On the Expressiveness of Pure Mobile Ambients

Pascal Zimmer [1]

INRIA Sophia Antipolis, 2004 route des Lucioles – BP 93
06902 SOPHIA ANTIPOLIS, FRANCE
Email: Pascal.Zimmer@sophia.inria.fr

Abstract

We consider the *Pure Ambient Calculus*, which is Cardelli and Gordon's *Ambient Calculus* (or more precisely its *safe* version by Levi and Sangiorgi) restricted to its mobility primitives, and we focus on its expressive power. Since it has no form of communication or substitution, we show how these notions can be simulated by mobility and modifications in the hierarchical structure of ambients. As an example, we give an encoding of the synchronous π-calculus into pure ambients and we state an operational correspondence result. In order to simplify the proof and give an intuitive understanding of the encoding, we design an intermediate language: the *π-Calculus with Explicit Substitutions and Channels*, which is a syntactic extension of the π-calculus with a specific operational semantics.

1 Introduction

The *ambient calculus* [3,4] was designed to model within a single framework both *mobile computing*, that is to say computation in mobile devices like a laptop, and *mobile computation*, that is to say mobile code moving between different devices, like applets or agents. It also shows how the notions of administrative domains, firewalls, authorizations... can be formalized in a calculus (for more discussion about the problems raised by mobility and computation over wide-area networks, see [1,2]). Informally, an ambient is a bounded place where computation happens. Ambients can be nested so as to form a hierarchy. Each of them has a name (not necessarily distinct from other ambient names), which will be used to control access. An ambient can be moved as a whole with all the computations and subambients it contains: it can enter another ambient or exit it. It can also be opened so that its contents get visi-

[1] Partially supported by the Ecole Normale Supérieure de Lyon, FRANCE

This is a preliminary version. The final version can be accessed at
URL: http://www.elsevier.nl/locate/entcs/volume39.html

ble at the current level, and communication between two processes can occur within an ambient (like in the π-calculus).

The purpose of this paper is to study the expressive power of the subcalculus obtained by removing all communication primitives, the *pure ambient calculus*. This subcalculus has no abstraction at all: it has neither output nor input prefix, no variable binding, no communication rule, and it cannot perform any substitution of variables globally in a process. Consequently, the only "tools" allowed are the hierarchical structure of ambients, their movements and openings. One can wonder what were the motivations for studying pure ambients. We wanted to understand what made the ambient calculus so expressive and which constructs were really important from a purely theoretical point of view. A similar question arose in previous work in the setting of the π-calculus [12]. After all, the pure ambient calculus is to the classical ambient calculus what CCS is to the π-calculus: the former has no operator of abstraction and no instantiation of variables, while the latter does.

As a first step in this direction, we managed to encode the finite sum-free synchronous π-calculus [11] in pure ambients. We give such an encoding at the end of this paper. The main problem we had to face was the simulation of substitution: the communication rule of the π-calculus binds a variable x to an output value m and performs this substitution in the continuation process in one single step. With pure ambients, we need to adopt another mechanism: every future reference to x has to be replaced dynamically by a reference to m. For this purpose, we create an ambient x acting as a "forwarder". Furthermore, we introduce explicit channels in the form of unique ambients for each channel name, so that matching input and output primitives can meet somewhere.

Concerning expressivity, it has been shown in [4] that mobile ambients without communication primitives were expressive enough to simulate Turing machines. However, Turing machines are a good model for sequential programming but are not well adapted in a concurrency framework. What we want is a "reasonable" encoding having at least the property of compositionality (i.e. such that $\langle\!\langle op(P_1, \ldots, P_n) \rangle\!\rangle$ is a function of $\langle\!\langle P_1 \rangle\!\rangle, \ldots, \langle\!\langle P_n \rangle\!\rangle$ for any operator op), which would not be the case if we had used an encoding via Turing machines (CCS is also Turing-complete, yet the π-calculus is much more powerful).

As a target calculus, we used *safe ambients*, which were first presented in [8]. They differ from the classical mobile ambients by the addition of *coactions*. In the ambient calculus, a movement is initiated only by the moving ambient and the target ambient has no control over it. On the contrary, in safe ambients both participants must agree by using matching action and coaction. In our attempts, it appeared that protocols were much simpler to implement in safe ambients. For example, when designing a communication mechanism based on requests answered by replicated servers (both being ambients), it is difficult to prevent a server from answering twice the same request. In safe ambients, the uniqueness of the answer is easier to achieve if there is only one coaction in each request.

In order to show an operational correspondence between the π-calculus and our encoding, we had to design an intermediate calculus to simplify the proof, the *π-Calculus with Explicit Substitutions and Channels* (π_{esc}-calculus in short). It is an extension of the π-calculus, with new primitives for variables and explicit channels. This appeared to be an interesting side-effect and not only a technical tool: it breaks up the communication and substitution mechanisms of the π-calculus into simpler steps, a few equivalence properties with the π-calculus can be proved, and it allows a better intuitive description of the mechanism underlying the encoding in pure ambients.

Related Work

Some encodings of the π-calculus into ambients have already been proposed in the literature [4,8], but all of them encoded the communications and substitutions of the π-calculus into communications and substitutions of the ambient calculus, whereas our encoding cannot use these mechanisms. Moreover, all of them encoded only the *asynchronous* π-calculus (π_a) and could not be easily extended so as to encode its synchronous version. Finally, except for the encoding of Levi and Sangiorgi [8], no operational correspondence result was ever completely proved for any of them.

For some restrictions of the π-calculus, substitution can be simulated in a different way. The *local* π ($L\pi$) [10] is an asynchronous π-calculus (without matching) with an additional constraint on the input construct $n(x).P$: x may not occur free in P in input position. In this calculus, the following is a correct algebraic law:

$$P\{{}^{b}/_{c}\} = (\nu c)\ (P \mid c \lhd b)$$

where c may not be free in P in input position, $b \neq c$ and $c \lhd b \triangleq\ !c(x).\overline{b}\langle x\rangle$ is a link forwarding every message for c to b. Note that this law is false in the full π_a-calculus, hence also in the π-calculus.

In the same way, an *equator* was first defined in [7] by:

$$\mathcal{E}(b,c) \triangleq b \lhd c \mid c \lhd b$$

and it was shown in [9] that

$$P\{{}^{b}/_{c}\} \cong_{\pi_a} (\nu c)\ (\mathcal{E}(b,c) \mid P)$$

($\cong_{\pi_a}$ being barbed congruence in the π_a-calculus). However, this equality is false in the full synchronous π-calculus because the use of forwarders breaks the sequentiality imposed by output prefixing. Moreover, even if those two laws show a relationship between substitution and other operators of the π-calculus, they are not encodings of substitutions.

