Outline of a Denotational Semantics for the π-Calculus

Ian Stark. Dipartimento di Informatica, Università di Pisa, June 1995.

It appears possible to give a fairly standard denotational semantics for the π-calculus, provided that we work within a functor category indexed by a category $\mathcal{I}$ of finite sets and injections. The construction is driven by the expansion law, and resembles Abramsky's domain of synchronisation trees [1]. It also extends Ingólfsdóttir's work on domain models of value-passing CCS [4].

The relevant predomain equations are the following:

$$\begin{aligned} Pi &\cong 1 + P \circ (Pi_\perp + In + Out) \\ In &\cong N \times (N \to Pi_\perp) \\ Out &\cong N \times (N \times Pi_\perp + (N \multimap Pi_\perp)). \end{aligned}$$

This captures the late semantics of the π-calculus. For Sangiorgi's πI-calculus, where every communication uses a new name [7], we would use $In \cong Out \cong N \times (N \multimap Pi_\perp)$.

To give this equation meaning, we need to choose some suitable O-category in which to solve it; together with a power operation '$P \circ -$', an object N, and a non-standard exponential '$\multimap$'. The intention is that N should be an object of names, and elements of $(N \multimap A)$ functions that, given a fresh name, return an element of A. The exact meanings of 'name' and 'fresh' are made explicit by the choice of category.

The object Pi is then recursively defined by the equations above. These express any π-calculus process as the set of things it may do: a silent action to become another process, an input action, or an output action. Both input and output actions are indexed by their subject: for example, a free output action is represented by an element of $(N \times (N \times Pi_\perp))$, comprising subject, object and succeeding process. Bound output is given by an element of $(N \times (N \multimap Pi_\perp))$, specifying the output port and an agent that takes a fresh name to a process.

As ever, the purpose in having a simple equation and a sophisticated category is that once the construction is made, further work is straightforward. So for example concretions and abstractions are respectively elements of $(N \times Pi_\perp)$ and $(N \to Pi_\perp)$, with the 'bound concretion' $\langle \nu x \rangle P$ an element of $(N \multimap Pi_\perp)$. The same setting could also be used to illustrate processes with sorts, or a π-calculus where transitions carry further annotations to allow analysis of concurrency.

The first section of this note describes an appropriate category, and how it can be used to solve the above equations. We go on to look at properties of the object Pi, and how elements of Pi interpret processes of the π-calculus. We then discuss issues

of correctness, adequacy and full abstraction; the final section outlines possibilities for further work.

1 Choosing a Category

For convenience, we work with predomains rather than domains; selective use of the lift operation then gives fine control over coalesced sum and smash product. This is mildly non-standard, and there is clearly a tradeoff here which needs to be handled carefully. For example, the category we obtain is almost cartesian closed and monoidal closed, but for countability reasons it only has exponentials of lifted objects. Pitts discusses the issues involved in recursively defining predomains in his paper on coinduction [6].

We use a functor category, to capture the fact that every π-calculus process is defined over some finite set of free names, which may change as the process performs input and output actions. As base category we use $\mathcal{B}$, ω-bifinite predomains (bottomless SFP domains) and continuous maps, because we want both function spaces and a concrete description of the Plotkin powerdomain. For index $\mathcal{I}$ we take the category of finite sets of names, and injections between them.

The object of processes $Pi_\perp$ is now a functor $\mathcal{I} \to \mathcal{B}$: so if s is a set of names, then $Pi_\perp s$ is a domain of processes with free names in s; similarly if $f : s \to s'$ is a morphism in $\mathcal{I}$, then $Pi_\perp f : Pi_\perp s \to Pi_\perp s'$ is a relabelling operator.

This functor category $\mathcal{C} = \mathcal{B}^{\mathcal{I}}$ inherits an O-structure from $\mathcal{B}$; in particular there is an evident category $\mathcal{C}^{\mathrm{e}}$ of embeddings, which we can use to solve domain equations. Finite limits can be taken pointwise in $\mathcal{C}$, and lifted function space is formed with the usual

$$(A \to B_\perp)s \;=\; Hom_{\mathcal{C}}(\mathcal{I}(s,-) \times A, B_\perp) \qquad A, B \in \mathcal{C},$$

taking advantage of the order structure on the homsets of $\mathcal{C}$. Lift and disjoint union are also available, taken pointwise. A certain amount of care is required in all of these: often one considers operations only on the collection of predomains, and functorality on the corresponding category is useful but not essential; here we do need an action on morphisms, because every object in $\mathcal{C}$ is a functor with a morphism part in $\mathcal{B}$.

