Technical Report

UCAM-CL-TR-589
ISSN 1476-2986

Number 589

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

new-HOPLA — a higher-order process language with name generation

Glynn Winskel, Francesco Zappa Nardelli

May 2004

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

new-HOPLA
a higher-order process language with name generation

Glynn Winskel
Computer Laboratory
University of Cambridge, UK

Francesco Zappa Nardelli
INRIA & Computer Laboratory
University of Cambridge, UK

Abstract

This paper introduces new-HOPLA, a concise but powerful language for higher-order nondeterministic processes with name generation. Its origins as a metalanguage for domain theory are sketched but for the most part the paper concentrates on its operational semantics. The language is typed, the type of a process describing the shape of the computation paths it can perform. Its transition semantics, bisimulation, congruence properties and expressive power are explored. Encodings of π-calculus and HOπ are presented.

1 The origins of new-HOPLA

This work is part of a general programme (reported in [8]), to develop a domain theory which scales up to the intricate languages, models and reasoning techniques used in distributed computation. This ambition led to a concentration on path based models, and initially on presheaf models because they can even encompass causal dependency models like event structures; so 'domains' is being understood more broadly than usual, to include presheaf categories. The general methodology has been to develop a domain theory with a rich enough life of its own to suggest a powerful metalanguage. A feature of presheaf models has been very useful here: in key cases there is often a strong correspondence between elements of the presheaf denotation and derivations in an operational semantics.

Of course, these days any process language worth its salt must address name generation. This paper reports on new-HOPLA, a compact but expressive language for higher-order nondeterministic processes with name generation. It extends the language HOPLA of Nygaard and Winskel [7], and like its predecessor has its origins in a domain theory for concurrency. Specifically it arose out of the metalanguage implicitly being used in giving a presheaf semantics to the π-calculus [2]. But a sketch of its mathematical origins and denotational semantics does not require that heavy an investment, and can be based on paths sets rather than presheaves.[1]

If for the moment we ignore name generation, a suitable category of domains is that of **Lin**. Its objects, *path orders*, are preorders $\mathbb{P}$ consisting of computation paths with the order $p \leq p'$ expressing how a path p extends to a path p'. A path order $\mathbb{P}$ determines a domain $\widehat{\mathbb{P}}$, that of its *path sets*, left-closed sets w.r.t. $\leq_{\mathbb{P}}$, ordered by inclusion. (Such a domain is a prime-algebraic complete lattice, in which the complete primes are precisely those path sets generated by individual paths.) The arrows of **Lin**, linear maps, from $\mathbb{P}$ to $\mathbb{Q}$ are join-preserving functions from $\widehat{\mathbb{P}}$ to $\widehat{\mathbb{Q}}$. The category **Lin** is monoidal-closed with a tensor given the product $\mathbb{P} \times \mathbb{Q}$ of path orders and a corresponding function space by $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$—it is easy to see that join-preserving

[1] Path sets arise by 'flattening' presheaves, which can be viewed as characteristic functions to truth values given in the category of sets, as sets of realisers, to simpler characteristic functions based on truth values $0 \leq 1$ [8].

functions from $\widehat{\mathbb{P}}$ to $\widehat{\mathbb{Q}}$ correspond to path sets of $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$. In fact **Lin** has enough structure to form a model of Girard's classical linear logic [4]. To exhibit its exponential ! we first define the category **Cts** to consist, like **Lin**, of path orders as objects but now with arrows the Scott-continuous functions between the domains of path sets. The inclusion functor **Lin** $\hookrightarrow$ **Cts** has a left adjoint ! : **Cts** $\rightarrow$ **Lin** which takes a path order $\mathbb{P}$ to a path order consisting of finite subsets of $\mathbb{P}$ with order

$$P \leq_{!\mathbb{P}} P' \text{ iff } \forall p \in P\ \exists p' \in P'.\ p \leq_{\mathbb{P}} p'$$

—so $!\mathbb{P}$ can be thought of as consisting of compound paths associated with several runs.

The higher-order process language HOPLA is built around constructions in the category **Lin**. Types of HOPLA, which may be recursively defined, denote objects of **Lin**, path orders circumscribing the computation paths possible. As such all types support operations of non-deterministic sum and recursive definitions, both given by unions. Sum types are provided by coproducts, and products, of **Lin**, both given by the disjoint juxtaposition of path orders; they provide injection and projection operations. There is a type of functions from $\mathbb{P}$ to $\mathbb{Q}$ given by $(!\mathbb{P})^{\mathrm{op}} \times \mathbb{Q}$, the function space of **Cts**; this gives the operation of application and lambda abstraction. To this the adjunction yields a primitive prefix operation, a map $\mathbb{P} \rightarrow !\mathbb{P}$, given by the unit at $\mathbb{P}$; it is accompanied by a destructor, a prefix-match operation, obtained from the adjunction's natural isomorphism. For further details, encodings of traditional process calculi in HOPLA and a full abstraction result, the reader is refrerred to [7, 9].

We are interested in extending HOPLA to allow name generation. We get our inspiration from the domain theory. As usual a domain theory for name generation is obtained by moving to a category in which standard domains are indexed functorially by the current set of names. The category $\mathcal{I}$ consists of finite sets of names related by injective functions. The functor category $\mathbf{Lin}^{\mathcal{I}}$ has as objects functors $\mathbb{P} : \mathcal{I} \rightarrow \mathbf{Lin}$, so path orders $\mathbb{P}(s)$ indexed by finite sets of names s standing for the computation paths possible with that current set of names; its arrows are natural transformations $\alpha = \langle \alpha_s \rangle_{s \in \mathcal{I}} : \mathbb{P} \rightarrow \mathbb{Q}$, with components in **Lin**. One important object in $\mathbf{Lin}^{\mathcal{I}}$ is the object of names $\mathbb{N}$ providing the current set of names, so $\mathbb{N}(s) = s$ regarded as a discrete order, at name set s. Types of new-HOPLA will denote objects of $\mathbf{Lin}^{\mathcal{I}}$.

The category has coproducts and products, both given by disjoint juxtaposition at each component. These provide a *sum type* $\Sigma_{i \in I} \mathbb{P}_i$ from a family of types $(\mathbb{P}_i)_{i \in I}$. It has *injections* producing a term $i{:}t$ of type $\Sigma_{i \in I} \mathbb{P}_i$ from a term t of type $\mathbb{P}_i$, for $i \in I$. *Projections* produce a term $\pi_i t$ of type $\mathbb{P}_i$ from a term t of the sum type.

There is a tensor got pointwise from the tensor of **Lin**. Given $\mathbb{P}$ and $\mathbb{Q}$ in $\mathbf{Lin}^{\mathcal{I}}$ we define $\mathbb{P} \otimes \mathbb{Q}$ in $\mathbf{Lin}^{\mathcal{I}}$ so that at $s \in \mathcal{I}$

$$\mathbb{P} \otimes \mathbb{Q}(s) \ = \ \mathbb{P}(s) \times \mathbb{Q}(s)\ .$$

In fact we will only use a special case of this construction to form tensor types $\mathbb{N} \otimes \mathbb{P}$, so $(\mathbb{N} \otimes \mathbb{P})(s) = s \times \mathbb{P}(s)$ at $s \in \mathcal{I}$. These are a form of 'dynamic sum' in which the components and the corresponding injections grow with the availability of new names. There are term constructors producing a term $n \cdot t$ of type $\mathbb{N} \otimes \mathbb{P}$ from a term t of type $\mathbb{P}$ and a name n. There are projections $\pi_n t$ forming a term of type $\mathbb{P}$ from a term t of tensor type.

