \ldots S_{2n-1}$ then, for each V-pointer $i \overset{\Psi}{\Rightarrow} 2n-1$ and for each P-view $s' = s \cdot S_{2n}$ such that $S_{2n} = \mathsf{target}(\Psi)$ and $2n-1 \curvearrowright 2n$, we have that $f^+_\sigma(s') = s' \cdot S_{2n+1}$ with $S_{2n+1} = \mathsf{source}(\Psi)$ and $i \curvearrowright 2n+1$, together with V-pointers $\{2n \overset{\Psi'_k}{\Rightarrow} 2n+1 \mid S_{2n} \mapsto S'_k\}$ where $\Psi'_k \overset{\mathrm{def}}{=} \Psi^{-1} \upharpoonright \widehat{S'_k}$.

One can check that f^+_σ is a well-defined partial function. We write σ^+ for the V-strategy on extended V-sequences generated by f^+_σ. Given V-strategies $\sigma : A \rightharpoonup B$ and $\tau : B \rightharpoonup C$, we define $\sigma;\tau$ by the following procedure. Below $x, x_i, \ldots$ denote occurrences of sorts on the game board.

(i) Composed plays are defined as in Section 2 using σ^+ and τ^+, but recording V-pointers.

(ii) Given each composed play, for each maximal sequence of V-pointers $x_0 \overset{\Phi_0}{\Rightarrow} x_1$, $x_2 \overset{\Phi_1}{\Rightarrow} x_3$, ..., $x_{2n} \overset{\Phi_n}{\Rightarrow} x_{2n+1}$ such that x_{2i-1} and x_{2i} are in the same column and $\mathsf{target}(\Phi_{i-1}) = \mathsf{source}(\Phi_i)$ for all $1 \le i \le n$, we erase all these V-pointers and instead put a V-pointer $x_0 \overset{\Phi'}{\Rightarrow} x_{2n+1}$ with $\Phi' \overset{\mathrm{def}}{=} \Phi_n \circ \Phi_{n-1} \circ \cdots \circ \Phi_0$.

(iii) We take the projection of the result of (ii) above onto the $\overline{A}$-C components including V-pointers, and define $\sigma;\tau$ as the set of all (non-extended) V-sequences in them.

For a strategy in $\mathbf{G}$, the above composition and that of Section 2 (§ Composition) coincide.

Proposition 4.1 *For V-strategies $\sigma : A \rightharpoonup B$ and $\tau : B \rightharpoonup C$, $\sigma;\tau$ is a V-strategy from A to C. Moreover, composition is associative and has id'_A (as defined above) as an identity.*

The proof is as for the category $\mathbf{G}$ (see [15]), checking innocence w.r.t. (composed) V-pointers too.

New category of games. The $\mathbf{Cpo}$-category $\mathbf{G}_i$ has arenas as objects and V-strategies as morphisms; identities and composition are as presented above; and the enrichment is given by the inclusion of innocent functions. The categories $\mathbf{G}_i$ and $\mathbf{G}$ have the same type structure. Universal arrows as projections and injections in $\mathbf{G}_i$ are obtained by turning copy-cats in the corresponding maps in $\mathbf{G}$ into V-pointers.

Proposition 4.2 *The category $\mathbf{G}_i$ is a model in the sense of Definition 3.5.*

5 Compilation of FPC into Pict via games

Lazy-sum interpretation. Having introduced $\mathbf{G}_i$ to solve the copy-cat problem, we present a solution to the problem of infinite sums. Our approach is to use an interpretation of FPC in which sums are interpreted *lazily*.

We consider a *lazy-sum* type $\oplus$ encapsulating a computation of sum type $+$. To this end we introduce the *lift functor* $(\_)_\perp \overset{\mathrm{def}}{=} 1 \rightharpoonup \_ : \mathcal{G} \to \mathcal{T}$, and define the *lazy-sum functor* $\oplus$ as the composite $+;(\_)_\perp : \mathcal{G} \times \mathcal{G} \to \mathcal{T}$. For a simple example consider the type of *lazy natural numbers* $\mu\mathtt{T}.\mathtt{1} \oplus \mathtt{T}$.

