Pi-Calculus, Dialogue Games and PCF*

J. M. E. Hyland†

DPMMS, University of Cambridge

16 Mill Lane, Cambridge

CB2 1SB ENGLAND

C.-H. L. Ong‡

Oxford University Computing Laboratory

Wolfson Building, Parks Road, Oxford

OX1 3QD ENGLAND

and DISCS, National University of Singapore

Abstract

Game semantics is an unusual denotational semantics in that it captures the intensional (or algorithmic) and dynamical aspects of the computation. This makes it an ideal semantical framework in which to seek to unify analyses of both the qualitative (correctness) as well as the quantitative (efficiency) properties of programming languages. This paper reports work arising from a recent construction of an order (or inequationally) fully abstract model for Scott's functional programming language PCF based on a kind of two-person (Player and Opponent) dialogue game of questions and answers [HO94]. In this model types are interpreted as games and terms as innocent strategies. The fully abstract game model may be said to be canonical for the semantical analysis of sequential functional languages. Unfortunately even for relatively simple PCF-terms, precise description of their denotations as strategies in [HO94] very rapidly becomes unwieldy and opaque. What is needed to remedy the situation is an expressive formal language which lends itself to a succinct and economical representation of innocent strategies. In this paper we give just such a representation in terms of an appropriately sorted polyadic π-calculus, reading input π-actions as Opponent's moves, and output π-actions as Player's moves. This correspondence captures every essential aspect of the dialogue game paradigm so precisely that the π-representation may as well be taken to be the basis for its formal *definition*. Although the π-representation of strategies already gives an encoding of PCF in the π-calculus – indirectly via the fully abstract denotation of PCF-terms as innocent strategies, we define by recursion a new encoding of PCF in the π-calculus directly. The two π-encodings of PCF are weakly bisimilar; if the latter is regarded as a compilation, then the former amounts to a more efficient version with compile-time optimization.

*To appear in *Proceedings of the 7th Annual ACM Conference on Functional Programming Languages and Computer Architecture, La Jolla, California, June 26-28, 1995.*

†martin@pmms.cam.ac.uk.

‡On leave from the National University of Singapore. lo@comlab.ox.ac.uk.

1 PCF: observational preorder and full abstraction

The functional language PCF [Sco93, Plo77] is essentially the simply typed λ-calculus augmented by basic arithmetic, conditionals and fixed-point operators at every type. PCF-types, ranged over by A, A_i, B, *etc.*, are defined by the following grammar in BNF:

$$A \quad ::= \quad \iota \quad | \quad o \quad | \quad A \Rightarrow A.$$

Every PCF-type A has a unique representation of the form $A_1 \Rightarrow (A_2 \Rightarrow \cdots \Rightarrow (A_n \Rightarrow \alpha)\cdots)$ where $n \geqslant 0$ and the program (or ground) type α is either ι or o. By convention we write A as $(A_1, \cdots, A_n, \alpha)$. The operational semantics of PCF (formulated in terms of a Martin-Löf style evaluation relation "$s \Downarrow n$") is presented in the Appendix. Programs are closed terms of program type. A denotational model $\mathcal{M}[\![-]\!]$ of PCF is said to be *order fully abstract* if for any closed terms s and t of the same type,

$$s \precsim t \quad \Longleftrightarrow \quad \mathcal{M}[\![s]\!] \sqsubseteq \mathcal{M}[\![t]\!]$$

where $\precsim$ is the *observational preorder* defined as follows: $s \precsim t$ iff for any (program) context $C[X]$, and for any program value n, if $C[s]$ and $C[t]$ are both programs and if $C[s] \Downarrow n$ then $C[t] \Downarrow n$. The Full Abstraction Problem for PCF is concerned with the search for an abstract, syntax-independent order fully abstract model of PCF. This problem has been studied intensively for over 15 years [Plo77, Mil77, Cur93, Ong95].

The next two sections set the scene: we give a racy overview of the category of arenas and innocent strategies in section 2, and sketch the fully abstract game interpretation of PCF in section 3. The reader is directed to [HO94] for a full account in which proofs of all results cited in the first two sections can be found. We introduce PCF-sorted polyadic π-calculus as a language for describing dialogue games in section 4. Innocent strategies are represented as π-terms in section 5. The direct encoding of PCF in the π-calculus is given in section 6.

2 Arenas and innocent strategies

Dialogue games are played by two players called Player (P) and Opponent (O). There are four kinds of moves:

O's question	[	P's answer	]
P's question	(	O's answer	)

We use the matching pairs of left and right parentheses "[" and "]", and "(" and ")" as meta-variables to represent moves of the respective kinds. The parenthetic notation reflects the condition that P-questions can only be answered by O, and O-questions by P.

Rules of the game

Dialogue games are played according to the following set of rules:

(r1) A play starts by O raising an *opening* question, and ends when it is answered. An opening question may not be raised more than once in a play.

(r2) Moves are made alternately by P and O, each making one move at a time.

(r3) Each play has the effect of tracing out a dialogue of questions and answers observing the following *Principles of Civil Conversation*.

Justification A player is only allowed to make a move provided it is justified: a question is raised only if an occurrence of its unique justifying question is *pending*– already asked but not yet answered; an answer is proffered only if an occurrence of the question expecting it is pending.

Priority or the "last asked first answered" rule: any answer offered must be an answer to the last pending question.

Unlike games which model proofs, there is no question of winning for either player in dialogue games.

A dialogue game is completely specified by its game tree. Paths of the tree represent legitimate plays of the game. We present a game tree in two stages:

- An *arena* defines the moves (questions and answers) of the game, which are nodes of the game tree, and spells out how one move justifies another. These are data specific to individual games but common to plays of the same game.
- The collection of paths of the game tree is then defined to be sequences of moves satisfying the rules (**r1**), (**r2**), (**r3**) and (**r4**) (to be introduced later) of the game. We shall refer to paths of the game tree as *legal positions* of the arena in question.

Definition 2.1 An *arena* is a structure

$$\langle\, \mathsf{Qn}(A), \leqslant_A, \mathsf{Ans}(A), \mathsf{qn}_A \,\rangle$$

where $\langle\, \mathsf{Qn}(A), \leqslant_A \,\rangle$ is an upside down forest[1] of questions; roots of the forest are the $\leqslant_A$-maximal elements. Questions of depths 0, 2, 4, *etc.* are O-questions; those of depths 1, 3, 5, *etc.* are P-questions. Questions of depth 0 are called *opening* moves; they have a special status. We say that a question q *justifies* another, say, q' if q is the unique question immediately above q' in the ordering. $\mathsf{Ans}(A)$ is a set of answers. To each question is associated a set of possible answers via the map $\mathsf{qn}_A : \mathsf{Ans}(A) \longrightarrow \mathsf{Qn}(A)$. For the sake of uniformity, we say that the question $\mathsf{qn}_A(a)$ *justifies* the answer a. Moves of the arena are elements of $\mathsf{Ans}(A) \cup \mathsf{Qn}(A)$.

[1] A forest is a partially-ordered set such that the lower-set of each element is a finite total order.

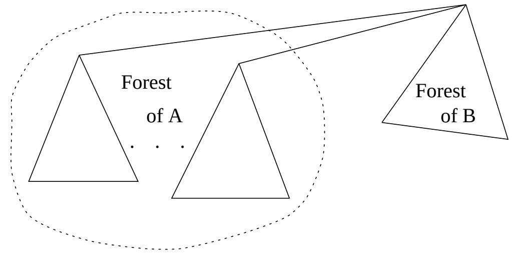

Figure 1: The forest of questions of $A \Rightarrow B$.

At any stage of a play, not every move is necessarily available to the players; however some moves may become available (or "justified" or "enabled") as the play unfolds. The notion of justification is formulated as the partial ordering $\leqslant_A$ over questions. We can now make precise the informally expressed condition (r3): a sequence s of moves, each of which (except the opening move) equipped with a pointer, is said to satisfy (r3) if for every initial subsequence $r \cdot q$ of s where q is a question, the pointer at q points to an occurrence of q' in r such that q' justifies q and the occurrence pointed to is pending. Similarly for every initial subsequence $r \cdot a$ of s where a is an answer, the pointer at a points to an occurrence of q' in r such that the occurrence is pending and a is a possible answer of q'. We say that the question q (similarly the answer a) is *explicitly justified* by (the appropriate occurrence of) q'. A *well-formed* sequence is then a sequence of moves equipped with an auxiliary sequence of pointers, one for each non-opening move, such that conditions (r1), (32) and (r3) are satisfied.