At any stage s, the current set of names, a new name can be generated and used in a term in place of a variable over names. This leads to the central idea of new-name abstractions of type $\delta\mathbb{P}$ where $\delta\mathbb{P}(s) = \mathbb{P}(s \dot{\cup} \{\star\})$ at name set s. As observed by Stark [13] the construction $\delta\mathbb{P}$ can be viewed as a space of functions from $\mathbb{N}$ to $\mathbb{P}$ but with the proviso that the input name is fresh. A new-name abstraction is written $new\alpha.t$ and has type $\delta\mathbb{P}$, where t is a term of type $\mathbb{P}$. New-name application is written $t[n]$, where t has type $\delta\mathbb{P}$, and requires that the name n is fresh w.r.t. the names of t.

The adjunction $\mathbf{Lin} \underset{\hookrightarrow}{\overset{!}{\underset{\perp}{\leftarrow}}} \mathbf{Cts}$ induces an adjunction

$$\mathbf{Lin}^{\mathcal{I}} \underset{\hookrightarrow}{\overset{!}{\underset{\perp}{\leftarrow}}} \mathbf{Cts}^{\mathcal{I}}$$

where the left adjoint is got by extending the original functor $! : \mathbf{Cts} \to \mathbf{Lin}$ in a pointwise fashion. The unit of the adjunction provides a family of maps from $\mathbb{P}$ to $!\mathbb{P}$ in $\mathbf{Cts}^{\mathcal{I}}$. As with HOPLA, these yield a prefix operation $!t$ of type $!\mathbb{P}$ for a term t of type $\mathbb{P}$. A type of the form $!\mathbb{P}$ is called a prefix type; its computation paths at any current name set first involve performing a prototypical action, also called '!'.

To support higher-order processes we need function spaces $\mathbb{P} \multimap \mathbb{Q}$ such that

$$\mathbf{Lin}^{\mathcal{I}}(\mathbb{R}, \mathbb{P} \multimap \mathbb{Q}) \;\cong\; \mathbf{Lin}^{\mathcal{I}}(\mathbb{R} \otimes \mathbb{P}, \mathbb{Q})$$

natural in $\mathbb{R}$ and $\mathbb{Q}$. Such function spaces do not exist in general—the difficulty is in getting a path order $\mathbb{P} \multimap \mathbb{Q}(s)$ at each name set s. However a function space $\mathbb{P} \multimap \mathbb{Q}$ does exist in the case where $\mathbb{P}f$ preserves complete primes and $\mathbb{Q}f$ preserves non-empty meets for each map $f : s \to s'$ in $\mathcal{I}$. This suggests limiting the syntax of types to special function spaces $\mathbb{N} \multimap \mathbb{Q}$ and $!\mathbb{P} \multimap \mathbb{Q}$, the function space in $\mathbf{Cts}^{\mathcal{I}}$. The function spaces are associated with operations of application and lambda abstraction.

These constructions provide the basis of new-HOPLA. It resembles, and indeed has been inspired by, the metalanguages for domain theories with name generation used implicitly in earlier work [3, 13, 2], as well as the language of FreshML [11]. The language new-HOPLA is distinguished through the path-based domain theories to which it is fitted and, as we will see, in itself forming a process language with an operational semantics. For space reasons, in this extended abstract we elide the proofs of the theorems; these can be found in [15].

2 The language

Types The type of names is denoted by $\mathbb{N}$. The types of processes are defined by the grammar below.

$$\begin{array}{rcl} \mathbb{P} & ::= & \mathbf{0} \mid \mathbb{N} \otimes \mathbb{P} \mid\, !\mathbb{P} \mid \delta\mathbb{P} \mid \mathbb{N} \to \mathbb{P} \mid \mathbb{P} \to \mathbb{Q} \mid \\ & & \Sigma_{i \in I} \mathbb{P}_i \mid \mu_j P_1 \ldots P_k.(\mathbb{P}_1 \ldots \mathbb{P}_k) \mid P \end{array}$$

The sum type $\Sigma_{i \in I} \mathbb{P}_i$ where I is a finite set, is most often written $i_1{:}\mathbb{P} + \cdots + i_k{:}\mathbb{P}$. The symbol P is drawn from a set of type variables used in defining recursive types; closed type expressions are interpreted as path orders. The type $\mu_j P_1 \ldots P_k.(\mathbb{P}_1 \ldots \mathbb{P}_k)$ is interpreted as the j-component, for $1 \leq j \leq k$, of the 'least' solution to the defining equations $P_1 = \mathbb{P}_1, \ldots, P_k = \mathbb{P}_k$, where the expressions $\mathbb{P}_1 \ldots \mathbb{P}_k$ may contain the P_j's.

Terms and actions We assume a countably infinite set of *name constants*, ranged over by $a, b, \ldots$ and a countably infinite set of *name variables*, ranged over by $\alpha, \beta, \ldots$ Names, either constants or variables, are ranged over by $m, n, \ldots$. We assume an infinite, countable, set of *process variables*, ranged over by $x, y, \ldots$

Every type is associated with actions processes of that type may do. The *actions* are defined by the grammar below:

$$p, q, r \quad ::= \quad x \mid\, !p \mid n \cdot p \mid i{:}p \mid \mathit{new}\alpha.p \mid n \mapsto p \mid u \mapsto p \mid p[n] \;.$$

As we will see shortly, well-typed actions are constructed so that they involve exactly one prototypical action ! and exactly one 'resumption variable' x. Whenever a term performs the action, the variable of the action matches the resumption of the term: the typings of an action thus relates the type of a term with the type of its resumption. According to the transition rules a process of prefix type $!\mathbb{P}$ may do actions of the form $!p$, while a process of tensor or sum type may do actions of the form $n \cdot p$ or $i{:}p$ respectively. A process of type $\delta\mathbb{P}$ does actions of the form $new\alpha.p$ meaning that at the generation of a new name, a say, as input the action $p[a/\alpha]$ is performed. Actions of function type $n \mapsto p$ or $u \mapsto p$ express the dependency of the action on the input of a name n or process u respectively. The final clause is necessary in building up actions because we sometimes need to apply a resumption variable to a new name.

The *terms* are defined by the grammar below:

t, u, v	$::=$	$\mathbf{0}$	$!t$		inactive process and prototypical action
		$n \cdot t$	$\pi_n t$		tensor and projection
		$\lambda x.t$	tu		process abstraction and application
		$\lambda\alpha.t$	tn		name abstraction and application
		$new\alpha.t$	$t[n]$		new name abstraction and application
		$recx.t$	x		recursive definition and process variables
		$i{:}t$	$\pi_i t$		injection and projection
		$\Sigma_{i\in I} t_i$	$\Sigma_{\alpha\in\mathbb{N}} t$		sum and sum over names
		$[t > p(x) \Rightarrow u]$			pattern matching

In new-HOPLA actions are used as patterns in terms $[t > p(x) \Rightarrow u]$ where we explicitly note the resumption variable x. If the term t can perform the action p the resumption of t is passed on to u via the variable x.

We assume an understanding of the *free name variables* (the binders of name variables are $\lambda\alpha.-$, $new\alpha.-$, and $\Sigma_{\alpha\in\mathbb{N}}-$) and of the *free process variables* (the binders of process variables are $\lambda x.-$, and $[t > p(x) \Rightarrow -]$) of a term. The *support* of a term, denoted $\mathsf{n}(t)$, is the set of the its free names, that is, the set of its name constants and of its free name variables.

