HOPLA—A Higher-Order Process Language

Mikkel Nygaard[1] and Glynn Winskel[2]

[1] BRICS*, University of Aarhus
[2] Computer Laboratory, University of Cambridge

Abstract. A small but powerful language for higher-order nondeterministic processes is introduced. Its roots in a linear domain theory for concurrency are sketched though for the most part it lends itself to a more operational account. The language can be viewed as an extension of the lambda calculus with a "prefixed sum", in which types express the form of computation path of which a process is capable. Its operational semantics, bisimulation, congruence properties and expressive power are explored; in particular, it is shown how it can directly encode process languages such as CCS, CCS with process passing, and mobile ambients with public names.

1 Introduction

We present an economic yet expressive language for higher-order nondeterministic processes that we discovered recently in developing a domain theory for concurrent processes. The language can be given a straightforward operational semantics, and it is this more operational account we concentrate on in the paper.

The language is typed. The type of a process describes the possible (shapes of) computation paths (or runs) the process can perform. Computation paths may consist simply of sequences of actions but they may also represent the input-output behaviour of a process. A typing judgement

$$x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q}$$

means that a process t yields computation paths in $\mathbb{Q}$ once processes with computation paths in $\mathbb{P}_1, \ldots, \mathbb{P}_k$ are assigned to the variables $x_1, \ldots, x_k$ respectively. The types $\mathbb{P}$ are built using a "prefixed sum," products, function spaces and recursive definition. It is notable that although we can express many kinds of concurrent processes in the language, the language itself does not have many features typical of process calculi built-in, beyond that of a nondeterministic sum and prefix operations.

In general, the language allows processes to be copied and discarded arbitrarily. In particular, the language will allow us to write terms which take a process as argument, copy it and then set those copies in parallel composition with each

---

* Basic Research in Computer Science (`www.brics.dk`), funded by the Danish National Research Foundation.

other. However some operations are by nature *linear* in certain arguments. Linearity, detected in the denotational semantics, translates into the property of preserving nondeterministic sums.

The operations associated with the prefix-sum type constructor are central to the expressiveness of the language. A prefix-sum type has the form $\Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha$; it describes computation paths in which first an action $\beta \in A$ is performed before resuming as a computation path in $\mathbb{P}_\beta$. The association between an initial action β and ensuing type $\mathbb{P}_\beta$ is deliberate; performing an action should lead to a unique type. The prefix sum is associated with *prefix operations* taking a process t of type $\mathbb{P}_\beta$ to $\beta.t$ of type $\Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha$, as well as a *prefix match* $[u > \beta.x \Rightarrow t]$, where u has prefix-sum type, x has type $\mathbb{P}_\beta$ and t generally involves the variable x. The term $[u > \beta.x \Rightarrow t]$ matches u against the pattern $\beta.x$ and passes the results of successful matches for x on to t. More precisely, prefix match is linear in the argument u so that the possibly multiple results of successful matches are nondeterministically summed together.

The language supports an operational semantics and bisimulation. It is expressive and, as examples, it is easy to translate CCS, a form of CCS with process passing, and mobile ambients with public names into the language. The translated languages inherit bisimulation. Linearity plays a role in deriving traditional expansion laws for the translations.

The reader may like to look ahead to the operational treatment of the language in Sects. 3–6 for a fuller description. These sections are essentially self-contained, with references to the account in Sect. 2 of the denotational semantics of the language, and how it arose from a process model of Girard's linear logic, only to highlight the correspondence to the operational semantics.

2 Denotational semantics

2.1 Presheaf Models

Let $\mathbb{P}$ be a small category. The category of presheaves over $\mathbb{P}$, written $\widehat{\mathbb{P}}$, is the category $[\mathbb{P}^{\mathrm{op}}, \mathbf{Set}]$ with objects the functors from $\mathbb{P}^{\mathrm{op}}$ (the opposite category) to the category of sets and functions, and maps the natural transformations between them. In our applications, the category $\mathbb{P}$ is thought of as consisting of computation-path shapes where a map $e : p \to q$ expresses how the path p is extended to the path q. A presheaf $X \in \widehat{\mathbb{P}}$ specifies for a typical path p the set $X(p)$ of computation paths of shape p, and acts on $e : p \to q$ to give a function $X(e)$ saying how paths of shape q restrict to paths of shape p. In this way a presheaf can model the nondeterministic branching of a process. For more information of presheaf models, see [14, 4].

A presheaf category has all colimits and so in particular all sums (coproducts); for any set I, the sum $\Sigma_{i \in I} X_i$ of presheaves X_i over $\mathbb{P}$ has a contribution $\Sigma_{i \in I} X_i(p)$, the disjoint union of sets, at $p \in \mathbb{P}$. The empty sum is the presheaf $\varnothing$ with empty contribution at each p. In process terms, a sum of presheaves represents a nondeterministic sum of processes.

2.2 A Linear Category of Presheaf Models

Because the presheaf category $\widehat{\mathbb{P}}$ is characterised abstractly as the free colimit completion of $\mathbb{P}$ we expect colimit-preserving functors to be useful. Define the category **Lin** to consist of small categories $\mathbb{P}, \mathbb{Q}, \ldots$ with maps $F : \mathbb{P} \to \mathbb{Q}$ the colimit-preserving functors $F : \widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$. **Lin** can be understood as a categorical model of classical linear logic, with the involution of linear logic, $\mathbb{P}^{\perp}$, given by $\mathbb{P}^{\mathrm{op}}$. The tensor product of $\mathbb{P}$ and $\mathbb{Q}$ is given by the product of small categories, $\mathbb{P} \times \mathbb{Q}$, and the function space from $\mathbb{P}$ to $\mathbb{Q}$ by $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$. On objects $\mathbb{P}$ and $\mathbb{Q}$, products (written $\mathbb{P} \,\&\, \mathbb{Q}$) and coproducts are both given by $\mathbb{P} + \mathbb{Q}$, the sum of categories $\mathbb{P}$ and $\mathbb{Q}$; the empty category $\mathbb{O}$ is a zero object. While rich in structure, **Lin** does not support all operations associated with process languages; since all maps are linear (i.e.. colimit-preserving), when applied to the inactive process $\varnothing$, the result will be $\varnothing$. So, in particular, a prefix operation cannot be interpreted as a map of **Lin**.

There are (at least) two reasonable responses to this: One is to move to a model of affine-linear logic where maps are allowed to ignore their arguments in giving non-trivial output. One such model is the category **Aff**, where maps are connected-colimit preserving functors. In [18] we used the category **Aff** to interpret an expressive affine-linear process language. Unfortunately, its operational semantics is proving difficult, and has not yet been extended to higher order.

This leads us to consider a second answer. Following the discipline of linear logic, suitable nonlinear maps are obtained as linear maps whose domain is under an exponential. As for the exponential ! of linear logic, there are many possible choices—see [18]. One is to interpret $!\mathbb{P}$ as the finite-colimit completion of $\mathbb{P}$. With this understanding of $!\mathbb{P}$, it can be shown that $\widehat{\mathbb{P}}$ with the inclusion functor $!\mathbb{P} \to \widehat{\mathbb{P}}$ is the free filtered colimit completion of $!\mathbb{P}$—see [15]. It follows that maps $!\mathbb{P} \to \mathbb{Q}$ in **Lin** correspond, to within isomorphism, to continuous (i.e., filtered colimit preserving) functors $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$.

