Appears in *Proceedings of FOSSACS 2005, Edinburgh*,
LNCS 3441, pp.155–169. Springer-Verlag, 2005

Free-Algebra Models for the π-Calculus

Ian Stark*

Laboratory for Foundations of Computer Science
School of Informatics, The University of Edinburgh, Scotland
Ian.Stark@ed.ac.uk

Abstract. The finite π-calculus has an explicit set-theoretic functor-category model that is known to be fully abstract for strong late bisimulation congruence. We characterize this as the initial free algebra for an appropriate set of operations and equations in the enriched Lawvere theories of Plotkin and Power. Thus we obtain a novel algebraic description for models of the π-calculus, and validate an existing construction as the universal such model.

The algebraic operations are intuitive, covering name creation, communication of names over channels, and nondeterminism; the equations then combine these features in a modular fashion. We work in an enriched setting, over a "possible worlds" category of sets indexed by available names. This expands significantly on the classical notion of algebraic theories, and in particular allows us to use nonstandard arities that vary as processes evolve.

Based on our algebraic theory we describe a category of models for the π-calculus, and show that they all preserve bisimulation congruence. We develop a direct construction of free models in this category; and generalise previous results to prove that all free-algebra models are fully abstract.

1 Introduction

There are by now a handful of models known to give a denotational semantics for the π-calculus [2, 3, 6, 7, 8, 10, 36]. All are fully abstract for appropriate operational equivalences, and all use functor categories to handle the central issue of names and name creation. In this paper we present a method for generating such models purely from their algebraic properties.

We address specifically the finite π-calculus model as presented by Fiore et al [8]. This uses the functor category $\mathcal{S}et^{\mathcal{I}}$, with index $\mathcal{I}$ the category of finite name sets and injections, and is fully abstract for strong late bisimulation congruence. We exhibit this as one among a category of algebraic models for the π-calculus: all such π-algebras respect bisimulation congruence, and we give a concrete description of the free π-algebra $Pi(X)$ for any object X of $\mathcal{S}et^{\mathcal{I}}$. We show that every free algebra is a fully-abstract model for the π-calculus, with the construction of Fiore et al. being the initial free algebra $Pi(0)$.

Our method builds on a recent line of research by Plotkin and Power who use algebraic theories in enriched categories to capture "notions of computation", in particular Moggi's *computational monads* [18, 26, 27, 28]. The general idea is to describe a computational feature — I/O, state, nondeterminism — by stating a characteristic collection

* Research supported by an EPSRC Advanced Research Fellowship

of operations with specified equations between them. These then induce the following suite of constructions: a notion of algebraic model for the feature; a computational monad; effectful actions to program with; and a modal logic for specification and reasoning. This approach also gives a flexible way to express interactions between features, by combining sets of operations [11, 12].

For the π-calculus, we apply and expand their technique. The enriched setting supports not only models that are objects in $\mathcal{S}et^{\mathcal{I}}$, but also arities from $\mathcal{S}et^{\mathcal{I}}$; so that we have operations whose arity depends on the names currently available. We use two different closed structures in $\mathcal{S}et^{\mathcal{I}}$: the usual cartesian exponential for arities, and a monoidal function space "$\multimap$" for operations parameterised by *fresh* names. Finally, the π-calculus depends on a very particular interaction between concurrency, communication and name generation, which we can directly express in equations relating the theories for each of these features. This precision in integrating different aspects of computation is a significant benefit of the algebraic approach over existing techniques for combining computational monads [13, 15, 19, 37].

The structure of the paper is as follows. In §2 we review the relevant properties of algebraic theories and the functor category $\mathcal{S}et^{\mathcal{I}}$. We then set out our proposed algebraic theory of π in §3. Following this, in §4 we show how models of the theory give a denotational semantics for the finite π-calculus (i.e., omitting recursion and replication), and prove that these interpretations respect bisimulation congruence (Prop. 2). Interestingly, parallel composition of processes is not in general admissible as a basic operation in the theory, although we are able to interpret it via expansion. We prove the existence of free algebras over $\mathcal{S}et^{\mathcal{I}}$ (Thm. 3) and show that they are all fully abstract (Thm. 5). In particular, the free algebra over the empty set is exactly the model of Fiore et al., and does support an internal definition of parallel composition (Prop. 4). Finally, we identify the monad induced by the theory of π, which gives a programming language semantics for mobile communicating concurrency. We conclude in §5 by indicating possible extensions and further applications of this work.

2 Background

We outline relevant material on algebraic theories and the target category $\mathcal{S}et^{\mathcal{I}}$. For π-calculus information, see one of the books [16, 32] or Parrow's handbook chapter [23].

2.1 Algebras and Notions of Computation

We sketch very briefly the theoretical basis for our development: for more on enriched algebraic theories see Robinson's clear and detailed exposition [31]; the link to computations and generic effects is described in [27, 28].

There is a well-established connection between algebraic theories and monads on the category $\mathcal{S}et$. For example, consider the following theory, which we shall use later for an algebra A of nondeterministic computations:

$$\begin{aligned} \mathit{choice} &: A \times A \longrightarrow A \\ \mathit{nil} &: 1 \longrightarrow A \end{aligned}$$

Operation choice for combining computations to be commutative, associative and idempotent with unit nil.

