Appears in the Proceedings of LICS '96

A Fully Abstract Domain Model for the π-Calculus

Ian Stark
BRICS*
Department of Computer Science
University of Aarhus
Denmark
Ian.Stark@brics.dk

Abstract

Abramsky's domain equation for bisimulation and the author's categorical models for names combine to give a domain-theoretic model for the π-calculus. This is set in a functor category which provides a syntax-free interpretation of fresh names, privacy, visibility and non-interference between processes. The model is fully abstract for strong late bisimilarity and equivalence (bisimilarity under all name substitutions).

1. Introduction

The π-calculus is a calculus of mobile processes [11]. It aims to provide a prototypical language for communicating systems where configurations and topology change dynamically. The notion of a *name* is central to the π-calculus: names are labels for communication channels, and names are the only values sent over them. This turns out to be a remarkably powerful system, which supports encodings of the λ-calculus and higher-order process calculi.

Most work on the π-calculus to date has been operational, looking directly at the actions a process may perform. This paper initiates a denotational approach, presenting a novel domain-theoretic model of name-passing processes, based on functor categories. The model is fully abstract and captures exactly the operational notions of transitions, strong late bisimilarity and equivalence. This improves models such as Hennessy's for higher-order processes [6] which implements dynamic rather than static binding, and a much coarser equivalence. However the most significant contribution is that the categorical constructions used provide a syntax-free handling of names, visibility, privacy, and interference between processes.

The method is quite straightforward; we solve the following predomain equations:

$$Pi \;\cong\; 1 + P(Pi_\perp + In + Out) \tag{1}$$

$$In \;\cong\; N \times (N \to Pi_\perp) \tag{2}$$

$$Out \;\cong\; N \times (N \times Pi_\perp + N \multimap Pi_\perp)\,. \tag{3}$$

These express a process as the set of things it may do: deadlock, silent transitions, input or output. Intuitively, N is an object of names, P is a power operation, and $(N \multimap Pi_\perp)$ is a non-standard exponential that takes a fresh name to a process that uses it. The catch is that we must first find a category $\mathcal{C}$ in which these make sense; this occupies Section 2 of this paper.

In Section 3 we solve the equations and look at the *object of processes* $Pi_\perp$. Section 4 interprets π-calculus processes in the category $\mathcal{C}$ and shows the model to be fully abstract. Section 5 outlines further possibilities for this denotational approach to name-passing calculi.

This work draws on two distinct lines of existing research. Hennessy and Plotkin [7] showed how the convex powerdomain models concurrency, and Abramsky [1] used this to present a domain equation for strong bisimulation in SCCS. More recently, Pitts and the author [15, 20, 21] have given a general categorical interpretation for dynamically generated names. The results presented here are the fruitful combination of both approaches.

Ingólfsdóttir has given domain models of value-passing CCS [8]; curiously, this is more arduous than the π-calculus, because of the infinity of possible communications. Independently of the work presented here, Fiore, Moggi and Sangiorgi [5] have developed a model of the π-calculus which starts from the same domain equations, but then uses rather different methods to carry out the interpretation; while Abramsky and Meredith have considered using non-wellfounded sets rather than domains.

*Basic Research in Computer Science, a centre of the Danish National Research Foundation. The author was previously supported by the European Community under the HCM network 'EuroFoCS' and ESPRIT basic research action 'CLICS-II'.

2. Choosing a Category $\mathcal{C}$

Any π-calculus process is defined over some finite set of free names, which may change as the process performs input and output. To capture this, we build our model in a functor category $\mathcal{B}^{\mathcal{I}}$, where $\mathcal{I}$ is the category of finite sets of names, and injections between them. The object of processes $Pi_\perp$ is then 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.

For base category $\mathcal{B}$ we take the category of bifinite predomains and continuous maps described by Pitts in [14, §5]. This has both function spaces and a concrete description of the Plotkin powerdomain; while selective use of the lift operation gives precise control over coalesced sum and smash product. Working with pointed domains would in fact give the same object $Pi_\perp$, but the intermediate constructions are unwieldy.