The lift functor comes equipped with an operation for encapsulating a value into a computation, $\mathsf{up}_X \overset{\mathrm{def}}{=} p\lambda(X \otimes 1 \cong X) : X \to X_\perp$, and a dual operation for evaluating a computation to a value, $\mathsf{dn}_X \overset{\mathrm{def}}{=} (X_\perp \cong X_\perp \otimes 1 \xrightarrow{\mathsf{ev}} X)$. By construction, we have that $\mathsf{up}_X;\mathsf{dn}_X = \mathrm{id}_X$. These operations together with the coproduct injections $\amalg_i$ provide corresponding injections for lazy-sums; viz. $\amalg'_i \overset{\mathrm{def}}{=}$

$(X_i \xrightarrow{\amalg_i} X_1 + X_2 \xrightarrow{\mathsf{up}_{X_1+X_2}} X_1 \oplus X_2)$. There is also a non-standard version of δ (see (1)) which is needed for interpreting the `case` statement, $\delta' \stackrel{\text{def}}{=} (X \otimes (Y \oplus Z) \xrightarrow{X \otimes \mathsf{dn}_{Y+Z}} X \otimes (Y + Z) \xrightarrow[\cong]{\delta} (X \otimes Y) + (X \otimes Z))$.

The *lazy-sum interpretation* is denoted by $(\![\_]\!)$ and is as the standard one but replacing $+$ by $\oplus$, δ by δ', and $\amalg_i$ by $\amalg_i'$, see Figure 3.

This lazy-sum interpretation is also computationally adequate.

Theorem 5.1 *In every model (in the sense of Definition 3.5) the lazy-sum interpretation is computationally sound and adequate.*

Untyped processes. The denotation of FPC terms via $(\![\cdot]\!)$ in $\mathbf{G}_i$ can be faithfully represented as concise (core) Pict codes [29], i.e. as terms of asynchronous Polyadic π-calculus. The set of *processes* of core Pict is given by the following grammar. Below let $a, b, \ldots$ and $x, y, \ldots$ range over a countable set $\mathcal{N}$ of names.

$$P \quad ::= \quad \overline{a}[x_1, \ldots, x_n] \ \mid \ a(x_1, \ldots, x_n).P \ \mid \ P|Q \ \mid \ (\nu x)P \ \mid \ !a(x_1, \ldots, x_n).P \ \mid \ 0$$

The notions of binding, free names, subject names (head of input/output) and object names (names carried in input/output) are standard. We let $P;Q$ stand for $(\nu\tilde{x})(P|Q)$ where $\tilde{x}$ are common free names in P and Q, while $\overline{a}(b) \cdot P$ denotes $(\nu b)(\overline{a}[b]|P)$ where b does not occur as a free object of a message in P. The structural congruence $\equiv$ and the reduction relation $\longrightarrow$ are again standard. We write $P \downarrow$ for $\exists P'.\ P \longrightarrow^* P'$ with $P' \not\longrightarrow$ and $P' \xrightarrow{l}$ for an output l. We use the late transition relation [25] with labels τ, $a(x_1 \ldots x_n)$ and $\overline{a}\langle(\nu A)x_1, \ldots, x_n\rangle$ where $A \subseteq \{x_1 \ldots x_n\}$. We set $sub(a(x_1 \ldots x_n)) = sub(\overline{a}\langle(\nu A)x_1 \ldots x_n\rangle) \stackrel{\text{def}}{=} a$ and $ob(a(x_1 \ldots x_n), i) = ob(\overline{a}\langle(\nu A)x_1 \ldots x_n\rangle, i) \stackrel{\text{def}}{=} x_i$. Using a sequence of input/output labels, say t, for which we assume the usual binding relation and binding convention, the weak transition $\stackrel{t}{\Longrightarrow}$ is again standard; and so is the weak bisimilarity $\approx$ (the late and early versions coincide [14]). We also use the following notion, cf. [23]: a π-term P is *strongly determinate* when for each P' such that $P \longrightarrow^* P'$, we have (1) $P' \xrightarrow{l} P_1$ and $P' \xrightarrow{l} P_2$ imply $P_1 \equiv P_2$; (2) $P' \xrightarrow{l}$ (l an output) implies $P' \not\longrightarrow$.

FPC sorting. In the lifted-sum interpretation, all types are interpreted in $\mathbf{G}_i$ as trees (i.e. arenas with at most one initial sort), as can be easily verified. This extends to environments. Henceforth, we restrict attention to *finitely branching* arenas induced by closed FPC types. These we call *FPC arenas*. (We believe that all FPC types lie in this class — note that, for closed α, $[\![\mu\mathtt{T}.\mathtt{T} \times \alpha]\!] = \mathbf{0}$.) Let A be an FPC arena. Since each branch of A as a tree is constructed according to the structure of a given FPC type, we can assign an ordering on the children nodes of each node of A, respecting the lexicographic ordering of tensor/sum composition (e.g. in $A_1 \oplus A_2$, the initial sort of A_1 is smaller). Given this ordering, the enabling relation can now be written $S \mapsto S_1, \ldots, S_n$, where the sequence $S_1 \ldots S_n$ gives all sorts enabled by S ordered as mentioned above. An *FPC sorting* is given by: (1) a family of pairwise disjoint countable sets of names $\mathcal{F} = \{\mathcal{F}_{(A,S)}\}$ indexed by pairs (A, S) consisting of an FPC arena A and a sort S in A, and (2) a partial function $\mathcal{F} \rightharpoonup \mathcal{F}^*$ following the just introduced enabling relation between sorts and their sequences. We write $a :_A S$, omitting the subindex when its clear from the context, to indicate that a is a name in $\mathcal{F}_{(A,S)}$. Under an FPC sorting, we now have the set of *well-sorted processes* following the standard definition [24]. Note that, in a sorted process, if $S \mapsto S_1, \ldots, S_n$ and $a : S$ then the name a carries (if ever) a sequence of names of length n of sorts $S_1, \ldots, S_n$.