Some variants of the π-calculus with explicit substitutions were also proposed. In the $\pi\xi$-calculus [5], processes are prefixed by a global environment

ξ which contains the name associations carried on in past communications. The main rule is:

$$\frac{P \xrightarrow{\omega} P'}{\xi :: P \overset{\delta(\xi,\xi',\omega)}{\longmapsto} \xi' :: P'} \qquad \text{with} \;\; \xi' \in \eta(\xi, \omega)$$

where the functions δ and η are defined according to the desired semantics (late, early, open), such that the environment ξ is extended with the name associations activated by the transition $P \xrightarrow{\omega} P'$. The main difference of this approach with our π_{esc}-calculus is that there is only one global environment outside the process, instead of multiple variables directly included in the syntax and taking advantage of name restriction. Moreover, in the $\pi\xi$-calculus, substitutions are performed outside the term (in $\delta(\xi, \xi', \omega)$) and are not included in the reductions.

Another variant is the calculus of explicit substitutions $\pi\sigma$ from [6], in which a rewrite system is used to perform name substitutions inside terms. Since processes are written in De Bruijn notation, this calculus looks very different from the π_{esc}-calculus. Furthermore, it performs substitutions in the whole output term (the rule is $(\overline{a}b)[s] \to \overline{a[s]}b[s])$, so that the transitive closure of substitutions is automatically computed, whereas in the π_{esc}-calculus, an arbitrary long chain of variables can be created. Moreover, the operational semantics of both $\pi\xi$ and $\pi\sigma$ are defined via a labelled transition system, whereas our calculus uses CHAM-style rules, and none of them introduces explicit channels in its syntax.

A final remark is that all dialects and variants of the π-calculus which have been studied so far have a construct for abstraction (usually embodied in the input prefix), hence computation involves some form of substitution. For us, the challenge consisted precisely in the fact that we did not have any such operator.

Outline

In Section 2, we give the necessary background on the π-calculus and safe ambients. We also introduce a special kind of substitution. In Section 3, we present extensively the π_{esc}-calculus and some associated tools. Section 4 defines encodings between the π-calculus and the π_{esc}-calculus, states the main relations between them and gives an overview of the proofs. The second part of the encoding, from the π_{esc}-calculus into pure ambients, is given in Section 5, together with an operational correspondence result. Finally, Section 6 gathers the results into a main theorem and gives the final encoding for the π-calculus. Proofs of the results stated in this paper should soon be available as a technical report [13].

2 Background

2.1 The π-Calculus

We start by reviewing the syntax of the monadic synchronous π-calculus we will use throughout the paper.

We will need to distinguish between names of channels and names of variables. For this reason, let $Name$ be a denumerably infinite set of names of channels (ranged over by n, m, p, $\ldots$), and Var a denumerably infinite set of names of variables (ranged over by x, y, $\ldots$).

The syntax of the π-calculus is then defined as follows.

$$\begin{array}{lllll} P ::= (\nu n)\ P & \text{restriction} & \qquad & M ::= n \in Name & \text{channel name} \\ \quad \mid \mathbf{0} & \text{nil process} & & \quad \mid x \in Var & \text{variable name} \\ \quad \mid P \mid Q & \text{parallel composition} & & & \\ \quad \mid\ !P & \text{replication} & & & \\ \quad \mid \overline{M}\langle M' \rangle.P & \text{output} & & & \\ \quad \mid M(x).P & \text{input} & & & \end{array}$$

In $(\nu n)\ P$ (resp. $M(x).P$), the name n (resp. x) is bound in P. We can always change this name using α-conversion, and we will consider that the resulting process is equal to the first one. If a name is not bound, it is called free. The set of free channel names (resp. free variable names) of P is denoted by $fn(P)$ (resp. $fv(P)$).

Below is the operational semantics of our π-calculus, given in the form of an one-step reduction relation, written $\longrightarrow$. The main rule is (π Red Comm) in which an input prefix and an output prefix on a same channel n are consumed, whereas the variable x is replaced by the value m (the construction $Q\{^m/_x\}$ is defined as the result of replacing each free occurrence of x in Q by m).

$$\frac{}{\overline{n}\langle m \rangle.P \mid n(x).Q \ \longrightarrow\ P \mid Q\{^m/_x\}}\ (\pi \text{ Red Comm})$$

$$\frac{P \ \longrightarrow\ P'}{(\nu n)\ P \ \longrightarrow\ (\nu n)\ P'}\ (\pi \text{ Red Res}) \qquad \frac{P \ \longrightarrow\ P'}{P \mid Q \ \longrightarrow\ P' \mid Q}\ (\pi \text{ Red Par})$$

$$\frac{P \equiv P' \qquad P' \ \longrightarrow\ Q' \qquad Q' \equiv Q}{P \ \longrightarrow\ Q}\ (\pi \text{ Red Struct})$$

This one-step reduction makes use of a *structural congruence* rewriting relation $\equiv$. Its definition is standard, with rules to commute processes in parallel, to change the scope of a restriction operator, unfold a replicated process, ... Its rules are given below.

$P \equiv P$	(π Struct Refl)
$P \equiv Q \;\Rightarrow\; Q \equiv P$	(π Struct Symm)
$P \equiv Q \equiv R \;\Rightarrow\; P \equiv R$	(π Struct Trans)
$P \equiv Q \;\Rightarrow\; (\nu n)\ P \equiv (\nu n)\ Q$	(π Struct Res)
$P \equiv Q \;\Rightarrow\; P \mid R \equiv Q \mid R$	(π Struct Par)
$P \equiv Q \;\Rightarrow\; !P \equiv !Q$	(π Struct Repl)
$P \equiv Q \;\Rightarrow\; \overline{M}\langle M'\rangle.P \equiv \overline{M}\langle M'\rangle.Q$	(π Struct Output)
$P \equiv Q \;\Rightarrow\; M(x).P \equiv M(x).Q$	(π Struct Input)
$P \mid \mathbf{0} \equiv P$	(π Struct Par Zero)
$P \mid Q \equiv Q \mid P$	(π Struct Par Comm)
$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$	(π Struct Par Assoc)
$(\nu n)\ (P \mid Q) \equiv P \mid (\nu n)\ Q \quad \text{if} \quad n \notin fn(P)$	(π Struct Res Par)
$(\nu n)\ (\nu m)\ P \equiv (\nu m)\ (\nu n)\ P$	(π Struct Res Res)
$!P \equiv P \mid !P$	(π Struct Repl Par)
$!\mathbf{0} \equiv \mathbf{0}$	(π Struct Repl Zero)

2.2 Pure Ambients

We present here the variant of the Safe Ambient Calculus we will use. It corresponds to the original Safe Ambients from [8] with the communication primitives removed. This restriction allows us to simplify the syntax (the original one needed a type system to reject some ill-formed terms). The complete syntax is defined as follows.