The object of names N in the category $\mathcal{C}$ is the inclusion taking a set of names s to the discrete predomain s. In fact, N is the only exponent that we shall need, and its (unlifted) function space has the following object part:

$$(N \to A)s \;\cong\; (As)^s \times A(s+1)\ .$$

Here naturality constraints in the functor category capture the fact that a function on names need only specify its behaviour at all existing names, and one new name.

Pitts describes an adaptation P of the Plotkin (convex) powerdomain to bifinite predomains, with the property that $P(D)_\perp = P^\natural(D_\perp)$ for $D \in \mathcal{B}$ [6, §5]. Applying this

by composition, we have an operation '$P \circ -$' on $\mathcal{C}$, whose action on objects is given by:

$$\mathcal{I} \xrightarrow{A} \mathcal{B} \quad \longmapsto \quad \mathcal{I} \xrightarrow{A} \mathcal{B} \xrightarrow{P} \mathcal{B}\,.$$

Abramsky's adjoining of the empty set [1, Def. 3.4] can then be represented as $1+P\circ(-)$, given that in $\mathcal{B}$ we have $P^0(D_\perp) \cong (1 + P(D))_\perp$.

With models of names in $Set^{\mathcal{I}}$, it is convenient to restrict attention to functors that preserve pullbacks, with the consequence that all the image morphisms in the base category are injections. This restriction corresponds to quite reasonable requirements on the behaviour of elements as the collection of available names is enlarged; it also happens to give exactly those functors that comprise the sheaf topos for the atomic topology on $\mathcal{I}^{\mathrm{op}}$.

It is not immediately clear whether a similar constraint can be laid on the category $\mathcal{C}$. The difficulty lies in identifying properties that are preserved by the power operation P; in particular, its action on morphisms does not in general preserve monos. Nevertheless the whole of $\mathcal{B}^{\mathcal{I}}$ does seem rather generous as a working category, and some restriction on the functors comprising $\mathcal{C}$ might be sensible.

An alternative solution, suggested by Moggi, is to choose $\mathcal{C}$ first, define the power operation P abstractly, and then only use its universal properties. Of course if the subcategory $\mathcal{C}$ is reasonable, we shall probably end up with the same object $Pi_\perp$, but in a more well-behaved setting.

For any element $a \in As$ we define the *support* of a to be the least $s' \subseteq s$ such that a is the image of some $a' \in As'$. This is intended to be the set of names which a actually needs. For example, the element $\perp \in A_\perp s$ always has empty support; while the element $x \in Ns$ for $x \in s$ has support $\{x\} \subseteq s$.

There is a symmetric monoidal structure $(1, \otimes)$ on $\mathcal{C}$, which captures a notion of 'non-interference' between elements. It arises from the disjoint sum '+' in the index category $\mathcal{I}$, by a standard construction due to B. J. Day [3]. The tensor is given as the following coend:

$$A \otimes B \;=\; Coend_{i,j\in\mathcal{I}}(\mathcal{I}(i+j,-) \times Ai \times Bj) \qquad A, B \in \mathcal{C}.$$

More explicitly, elements of $(A \otimes B)s$ are pairs from As and Bs with disjoint support:

$$\begin{aligned}(A \otimes B)s \;=\; \{\langle a, b\rangle \in (A \times B)s \mid \exists f : i \to s,\, g : j \to s,\, a' \in Ai,\, b' \in Bj\,.\\ a = Afa' \;\;\&\;\; b = Bgb' \;\;\&\;\; Im\, f \cap Im\, g = \emptyset\}.\end{aligned}$$

These can be regarded as pairs of mutually non-interfering elements. The identity is the constant functor to the one-element predomain 1.

There is a corresponding lifted function space '$\multimap$', with

$$(A \multimap B_\perp)s \;=\; Hom_{\mathcal{C}}(\mathcal{I}(s,-) \otimes A, B_\perp) \qquad A, B \in \mathcal{C} \quad s \in \mathcal{I}.$$

$$\begin{array}{rcl} N \multimap 1 & \cong & 1 \\ N \multimap (A \times B) & \cong & (N \multimap A) \times (N \multimap B) \\ N \multimap (A + B) & \cong & (N \multimap A) + (N \multimap B) \\ N \multimap (P \circ A) & \cong & P \circ (N \multimap A) \\ N \multimap A_\perp & \cong & (N \multimap A)_\perp \\ N \multimap N & \cong & N + 1 \end{array}$$

Figure 1: Some isomorphisms involving the object of names N.

Now any $f \in \mathcal{I}(s, s')$ has support $Im\, f$, so elements of $(\mathcal{I}(s, s') \otimes As')$ comprise pairs $\langle f, a \rangle$ where a has support disjoint from $Im\, f$. Informally, an element of $(A \multimap B_\perp)s$ is a function defined at all later stages, but only on arguments that cannot interfere with it.