2.3 A Cartesian Closed Category of Presheaf Models

Define **Cts** to be the category consisting of small categories $\mathbb{P}, \mathbb{Q}, \ldots$ as objects and morphisms $F : \mathbb{P} \to \mathbb{Q}$ the continuous functors $F : \widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$; they compose as functors. Clearly **Lin** is a subcategory of **Cts**, one which shares the same objects. We have

$$\mathbf{Cts}(\mathbb{P}, \mathbb{Q}) \simeq \mathbf{Lin}(!\mathbb{P}, \mathbb{Q})$$

for all small categories $\mathbb{P}, \mathbb{Q}$. The category **Cts** is the coKleisli category of the comonad based on finite-colimit completions.[3] The unit of the corresponding adjunction is given by maps $\mathit{copy} : \mathbb{P} \to !\mathbb{P}$ of **Cts**, used to interpret prefixing below. We can easily characterize those maps in **Cts** which are in **Lin**:

Proposition 1. *Suppose $F : \widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ is a functor which preserves filtered colimits. Then, F preserves all colimits iff F preserves finite coproducts.*

---

[3] We are glossing over 2-category subtleties as ! is really a pseudo functor.

In other words a continuous map is linear iff it preserves sums.

There is an isomorphism

$$!(\mathbb{P} \mathbin{\&} \mathbb{Q}) \cong {!\mathbb{P}} \times {!\mathbb{Q}}$$

making **Cts** cartesian closed; this immediately allows us to interpret the simply-typed lambda calculus with pairing in **Cts**. Products $\mathbb{P} \mathbin{\&} \mathbb{Q}$ in **Cts** are given as in **Lin** but now viewing the projections as continuous functors. The function space $\mathbb{P} \to \mathbb{Q}$ is given by $(!\mathbb{P})^{\mathrm{op}} \times \mathbb{Q}$.

The category **Cts** does not have coproducts. However, we can build a useful sum in **Cts** with the help of the coproduct of **Lin** and !. Let $(\mathbb{P}_\alpha)_{\alpha \in A}$ be a family of small categories. Their *prefixed sum*,

$$\Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha \enspace ,$$

is based on the coproduct in **Lin** given by $\Sigma_{\alpha \in A} !\mathbb{P}_\alpha$ with corresponding injections $in_\beta : !\mathbb{P}_\beta \to \Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha$, for $\beta \in A$. The *injections*

$$\beta.(-) : \mathbb{P}_\beta \to \Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha$$

in **Cts**, for $\beta \in A$, are defined to be the compositions $\beta.(-) = in_\beta \circ copy$. As the notation suggests, $\beta.(-)$ is used to interpret prefixing with β.

The construction above satisfies a property analogous to the universal property of a coproduct. Suppose $F_\alpha : \mathbb{P}_\alpha \to \mathbb{Q}$ are maps in **Cts** for all $\alpha \in A$. Then, *within* **Lin**, we find a mediating map

$$F : \Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha \to \mathbb{Q}$$

determined to within isomorphism such that

$$F \circ \alpha.(-) \cong F_\alpha$$

for all $\alpha \in A$. Since **Lin** is a subcategory of **Cts**, the mediating map F also belongs to **Cts**, but here it is not uniquely determined, not even up to isomorphism. Therefore, the prefixed sum is not a coproduct in **Cts**, but the linearity of the mediating map is just what we need for interpreting prefix match. Consider a prefix match term $[u > \beta.x \Rightarrow t]$ where t denotes a map $F_\beta : \mathbb{P}_\beta \to \mathbb{Q}$. We interpret it as the mediating map obtained for F_β together with constantly $\varnothing$ maps $F_\alpha : \mathbb{P}_\alpha \to \mathbb{Q}$ for $\alpha \in A$ different from β, applied to the denotation of u.

2.4 Rooted Presheaves and Operational Semantics

The category $!\mathbb{P}$ has an initial element $\bot$, given by the empty colimit, and a presheaf over $!\mathbb{P}$ is called *rooted* if it has a singleton contribution at $\bot$—see [14]. As an example, the denotation of $\beta.t$ with t closed is rooted. We can decompose any presheaf X over $!\mathbb{P}$ as a sum of rooted presheaves $\Sigma_{i \in X(\bot)} X_i$, each X_i a presheaf over $!\mathbb{P}$. This is the key to the correspondence between the denotational semantics and the operational semantics of Sect. 4. Judgements $t \xrightarrow{\beta} t'$, with t of prefix-sum type $\Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha$ and $\beta \in A$, in the operational semantics will express that $copy[\![t']\!]$ is a rooted component of $[\![t]\!]$ restricted to $!\mathbb{P}_\beta$. In fact, derivations of transitions $t \xrightarrow{\beta} t'$ will be in 1-1 correspondence with such components.

3 A Higher-Order Process Language

The types of the language are given by the grammar

$$\mathbb{P}, \mathbb{Q} ::= \Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha \mid \mathbb{P} \to \mathbb{Q} \mid \mathbb{P} \,\&\, \mathbb{Q} \mid P \mid \mu_j \vec{P}.\vec{\mathbb{P}} \ .$$

P is drawn from a set of type variables used in defining recursive types; $\mu_j \vec{P}.\vec{\mathbb{P}}$ abbreviates $\mu_j P_1, \ldots, P_k.(\mathbb{P}_1, \ldots, \mathbb{P}_k)$ and is interpreted as the j-component, for $1 \leq j \leq k$, of the "least" solution (given by a suitable ω-colimit in the category of small categories and functors) to the defining equations $P_1 = \mathbb{P}_1, \ldots, P_k = \mathbb{P}_k$, in which the expressions $\mathbb{P}_1, \ldots, \mathbb{P}_k$ may contain the P_j's. We shall write $\mu \vec{P}.\vec{\mathbb{P}}$ as an abbreviation for the k-tuple with j-component $\mu_j \vec{P}.\vec{\mathbb{P}}$.

The constructions of **Cts** form the basis of a syntax of terms:

$$t, u ::= x \mid \mathit{rec}\, x.t \mid \Sigma_{i \in I} t_i \mid \alpha.t \mid [u > \alpha.x \Rightarrow t] \mid \lambda x.t \mid t\ u \mid (t, u) \mid \mathit{fst}\, t \mid \mathit{snd}\, t$$

In a nondeterministic sum term, $\Sigma_{i \in I} t_i$, the indexing set I may be an arbitrary set; we write $t_1 + \cdots + t_k$ for a typical finite sum and $\varnothing$ when I is empty. The variable x in the match term $[u > \alpha.x \Rightarrow t]$ is a binding occurrence and so binds later occurrences of the variable in the body t. We shall take for granted an understanding of free and bound variables, and substitution on raw terms.