$P ::= (\nu n)\ P$	restriction	$Cap ::=$	$in\ n$	entering
$\mid \mathbf{0}$	nil process	$\mid$	$\overline{in}\ n$	co-entering
$\mid P \mid Q$	parallel composition	$\mid$	$out\ n$	exiting
$\mid\ !P$	replication	$\mid$	$\overline{out}\ n$	co-exiting
$\mid n[P]$	ambient	$\mid$	$open\ n$	opening
$\mid Cap.P$	capability	$\mid$	$\overline{open}\ n$	co-opening

The basic constructs of concurrency calculi are present: restriction of names, nil process, parallel composition and replication. They behave as in the π-calculus. An ambient is written $n[P]$ where n is the name of the ambient and P is the process running inside it. Actions are called *capabilities* and are written $Cap.P$. There are three possible capabilities: one to enter an ambient ($in\ n$), one to exit an ambient ($out\ n$) and one to open an ambient ($open\ n$), each of them having a corresponding *cocapability* (namely $\overline{in}\ n$, $\overline{out}\ n$ and $\overline{open}\ n$). In order for a movement to take place, two corresponding capability and cocapability (that is, with the same name) must be present at the right place, as shown by the following reduction rules:

$$n[\ in\ m\ .\ P \mid Q] \mid m[\ \overline{in}\ m\ .R \mid S] \ \hookrightarrow\ m[n[P \mid Q] \mid R \mid S] \qquad \text{(SA In)}$$

$$m[n[\ out\ m\ .P \mid Q] \mid \ \overline{out}\ m\ .R \mid S] \ \hookrightarrow\ n[P \mid Q] \mid m[R \mid S] \qquad \text{(SA Out)}$$

$$open\ n\ .P \mid n[\ \overline{open}\ n\ .Q \mid R] \ \hookrightarrow\ P \mid Q \mid R \qquad \text{(SA Open)}$$

The operational semantics is completed by four other rules, so that reduction can occur under restriction, in parallel processes, inside ambients or after a structural congruence rewriting (which is very similar to the structural congruence for the π-calculus).

$$\frac{P\ \hookrightarrow\ Q}{(\nu n)\ P\ \hookrightarrow\ (\nu n)\ Q}\ \text{(SA Res)} \qquad \frac{P\ \hookrightarrow\ Q}{P \mid R\ \hookrightarrow\ Q \mid R}\ \text{(SA Par)}$$

$$\frac{P\ \hookrightarrow\ Q}{n[P]\ \hookrightarrow\ n[Q]}\ \text{(SA Amb)}$$

$$\frac{P \equiv P' \qquad P'\ \hookrightarrow\ Q' \qquad Q' \equiv Q}{P\ \hookrightarrow\ Q}\ \text{(SA Struct)}$$

The main difference with the classical ambient calculi is the lack of communication primitives, namely the asynchronous output $\langle M \rangle$ and the input binder $(x).P$. Furthermore, cocapabilities were not to be found in the original presentation of ambients [4].

2.3 Substitutions

In this Section, we introduce a special kind of substitution, having a tree structure. We will keep the same term, but every occurrence of "substitution" in the rest of the paper refers to the following definition.

Definition 2.1 A substitution is a partial application $\sigma : Var \to Var \cup Name$ such that:

- $\forall x \in dom(\sigma), x\sigma \in Name \cup dom(\sigma)$ (i.e. $im(\sigma) \subseteq Name \cup dom(\sigma)$)
- $\forall x \in dom(\sigma)$, there is $k \in \mathbb{N}^*$ such that $x\sigma^k \in Name$ (i.e. there are no cycles)

Let us define the graph of a substitution: its set of vertices is $dom(\sigma) \cup Name$ and its edges are $(x, x\sigma)$ for $x \in dom(\sigma)$. With the above definition, one can easily show that the graph of a substitution has a forest structure (a set of trees), with roots in $Name$ and all other nodes in $dom(\sigma) \subseteq Var$. Consequently, we can define $\sigma^* : dom(\sigma) \to Name$ the transitive closure of σ, associating to each variable the name at the root of the corresponding tree.

If $x \notin dom(\sigma)$ and $M \in Name \cup dom(\sigma)$, we define an extension of σ, written $\sigma' = \{^M/_x\} \uplus \sigma$, by $x\sigma' = M$ and $y\sigma' = y\sigma$ for $y \neq x$. It is easy to check that σ' is still a correct substitution.

The empty substitution is written $\varnothing$, and we also define $fn(\sigma) \triangleq im(\sigma) \cap Name$. Moreover, we extend naturally the domain of substitutions so that we can apply them to processes.

3 The Intermediate Calculus π_{esc}

In this Section, we introduce our π-Calculus with Explicit Substitutions and Channels.

3.1 Syntax

Syntactically, the π_{esc}-calculus is an extension of the π-calculus, with supplementary constructs to handle substitutions and channels. Its complete definition follows:

$P ::= (\nu n)\ P$	restriction	$M ::= n \in Name$	channel name
$\mid \mathbf{0}$	nil process	$\mid x \in Var$	variable name
$\mid P \mid Q$	parallel comp.		
$\mid\ !P$	replication	$S ::= \varepsilon$	empty channel
$\mid \overline{M}\langle M' \rangle.P$	output	$\mid S \mid S'$	parallel comp.
$\mid M(x).P$	input	$\mid \langle M \rangle.P$	concretion
$\mid [n : S]$	explicit channel	$\mid (x).P$	abstraction
$\mid (\nu x : M)\ P$	explicit variable with $x \neq M$		

First, the construction $(\nu x : M)\ P$ (with $x \neq M$) represents a new variable x whose content is M. The name x is bound in P (as n is bound in $(\nu n)\ P$). Intuitively, every free occurrence of the name x in P refers to this variable and can be replaced by M without changing the behaviour of the process P.

The construction $[n : S]$ represents an explicit channel of name n, whose content is a set S of abstractions and concretions performed on that channel. More precisely, S is not exactly a set but a parallel composition of abstractions and concretions (we preferred this approach to keep a symmetry with parallel composition for processes). S can be either ε (the empty channel), a parallel composition $S \mid S'$, a concretion $\langle M \rangle.P$ for an output or an abstraction $(x).P$ for an input (they correspond respectively to the processes $\overline{n}\langle M \rangle.P$ and $n(x).P$). Intuitively, when a process performs an output or input on n, the request is put inside the channel of that name (if there is one).

3.2 Reduction Rules

We give now an operational semantics for our calculus. Reduction rules are of the form $\sigma : P \longmapsto P'$ for two processes P and P', and a substitution σ, which acts as an environment containing the values of free variables in P. As a side condition, we restrict the rules' definitions to processes P and substitutions σ such that $fv(P) \subseteq dom(\sigma)$ (so that we can find the value of every free variable appearing in P).