Compilation. We fix an FPC sorting for the rest of the section. The compilation of FPC terms into Pict codes is given in Figure 4. For a given FPC term $\Gamma \vdash M : \alpha$, the compilation of M is parameterised by a sequence of pairwise distinct names, say $w_1 \ldots w_n u$, such that $w_i : S_i$ for each i and $u : S'$, where $S \mapsto S_1, \ldots, S_n$ for the initial sort S of $[\![\Gamma]\!]$ and S' is the initial sort of $[\![\alpha]\!]$. In the figure, we assume that all mentioned names (including those in each vector) are pairwise distinct unless textually identical, and that, their usage follows the given FPC sorting, including the length of each vector of names. Note that $\Gamma \equiv \mathtt{x}_1 : \beta_1, \ldots, \mathtt{x}_k : \beta_k$ *does not* imply $n = k$, since each β_i can itself be a non-trivial tensor. In this case, we may write $\tilde{v}_1\tilde{v}_2 \ldots \tilde{v}_k$ etc. for $\tilde{w}$, where each $\tilde{v}_i$ corresponds to β_i above. It is easy to check that the resulting processes are indeed well-sorted. Intuitively, the compilation can be considered as representing the behaviour of a strategy deprived of the initial Opponent signal. For example, when parameterised by w and v, the term $\mathtt{x} : \alpha \rightleftharpoons \alpha \vdash \mathtt{x} : \alpha \rightleftharpoons \alpha$ is mapped to $\overline{v}[w]$; which, in "strategy form", becomes $u(w).\overline{v}[w]$.

Correctness of compilation. Our aim in the rest of the section is to show that $[\![M]\!]^{\tilde{w}}_v$ captures the interactive behaviour of $(\![M]\!)$, and, as a result, that the compilation is computationally adequate with respect to the operational semantics of FPC. For this purpose, we first introduce the notion of *representation*. Let $\sigma : A \rightharpoonup B$ be a V-strategy between non-zero finite non-lazy sums of FPC arenas, satisfying: *all involved V-pointers use isomorphisms which respect the natural ordering underlying the FPC sorting*. Note that all V-pointers involved in strategies resulting from our denotational semantics satisfy this condition. Now let $s = S_0 \ldots S_{n-1} \in \sigma$. Assume pairwise distinct names $u_k : T_k$ $(0 \leq k \leq m_1)$ and $v_k : R_k$ $(0 \leq k \leq m_2)$

$$[\![\Theta,\Gamma\vdash\Gamma_i]\!] \stackrel{\mathrm{def}}{=} \pi_i$$

$$[\![\Theta,\Gamma\vdash \mathtt{inl}(M):\alpha_1+\alpha_2]\!] \stackrel{\mathrm{def}}{=} [\![\Theta,\Gamma\vdash M:\alpha_1]\!];\amalg_1$$

$$[\![\Theta,\Gamma\vdash \mathtt{inr}(M):\alpha_1+\alpha_2]\!] \stackrel{\mathrm{def}}{=} [\![\Theta,\Gamma\vdash M:\alpha_2]\!];\amalg_2$$

$$[\![\Theta,\Gamma\vdash \mathtt{case}\ M\ \mathtt{of}\ \mathtt{inl}(\mathtt{x}_1).M_1\ \mathtt{or}\ \mathtt{inr}(\mathtt{x}_2).M_2:\alpha]\!]$$

$$\stackrel{\mathrm{def}}{=} \Delta;([\![\Theta\vdash\Gamma]\!]\otimes[\![\Theta,\Gamma\vdash M:\alpha_1+\alpha_2]\!]);\delta;\big[[\![\Theta,\langle\Gamma,\mathtt{x}_1:\alpha_1\rangle\vdash M_1:\alpha]\!],[\![\Theta,\langle\Gamma,\mathtt{x}_2:\alpha_2\rangle\vdash M_2:\alpha]\!]\big]$$