A model of this theory is a triple $\langle A, choice, nil\rangle$ of a carrier set A with two maps satisfying the relevant commuting diagrams; and these models form a category $\mathcal{ND}(\mathcal{S}et)$ of "nondeterministic sets". The forgetful functor U into $\mathcal{S}et$ has a left adjoint, giving the free algebra FX over any set X.

$$\text{free} \quad F \quad \begin{array}{c} \mathcal{ND}(\mathcal{S}et) \\ \uparrow \\ \left(\dashv \right) \\ \downarrow \\ \mathcal{S}et \end{array} \quad U \quad \text{forgetful}$$

In fact, this functor F is the finite powerset $\langle \mathcal{P}_{fin}, \cup, \emptyset\rangle$, and $\mathcal{ND}(\mathcal{S}et)$ is *monadic* over $\mathcal{S}et$: it is equivalent to the category of algebras for the monad $\mathcal{P}_{fin}$.

The situation here is quite general, with a precise correspondence between single-sorted algebraic theories and finitary monads on $\mathcal{S}et$ (i.e., monads that preserve filtered colimits). Kelly and Power [14, 29] extend this to an enriched setting: carriers for the algebras may be from categories other than $\mathcal{S}et$; the arities of operations can be not just natural numbers, but certain objects in a category; and equations can be replaced with other constraint systems — for example, ordered categories support inequations.

Building on this, Plotkin and Power investigate algebraic theories that induce a "computational" monad T [18]. They characterize when an operation $f : (TX)^m \to (TX)^n$ on computations is *algebraic* and hence admissible as an operation of the relevant theory. Moreover, they prove that every such algebraic operation corresponds to a computational *effect* of type $e_f : n \to Tm$ (note the reversal of indices m and n). In the example above, $\mathcal{P}_{fin}$ is the standard computational monad for finite nondeterminism, and its effects are $\mathsf{arb} : 1 \longrightarrow \mathsf{T2}$ and $\mathsf{deadlock} : 1 \longrightarrow \mathsf{T0}$. These two are enough to code up nondeterministic programming: $\mathsf{arb}()$ is a nondeterministic true or false, and $\mathsf{deadlock}()$ is the empty choice.

Thus not only do algebraic theories characterize computational monads as free algebras, but they also provide the necessary terms to program with them. They also support combining monads, a traditionally challenging area, by taking the union of theories and possibly introducing new equations describing how they interact [11, 12].

As a final example, the theory for input/output of data values from some fixed set V is:

$$in : A^V \longrightarrow A \qquad out : A \longrightarrow A^V \qquad \text{with no equations.}$$

This induces the *resumptions* monad for computations performing I/O:

$$T(-) = \mu X.(X^V + V \times X + (-))$$

as well as the effects $\mathsf{read} : 1 \longrightarrow \mathsf{TV}$ and $\mathsf{write} : \mathsf{V} \to \mathsf{T1}$.

2.2 The Category $\mathcal{S}et^{\mathcal{I}}$

We construct our models for π over the functor category $\mathcal{S}et^{\mathcal{I}}$, where $\mathcal{I}$ is the category of finite sets and injections. Typically we treat objects $s, s' \in \mathcal{I}$ in the index category as finite sets of names. The intuition is that an object $X \in \mathcal{S}et^{\mathcal{I}}$ is a *varying* set: if $s \in \mathcal{I}$ is the set of names available in some context, then $X(s)$ is the set of X-values using them.

As the set of names available changes, so does this set of values. Functor categories of *possible worlds* like this are well established for modelling local state in programming languages [20, 22, 30] and local names in particular [17, 25, 35]. Similar categories of varying sets also appear in models for variable binding [5] and name binding (see, for example, [33] and citations there).

Category $\mathcal{S}et^{\mathcal{I}}$ is complete and cocomplete, with limits and colimits taken pointwise. It is cartesian closed, with a convenient way to calculate function spaces using natural transformations between functors:

$$\begin{array}{ll} X \times Y & (X \times Y)(s) = X(s) \times Y(s) \\ X \to Y \text{ or } Y^X & \qquad Y^X(s) = \mathcal{S}et^{\mathcal{I}}[X(s + \_), Y(s + \_)] \end{array}$$

Thus elements in the varying set of functions from X to Y over names s must take account of values in $X(s + s')$, uniformly for all extended name sets $s + s'$.

There is also a symmetric monoidal closed structure $(\otimes, \multimap)$ around the *Day tensor* [4], induced by disjoint union $(s + s')$ in $\mathcal{I}$.

$$X \otimes Y = \int^{s,s' \in \mathcal{I}} X(s) \times Y(s') \times \mathcal{I}[s + s', \_]$$

All the constructions in this paper remain within the subcategory of functors in $\mathcal{S}et^{\mathcal{I}}$ that preserve pullbacks. For such functors we can give an explicit presentation of the monoidal structure:

$$\begin{aligned} (X \otimes Y)(s) = \big\{\, & (x, y) \in (X \times Y)(s) \\ & \big|\ \exists \text{disjoint } s_1, s_2 \subseteq s \;.\; x \in X(s_1), y \in Y(s_2) \,\big\} \\ (X \multimap Y)(s) = \; & \mathcal{S}et^{\mathcal{I}}[X(\_), Y(s + \_)] \end{aligned}$$

Elements of $(X \otimes Y)$ denote pairs of elements from X and Y that use disjoint name sets. Elements of the monoidal function space $(X \multimap Y)$ are functions defined only at X-values that use just fresh names.