Construction of arenas

We can already define product and function space of arenas. For product we simply take the obvious "disjoint sum" of the arenas A and B as directed graphs. For function space $A \Rightarrow B$, it is simplest to draw a picture as in Figure 1. (In the picture there is only one opening move in B.) The opening moves of $A \Rightarrow B$ are those of B and to the tree "below" each such opening move, we graft onto it a copy of the forest of questions of A.

This is a good place to consider some examples.

Example 2.2 The natural numbers arena, denoted ι (by abuse of notation), is specified by the following data. The (singleton) forest of questions consists of the opening O-question "[". There are as many (P-) answers as there are natural numbers: $]_0,]_1,]_2, \cdots$ which are all possible answers to the only question "[". The Boolean arena o is defined similarly: the (singleton) forest of questions is the opening O-question "[" whose possible P-answers are "$]_{\mathsf{t}}$" and "$]_{\mathsf{f}}$". More generally, for any PCF-type $A = (A_1, \cdots, A_n, \iota)$, the forest of questions of the corresponding PCF-arena A (we shall systematically confuse the PCF-type with its interpretation as an arena) is an inverted finite tree. It is useful to establish a naming convention for questions of a PCF-arena. Take $A = (((\iota, \iota), \iota, \iota), \iota, \iota)$ say. We draw its tree of questions as in Figure 2. The questions are identified by a finite sequence of natural numbers (whose obvious definition we omit) as shown in the Figure.

Views and legal positions

Intuitively a strategy (for P, say) is a method or rule that determines P's move at a position which ends with an O-

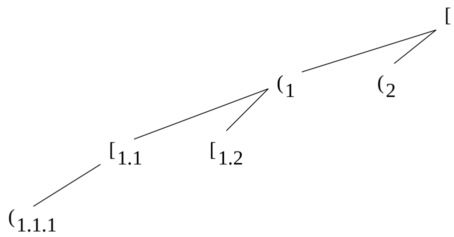

Figure 2: Forest of questions of arena $(((\iota,\iota),\iota,\iota),\iota,\iota)$.

move. A strategy is said to be history-sensitive if the next move is determined as a function of the history of the play. We reject history-sensitive strategies in favour of strategies which operate on the basis of a certain abstraction of the history of play, called view. *Player's view*, or *P-view*, $\ulcorner p \urcorner$ of a well-formed sequence p of moves is defined recursively. Let q range over well-formed sequences of moves, and r over segments of well-formed sequences.

$$\begin{array}{rcll} \ulcorner [\urcorner & \stackrel{\mathrm{def}}{=} & [& \text{if "[" is an opening move} \\ \ulcorner q \cdot (\cdot r \cdot [\urcorner & \stackrel{\mathrm{def}}{=} & \ulcorner q \urcorner \cdot (\cdot [& \text{if "(" explicitly justifies "["} \\ \ulcorner q \cdot) \urcorner & \stackrel{\mathrm{def}}{=} & \ulcorner q \urcorner \cdot) & \\ \ulcorner q \cdot [\cdot r \cdot] \urcorner & \stackrel{\mathrm{def}}{=} & \ulcorner q \urcorner & \text{if "]" explicitly answers "["} \\ \ulcorner q \cdot (\urcorner & \stackrel{\mathrm{def}}{=} & \ulcorner q \urcorner \cdot (. & \end{array}$$

For example the P-view of a well-formed sequence of moves may have the shape

$$[\cdot (\cdot) \cdot (\cdot) \cdot (\cdot [\cdot (\cdot) \cdot (\cdot) \cdot (\cdot [\cdot (\cdot) \cdots .$$

By construction whenever there is a pattern "$(\cdot [$" (respectively "$(\cdot)$") in a P-view, the O-question "[" (respectively O-answer ")") is explicitly justified by the P-question "(". There can be no segments of the form "$[\cdots]$" in a P-view. This may be read as the following: Player ignores answers to questions posed by Opponent. There is a dual definition of the *O-view* of a well-formed sequence, which we omit.

Recall that legal positions of an arena are the paths of the associated game tree.

Definition 2.3 A *legal position* of an arena A is a well-formed sequence t which satisfies condition **(r4)** called *visibility*:

> For any initial subsequence $s \cdot ($ of t, the O-question "[" explicitly justifying the P-question "(" occurs in the P-view of s. Similarly for any initial subsequence $s \cdot [$ of t, the P-question explicitly justifying "[" occurs in the O-view of s.

Example 2.4 The sequence s of moves with the associated pointers in Figure 3 is a legal position of the arena $A = ((\iota,\iota),\iota,\iota)$. The question-moves $[_a$, $(_b$, $[_c$ and $(_d$ are occurrences of the questions $[$, $(_1$, $[_{1.1}$ and $(_2$ of the arena A respectively. For example

$$\begin{array}{rcll} \ulcorner [_a \cdot (_b \cdot [_c \cdot (_d \cdot)_n \urcorner & = & [_a \cdot (_b \cdot [_c \cdot (_d \cdot)_n & \text{and} \\ \ulcorner [_a \cdot (_b \cdot [_c \cdot (_d \cdot)_n \cdot] \cdot)_m \urcorner & = & [_a \cdot (_b \cdot)_m . & \end{array}$$

Innocent strategies

We are interested in deterministic strategies for playing dialogue games. Formally a P-*strategy* σ of an arena A is a non-empty prefix-closed collection of legal positions of A satisfying the following conditions:

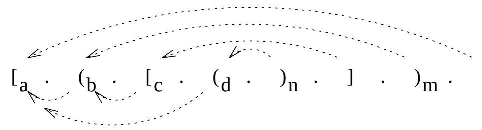

Figure 3: The legal position s.

- **(s1)** *Determinacy* For any $s \in \sigma$ at which Player is to move, if both $s \cdot a$ and $s \cdot b$ are in σ then $a = b$.
- **(s2)** *Contingent completeness* For any $s \in \sigma$ at which Opponent is to move and for any O-move a, if $s \cdot a$ is a legal position then it is in σ.

So a P-strategy of arena A corresponds to a subtree of the game tree that has the following properties: at each node labelled by an O-move, there is at most one offspring, namely, P's unique response (if there is any). Note that our game trees are inverted trees.

Definition 2.5 A P-strategy σ is *innocent* if it is determined by a partial function of a certain kind, mapping P-views (of legal positions at which Player is to move) to P-moves in the following way: for any legal position $s \in \sigma$ at which P is to move, and for any P-move a,

$$s \cdot a \in \sigma \quad \Longleftrightarrow \quad f(\ulcorner s \urcorner) \text{ is defined, and } f(\ulcorner s \urcorner) = a.$$

We call such a function f a *defining* partial function for the innocent strategy σ. Clearly innocent strategies may be specified by such defining functions.

The category $\mathbb{CA}$ is defined by the following data: objects of the category are arenas; maps from arenas A to B are innocent P-strategies of the arena $A \Rightarrow B$. Innocent strategies (as maps) are composed in a way reminiscent of the CSP-style [Hoa85] "parallel composition plus hiding". The terminal object of the category is the empty game. The product and function space arenas defined in the preceding *are* the corresponding categorical constructions. For any arena A, the collection of innocent strategies of A ordered by inclusion forms a dI-domain. A major result of [HO94] is the following:

Theorem 2.6 *The category* $\mathbb{CA}$ *is cartesian closed, and enriched over dI-domains*[2]. $\square$

For example, innocent strategies of the arena ι have the structure of a flat natural numbers CPO. The least element is the everywhere undefined strategy. The innocent strategy denoting n behaves as follows: in response to the opening question "[", P returns the answer "$]_n$", and the play concludes.

3 A fully abstract game model of PCF

Any CPO-enriched cartesian closed category (with natural numbers and Booleans) gives rise to a model of PCF; see *e.g.* [LS86] for the general picture or [HO94, Part I] for an analysis specific to PCF. Given such a category, the interpretations of λ-abstraction, application and the fixed-point operator are completely (and standardly) determined by the categorical structure. So the interpretation $[\![s]\!]$ of any PCF-term $s : A$ as an innocent strategy of the arena A is well-defined, and can be worked out based on the information given so far.

[2] A dI-domain is a consistently complete, prime algebraic CPO that satisfies axiom (I): every compact element dominates only finitely many elements.