There are natural maps:

$$\begin{array}{rcll} (A \otimes B) & \longrightarrow & (A \times B) & \text{inclusion} \\ (A \to B_\perp) & \longrightarrow & (A \multimap B_\perp) & \text{surjection.} \end{array}$$

If $A : \mathcal{I} \to \mathcal{B}$ is a constant functor then both of these are isomorphisms.

An important example of this exponential is that the operation $(- \otimes N)$ has a right adjoint $(N \multimap -)$, where N is the object of names given above. As we shall see, this is legitimate even without lifting. An element of $(N \multimap A)s$ is a function that takes any fresh name $x \notin s$ to an element of $A(s + \{x\})$. As is usual with exponentials in a functor category, naturality places some restrictions here: all fresh names must be treated equally, and hence we have the isomorphism

$$(N \multimap A)(\_) \;\cong\; A(\_ + 1).$$

This immediately gives the useful isomorphims of Figure 1. The first two of these are usual for exponentiation; the others are perhaps a little surprising.

Taking these definitions of N, '$P \circ -$' and '$\multimap$', we can use the O-categorical structure of $\mathcal{C}$ to construct a recursively defined object Pi as an initial solution to the equations given earlier. The method is standard, though it is significant that Pi itself only appears lifted on the right hand side of the equations. Indeed, because it also only appears positively, we could solve the equations in $Set^{\mathcal{I}}$; but would then be unable to interpret recursion or replication of processes.

The symmetric monoidal (lift-) closed structure on $\mathcal{C}$ is central to its role as a model for systems involving names. A similar construction can be made in $Set^{\mathcal{I}}$, and the same

applies for other choices of 'possible world' categories used by various authors to model state in Algol-like languages [5].

As a further example in $Set^{\mathcal{I}}$, the monad for dynamic name creation from my thesis [8, Chap. 3, Sect. 5] can be expressed as an $\mathcal{I}$-shaped colimit:

$$TA \;\cong\; \underset{\rightarrow\mathcal{I}}{Lim}(N^{\otimes(-)} \multimap A)\,,$$

where $N^{\otimes s} = N \otimes \cdots \otimes N$ has $|s|$ copies, for s an object of $\mathcal{I}$. Loosely, this says that a computation of type A is a function taking zero or more fresh names and returning a value of type A that may use them.

2 Properties of Pi

In order to interpret π-calculus processes in the category $\mathcal{C}$, we need to give some operations on the object $Pi_\perp$. While we could do this entirely in terms of its universal properties as a powerdomain, things may be a little more clear if we look first at the structure of the domain $Pi_\perp s$, and the two predomains $In\, s$ and $Out\, s$, for some $s \in \mathcal{I}$.

The standard union '$\uplus$' and singleton '$\{\!|-|\!\}$' maps for powerdomains give rise in an obvious way to morphisms into $Pi_\perp$, making allowance for the empty set:

$$\begin{array}{rcll} \emptyset : 1 & \longrightarrow & Pi_\perp & \text{left inclusion} \\ \uplus : Pi_\perp \times Pi_\perp & \longrightarrow & Pi_\perp & \\ \{\!|-|\!\} : (Pi_\perp + In + Out)_\perp & \longrightarrow & Pi_\perp\,. & \end{array}$$

Figure 2 uses these to present definitions of three sets $\mathcal{K}(Pi_\perp s)$, $\mathcal{K}(In\, s)$ and $\mathcal{K}(Out\, s)$, which consist of all the finite elements for the corresponding predomains. Here the elements $\{\!|\perp|\!\}$ and $\emptyset$ of $\mathcal{K}(Pi_\perp s)$ represent the undetermined process and the inactive process 0 respectively. Element $\{\!|\perp|\!\}$ is the least in $Pi_\perp s$, while $\emptyset$ is incomparable save for $\{\!|\perp|\!\} \sqsubseteq \emptyset$.

Input and bound output both involve N-exponentials, for which we use the following isomorphisms:

$$\begin{array}{rcccl} \lambda y.p & \in & \mathcal{K}((N \to Pi_\perp)s) & \cong & \mathcal{K}(Pi_\perp s)^s \times \mathcal{K}(Pi_\perp(s+1)) \\ \lambda \underline{y}.p & \in & \mathcal{K}((N \multimap Pi_\perp)s) & \cong & \mathcal{K}(Pi_\perp(s+1))\,. \end{array}$$