Within the functor category $\mathcal{B}^{\mathcal{I}}$ we take the full subcategory $\mathcal{C}$ of functors that are pullback-preserving, and also bifinite *as functors*; that is, they are bilimits of ω-chains of functors into the category of finite posets. This 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 [19]. Finite limits, colimits and lifting can all be taken pointwise in $\mathcal{C}$, and there is a lifted function space, with object part

$$(A \to B_\perp)s = Hom_{\mathcal{C}}\big(A(s + \_), B_\perp(s + \_)\big) \ . \qquad (4)$$

This is the standard construction for exponentials in a functor category: the requirement that A and B be bifinite as functors ensures that the hom-object is a bifinite predomain, and the disjoint union '$+$' in $\mathcal{I}$ leads to a particularly simple presentation. Informally, a function from A to $B_\perp$ over the names s must take account of arguments defined over any larger set s', with naturality ensuring that fresh names are treated alike.

Pitts gives an adaptation P of the Plotkin (convex) powerdomain $P^\natural$ to bifinite predomains, with the property that $P(D)_\perp = P^\natural(D_\perp)$ for $D \in \mathcal{B}$ [14, §5]. This gives a pointwise power operation on $\mathcal{C}$, and following Abramsky [1, Def. 3.4] we adjoin the empty set as $1 + P(-)$.

2.1. $\mathcal{C}$ is Symmetric Monoidal Closed

Naturally arising from its presentation as a functor category, $\mathcal{C}$ has additional structure that captures the distinctive behaviour of a system based on names.

For any element $a \in As$ 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 the set of names which a actually needs; the requirement that A preserve pullbacks ensures that it is unique. There is then a symmetric monoidal structure $(1, \otimes)$ on $\mathcal{C}$, with elements of $(A \otimes B)s$ being 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. In fact this tensor comes from the disjoint union '$+$' in the index category $\mathcal{I}$ by a general construction of Day [4].

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

$$(A \multimap B_\perp)s = Hom_{\mathcal{C}}\big(A, B_\perp(s + \_)\big) \qquad (5)$$

Informally, an element of $(A \multimap B_\perp)s$ is a function defined at all larger name sets s', but only on arguments not using the names in s. Thus for example the evaluation map

$$(A \multimap B_\perp) \otimes A \ \longrightarrow \ B_\perp$$

expresses the situation where a function and its argument may not share names.

The monoidal and cartesian closed structures are related, with a natural inclusion

$$(A \otimes B) \ \hookrightarrow \ (A \times B) \qquad (6)$$

and a corresponding surjection

$$(A \to B_\perp) \ \twoheadrightarrow \ (A \multimap B_\perp) \ . \qquad (7)$$

2.2. The Object of Names N

The object of names N in the category $\mathcal{C}$ is the inclusion taking a set of names s to the discrete predomain s. As it happens, N is the only exponent that we need, and there are particularly simple forms for the function spaces:

$$\begin{aligned}(N \to A)s \ &\cong \ (As)^s \times A(s+1) \\ (N \multimap A)s \ &\cong \ A(s+1) \ .\end{aligned}$$

These arise from the naturality constraints of equations (4) and (5): an element of $(N \to A)$ is determined by its behaviour at all existing names, and one new name; while elements of $(N \multimap A)$ are elements of A using one fresh name.

Given an object of names, we can now step back a little from the detailed structure of $\mathcal{C}$. For a set of names s, we write N^s for the $|s|$-fold product $N \times \cdots \times N$; this is the object of *s-environments*, with elements of $N^s s'$ being substitutions $\rho : s \to s'$. Similarly, $N^{\otimes s}$ is the monoidal product $N \otimes \cdots \otimes N$; the object of *distinct* s-environments, where no two names are identified.

3. The Object $Pi_\perp$

The definitions of N, $P(-)$ and '$-\!\circ$' above give meaning to the equations (1)–(3); using the O-categorical structure of $\mathcal{C}$ we can solve them to obtain an object Pi. Lifted, this gives the *object of processes* $Pi_\perp$. The method is standard [19], 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 adapt the equations to give a solution in $Set^{\mathcal{I}}$; but would then be unable to interpret recursion or replication of processes.

Formally, operations on $Pi_\perp$ are defined by unfolding the equations and using various properties of N, $P(-)$ and '$-\!\circ$'. This abstract approach ensures that we stay within $\mathcal{C}$, and also allows for other choices of category by highlighting the general structure required. Such anonymous manipulation of morphisms is sound but unilluminating, so we give here a more explicit presentation, using the detailed structure of $Pi_\perp$ and $\mathcal{C}$.