We say that a name n is *fresh* for a term t if $n \notin \mathsf{n}(t)$.

2.1 Transition rules

The behaviour of terms is defined by a transition relation of the form

$$s \vdash t \xrightarrow{p\ (x)} t'$$

where s is a finite set of name constants such that $\mathsf{n}(t) \subseteq s$. The transition above should be read as 'with current names s the term t can perform the action p and resume as t''. We generally note the action's resumption variable in the transitions; this simplifies the transition rules in which the resumption variable must be explicitly manipulated.

So the transition relation is given at stages indexed by the set of current names s. The body of an abstraction over names $\lambda\alpha.t$ can only be instantiated with a name in s, and an abstraction over processes $\lambda x.t$ can only be instantiated with a process whose support is contained in s. As the transition relation is indexed by the current set of names, it is possible to generate new names at run-time. Indeed, the transition rule for new-name abstraction $new\alpha.t$ extends the set s of current names with a new name $a \notin s$; this name a is then passed to t via the variable α. The transition rules must respect the typings of actions and terms given in the next section. Formally:

$$\frac{}{!\mathbb{P}; s \vdash\, !t \xrightarrow{!x\ (x)} t} \qquad \frac{\mathbb{P}; s \vdash t \xrightarrow{p} t' \quad a \in s}{\mathbb{N} \otimes \mathbb{P}; s \vdash a \cdot t \xrightarrow{a \cdot p} t'} \qquad \frac{\mathbb{N} \otimes \mathbb{P}; s \vdash t \xrightarrow{a \cdot p} t'}{\mathbb{P}; s \vdash \pi_a t \xrightarrow{p} t'}$$

$$\frac{\mathbb{P}; s \vdash t_i \xrightarrow{p\ (x)} t'}{\mathbb{P}; s \vdash \Sigma_{i \in I} t_i \xrightarrow{p\ (x)} t'} \qquad \frac{\mathbb{P}; s \vdash t[a/\alpha] \xrightarrow{p\ (x)} u \quad a \in s}{\mathbb{P}; s \vdash \Sigma_{\alpha \in \mathbb{N}} t \xrightarrow{p\ (x)} u} \qquad \frac{\mathbb{P}; s \vdash t[recy.t/y] \xrightarrow{p\ (x)} u}{\mathbb{P}; s \vdash recy.t \xrightarrow{p\ (x)} u}$$

$$\frac{\mathbb{P}_i; s \vdash t \xrightarrow{p\ (x)} t'}{\Sigma_{i \in I} \mathbb{P}_i; s \vdash i{:}t \xrightarrow{i{:}p\ (x)} t'} \qquad \frac{\Sigma_{i \in I} \mathbb{P}_i; s \vdash t \xrightarrow{i{:}p\ (x)} t'}{\mathbb{P}_i; s \vdash \pi_i t \xrightarrow{p\ (x)} t'} \qquad \frac{\mathbb{Q}; s \vdash t[u/x] \xrightarrow{p\ (x)} v \quad s \vdash u : \mathbb{P}}{\mathbb{P} \to \mathbb{Q}; s \vdash \lambda x.t \xrightarrow{u \mapsto p\ (x)} v}$$

$$\frac{\mathbb{P} \to \mathbb{Q}; s \vdash t \xrightarrow{u \mapsto p\ (x)} v}{\mathbb{Q}; s \vdash tu \xrightarrow{p\ (x)} v} \qquad \frac{\mathbb{P}; s \vdash t[a/\alpha] \xrightarrow{p\ (x)} v \quad a \in s}{\mathbb{N} \to \mathbb{P}; s \vdash \lambda \alpha.t \xrightarrow{a \mapsto p\ (x)} v} \qquad \frac{\mathbb{N} \to \mathbb{P}; s \vdash t \xrightarrow{a \mapsto p\ (x)} v}{\mathbb{P}; s \vdash ta \xrightarrow{p\ (x)} v}$$

$$\frac{\mathbb{P}; s \dot\cup \{a\} \vdash t[a/\alpha] \xrightarrow{p[a/\alpha]\ (x)} u[a/\alpha]}{\delta\mathbb{P}; s \vdash new\alpha.t \xrightarrow{new\alpha.p[x'[\alpha]/x]\ (x')} new\alpha.u} \qquad \frac{\delta\mathbb{P}; s \vdash t \xrightarrow{new\alpha.p[x'[\alpha]/x]\ (x')} u}{\mathbb{P}; s \dot\cup \{a\} \vdash t[a] \xrightarrow{p[a/\alpha]\ (x)} u[a]}$$

$$\frac{\mathbb{P}; s \vdash t \xrightarrow{p\ (x)} t' \quad \mathbb{Q}; s \vdash u[t'/x] \xrightarrow{q\ (x')} v}{\mathbb{Q}; s \vdash [t > p(x) \Rightarrow u] \xrightarrow{q\ (x')} v}$$

In the rule for new name abstraction, the conditions $a \notin \mathsf{n}(p)$ and $a \notin \mathsf{n}(u)$ must hold.

Table 1: new-HOPLA: transition rules

Definition 2.1 (Transition relation) *For closed terms t such that $s \vdash t : \mathbb{P}$ and path patterns such that $s;;x{:}\mathbb{Q} \Vdash p : \mathbb{P}$ the rules reported in Table 1 define a relation $\mathbb{P}; s \vdash t \xrightarrow{p\ (x)} u$, called the* transition relation.

To help familiarise the reader with the transition semantics, we present some derivations.
— *The operational semantics validates β-equivalence:*

$$\frac{\mathbb{Q}; s \vdash t[u/x] \xrightarrow{p\,(y)} t'}{\dfrac{\mathbb{P} \to \mathbb{Q}; s \vdash \lambda x.t \xrightarrow{u \mapsto p\,(y)} t'}{\mathbb{Q}; s \vdash (\lambda x.t)u \xrightarrow{p\,(y)} t'}} \qquad \frac{\mathbb{Q}; s \vdash t[a/\alpha] \xrightarrow{p\,(y)} t'}{\dfrac{\mathbb{N} \to \mathbb{Q}; s \vdash \lambda \alpha.t \xrightarrow{a \mapsto p\,(y)} t'}{\mathbb{Q}; s \vdash (\lambda \alpha.t)a \xrightarrow{p\,(y)} t'}}$$

These derivations illustrate how β-equivalence is validated by the transition relation in the sense that a term $(\lambda x.t)u$ (resp. $(\lambda \alpha.t)a$) has the same transition capabilities as the term $t[u/x]$ (resp. $t[a/\alpha]$): for example, a term $(\lambda x.t)u$ performs an action p and resumes as t' iff the term $t[u/x]$ performs the same action p resuming as t'—the 'only if' part follows by the uniqueness of the derivations.