The first two rules allow us to replace an output or input prefix on a variable x by the same prefix on the value M of x. If M is another variable, we would just apply the same rule again (since in this case $M \in dom(\sigma)$ by definition of a substitution). We continue like this until M is a channel name. Note also that we do not perform substitutions on M' in the rule (π_{esc} Red Subst Out).

$$\frac{x\sigma = M}{\sigma : \overline{x}\langle M' \rangle.P \longmapsto \overline{M}\langle M' \rangle.P} \quad (\pi_{esc} \text{ Red Subst Out})$$

$$\frac{x\sigma = M}{\sigma : x(y).P \longmapsto M(y).P} \quad (\pi_{esc} \text{ Red Subst In})$$

The next two rules were already outlined above: if a channel n and a prefixed process on n meet in a parallel composition, the request is put inside the channel (we then omit the name n since all abstractions and concretions in $[n : S]$ must relate to n).

$$\frac{}{\sigma : [n : S] \mid \overline{n}\langle M \rangle.P \longmapsto [n : S \mid \langle M \rangle.P]} \quad (\pi_{esc} \text{ Red Output})$$

$$\frac{}{\sigma : [n : S] \mid n(x).P \longmapsto [n : S \mid (x).P]} \quad (\pi_{esc} \text{ Red Input})$$

When a concretion $\langle M \rangle.P$ and an abstraction $(x).Q$ are present in the same channel, communication can effectively occur. The two continuations P and Q are then placed outside the channel, except that a new variable x with content M is created in front of Q. This is the purpose of the following rule, which corresponds to (π Red Comm) (the side condition $x \neq M$ can always be satisfied by α-conversion on x).

$$\frac{x \neq M}{\sigma : [n : S \mid \langle M \rangle.P \mid (x).Q] \longmapsto [n : S] \mid P \mid (\nu x : M)\ Q} \quad (\pi_{esc} \text{ Red Comm})$$

The next rule allows a reduction to occur under a variable restriction $(\nu x : M)$. The only side-effect is that the binding $\{^M/_x\}$ must be added to the environment σ (the side condition $x \notin dom(\sigma)$ can always be satisfied by α-conversion on x, and the condition $M \in Name \cup dom(\sigma)$ is automatically satisfied because $fv((\nu x : M)\ P) \subseteq dom(\sigma)$).

$$\frac{x \notin dom(\sigma) \qquad \{^M/_x\} \uplus \sigma : P \ \longmapsto\ P'}{\sigma : (\nu x : M)\ P \ \longmapsto\ (\nu x : M)\ P'} \ (\pi_{esc} \text{ Red Var})$$

Finally, the last three rules complete the calculus: reduction can occur under the scope restriction of a channel name, in a parallel composition or by mean of a structural congruence rewriting.

$$\frac{\sigma : P \ \longmapsto\ P'}{\sigma : (\nu n)\ P \ \longmapsto\ (\nu n)\ P'} \ (\pi_{esc} \text{ Red Res})$$

$$\frac{\sigma : P \ \longmapsto\ P'}{\sigma : P \mid Q \ \longmapsto\ P' \mid Q} \ (\pi_{esc} \text{ Red Par})$$

$$\frac{P \equiv P' \qquad \sigma : P' \ \longmapsto\ Q' \qquad Q' \equiv Q}{\sigma : P \ \longmapsto\ Q} \ (\pi_{esc} \text{ Red Struct})$$

The congruence relation $\equiv$ is the same as in the π-calculus, with additional rules for the new constructs and their interaction with the old ones (in particular the scope of $(\nu x : M)$ can be stretched or commuted with (νn) provided that there are no name captures). Here is the list of rules to be added to those of the π-calculus:

$S \equiv S$	$(\pi_{esc}$ Struct Refl)
$S \equiv S' \ \Rightarrow\ S' \equiv S$	$(\pi_{esc}$ Struct Symm)
$S \equiv S' \equiv S'' \ \Rightarrow\ S \equiv S''$	$(\pi_{esc}$ Struct Trans)
$S \equiv S' \ \Rightarrow\ [n : S] \equiv [n : S']$	$(\pi_{esc}$ Struct Channel)
$P \equiv Q \ \Rightarrow\ (\nu x : M)\ P \equiv (\nu x : M)\ Q$	$(\pi_{esc}$ Struct Var)
$S' \equiv S'' \ \Rightarrow\ S \mid S' \equiv S \mid S''$	$(\pi_{esc}$ Struct Abs)
$P \equiv Q \ \Rightarrow\ \langle M \rangle.P \equiv \langle M \rangle.Q$	$(\pi_{esc}$ Struct Out Abs)
$P \equiv Q \ \Rightarrow\ (x).P \equiv (x).Q$	$(\pi_{esc}$ Struct In Abs)
$S \mid \varepsilon \equiv S$	$(\pi_{esc}$ Struct Abs Zero)
$S \mid S' \equiv S' \mid S$	$(\pi_{esc}$ Struct Abs Comm)
$S \mid (S' \mid S'') \equiv (S \mid S') \mid S''$	$(\pi_{esc}$ Struct Abs Assoc)
$(\nu x : M)\ (P \mid Q) \equiv P \mid (\nu x : M)\ Q$ if $x \notin fv(P)$	$(\pi_{esc}$ Struct Var Par)

$$(\nu n)\ (\nu x : M)\ P \equiv (\nu x : M)\ (\nu n)\ P \ \text{ if } n \neq M \qquad (\pi_{esc} \text{ Struct Res Var})$$

$$(\nu x : M)\ (\nu y : M')\ P \equiv (\nu y : M')\ (\nu x : M)\ P$$

$$\text{if } x \neq y,\ x \neq M' \text{ and } y \neq M\ (\pi_{esc} \text{ Struct Var Var})$$

3.3 Channel Presentation and Valid Processes

To follow our intuition, we will need to cut down the set of allowed processes in the π_{esc}-calculus. Indeed, we need to ensure that the channels are well placed and unique. Consider for instance the process $\overline{n}\langle m\rangle.[p : S]$. The channel p would be unreachable, and thus useless, until the output on n has been performed. Consider also the following process:

$$[n : S] \mid [n : S'] \mid \overline{n}\langle m\rangle.P \mid n(x).Q$$

Since there are two channels, the two prefixed processes could go into different ones, leading to

$$[n : S \mid \langle m\rangle.P] \mid [n : S' \mid (x).Q]$$

and communication would never occur between P and Q !

For this reason, we first need to be able to detect a channel. For this purpose, we define a *presentation predicate* $P \Downarrow_1 n$, which means intuitively that a channel $[n : S]$ is present in P and not hidden by scope restriction. The formal definition of this predicate is very easy to write: the only axiom is $[n : S] \Downarrow_1 n$ and all other rules perform only inductive calls (except for $(\nu m)\ P \Downarrow_1 n$ which checks $m \neq n$). Moreover, we will write $pr(P) \triangleq \{n \in Name / P \Downarrow_1 n\}$ the set of channels presented by P.