The two closed structures are related:

$$\begin{aligned} into_{X,Y} &: X \otimes Y \longrightarrow X \times Y \\ onto_{X,Y} &: (X \to Y) \longrightarrow (X \multimap Y) \;. \end{aligned}$$

Where functors X and Y are pullback-preserving, these are an inclusion and surjection, respectively.

We use a variety of objects in $\mathcal{S}et^{\mathcal{I}}$. For any fixed set S, there is a corresponding constant functor $S \in \mathcal{S}et^{\mathcal{I}}$. The *object of names* $N \in \mathcal{S}et^{\mathcal{I}}$ is the inclusion functor mapping any $s \in \mathcal{I}$ to the same $s \in \mathcal{S}et$. From this we build $(N \times N \times \cdots \times N) = N^k$, the object of k-tuples of names, and $(N \otimes N \otimes \cdots \otimes N) = N^{\otimes k}$ of distinct k-tuples, with an inclusion $into : N^{\otimes k} \hookrightarrow N^k$ between them.

We have the *shift* functor δ on objects of $\mathcal{S}et^{\mathcal{I}}$:

$$\delta : \mathcal{S}et^{\mathcal{I}} \longrightarrow \mathcal{S}et^{\mathcal{I}} \quad \text{defined by} \quad \delta X(\_) = X(\_ + 1) \;.$$

In fact $\delta(-) \cong N \multimap (-)$, and elements of δX are elements of X that may use a single fresh name, uniformly in the choice of that name. This functor is well known, for example as *dynamic allocation* in [6, 7]; it also appears as the *atom abstraction* operator $[N]X$ of FM-set theory identified by Gabbay and Pitts [9, 24]. Note that shifting the object of names gives a coproduct: $\delta N \cong (N + 1)$.

The representable objects in $\mathcal{S}et^{\mathcal{I}}$ are $1, N, (N \otimes N), (N \otimes N \otimes N), \ldots$ The finitely presentable objects are the finite colimits of these, including in particular finite constant sets S, and all finite products of N: for example, $(N \times N) \cong N + (N \otimes N)$. These are the objects available as arities for algebraic theories over $\mathcal{S}et^{\mathcal{I}}$.

Finally, the category $\mathcal{S}et^{\mathcal{I}}$ is locally finitely presentable as a closed category, with respect to both cartesian and monoidal structures. This is a completeness requirement for building algebraic theories: [29, §2] and [31, §3] expand on what this involves.

3 Theory of π

The algebraic approach supports a modular presentation of theories, and we use this to manage the combination of features that come together in the π-calculus. This section presents in turn separate theories for nondeterminism, communication along channels, and dynamic name creation; followed by equations specifying exactly how these features should interact.

We assume a carrier object $A \in \mathcal{S}et^{\mathcal{I}}$, and describe the operations and equations required for A to model the π-calculus.

3.1 Nondeterministic Choice

For nondeterminism we need a binary $choice$ operation that is commutative, associative and idempotent with a unit nil.

$$\begin{aligned} choice &: A^2 \longrightarrow A & choice(p,q) &= choice(q,p) \\ nil &: 1 \longrightarrow A & choice(nil,p) &= choice(p,p) = p \\ & & choice(p,(choice(q,r)) &= choice(choice(p,q),r) \end{aligned}$$

In process calculus terms, $choice$ captures nondeterministic sum $P + Q$ and nil the deadlocked process 0.

3.2 Communication

Communication in the π-calculus is along named channels, sending names themselves as data. The relevant theory is a specialised version of that for I/O given earlier.

$$\begin{aligned} out &: A \longrightarrow A^{N \times N} \\ in &: A^N \longrightarrow A^N \\ tau &: A \longrightarrow A \end{aligned}$$

(No required equations)

These three operations correspond to the three prefixing constructions of the π-calculus: output $\bar{x}y.P$, input $x(y).P$ and silent action $\tau.P$. Argument and result arities follow the bound and free occurrences of names respectively:

- out is parameterized in the result $A^{N\times N}$ by both channel and data names;
- in accepts argument A^N parameterized by the data value, with result A^N parameterized by channel name.

The appearance of A^N and $A^{N\times N}$ here give our first nonstandard arities, N and $N\times N$, to describe operations whose arity varies according to the names currently available. We follow [27] in using formal indices to write these down: with terms like $out_{x,y}(p)$ and $in_x(q_y)$, where x and y are name parameters.

3.3 Dynamic Name Creation

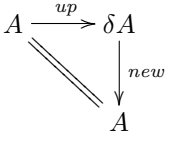
Processes in the π-calculus can dynamically generate fresh communication channels: term $\nu n.P$ is the process that creates a new channel, binds it to the name n, and then becomes process P which may then use the new channel.

Our theory for this is a modification of Plotkin and Power's $block$ operation for local state [27, §4]. We require a single operation new with a monoidal arity.

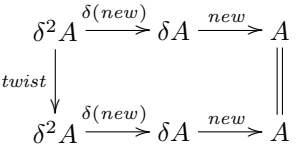
$$new : \delta A \to A \qquad new(x.p) = p \quad \text{for } p \text{ independent of } x$$
$$\delta A \cong N \multimap A \qquad new(x.new(y.p)) = new(y.new(x.p))$$

The argument δA means that new is an operation of arity N in the monoidal closed structure of $\mathcal{S}et^{\mathcal{I}}$. Recall that elements of δA are elements of A that depend on a single fresh name, uniformly in the choice of that fresh name. In the equations for new we write $x.p$ for the term p indexed by fresh x, borrowing Gabbay and Pitts's notation for atom abstraction [9]. (Plotkin and Power write this as $\langle p\rangle_x$.)