An element written $\lambda y.p$ is then a function from any name, old or new, to a finite element of $Pi_\perp$. Here p is not an element itself, but rather gives elements $p[z/y] \in \mathcal{K}(Pi_\perp(s \cup \{z\}))$ for any name z, with all fresh names treated equally. Element $\lambda \underline{y}.p$ is less general: the underbar on y is to indicate that it is certain to be instantiated to a fresh name, so we can essentially take $p \in \mathcal{K}(Pi_\perp(s + \{y\}))$ for some $y \notin s$. This representation of finite elements recalls the surjection noted earlier:

$$\begin{array}{rcl} (N \to Pi_\perp) & \longrightarrow & (N \multimap Pi_\perp) \\ \lambda y.p & \longmapsto & \lambda \underline{y}.p\,. \end{array}$$

General processes $\mathcal{K}(Pi_\perp s)$:

$$\{\!|\perp|\!\}, \emptyset \in \mathcal{K}(Pi_\perp s)$$

$$\begin{array}{rcll} p \in \mathcal{K}(Pi_\perp s) & \Rightarrow & \{\!|tau(p)|\!\} \in \mathcal{K}(Pi_\perp s) & \text{silent action} \\ i \in \mathcal{K}(In\, s) & \Rightarrow & \{\!|in(i)|\!\} \in \mathcal{K}(Pi_\perp s) & \text{input action} \\ o \in \mathcal{K}(Out\, s) & \Rightarrow & \{\!|out(o)|\!\} \in \mathcal{K}(Pi_\perp s) & \text{output action} \end{array}$$

$$p, q \in \mathcal{K}(Pi_\perp s) \Rightarrow p \uplus q \in \mathcal{K}(Pi_\perp s) \quad \text{choice}$$

Input processes $\mathcal{K}(In\, s)$:

$$x \in s,\ \lambda y.p \in \mathcal{K}((N \to Pi_\perp)s) \Rightarrow (x, \lambda y.p) \in \mathcal{K}(In\, s)$$

Output processes $\mathcal{K}(Out\, s)$:

$$x, y \in s,\ p \in \mathcal{K}(Pi_\perp s) \Rightarrow (x, y, p) \in \mathcal{K}(In\, s) \quad \text{free output}$$

$$x \in s,\ \lambda \underline{y}.p \in \mathcal{K}((N \multimap Pi_\perp)s) \Rightarrow (x, \lambda \underline{y}.p) \in \mathcal{K}(In\, s) \quad \text{bound output}$$

Figure 2: Finite elements of $Pi_\perp s$, $In\, s$ and $Out\, s$.

Rather than work explicitly with elements of $Pi_\perp s$, it would be preferable to talk of morphisms to $Pi_\perp$ in $\mathcal{C}$, using some internal language. Indeed the use of $\lambda y.p$ and $\lambda \underline{y}.p$ above anticipates just such a language. Unfortunately, there are well-known difficulties in doing this for a category with distinct cartesian and monoidal closed structures. On the other hand, the morphisms that we want to describe are very limited (only into $Pi_\perp$ and $(N \multimap Pi_\perp)$, and only from '$\times$' and '$\otimes$' products of N), so it might yet be possible.

We can now define two particular morphisms of $\mathcal{C}$:

$$\begin{aligned} new : \quad & (N \multimap Pi_\perp) \quad \longrightarrow \quad Pi_\perp \\ par : \quad & Pi_\perp \times Pi_\perp \quad \longrightarrow \quad Pi_\perp \, . \end{aligned}$$

The morphism *new* is used to interpret name restriction. It takes an agent expecting a new name to a process, essentially by providing a fresh private name. The morphism *par* interprets parallel composition as interleaving.

Both of these maps could be (recursively) defined by unfolding the definition of $Pi_\perp$ and using various properties of N, '$\multimap$' and '$P \circ -$', including the isomorphisms of Figure 1. Such an abstract description would then allow other choices of category, by highlighting the general structure required. However, without a suitable internal language, this manipulation of morphisms is rather unilluminating; so we give instead an explicit description, specific to the functor category $\mathcal{C}$.

As *new* and *par* are just natural transformations, it is enough to give their action at each stage $s \in \mathcal{I}$:

$$\begin{aligned} new_s : \quad & (N \multimap Pi_\perp)s \quad \longrightarrow \quad Pi_\perp s \\ par_s : \quad & Pi_\perp s \times Pi_\perp s \quad \longrightarrow \quad Pi_\perp s, \end{aligned}$$

where we recall that $(N \multimap Pi_\perp)s$ is isomorphic to $Pi_\perp(s{+}1)$. Both maps are continuous, so we need only specify their value at finite elements. Certain linearities with respect to '$\uplus$' also help.

Figure 3 defines the map new_s in this way. The various clauses for input and output reflect the fact that under name restriction:

- any action on the restricted channel becomes unavailable;
- free output may become bound output.

Note that we use $s \cup \{z\}$ for set union and $s + \{z\}$ for disjoint union.