Let $\mathbb{P}_1, \ldots, \mathbb{P}_k, \mathbb{Q}$ be closed type expressions and assume that the variables $x_1, \ldots, x_k$ are distinct. A syntactic judgement $x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \vdash t : \mathbb{Q}$ can be interpreted as a map $\mathbb{P}_1 \,\&\, \cdots \,\&\, \mathbb{P}_k \to \mathbb{Q}$ in **Cts**. We let Γ range over environment lists $x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k$, which we may treat as finite maps from variables to closed type expressions. The term formation rules are:

$$\frac{\Gamma(x) = \mathbb{P}}{\Gamma \vdash x : \mathbb{P}} \qquad \frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{P}}{\Gamma \vdash \mathit{rec}\, x.t : \mathbb{P}} \qquad \frac{\Gamma \vdash t_j : \mathbb{P} \quad \text{all } j \in I}{\Gamma \vdash \Sigma_{i \in I} t_i : \mathbb{P}}$$

$$\frac{\Gamma \vdash t : \mathbb{P}_\beta \quad \text{where } \beta \in A}{\Gamma \vdash \beta.t : \Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha} \qquad \frac{\Gamma \vdash u : \Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha \quad \Gamma, x : \mathbb{P}_\beta \vdash t : \mathbb{Q} \quad \text{where } \beta \in A}{\Gamma \vdash [u > \beta.x \Rightarrow t] : \mathbb{Q}}$$

$$\frac{\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}}{\Gamma \vdash \lambda x.t : \mathbb{P} \to \mathbb{Q}} \qquad \frac{\Gamma \vdash t : \mathbb{P} \to \mathbb{Q} \quad \Gamma \vdash u : \mathbb{P}}{\Gamma \vdash t\ u : \mathbb{Q}}$$

$$\frac{\Gamma \vdash t : \mathbb{P} \quad \Gamma \vdash u : \mathbb{Q}}{\Gamma \vdash (t, u) : \mathbb{P} \,\&\, \mathbb{Q}} \qquad \frac{\Gamma \vdash t : \mathbb{P} \,\&\, \mathbb{Q}}{\Gamma \vdash \mathit{fst}\, t : \mathbb{P}} \qquad \frac{\Gamma \vdash t : \mathbb{P} \,\&\, \mathbb{Q}}{\Gamma \vdash \mathit{snd}\, t : \mathbb{Q}}$$

$$\frac{\Gamma \vdash t : \mathbb{P}_j[\mu \vec{P}.\vec{\mathbb{P}}/\vec{P}]}{\Gamma \vdash t : \mu_j \vec{P}.\vec{\mathbb{P}}} \qquad \frac{\Gamma \vdash t : \mu_j \vec{P}.\vec{\mathbb{P}}}{\Gamma \vdash t : \mathbb{P}_j[\mu \vec{P}.\vec{\mathbb{P}}/\vec{P}]}$$

We have a syntactic substitution lemma:

Lemma 2. *Suppose* $\Gamma, x : \mathbb{P} \vdash t : \mathbb{Q}$ *and* $\Gamma \vdash u : \mathbb{P}$. *Then* $\Gamma \vdash t[u/x] : \mathbb{Q}$.

The semantic counterpart essentially says that the denotation of $t[u/x]$ is the functorial composition of the denotations of t and u.

4 Operational Semantics

With actions given by the grammar

$$a ::= \alpha \mid u \mapsto a \mid (a, -) \mid (-, a) \enspace ,$$

the operational rules below define a transition semantics for closed, well-formed terms:

$$\frac{t[\mathit{rec}\, x.t/x] \xrightarrow{a} t'}{\mathit{rec}\, x.t \xrightarrow{a} t'} \qquad \frac{t_j \xrightarrow{a} t'}{\Sigma_{i \in I} t_i \xrightarrow{a} t'}\ j \in I$$

$$\frac{}{\alpha.t \xrightarrow{\alpha} t} \qquad \frac{u \xrightarrow{\alpha} u' \quad t[u'/x] \xrightarrow{a} t'}{[u > \alpha.x \Rightarrow t] \xrightarrow{a} t'}$$

$$\frac{t[u/x] \xrightarrow{a} t'}{\lambda x.t \xrightarrow{u \mapsto a} t'} \qquad \frac{t \xrightarrow{u \mapsto a} t'}{t\ u \xrightarrow{a} t'}$$

$$\frac{t \xrightarrow{a} t'}{(t, u) \xrightarrow{(a,-)} t'} \qquad \frac{u \xrightarrow{a} u'}{(t, u) \xrightarrow{(-,a)} u'} \qquad \frac{t \xrightarrow{(a,-)} t'}{\mathit{fst}\, t \xrightarrow{a} t'} \qquad \frac{t \xrightarrow{(-,a)} t'}{\mathit{snd}\, t \xrightarrow{a} t'}$$

In the rule for lambda-abstraction, we must have $x : \mathbb{P} \vdash t : \mathbb{Q}$ and $\vdash u : \mathbb{P}$ for some $\mathbb{P}, \mathbb{Q}$.

To show that the rules are type-correct, we assign types to actions a using a judgement of the form $\mathbb{P} : a : \mathbb{P}'$. Intuitively, after performing the action a, what remains of a computation path in $\mathbb{P}$ is a computation path in $\mathbb{P}'$. For $\beta \in A$ we take $\Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha : \beta : \mathbb{P}_\beta$ and inductively

$$\frac{\vdash u : \mathbb{P} \quad \mathbb{Q} : a : \mathbb{P}'}{\mathbb{P} \to \mathbb{Q} : u \mapsto a : \mathbb{P}'} \qquad \frac{\mathbb{P} : a : \mathbb{P}'}{\mathbb{P} \,\&\, \mathbb{Q} : (a, -) : \mathbb{P}'} \qquad \frac{\mathbb{P}_j[\mu \vec{P}.\vec{\mathbb{P}}/\vec{P}] : a : \mathbb{P}'}{\mu_j \vec{P}.\vec{\mathbb{P}} : a : \mathbb{P}'}$$

—with a symmetric rule for $(-, a)$. Notice that in $\mathbb{P} : a : \mathbb{P}'$, the type $\mathbb{P}'$ is unique given $\mathbb{P}$ and a.

Proposition 3. *Suppose* $\vdash t : \mathbb{P}$. *If* $t \xrightarrow{a} t'$ *then* $\mathbb{P} : a : \mathbb{P}'$ *and* $\vdash t' : \mathbb{P}'$.

We interpret $\mathbb{P} : a : \mathbb{P}'$ as a map $\mathbb{P} \to\, !\mathbb{P}'$ of **Lin** by letting the judgement for β above denote restriction to $!\mathbb{P}_\beta$, and inductively interpreting $u \mapsto a, (a, -), (-, a)$ as the denotation of a precomposed with the map given by application to $[\![u]\!]$, and first and second projections, respectively. The rules are then sound in the sense that if $t \xrightarrow{a} t'$ then (identifying a term with its denotation) *copy* t' is a rooted component of $a(t)$. In fact, there is a 1-1 correspondence between derivations with conclusion $t \xrightarrow{a} t'$, for some t', and the rooted components of $a(t)$, so the rules are also complete.