In the same way, we can easily define another predicate $P \Downarrow_2 n$, meaning that there are two different channels of name n in P. For instance, we would derive $P \mid Q \Downarrow_2 n$ if both $P \Downarrow_1 n$ and $Q \Downarrow_1 n$ hold at the same time.

The exact rules are (for $i = 1, 2$):

$$\frac{P \Downarrow_i n \qquad m \neq n}{(\nu m)\ P \Downarrow_i n}\ (\pi_{esc} \text{ Pres Res}) \qquad \frac{P \Downarrow_i n}{P \mid Q \Downarrow_i n}\ (\pi_{esc} \text{ Pres ParL})$$

$$\frac{Q \Downarrow_i n}{P \mid Q \Downarrow_i n}\ (\pi_{esc} \text{ Pres ParR}) \qquad \frac{P \Downarrow_1 n \qquad Q \Downarrow_1 n}{P \mid Q \Downarrow_2 n}\ (\pi_{esc} \text{ Pres Par}_2)$$

$$\frac{P \Downarrow_i n}{!P \Downarrow_i n}\ (\pi_{esc} \text{ Pres Repl}) \qquad \frac{P \Downarrow_1 n}{!P \Downarrow_2 n}\ (\pi_{esc} \text{ Pres Repl}_2)$$

$$\frac{P \Downarrow_i n}{\overline{M}\langle M'\rangle.P \Downarrow_i n}\ (\pi_{esc} \text{ Pres Output}) \qquad \frac{P \Downarrow_i n}{M(x).P \Downarrow_i n}\ (\pi_{esc} \text{ Pres Input})$$

$$\frac{}{[n : S] \Downarrow_1 n}\ (\pi_{esc} \text{ Pres Channel}_1) \qquad \frac{S \Downarrow_1 n}{[n : S] \Downarrow_2 n}\ (\pi_{esc} \text{ Pres Channel}_2)$$

$$\frac{S \Downarrow_i m}{[n : S] \Downarrow_i m} \ (\pi_{esc} \text{ Pres Channel}) \qquad \frac{P \Downarrow_i n}{(\nu x : M)\ P \Downarrow_i n} \ (\pi_{esc} \text{ Pres Var})$$

$$\frac{S \Downarrow_1 n}{S \mid S' \Downarrow_1 n} \ (\pi_{esc} \text{ Pres AbsL}) \qquad \frac{S' \Downarrow_1 n}{S \mid S' \Downarrow_1 n} \ (\pi_{esc} \text{ Pres AbsR})$$

$$\frac{S \Downarrow_1 n \qquad S' \Downarrow_1 n}{S \mid S' \Downarrow_2 n} \ (\pi_{esc} \text{ Pres Abs}_2) \qquad \frac{P \Downarrow_i n}{\langle M \rangle.P \Downarrow_i n} \ (\pi_{esc} \text{ Pres Out Abs})$$

$$\frac{P \Downarrow_i n}{(x).P \Downarrow_i n} \ (\pi_{esc} \text{ Pres In Abs})$$

The following Lemma will be helpful in the next Section.

Lemma 3.1 $pr(P) \subseteq fn(P)$

Now is the time to define a small type system on processes. We define the predicate $\vdash P : OK$ inductively on P, by checking that channels do not appear after prefixes or replications, and that there is at most one channel after a name restriction.

$$\frac{\vdash P : OK \qquad P \not\Downarrow_2 n}{\vdash (\nu n)\ P : OK} \ (\pi_{esc} \text{ OK Res})$$

$$\frac{}{\vdash \mathbf{0} : OK} \ (\pi_{esc} \text{ OK Zero}) \qquad \frac{\vdash P : OK \qquad \vdash Q : OK}{\vdash P \mid Q : OK} \ (\pi_{esc} \text{ OK Par})$$

$$\frac{\vdash P : OK \qquad \forall n \in Name \ \ P \not\Downarrow_1 n}{\vdash !P : OK} \ (\pi_{esc} \text{ OK Repl})$$

$$\frac{\vdash P : OK \qquad \forall n \in Name \ \ P \not\Downarrow_1 n}{\vdash \overline{M}\langle M' \rangle.P : OK} \ (\pi_{esc} \text{ OK Output})$$

$$\frac{\vdash P : OK \qquad \forall n \in Name \ \ P \not\Downarrow_1 n}{\vdash M(x).P : OK} \ (\pi_{esc} \text{ OK Input})$$

$$\frac{\vdash S : OK}{\vdash [n : S] : OK} \ (\pi_{esc} \text{ OK Channel}) \qquad \frac{\vdash P : OK}{\vdash (\nu x : M)\ P : OK} \ (\pi_{esc} \text{ OK Var})$$

$$\frac{}{\vdash \varepsilon : OK} \ (\pi_{esc} \text{ OK Eps}) \qquad \frac{\vdash S : OK \qquad \vdash S' : OK}{\vdash S \mid S' : OK} \ (\pi_{esc} \text{ OK Abs})$$

$$\frac{\vdash P : OK \qquad \forall n \in Name \ \ P \not\Downarrow_1 n}{\vdash \langle M \rangle.P : OK} \ (\pi_{esc} \text{ OK Out Abs})$$

$$\frac{\vdash P : OK \qquad \forall n \in Name \ \ P \not\Downarrow_1 n}{\vdash (x).P : OK} \ (\pi_{esc} \text{ OK In Abs})$$

The following Lemma details the syntactic structure of a process presenting a channel n (after type-checking). This corresponds to the desired intuition: if $P \Downarrow_1 n$, a channel $[n : S]$ is present at the highest level, i.e. only under some restrictions.

Lemma 3.2 *If $P \Downarrow_1 n$ and $\vdash P : OK$, then $P \equiv (\nu n_1)\ \ldots (\nu n_k)\ (\nu x_1 : M_1)\ \ldots (\nu x_{k'} : M_{k'})\ ([n : S] \mid P')$ with $n \neq n_i$.*

Finally, we will say that a process P is *valid* and write $\vdash P : Valid$ if $\vdash P : OK$ and $P \not\Downarrow_2 n$ for all name $n \in Name$.

$$\frac{\vdash P : OK \qquad \forall n \in Name \;\; P \not\Downarrow_2 n}{\vdash P : Valid} \;(\pi_{esc}\text{ Valid})$$

From now on, we will focus mainly on valid processes only. The following lemma shows that this property is preserved by reduction.