$$[\![\Theta,\Gamma\vdash\langle M_1,M_2\rangle:\alpha_1\times\alpha_2]\!] \stackrel{\mathrm{def}}{=} \Delta;([\![\Theta,\Gamma\vdash M_1:\alpha_1]\!]\otimes[\![\Theta\vdash\Gamma]\!]);([\![\Theta\vdash\alpha_1]\!]\otimes[\![\Theta,\Gamma\vdash M_2:\alpha_2]\!])$$

$$[\![\Theta,\Gamma\vdash \mathtt{fst}(M):\alpha_1]\!] \stackrel{\mathrm{def}}{=} [\![\Theta,\Gamma\vdash M:\alpha_1\times\alpha_2]\!];\pi_1$$

$$[\![\Theta,\Gamma\vdash \mathtt{snd}(M):\alpha_2]\!] \stackrel{\mathrm{def}}{=} [\![\Theta,\Gamma\vdash M:\alpha_1\times\alpha_2]\!];\pi_2$$

$$[\![\Theta,\Gamma\vdash \lambda\mathtt{x}.M:\alpha_1\rightharpoonup\alpha_2]\!] \stackrel{\mathrm{def}}{=} \mathrm{p}\lambda([\![\Theta,\langle\Gamma,\mathtt{x}:\alpha_1\rangle\vdash M:\alpha_2]\!])$$

$$[\![\Theta,\Gamma\vdash M(M_1):\alpha_2]\!] \stackrel{\mathrm{def}}{=} \Delta;([\![\Theta,\Gamma\vdash M:\alpha_1\rightharpoonup\alpha_2]\!]\otimes[\![\Theta\vdash\Gamma]\!]);([\![\Theta\vdash\alpha_1\rightharpoonup\alpha_2]\!]\otimes[\![\Theta,\Gamma\vdash M_1:\alpha_1]\!]);\mathsf{ev}$$

$$[\![\Theta,\Gamma\vdash \mathtt{intro}(M):\mu\mathtt{T}.\alpha]\!] \stackrel{\mathrm{def}}{=} [\![\Theta,\Gamma\vdash M:\alpha[\mathtt{T}\mapsto\mu\mathtt{T}.\alpha]]\!]$$

$$[\![\Theta,\Gamma\vdash \mathtt{elim}(M):\alpha[\mathtt{T}\mapsto\mu\mathtt{T}.\alpha]]\!] \stackrel{\mathrm{def}}{=} [\![\Theta,\Gamma\vdash M:\mu\mathtt{T}.\alpha]\!]$$

Figure 2. Interpretation of well-formed expressions

$$(\![\Theta\vdash\tau_1+\tau_2]\!) \stackrel{\mathrm{def}}{=} \langle(\![\Theta\vdash\tau_1]\!),(\![\Theta\vdash\tau_2]\!)\rangle;\oplus$$

$$(\![\Theta,\Gamma\vdash \mathtt{inl}(M):\tau_1+\tau_2]\!) \stackrel{\mathrm{def}}{=} (\![\Theta,\Gamma\vdash M:\tau_1]\!);\amalg'_1$$

$$(\![\Theta,\Gamma\vdash \mathtt{inr}(M):\tau_1+\tau_2]\!) \stackrel{\mathrm{def}}{=} (\![\Theta,\Gamma\vdash M:\tau_2]\!);\amalg'_2$$

$$(\![\Theta,\Gamma\vdash \mathtt{case}\ M\ \mathtt{of}\ \mathtt{inl}(\mathtt{x}_1).M_1\ \mathtt{or}\ \mathtt{inr}(\mathtt{x}_2).M_2:\tau]\!)$$

$$\stackrel{\mathrm{def}}{=} \Delta;((\![\Theta\vdash\Gamma]\!)\otimes[\![\Theta,\Gamma\vdash M:\tau_1+\tau_2]\!]);\delta';[(\![\Theta,\langle\Gamma,\mathtt{x}_1:\tau_1\rangle\vdash M_1:\tau]\!),(\![\Theta,\langle\Gamma,\mathtt{x}_2:\tau_2\rangle\vdash M_2:\tau]\!)]$$