Strictly, all our equations are shorthand for certain diagrams in $\mathcal{S}et^{\mathcal{I}}$ which must commute. These two state that the creation of unused fresh names cannot be observed, and computation is independent of the order in which fresh names are created. In diagram form, these are



and



where $up : 1 \to \delta$ and $twist : \delta^2 \to \delta^2$ are the evident natural transformations on the shift functor.

3.4 Other Operations

There are a few further constructions that might be candidates for inclusion in a theory of π.

Name testing Some forms of the π-calculus allow direct comparison of names, with prefixes like match $[x = y]P$, mismatch $[x \neq y]Q$, or two-branched testing $(x = y)\ ?\ P : Q$. It turns out that these operations are already in the theory. The $\mathcal{S}et^{\mathcal{I}}$ map of arities $(N \times N) \cong N + (N \otimes N) \longrightarrow 1 + 1$ induces an operation $test$ from which others follow, using nil:

$$test : A^2 \longrightarrow A^{N \times N} \qquad eq : A \longrightarrow A^{N \times N} \qquad neq : A \longrightarrow A^{N \times N} .$$

Bound output The bound output prefix $\bar{x}(y).P$ for the π-calculus is equivalent to $\nu y(\bar{x}y.P)$. There is an analogous derived operation in the theory:

$$bout : \delta A \longrightarrow A^N \qquad bout_x(y.p) \stackrel{def}{=} new(y.out_{x,y}(p))$$

Because this is definable in terms of the operations given earlier, it can be included without affecting the induced theory or its algebras.

Parallel composition The usual process calculus construction $(P \mid Q)$ is not directly admissible as an operation in our theory of π. This is because it is not *algebraic* in the sense of Plotkin and Power [28]. Informally, it does not commute with composition of computations: in a programming language, $(\mathsf{M} \mid \mathsf{M}'); \mathsf{N}$ is not in general equivalent to $(\mathsf{M}; \mathsf{N}) \mid (\mathsf{M}'; \mathsf{N})$. We shall see more on this later, in §4.

3.5 Combining Equations

To complete the theory of π we give equations to specify how the component theories interact. The algebraic approach gives us some flexibility in doing so, as investigated in [11, 12]. For example, we can assert no additional equations, giving the *sum* of theories [12, §3]; we can require that the operations from two theories commute with each other, to give the commutative combination, or *tensor*, of theories [12, §4]; or we can choose some other custom interaction. To assemble the component theories of π, we use all three methods:

- The sum of the theories of nondeterminism and communication.
- The commuting combination of nondeterminism and name creation.
- A custom set of equations for name creation and communication; mostly commuting, but some specific interaction.

These expand into three sets of equations. The first have effect by their absence:

Sum of component theories

No equations required for $choice$ or nil with out, in or tau.

The commuting combination of theories says that operations act independently:

Commuting component theories

$$new(x.choice(p,q)) = choice(new(x.p), new(x.q))$$

$$\begin{aligned} new(z.out_{x,y}(p)) &= out_{x,y}(new(z.p)) && z \notin \{x,y\} \\ new(z.in_x(p_y)) &= in_x(new(z.p_y)) && z \notin \{x,y\} \\ new(z.tau(p)) &= tau(new(z.p)) \end{aligned}$$

Recall that these equations with formal indices and side conditions are a shorthand for four commuting diagrams in $\mathcal{S}et^{\mathcal{I}}$.

Finally, just two equations for interaction capture the precise flavour of the π-calculus: that the binder $\nu x.(-)$ is both creation (of new channels) and restriction (of communication on them).

Interaction between component theories

$$\begin{aligned} new(x.out_{x,y}(p)) &= nil \\ new(x.in_x(p_y)) &= nil \end{aligned}$$

4 Algebraic Models for π

We now turn to look at models for the theory of π. We define what these are, and show that every such model gives a denotational semantics for the π-calculus that respects bisimulation congruence. We give a construction for free models in $\mathcal{S}et^{\mathcal{I}}$, and prove that the category of models is monadic over $\mathcal{S}et^{\mathcal{I}}$. We show that all free models are fully abstract for bisimulation congruence, and in particular that the initial free model is isomorphic to the construction of Fiore et al.

4.1 Categories of Algebras

Definition 1. *A* π-algebra *in* $\mathcal{S}et^{\mathcal{I}}$ *is an object* A *together with maps* $(choice, nil, out, in, tau, new)$ *satisfying the equations of* §§3.1–3.3 *and* 3.5 *above. These algebras form a category* $\mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$*, with morphisms the maps* $f : A \to B$ *that commute with all operations. The forgetful functor* $U : \mathcal{PI}(\mathcal{S}et^{\mathcal{I}}) \to \mathcal{S}et^{\mathcal{I}}$ *takes a* π*-algebra to its carrier object.*

For any π-algebra $A \in \mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$ we can build a denotational semantics of the finite π-calculus: if P is a process with free names in set s, then there is a map

$$[\![s \vdash P]\!]_A : N^{|s|} \longrightarrow A \ .$$

Here $N^{|s|}$ represents an environment instantiating the free names s.

The interpretation itself is comparatively straightforward. Process sum, nil and the π-calculus prefixes are interpreted directly by the corresponding π-algebra operations.