Lemma 3.3 (Subject Reduction) *If $\sigma : P \longmapsto Q$ and $\vdash P : Valid$, then $\vdash Q : Valid$.*

3.4 Channel Closure

Now that we eliminated the excessive channels, we will have to add a few ! Consider the process $\overline{n}\langle m\rangle.P \mid n(x).Q$. It cannot reduce because no explicit channel is present for n. If we put an empty channel $[n : \varepsilon]$ in parallel, communication will take place. For this reason, we will define the *channel closure* of a process by adding explicit empty channels when needed. Since the same problem can appear under a scope restriction (for instance, $(\nu n)\ (\overline{n}\langle m\rangle.P \mid n(x).Q)$ cannot reduce), we will need to take care of this case too.

Definition 3.4 We first take scope restrictions into account. $cl(P)$ is a homomorphism for all constructs, except for:

$$cl((\nu n)\ P) \triangleq \begin{cases} (\nu n)\ ([n : \varepsilon] \mid cl(P)) & \text{if} \quad P \not\Downarrow_1 n \\ (\nu n)\ cl(P) & \text{if} \quad P \Downarrow_1 n \end{cases}$$

Then, the channel closure of a process w.r.t. a substitution σ consists in adding an empty channel for each free name in P or σ for which P does not present a channel. Formally,

$$cl_\sigma(P) \triangleq [n_1 : \varepsilon] \mid \ \ldots \ \mid [n_k : \varepsilon] \mid cl(P)$$

where $\{n_1, \ldots, n_k\} = (fn(P) \cup fn(\sigma)) \setminus pr(P)$ (cf. Lemma 3.1).

Note 1 *This is not completely well-defined, since if we take two different enumerations for $(fn(P) \cup fn(\sigma)) \setminus pr(P)$, the resulting processes will only be structurally congruent. This is why all our results involving $cl_\sigma(P)$ will be up to $\equiv$.*

We will say that P is channel-closed w.r.t. σ if $cl_\sigma(P) \equiv P$ (that is if P has all channels to guarantee communication). It is pure routine to check that this property is preserved by reduction in the π_{esc}-calculus.

4 Relations between the π and π_{esc}-Calculi

4.1 Back to the π-Calculus

In this Section, we prove a few equivalence properties between the π-calculus and the π_{esc}-calculus. The proofs rely mainly on our ability to translate a π_{esc}-process back into a π-process. This translation is written $[\![P]\!]$ (parameterized by a name n for the content of a channel) and is defined inductively by the following rules:

$$
\begin{array}{ll}
[\![(\nu n)\ P]\!] \triangleq (\nu n)\ [\![P]\!] & [\![[n : S]]\!] \triangleq [\![S]\!]_n \\
[\![\mathbf{0}]\!] \triangleq \mathbf{0} & [\![(\nu x : M)\ P]\!] \triangleq [\![P]\!]\{^M/_x\} \\
[\![P \mid Q]\!] \triangleq [\![P]\!] \mid [\![Q]\!] & [\![\varepsilon]\!]_n \triangleq \mathbf{0} \\
[\![!P]\!] \triangleq ![\![P]\!] & [\![S \mid S']\!]_n \triangleq [\![S]\!]_n \mid [\![S']\!]_n \\
[\![\overline{M}\langle M'\rangle.P]\!] \triangleq \overline{M}\langle M'\rangle.[\![P]\!] & [\![\langle M\rangle.P]\!]_n \triangleq \overline{n}\langle M\rangle.[\![P]\!] \\
[\![M(x).P]\!] \triangleq M(x).[\![P]\!] & [\![(x).P]\!]_n \triangleq n(x).[\![P]\!]
\end{array}
$$

In fact, $[\![P]\!]$ is a homomorphism for all constructs, except for channels and variable restrictions. In the former case, we just have to add the name of the channel back in front of abstractions and concretions. The latter case is more interesting: we perform the substitution required by the variable restriction, that is $[\![(\nu x : M)\ P]\!]$ is $[\![P]\!]$ in which we replace every free occurrence of x by M.

4.2 Results

When should we say that a π-process and a π_{esc}-process are "equivalent" ? Following our intuition, a π_{esc}-process P evolving in an environment σ should be translated into the π-process $[\![P]\!]\sigma^*$. Here we need to take the bindings of σ into account, because the free variables of P coming from previous communications should be replaced by their value. We apply the transitive closure σ^* in one step so that all free variables are converted into names of channels (in fact, it can be shown that $[\![P]\!]\sigma^*$ is equal to $[\![(\nu x_1 : M_1)\ \ldots (\nu x_k : M_k)\ P]\!]$ if $\sigma = \{^{M_k}/_{x_k}\} \uplus \ldots \uplus \{^{M_1}/_{x_1}\}$).

The following technical lemma can be proved quite easily. It shows that every reduction step in the π_{esc}-calculus corresponds to zero or one step in the π-calculus.

Lemma 4.1 *If $\sigma : P \longmapsto Q$, then $[\![P]\!]\sigma^*\ \mathcal{R}\ [\![Q]\!]\sigma^*$ where $\mathcal{R}$ is either $\equiv$ or $\longrightarrow$.*

The converse lemma is more complex. Additional hypotheses restrict the result to valid processes and appropriate environments only. It states that every reduction step in the π-calculus can be simulated by one or more reduction steps in the π_{esc}-calculus. Moreover, this simulation is not defined directly on P, but on its channel closure $cl_\sigma(P)$ (for instance, the π-processes in Section 3.4 reduce in the π-calculus, but only their channel closures reduce in the π_{esc}-calculus).

Lemma 4.2 *If $[\![P]\!]\sigma^* \longrightarrow Q$, $\vdash P : Valid$ and $fv(P) \subseteq dom(\sigma)$, then there is a process P' such that $\sigma : cl_\sigma(P) \longmapsto^+ P'$ and $[\![P']\!]\sigma^* \equiv Q$.*

This lemma is much more difficult to prove. We try to explain why and give a few hints.