3.1. Finite Elements

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:

$$\emptyset : 1 \longrightarrow Pi_\perp \qquad \uplus : Pi_\perp \times Pi_\perp \longrightarrow Pi_\perp$$
$$\{\!|-|\!\} : (Pi_\perp + In + Out)_\perp \longrightarrow Pi_\perp \ .$$

Figure 1 uses these to present 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. 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 representation of finite elements:

$$\lambda y.p \in \mathcal{K}((N \to Pi_\perp)s)$$
$$\lambda \underline{y}.p \in \mathcal{K}((N \multimap Pi_\perp)s) \ .$$

An element written $\lambda y.p$ is 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 indicates 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 any $y \notin s$. This notation recalls the surjection between the two kinds of exponential, noted in equation (7):

$$(N \to Pi_\perp) \longrightarrow\!\!\!\!\!\rightarrow (N \multimap Pi_\perp)$$
$$\lambda y.p \longmapsto \lambda \underline{y}.p \ .$$

3.2. Maps new and par

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

$$new : (N \multimap Pi_\perp) \longrightarrow Pi_\perp$$
$$par : Pi_\perp \times Pi_\perp \longrightarrow Pi_\perp \ .$$

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. In both cases existing knowledge of operational behaviour, such as the expansion law [11], is used to guide the denotational construction.

Although both maps are best defined abstractly, we can present them concretely by giving their action at finite elements, over some $s \in \mathcal{I}$. For example, new_s acts on free output according to:

$$new_s(\lambda x.\{\!|out(y,z,p)|\!\}) = \begin{cases} \emptyset & x = y \\ \{\!|out(y, \lambda x.p)|\!\} & x = z \neq y \\ \{\!|out(y,z,new_s(\lambda x.p))|\!\} & \text{otherwise.} \end{cases}$$

This expresses the fact that actions on the restricted channel become unavailable, and free output may become bound output.

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 most significant clause is that for communication, where an input action may match either free output:

$$lcom_s(\ \{\!|out(x,y,p)|\!\},\ \{\!|in(x,\lambda z.q)|\!\}\ ) = \{\!|tau(par_s(p, q[y/z]))|\!\},$$

bound output:

$$lcom_s(\ \{\!|out(x,\lambda \underline{y}.p)|\!\},\ \{\!|in(x,\lambda z.q)|\!\}\ ) = \{\!|tau(new_s(\lambda \underline{y}.par_{s+\{y\}}(p, q[y/z])))|\!\},$$

or nothing at all:

$$lcom_s(\{\!|a|\!\}, \{\!|b|\!\}) = \emptyset$$

when neither of the previous two apply.

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

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

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

Input and output agents:

$$\begin{array}{rcll} x \in s,\, \lambda y.p \in \mathcal{K}((N \to Pi_\perp)s) & \Rightarrow & (x, \lambda y.p) \in \mathcal{K}(In\, s) & \text{input} \\ x, y \in s,\, p \in \mathcal{K}(Pi_\perp s) & \Rightarrow & (x, y, p) \in \mathcal{K}(Out\, s) & \text{free output} \\ x \in s,\, \lambda \underline{y}.p \in \mathcal{K}((N \multimap Pi_\perp)s) & \Rightarrow & (x, \lambda \underline{y}.p) \in \mathcal{K}(Out\, s) & \text{bound output} \end{array}$$

Figure 1. Finite elements of $Pi_\perp s$, $In\, s$ and $Out\, s$.

4. Interpretation of the π-Calculus

With the machinery of the previous section, we construct an interpretation of the π-calculus in the category $\mathcal{C}$: for any process P with free names in s, there is a morphism $[\![P]\!]_s : N^s \longrightarrow Pi_\perp$ from the object of s-environments. Again the formal definition is by manipulation of morphisms in $\mathcal{C}$, without reference to the underlying functors; while for clarity we present here a more concrete description.

We interpret the full π-calculus, with unguarded sums and both match and mismatch operators, $[x = y]P$ and $[x \neq y]P$. However, none of these operations are essential; as we do not rely on any equational transformations of process terms, the model also handles any subset of the language. Infinite processes are available through guarded replication $!\alpha.P$ for any prefix α, though the interpretation works equally well for unguarded replication and guarded recursion.