The interaction between the cartesian and monoidal closed structures of $\mathcal{C}$ plays a significant part in these definitions. For example, in the clause for input, the exchange of $\lambda \underline{x}.$ and $\lambda z.$ is justified by the canonical map

$$(N \multimap (N \to Pi_\perp)) \;\longrightarrow\; (N \to (N \multimap Pi_\perp)) \, ,$$

$$
\begin{array}{rcl}
new_s(\lambda \underline{x}.\emptyset) & = & \emptyset \\
new_s(\lambda \underline{x}.\{\!|\bot|\!\}) & = & \{\!|\bot|\!\} \\
new_s(\lambda \underline{x}.(p \uplus q)) & = & new_s(\lambda \underline{x}.p) \uplus new_s(\lambda \underline{x}.q) \\
\\
new_s(\lambda \underline{x}.\{\!|tau(p)|\!\}) & = & \{\!|tau(new_s(\lambda \underline{x}.p))|\!\} \\
\\
new_s(\lambda \underline{x}.\{\!|in(y, \lambda z.p)|\!\}) & = & \left\{ \begin{array}{ll} \emptyset & x = y \\ \{\!|in(y, \lambda z.new_{(s \cup \{z\})}(\lambda \underline{x}.p))|\!\} & x \neq y \\ \quad \text{with } \alpha\text{-conversion to ensure } & x \neq z \end{array} \right. \\
\\
new_s(\lambda \underline{x}.\{\!|out(y, z, p)|\!\}) & = & \left\{ \begin{array}{ll} \emptyset & x = y \\ \{\!|out(y, \lambda \underline{x}.p)|\!\} & x = z \neq y \\ \{\!|out(y, z, new_s(\lambda \underline{x}.p))|\!\} & \text{otherwise} \end{array} \right. \\
\\
new_s(\lambda \underline{x}.\{\!|out(y, \lambda \underline{z}.p)|\!\}) & = & \left\{ \begin{array}{ll} \emptyset & x = y \\ \{\!|out(y, \lambda \underline{z}.new_{(s+\{z\})}(\lambda \underline{x}.p))|\!\} & x \neq y \\ \quad \text{with } \alpha\text{-conversion to ensure } & x \neq z \end{array} \right.
\end{array}
$$

Figure 3: Definition of the map new_s for name restriction

and this in turn is derived from the inclusion

$$N \otimes (N \times A) \longrightarrow N \times (N \otimes A)$$

which holds for any A.

For parallel composition, we break down the map par_s with two auxiliary maps:

$$par_s(p,q) \;=\; lpar_s(p,q) \uplus lpar_s(q,p) \uplus lcom_s(p,q) \uplus lcom_s(q,p)\ .$$

Here $lpar_s(p,q)$ is prioritised parallel composition: first p does a transition, then q interleaves with its residue. Process $lcom_s(p,q)$ allows p to send to q, and interleaves their residues. The two maps are defined by mutual recursion, according to the equations in Figure 4; again, by continuity and some degree of linearity, the significant clauses are those for singletons of finite elements.

3 Interpreting the π-calculus

With the machinery of the previous section, it is straightforward to construct an interpretation of the π-calculus in the category $\mathcal{C}$. For any process P with free names in s, we describe a morphism

$$[\![P]\!]_s : N^s \longrightarrow Pi_\perp\ ,$$

where N^s is the object of s-environments. Again this would be done best abstractly, using an internal language for morphisms of $\mathcal{C}$. In the absence of such a language, we instead describe first certain elements of $Pi_\perp s$, and then build the morphisms from these.

For a π-calculus process P with free names in s, Figure 5 defines an element

$$(\!|P|\!)_s \in Pi_\perp s$$

by recursion on the structure of P. Notice that the definition of replication uses the order structure of the domain, with $(\!|!P|\!)_s$ a least fixed point of par_s. If we replace replication with mutually recursive process definitions, these too would be solved as least fixed points in $Pi_\perp s$.

To raise $(\!|P|\!)_s$ from an element to a morphism, suppose that $s' \in \mathcal{I}$ and $\rho \in N^s s'$; that is, $\rho : s \to s'$ is some substitution on names. Then the natural transformation

$$[\![P]\!]_s : N^s \longrightarrow Pi_\perp$$

is defined by its action at stages such as s', with

$$[\![P]\!]_s s' \rho \;=\; (\!|\, P[\rho(x)/x \mid x \in s] \,|\!)_{s'}\ .$$

The most significant difference between the two forms is that interpreting a process as an element of $Pi_\perp s$ assumes that all names are distinct, whereas the morphism $(N^s \to Pi_\perp)$ in $\mathcal{C}$ includes behaviour under all possible name identifications. Thus $(\!|-|\!)_s$ is the 'ground' notion, and $[\![-]\!]_s$ the more general one.