As a side-remark, the operational rules incorporate evaluation in the following sense: Let values v be given by $v ::= \alpha.t \mid \lambda x.t \mid (t, u)$. Then we can define a nondeterministic evaluation-relation $\Downarrow$ such that

Proposition 4. *If* d *is a derivation of* $t \xrightarrow{a} t'$, *then there is a value* v *such that* $t \Downarrow v$ *and* $v \xrightarrow{a} t'$ *by a subderivation of* d.

5 Equivalences

After introducing some notation regarding relations, we explore four standard notions of equivalence for our language. A relation R between typing judgements is said to respect types if, whenever R relates

$$\Gamma_1 \vdash t_1 : \mathbb{P}_1 \quad \text{and} \quad \Gamma_2 \vdash t_2 : \mathbb{P}_2 \ ,$$

we have syntactic identities $\Gamma_1 \equiv \Gamma_2$ and $\mathbb{P}_1 \equiv \mathbb{P}_2$. Below, all our relations will respect types, so we'll often suppress the typing information, writing just $t_1 \; R \; t_2$.

Suppose Γ is an environment list

$$x_1 : \mathbb{P}_1, \ldots, x_k : \mathbb{P}_k \ .$$

A Γ-closure is a substitution $[\vec{u}/\vec{x}]$ such that for $1 \leq j \leq k$, $\vdash u_j : \mathbb{P}_j$. If R relates only closed terms, we write R° for its open extension, relating $\Gamma \vdash t_1 : \mathbb{P}$ and $\Gamma \vdash t_2 : \mathbb{P}$ if for all Γ-closures $[\vec{u}/\vec{x}]$ we have $t_1[\vec{u}/\vec{x}] \; R \; t_2[\vec{u}/\vec{x}]$.

We'll write R_{c} for the restriction of a type-respecting relation R to closed terms.

For a type-respecting relation R we write R also for the induced relation on actions, given as the least congruence on actions so that $u_1 \mapsto a \; R \; u_2 \mapsto a$ if $u_1 \; R \; u_2$.

5.1 Bisimulation

A type-respecting relation R on closed terms is a *bisimulation* [19, 21] if the following holds. If $t_1 \; R \; t_2$, then

1. if $t_1 \xrightarrow{a} t_1'$, then $t_2 \xrightarrow{a} t_2'$ for some t_2' such that $t_1' \; R \; t_2'$;
2. if $t_2 \xrightarrow{a} t_2'$, then $t_1 \xrightarrow{a} t_1'$ for some t_1' such that $t_1' \; R \; t_2'$.

Bisimilarity, $\sim$, is the largest bisimulation.

Theorem 5. *Bisimilarity is a congruence.*

Proof. We use Howe's method [11] as adapted to a typed setting by Gordon [9]. In detail, we define the precongruence candidate $\hat{\sim}$ as follows:

$$\frac{x \sim^{\circ} w}{x \mathrel{\hat{\sim}} w} \qquad \frac{t \mathrel{\hat{\sim}} t' \quad \mathit{rec}\, x.t' \sim^{\circ} w}{\mathit{rec}\, x.t \mathrel{\hat{\sim}} w} \qquad \frac{t_j \mathrel{\hat{\sim}} t_j' \quad \text{all } j \in I \quad \Sigma_{i \in I} t_i' \sim^{\circ} w}{\Sigma_{i \in I} t_i \mathrel{\hat{\sim}} w}$$

$$\frac{t \mathrel{\hat{\sim}} t' \quad \alpha.t' \sim^{\circ} w}{\alpha.t \mathrel{\hat{\sim}} w} \qquad \frac{t \mathrel{\hat{\sim}} t' \quad u \mathrel{\hat{\sim}} u' \quad [u' > \alpha.x \Rightarrow t'] \sim^{\circ} w}{[u > \alpha.x \Rightarrow t] \mathrel{\hat{\sim}} w}$$

$$\frac{t \mathrel{\hat{\sim}} t' \quad \lambda x.t' \sim^{\circ} w}{\lambda x.t \mathrel{\hat{\sim}} w} \qquad \frac{t \mathrel{\hat{\sim}} t' \quad u \mathrel{\hat{\sim}} u' \quad t' \, u' \sim^{\circ} w}{t \, u \mathrel{\hat{\sim}} w}$$

$$\frac{t \mathrel{\hat{\sim}} t' \quad u \mathrel{\hat{\sim}} u' \quad (t', u') \sim^{\circ} w}{(t, u) \mathrel{\hat{\sim}} w} \qquad \frac{t \mathrel{\hat{\sim}} t' \quad \mathit{fst}\, t' \sim^{\circ} w}{\mathit{fst}\, t \mathrel{\hat{\sim}} w} \qquad \frac{t \mathrel{\hat{\sim}} t' \quad \mathit{snd}\, t' \sim^{\circ} w}{\mathit{snd}\, t \mathrel{\hat{\sim}} w}$$

Following Howe we now have: (i) $\hat{\sim}$ is reflexive; (ii) $\hat{\sim}$ is operator respecting; (iii) $\sim^{\circ} \subseteq \hat{\sim}$; (iv) if $t \mathrel{\hat{\sim}} t'$ and $t' \sim^{\circ} w$ then $t \mathrel{\hat{\sim}} w$; (v) if $t \mathrel{\hat{\sim}} t'$ and $u \mathrel{\hat{\sim}} u'$ then $t[u/x] \mathrel{\hat{\sim}} t'[u'/x]$ whenever the substitutions are well-formed; (vi) since $\sim$ is an equivalence relation, the transitive closure $\hat{\sim}^{+}$ of $\hat{\sim}$ is symmetric, and therefore, so is $\hat{\sim}^{+}_{\mathsf{c}}$.

Now we just need to show that $\hat{\sim}_{\mathsf{c}}$ is a simulation, because then $\hat{\sim}^{+}_{\mathsf{c}}$ is a bisimulation by (vi), and so $\hat{\sim}^{+}_{\mathsf{c}} \subseteq \sim$. In particular, $\hat{\sim}_{\mathsf{c}} \subseteq \sim$. By (i) and (v), it follows that $\hat{\sim} \subseteq \sim^{\circ}$, and so by (iii), $\hat{\sim} = \sim^{\circ}$. Hence, $\sim$ is a congruence because it is an equivalence relation and by (ii) it is operator respecting.

We prove that $\hat{\sim}_{\mathsf{c}}$ is a simulation by induction on the derivations of the operational semantics and using (iv-v). In fact, we need an induction hypothesis slightly stronger than one might expect:

> if $t_1 \mathrel{\hat{\sim}_{\mathsf{c}}} t_2$ and $t_1 \xrightarrow{a_1} t_1'$, then for all a_2 with $a_1 \mathrel{\hat{\sim}_{\mathsf{c}}} a_2$ we have $t_2 \xrightarrow{a_2} t_2'$ for some t_2' such that $t_1' \mathrel{\hat{\sim}_{\mathsf{c}}} t_2'$.