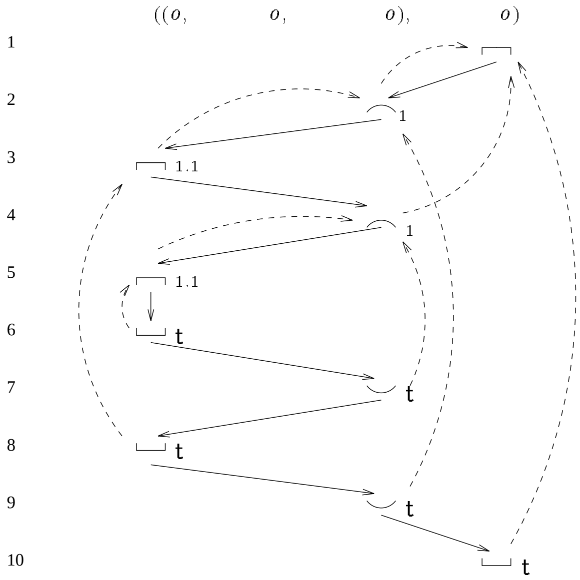

Figure 4: Trace of $F \cdot$ l-or.

Example 3.1 Consider $F = \lambda f.f(f\mathsf{t}\Omega)(f\Omega\mathsf{t})$, a type-2 term of type $((o,o,o),o)$. We show in Figure 4 a legal position of the innocent strategy $[\![F]\!]$ which is the trace of playing F against "left-or" $\mathsf{l\text{-}or} = \lambda xy.\mathsf{cond}x\mathsf{t}(\mathsf{cond}y\mathsf{tf})$.

The main result of [HO94] is the following theorem:

Theorem 3.2 (Hyland-Ong) *The observational quotient* $\hat{\mathbb{CA}}$ *of the category* $\mathbb{CA}$ *gives rise to an order-extensional, order fully abstract, universal model*[3] *of* PCF. □

The proof is based on a Strong Definability Theorem: there is an order-isomorphism between syntax (*finite canonical forms* of **P**, which is essentially PCF extended by definition by (any finite number of) cases, ordered by Ω-matching), and semantics (compact innocent strategies ordered by inclusion). The strong definability result extends to a universality result for $\mathbb{CA}$: modulo observational equivalence, all recursive innocent strategies are PCF-definable. The definition of canonical form is presented in the Appendix.

A problem of representation

Innocent strategies are formally defined as certain collections of legal positions. We observed that it may also be specified by defining functions. Even for relatively simple PCF-terms, precise description of their denotations as innocent strategies in either style very quickly becomes unwieldy and opaque. It would be highly desirable if an expressive calculus which lent itself to an economical description of such uniform strategies were available. It is to this question of representation that we now turn.

4 Polyadic π-calculus

The π-calculus was introduced by Milner *et al.* [MPW92] as an extension of the process algebra CCS in which processes which have changing structures may be expressed naturally. The key features of the π-calculus may be brought out in sharp relief by contrasting and comparing it with the λ-calculus. While the λ-calculus is canonical for calculation with functions, Milner *et al.* regard the π-calculus as "a step towards a canonical treatment of concurrent processes" which gives priority to the concept of *names*. In the π-calculus there are no constants and variables: they are subsumed under names. Names are (used to identify) communication links, and computation is represented purely as the communication of names across links. The linkage between component agents of a system may be changed as a result of a communication (of names) between neighbours. A distinctive feature of the π-calculus is its ostensibly weak communication capabilities: in contrast to the λ-calculus, that which is transmitted or passed in the π-calculus is never an agent, but rather access to an agent, *i.e.*, names. Remarkably, as Milner showed in [Mil90], the "parsimonious" communication constructs of the π-calculus do not make it any less expressive than the λ-calculus.

PCF-sorting

In the following we shall focus on the *polyadic* π-calculus (but refer to it simply as the π-calculus) as introduced in [Mil91]. We presuppose an infinite collection of names which are ranged over by symbols such as $x, x', x_i, f, a, b, u, v$, *etc.* Names are sorted as follows. (We shall assume familiarity with the notion of sort and sorting in [Mil91].) The set $\mathcal{S}$ of subject sorts is defined to be

$$\{\, \mathsf{ans}^\iota, \mathsf{ans}^o, \mathsf{qn}^\iota, \mathsf{qn}^o \,\} \cup \{\, \underline{A} : A \text{ is a PCF-type} \,\}.$$

The sorts qn^ι and qn^o are precisely the sorts of natural numbers questions and Boolean questions respectively; and ans^ι and ans^o are the sorts of natural number answers and Boolean answers respectively. There are denumerably many names of each subject sort. In particular the names of sort ans^ι include $0, 1, 2, etc.$, and those of sort ans^o include t and f. Recall that object sorts are just finite sequences of subject sorts, written as $(s_1, \cdots, s_n)$. A *sorting* is a partial function from subject sorts to object sorts; it associates to each subject sort S the sort of name-vector which each name x of sort S, written $x : S$, can carry or transmit. The PCF-sorting is defined to be the following partial function: for $n \geqslant 1$,

$$\begin{cases} \underline{A} \mapsto (\underline{A_1}, \cdots, \underline{A_n}, \mathsf{qn}^\iota) & \text{if } A = (A_1, \cdots, A_n, \iota) \\ \underline{A} \mapsto (\underline{A_1}, \cdots, \underline{A_n}, \mathsf{qn}^o) & \text{if } A = (A_1, \cdots, A_n, o) \\ \underline{\iota} \mapsto (\mathsf{qn}^\iota) & \underline{o} \mapsto (\mathsf{qn}^o) \\ \mathsf{qn}^\iota \mapsto (\mathsf{ans}^\iota) & \mathsf{qn}^o \mapsto (\mathsf{ans}^o). \end{cases}$$

With reference to PCF-sorting, a name x of sort $\underline{A}$ where $A = (A_1, \cdots, A_n, \iota)$ and $n \geqslant 0$, can carry or transmit name-vectors of sort $(\underline{A_1}, \cdots, \underline{A_n}, \mathsf{qn}^\iota)$. Note that PCF-sorting is undefined on ans^ι and ans^o. This has the effect that a name $n : \mathsf{ans}^\iota$, say, cannot appear as the subject of an action. Such names are in effect (program) values. Thus we shall call the names $0, 1, 2, 3, etc.$ and t and f *value-names.*

PCF-sorted π-terms

The π-calculus consists of a collection of π-*terms* which intuitively stand for processes or *agents*[4]. We shall use $P, Q, R,$

[3] Similar results, based on somewhat different games, have been obtained independently by Abramsky, Jagadeesan and Malacaria [AJM94], and also by Nickau [Nic94]. Subsequently O'Hearn and Riecke [OR94] described a construction of the fully abstract model of PCF by a kind of logical relations.

[4] We shall use the word agent interchangeably with process in the sense of [MPW92], as opposed to [Mil91] which defines agents to be a syntactic subcategory of processes.

etc. to range over the collection $\mathbf{\Pi}$ of π-*terms*. The syntax of PCF-sorted π-terms is defined by the following grammar in BNF:

$$\begin{array}{rcll} P & ::= & \mathbf{0} & \text{zero-term} \\ & | & g.P & \text{guarded term} \\ & | & P \mid P & \text{parallel composition} \\ & | & !P & \text{replication} \\ & | & (x_1, \cdots, x_n)P & \text{restriction} \\ & | & [x = y]P & \text{match} \\ & | & A[x_1, \cdots, x_n] & \text{defined agent (by recursion).} \end{array}$$

Guarded terms $g.P$ are terms that are prefixed by actions. There are four kinds of actions:

$$\begin{array}{rcll} g & ::= & \tau & \text{silent action} \\ & | & x(y_1, \cdots, y_n, a) & \text{input action} \\ & | & \overline{x}\langle\, y_1, \cdots, y_n, a \,\rangle & \text{free output action} \\ & | & \overline{x}(y_1, \cdots, y_n, a) & \text{bound output action} \end{array}$$

where $n \geqslant 0$. In the three non-silent actions, x, of sort $\underline{A}$ where $A = (A_1, \cdots, A_n, \iota)$, is known as the *subject* of the action; $a : \mathsf{qn}^\iota$ is called the *principal object*, and $y_i : \underline{A_i}$ a *subsidiary object* of the action. In all three cases, x is a free name; the names $y_1, \cdots, y_n$ and a are free in the free output action but bound in the other two. The collection of names, free names and bound names of a process P, denoted $\mathrm{N}(P)$, $\mathrm{FN}(P)$ and $\mathrm{BN}(P)$ respectively, are as defined in [MPW92]. The bound output action[5] may be expressed in terms of the restriction operation and the (free) output action: $\overline{x}(y).P = (y)\overline{x}\langle\, y \,\rangle.P$. We follow the notational convention of [Mil90]; notably the guarded term $g.\mathbf{0}$ is often abbreviated to g.