Parallel composition:

$$par_s(p,q) \;=\; lpar_s(p,q) \uplus lpar_s(q,p) \uplus lcom_s(p,q) \uplus lcom_s(q,p)\ .$$

Prioritised version $lpar_s$:

$$\begin{array}{rcl}
lpar_s(\{\!|\bot|\!\},q) & = & \{\!|\bot|\!\} \\
lpar_s(\emptyset,q) & = & \emptyset \\
lpar_s(p \uplus p',q) & = & lpar_s(p,q) \uplus lpar_s(p',q) \\
lpar_s(\{\!|tau(p)|\!\},q) & = & \{\!|tau(par_s(p,q))|\!\} \\
lpar_s(\{\!|in(x,\lambda y.p)|\!\},q) & = & \{\!|in(x,\lambda y.par_{s\cup\{y\}}(p,q))|\!\} \\
lpar_s(\{\!|out(x,y,p)|\!\},q) & = & \{\!|out(x,y,par_s(p,q))|\!\} \\
lpar_s(\{\!|out(x,\lambda \underline{y}.p)|\!\},q) & = & \{\!|out(x,\lambda \underline{y}.par_s(p,q))|\!\}
\end{array}$$

Single communication $lcom_s$:

$$\begin{array}{rcl}
lcom_s(\{\!|\bot|\!\},q) & = & \{\!|\bot|\!\} \\
lcom_s(p,\{\!|\bot|\!\}) & = & \{\!|\bot|\!\} \\
lcom_s(\emptyset,q) & = & \emptyset \\
lcom_s(p,\emptyset) & = & \emptyset
\end{array}$$

$$\left\{\begin{array}{rcll}
lcom_s(\,\{\!|out(x,y,p)|\!\},\{\!|in(x,\lambda z.q)|\!\}\,) & = & \{\!|tau(par_s(p,q[y/z]))|\!\} & \\
lcom_s(\,\{\!|out(x,\lambda \underline{y}.p)|\!\},\{\!|in(x,\lambda z.q)|\!\}\,) & = & \{\!|tau(new_s(\lambda \underline{y}.par_{s+\{y\}}(p,q[y/z])))|\!\} & \\
lcom_s(\{\!|a|\!\},\{\!|b|\!\}) & = & \emptyset & \text{otherwise}
\end{array}\right.$$

$$\begin{array}{rcl}
lcom_s(p \uplus p',q) & = & lcom_s(p,q) \uplus lcom_s(p',q) \\
lcom_s(p,q \uplus q') & = & lcom_s(p,q) \uplus lcom_s(p,q')
\end{array}$$

Figure 4: Definition of the map par_s for parallel composition.

$$
\begin{array}{rcl}
(\!|\bar{x}y.P|\!)_s & = & \{\!|\, out(x, y, (\!|P|\!)_s)\,|\!\} \\
(\!|x(y).P|\!)_s & = & \{\!|\, in(x, \lambda z.(\!|P[z/y]|\!)_{s\cup\{z\}})\,|\!\} \\
(\!|\nu x\, P|\!)_s & = & new_s(\lambda \underline{x}.(\!|P|\!)_{s+\{x\}}) \\
(\!|[x = y]P|\!)_s & = & \begin{cases} \emptyset & x \neq y \\ (\!|P|\!)_s & x = y \end{cases} \\
 & & \\
(\!|0|\!)_s & = & \emptyset \\
(\!|P + Q|\!)_s & = & (\!|P|\!)_s \uplus (\!|Q|\!)_s \\
(\!|P \mid Q|\!)_s & = & par_s((\!|P|\!)_s, (\!|Q|\!)_s) \\
(\!|!P|\!)_s & = & \mu p.par_s((\!|P|\!)_s, p)
\end{array}
$$

Figure 5: Interpretation of π-calculus processes as elements of $Pi_\perp s$.

4 Soundness and Other Matters

This section describes soundness, adequacy and full abstraction results for the denotational semantics of the π-calculus described above, together with an indication of proof methods. We look in turn at how the interpretation respects transitions, bisimilarity, equivalence and equivalence up to distinctions.

The operational semantics of a π-calculus process P is given by its transitions; these are soundly interpreted in $\mathcal{C}$, with results such as

$$P \xrightarrow{\bar{x}y} Q \quad \Longrightarrow \quad \{\!|\, out(x, y, (\!|Q|\!)_s)\,|\!\} \in (\!|P|\!)_s \ .$$

The use of the element form $(\!|-|\!)_s$ reflects the fact that transition in the π-calculus is a 'ground' notion. Proof is by rule induction: the translation respects every rule of the operational semantics.