By (i), $a \mathrel{\hat{\sim}_{\mathsf{c}}} a$ for all actions a, from which it follows that $\hat{\sim}_{\mathsf{c}}$ is a simulation. □

Proposition 6. *The following pairs of closed, well-formed terms are bisimilar:*

$$\begin{array}{lrcl}
\textit{1.} & \mathit{rec}\, x.t & \sim & t[\mathit{rec}\, x.t/x] \\[6pt]
\textit{2.} & [\alpha.u > \alpha.x \Rightarrow t] & \sim & t[u/x] \\
\textit{3.} & [\beta.u > \alpha.x \Rightarrow t] & \sim & \varnothing \quad \textit{if } \alpha \neq \beta \\
\textit{4.} & [\Sigma_{i\in I} u_i > \alpha.x \Rightarrow t] & \sim & \Sigma_{i\in I}[u_i > \alpha.x \Rightarrow t] \\
\textit{5.} & [u > \alpha.x \Rightarrow \Sigma_{i\in I} t_i] & \sim & \Sigma_{i\in I}[u > \alpha.x \Rightarrow t_i] \\[6pt]
\textit{6.} & (\lambda x.t)\; u & \sim & t[u/x] \\
\textit{7.} & \lambda x.(t\; x) & \sim & t \\
\textit{8.} & \lambda x.(\Sigma_{i\in I} t_i) & \sim & \Sigma_{i\in I}(\lambda x.t_i) \\
\textit{9.} & (\Sigma_{i\in I} t_i)\; u & \sim & \Sigma_{i\in I}(t_i\; u) \\[6pt]
\textit{10.} & \mathit{fst}(t,u) & \sim & t \\
\textit{11.} & \mathit{snd}(t,u) & \sim & u \\
\textit{12.} & t & \sim & (\mathit{fst}\, t, \mathit{snd}\, t) \\
\textit{13.} & (\Sigma_{i\in I} t_i, \Sigma_{i\in I} u_i) & \sim & \Sigma_{i\in I}(t_i, u_i) \\
\textit{14.} & \mathit{fst}(\Sigma_{i\in I} t_i) & \sim & \Sigma_{i\in I}(\mathit{fst}\, t_i) \\
\textit{15.} & \mathit{snd}(\Sigma_{i\in I} t_i) & \sim & \Sigma_{i\in I}(\mathit{snd}\, t_i)
\end{array}$$

In each case $t_1 \sim t_2$, we can prove that the identity relation extended by the pair (t_1, t_2) is a bisimulation, so the correspondence is very tight, as is to be expected since in the denotational semantics, we have $[\![t_1]\!] \cong [\![t_2]\!]$.

Proposition 7. *Let t_1, t_2 be closed terms of type $\mathbb{P} \to \mathbb{Q}$. The following are equivalent:*

1. $t_1 \sim t_2$;
2. $t_1\; u \sim t_2\; u$ *for all closed terms u of type $\mathbb{P}$;*
3. $t_1\; u_1 \sim t_2\; u_2$ *for all closed terms $u_1 \sim u_2$ of type $\mathbb{P}$.*

5.2 Applicative Bisimulation

A type-respecting relation R on closed terms is an *applicative bisimulation* [1] if the following holds:

1. If $\vdash t_1 \; R \; t_2 : \Sigma_{\alpha \in A} \alpha.\mathbb{P}_\alpha$, we have
   (a) if $t_1 \xrightarrow{\beta} t'_1$, then $t_2 \xrightarrow{\beta} t'_2$ for some t'_2 such that $\vdash t'_1 \; R \; t'_2 : \mathbb{P}_\beta$;
   (b) if $t_2 \xrightarrow{\beta} t'_2$, then $t_1 \xrightarrow{\beta} t'_1$ for some t'_1 such that $\vdash t'_1 \; R \; t'_2 : \mathbb{P}_\beta$.
2. If $\vdash t_1 \; R \; t_2 : \mathbb{P} \to \mathbb{Q}$ then for all $\vdash u : \mathbb{P}$ we have $\vdash t_1 \; u \; R \; t_2 \; u : \mathbb{Q}$.
3. If $\vdash t_1 \; R \; t_2 : \mathbb{P} \,\&\, \mathbb{Q}$ then $\vdash \mathit{fst}\, t_1 \; R \; \mathit{fst}\, t_2 : \mathbb{P}$ and $\vdash \mathit{snd}\, t_1 \; R \; \mathit{snd}\, t_2 : \mathbb{Q}$.
4. If $\vdash t_1 \; R \; t_2 : \mu_j \vec{P}.\vec{\mathbb{P}}$ then $\vdash t_1 \; R \; t_2 : \mathbb{P}_j[\mu \vec{P}.\vec{\mathbb{P}}/\vec{P}]$.

Applicative bisimilarity, $\sim_A$, is the largest applicative bisimulation. We have

Proposition 8. *Bisimilarity and applicative bisimilarity coincide.*

Proof. Since $\sim$ is a congruence, it follows that $\sim$ is an applicative bisimulation, and so $\sim \subseteq \sim_A$. Conversely, we can show that $\sim_A$ is a bisimulation by structural induction on (typing derivations of) actions a, and so $\sim_A \subseteq \sim$. ☐

5.3 Higher Order Bisimulation

A type-respecting relation R on closed terms is a *higher order bisimulation* [23] if the following holds: If $t_1 \; R \; t_2$, then

1. if $t_1 \xrightarrow{a_1} t'_1$, then $t_2 \xrightarrow{a_2} t'_2$ for some a_2, t'_2 such that $a_1 \; R \; a_2$ and $t'_1 \; R \; t'_2$;
2. if $t_2 \xrightarrow{a_2} t'_2$, then $t_1 \xrightarrow{a_1} t'_1$ for some a_1, t'_1 such that $a_1 \; R \; a_2$ and $t'_1 \; R \; t'_2$.

Higher order bisimilarity, $\sim_H$, is the largest higher order bisimulation.

Proposition 9. *Bisimilarity and higher order bisimilarity coincide.*

Proof. Clearly, bisimilarity is a higher order bisimulation so that $\sim \subseteq \sim_H$. For the converse, we observe that the proof of Theorem 5 goes through if we replace $\sim$ by $\sim_H$, and so $\sim_H$ is a congruence. It then follows by structural induction on actions a that $\sim_H$ is a bisimulation, so that $\sim_H \subseteq \sim$. ☐

5.4 Contextual Equivalence

Let the type $\mathbb{1}$ be given as $\bullet.\mathbb{O}$ where $\mathbb{O}$ is the empty prefixed sum. If $\vdash t : \mathbb{1}$ we'll write $t \xrightarrow{\bullet}$ if there exists a t' such that $t \xrightarrow{\bullet} t'$. Two terms $\Gamma \vdash t_1 : \mathbb{P}$ and $\Gamma \vdash t_2 : \mathbb{P}$ are said to be *contextually equivalent* [16,8] if $C(t_1) \xrightarrow{\bullet}$ iff $C(t_2) \xrightarrow{\bullet}$ for all contexts C such that $C(t_1), C(t_2)$ are closed and have type $\mathbb{1}$.