Figure 3. Lazy-sum interpretation

$$[\![\Gamma \vdash \mathtt{x}_i : \alpha_i]\!]_v^{\tilde{x}_1..\tilde{x}_n} \stackrel{\text{def}}{=} \overline{v}[\tilde{x}_i]$$

$$[\![\Gamma \vdash \langle M_1, M_2\rangle : \beta_1 \times \beta_2]\!]_v^{\tilde{w}} \stackrel{\text{def}}{=} ([\![\Gamma \vdash M_1]\!]_{c_1}^{\tilde{w}} \; ; \; c_1(\tilde{x}_1).([\![\Gamma \vdash M_2]\!]_{c_2}^{\tilde{w}} \; ; \; c_2(\tilde{x}_2).\overline{v}[\tilde{x}_1\tilde{x}_2])$$

$$[\![\Gamma \vdash \mathtt{inl}(M) : \alpha + \beta]\!]_v^{\tilde{w}} \stackrel{\text{def}}{=} ([\![\Gamma \vdash M]\!]_c^{\tilde{w}} \; ; \; c(\tilde{x}).\overline{v}(y)\cdot !y(z_1z_2).\overline{z_1}[\tilde{x}])$$

$$[\![\Gamma \vdash \mathtt{inr}(M) : \alpha + \beta]\!]_v^{\tilde{w}} \stackrel{\text{def}}{=} ([\![\Gamma \vdash M]\!]_c^{\tilde{w}} \; ; \; c(\tilde{x}).\overline{v}(y)\cdot !y(z_1z_2).\overline{z_2}[\tilde{x}])$$

$$\begin{aligned}&[\![\Gamma \vdash \mathtt{case}\ M\ \mathtt{of}\ \mathtt{inl}(\mathtt{x}_1).M_1\ \mathtt{or}\ \mathtt{inr}(\mathtt{x}_2).M_2 : \beta]\!]_v^{\tilde{w}}\\ &\quad \stackrel{\text{def}}{=} [\![\Gamma \vdash M]\!]_c^{\tilde{w}} \; ; \; c(y).\overline{y}(z_1z_2) \cdot (z_1(\tilde{x}_1).[\![\langle \Gamma, \mathtt{x}_1 : \beta_1\rangle \vdash M_1]\!]_v^{\tilde{w}\tilde{x}_1} \mid z_2(\tilde{x}_2).[\![\langle \Gamma, \mathtt{x}_2 : \beta_2\rangle \vdash M_2]\!]_v^{\tilde{w}\tilde{x}_2})\end{aligned}$$

$$[\![\Gamma \vdash \lambda x : \alpha.M : \alpha \rightharpoonup \beta]\!]_v^{\tilde{w}} \stackrel{\text{def}}{=} \overline{v}(z)\cdot !z(\tilde{y}c).[\![\langle \Gamma, \mathtt{x} : \alpha\rangle \vdash M]\!]_c^{\tilde{w}\tilde{y}}$$

$$[\![\Gamma \vdash M_1(M_2) : \beta]\!]_v^{\tilde{w}} \stackrel{\text{def}}{=} ([\![\Gamma \vdash M_1]\!]_{c_1}^{\tilde{w}} \; ; \; c_1(y).([\![\Gamma \vdash M_2]\!]_{c_2}^{\tilde{w}} \; ; \; c_2(\tilde{z}).\overline{y}[\tilde{z}v]))$$

$$[\![\Gamma \vdash \mathtt{intro}(M) : \mu\mathtt{T}.\alpha]\!]_v^{\tilde{w}} \stackrel{\text{def}}{=} [\![\Gamma \vdash M : \alpha[\mathtt{T} \mapsto \mu\mathtt{T}.\alpha]]\!]_v^{\tilde{w}}$$

$$[\![\Gamma \vdash \mathtt{elim}(M) : \alpha[\mathtt{T} \mapsto \mu\mathtt{T}.\alpha]]\!]_v^{\tilde{w}} \stackrel{\text{def}}{=} [\![\Gamma \vdash M : \mu\mathtt{T}.\alpha]\!]_v^{\tilde{w}}$$

Figure 4. Direct Compilation

$$\pi_{i}{}_v^u \stackrel{\text{def}}{=} u(\tilde{x}_1..\tilde{x}_n).\overline{v}[\tilde{x}_i] \qquad \amalg'^u_{iv} \stackrel{\text{def}}{=} u(\tilde{x}).\overline{v}(y)\cdot !y(z_1z_2).\overline{z}_i[\tilde{x}]$$

$$\delta'^u_{v_1v_2} \stackrel{\text{def}}{=} u(\tilde{w}e).\overline{e}(c_1c_2).(c_1(\tilde{x}_1).\overline{v_1}[\tilde{w}\tilde{x}_1] \mid (c_2(\tilde{x}_2).\overline{v_2}[\tilde{w}\tilde{x}_2])) \qquad \mathsf{ev}_v^u \stackrel{\text{def}}{=} u(w\tilde{x}).\overline{w}[\tilde{x}v]$$

$$\mathcal{C}[\![\Gamma \vdash \mathtt{x}_i : \alpha_i]\!]_v^u \stackrel{\text{def}}{=} \pi_{i}{}_v^u$$