— *The action of the tensor and sum type:*

$$\frac{\mathbb{P}; s \vdash t \xrightarrow{p\,(y)} t'}{\mathbb{N} \otimes \mathbb{P}; s \vdash a \cdot t \xrightarrow{a \cdot p\,(y)} t'} \qquad \frac{\mathbb{P}_i; s \vdash t \xrightarrow{p\,(y)} t'}{\Sigma_{i \in I} \mathbb{P}_i; s \vdash i{:}t \xrightarrow{i : p\,(y)} t'}$$

$$\frac{\mathbb{N} \otimes \mathbb{P}; s \vdash a \cdot t \xrightarrow{a \cdot p\,(y)} t'}{\mathbb{P}; s \vdash \pi_a(a \cdot t) \xrightarrow{p\,(y)} t'} \qquad \frac{\Sigma_{i \in I} \mathbb{P}_i; s \vdash i{:}t \xrightarrow{i : p\,(y)} t'}{\mathbb{P}_i; s \vdash \pi_i(i{:}t) \xrightarrow{p\,(y)} t'}$$

By uniqueness of the derivations, a term t has the same transition capabilites as the terms $\pi_a(a \cdot t)$ and $\pi_i(i{:}t)$.

— *New name abstraction and β-equivalence:*

$$\frac{\mathbb{Q}; s \dot{\cup} \{a\} \vdash t[a/\alpha] \xrightarrow{p[a/\alpha]\,(y)} t'[a/\alpha]}{\dfrac{\delta\mathbb{Q}; s \vdash new\alpha.t \xrightarrow{new\alpha.\alpha \mapsto p[y'[\alpha]/y]\,(y')} new\alpha.t'}{\mathbb{Q}; s \dot{\cup} \{a\} \vdash (new\alpha.t)[a] \xrightarrow{p[a/\alpha]\,(y)} (new\alpha.t')[a]}}$$

Again by uniqueness of the derivation, $(new\alpha.t)[a]$ and $t[a/\alpha]$ have the same transition capabilities for a fresh name a.

— *Matching the prefix action:*

$$\frac{!\mathbb{P}; s \vdash !t \xrightarrow{!x\,(x)} t \qquad \mathbb{Q}; s \vdash u[t/x] \xrightarrow{p\,(y)} u'}{\mathbb{Q}; s \vdash [!t > !x(x) \Rightarrow u] \xrightarrow{p\,(y)} u'}$$

The derivation above illustrates the simplest case of pattern matching. The term being tested $!t$ emits the prototypical action ! and continues as t. Then the computation path $!x$ is matched against the path pattern $!x$; matching is successful and the continuation t is bound to x in u, which then executes. Pattern matching allows for the testing of arbitrary actions, even if a little care is needed when new names are involved.

— *Matching and new name abstraction:*

$$\frac{\dfrac{!\mathbb{P}; s \dot{\cup} \{a\} \vdash !t[a/\alpha] \xrightarrow{!x'\,(x')} t[a/\alpha]}{\delta!\mathbb{P}; s \vdash new\alpha.!t \xrightarrow{new\alpha.!x[\alpha]\,(x)} new\alpha.t} \qquad \mathbb{Q}; s \vdash u[new\alpha.t/x] \xrightarrow{p\,(y)} u'}{\mathbb{Q}; s \vdash [new\alpha.!t > new\alpha.!x[\alpha](x) \Rightarrow u] \xrightarrow{p\,(y)} u'}$$

The resumption of a term of type $\delta\mathbb{P}$ after a transition is always a new name abstraction, and the new name generated in testing a pattern (above it is a) is local to the test.

2.2 Typing judgements

Consider a term $t = t'[\alpha]$. As we have discussed in the previous section, the square brackets denote new-name application: any name instantiating α should be fresh for the term t'. Consider now the context $C[-] = \lambda\alpha.-$. In the term $C[t] = \lambda\alpha.(t'[\alpha])$, the variable α is abstracted via a lambda abstraction, and may be instantiated with any current name. In particular it may be instantiated with names that belong to the support of t', thus breaking the hypothesis that t' has been applied to a fresh name. The same problem arises with contexts of the form $C[-] = \Sigma_{\alpha \in \mathbb{N}} -$. Moreover, if the process variable x is free in t, a context like $C[-] = \lambda x.-$

might instantiate x with an arbitrary term u. As the name instantiating α might belong to the support of u, nothing ensures it is still fresh for the term $t[u/x]$.

The type system must sometimes ensure that name variables are instantiated by fresh names. To impose this restriction, the typing context contains not only typing assumptions about name and process variables, such as $\alpha{:}\mathbb{N}$ and $x{:}\mathbb{P}$, but also *freshness assumptions* (or *distinctions*) about them, written (α, β) or (α, x). Here the intended meaning of (α, β) is that, in any environment, the names instantiating the variables α and β must be *distinct*. A freshness assumption like (α, x), where x is a process variable, records that in any environment the name instantiating α must be fresh for the term instantiating x.

Using this auxiliary information, the type system assumes that it is safe to abstract a variable, using lambda abstraction or sum over names, only if no freshness assumptions have been made on it.

The type system of new-HOPLA terms can be specified using judgements of the form:

$$A; \Gamma; d \vdash t : \mathbb{P}$$

where

- $A \equiv \alpha_1{:}\mathbb{N}, \ldots, \alpha_k{:}\mathbb{N}$ is a collection of name variables;
- $\Gamma \equiv x_1{:}\mathbb{P}_1, \ldots, x_k{:}\mathbb{P}_k$ is a partial function from of process variables, together with their types;
- d is a set of pairs $(\alpha, x) \in A \times \Gamma$, and $(\alpha, \beta) \in A \times A$, keeping track of the *freshness assumptions*.

Notation: We write $d \setminus \alpha$ for the set of freshness assumptions obtained from d by deleting all pairs containing α. The order in which variables appear in a distinction is irrelevant; we will write $(\alpha, \beta) \in d$ as a shorthand for $(\alpha, \beta) \in d$ *or* $(\beta, \alpha) \in d$. When we write $\Gamma \cup \Gamma'$ we allow the environments to overlap; the variables need not be disjoint provided the environments are consistent.

Actions are typed along the same lines, even if type judgements explicitly report the resumption variable:

$$A; \Gamma; d; ; x{:}\mathbb{R} \Vdash p : \mathbb{P} \ .$$

The meaning of the environment $A; \Gamma; d$ is exactly the same as above. The variable x is the resumption variable of the pattern p, and its type is $\mathbb{R}$.

The type system of new-HOPLA is reported in Table 2 and Table 3.

The rule responsible for generating freshness assumptions is the rule for new-name application. If the term t has been typed in the environment $A; \Gamma; d$ and α is a new name variable (that is, $\alpha \notin A$), then the term $t[\alpha]$ is well-typed under the hypothesis that any name instantiating the variable α is distinct from all the names in terms instantiating the variables that can appear in t. This is achieved adding the set of freshness assumptions $\{\alpha\} \times (\Gamma \cup A)$ to d (when convenient, as here, we will confuse an environment with its domain).

The rule for pattern matching also modifies the freshness assumptions. The operational rule of pattern matching substitutes a subterm of t, whose names are contained in A', for x. Accordingly, the typing rule initially checks that no name in A' belongs to the set of the variables supposed fresh for x. Our attention is then drawn to the term $u[t'/x]$, where t' is a subterm of t. A name variable $\alpha \in A$ supposed fresh from x when typing u, must now be supposed fresh from all the free variables of t'. This justifies the freshness assumptions $\{\{\alpha\} \times (A' \cup \Gamma') \mid (\alpha, x) \in d\}$.