This then (modulo PCF-sorting) is the π-calculus as presented in [MPW92], but extended by polyadic features as introduced in [Mil91] and replication in [Mil90]. Note that we do not need the summation constructor. Details of the definition (*e.g.* the meaning of actions) which have been omitted may be found in these references.

Remark 4.1 We shall have occasion to consider an infinitary version of the π-calculus which involves the parallel composition of an infinite number of processes, all of which "begin" with a match construct. Further the cases associated with the match constructs of these infinite compositions are all mutually exclusive. Typically it has the form

$$[d = 0]P_0 \mid [d = 1]P_1 \mid \cdots \mid [d = n]P_n \mid \cdots$$

which we shall abbreviate to $\prod_{m \in \omega}[d = m]P_m$.

Labelled transition relation and (late) weak bisimulation

The labelled transition relation $\xrightarrow{\alpha}$ on π-terms where α ranges over actions is defined by induction over the rules in Table 6 in the Appendix, as adapted from [MPW92]. Following [Mil90] we separate the structural laws which "govern the neighbourhood relation" among processes, from the rules which specify their interaction. Following a key insight of Berry and Boudols' Chemical Abstract Machine [BB90], the former is defined directly as a congruence relation $\equiv$ on agents as follows.

[5] We are aware of the attractive new-name constructor $\nu x.(-)$ which is equivalent to the bound output action but we prefer the latter since it is better suited to our purpose.

Definition 4.2 We define *structural congruence*, written $\equiv$, to be the least congruence over π-terms satisfying the following:

1. $P \equiv Q$ whenever P is α-convertible to Q
2. $P \mid \mathbf{0} \equiv P$, $P \mid Q \equiv Q \mid P$, $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
3. $!P \equiv P \mid !P$
4. $(x_1, \cdots, x_n)\mathbf{0} \equiv \mathbf{0}$, $(\vec{x})P \equiv (\vec{y})P$ where $\vec{y}$ is a permutation of $\vec{x}$
5. $(x, \vec{y})(P \mid Q) \equiv (\vec{y})(P|(x)Q)$ if x not free in P
6. $(x)g.P \equiv \mathbf{0}$ if x is the subject of the action g.

Note that case (6) above is non-standard. These structural laws are preserved by the labelled reduction relation by definition, as decreed by rule (**struct**). We shall write the relation $\xrightarrow{\tau}$ (transition by silent action) simply as $\rightarrow$, and write $\twoheadrightarrow$ as the reflexive, transitive closure of $\rightarrow$. Finally we shall assume the definition of *weak (late) bisimulation* as given in [MPW92] and [Mil89].

5 Representing innocent strategies

We shall only consider dialogue games in PCF-arenas *i.e.* arenas which are the denotations of PCF-types. For ease of exposition we assume that for each PCF-type A, the names of sort $\underline{A}$ include all PCF-variables of type A.

Game reading of π-actions

Recall that a legal position is a sequence of moves equipped with an auxiliary sequence of pointers satisfying conditions (r1) to (r4). We shall refer to a move together with information concerning justification pointers that emanate from or point to it as an *explicit move*. The representation of dialogue games in the π-calculus is based on a simple association:

$$\begin{array}{rcl} \text{explicit O-moves} & : & \text{input actions} \\ \text{explicit P-moves} & : & \text{output actions.} \end{array}$$

- An *O-question* a of type $A = (A_1, \cdots, A_n, \iota)$ corresponds to the input action

  $$u(f_1, \cdots, f_n, a)$$

  where $u : \underline{A}$, and so, $f_i : \underline{A_i}$ for each i, and $a : \mathsf{qn}^\iota$. The π-action encodes the following information: the (non-opening) O-question a is explicitly justified by the explicit P-question whose corresponding π-action contains u as a subsidiary object. (The subject u marks the source of the justification pointer emanating from a.) Each subsidiary object f_i marks the target of the justification pointer of some P-question (which comes after and) explicitly justified by a.

- A *P-answer* v of an O-question a is represented by a *free* output action of the form

  $$\overline{a}\langle\, v \,\rangle$$

  where the name v is of sort ans^ι or ans^o. The subject a of the action is the O-question to which v is a possible answer. Observe that the occurrence of a in the action marks the source of the justification pointer emanating from the P-answer.

$u(f,x,a). \quad \overline{f}(g,b). \quad g(c). \quad \overline{x}(d). \quad d(v). \quad \overline{c}\langle v \rangle. \quad b(v'). \quad \overline{a}\langle v' \rangle$

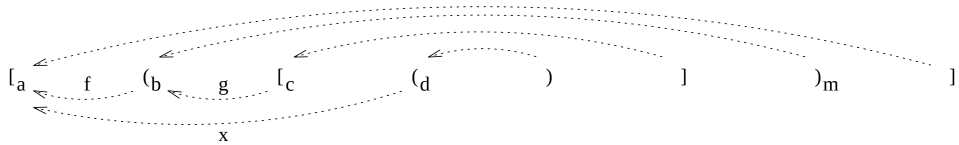

Table 1: π-translation of the legal position s.

- A *P-question* b of type $B = (B_1, \cdots, B_m, \iota)$ is represented by a *bound* output action

$$\overline{f}(g_1, \cdots, g_m, b)$$

where $f : \underline{B}$, and so, $g_i : \underline{B_i}$ and $b : \mathsf{qn}^\iota$. The subject f indicates that the P-question is explicitly justified by the explicit O-question which has f as one of its subsidiary objects.

- An *O-answer* d of the P-question b is represented by an input action of the form

$$b(d)$$

where d is of sort ans^ι or ans^o.

Example 5.1 There is already enough information to give the π-representation of legal positions. To illustrate we consider the legal position s of the arena $A = ((\iota,\iota),\iota,\iota)$ (see Example 2.4). In Table 1, s and its π-representation are shown; the π-action corresponding to each move is directly above the move in the picture. Note that the question "$[_c$", say, is explicitly justified by "$(_b$", and "$)_m$" is an answer specific to the question "$(_b$". In the former case, the justification pointer from "$[_c$" to "$(_b$" is represented by marking the source of the arrow g as the subject of the input action corresponding to "$[_c$", and by marking the target of the arrow as a subsidiary object of the output action corresponding to "$(_b$". The justification pointer of the latter case is similarly represented; note that the target of the arrow is the principal object of the output action corresponding to "$(_b$".

The game reading of π-actions as questions and answers gives rise to a completely adequate (and accurate) representation of legal positions. An important feature of the representation is the unique identification of each question move (of a legal position) by a *new* and *private* name. This name is the principal object of the π-action representing the question move. In the case of s in Example 5.1, observe that the first four questions in order are uniquely identified by names a, b, c and d respectively. This scheme may be extended to represent innocent strategies. We first give an informal explanation of how the capabilities of the π-calculus are employed to give expression to the salient features of innocent strategies.

1. *Generation of new, private links.* Since a question (except the opening question) may be asked repeatedly in an innocent strategy, new names are generated *dynamically* so that each name is unique to a specific instance of the question.

2. *Match.* Just as strategies may respond to moves according to a case analysis, so agents representing them are defined by cases. We use the match constructs in conjunction with parallel composition to implement this facility.

3. *Replication.* Since an innocent strategy may raise a question repeatedly, (the corresponding part of) a responding strategy may need to be played several times during a computation. Such repetitive agents are constructed by using the replicate construct !– of the π-calculus.

4. *Parallel composition.* The composition of strategies (*qua* maps of the category $\mathbb{CA}$) corresponds to "parallel composition plus hiding (with replication)".

An exact representation of strategies

We are now in a position to spell out an exact representation of innocent strategies in (infinitary) π-calculus. An innocent strategy $\sigma : (A_1, \cdots, A_n, \iota)$ corresponds exactly to a (possibly) infinitary *canonical form* $\lambda f_1 \cdots f_n.s_\sigma$; see Theorem A.3. For each innocent strategy σ of type A, and for any name u of sort $\underline{A}$, we define a π-term $[\sigma]_u$ which corresponds exactly to σ in a sense to be made precise.