The converse to soundness is that the model is *adequate* with respect to transitions: this involves properties like

$$\{\!|\, tau(q)\,|\!\} \in (\!|P|\!)_s \quad \Longrightarrow \quad \exists Q \ . \ P \xrightarrow{\tau} Q \ \&\ (\!|Q|\!)_s = q \ .$$

To prove this would require a formal approximation relation similar to that used in adequacy proofs for the λ-calculus.

The remaining adequacy results all follow in a fairly straightforward way from the fact that we have a sound and adequate model of transitions in the π-calculus. Adequacy with respect to strong bisimilarity (strong ground equivalence) is that

$$(\!|P|\!)_s = (\!|Q|\!)_s \quad \Longrightarrow \quad P \mathrel{\dot{\sim}} Q \ .$$

Again as this is a ground notion an interpretation in terms of elements is appropriate. Later we shall see how to express the same assertion with morphisms alone.

Given that we work with predomains, we might expect that the order structure on $Pi_\perp$ would correspond to similarity of π-calculus processes. However this is not the case: the ordering on $Pi_\perp$ only relates to under-specified behaviour, appropriate for defining $[\![!P]\!]_s$ as a limit; while similarity in the π-calculus occurs when the (fully specified) behaviour of one process is included in that of another.

As an aside, we could add a divergent process Ω to the π-calculus, with a matching assertion $P{\uparrow}$ 'process P may diverge'. The approximation relation in $Pi_\perp$ then corresponds to 'partial bisimilarity' of π-calculus processes, defined as

$$P \mathrel{\dot{\lesssim}} Q \quad \Longleftrightarrow \quad P \text{ simulates } Q \\ \qquad\qquad\qquad\qquad \& \text{ if } P{\downarrow} \text{ then } Q{\downarrow} \text{ and } Q \text{ simulates } P$$

where $P{\downarrow}$ 'process P converges' is the negation of $P{\uparrow}$.

Moving to non-ground notions, adequacy for strong equivalence is expressed by

$$[\![P]\!]_s = [\![Q]\!]_s \quad \Longrightarrow \quad P \sim Q \ .$$

As noted earlier, the morphisms $[\![-]\!]_s : N^s \to Pi_\perp$ include behaviour under all possible name identifications. Hence this is the right form to express the fact that $P \sim Q$ if and only if they are strongly bisimilar under all substitutions.

The object N^s here has as elements all possible substitutions on the names in s, including those where different names become identified. By comparison, the object

$$N^{\otimes s} \ = \ N \otimes \cdots \otimes N \qquad \text{with } |s| \text{ copies of } N$$

comprises all s-environments which keep the names distinct. There is an inclusion $\otimes \hookrightarrow \times$ through which we can define another interpretation of processes as morphisms:

$$[\![-]\!]_{\neq s} : N \otimes \cdots \otimes N \hookrightarrow N \times \cdots \times N \xrightarrow{[\![-]\!]_s} Pi_\perp$$

with $|s|$ copies of N in both cases. This $[\![P]\!]_{\neq s}$ is the ground interpretation of P, keeping all names distinct. Indeed the object $N^{\otimes s}$ is isomorphic to $\mathcal{I}(s, -)$, so by the Yoneda Lemma,

$$Hom_{\mathcal{C}}(N^{\otimes s}, Pi_\perp) \ \cong \ Pi_\perp s$$

and the morphism $[\![-]\!]_{\neq s}$ corresponds to the element $(\!|-|\!)_s$ across this isomorphism. We can use this to reinterpret adequacy for bisimilarity in terms of morphisms:

$$[\![P]\!]_{\neq s} = [\![Q]\!]_{\neq s} \quad \Longrightarrow \quad P \mathrel{\dot{\sim}} Q \ .$$

Between bisimilarity and equivalence lies a spectrum of relations indexed by *distinctions*. These express permanent inequalities of names; specifically, a distinction D is a symmetric,

irreflexive relation on a set of names s. Corresponding to any distinction there is an object N^D: if s' is some other set of names, then the elements of $N^D s'$ are substitutions $\rho : s \to s'$ that respect the distinction D. For example, the objects $N^{\otimes s}$ and N^s correspond to the total and empty distinctions respectively. As a more complicated example, consider the distinction

$$\{\, w,\, x \neq y \neq z \,\} \quad \text{on the set} \quad \{w, x, y, z\}.$$

This corresponds to the object

$$N_w \times (N_y \otimes (N_x \times N_z))$$

where for clarity the copies of N are indexed by the name they interpret. Not all distinctions can be so simply represented; however, there is always a canonical inclusion $N^D \hookrightarrow N^s$. One of the difficulties in devising an internal language for $\mathcal{C}$, to describe morphisms into $Pi_\perp$, is the need for objects like N^D as contexts ('typotheses').