The rest of the type system follows along the lines of type systems for the simply typed λ-calculus.

$$\frac{}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\!\vdash\ !x : !\mathbb{R}} \qquad \frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\!\vdash\ p : \mathbb{P}}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\!\vdash\ \alpha \cdot p : \mathbb{N} \otimes \mathbb{P}}\alpha \in A$$

$$\frac{\alpha{:}\mathbb{N}, A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\!\vdash\ p : \mathbb{P}}{A;\Gamma;(d \setminus \alpha);;x'{:}\delta\mathbb{R} \Vdash\!\!\vdash\ new\alpha.p[x'[\alpha]/x] : \delta\mathbb{P}} \qquad \frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\!\vdash\ p : \mathbb{P}}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\!\vdash\ \alpha \mapsto p : \mathbb{P}}\alpha \in A$$

$$\frac{A;\Gamma;d \vdash u : \mathbb{Q} \qquad A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\!\vdash\ p : \mathbb{P}}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\!\vdash\ u \mapsto p : \mathbb{Q} \to \mathbb{P}} \qquad \frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\!\vdash\ p : \mathbb{P}_j \quad j \in I}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\!\vdash\ (j{:}p) : \Sigma_{i\in I}\mathbb{P}_i}$$

$$\frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\!\vdash\ t : \mathbb{P}_j[\mu\vec{P}.\vec{\mathbb{P}}/\vec{P}]}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\!\vdash\ t : \mu_j P : \vec{\mathbb{P}}} \qquad \frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\!\vdash\ p : \mathbb{P}}{A';\Gamma';d';;x{:}\mathbb{R} \Vdash\!\!\vdash\ p : \mathbb{P}} \begin{array}{l} A \subseteq A' \\ \Gamma \subseteq \Gamma' \\ d \subseteq d' \end{array}$$

Table 2: new-HOPLA: typing rules for actions

The type system assumes that terms do not contain name constants. This is to avoid the complications in a type system coping with both name variables and constants at the same time. We write $s \vdash t : \mathbb{P}$ when there is a judgement $A;\emptyset;d \vdash \sigma t' : \mathbb{P}$ and a substitution σ for A respecting the distinctions d such that t is $\sigma t'$. In particular, $s \vdash t : \mathbb{P}$ iff there is a canonical judgement $A;\emptyset;\{(\alpha,\beta) \mid \alpha \neq \beta\} \vdash t' : \mathbb{P}$, in which the substitution σ is a bijection between name variables and names and t is $\sigma t'$. Similarly for patterns.

We can now prove that the operational rules are type correct.

Lemma 2.2 (Substitution Lemma) *Suppose that $A';\Gamma';d' \vdash t : \mathbb{Q}$ and $A;x{:}\mathbb{Q},\Gamma;d \vdash u : \mathbb{P}$, where $\Gamma \cup \Gamma'$ is consistent and $A' \cap \{\alpha \mid (\alpha,x) \in d\} = \emptyset$. Then,*

$$A \cup A';\Gamma \cup \Gamma';\overline{d} \vdash u[t/x] : \mathbb{P}$$

where $\overline{d} = (d \setminus x) \cup d' \cup \{\{\alpha\} \times (A' \cup \Gamma') \mid (\alpha,x) \in d\}$.

Theorem 2.3 (Transitions preserve types) *If $s \vdash t : \mathbb{P}$ and $s;;x{:}\mathbb{R} \Vdash\!\!\vdash\ p : \mathbb{P}$ and $\mathbb{P};s \vdash t \xrightarrow{p\ (x)} t'$, then $s \vdash t' : \mathbb{R}$.*

3 Equivalences

After introducing some notations regarding relations, we explore the bisimulation equivalence that arises from the transition semantics.

A relation $\mathcal{R}$ between typing judgements is said to respect types if, whenever $\mathcal{R}$ relates $E_1 \vdash t_1 : \mathbb{P}_1$ and $E_2 \vdash t_2 : \mathbb{P}_2$, we have $E_1 = E_2$ and $\mathbb{P}_1 = \mathbb{P}_2$. We are mostly interested in relations between closed terms, and we write $s \vdash t\ \mathcal{R}\ u : \mathbb{P}$ to denote $(s \vdash t : \mathbb{P}, s \vdash q : \mathbb{P}) \in \mathcal{R}$.

Definition 3.1 (Bisimilarity) *A type-respecting relation on closed terms, $\mathcal{R}$, is a* bisimulation *if*

1. *$s \vdash t\ \mathcal{R}\ u : \mathbb{P}$ and $\mathbb{P};s' \vdash t \xrightarrow{p(x)} t'$ for $s' \supseteq s$ imply that there exists a term u' such that $\mathbb{P};s' \vdash u \xrightarrow{p(x)} u'$ and $s' \vdash t'\ \mathcal{R}\ u' : \mathbb{R}$;*
2. *$s \vdash t\ \mathcal{R}\ u : \mathbb{P}$ and $\mathbb{P};s' \vdash u \xrightarrow{p(x)} u'$ for $s' \supseteq s$ imply that there exists a term t' such that $\mathbb{P};s' \vdash t \xrightarrow{p(x)} t'$ and $s' \vdash t'\ \mathcal{R}\ u' : \mathbb{R}$;*

$$\frac{}{A;\Gamma;d\vdash\emptyset:\mathbb{P}} \qquad \frac{}{A;x{:}\mathbb{P},\Gamma;d\vdash x:\mathbb{P}} \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}}{A';\Gamma';d'\vdash t:\mathbb{P}}\ \begin{array}{l}A\subseteq A'\\ \Gamma\subseteq\Gamma'\\ d\subseteq d'\end{array}$$

$$\frac{\alpha{:}\mathbb{N},A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash\Sigma_{\alpha\in\mathbb{N}}t:\mathbb{P}}\alpha\notin d \qquad \frac{\alpha{:}\mathbb{N},A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash\lambda\alpha.t:\mathbb{N}\to\mathbb{P}}\alpha\notin d$$

$$\frac{A;x{:}\mathbb{Q},\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash\lambda x.t:\mathbb{Q}\to\mathbb{P}}x\notin d \qquad \frac{A;x{:}\mathbb{P},\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash recx.t:\mathbb{P}}x\notin d$$

$$\frac{\alpha{:}\mathbb{N},A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;(d\setminus\alpha)\vdash new\alpha.t:\delta\mathbb{P}} \qquad \frac{A;\Gamma;d\vdash t:\delta\mathbb{P}}{\alpha{:}\mathbb{N},A;\Gamma;d\cup(\{\alpha\}\times(\Gamma\cup A))\vdash t[\alpha]:\mathbb{P}}$$

$$\frac{A;\Gamma;d\vdash t:\mathbb{N}\to\mathbb{P}}{A;\Gamma;d\vdash t\alpha:\mathbb{P}}\alpha\in A \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}\to\mathbb{Q}\quad A;\Gamma;d\vdash u:\mathbb{P}}{A;\Gamma;d\vdash tu:\mathbb{Q}}$$

$$\frac{A;\Gamma;d\vdash t_i:\mathbb{P}\quad\forall i\in I}{A;\Gamma;d\vdash\Sigma_{i\in I}t_i:\mathbb{P}} \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}_i}{A;\Gamma;d\vdash i{:}t:\Sigma_{i\in I}\mathbb{P}_i} \qquad \frac{A;\Gamma;d\vdash t:\Sigma_{i\in I}\mathbb{P}_i}{A;\Gamma;d\vdash\pi_i t:\mathbb{P}_i} \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash !t:!\mathbb{P}}$$

$$\frac{A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash\alpha\cdot t:\mathbb{N}\otimes\mathbb{P}}\alpha\in A \qquad \frac{A;\Gamma;d\vdash t:\mathbb{N}\otimes\mathbb{P}}{A;\Gamma;d\vdash\pi_\alpha t:\mathbb{P}}\alpha\in A \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}_j[\mu\vec{P}.\vec{\mathbb{P}}/\vec{P}]}{A;\Gamma;d\vdash t:\mu_j P:\vec{\mathbb{P}}}$$

$$\frac{A';\Gamma';d'\vdash t:\mathbb{P} \qquad A';\Gamma';d';;x{:}\mathbb{R}\Vdash p:\mathbb{P} \qquad A;x{:}\mathbb{R},\Gamma;d\vdash u:\mathbb{Q}}{A\cup A';\Gamma\cup\Gamma';\overline{d}\vdash[t>p(x)\Rightarrow u]:\mathbb{Q}}A'\cap\{\alpha\mid(\alpha,x)\in d\}=\emptyset$$

$$\text{where } \overline{d}=(d\setminus x)\cup d'\cup\{\{\alpha\}\times(A'\cup\Gamma')\mid(\alpha,x)\in d\}$$