Definition 5.2 More generally for a strategy σ of type

$$A = (A_1, \cdots, A_k, A_{k+1}, \cdots, A_{k+l}, \iota)$$

we define a translation

$$[\sigma]_{u,f_1,\cdots,f_k},$$

with indicated free names $u, f_1, \cdots, f_k$, of the form

$$u(f_{k+1}, \cdots, f_{k+l}, a).(\sigma)_{f_1,\cdots,f_{k+l},a}.$$

We define the translation $\sigma \mapsto [\sigma]_{u,f_1,\cdots,f_k}$ by recursion on the structure of $s_\sigma \in \mathsf{CF}(f_1 : A_1, \cdots, f_{k+l} : A_{k+l})$ where s_σ is the canonical form corresponding precisely to σ (see Definition A.2). The three cases of the definition

$$\sigma \text{ (equivalently } s_\sigma) \quad \mapsto \quad [\sigma]_{u,f_1,\cdots,f_k}$$

are presented in Table 2.

Note that $[\sigma]_{u,f_1,\cdots,f_k}$ corresponds to

$$f_1 : A_1, \cdots, f_k : A_k \vdash \sigma(f_1, \cdots, f_k) : (A_{k+1}, \cdots, A_{k+l}, \iota).$$

This translation is necessarily into infinitary π-calculus. But compact strategies correspond to finite canonical forms (see [HO94, Part II]), and for these we get a translation into π-calculus proper.

Admissible traces

A trace of an agent is a sequence of actions which the agent can perform in succession. The behaviour of an agent may be described in terms of traces. Formally let P and P' range over π-terms. A *trace* of P is a sequence of actions defined inductively as follows:

For $s_\sigma \in \mathsf{CF}(f_1 : A_1, \cdots, f_n : A_n)$:

(1) s_σ is Ω: define $[\sigma]_{u,f_1,\cdots,f_k}$ to be $u(f_{k+1}, \cdots, f_{k+l}, a).\mathbf{0}$.

(2) s_σ is n for some program value n: define $[\sigma]_{u,f_1,\cdots,f_k}$ to be $u(f_{k+1}, \cdots, f_{k+l}, a).\overline{a}\langle\, n\,\rangle$.

(3) s_σ is $\mathsf{case}\ f_i(\lambda\widetilde{y_1}.t_1)\cdots(\lambda\widetilde{y_r}.t_r)[\prod_{m\in\omega} r_m]$ where $A_i = (C_1, \cdots, C_r, \iota)$; for each $1 \leqslant j \leqslant r$, $C_j = (D_{j1}, \cdots, D_{jp_j}, \iota)$, further for some innocent strategy τ_j of type $(\widetilde{A}, \widetilde{D_j}, \iota)$, $t_j = s_{\tau_j}$ (i.e. $t_j \in \mathsf{CF}(\widetilde{f} : \widetilde{A}, \widetilde{y_j} : \widetilde{D_j})$ is the associated CF of τ_j); for each $m \in \omega$, for some innocent strategy ρ_m of type A with $r_m = s_{\rho_m}$. Define $[\sigma]_{u,f_1,\cdots,f_k}$ to be $u(f_{k+1}, \cdots, f_{k+l}, a).(\sigma)_{f_1,\cdots,f_{k+l},a}$ which is

$$u(f_{k+1}, \cdots, f_{k+l}, a).\overline{f_i}(g_1, \cdots, g_r, b).![\tau_1]_{g_1,f_1,\cdots,f_{k+l}} | \cdots | ![\tau_r]_{g_r,f_1,\cdots,f_{k+l}} | b(d).\prod_m [d = m](\rho_m)_{f_1,\cdots,f_{k+l},a}.$$

Table 2: Definition of the encoding σ (equivalently s_σ) $\mapsto [\sigma]_{u,f_1,\cdots,f_k}$.

- ϵ (the empty sequence) is a trace of the zero agent
- if σ is a trace of P' and if $P \stackrel{\alpha}{\longrightarrow} P'$ then $\alpha \cdot \sigma$ is a trace of P.

Not every trace of a π-term (encoding an innocent strategy) is of interest to us. To begin with we should disregard silent actions which may be thought of as compile-time optimization (about which more anon). Our intention is to capture (traces describing) the behaviour of a agent *only* when it interacts with an environment (of agents) behaving in accord with PCF-computation.

Definition 5.3 An *observable trace* of an agent P is defined to be the subsequence of non-silent actions that is obtained from a trace of P by deleting all τ-actions. An observable trace σ is said to be *admissible* if it satisfies the following conditions:

- σ is a sequence of strictly alternating input and output actions
- whenever (the π-action translate of) an O-answer appears in σ, it answers the (π-action translate of the) last pending P-question.

Theorem 5.4 *The tree of legal positions corresponding to the innocent strategy σ is isomorphic to the tree of admissible traces of $[\sigma]_u$, in a way which is faithful to the correspondence between moves and actions (as explained at the start of section 5).* □

Generalized composition of strategies

Suppose that $\sigma : A_1 \times \cdots \times A_k \Rightarrow A'$ and that $\tau_i : B_1 \times \cdots \times B_l \Rightarrow A_i$ for $1 \leqslant i \leqslant k$. Then we have a generalized composite of strategies

$$\sigma(\tau_1, \cdots, \tau_k) : B_1 \times \cdots \times B_l \Rightarrow A'.$$

This is easily seen to be well-defined by appealing to the cartesian closed structure of the category $\mathbb{CA}$.

Theorem 5.5 *The result of "parallel composition plus hiding" (with replication) on the respective translates of $\sigma, \tau_1, \cdots, \tau_k$, namely*

$$(f_1, \cdots, f_k)[\sigma]_{u,f_1,\cdots,f_k} \mid ![\tau_1]_{f_1,g_1,\cdots,g_l} \mid \cdots \mid ![\tau_k]_{f_k,g_1,\cdots,g_l}$$

is weakly bisimilar to $[\sigma(\tau_1, \cdots, \tau_k)]_{u,g_1,\cdots,g_l}$. □

Now consider a special case with $\sigma : (A_1, \cdots, A_k, \iota)$ and $\tau_i : A_i$. Then

$$(f_1, \cdots, f_k)[\sigma]_{u,f_1,\cdots,f_k} \mid ![\tau_1]_{f_1} \mid \cdots \mid ![\tau_k]_{f_k}$$

is weakly bisimilar to

- $u(a).\overline{a}\langle\, n\,\rangle$ in case $\sigma(\tau_1, \cdots, \tau_k)$ is "output n"
- $u(a).\mathbf{0}$ in case $\sigma(\tau_1, \cdots, \tau_k)$ is "undefined".

We introduce a notation: for π-terms P and P', we write $P \downarrow Q$ to mean that $P \twoheadrightarrow Q$ and $\neg[\exists R.Q \rightarrow R\ \&\ R \not\equiv Q]$. Note that the silent reduction $\longrightarrow$ preserves the structural congruence $\equiv$. We can rewrite the previous special case as the following.

Corollary 5.6 *The composite strategy $\sigma(\tau_1, \cdots, \tau_k)$ is the innocent strategy n in the arena ι if and only if*

$$(u, \widetilde{f})([\sigma]_{u,f_1,\cdots,f_k} \mid ![\tau_1]_{f_1} \mid \cdots \mid ![\tau_k]_{f_k} \mid \overline{u}\langle\, a\,\rangle) \downarrow \overline{a}\langle\, n\,\rangle.$$

□

6 Encoding PCF in the π-calculus

The π-representation of innocent strategies already gives a translation of PCF into the π-calculus: given a PCF-term s, take its denotation as an innocent strategy $[\![\, s\,]\!] = \sigma_s$, then translate it into the π-calculus *i.e.* $[\sigma_s]_u$. We shall call this the *compact* translation. Any translation of PCF into the π-calculus amounts to a kind of compilation, for the π-calculus may be regarded as the description language of a system of name-passing processes whose structures may evolve dynamically. The snag of the compact encoding is that the π-encoding of PCF-terms is given in terms of finite agents only for terms which denote compact innocent strategies. (The advantage as we shall see shortly is that it corresponds to a more efficient compilation.) In this section we give a *direct* encoding of PCF in the π-calculus.