The following two terms can be shown contextually equivalent:

$$t_1 \equiv \alpha.\varnothing + \alpha.\beta.\varnothing \quad \text{and} \quad t_2 \equiv \alpha.\beta.\varnothing .$$

However, they are clearly not bisimilar, so contextual equivalence fails to take account of the nondeterministic branching of processes.

6 Examples

The higher-order language is quite expressive as the following examples show.

6.1 CCS

As in CCS [21], let N be a set of names and $\bar{N}$ the set of complemented names $\{\bar{n} \mid n \in N\}$. Let l range over labels $L = N \cup \bar{N}$, with complementation extended to L by taking $\bar{\bar{n}} = n$, and let τ be a distinct label. We can then specify a type $\mathbb{P}$ as

$$\mathbb{P} = \tau.\mathbb{P} + \Sigma_{n \in N} n.\mathbb{P} + \Sigma_{n \in N} \bar{n}.\mathbb{P} \enspace .$$

Below, we let α range over $L \cup \{\tau\}$. The terms of CCS can be expressed in the higher-order language as the following terms of type $\mathbb{P}$:

$$\begin{array}{rlrl}
[\![x]\!] \equiv x & & [\![\mathit{rec}\, x.P]\!] \equiv \mathit{rec}\, x.[\![P]\!] \\
[\![\alpha.P]\!] \equiv \alpha.[\![P]\!] & & [\![\Sigma_{i \in I} P_i]\!] \equiv \Sigma_{i \in I} [\![P_i]\!] \\
[\![P|Q]\!] \equiv \mathit{Par}\ [\![P]\!]\ [\![Q]\!] & & [\![P \setminus S]\!] \equiv \mathit{Res}_S\ [\![P]\!] \\
[\![P[f]]\!] \equiv \mathit{Rel}_f\ [\![P]\!] & &
\end{array}$$

Here, $\mathit{Par} : \mathbb{P} \to (\mathbb{P} \to \mathbb{P})$, $\mathit{Res}_S : \mathbb{P} \to \mathbb{P}$, and $\mathit{Rel}_f : \mathbb{P} \to \mathbb{P}$ are abbreviations for the following recursively defined processes:

$$\begin{aligned}
\mathit{Par} \equiv \mathit{rec}\, p.\lambda x.\lambda y.\, & \Sigma_\alpha[x > \alpha.x' \Rightarrow \alpha.(p\ x'\ y)] + \\
& \Sigma_\alpha[y > \alpha.y' \Rightarrow \alpha.(p\ x\ y')] + \\
& \Sigma_l[x > l.x' \Rightarrow [y > \bar{l}.y' \Rightarrow \tau.(p\ x'\ y')]] \\
\mathit{Res}_S \equiv \mathit{rec}\, r.\lambda x.\, & \Sigma_{\alpha \notin (S \cup \bar{S})}[x > \alpha.x' \Rightarrow \alpha.(r\ x')] \\
\mathit{Rel}_f \equiv \mathit{rec}\, r.\lambda x.\, & \Sigma_\alpha[x > \alpha.x' \Rightarrow f(\alpha).(r\ x')]
\end{aligned}$$

Proposition 10. *If* $P \xrightarrow{\alpha} P'$ *is derivable in CCS then* $[\![P]\!] \xrightarrow{\alpha} [\![P']\!]$ *in the higher-order language.*

Conversely, if $[\![P]\!] \xrightarrow{a} t'$ *in the higher-order language, then* $a \equiv \alpha$ *and* $t' \equiv [\![P']\!]$ *for some* α, P' *such that* $P \xrightarrow{\alpha} P'$ *according to CCS.*

It follows that the translations of two CCS terms are bisimilar in the general language iff they are strongly bisimilar in CCS.

We can recover the expansion law for general reasons. Write $P|Q$ for the application $\mathit{Par}\ P\ Q$, where P and Q are terms of type $\mathbb{P}$. Suppose

$$P \sim \Sigma_\alpha \Sigma_{i \in I(\alpha)} \alpha.P_i \quad \text{and} \quad Q \sim \Sigma_\alpha \Sigma_{j \in J(\alpha)} \alpha.Q_j .$$

Using Proposition 6 items 1 and 6, then items 2-4, we get

$$\begin{aligned}
P|Q \sim\ & \Sigma_\alpha[P > \alpha.x' \Rightarrow \alpha.(x'|Q)] + \\
& \Sigma_\alpha[Q > \alpha.y' \Rightarrow \alpha.(P|y')] + \\
& \Sigma_l[P > l.x' \Rightarrow [Q > \bar{l}.y' \Rightarrow \tau.(x'|y')]] \\
\sim\ & \Sigma_\alpha \Sigma_{i \in I(\alpha)} \alpha.(P_i|Q) + \\
& \Sigma_\alpha \Sigma_{j \in J(\alpha)} \alpha.(P|Q_j) + \\
& \Sigma_l \Sigma_{i \in I(l), j \in J(\bar{l})} \tau.(P_i|Q_j) \enspace .
\end{aligned}$$

6.2 Higher-Order CCS

In [10], Hennessy considers a language like CCS but where processes are passed at channels C; the language can be seen as an extension of Thomsen's CHOCS [23]. For a translation into our language, we follow Hennessy in defining types that satisfy the equations

$$\mathbb{P} = \tau.\mathbb{P} + \Sigma_{c \in C} c!.\mathbb{C} + \Sigma_{c \in C} c?.\mathbb{F} \qquad \mathbb{C} = \mathbb{P} \,\&\, \mathbb{P} \qquad \mathbb{F} = \mathbb{P} \to \mathbb{P} \ .$$

We are chiefly interested in the parallel composition of processes, $Par_{\mathbb{P},\mathbb{P}}$ of type $\mathbb{P} \,\&\, \mathbb{P} \to \mathbb{P}$. But parallel composition is really a family of mutually dependent operations also including components such as $Par_{\mathbb{F},\mathbb{C}}$ of type $\mathbb{F} \,\&\, \mathbb{C} \to \mathbb{P}$ to say how abstractions compose in parallel with concretions etc. All these components can be tupled together in a product and parallel composition defined as a simultaneous recursive definition whose component at $\mathbb{P} \,\&\, \mathbb{P} \to \mathbb{P}$ satisfies

$$\begin{aligned} P|Q = {} & \Sigma_\alpha[P > \alpha.x \Rightarrow \alpha.(x|Q)] + \\ & \Sigma_\alpha[Q > \alpha.y \Rightarrow \alpha.(P|y)] + \\ & \Sigma_c[P > c?.f \Rightarrow [Q > c!.p \Rightarrow \tau.((f \ \mathit{fst}\, p)|\, \mathit{snd}\, p)]] + \\ & \Sigma_c[P > c!.p \Rightarrow [Q > c?.f \Rightarrow \tau.(\mathit{snd}\, p|(f \ \mathit{fst}\, p))]] \ , \end{aligned}$$

where, e.g., $P|Q$ abbreviates $Par_{\mathbb{P},\mathbb{P}}\ (P, Q)$. In the summations $c \in C$ and α ranges over $c!, c?, \tau$.