Table 3: new-HOPLA: typing rules for processes

where $\mathbb{R}$ *is the type of the resumption variable* x *in* p*. Let* bisimilarity*, denoted* $\sim$*, be the largest bisimulation.*

We say that two closed terms t and q are bisimilar if $s\vdash t\sim q:\mathbb{P}$ for some s and $\mathbb{P}$.

In the definition of bisimulation, the universal quantification on sets of names s' is required, otherwise we would relate

$$\{a\}\vdash\lambda\alpha.[\alpha!\mathbf{0}>a!x\Rightarrow\mathbf{!0}]:\mathbb{N}\otimes\mathbf{!0} \qquad \text{and} \qquad \{a\}\vdash\lambda\alpha.!\mathbf{0}:\mathbb{N}\otimes\mathbf{!0}$$

while the two terms above behave differently in a world where a is not the only current name.

Using an extension of Howe's method [6] as adapted by Gordon and Pitts to a typed setting [5, 10], we show that bisimilarity is preserved by well typed contexts.

Theorem 3.2 *Bisimilarity* $\sim$ *is an equivalence relation and a congruence.*

Proposition 3.3 *For closed, well-formed, terms we have*

$$s \vdash (\lambda x.t)u \sim t[u/x] : \mathbb{P} \qquad s \vdash (\lambda \alpha.t)a \sim t[a/\alpha] : \mathbb{P}$$
$$s \vdash \lambda x.(tx) \sim t : \mathbb{P} \to \mathbb{Q} \qquad s \vdash \lambda \alpha.(t\alpha) \sim t : \mathbb{N} \to \mathbb{P}$$
$$s \vdash \lambda x.(\Sigma_{i\in I} t_i) \sim \Sigma_{i\in I}(\lambda x.t_i) : \mathbb{P} \to \mathbb{Q} \qquad s \vdash \lambda \alpha.(\Sigma_{i\in I} t_i) \sim \Sigma_{i\in I}(\lambda \alpha.t_i) : \mathbb{N} \to \mathbb{P}$$
$$s \vdash (\Sigma_{i\in I} t_i)u \sim \Sigma_{i\in I}(t_i u) : \mathbb{P} \qquad s \vdash (\Sigma_{i\in I} t_i)a \sim \Sigma_{i\in I}(t_i a) : \mathbb{P}$$

$$s \vdash \pi_\beta(\beta \cdot t) \sim t : \mathbb{P} \qquad s \vdash \pi_\beta(\alpha \cdot t) \sim \mathbf{0} : \mathbb{P}$$
$$s \vdash t \sim \Sigma_{\alpha\in\mathbb{N}} \alpha \cdot (\pi_\alpha t) : \mathbb{N} \otimes \mathbb{P}$$
$$s \vdash \beta \cdot (\Sigma_{i\in I} t_i) \sim \Sigma_{i\in I} \beta \cdot t_i : \mathbb{P} \qquad s \vdash \pi_\beta(\Sigma_{i\in I} t_i) \sim \Sigma_{i\in I} \pi_\beta t_i : \mathbb{P}$$

$$s \vdash [!u > !x \Rightarrow t] \sim t[u/x] : \mathbb{P} \qquad s \vdash [\Sigma_{i\in I} u_i > !x \Rightarrow t] \sim \Sigma_{i\in I}[u_i > !x \Rightarrow t] : \mathbb{P}$$

Proposition 3.4 *Bisimilarity validates β-reduction on new-name abstraction:*

$$s \dot\cup \{a\} \vdash (new\alpha.t)[a] \sim t[a/\alpha] : \mathbb{P} \ .$$

Corollary 3.5 *If $s \vdash new\alpha.t \sim new\alpha.u : \delta\mathbb{P}$ then $s \dot\cup \{a\} \vdash t[a/\alpha] \sim u[a/\alpha] : \mathbb{P}$ for all $a \notin \mathsf{n}(t, u)$.*

4 Examples

In this section, we illustrate how new-HOPLA can be used to give semantics to well-known process algebras. We define a 'fully abstract' encoding of π-calculus that preserves and reflects both the reduction relation and strong bisimilarity. We also report an encoding of Higher-Order π-calculus. Encodings of polyadic π-calculus and of Mobile Ambients can be found in [15].

We introduce an useful product type $\mathbb{P}\,\&\,\mathbb{Q}$, which is not primitive in new-HOPLA. It is definable as $1{:}\mathbb{P} + 2{:}\mathbb{Q}$. The projections are given by $fst(t) = \pi_1(t)$ and $snd(t) = \pi_2(t)$, while pairing is defined as $(t, u) = 1{:}t + 2{:}u$. For actions $(p, -) = 1{:}p$, $(-, q) = 2{:}q$. It is then easy to verify that $s \vdash fst(t, u) \sim t : \mathbb{P}$, that $s \vdash snd(t, u) \sim u : \mathbb{Q}$, and that $s \vdash (fst(t, u), snd(t, u)) \sim (t, u) : \mathbb{P}\,\&\,\mathbb{Q}$, for all $s \supseteq \mathsf{n}(t) \cup \mathsf{n}(u)$.

4.1 The late semantics of π-calculus

We denote *name constants* with $a, b, \ldots$, and *name variables* with $\alpha, \beta, \ldots$; the letters $n, m, \ldots$ range over both name constants and name variables. The terms of the language are constructed according the following grammar:

$$P, Q \ ::= \ \mathbf{0} \ \mid \ P \mid Q \ \mid \ (\boldsymbol{\nu}\alpha)P \ \mid \ \overline{n}m.P \ \mid \ n(\alpha).P \ .$$

The late labelled transition system (denoted $\xrightarrow{\alpha}_l$) and the definition of strong late bisimulation (denoted $\sim_l$) are standard [12].