4.1. Translation

This is in two stages: we begin with certain elements of the domain $Pi_\perp s$, and then build the morphisms from these. For a π-calculus process P with free names in s, Figure 2 describes an element $(\!|P|\!)_s \in Pi_\perp s$, inductively on the structure of P. Guarded replication $!\alpha.P$ is interpreted using the order structure of $Pi_\perp s$; for unguarded replication we can either use $(\!|!P|\!)_s = \mu p.lpar_s((\!|P \mid P|\!)_s, p)$ or recall Sangiorgi's result that these two forms are interdefinable up to strong equivalence [17, §6.3]. Note the use of prioritised parallel composition $lpar$, non-strict in its right argument, to ensure that the resulting least fixed point is a fully determined process. Guarded recursion can be handled similarly.

To raise $(\!|P|\!)_s$ from an element to a morphism we define $[\![P]\!]_s : N^s \longrightarrow Pi_\perp$ by taking $[\![P]\!]_s s'\rho = (\!|P[\rho]|\!)_{s'}$ for $s' \in \mathcal{I}$ and $\rho \in N^s s'$, that is $\rho : s \to s'$. The difference between the two forms is that interpreting a process as an element assumes that all names are distinct, whereas the morphism includes behaviour under all possible name identifications. Thus $(\!|-|\!)_s$ is the 'ground' notion, and $[\![-]\!]_s$ the more general one.

4.2. Results

We consider the *strong*, *late* semantics for process behaviour, where τ-actions are significant and input actions commit on a channel before value transmission. A suitable transition relation for this semantics appears in [11], which then derives a notion of *bisimilarity* between π-calculus processes. This is not closed under input prefix, so processes are further defined to be *equivalent* if they are bisimilar under all name substitutions. Both of these relations are captured exactly by the model in $\mathcal{C}$, thanks to the following strong result on transitions.

Theorem 1. *If P is a π-calculus process then its interpretation in $\mathcal{C}$ both reflects and preserves transitions. For example:*

$$P \xrightarrow{\bar{x}y} Q \;\Rightarrow\; out(x, y, (\!|Q|\!)_s) \in (\!|P|\!)_s$$
$$tau(q) \in (\!|P|\!)_s \;\Rightarrow\; \exists Q \,.\, P \xrightarrow{\tau} Q \;\;\&\;\; (\!|Q|\!)_s = q$$

being respectively instances of soundness and adequacy.

Proof. Soundness is shown by rule induction: the translation respects every rule of the operational semantics. Adequacy requires an inductively defined formal approxima-

$$
\begin{aligned}
(\!|\bar{x}y.P|\!)_s &= \{\!|\, out(x, y, (\!|P|\!)_s)|\!\} & (\!|0|\!)_s &= \emptyset \\
(\!|x(y).P|\!)_s &= \{\!|\, in(x, \lambda y.(\!|P|\!)_{s+\{y\}})|\!\} & (\!|P + Q|\!)_s &= (\!|P|\!)_s \uplus (\!|Q|\!)_s \\
(\!|\nu x\, P|\!)_s &= new_s(\lambda x.(\!|P|\!)_{s+\{x\}}) & (\!|P \mid Q|\!)_s &= par_s((\!|P|\!)_s, (\!|Q|\!)_s) \\
(\!|!\alpha.P|\!)_s &= \mu p.lpar_s((\!|\alpha.P|\!)_s, p)
\end{aligned}
$$

$$
(\!|[x = y]P|\!)_s = \begin{cases} (\!|P|\!)_s & x = y \\ \emptyset & x \neq y \end{cases}
\qquad
(\!|[x \neq y]P|\!)_s = \begin{cases} \emptyset & x = y \\ (\!|P|\!)_s & x \neq y \end{cases}
$$

Figure 2. Interpretation of π-calculus processes as elements of $Pi_\perp s$.

tion relation $p \lhd_s P$ between elements of $Pi_\perp s$ and processes over names s; loosely, any transition in the element p can be matched by the process P. The method is standard, but its application here is rather delicate. We only consider elements p that are 'image finite' — finite sets, in the explicit representation of the powerdomain. All elements arising from the translation $(\!|-|\!)$ are image finite, even those involving replication, and we can show that $(\!|P|\!)_s \lhd_s P$ by induction on the structure of P, from which the result follows. Note that this is a proof directly on the transitions, so it does not depend on the expansion law or any other equations. □