Take a PCF-term s of type $A = (A_1, \cdots, A_n, \iota)$. For any name $u : \underline{A}$, we define by recursion a π-term $\lceil s \rceil^u$ which has the general form

$$u(f_1, \cdots, f_n, a).\boxed{}.\overline{a}\langle\, v\,\rangle.$$

By definition of the PCF-sorting, the names $f_1, \cdots, f_n$ and a have sorts $\underline{A_1}, \cdots, \underline{A_n}$ and qn^ι respectively; and v is of sort ans^ι.

$$\ulcorner n \urcorner^u \stackrel{\text{def}}{=} u(a).\overline{a}\langle\, n \,\rangle$$

$$\ulcorner \mathsf{succ} \urcorner^u \stackrel{\text{def}}{=} u(x,a).\overline{x}(b).b(n).([n=0]\overline{a}\langle\, 1 \,\rangle \mid [n=1]\overline{a}\langle\, 2 \,\rangle \mid \cdots)$$

$$\ulcorner \mathsf{zero?} \urcorner^u \stackrel{\text{def}}{=} u(x,a).\overline{x}(b).b(n).([n=0]\overline{a}\langle\, \mathsf{t} \,\rangle \mid [n=1]\overline{a}\langle\, \mathsf{f} \,\rangle \mid [n=2]\overline{a}\langle\, \mathsf{f} \,\rangle \mid \cdots)$$

$$\ulcorner \mathsf{cond} \urcorner^u \stackrel{\text{def}}{=} u(x,y_1,y_2,a).\overline{x}(b).b(s).([s=\mathsf{t}]\overline{y_1}(c_1).c_1(d).\overline{a}\langle\, d \,\rangle + [s=\mathsf{f}]\overline{y_2}(c_2).c_2(d).\overline{a}\langle\, d \,\rangle)$$

$$\ulcorner x \urcorner^u \stackrel{\text{def}}{=} u(f_1,\cdots,f_n,a).\overline{x}(g_1,\cdots,g_n,b).!\ulcorner f_1 \urcorner^{g_1} \mid \cdots \mid !\ulcorner f_n \urcorner^{g_n} \mid b(d).\overline{a}\langle\, d \,\rangle$$

$$\ulcorner \lambda f_1 : A_1.s \urcorner^u \stackrel{\text{def}}{=} u(f_1,\cdots,f_n,a).(v)(\overline{v}\langle\, f_2,\cdots,f_n,a \,\rangle \mid \ulcorner s \urcorner^v)$$

$$\ulcorner st \urcorner^u \stackrel{\text{def}}{=} u(f_2,\cdots,f_n,a).(v,f_1)(\overline{v}\langle\, f_1,\cdots,f_n,a \,\rangle \mid \ulcorner s \urcorner^v \mid !\ulcorner t \urcorner^{f_1})$$

$$\ulcorner \mathbf{Y} \urcorner^u \stackrel{\text{def}}{=} u(f,f_1,\cdots,f_n,a).\overline{f}(g,g_1,\cdots,g_n,b).!\ulcorner \mathbf{Y} f \urcorner^g \mid !\ulcorner f_1 \urcorner^{g_1} \mid \cdots \mid !\ulcorner f_n \urcorner^{g_n} \mid b(d).\overline{a}\langle\, d \,\rangle.$$

Table 3: Definition of the direct π-encoding $s \mapsto \ulcorner s \urcorner^u$.

$$\ulcorner (\lambda x.x)2 \urcorner^u \quad = \quad u(a).(v,x)(\overline{v}\langle\, x,a \,\rangle \mid v(x',a').\overline{x'}(b).b(d).\overline{a'}\langle\, d \,\rangle \mid !x(a'').\overline{a''}\langle\, 2 \,\rangle).$$

Table 4: The encoding of $(\lambda x.x)2$.

The agent $\ulcorner s \urcorner^u$ interacts with the environment by first communicating via the opening port u. It expects to receive an $(n+1)$-vector of names which will be bound to $f_1,\cdots,f_n$ and a respectively. Note that this is the *only* action $\ulcorner s \urcorner^u$ can engage at the beginning. Suppose at some point after interacting with the environment, the name a has been instantiated to a'. The final action of the agent is to send the value-name v (or the name to which v has been instantiated, if v is a bound name) along the link a'.

Definition 6.1 The recursive definition $s \mapsto \ulcorner s \urcorner^u$ is presented in Table 3. Fix the opening port as u and let $A = (A_1,\cdots,A_n,\iota)$. Some explanatory remarks in connection with the definition are in order. Note that we shall sometimes (*e.g.* in the definition of $\ulcorner \mathbf{Y} \urcorner^u$) represent a variable f occurring in a PCF-term s and a name f appearing in the π-translate $\ulcorner s \urcorner^u$ by the same letter; this should not cause any confusion.

- The agent $\ulcorner n \urcorner^u$ interacts with the environment via the port $u : \underline{\iota}$: upon receipt of a link name, say, a', the agent sends out the value-name n along a' and becomes inactive *i.e.* the zero agent.
- *Successor, predecessor* (omitted) and *test for zero* are "infinite" objects: they are not compact as innocent strategies.
- The π-encoding of a PCF-*variable* $x : A$ is defined by recursion over the type A of x. Note that for each i, f_i and g_i are of sort $\underline{A_i}$. In effect the encoding of x is performed on its full η-expansion.
- In the case of the λ-*abstraction*, the body s has type $(A_2,\cdots,A_n,\iota)$, with each $f_i : \underline{A_i}$.
- We assume that the *application* st has type
$$(A_2,\cdots,A_n,\iota)$$
with $s : A$ and $t : A_1$.
- Finally we take the *fix-point* constant $\mathbf{Y}$ of type $(A \Rightarrow A) \Rightarrow A$; and so, the name f is of sort $\underline{B}$ where $B = (A,A_1,\cdots,A_n,\iota)$, with each $f_i : \underline{A_i}$. Note that this is the only place where an agent is defined by recursion.

  The "f" in the subterm $\ulcorner \mathbf{Y} f \urcorner^g$ occurring in the definition of $\ulcorner \mathbf{Y} \urcorner^u$ is a PCF-variable; the translate $\ulcorner \mathbf{Y} f \urcorner^g$ of the application $\mathbf{Y} f$ is (well-)defined by recursion.

Remark 6.2 The $s \mapsto \ulcorner s \urcorner^u$ encoding takes PCF into infinitary π-calculus. There are two ways by which the encoding can be modified to one which maps into π-calculus proper. The first replaces the constants succ, pred and $\mathsf{zero?}$ by their obvious compact approximants succ_m, pred_m and $\mathsf{zero?}_m$ respectively for each $m \in \omega$. For example $\mathsf{succ}_m n$ evaluates to $n+1$ provided $n \leqslant m$. The other way is to change the way natural numbers (program values) are encoded. There are various possibilities, some of which are discussed in [Mil91]. However such a change would destroy the correspondence between π-actions and game moves, which is the main theme of this work.

The free names of an agent represents its current knowledge of, or linkage to, other agents. An agent that has no free name can only possibly perform silent action. Take a PCF-term $s : A$ with opening port u, it is easy to see by recursion that
$$\mathrm{FN}(\ulcorner s \urcorner^u) \quad = \quad \{\, u \,\} \cup \mathrm{FV}(s).$$
So if s is a program, after performing the action $u(a)$, the agent $\ulcorner s \urcorner^u$ can only perform silent action.

Example 6.3 As an example we encode $t = (\lambda x.x)2 : \iota$. Note that the encoding $\ulcorner t \urcorner^u$, which is shown in Table 4, begins with an input action $u(a)$. Now consider the (silent) reduction of $(u)(\ulcorner t \urcorner^u \mid \overline{u}\langle\, c \,\rangle)$, for some name $c : \mathsf{qn}^\iota$ which serves as a kind of buffer. This easily reduces to
$$(x)(\overline{x}(b).b(d).\overline{c}\langle\, d \,\rangle \mid !x(a'').\overline{a''}\langle\, 2 \,\rangle)$$

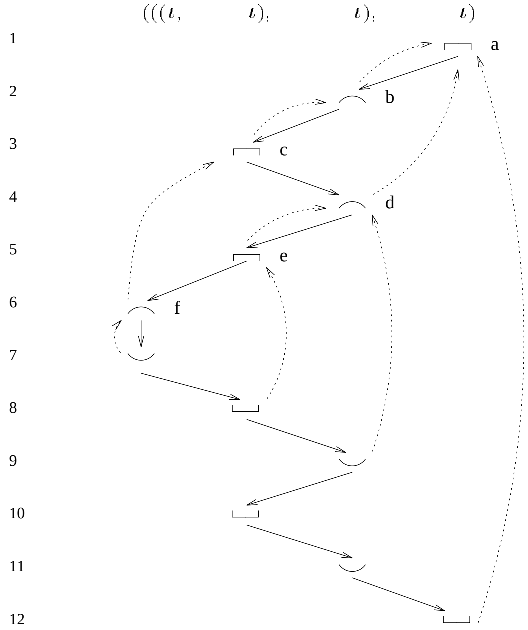

Figure 5: Trace of G.