The bisimulation induced on higher-order CCS terms is perhaps the one to be expected; a corresponding bisimulation relation is defined like an applicative bisimulation but restricted to the types of processes $\mathbb{P}$, concretions $\mathbb{C}$, and abstractions $\mathbb{F}$.

6.3 Mobile Ambients with Public Names

We can translate the Ambient Calculus with public names [2] into the higher-order language, following similar lines to the process-passing language above. Assume a fixed set of ambient names $n, m, \ldots \in N$. Following [3], the syntax of ambients is extended beyond processes (P) to include concretions (C) and abstractions (F):

$$\begin{aligned} P &::= \varnothing \mid P|P \mid \mathit{rep}\, P \mid n[P] \mid \mathit{in}\ n.P \mid \mathit{out}\ n.P \mid \mathit{open}\ n!.P \mid \\ &\quad\ \ \tau.P \mid \mathit{mvin}\ n!.C \mid \mathit{mvout}\ n!.C \mid \mathit{open}\ n?.P \mid \mathit{mvin}\ n?.F \mid x \\ C &::= (P, P) \qquad F \ ::= \ \lambda x.P \ . \end{aligned}$$

The notation for actions departs a little from that of [3]. Here some actions are marked with ! and others with ?—active (or inceptive) actions are marked by ! and passive (or receptive) actions by ?. We say actions α and β are *complementary* iff one has the form *open* $n!$ or *mvin* $n!$ while the other is *open* $n?$ or *mvin* $n?$ respectively. Complementary actions can synchronise together to form a τ-action. We adopt a slightly different notation for concretions ((P, R) instead of $\langle P \rangle R$) and abstractions ($\lambda x.P$ instead of $(x)P$) to make their translation into the higher-order language clear.

Types for ambients are given recursively by (n ranges over N):

$$\begin{aligned}\mathbb{P} = {} & \tau.\mathbb{P} + \Sigma_n \mathit{in}\ n.\mathbb{P} + \Sigma_n \mathit{out}\ n.\mathbb{P} + \Sigma_n \mathit{open}\ n!.\mathbb{P} + \Sigma_n \mathit{mvin}\ n!.\mathbb{C} + \\ & \Sigma_n \mathit{mvout}\ n!.\mathbb{C} + \Sigma_n \mathit{open}\ n?.\mathbb{P} + \Sigma_n \mathit{mvin}\ n?.\mathbb{F} \\ \mathbb{C} = {} & \mathbb{P} \,\&\, \mathbb{P} \qquad \mathbb{F} \;=\; \mathbb{P} \to \mathbb{P} \ .\end{aligned}$$

The eight components of the prefixed sum in the equation for $\mathbb{P}$ correspond to the eight forms of ambient actions τ, $\mathit{in}\ n$, $\mathit{out}\ n$, $\mathit{open}\ n!$, $\mathit{mvin}\ n!$, $\mathit{mvout}\ n!$, $\mathit{open}\ n?$ and $\mathit{mvin}\ n?$. We obtain the prefixing operations as injections into the appropriate component of the prefixed sum $\mathbb{P}$.

Parallel composition is really a family of operations, one of which is a binary operation between processes but where in addition there are parallel compositions of abstractions with concretions, and even abstractions with processes and concretions with processes. The family of operations

$$\begin{aligned}&(-|-) : \mathbb{F} \,\&\, \mathbb{C} \to \mathbb{P},\ (-|-) : \mathbb{F} \,\&\, \mathbb{P} \to \mathbb{F},\ (-|-) : \mathbb{C} \,\&\, \mathbb{P} \to \mathbb{C},\\ &(-|-) : \mathbb{C} \,\&\, \mathbb{F} \to \mathbb{P},\ (-|-) : \mathbb{P} \,\&\, \mathbb{F} \to \mathbb{F},\ (-|-) : \mathbb{P} \,\&\, \mathbb{C} \to \mathbb{C}\end{aligned}$$

are defined in a simultaneous recursive definition:

Processes in parallel with processes:

$$\begin{aligned}P|Q = {} & \Sigma_\alpha[P > \alpha.x \Rightarrow \alpha.(x|Q)] + \Sigma_\alpha[Q > \alpha.y \Rightarrow \alpha.(P|y)] + \\ & \Sigma_n[P > \mathit{open}\ n!.x \Rightarrow [Q > \mathit{open}\ n?.y \Rightarrow \tau.(x|y)]] + \\ & \Sigma_n[P > \mathit{open}\ n?.x \Rightarrow [Q > \mathit{open}\ n!.y \Rightarrow \tau.(x|y)]] + \\ & \Sigma_n[P > \mathit{mvin}\ n?.f \Rightarrow [Q > \mathit{mvin}\ n!.p \Rightarrow \tau.((f\ \ \mathit{fst}\, p)|\,\mathit{snd}\, p)]] + \\ & \Sigma_n[P > \mathit{mvin}\ n!.p \Rightarrow [Q > \mathit{mvin}\ n?.f \Rightarrow \tau.(\mathit{snd}\, p|(f\ \ \mathit{fst}\, p))]] \ .\end{aligned}$$

Abstractions in parallel with concretions: $F|C = (F\ \ \mathit{fst}\, C)|\,\mathit{snd}\, C$.
Abstractions in parallel with processes: $F|P = \lambda x.((F\ x)|P)$.
Concretions in parallel with processes: $C|P = (\mathit{fst}\, C, (\mathit{snd}\, C|P))$.

The remaining cases are given symmetrically. Processes P, Q of type $\mathbb{P}$ will—up to bisimilarity—be sums of prefixed terms, and by Proposition 6, their parallel composition satisfies the obvious expansion law.

Ambient creation can be defined recursively in the higher-order language:

$$\begin{aligned}m[P] = {} & [P > \tau.x \Rightarrow \tau.m[x]] + \\ & \Sigma_n[P > \mathit{in}\ n.x \Rightarrow \mathit{mvin}\ n!.(m[x], \varnothing)] + \\ & \Sigma_n[P > \mathit{out}\ n.x \Rightarrow \mathit{mvout}\ n!.(m[x], \varnothing)] + \\ & [P > \mathit{mvout}\ m!.p \Rightarrow \tau.(\mathit{fst}\, p|m[\mathit{snd}\, p])] + \\ & \mathit{open}\ m?.P + \mathit{mvin}\ m?.\lambda y.m[P|y] \ .\end{aligned}$$

The denotations of ambients are determined by their capabilities: an ambient $m[P]$ can perform the internal (τ) actions of P, enter a parallel ambient ($\mathit{mvin}\ n!$) if called upon to do so by an $\mathit{in}\ n$-action of P, exit an ambient n ($\mathit{mvout}\ n!$) if P so requests through an $\mathit{out}\ n$-action, be exited if P so requests through an $\mathit{mvout}\ m!$-action, be opened ($\mathit{open}\ m?$), or be entered by an ambient ($\mathit{mvin}\ m?$);

initial actions of other forms are restricted away. Ambient creation is at least as complicated as parallel composition. This should not be surprising given that ambient creation corresponds intuitively to putting a process behind (so in parallel with) a wall or membrane which if unopened mediates in the communications the process can do, converting some actions to others and restricting some others away. The tree-containment structure of ambients is captured in the chain of *open* m?'s that they can perform.