- Channel closure does not mix well with an inductive proof. This comes from the fact that channel closure is not defined inductively on terms. Consequently, for almost every construct, we need a preliminary lemma that analyses this special case and relates the channel closure of the process with the channel closures of its sub-components. Sometimes, there is more than a single answer, depending on the context.
- Empty channels do not mix well with structural congruence rewriting. For instance, if the first step of reduction is

$$[\![[n:\varepsilon] \mid P]\!]\sigma^* \;=\; \mathbf{0} \mid [\![P]\!]\sigma^* \;\equiv\; [\![P]\!]\sigma^* \;\longrightarrow\; Q$$

  we cannot proceed directly by induction since the resulting process P does not present channel n anymore (structural congruence has "erased" it), hence the channel closures of $[n : \varepsilon] \mid P$ and P are different. This example is simple, but in the general case, empty channel erasing can occur anywhere in a term. So we need a result to relate the channel closure of P with P' when $[\![P]\!]\sigma^* \equiv P'$ is the first step of reduction.
- Channels do not mix well with parallel composition. This is the problem which needs the longest technical development. Suppose that

$$[\![P \mid P']\!]\sigma^* \longrightarrow Q \mid [\![P']\!]\sigma^*$$

  was derived from $[\![P]\!]\sigma^* \longrightarrow Q$ by (π Red Par). Suppose also that this reduction involves a communication on channel n, and that $P \not\Downarrow_1 n$ and $P' \Downarrow_1 n$ (that is, the explicit channel n is in the P' part). Therefore, by induction, we will get a simulation on $cl_\sigma(P) = [n : \varepsilon] \mid P_1$ since $P \not\Downarrow_1 n$. But now the corresponding reductions of $cl_\sigma(P \mid P')$ involving channel n should use the explicit channel in P' and not the empty channel $[n : \varepsilon]$ we added in the channel closure! In the general case, we need a result showing that reductions involving empty channels from closure can be replaced by reductions where communications are reported on (possibly non-empty) channels from a process in parallel.

These are technical lemmas, but in practice and in the rest of this paper, we will restrict ourselves to valid processes, without free variables and channel-closed w.r.t. $\varnothing$. In this case, the operational correspondence is much simpler:

Corollary 4.3

- *If* $\varnothing : P \longmapsto Q$, *then* $[\![P]\!] \; \mathcal{R} \; [\![Q]\!]$.
- *If* $[\![P]\!] \longrightarrow Q$, P *is channel-closed w.r.t.* $\varnothing$, $\vdash P : Valid$ *and* $fv(P) = \emptyset$, *then there is a process* P' *such that* $\varnothing : P \longmapsto^{+} P'$ *and* $[\![P']\!] \equiv Q$.

4.3 Observational Equivalence

To complete our results, we managed to prove an observational equivalence property. The observability predicate $P \downarrow M$ is defined on π-processes in the usual way (for example, $n(x).P \downarrow n$), and can be easily extended to π_{esc}-processes (for variables, substitution must be performed, i.e. $(\nu x : M)\ P \downarrow M$ when $P \downarrow x$).

For the π-calculus:

$$\frac{P \downarrow M \qquad n \neq M}{(\nu n)\ P \downarrow M}\ \text{(Obs Res)} \qquad \frac{P \downarrow M}{P \mid Q \downarrow M}\ \text{(Obs ParL)}$$

$$\frac{Q \downarrow M}{P \mid Q \downarrow M}\ \text{(Obs ParR)} \qquad \frac{P \downarrow M}{!P \downarrow M}\ \text{(Obs Repl)}$$

$$\frac{}{\overline{M}\langle M' \rangle.P \downarrow M}\ \text{(Obs Output)} \qquad \frac{}{M(x).P \downarrow M}\ \text{(Obs Input)}$$

For the π_{esc}-calculus, we must add:

$$\frac{S \not\equiv \varepsilon}{[n : S] \downarrow n}\ \text{(Obs Channel)}$$

$$\frac{P \downarrow M \qquad x \neq M}{(\nu x : M')\ P \downarrow M}\ \text{(Obs Var}_1\text{)} \qquad \frac{P \downarrow x}{(\nu x : M)\ P \downarrow M}\ \text{(Obs Var}_2\text{)}$$

Then one can show the following relation:

Proposition 4.4 *For every process* P *in the* π_{esc}*-calculus,* $P \downarrow M \Leftrightarrow [\![P]\!] \downarrow M$.

4.4 From the π-Calculus to the π_{esc}-Calculus

There is a simple way to transform a π-process into a "correct" π_{esc}-process: replace every construct $(\nu n)\ P$ with $(\nu n)\ ([n : \varepsilon] \mid P)$ and add an empty channel for every free name of P. In fact, this is exactly the definition of the channel-closure $cl_{\varnothing}(P)$ (if we view the π-process P as a π_{esc}-process). It has the following interesting properties: $cl_{\varnothing}(P)$ is valid, channel-closed w.r.t. $\varnothing$ and has no free variables if P has none (these properties allow us to use Corollary 4.3).

4.5 On the Choice of the π_{esc}-Calculus

Explicit channels and variables are similar in their structure, but we used different syntaxes: two constructs (νn) and $[n : S]$ for channels, and the single construct $(\nu x : M)$ for variables. One may ask why we retained this combination. Now is the time to answer this question.

We could have chosen to separate variables into a restriction (νx) and an explicit variable $[x : M]$, with rule (π_{esc} Red Subst Out) being $\sigma : [x : M] \mid \overline{x}\langle M' \rangle.P \longmapsto [x : M] \mid \overline{M}\langle M' \rangle.P$ (and similarly for (π_{esc} Red Subst In)). But in order to evaluate $[\![(\nu x)\ P]\!]$, we would have needed a way to reach the object $[x : M]$ in P and get the value M. This would have led to a very long technical development.

On the other hand, we could have chosen to include the content of a channel in the restriction operator with $(\nu n : S)$. In this case, we get a restriction interference. For instance, the process $(\nu n : \varepsilon)\ (\nu x : n)\ \overline{n}\langle x \rangle.P$ should reduce by putting the concretion $\langle x \rangle.P$ into n, but neither $(\nu n : \langle x \rangle.P)\ (\nu x : n)\ \mathbf{0}$ nor $(\nu x : n)\ (\nu n : \langle x \rangle.P)\ \mathbf{0}$ would be correct: in each case, a bound name becomes free ...

5 Encoding the π_{esc}-Calculus in Pure Ambients

5.1 The Encoding

The main mechanism underlying the encoding is a kind of communication based on the request/server model. In pure ambients, a request willing to communicate with n will be an ambient named rw with the process *request rw n* inside it (in our encoding, rw will be only *read* or *write*). Its first movement is to enter into n. Symmetrically, a server is a replicated ambient *enter* inside the destination n which tries to enter the request and take its control. This mechanism is similar to the encoding of *objective moves* of [4]. Let us first define some useful abbreviations:

$$server\ n\ .P \triangleq\ !\ enter[\ in\ n\ .\ \overline{open}\ enter\ .P]$$

$$request\ rw\ n\ \triangleq\ in\ n\ .\ \overline{in}\ rw\ .\ open\ enter$$

$$request\ rw\ x\ \triangleq\ in\ x\ .\ \overline{in}\ rw\ .\ open\ enter\ .\ out\ x$$

$$fwd\ M\ \triangleq\ server\ write\ .\ request\ write\ M$$

$$\mid\ server\ read\ .\ request\ read\ M$$

$$n\ be\ m\ .P \triangleq m[\ out\ n\ .\ \overline{in}\ m\ .(\ open\ n\ \mid P)] \mid\ \overline{out}\ n\ .\ in\ m\ .\ \overline{open}\ n$$

$$allowIO\ n\ \triangleq\ !\ \overline{in}\ n\ \mid\ !\ \overline{out}\ n$$

For example, here is the general reduction of a request and an ambient n containing a server:

$$\begin{array}{rl} & n[\ server\ rw\ .P \mid\ allowIO\ n\ ] \mid rw[\ request\ rw\ n\ \mid Q] \\ \hookrightarrow^{+} & n[\ server\ rw\ .P \mid\ allowIO\ n\ \mid rw[P \mid Q]] \end{array}$$