which is $\equiv$ to

$$(x)(\overline{x}(b).b(d).\overline{c}\langle\, d\,\rangle \mid x(a'').\overline{a''}\langle\, 2\,\rangle \mid !x(a'').\overline{a''}\langle\, 2\,\rangle)$$

which reduces to

$$(x,b)(b(d).\overline{c}\langle\, d\,\rangle \mid \overline{b}\langle\, 2\,\rangle \mid !x(a'').\overline{a''}\langle\, 2\,\rangle)$$

and so to

$$(x)(\overline{c}\langle\, 2\,\rangle \mid !x(a'').\overline{a''}\langle\, 2\,\rangle)$$

which is $\equiv$ to

$$\overline{c}\langle\, 2\,\rangle \mid (x)(!x(a'').\overline{a''}\langle\, 2\,\rangle).$$

By (6) and (2) of Definition 4.2, corresponding to "garbage collection", the last agent is $\equiv$ to $\overline{c}\langle\, 2\,\rangle$. Hence

$$(u)(\lceil(\lambda x.x)2\rceil^u \mid \overline{u}\langle\, c\,\rangle) \quad\downarrow\quad \overline{c}\langle\, 2\,\rangle.$$

Example 6.4 Consider the type-3 functional

$$G \quad = \quad \lambda g.g(\lambda x.g(\lambda y.x)) : (((\iota,\iota),\iota),\iota).$$

A legal position of the innocent strategy denotation of G corresponding to playing G against, say, $\lambda f.f1 : ((\iota,\iota),\iota)$ is shown in in Figure 5. This trace should be compared with the π-term $\lfloor G\rfloor^u$ as shown in Table 5 which is weakly bisimilar to the direct encoding $\lceil G\rceil^u$. (We omit the definition of $h \mapsto \lfloor h\rfloor^u$ for h ranging over head normal forms of PCF.) Observe that the correspondence between moves and actions extends to one between a strategy and its π-translate.

Soundness of the π-encoding

What are the good properties of the translation? We summarize our results in the following. The first concerns the soundness of the encoding relative to the innocent strategy interpretation, or the compact translation.

Theorem 6.5 *Take any closed* PCF*-term $s : A$. Write the innocent strategy interpretation $[\![\, s\,]\!]$ of s as σ_s.*

(i) The tree of admissible traces of $\lceil s\rceil^u$ is isomorphic to the tree of admissible traces of $[\sigma_s]_u$.

(ii) $\lceil s\rceil^u$ is weakly bisimilar to $[\sigma_s]_u$. □

It is a standard result that the following "cut-rule" is valid in PCF: if

$$\begin{array}{rcl} f_1 : A_1, \cdots, f_k : A_k & \vdash & s : A' \quad \text{and} \\ g_1 : B_1, \cdots, g_l : B_l & \vdash & t_i : A_i \end{array}$$

for each $1 \leqslant i \leqslant k$, then

$$g_1 : B_1, \cdots, g_l : B_l \vdash \widetilde{s[t_i/f_i]} : A'.$$

There is a corresponding result in the π-translation:

Proposition 6.6 $\lceil \widetilde{s[t_i/f_i]}\rceil^u$ *is weakly bisimilar to*

$$(f_1, \cdots, f_k)(\lceil s\rceil^u \mid !\lceil t_1\rceil^{f_1} \mid \cdots \mid !\lceil t_k\rceil^{f_k}).$$

□

Since substitution in PCF is modelled by generalized composition, this is a consequence of Theorem 5.5 and Theorem 6.5(ii).

A π-term P is said to be *deterministic* if whenever $P \twoheadrightarrow Q$ and also $Q \rightarrow Q'$ and $Q \rightarrow Q''$ then $Q' \equiv Q''$.

Theorem 6.7 *For any* PCF*-program s,*

(i) $s \Downarrow n \iff (u)(\lceil s\rceil^u \mid \overline{u}\langle\, a\,\rangle) \downarrow \overline{a}\langle\, n\,\rangle$

(ii) the agent $(u)(\lceil s\rceil^u \mid \overline{u}\langle\, a\,\rangle)$ is deterministic. □

Part (i) of the theorem is a corollary of Theorem 6.5 and the computational adequacy of the dialogue game interpretation of PCF. Part (ii) can be proved by a case analysis of the syntactic shape of s.

The PCF evaluation relation $s \Downarrow n$ follows a left-most, weak (*i.e.* no reduction "under a λ") reduction strategy. However our direct encoding of PCF into the π-calculus actually reflects a *head* reduction strategy in PCF. This means that reductions may take place "under a λ" if the head redex happens to be a subterm of a λ-abstraction. Note that for terms of program type, the two reduction strategies coincide.

7 Conclusion and further directions

Our π-encoding of PCF is not the first of its kind: Milner first studied the encoding of the pure, untyped λ-calculus in the π-calculus in [Mil90]. He considered translations of the lazy λ-calculus [AO93] and Plotkin's call-by-value λ-calculus [Plo75] into the π-calculus. Comparing the appropriate dialogue game π-encoding of the lazy λ-calculus with Milner's encoding, it is clear that the two are conceptually quite unrelated, apart from the idea of the π-representation of λ-calculus substitution by "parallel composition with hiding and replication" which is common to both. Also Milner's encoding seems much simpler, and corresponds to a more efficient implementation. The novelty of our encoding lies in the facility it provides for an accurate representation of the innocent strategy denotation of the λ-calculus in terms of the π-calculus.

There are various ways by which the work described in the paper can be extended. Both the dialogue game model and the corresponding π-representation can be modified to give an interpretation for call-by-value and for lazy PCF. The same programme can also be carried out for the untyped λ-calculi; for example, the lazy λ-calculus and the

$$\lfloor G \rfloor^u \stackrel{\text{def}}{=} u(g,a).\overline{g}(h,b).![h(y,c).\overline{g}(k,d).![k(x,e).\overline{y}(f).f(v).\overline{e}\langle v \rangle] | d(v').\overline{c}\langle v' \rangle] | b(v'').\overline{a}\langle v'' \rangle$$

Table 5: The encoding of G optimized.

call-by-value λ-calculus. In a somewhat different direction, we have only just begun to explore ways of representing strategies. Although we have achieved a representation that is in complete accord with the dialogue game paradigm and respects the correspondence between actions and moves, it is still not optimized for capturing the *uniform*[6] or parametric nature of (innocent) strategies which are denotations of λ-terms. Here we have in mind the various kinds of "tit-for-tat" strategies in which P simply copies O-moves from one "component" of the play to the other. Strategies of such nature occur also in various game models of linear logic; see *e.g.* [Bla92, AJ94, HO93]. It would be very useful to have a generic calculus capable of capturing a general class of such schematic strategies. It has been suggested to us that a calculus along the lines of Sangiorgi's higher-order π-calculus [San93] may well fit our requirements, but we have not yet investigated the matter.

In this paper we have presented the polyadic π-calculus as a formal language for representing innocent strategies which have recently been used to construct a fully abstract game model of PCF. Our results show that the representation is so precise that it may as well be taken to be the basis for a formal definition of innocent strategies. We have also given a direct encoding of PCF into the π-calculus which is sound, and agrees with the fully abstract innocent strategy interpretation.

Acknowledgements

This work has received financial support from EPSRC research grant GR/J97366 *Systematic Programming Semantics*, and from EU ESPRIT Basic Research Action Project *Categorical Logic in Computer Science*-II. We thank Pierre-Louis Curien for numerous insightful email and fax messages on game interpretation of PCF, and David Walker for most helpful comments on a draft of this paper.

References

[AJ94] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59:543–574, 1994.

[AJM94] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF (extended abstract). In *Theoretical Aspects of Computer Software: TACS'94, Sendai, Japan*, pages 1–15. Springer-Verlag, 1994. LNCS Vol. 789.

[AO93] S. Abramsky and C.-H. L. Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 105:159–267, 1993.

[BB90] G. Berry and G. Boudol. The Chemical Abstract Machine. In *Conference Record of the 7th Annual* ACM *Symposium on Principles of Programming Languages*, pages 81–94. ACM Press, 1990.

[6] The connotation here is with parametric (as opposed to ad hoc) polymorphism in the sense of Strachey (see e.g. [Rey83]).

[Bla92] A. Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56:183–220, 1992.