Corollary 2 (Adequacy). *The model in $\mathcal{C}$ is adequate for strong bisimilarity '$\dot{\sim}$' and strong equivalence '$\sim$' between processes:*

$$
\begin{aligned}
(\!|P|\!)_s = (\!|Q|\!)_s &\;\Rightarrow\; P \dot{\sim} Q \\
[\![P]\!]_s = [\![Q]\!]_s &\;\Rightarrow\; P \sim Q\ .
\end{aligned}
$$

Proof. We combine the result above with the definition of bisimulation and equivalence in terms of transitions. Roughly speaking, if two processes have equal interpretations then they have matching transitions, and this is enough to show strong bisimilarity or equivalence as appropriate. Again we have the distinction between ground notions (elements, bisimilarity) and more general ones that consider all possible name substitutions (morphisms, equivalence). □

Thus the model in $\mathcal{C}$ can be used to prove equivalences between specific processes, and to verify algebraic laws for the π-calculus [11, 12]. Indeed because we have not even used the basic structural equivalences, such as commutativity of '$+$', the model proves their validity too. Furthermore, this method is complete.

Theorem 3. *The object $Pi_\perp$ is 'internally fully abstract', in that elements are determined entirely by their transitions and whether they contain $\perp$.*

Proof. In [14], Pitts provides proof principles that give exactly this result for a wide range of recursive predomain constructions. The general scheme is to define a notion of bisimilarity between domain elements, and then use coinduction to show that elements are equal if and only if they are bisimilar. For $Pi_\perp$ we can repeat this work in the functor category $\mathcal{C}$. □

Corollary 4 (Full Abstraction). *The model in $\mathcal{C}$ is fully abstract for strong bisimilarity and strong equivalence:*

$$
\begin{aligned}
P \dot{\sim} Q &\;\Rightarrow\; (\!|P|\!)_s = (\!|Q|\!)_s \\
P \sim Q &\;\Rightarrow\; [\![P]\!]_s = [\![Q]\!]_s\ .
\end{aligned}
$$

Proof. Theorem 3 involves a form of bisimilarity between domain elements. By Theorem 1 on transitions, operational bisimilarity between two processes corresponds exactly to this domain bisimilarity between their interpretations. Internal full abstraction then gives that such bisimilar processes actually have equal interpretations. The same result for strong equivalence follows by considering all possible name substitutions. □

Figure 3 summarises how the various operational notions of process behaviour are interpreted by the categorical model.

5. Further Work

The purpose in having simple equations and a sophisticated category is that, once the tools are assembled, further work is simple and intuitive. For example we can immediately give a fully abstract interpretation of equivalence up to *distinctions* or even up to Parrow and Sangiorgi's *conditions* on name sets [12], completing a spectrum between bisimilarity and equivalence. The key observation here is that every subobject of N^s represents s-environments with some restriction upon which names must or must not be identified. Alternatively, replacing equations (2) and (3) with $In \cong Out \cong N \times (N \multimap Pi_\perp)$ gives a model of the πI-calculus, where all communicated names are fresh [18]; this

	Operation	Denotation
Transitions:	$P \xrightarrow{\bar{x}y} Q$	$\{\!\|out(x, y, (\!\|Q|\!)_s)|\!\} \in (\!|P|\!)_s$ *etc.*
Bisimilarity:	$P \dot{\sim} Q$	$(\!|P|\!)_s = (\!|Q|\!)_s \in Pi_\perp s$
Equivalence:	$P \sim Q$	$[\![P]\!]_s = [\![Q]\!]_s \ : \ N^s \longrightarrow Pi_\perp$

Figure 3. Operational notions and their categorical interpretation.

clearly highlights the symmetry between input and output in πI. For a second-order π-calculus an obvious first step is to replace N by $Pi_\perp$ as the object of the input and output clauses; though it is not yet clear what this corresponds to operationally. It might also be possible to adapt the equations to model early, weak or open variants of bisimilarity. Similarly, we could look at processes with sorts, or transitions annotated to allow more detailed analysis of concurrency [3, 13, 16].