We can define morphisms that interpret the behaviour of a process under some distinction D:

$$[\![-]\!]_D : N^D \hookrightarrow N^s \xrightarrow{[\![-]\!]_s} Pi_\perp \ .$$

If two processes are bisimilar under all name substitutions that respect some distinction D, then they are said to be *D-equivalent*. Adequacy for D-equivalence is expressed by

$$[\![P]\!]_D = [\![Q]\!]_D \quad \Longrightarrow \quad P \sim_D Q \ .$$

This specializes to the assertions of adequacy for bisimilarity and equivalence given above.

Full abstraction results are the converse of all of these, as follows:

$$\begin{array}{rcl} P \sim_D Q & \Longrightarrow & [\![P]\!]_D \ = \ [\![Q]\!]_D \\ P \mathrel{\dot\sim} Q & \Longrightarrow & [\![P]\!]_{\neq s} = \ [\![Q]\!]_{\neq s} \\ P \sim Q & \Longrightarrow & [\![P]\!]_s \ = \ [\![Q]\!]_s \ . \end{array}$$

Proof is straightforward, provided that we have an 'internal full abstraction' result for the domain $Pi_\perp s$. This says that elements of the domain are completely determined by the transitions they interpret; see Pitts [6, §5] for a general coinductive proof method.

Figure 6 summarises how the various operational notions of process behaviour are interpreted by the categorical model.

	Operation	Denotation
Transitions:	$P \xrightarrow{\bar{x}y} Q$	$\{\!\|out(x,y,(\![Q]\!)_s)\|\!\} \in (\![P]\!)_s \quad etc.$
Bisimilarity:	$P \dot{\sim} Q$	$(\![P]\!)_s = (\![Q]\!)_s \ \in Pi_\perp s$
		$[\![P]\!]_{\neq s} = [\![Q]\!]_{\neq s} \ : \ N^{\otimes s} \longrightarrow Pi_\perp$
Equivalence:	$P \sim Q$	$[\![P]\!]_s = [\![Q]\!]_s \ : \ N^s \longrightarrow Pi_\perp$
D-equivalence:	$P \sim_D Q$	$[\![P]\!]_D = [\![Q]\!]_D \ : \ N^D \longrightarrow Pi_\perp$

Figure 6: Behaviour of π-calculus processes: operational notions and their interpretation in the categorical model.

5 Further Work

This might include any of the following.

- Expand and complete proofs of correctness, adequacy, full abstraction.
- Early as well as late semantics for input actions.
- Open bisimilarity, weak bisimilarity, ...
- Internal language for $\mathcal{C}$. With graphs as typotheses to express non-interference?
- What logic corresponds to this model?
- More abstract construction of the model and interpretation.
- Connection to Moggi's work on models for CCS-like languages.
- What happens when this method of indexing by $\mathcal{I}$ is applied to other categorical models of concurrency [2, 9] ?
- Higher-order π-calculus. This might just be a matter of replacing N by $Pi_\perp$ as the object of input and output clauses, but it can hardly be that simple.
- What does the induced model of the lambda-calculus look like?

Other possibilities will no doubt suggest themselves to the reader.

References

[1] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92(2):161–218, June 1991.

[2] A. Corradini, G. Ferrari, and U. Montanari. Transition systems with algebraic structure as models of computations. In *Semantics of Systems of Concurrent Processes*, Lecture Notes in Computer Science 469, pages 185–222. Springer-Verlag, 1990.

[3] B. J. Day. On closed categories of functors. In *Reports of the Midwest Category Seminar*, Lecture Notes in Mathematics 137, pages 1–38. Springer-Verlag, 1970.

[4] A. Ingólfsdóttir. A semantic theory for value-passing processes, late approach. Part I: A denotational model and its complete axiomatization. BRICS Report RS-95-3, Department of Computer Science, University of Aarhus, January 1995.

[5] P. W. O'Hearn, A. J. Power, M. Takeyama, and R. D. Tennent. Syntactic control of interference revisited. In *Mathematical Foundations of Programming Semantics: Proceedings of the 11th International Conference*, Electronic Notes in Theoretical Computer Science 1. Elsevier, 1995.

[6] A. M. Pitts. A co-induction principle for recursively defined domains. *Theoretical Computer Science*, 124:195–219, 1994.

[7] D. Sangiorgi. π-calculus, internal mobility, and agent-passing calculi. Draft Paper, November 1994.

[8] I. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, December 1994. Also published as Technical Report 363, University of Cambridge Computer Laboratory.

[9] G. Winskel and M. Nielsen. Models for concurrency. BRICS Report RS-94-12, Department of Computer Science, University of Aarhus, May 1994. To appear as a chapter in the *Handbook of Logic and the Foundations of Computer Science*, vol. 4, Oxford University Press.