$$\begin{aligned}&\mathcal{C}[\![\Gamma \vdash \langle M_1, M_2\rangle : \beta_1 \times \beta_2]\!]_v^u\\ &\quad \stackrel{\text{def}}{=} u(\tilde{w}).((\mathcal{C}[\![\Gamma \vdash M_1]\!]_{c_1}^{e_1}; \overline{e_1}[\tilde{w}]) \; ; \; c_1(\tilde{x}_1).((\mathcal{C}[\![\Gamma \vdash M_2]\!]_{c_2}^{e_2}; \overline{e_2}[\tilde{w}]) \; ; \; c_2(\tilde{x}_2).\overline{v}[\tilde{x}_1\tilde{x}_2])\end{aligned}$$

$$\mathcal{C}[\![\Gamma \vdash \mathtt{inl}(M) : \alpha + \beta]\!]_v^u \stackrel{\text{def}}{=} \mathcal{C}[\![\Gamma \vdash M]\!]_c^u ; \amalg'^c_{1v}$$

$$\mathcal{C}[\![\Gamma \vdash \mathtt{inr}(M) : \alpha + \beta]\!]_v^u \stackrel{\text{def}}{=} \mathcal{C}[\![\Gamma \vdash M]\!]_c^u ; \amalg'^c_{2v}$$

$$\begin{aligned}&\mathcal{C}[\![\Gamma \vdash \mathtt{case}\ M\ \mathtt{of}\ \mathtt{inl}(\mathtt{x}_1).M_1\ \mathtt{or}\ \mathtt{inr}(\mathtt{x}_2).M_2 : \beta]\!]_v^u\\ &\quad \stackrel{\text{def}}{=} \mathcal{C}[\![\Gamma \vdash \langle \Gamma | M\rangle]\!]_c^u \; ; \; \delta'^c_{c_1c_2} ; (\mathcal{C}[\![\langle \Gamma, \mathtt{x}_1 : \beta_1\rangle \vdash M_1]\!]_v^{c_1} | \mathcal{C}[\![\langle \Gamma, \mathtt{x}_2 : \beta_2\rangle \vdash M_2]\!]_v^{c_2})\end{aligned}$$

$$\mathcal{C}[\![\Gamma \vdash \lambda x : \alpha.M : \alpha \rightharpoonup \beta]\!]_v^u \stackrel{\text{def}}{=} u(\tilde{w}).\overline{v}(z)\cdot !z(\tilde{y}c).(\mathcal{C}[\![\langle \Gamma, \mathtt{x} : \alpha\rangle \vdash M]\!]_c^e ; \overline{e}[\tilde{w}\tilde{y}])$$

$$\mathcal{C}[\![\Gamma \vdash M_1(M_2) : \beta]\!]_v^u \stackrel{\text{def}}{=} \mathcal{C}[\![\Gamma \vdash \langle M_1, M_2\rangle]\!]_c^u ; \mathsf{ev}_v^c$$

$$\mathcal{C}[\![\Gamma \vdash \mathtt{intro}(M) : \mu\mathtt{T}.\alpha]\!]_v^u \stackrel{\text{def}}{=} \mathcal{C}[\![\Gamma \vdash M : \alpha[\mathtt{T} \mapsto \mu\mathtt{T}.\alpha]]\!]_v^u$$

$$\mathcal{C}[\![\Gamma \vdash \mathtt{elim}(M) : \alpha[\mathtt{T} \mapsto \mu\mathtt{T}.\alpha]]\!]_v^u \stackrel{\text{def}}{=} \mathcal{C}[\![\Gamma \vdash M : \mu\mathtt{T}.\alpha]\!]_v^u$$

Figure 5. Categorical Compilation

where $T_0, .., T_{m_1}$ and $R_0, .., R_{m_2}$ are initial sorts of A and B, respectively. Then an *instantiation of s w.r.t. $\tilde{u}, \tilde{v}$* is a sequence of non-τ labels of length n, say $l_0 l_1 \ldots l_{n-1}$, such that: (1) $sub(l_i) : S_i$ for $0 \leq i \leq n-1$; (2) If $S_i = T_k$ (resp. $S_i = R_k$), $sub(l_i) = u_k$ (resp. $sub(l_i) = v_k$); (3) If S_i is by Opponent (resp. Player) then l_i is input (resp. output); (4) If $i \curvearrowleft j$ and $S_i \mapsto U_1, \ldots, U_m$ such that $S_j = U_k$, then $sub(l_j) = ob(l_i, k)$; (5) If $i \overset{\Psi}{\Rightarrow} j$ is a V-pointer in s and $S_i \mapsto U_1, \ldots, U_k, \ldots, U_m$ as well as $S_j \mapsto U'_1, \ldots, U'_{k'}, \ldots, U'_{m'}$ where $U_k = \mathsf{source}(\Psi)$ and $U'_{k'} = \mathsf{target}(\Psi)$, then $ob(l_i, k) = ob(l_j, k')$ and $ob(l_j, j)$ occurs free in l_j; and (6) object names in the output labels occur free only in the case of (5). Instantiations are considered modulo α-equality. If $s \in \sigma$ and t is an instantiation of s w.r.t. $\tilde{u}$ and $\tilde{v}$, then we say t is a *play of σ instantiated at $\tilde{u}$ and $\tilde{v}$*. The definition of representation follows.