Binding of fresh names involves managing the monoidal structure; we use a construction $\nu(-)$ on maps into A:

$$\begin{aligned} p : N^{|s|+1} &\longrightarrow A && \text{Given a map } p; \\ N \otimes N^{|s|} \xrightarrow{into} N \times N^{|s|} &\longrightarrow A && \text{precompose inclusion;} \\ N^{|s|} &\longrightarrow (N \multimap A) && \text{take the monoidal transpose;} \\ N^{|s|} &\longrightarrow \delta A \xrightarrow{new} A && \text{and apply the } new \text{ operator} \\ \nu p : N^{|s|} &\longrightarrow A && \text{to get the restricted map } \nu p. \end{aligned}$$

We then define $[\![s \vdash \nu x.P]\!]_A = \nu([\![s, x \vdash P]\!]_A)$.

As noted earlier, parallel composition is not algebraic, so we have no general map for its action on A. However, for any specific finite processes P and Q we can use the expansion law for congruence [23, Table 9] to express $(P \mid Q)$ as a sum of smaller processes, and so obtain an interpretation in the π-algebra A, recursively:

$$\text{if} \qquad P \mid Q = \sum_{i=1}^{k} R_i \qquad \text{(canonical choice of expansion)}$$

$$\text{then} \qquad [\![s \vdash P \mid Q]\!]_A = choice([\![s \vdash R_1]\!]_A, choice([\![s \vdash R_2]\!]_A, \dots)) : N^{|s|} \longrightarrow A \,.$$

This external expansion makes the translation not wholly compositional; later we shall improve on this, for one particular π-algebra, by expressing parallel composition within the algebra itself.

The interpretation $[\![s \vdash P]\!]_A$ respects weakening of the name context s, so we usually omit it and write $[\![P]\!]_A$.

Once defined, this interpretation induces a notion of equality over a model: for any π-algebra A and finite processes P, Q we write

$$A \models P = Q \quad \overset{def}{\iff} \quad [\![P]\!]_A = [\![Q]\!]_A$$

$$\text{and} \quad \mathcal{S}et^{\mathcal{I}} \models P = Q \quad \overset{def}{\iff} \quad A \models P = Q \text{ for all } A \in \mathcal{PI}(\mathcal{S}et^{\mathcal{I}}).$$

Proposition 2. *All π-algebra models respect (strong, late) bisimulation congruence. For any $A \in \mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$ and finite processes P, Q:*

$$P \approx Q \quad \Longrightarrow \quad A \models P = Q$$

and more generally:

$$P \approx Q \quad \Longrightarrow \quad \mathcal{S}et^{\mathcal{I}} \models P = Q \,.$$

Proof. We draw on the known axiomatization of bisimulation congruence for finite processes, as given for example in [23, §8.2]. All these axioms are provable in the theory of π and hence hold in every algebra for the theory. □

4.2 Free π-algebras in $\mathcal{S}et^{\mathcal{I}}$

The previous section proposes a theory of algebraic models for the π-calculus; but it does not yet give us any concrete π-algebras. For these we seek a free π-algebra functor $F : \mathcal{S}et^{\mathcal{I}} \to \mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$, left adjoint to the forgetful U. Kelly and Power [14, 29] show the existence in general of such algebras for enriched theories; but there are two difficulties in our situation. First, their results are in terms of a general colimit, and for any specific theory one would also like a direct form if possible. Second, and more serious, they treat a single enrichment, while we have two together.

We can overcome both of these difficulties, in the specific case of $\mathcal{S}et^{\mathcal{I}}$: we have an explicit description of the free π-algebras, and an accompanying proof that they are so.

Before presenting the free algebras for the full theory of π, we detour briefly through those for each of its component theories, to see how they fit together. For simplicity we present not the free functors F, but the associated monads $(U \circ F)$ on $\mathcal{S}et^{\mathcal{I}}$.

The monad for finite nondeterminism is the finite covariant powerset, extended pointwise to $\mathcal{S}et^{\mathcal{I}}$:

$$T_{nondet}(-) = \mathcal{P}_{\mathit{fin}}(-) \ .$$

The monad for communication is a version of the resumptions monad, with components for output, input and silent action:

$$T_{comm}(-) = \mu X.(N \times N \times X + N \times X^N + X + (-)) \ .$$

Here $\mu X.(-)$ is the least fixed point, which in $\mathcal{S}et^{\mathcal{I}}$ is a straightforward pointwise union. Informally, an element of $(T_{comm}Y)(s)$ is a finite trace of π-calculus actions using names from s, finishing with a value from Y; with the refinement that at input actions the function space X^N gives a branching over possible input names, including uniform treatment of new names.

The monad for dynamic name creation is that originating with Moggi [17, §4.1.4] and investigated in [35].

$$T_{new}(-) = \mathcal{D}yn(-) = \varinjlim_{s \in \mathcal{I}} \left(N^{\otimes |s|} \multimap (-) \right) \ .$$

This is a colimit over possible sets of fresh names. In particular, the object part has $\mathcal{D}yn(X)(s) = \sum_{s' \in \mathcal{I}} X(s + s')/\sim$, where $\sim$ is an equivalence relation generated by injections between fresh name sets $s' \rightarrowtail s''$. For full element-by-element details of the $\mathcal{D}yn$ construction, see [35, §5].