We can specify a type $\mathbb{P}$ as

$$\mathbb{P} \ = \ \tau{:}!\mathbb{P} \ + \ \mathsf{out}{:}\mathbb{N} \otimes \mathbb{N} \otimes !\mathbb{P} \ + \ \mathsf{bout}{:}\mathbb{N} \otimes !(\delta\mathbb{P}) \ + \ \mathsf{inp}{:}\mathbb{N} \otimes !(\mathbb{N} \to \mathbb{P}) \ .$$

The terms of π-calculus can be expressed in new-HOPLA as the following terms of type $\mathbb{P}$:

$$[\![\mathbf{0}]\!] = \mathbf{0} \qquad [\![\overline{n}m.P]\!] = \mathsf{out}{:}n \cdot m \cdot ![\![P]\!] \qquad [\![n(\beta).P]\!] = \mathsf{inp}{:}n \cdot !(\lambda\beta.[\![P]\!])$$
$$[\![(\boldsymbol{\nu}\alpha)P]\!] = Res\,(new\alpha.[\![P]\!]) \qquad [\![P \mid Q]\!] = [\![P]\!] \ \| \ [\![Q]\!]$$

Here, $Res : \delta\mathbb{P} \to \mathbb{P}$ and $|| : \mathbb{P}\&\mathbb{P} \to \mathbb{P}$ (we use infix notation for convenience) and are abbrevations for the following recursively defined processes:

$$\begin{array}{rcl}
Res\ t & = & [t > new\alpha.\tau{:}!(x[\alpha])) \Rightarrow \tau{:}!Res\, x] \\
& + & \Sigma_{\beta\in\mathbb{N}}\Sigma_{\gamma\in\mathbb{N}}[t > new\alpha.\mathsf{out}{:}\beta\cdot\gamma\cdot!(x[\alpha]) \Rightarrow \mathsf{out}{:}\beta\cdot\gamma\cdot!Res\, x] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > new\alpha.\mathsf{out}{:}\beta\cdot\alpha\cdot!(x[\alpha]) \Rightarrow \mathsf{bout}{:}\beta\cdot!x] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > new\alpha.\mathsf{bout}{:}\beta\cdot!(x[\alpha]) \Rightarrow \mathsf{bout}{:}\beta\cdot!new\gamma\cdot Res\,(new\eta.x[\eta][\gamma])] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > new\alpha.\mathsf{inp}{:}\beta\cdot!(x[\alpha]) \Rightarrow \mathsf{inp}{:}\beta\cdot!\lambda\gamma.Res\,(new\eta.x[\eta](\gamma))] \\
\\
t\ ||\ u & = & [t > \tau{:}!x \Rightarrow \tau{:}!(x\ \ ||\ \ u)] \\
& + & \Sigma_{\beta\in\mathbb{N}}\Sigma_{\gamma\in\mathbb{N}}[t > \mathsf{out}{:}(\beta\cdot\gamma\cdot!x) \Rightarrow [u > \mathsf{inp}{:}(\beta\cdot!y) \Rightarrow \tau{:}!(x\ \ ||\ \ y\gamma)]] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{bout}{:}(\beta\cdot!x) \Rightarrow [u > \mathsf{inp}{:}(\beta\cdot!y) \Rightarrow \tau{:}!Res\,(new\eta.(x[\eta]\ \ ||\ \ y\eta))]] \\
& + & \Sigma_{\beta\in\mathbb{N}}\Sigma_{\gamma\in\mathbb{N}}[t > \mathsf{out}{:}\beta\cdot\gamma\cdot!x \Rightarrow \mathsf{out}{:}\beta\cdot\gamma\cdot!(x\ \ ||\ \ u)] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{bout}{:}\beta\cdot!x \Rightarrow \mathsf{bout}{:}\beta\cdot!new\eta.(x[\eta]\ \ ||\ \ u)] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{inp}{:}\beta\cdot!x \Rightarrow \mathsf{inp}{:}\beta\cdot!\lambda\eta.(x(\eta)\ \ ||\ \ u)] \\
& + & \text{symmetric cases}
\end{array}$$

where η is chosen such that $\eta \notin \mathsf{n}(u)$. Informally, the restriction map $Res : \delta\mathbb{P} \to \mathbb{P}$ pushes restrictions inside processes as far as possible. The five summands corresponds to the five equations below:

$$(\boldsymbol{\nu}\alpha)\tau.P \sim_l \tau.(\boldsymbol{\nu}\alpha)P$$

$$(\boldsymbol{\nu}\alpha)\overline{m}n.P \sim_l \overline{m}n.(\boldsymbol{\nu}\alpha)P \ \text{ if } \alpha \neq m, n \qquad (\boldsymbol{\nu}\alpha)\overline{m}\alpha.P \sim_l \overline{m}(\alpha).P \ \text{ if } \alpha \neq m$$

$$(\boldsymbol{\nu}\alpha)\overline{m}(\beta).P \sim_l \overline{m}(\beta).(\boldsymbol{\nu}\alpha)P \ \text{ if } \alpha \neq m \qquad (\boldsymbol{\nu}\alpha)m\beta.P \sim_l m\beta.(\boldsymbol{\nu}\alpha)P \ \text{ if } \alpha \neq m$$

where $\overline{m}(\alpha)$ is an abbreviation to express bound-output, that is, $(\boldsymbol{\nu}\alpha)\overline{m}\alpha$. The map Res implicitly also ensures that $(\boldsymbol{\nu}\alpha)P \sim_l 0$ if none of the above cases applies. The parallel composition map $||$ captures the *(late) expansion law* of π-calculus. There is a strong correspondence between actions performed by a closed π-calculus process and the actions of its encoding.

Theorem 4.1 *Let P a closed π-calculus process. If $P \xrightarrow{\tau}_l P'$ is derivable in π-calculus, then $\mathsf{n}([\![P]\!]) \vdash [\![P]\!] \xrightarrow{\tau:!} \sim [\![P']\!]$. Conversely, if $\mathsf{n}([\![P]\!]) \vdash [\![P]\!] \xrightarrow{\tau:!} t$ in new-HOPLA, then $P \xrightarrow{\tau}_l P'$ for some P', and $\mathsf{n}(t) \vdash t \sim [\![P']\!] : \mathbb{P}$.*

The encoding also preserves and reflects late strong bisimulation.

Theorem 4.2 *Let P and Q be two closed π-calculus processes. If $P \sim_l Q$ then $\mathsf{n}(P) \cup \mathsf{n}(Q) \vdash [\![P]\!] \sim [\![Q]\!] : \mathbb{P}$. Conversely, if $\mathsf{n}([\![P]\!]) \cup \mathsf{n}([\![Q]\!]) \vdash [\![P]\!] \sim [\![Q]\!] : \mathbb{P}$, then $P \sim_l Q$.*

Along the same lines, new-HOPLA can encode the early semantics of π-calculus. The type of the input action assigned to π-calculus terms captures the difference between the two semantics. In the late semantics a process performing an input action has type $\mathsf{inp}{:}\mathbb{N} \otimes !(\mathbb{N} \to \mathbb{P})$: the type of the continuation $(\mathbb{N} \to \mathbb{P})$ ensures that the continuation is actually an *abstraction* that will be instantiated with the received name when interaction takes place. In the early semantics, the type of a process performing an input action is changed into $\mathsf{inp}{:}\mathbb{N} \otimes \mathbb{N} \to !\mathbb{P}$. Performing an input action now involves picking up a name before executing the prototypical action, and in the continuation (whose type is $\mathbb{P}$) the formal variable has been instantiated with the received name. Details can be found in [15].