Definition 5.2 (Representation) *For $\sigma : A \multimap B$ as above, a process P* represents *$\sigma : A \multimap B$* w.r.t. $\tilde{u}$ and $\tilde{v}$*, or simply* represents *σ, when there is a relation $\succ$ between the set of FPC-sorted strongly determinate processes and the set of plays of σ instantiated at $\tilde{u}$ and $\tilde{v}$ such that:*

(i) $P \succ \varepsilon$,

(ii) If $P' \succ t$, then (1) for an output l, tl is a play of σ instantiated at $\tilde{u}$ and $\tilde{v}$ iff $P' \overset{l}{\Longrightarrow}$; and, for such l, $P' \overset{l}{\Longrightarrow} P''$ implies $P'' \succ tl$, and (2) for an input l, if tl is a play of σ instantiated at $\tilde{u}$ and $\tilde{v}$ then $P' \overset{l}{\Longrightarrow}$; and, for such l, $P' \overset{l}{\Longrightarrow} P''$ implies $P'' \succ tl$.

The key property of representation follows.

Lemma 5.3 *If P (resp. Q) represents $\sigma : A \multimap B$ w.r.t. $\tilde{a}$ and $\tilde{b}$ (resp. $\tau : B \multimap C$ w.r.t. $\tilde{b}$ and $\tilde{c}$), then $P; Q$ represents $\sigma; \tau : A \multimap C$ w.r.t. $\tilde{a}$ and $\tilde{c}$.*

The proof uses an alternative, equivalent presentation of composition of V-strategies which precisely corresponds to interaction between representing processes.

We now wish to show that, for every closed FPC term M, the Pict code $u(\varepsilon).[\![M]\!]^{\varepsilon}_{v}$ and the term M have the same terminating behaviour. For that purpose we introduce an intermediate compilation whose construction is closer to the inductive definition of our denotational semantics while being bisimilar to the compilation in Figure 4. We list this intermediate compilation in Figure 5. This compilation is parameterised by two names, which belong to the respective initial sorts of the domain and codomain. In the `case` statement, the term $\langle \Gamma | M \rangle$ has the following inductive definition: $\langle \langle \rangle | M \rangle \overset{\text{def}}{=} M$, and $\langle \langle \Gamma, \mathtt{x} : \alpha \rangle | M \rangle \overset{\text{def}}{=} \langle \Gamma | \langle \mathtt{x}, M \rangle \rangle$ where $\langle \mathtt{x}, M \rangle$ denotes the standard pairing.

Proposition 5.4 *For all FPC terms $\Gamma \vdash M : \alpha$,*

(i) $[\![M]\!]^{\tilde{x}}_{v}$ is strongly determinate.

(ii) $u(\tilde{x}).[\![M]\!]^{\tilde{x}}_{v} \approx \mathcal{C}[\![M]\!]^{u}_{v}$.

(iii) $\mathcal{C}[\![M]\!]^{u}_{v}$ represents $(\![M]\!)$ in $\mathbf{G}_i$.

For (i), the case of application requires an analysis of IO-modes [28]. For (ii), we use simple algebraic transformations. For (iii) we proceed by induction using Lemma 5.3.

Let M be a closed FPC term. Then, $M \checkmark$ iff $(\![M]\!) \downarrow$, by Theorem 5.1; that is, by (iii) above, iff $\mathcal{C}[\![M]\!]^{u}_{v} \xrightarrow{u(\varepsilon)} \overset{l}{\Longrightarrow}$ where l is an output with $sub(l) = v$. But this in turn is equivalent to $[\![M]\!]^{\varepsilon}_{v} \downarrow$, by (i) and (ii). Thus we have the following result.

Theorem 5.5 (Computational adequacy) *For all closed FPC terms M, we have that $M \checkmark$ iff $[\![M]\!]^{\varepsilon}_{u} \downarrow$.*

References

[1] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. To appear, 1994.

[2] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for idealized algol with active expressions. *Electronic Notes in Theoretical Computer Science*, 3, 1996.