Taking the approach of combining monads through monad *transformers* [15], we can try to interleave these to obtain a candidate monad for π:

$$T_{bad}(-) = \mu X.(\mathcal{P}_{\mathit{fin}}(\mathcal{D}yn(N \times N \times X + N \times X^N + X + (-)))) \ .$$

Working from the outside in, this asserts that: a π-calculus process is a recursive system (μX); which may have several courses of action ($\mathcal{P}_{\mathit{fin}}$); that each may create fresh names ($\mathcal{D}yn$); and then perform some I/O action, to give some further process.

However, this is not yet quite right: T_{bad} does not validate any of the equations of §3.5 for combining the different π-calculus effects. For example, in T_{bad} restriction

new does not commute with $choice$; nor does it in fact restrict, as there are terms in the monad for external I/O on a new-bound channel.

To find the correct monad for π, we use an observation from existing operational treatments: name creation is only observable through the emission of fresh names in bound output. This leads to the following corrected definition:

$$T_\pi(-) = \mu X.(\mathcal{P}_{fin}(N \times N \times X + N \times \delta X + N \times X^N + X + \mathcal{D}yn(-))) \; . \quad (1)$$

This still expresses a π-calculus process as a recursive system (μX) with several courses of action ($\mathcal{P}_{fin}$); but the general application of $\mathcal{D}yn(-)$ has been replaced by a bound output term $N \times \delta X$ in the I/O expression. The core of this expression matches the functor H of Fiore et al. [8, §4.4].

The monad T_π is now a correct representation for π-calculus behaviour, and for any object $X \in \mathcal{S}et^{\mathcal{I}}$ we can equip $T_\pi(X)$ with the six required operations to make it a π-algebra $Pi(X)$. The most interesting case is new; this is defined recursively by cases, using the equations from §§3.3 and 3.5, and following essentially the pattern of [36] and [8, Table 4].

We thus obtain the desired free functor $Pi : \mathcal{S}et^{\mathcal{I}} \to \mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$, and hence a supply of concrete π-algebras. This completes the adjunction $Pi \dashv U$, with monad $U \circ Pi$ being T_π. What is more, the adjunction is monadic, so that $\mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$ is equivalent to the category of algebras for the monad T_π. To summarise:

Theorem 3.

(i) *The forgetful functor $U : \mathcal{PI}(\mathcal{S}et^{\mathcal{I}}) \to \mathcal{S}et^{\mathcal{I}}$ has a left adjoint Pi giving a free π-algebra $Pi(X)$ over any $X \in \mathcal{S}et^{\mathcal{I}}$.*

(ii) *The comparison functor from $\mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$ to T_π-Alg is an equivalence of categories.*

Proof (sketch).

(i) Once we have an explicit form for Pi, it only remains to check that $Pi(X)$ is initial among π-algebras over X. Given any π-algebra A with $X \to UA$ in $\mathcal{S}et^{\mathcal{I}}$, we must extend this to an algebra map $Pi(X) \to A$. The extension is uniquely determined by the fact that every element of $Pi(X)$ can be generated from X using operations from the theory of π.

(ii) We apply Beck's theorem to show that the adjunction is monadic. The development closely follows Power's in [29, §4], specialised to the case at hand. There is some new work to take account of the two closed structures, which is done using the properties of the function spaces $N \multimap X$ and X^N presented in §2.2. □

4.3 Fully-Abstract π-algebras

The interpretation in §4.1 of π-calculus terms in an arbitrary π-algebra is not altogether compositional, in that we expand out parallel processes. If we specialise to the initial free π-algebra $Pi(0)$ then we can do better.

Proposition 4. *Writing $P \in \mathcal{S}et^{\mathcal{I}}$ for the carrier object of $Pi(0)$, there is a map $par : P^2 \to P$ in $\mathcal{S}et^{\mathcal{I}}$ such that for all finite π-calculus processes P, Q:*

$$[\![P \mid Q]\!]_{Pi(0)} = par([\![P]\!]_{Pi(0)}, [\![Q]\!]_{Pi(0)}) \ .$$

Using par instead of the expansion rule then gives a purely compositional presentation of the denotational semantics in $Pi(0)$ for finite π-calculus processes.

Proof. We decompose par as a sum of interleaving merge and synchronization, and then define each of these recursively by cases on the expansion (1) of $Pi(0)$ — where the base case uses the fact that $\mathcal{D}yn(0)$ is empty. This is the procedure known from existing denotational models, such as [36, §3.2] and [8, §4.6]. Note that par is, as expected, not a map of π-algebras. □

This semantics in $Pi(0)$ is in fact isomorphic to the fully-abstract model described by Fiore et al. in [8, Thm 6.4]. We can extend their analysis to all free π-algebras.

Theorem 5. *For any object $X \in \mathcal{S}et^{\mathcal{I}}$, the free π-algebra $Pi(X)$ is fully abstract for (strong, late) bisimulation congruence. For all finite π-calculus processes P, Q:*

$$P \approx Q \quad \Longleftrightarrow \quad Pi(X) \models P = Q$$

and hence also:

$$P \approx Q \quad \Longleftrightarrow \quad \mathcal{S}et^{\mathcal{I}} \models P = Q \ .$$

Proof. The forward direction is Prop. 2, and the reverse direction for $Pi(0)$ comes from the full abstraction result of [8]. We lift this to general $Pi(X)$ by factoring the interpretation $[\![-]\!]_{Pi(X)}$ as $[\![-]\!]_{Pi(0)}$ followed by the monomorphism $Pi(0) \rightarrowtail Pi(X)$. □