4.2 Higher-Order π-calculus

The language we consider can be found in [12], with one main difference: rather than introducing a unit value, we allow processes in addition to abstractions to be communicated. For brevity, we gloss over typing issues. The syntax of terms and values is defined below.

$$P, Q \quad ::= \quad V \bullet W \;\mid\; n(x).P \;\mid\; \overline{n}(V).P \;\mid\; P \mid Q \;\mid\; x \;\mid\; (\boldsymbol{\nu}\alpha)P \;\mid\; \mathbf{0}$$

$$V, W \quad ::= \quad P \;\mid\; (x).P$$

The reduction semantics for the language is standard [12]; we only recall the axioms that define the reduction relation:

$$(x).P \bullet V \;\rightarrowtriangle\; P[V/x] \qquad\qquad \overline{n}(V).P \mid n(x).Q \;\rightarrowtriangle\; P \mid Q[V/x]\ .$$

Types for HOπ are given recursively by

$$\mathbb{P} = \tau{:}!\mathbb{P} \;+\; \mathsf{out}{:}\mathbb{N} \otimes\, !\mathbb{C} \;+\; \mathsf{inp}{:}\mathbb{N} \otimes\, !(\mathbb{F} \to \mathbb{P}) \qquad \mathbb{C} = 0{:}\mathbb{F}\&\mathbb{P} + \;1{:}\delta\mathbb{C} \qquad \mathbb{F} = 2{:}\mathbb{P} + 3{:}\mathbb{F} \to \mathbb{P}\ .$$

Concretions of the form $(\boldsymbol{\nu}\tilde{\alpha})\langle V\rangle P$ correspond to terms of type $\mathbb{C}$; recursion on types is used to encode the tuple of restricted names $\tilde{\alpha}$. The function $[\![-]\!]_v$ translates values into the following terms of type $\mathbb{F}$:

$$[\![P]\!]_v = 2{:}[\![P]\!] \qquad [\![(x).P]\!]_v = 3{:}\lambda x.[\![P]\!]$$

while the function $[\![-]\!]$ translates processes into terms of type $\mathbb{P}$:

$$[\![V \bullet W]\!] = \tau{:}!(\pi_3[\![V]\!]_v)(\pi_2[\![W]\!]_v + \pi_3[\![W]\!]_v) \qquad [\![P \mid Q]\!] = [\![P]\!] \;||\; [\![Q]\!] \qquad [\![x]\!] = x \qquad [\![\mathbf{0}]\!] = \mathbf{0}$$

$$[\![n(x).P]\!] = \mathsf{inp}{:}n \cdot !(\lambda x.[\![P]\!]) \qquad [\![\overline{n}(V)]\!] = \mathsf{out}{:}n \cdot !([\![V]\!]_v, [\![P]\!]) \qquad [\![(\boldsymbol{\nu}\alpha)P]\!] = \mathit{Res}\ \mathit{new}\alpha.[\![P]\!]\ .$$

The restriction map $\mathit{Res} : \delta\mathbb{P} \to \mathbb{P}$ filters the actions that a process emits, blocking actions that refer to the name that is being restricted. Output actions cause names to be extruded: the third summand records these names in the appropriate concretion.

$$\begin{array}{rcl}
\mathit{Res}\ t & = & [t > \mathit{new}\alpha.\tau{:}!x[\alpha] \Rightarrow \tau{:}!\mathit{Res}\ x] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{inp}{:}(\beta \cdot !x[\alpha]) \Rightarrow \mathsf{inp}{:}(\beta \cdot !\lambda y.\mathit{Res}\ (\mathit{new}\gamma.x[\gamma](y)))] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{out}{:}(\beta \cdot !x[\alpha]) \Rightarrow \mathsf{out}{:}(\beta \cdot !3{:}x)]
\end{array}$$

Parallel composition is a family of mutually dependent operations also including components such as $||_i$ of type $\mathbb{C}\&\mathbb{F} \to \mathbb{P}$ to say how values 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:

— *Processes in parallel with processes:*

$$\begin{array}{rcl}
t \;||\; u & = & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{out}{:}\beta \cdot !x \Rightarrow [u > \mathsf{inp}{:}\beta \cdot !y \Rightarrow \tau{:}!(x \;||\; y)]] \\
& + & \Sigma_{\beta\in\mathbb{N}}[u > \mathsf{inp}{:}\beta \cdot !y \Rightarrow \mathsf{inp}{:}\beta \cdot !(t \;||_a\; y)] \\
& + & \Sigma_{\beta\in\mathbb{N}}[u > \mathsf{out}{:}\beta \cdot !y \Rightarrow \mathsf{out}{:}\beta \cdot !(t \;||_c\; y)] \\
& + & [u > \tau{:}!y \Rightarrow \tau{:}!(t \;||\; y)] \\
& + & \text{symmetric cases}
\end{array}$$

— *Concretions in parallel with values*

$$c \;||_i\; f \;=\; \mathit{snd}(\pi_0 c) \;||\; (\pi_3 f)(\pi_2(\mathit{fst}(\pi_0 c)) + \pi_3(\mathit{fst}(\pi_0 c))) + \mathit{Res}\ (\mathit{new}\alpha.(((\pi_1 c)[\alpha]) \;||_i\; f))$$

— *Concretions in parallel with processes*

$$c \;||_c\; t \;=\; 0{:}(fst(\pi_0 c), snd(\pi_0 c) \;||\; t) + 1{:}(new\alpha.((\pi_1 c)[\alpha] \;||_c\; t))$$

— *Values in parallel with processes*

$$f \;||_a\; t \;=\; \lambda x.(((\pi_3 f)x) \;||\; u)$$

The remaining cases are given symmetrically. The proposed encoding agrees with the reduction semantics of HOπ. The resulting bisimulation is analogous to the so called *higher-order bisimulation* [1, 14], and as such it is strictly finer than observational equivalence. It is an open problem whether it is possible to provide an encoding of HOπ that preserves and reflects the natural observational equivalence.

References

[1] G. Boudol. Towards a lambda calculus for concurrent and communicating systems. In *Proc. TAPSOFT '89*, volume 351 of *LNCS*, pages 149–161. Springer Verlag, 1989.

[2] G. L. Cattani, I. Stark, and G. Winskel. Presheaf models for the π-calculus. In *Proc. CTCS'97*, volume 1290 of *LNCS*. Springer Verlag, 1997.

[3] M. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the π-calculus. In *Proc. 11th LICS*. IEEE Computer Society Press, 1996.

[4] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[5] A. D. Gordon. Bisimilarity as a theory of functional programming: mini-course. Notes Series BRICS-NS-95-3, BRICS, Department of Computer Science, University of Aarhus, July 1995.

[6] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.

[7] M. Nygaard and G. Winskel. Hopla—a higher-order process language. In *Proc. CONCUR'02*, volume 2421 of *LNCS*. Springer Verlag, 2002.

[8] M. Nygaard and G. Winskel. Domain theory for concurrency. To appear in *Theoretical Computer Science*, special issue on domain theory, 2003.

[9] M. Nygaard and G. Winskel. Full abstraction for HOPLA. In *Proc. CONCUR'03*, LNCS. Springer Verlag, 2003.

[10] A. M. Pitts. Operationally-based theories of program equivalence. In P. Dybjer and A. M. Pitts, editors, *Semantics and Logics of Computation*, Publications of the Newton Institute, pages 241–298. Cambridge University Press, 1997.

[11] A. M. Pitts and M. J. Gabbay. A metalanguage for programming with bound names modulo renaming. In *Proc. MPC 2000*, volume 1837 of *LNCS*. Springer Verlag, 2000.

[12] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[13] I. Stark. A fully-abstract domain model for the π-calculus. In *Proc. 11th LICS*. IEEE Computer Society Press, 1996.

[14] B. Thomsen. *Calculi for Higher Order Communicating Systems*. PhD thesis, Department of Computing, Imperial College, 1990.

[15] F. Zappa Nardelli. *De la sémantique des processus d'ordre supérieur*. PhD thesis, Université de Paris 7, 2003. Available in English from `http://www.di.ens.fr/~zappa`.