[Cur93] P.-L. Curien. *Categorical Combinators, Sequential Algorithms, and Functional Programming.* Birkhäuser, second edition, 1993. Progress in Theoretical Computer Science Series.

[HO93] J. M. E. Hyland and C.-H. L. Ong. Fair games and full completeness for Multiplicative Linear Logic without the mix-rule. ftp-able at `theory.doc.ic.ac.uk` in directory `papers/Ong`, 1993.

[HO94] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. preliminary version, 110 pages, ftp-able at `theory.doc.ic.ac.uk` in directory `papers/Ong`, 1994.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[LS86] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic.* Cambridge Studies in Advanced Mathematics No. 7. Cambridge University Press, 1986.

[Mil77] R. Milner. Fully abstract models of typed lambda-calculus. *Theoretical Computer Science*, 4:1–22, 1977.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[Mil90] R. Milner. Functions as processes. Technical Report 1154, INRIA, 1990.

[Mil91] R. Milner. Polyadic π-calculus: a tutorial. Technical Report ECS-LFCS-91-180, LFCS, University of Edinburgh, 1991. Also in *Logic and Algebra of Specification*, edited by F. L. Bauer, W. Brauer and H. Schwichtenberg, Springer-Verlag, 1993.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100:1–77, 1992.

[Nic94] H. Nickau. Hereditarily sequential functionals. In *Proceedings of the Symposium of Logical Foundations of Computer Science.* Springer-Verlag, 1994. LNCS.

[Ong95] C.-H. L. Ong. Correspondence between operational and denotational semantics. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Vol 4.* Oxford Univ. Press, 1995. To appear.

[OR94] P. W. O'Hearn and J. G. Riecke. Kripke logical relations and PCF. To appear, 1994.

[Plo75] G. D. Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[Plo77] G. D. Plotkin. LCF as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[Rey83] J. C. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing 1983*, pages 513–523, 1983.

[San93] D. Sangiorgi. Expressing mobility in process algebras: First-order and higher-order paradigms. Technical Report CST-99-93, University of Edinburgh, 1993. PhD thesis.

[Sco93] D. S. Scott. A type-theoretical alternative to CUCH, ISWIM and OWHY. *Theoretical Computer Science*, 121:411–440, 1993. In Böhm Festschrift, a special issue of the Journal. The article has been widely circulated as an unpublished manuscript since 1969.

A Appendix

A.1 PCF

Definition A.1 The definition of the language PCF may be found in various places, see *e.g.* [Plo77] or [Ong95]. The operational semantics of PCF may be defined by induction over the following rules: v ranges over values which are constants and λ-abstractions; we read $s \Downarrow v$ as "s reduces to value v"

$$v \Downarrow v \qquad \frac{u[t/x] \Downarrow v}{(\lambda x.u)t \Downarrow v} \qquad \frac{s \Downarrow v \quad vt \Downarrow v'}{st \Downarrow v'}$$

$$\frac{s \Downarrow \mathsf{t} \qquad u \Downarrow v}{\mathsf{cond}^{\beta} s u u' \Downarrow v} \qquad \frac{s \Downarrow \mathsf{f} \qquad u' \Downarrow v}{\mathsf{cond}^{\beta} s u u' \Downarrow v}$$

$$\frac{s \Downarrow n}{\mathsf{succ} s \Downarrow n+1} \qquad \frac{s \Downarrow n+1}{\mathsf{pred} s \Downarrow n} \qquad \frac{s \Downarrow 0}{\mathsf{pred} s \Downarrow 0}$$

$$\frac{s \Downarrow 0}{\mathsf{zero?} s \Downarrow \mathsf{t}} \qquad \frac{s \Downarrow n+1}{\mathsf{zero?} s \Downarrow \mathsf{f}} \qquad \frac{s \mathbf{Y}^{A}(s) \Downarrow v}{\mathbf{Y}^{A}(s) \Downarrow v}$$

A.2 Canonical forms

Definition A.2 We define the infinitary language $\mathbf{P}$ which is PCF augmented by an infinitary definition by cases construct. For ease of presentation we assume that ι is the only program type. The typing rule governing the case construct is:

$$\frac{s : \iota \qquad r_m : \iota \quad m \in \omega}{\mathsf{case}\; s[\prod_{m \in \omega} r_m] : \iota}$$

The operational semantics of $\mathbf{P}$ is defined by the following rule scheme, in addition to those that define the operational semantics of PCF:

$$\frac{s \Downarrow j \qquad r_j \Downarrow n}{\mathsf{case}\; s[\prod_{m \in \omega} r_m] \Downarrow n} \quad j \in \omega$$

For any PCF-types $A_1, \cdots, A_n$ where $n \geqslant 0$, we define the collection

$$\mathsf{CF}(f_1 : A_1, \cdots, f_n : A_n)$$

of *canonical forms* (CFs) of $\mathbf{P}$ with free variables appearing in the list $f_1, \cdots, f_n$ as follows.

- The ground-type Ω and program values $n \geqslant 0$ are in $\mathsf{CF}(\widetilde{f} : \widetilde{A})$.

- For any $\widetilde{f} : \widetilde{A} \equiv f_1 : A_1, \cdots, f_n : A_n$ and for any $1 \leqslant i \leqslant n$ where

$$\begin{array}{rcll} A_i & \equiv & (C_1, \cdots, C_r, \iota) & \text{and where} \\ C_j & \equiv & (D_{j1}, \cdots, D_{jp_j}, \iota) & \text{for each } 1 \leqslant j \leqslant r; \end{array}$$

  if $r_m \in \mathsf{CF}(\widetilde{f} : \widetilde{A})$ for each $m \in \omega$ and if

$$t_j \in \mathsf{CF}(\widetilde{f} : \widetilde{A}, \widetilde{y_j} : \widetilde{D_j})$$

  for each $1 \leqslant j \leqslant r$, then

$$\mathsf{case}\; f_i(\lambda \widetilde{y_1}.t_1) \cdots (\lambda \widetilde{y_r}.t_r)[\prod_{m \in \omega} r_m] \in \mathsf{CF}(\widetilde{f} : \widetilde{A}).$$

Note that a CF is by definition of program type; and it is either Ω, or a number n, or a definition-by-cases construct.

A *finite canonical form* (FCF) is a canonical form in which all occurrences of the case construct have finite branches.

Theorem A.3 *For each* PCF*-type $A = (A_1, \cdots, A_n, \iota)$, there is an order-isomorphism between innocent strategies $\sigma : A$ ordered by inclusion, and canonical forms*

$$s_\sigma \in \mathsf{CF}(f_1 : A_1, \cdots, f_n : A_n)$$

ordered by the Ω-match ordering. □

We omit the definition of the order-ismorphism $\sigma \mapsto s_\sigma : \mathsf{CF}(\widetilde{f} : \widetilde{A})$.

(in)	$x(\vec{y}).P \xrightarrow{x(\vec{y})} P$	**(out)**	$\overline{x}\langle \vec{y} \rangle.P \xrightarrow{\overline{x}\langle \vec{y} \rangle} P$
(par)	$\dfrac{P \xrightarrow{\gamma} P'}{P \mid Q \xrightarrow{\gamma} P' \mid Q}$ if $\mathrm{BN}(\gamma) \cap \mathrm{FN}(Q) = \varnothing$	**(com)**	$\dfrac{P \xrightarrow{\overline{x}\langle \vec{y} \rangle} P' \qquad Q \xrightarrow{x(\vec{z})} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'[\vec{y}/\vec{z}]}$
(close)	$\dfrac{P \xrightarrow{\overline{x}(\vec{w})} P' \qquad Q \xrightarrow{x(\vec{w})} Q'}{P \mid Q \xrightarrow{\tau} (\vec{w})P' \mid Q'}$	**(open)**	$\dfrac{P \xrightarrow{\overline{x}\langle \vec{y} \rangle} P'}{(\vec{y})P \xrightarrow{\overline{x}(\vec{y})} P'}$ if $x \notin \vec{y}$
(res)	$\dfrac{P \xrightarrow{\gamma} P'}{(y)P \xrightarrow{\gamma} (y)P'}$ if $y \notin \mathrm{N}(\gamma)$	**(struct)**	$\dfrac{Q \equiv Q' \quad Q' \xrightarrow{\gamma} P' \quad P' \equiv P}{Q \xrightarrow{\gamma} P}$
(match)	$\dfrac{P \xrightarrow{\gamma} P'}{[x = x].P \xrightarrow{\gamma} P'}$		

Table 6: Definition of the labelled transition relation $\xrightarrow{\gamma}$.