4.4 Monads and Effects for π

The operations and equations in the theory of π fit very well with a process-calculus view of concurrency. However, the monad T_π of (1) is also a "computational" monad in the style of Moggi, and gives a programming language semantics of mobile communicating systems. The operations of §3 then induce corresponding generic effects [28]:

$$\begin{array}{rl@{\qquad}rl}
\mathit{choice} : & A^2 \longrightarrow A & \mathsf{arb} : & 1 \longrightarrow \mathsf{T2} \\
\mathit{nil} : & 1 \longrightarrow A & \mathsf{deadlock} : & 1 \longrightarrow \mathsf{T0} \\
\mathit{out} : & A \longrightarrow A^{N \times N} & \mathsf{send} : & \mathsf{N} \times \mathsf{N} \longrightarrow \mathsf{T1} \\
\mathit{in} : & A^N \longrightarrow A^N & \mathsf{receive} : & \mathsf{N} \longrightarrow \mathsf{TN} \\
\mathit{tau} : & A \longrightarrow A & \mathsf{skip} : & 1 \longrightarrow \mathsf{T1} \\
\mathit{new} : & \delta A \longrightarrow A & \mathsf{fresh} : & 1 \longrightarrow \mathsf{TN}
\end{array}$$

For example, $\mathsf{receive}(\mathsf{c})$ fetches a value from channel c, and $\mathsf{fresh}()$ returns a newly allocated channel. In a suitable computational metalanguage these give a semantics for programing languages that combine higher-order functions with communicating concurrency. Alternatively, they can be used just as they stand in a language like Haskell that explicitly handles computational monads: $\mathsf{do}\{\mathsf{x} \leftarrow \mathsf{receive}(\mathsf{c}); \mathsf{send}(\mathsf{c}', \mathsf{x})\}$.

5 Extensions and Further Work

In this paper we have examined only finite π-calculus processes. We propose to give algebras for the full π-calculus, with replication and recursion, by introducing order structure with models in $\mathcal{C}po^{\mathcal{I}}$. Plotkin and Power have already investigated $\mathcal{C}po$-enrichment in work on effects for PCF: in particular, taking the least upper bound of ω-chains is then an algebraic operation of (countable) arity. Our target is the existing domain models in $\mathcal{C}po^{\mathcal{I}}$, noting that Fiore et al. give a method for lifting full abstraction in $\mathcal{S}et^{\mathcal{I}}$ up to $\mathcal{C}po^{\mathcal{I}}$.

Order enrichment also offers the possibility of inequations in theories. For the $choice$ operation these can distinguish between upper, lower and convex powerdomains, and we conjecture that such theories for π could characterize Hennessy's fully-abstract models for must and may-testing [10].

Alternative calculi like asynchronous π and πI can be treated by changing the arity of the out operation; process passing and higher-order π seem much more challenging. For different kinds of equivalence, we can follow existing models by varying arities and translation details: this is enough to capture early bisimulation congruence, early/late bisimilarity (not congruences), and bisimilarity up to name constraints. More interesting, though, is the possibility to leave the operations for π untouched and instead adjust only the equations. For example, we might add the characteristic EARLY equation of [23, §9.1] to the π-theory, and then compare this to the explicit model of early bisimulation congruence in [7]. The same approach applies to open bisimilarity and weak bisimulations, known to be challenging for categorical models: Parrow sets out equational axiomatizations for all these in [23, §9], and we now need to explore the algebraic theories they generate.

Pitts and others have championed *nominal sets* and Fraenkel-Mostowski set theory as a foundation for reasoning with names [9, 24, 34]. If we move from $\mathcal{S}et^{\mathcal{I}}$ to its full subcategory of pullback-preserving functors then we have the Schanuel topos, which models FM set theory. As noted earlier, all of our constructions lie within this, and we conjecture that our π-calculus models are examples of universal algebra within FM set theory (given first an investigation of what that is).

Prop. 4 presented an internal par for $Pi(0)$, giving a fully compositional interpretation for the π-calculus. In fact we can define an internal par_μ for any free π-algebra $Pi(X)$, given an associative and commutative multiplication $\mu : X \times X \to X$. These non-initial free algebras are (fully-abstract) models for implementations of the π-calculus over a set of basic processes. For example, $Pi(1)$ models the π-calculus with an extra process "✓" marking completion, which extends the programming language interpretation of §4.4 with a semantics for terminating threads and thread rendezvous.

More generally, the full range of π-algebras in $\mathcal{PI}(\mathcal{S}et^{\mathcal{I}})$ may be useful to model applications of the π-calculus with domain-specific terms, equations and processes. There are many such ad-hoc extensions, notably those brought together by Abadi and Fournet under the banner of *applied* π [1].

In ongoing work, Plotkin has given a construction for modal logics from algebraic theories. Applying this to the theory of π gives a modal logic for the π-calculus up to bisimulation congruence. This can represent Hennessy-Milner logic, and also has

modalities for choice and name creation; though no "spatial" modality for parallel composition.

We can extend our notion of π-algebra to other categories $\mathcal{C}$, enriched over $\mathcal{S}et^{\mathcal{I}}$. However, we do not yet have conditions for the existence of free algebras, or for full abstraction, in general $\mathcal{C}$. This would require further investigation of the properties of algebras enriched over a doubly closed structure, as in $\mathcal{S}et^{\mathcal{I}}$.