There are various ways we might try to give a more abstract presentation of the model in $\mathcal{C}$. An *internal language* would allow a more succinct description of the morphisms used; unfortunately, the rich interaction between the cartesian and monoidal closed structures, that makes possible this model of names, also leads to well-known problems in the formulation of an internal language. The approaches of *axiomatic* and *synthetic* domain theory would allow us to treat the objects of $\mathcal{C}$ not as functors but as standard domains or even sets; indeed this work provides a concrete example of the need for such generalised domain theory.

The work of Fiore, Moggi and Sangiorgi [5] follows up some of these lines. They give an internal language for their category, not for the whole monoidal closed structure but just the essential operation $N \multimap (-)$ (which they call δ). This leads to a more abstract presentation of how process terms are interpreted as morphisms. With the proof of full abstraction though, a tradeoff becomes apparent: although their proof uses no explicit handling of domain elements, it does require that π-calculus processes be manipulated into a certain normal form. Axioms and rule schemes for operational reasoning about processes are thus essential; while our proofs have needed no such *a priori* results on bisimilarity or equivalence.

Following Abramsky [1] we might use Stone duality and seek a *domain logic* for $Pi_\perp$. It would be interesting to see how this compares with the range of modal logics already proposed for π-calculus processes [2, 10]. We can also consider what this denotational semantics suggests for the formulation of the π-calculus itself. Concretion and abstraction are naturally interpreted by the objects $(N \times Pi_\perp)$ and $(N \to Pi_\perp)$ — perhaps other constructions in $\mathcal{C}$ also give useful operational notions.

Finally, we may ask what happens when other categorical models of concurrency, such as those in [9, 22], are indexed by $\mathcal{I}$: does the monoidal closed structure still capture some appropriate notion of a 'name'?

References

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

[2] R. Amadio and M. Dam. Toward a modal theory of types for the π-calculus. Research Report R96:03, Swedish Institute of Computer Science, 1996.

[3] M. Boreale and D. Sangiorgi. A fully abstract semantics for causality in the π-calculus. In *Proceedings of STACS '95: 12th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 900. Springer-Verlag, 1995.

[4] 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.

[5] M. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the π-calculus. In *Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1996.

[6] M. Hennessy. A fully abstract denotational model for higher-order processes (extended abstract). In *Proceedings of the Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 397–408. IEEE Computer Society Press, 1993.

[7] M. Hennessy and G. Plotkin. Full abstraction for a simple parallel programming language. In *Mathematical Foundations of Computer Science: Proceedings of the 8th International Symposium MFCS '79*, Lecture Notes in Computer Science 74, pages 108–120. Springer-Verlag, 1979.

[8] 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, Jan. 1995.

[9] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. BRICS Report RS-94-7, Department of Computer Science, University of Aarhus, May 1994. To appear in *Information and Computation*.

[10] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. Technical Report ECS-LFCS-91-136, Laboratory for Foundations of Computer Science, University of Edinburgh, Apr. 1991.

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

[12] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. *Information and Computation*, 120(2):172–197, Aug. 1994.

[13] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. To appear in *Mathematical Structures in Computer Science*. A summary was presented at LICS '93.

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

[15] A. M. Pitts and I. Stark. Observable properties of higher order functions that dynamically create local names, or: What's *new*? In *Mathematical Foundations of Computer Science: Proceedings of the 18th International Symposium MFCS '93*, Lecture Notes in Computer Science 711, pages 122–141. Springer-Verlag, 1993.

[16] D. Sangiorgi. Locality and true-concurrency in calculi for mobile processes. In *Theoretical Aspects of Computer Software: TACS '94*, Lecture Notes in Computer Science 789. Springer-Verlag, 1994.

[17] D. Sangiorgi. On the bisimulation proof method. Technical Report Report ECS-LFCS-94-299, Laboratory for Foundations of Computer Science, University of Edinburgh, Aug. 1994.

[18] D. Sangiorgi. π-calculus, internal mobility, and agent-passing calculi. Rapport de recherche 2539, INRIA, Sophia Antipolis, Apr. 1995.

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

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

[21] I. Stark. Categorical models for local names. *Lisp and Symbolic Computation*, 9(1):77–107, Feb. 1996.

[22] G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume IV. Oxford University Press, Apr. 1995.