[3] S. Abramsky and G. McCusker. Call-by-value games. In Proc. of CSL '97, LNCS. Springer-Verlag, 1998. To appear.

[4] G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20:265–322, 1982.

[5] G. Boudol. The π-calculus in direct style. In *POPL '97 Conf.*, pages 228–241, 1997.

[6] S. Brock and G. Ostheimer. Process semantics of graph reduction. In *CONCUR '95 Conf.*, volume 962 of *LNCS*, pages 471–485. Springer-Verlag, 1995.

[7] R. Cartwright, P.-L. Curien, and M. Felleisen. Fully abstract semantics for observably sequential languages. *Information and Computation*, 111(2):297–401, 1994.

[8] A. Edalat and M. Smyth. I-categories as a framework for solving domain equations. *Theoretical Computer Science*, 115:77–106, 1993.

[9] M. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. Distinguished Dissertations in Computer Science. Cambridge University Press, 1996.

[10] M. Fiore and G. Plotkin. An axiomatisation of computationally adequate domain-theoretic models of FPC. In 9th *LICS Conf.*, pages 92–102. Computer Society Press, 1994.

[11] P. Freyd. Algebraically complete categories. In *Category Theory*, volume 1488 of *LNM*, pages 131–156. Springer-Verlag, 1991.

[12] J.-Y. Girard. Geometry of interaction I: Interpretation of system F. In *Logic Colloquium '88*. North Holland, 1989.

[13] C. Gunter. *Semantics of Programming Languages: Structures and Techniques*. The MIT Press, 1992.

[14] K. Honda. Two bisimilarities in ν-calculus. Technical Report 92-002, Department of Computer Science, Keio University, 1992.

[15] K. Honda and N. Yoshida. Game-theoretic analysis of call-by-value computation (full version). Available by ftp from ftp.dcs.ed.ac.uk/export/kohei/cbvfull.ps.gz (An extended abstract appeared in *Proc. of ICALP '97*, volume 1256 of *lncs*, pages 225–236. Springer-Verlag, 1997.).

[16] H. Huwig and A. Poigné. A note on inconsistencies caused by fixpoints in a cartesian closed category. *Theoretical Computer Science*, 73:101–112, 1990.

[17] M. Hyland and L. Ong. On full abstraction for PCF: I, II and III. To appear, 1994.

[18] M. Hyland and L. Ong. Pi-calculus, dialogue games and PCF. In *FPCA '95 Conf.*, pages 96–107, 1995.

[19] K. Larsen and G. Winskel. Using information systems to solve recursive domain equations. *Information and Computation*, 91(2):232–258, 1991.

[20] F. Lawvere. Diagonal arguments and cartesian closed categories. In *Category Theory, Homology Theory and their Applications II*, volume 92 of *LNM*. Springer-Verlag, 1969.

[21] G. Longo and E. Moggi. Cartesian closed categories of enumerations for effective type-structures. In *Symposium on Semantics of Data Types*, volume 173 of *LNCS*. Springer-Verlag, 1984.

[22] G. McCusker. *Games and Full Abstraction for a Functional Metalanguage with Recursive Types*. PhD thesis, University of London, 1996.

[23] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–146, 1992.

[24] R. Milner. Polyadic π-calculus: A tutorial. In *Proceedings of the International Summer School in Marktoberdorf*, 1992.

[25] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.

[26] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. MIT Press, 1990.

[27] E. Moggi. Categories of partial morphisms and the partial lambda-calculus. In *Proceedings of the Workshop on Category Theory and Computer Programming*, volume 240 of *LNCS*, pages 242–251. Springer-Verlag, 1986.

[28] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. In 8^{th} *LICS Conf.*, pages 187–215. Computer Society Press, 1993.

[29] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. To appear in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, The MIT Press, 1998.

[30] G. Plotkin. Denotational semantics with partial functions. Lecture at C.S.L.I. Summer School, 1985.

[31] J. Power and E. Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, 1997.

[32] J. Power and G. Rosolini. Fixpoint operators for domain equations. To appear, 1996.

[33] J. Riecke and A. Sandholm. Relational account of call-by-value sequentiality. In 12^{th} *LICS Conf.*, pages 258–267. Computer Society Press, 1997.

[34] E. Robinson and G. Rosolini. Categories of partial maps. *Information and Computation*, 79:95–130, 1988.

[35] D. Sangiorgi. π-calculus, internal mobility, and agent-passing calculi. *Theoretical Computer Science*, 167(2):235–271, 1996.

[36] M. Smyth and G. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal of Computing*, 11(4):761–783, 1982.

[37] D. Walker. Objects in the pi-calculus. *Information and Computation*, 116(2):253–271, 1995.

[38] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. The MIT Press, 1993.