An alternative path, following a suggestion of Fiore, is to give a theory of name testing that exhibits $\mathcal{S}et^{\mathcal{I}}$ as monadic over $\mathcal{S}et^{\mathcal{F}}$, where $\mathcal{F}$ is the category of finite name sets and all maps. We have a candidate theory, and conjecture that in combination with our existing theory of π, this would allow us to generate algebraic models of π in $\mathcal{S}et^{\mathcal{F}}$ using only cartesian closed structure.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Conf. Rec. POPL 2001*, pages 104–115. ACM Press, 2001.

[2] G. L. Cattani and P. Sewell. Models for name-passing processes: Interleaving and causal. *Inf. Comput.*, 190(2):136–178, 2004.

[3] G. L. Cattani, I. Stark, and G. Winskel. Presheaf models for the π-calculus. In *Proc. CTCS '97*, LNCS 1290, pages 106–126. Springer-Verlag, 1997.

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

[5] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proc. LICS '99*, pages 193–202. IEEE Comp. Soc. Press, 1999.

[6] M. Fiore and S. Staton. Comparing operational models of name-passing process calculi. In *Proc. CMCS 2004*, ENTCS 106, pages 91–104. Elsevier, 2004.

[7] M. Fiore and D. Turi. Semantics of name and value passing. In *Proc. LICS 2001*, pages 93–104. IEEE Comp. Soc. Press, 2001.

[8] M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the π-calculus. *Inf. Comput.*, 179(1):76–117, 2002.

[9] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3–5):341–363, 2001.

[10] M. Hennessy. A fully abstract denotational semantics for the π-calculus. *Theor. Comput. Sci.*, 278(1–2):53–89, 2002.

[11] J. M. E. Hyland, G. Plotkin, and A. J. Power. Combining computational effects: Commutativity and sum. In *Proc. TCS 2002*, pages 474–484. Kluwer, 2002.

[12] J. M. E. Hyland, G. Plotkin, and A. J. Power. Combining effects: Sum and tensor. To appear, 2004.

[13] M. P. Jones and L. Duponcheel. Composing monads. Research Report YALEU/DCS/RR-1004, Yale University Department of Computer Science, 1993.

[14] G. M. Kelly and A. J. Power. Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads. *J. Pure Appl. Algebra*, 89:163–179, 1993.

[15] S. Liang, P. Hudak, and M. P. Jones. Monad transformers and modular interpreters. In *Conf. Rec. POPL '95*, pages 333–343. ACM Press, 1995.

[16] R. Milner. *Communicating and Mobile Systems: The Pi-Calculus*. CUP, 1999.

[17] E. Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Laboratory for Foundations of Computer Science, University of Edinburgh, 1990.

[18] E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

[19] J. Newburn. All about monads, v1.1.0. `http://www.nomaware.com/monads`.
[20] P. W. O'Hearn and R. D. Tennent. Parametricity and local variables. *J. ACM*, 42(3):658–709, 1995. Reprinted in [21].
[21] P. W. O'Hearn and R. D. Tennent, editors. *Algol-like Languages*. Birkhauser, 1996.
[22] F. J. Oles. Functor categories and store shapes. Chapter 11 of [21].
[23] J. Parrow. An introduction to the π-calculus. In *Handbook of Process Algebra*, pages 479–543. Elsevier, 2001.
[24] A. M. Pitts. Nominal logic, a first order theory of names and binding. *Inf. Comput.*, 186:165–193, 2003. Errata, Sept. 2004.
[25] A. M. Pitts and I. Stark. Observable properties of higher order functions that dynamically create local names, or: What's *new*? In *Proc. MFCS '93*, LNCS 711, pages 122–141. Springer-Verlag, 1993.
[26] G. Plotkin and A. J. Power. Computational effects and operations: An overview. Submitted for publication, 2002.
[27] G. Plotkin and A. J. Power. Notions of computation determine monads. In *Proc. FoSSaCS 2002*, LNCS 2303, pages 342–356. Springer-Verlag, 2002. Erratum, Aug. 2002.
[28] G. Plotkin and A. J. Power. Algebraic operations and generic effects. *Appl. Categ. Struct.*, 11(1):69–94, 2003.
[29] A. J. Power. Enriched Lawvere theories. *Theory Appl. Categ.*, 6(7):83–93, 1999.
[30] J. C. Reynolds. The essence of Algol. In *Proc. 1981 Int. Symp. on Algorithmic Languages*, pages 345–372. North Holland, 1981. Reprinted in [21].
[31] E. Robinson. Variations on algebra: Monadicity and generalisations of equational theories. *Formal Asp. Comput.*, 13(3–5):308–326, 2002.
[32] D. Sangiorgi and D. Walker. *The π-Calculus: A Theory of Mobile Processes*. CUP, 2001.
[33] U. Schöpp and I. Stark. A dependent type theory with names and binding. In *Proc. CSL 2004*, LNCS 3210, pages 235–249. Springer-Verlag, 2004.
[34] M. R. Shinwell, A. M. Pitts, and M. J. Gabbay. FreshML: Programming with binders made simple. In *Proc. ICFP 2003*, pages 263–274. ACM Press, 2003. Erratum, May 2004.
[35] I. Stark. Categorical models for local names. *LISP Symb. Comput.*, 9(1):77–107, 1996.
[36] I. Stark. A fully abstract domain model for the π-calculus. In *Proc. LICS '96*, pages 36–42. IEEE Comp. Soc. Press, 1996.
[37] P. Wadler and D. King. Combining monads. In *Proc. 1992 Glasgow Workshop on Functional Programming*, pages 134–143. Springer-Verlag, 1993.