By the properties of prefix match (Proposition 6, items 2-4), there is an expansion theorem for ambient creation. For a process P with $P \sim \Sigma_\alpha \Sigma_{i\in I(\alpha)} \alpha.P_i$, where α ranges over atomic actions of ambients,

$$\begin{aligned} m[P] \sim\ & \Sigma_{i\in I(\tau)} \tau.m[P_i] + \\ & \Sigma_n \Sigma_{i\in I(in\ n)}\, mvin\ n!.(m[P_i], \varnothing) + \\ & \Sigma_n \Sigma_{i\in I(out\ n)}\, mvout\ n!.(m[P_i], \varnothing) + \\ & \Sigma_{i\in I(mvout\ m!)} \tau.(fst\, P_i | m[snd\, P_i]) + \\ & open\ m?.P + mvin\ m?.(\lambda y.m[P|y]) \ . \end{aligned}$$

7 Discussion

Matthew Hennessy's work on domain models of concurrency [10] is analogous to the domain theory we describe; our use of presheaf categories, functor categories of the form $[\mathbb{P}^{\mathrm{op}}, \mathbf{Set}]$ for a category $\mathbb{P}$, is mirrored in Hennessy's work by domains of the form $[\mathbb{P}^{\mathrm{op}}, \mathbf{2}]$ for a partial order $\mathbb{P}$ (here $\mathbf{2}$ is the partial order $0 < 1$). In this sense Hennessy's work anticipates the path-based domain theory used here.

Flemming Nielson's Typed Parallel Language (TPL) [17] is essentially a juxtaposition of value-passing CCS and simply-typed lambda calculus. Our language gains considerably in simplicity by integrating evaluation and transition semantics. The focus of TPL is a static type system in which processes are assigned a representation of their possible future communication possibilities, with no notion of causality. In contrast, our type system (which is also static) reflects that the capabilities of a process may change over time.

Andrew Gordon uses transition semantics and bisimulation to reason about the lambda calculus [9]. Our transitions correspond roughly to those of Gordon's "active" types, integrating evaluation and observation. "Passive" types include function space, for which Gordon uses transitions of the form $t \xrightarrow{@u} t\ u$ (where, in our notation, t has type $\mathbb{P} \to \mathbb{Q}$ and u has type $\mathbb{P}$). Such transitions have no counterpart in our semantics, because they would destroy the correspondence between derivations in the operational semantics and rooted components in the presheaf model.

Every presheaf category possesses a notion of bisimulation, derived from open maps [13, 14]. Open maps are a generalization of functional bisimulations, or zig-zag morphisms, known from transition systems. Presheaves over $\mathbb{P}$ are open-map bisimilar iff there is a span of surjective open maps between them. The maps of **Lin** and **Aff** preserve open maps and so open-map bisimulation [7, 4], giving congruence results for free when a process language is interpreted in these categories. Interestingly, although the operational bisimulation of Park/Milner is a congruence for the higher-order language, maps of **Cts** need not preserve open maps. This suggests that one should look at other notions of open map; in fact, as shown in [7], to every comonad T on **Lin** there is a corresponding notion of "T-open map", and thus a notion of "T-bisimulation". In the case of the exponential !, the corresponding !-bisimulation degenerates into isomorphism, but there are models where !-bisimulation and open-map bisimulation coincide, and one may hope to recover operational bisimulation as !-bisimulation for some choice of exponential (or expose it as a union of such bisimulations). In fact, it appears that the correspondence between the denotational and operational semantics can be proved abstractly, depending only on properties of the prefixed sum, so that it holds also for other choices of exponential.

The language has no distinguished invisible action τ, so there is the issue of how to support more abstract operational equivalences such as weak bisimulation, perhaps starting from [5].

Work is in progress on extending the idea of prefixed sum to include name-generation as in Milner's π-calculus. The π-calculus already has a presheaf semantics [6] but it has not been extended to higher order (see [22] for an operational extension). We hope to be able to link up with the work on the ν-calculus by Pitts and Stark [20] and Jeffrey and Rathke [12].

Acknowledgments. We thank the encyclopaedic Andy Pitts for helpful advice. Part of this paper was prepared during a pleasant visit of MN to DISI—the people there are thanked for their hospitality.

References

1. S. Abramsky. The lazy lambda calculus. In D. Turner (ed): *Research Topics in Functional Programming*. Addison-Wesley, 1990.
2. L. Cardelli and A. D. Gordon. Anytime, anywhere. Modal logics for mobile ambients. In *Proc. POPL'00.*
3. L. Cardelli and A. D. Gordon. A commitment relation for the ambient calculus. Note. October 6th, 2000.
4. G. L. Cattani. *Presheaf Models for Concurrency.* BRICS Dissertation Series DS-99-1, 1999.
5. G. L. Cattani, M. Fiore, and G. Winskel. Weak bisimulation and open maps. In *Proc. LICS'99.*
6. G. L. Cattani, I. Stark, and G. Winskel. Presheaf models for the π-calculus. In *Proc. CTCS'97*, LNCS 1290.
7. G. L. Cattani and G. Winskel. Profunctors, open maps and bisimulation. Manuscript, 2000.
8. A. D. Gordon and L. Cardelli. Equational properties of mobile ambients. In *Proc. FoSSaCS'99.*
9. A. D. Gordon. Bisimilarity as a theory of functional programming. In *Proc. MFPS'95*, ENTCS 1.
10. M. Hennessy. A fully abstract denotational model for higher-order processes. *Information and Computation*, 112(1):55–95, 1994.

11. D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.
12. A. Jeffrey and J. Rathke. Towards a theory of bisimulation for local names. In *Proc. LICS'99.*
13. A. Joyal and I. Moerdijk. A completeness theorem for open maps. *Annals of Pure and Applied Logic*, 70:51–86, 1994.
14. A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127:164–185, 1996.
15. G. M. Kelly. *Basic concepts of enriched category theory.* London Math. Soc. Lecture Note Series 64, CUP, 1982.
16. J. H. Morris. *Lambda-Calculus Models of Programming Languages.* PhD thesis, MIT, December 1968.
17. F. Nielson. The typed λ-calculus with first-class processes. In *Proc. PARLE'89*, LNCS 366.
18. M. Nygaard and G. Winskel. Linearity in process languages. In *Proc. LICS'02.*
19. D. Park. Concurrency and automata on infinite sequences. In *Proc. 5th GI Conference*, LNCS 104, 1981.
20. A. M. Pitts and I. D. B. Stark. Observable properties of higher order functions that dynamically create local names, or: What's *new*? In *Proc. MFCS'93*, LNCS 711.
21. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
22. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, University of Edinburgh, 1992.
23. B. Thomsen. A calculus of higher order communicating systems. In *Proc. POPL'89.*