A variable x whose value is M will simply be an ambient named x with two servers inside it that replace every request with a similar request on M. Thus, a variable is simply a forwarder.

$$\begin{array}{rl} & x[\ fwd\ M\ \mid\ allowIO\ x\ ] \mid rw[\ request\ rw\ x\ \mid P] \\ \hookrightarrow^{+} & x[\ fwd\ M\ \mid\ allowIO\ x\ ] \mid rw[\ request\ rw\ M\ \mid P] \end{array}$$

for $rw = read$ or $write$.

A channel n is simulated by an ambient named n with a special server for $read$ requests (there is no server for $write$ requests). When n contains a $read$ request, it tries to find and take control of a $write$ request (with always the same request/server mechanism). When it is done, the $read$ request is replaced by an ambient x whose content is the forwarder of the $write$ request. Then, the two continuations are activated. Some intermediate ambient renamings are necessary to avoid interferences.

We will not detail the encoding further as it is not very instructive. Its full definition is presented below.

$$\{\!\{(\nu n)\ P\}\!\} \triangleq (\nu n)\ \{\!\{P\}\!\}$$

$$\{\!\{\mathbf{0}\}\!\} \triangleq \mathbf{0}$$

$$\{\!\{P \mid Q\}\!\} \triangleq \{\!\{P\}\!\} \mid \{\!\{Q\}\!\}$$

$$\{\!\{!P\}\!\} \triangleq !\{\!\{P\}\!\}$$

$$\begin{array}{l} \{\!\{\overline{M}\langle M'\rangle.P\}\!\} \triangleq (\nu p)\ \ (\ write\ [\ request\ write\ M \\ \qquad\qquad\qquad\qquad\qquad\qquad \mid fwd\ M' \\ \qquad\qquad\qquad\qquad\qquad\qquad \mid p[\ out\ read\ .\ \overline{open}\ p\ .\{\!\{P\}\!\}]] \\ \qquad\qquad\qquad\qquad \mid open\ p\ ) \end{array}$$

$$\begin{array}{l} \{\!\{M(x).P\}\!\} \triangleq (\nu p)\ \ (\ read\ [\ request\ read\ M \\ \qquad\qquad\qquad\qquad\qquad \mid open\ write\ .\ \overline{out}\ read\ .\ (\nu x)\ \ read\ be\ x\ . \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad (\ \overline{out}\ x\ .\ allowIO\ x \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \mid p[\ out\ x\ .\ \overline{open}\ p\ .\{\!\{P\}\!\}])] \\ \qquad\qquad\qquad\qquad \mid open\ p\ ) \end{array}$$

$$
\begin{aligned}
\{\!\{[n:S]\}\!\} \;\triangleq\;& (\nu p_1)\ \ldots(\nu p_k) \qquad\qquad \text{(where } p_i \text{ are the fresh names of } S\text{)}\\
&\quad (\ n\ [\ allowIO\ n\\
&\qquad \mid server\ read\ .\ (\nu p)\\
&\qquad\quad (\ \overline{out}\ read\ .\ read\ be\ p\ .\ \overline{in}\ p\ .\ out\ n\ .\ p\ be\ read\\
&\qquad\quad \mid enter[\ out\ read\ .\ in\ write\ .\ \overline{open}\ enter\ .\ in\ p\ .\ \overline{open}\ write\ ])\\
&\qquad \mid \{\!\{S\}\!\}_n]\\
&\quad \mid open\ p_1\ \mid \ldots \mid\ open\ p_k\ )\\
\{\!\{(\nu x:M)\ P\}\!\} \;\triangleq\;& (\nu x)\ \ (\ x[\ fwd\ M\ \mid\ allowIO\ x\ ]\\
&\qquad\quad \mid \{\!\{P\}\!\})\\
\{\!\{\varepsilon\}\!\}_n \;\triangleq\;& \mathbf{0}\\
\{\!\{S \mid S'\}\!\}_n \;\triangleq\;& \{\!\{S\}\!\}_n \mid \{\!\{S'\}\!\}_n\\
\{\!\{\langle M\rangle.P\}\!\}_n \;\triangleq\;& write\ [\ \overline{in}\ write\ .\ open\ enter\\
&\qquad \mid fwd\ M\\
&\qquad \mid p[\ out\ read\ .\ \overline{open}\ p\ .\{\!\{P\}\!\}]] \qquad \text{(where } p \text{ is fresh)}\\
\{\!\{(x).P\}\!\}_n \;\triangleq\;& (\nu q)\ \ (\ q\ [\ \overline{in}\ q\ .\ out\ n\ .\ q\ be\ read\\
&\qquad\quad \mid open\ write\ .\ \overline{out}\ read\ .\ (\nu x)\ \ read\ be\ x\ .\\
&\qquad\qquad (\ \overline{out}\ x\ .\ allowIO\ x\\
&\qquad\qquad \mid p[\ out\ x\ .\ \overline{open}\ p\ .\{\!\{P\}\!\}])] \qquad \text{(where } p \text{ is fresh)}\\
&\qquad \mid enter[\ in\ write\ .\ \overline{open}\ enter\ .\ in\ q\ .\ \overline{open}\ write\ ])
\end{aligned}
$$

To manage substitutions, we add the following definition:

$$
\begin{aligned}
\{\!\{\{^{M_1}/_{x_1}\} \uplus \ldots \uplus \{^{M_k}/_{x_k}\}, P\}\!\} \;\triangleq\;& \quad x_1[\ fwd\ M_1\ \mid\ allowIO\ x_1\ ]\\
&\mid \ \ldots\\
&\mid\ x_k[\ fwd\ M_k\ \mid\ allowIO\ x_k\ ]\\
&\mid \{\!\{P\}\!\}
\end{aligned}
$$

5.2 Results

Before we state some properties, we need to distinguish two kinds of reductions in safe ambients. *Principal reductions*, written $\overset{pr}{\hookrightarrow}$, correspond intuitively to the first reductions of the encodings into pure ambients of the axiomatic reduction rules from the π_{esc}-calculus. More precisely, we can pinpoint them by "marking" some specific capabilities in the encoding. These are the *in n* and *in x* capabilities in *request rw n* and *request rw x*, and the *in write* capability in the ambient *enter* in $\{\![n : S]\}\!\}$. Every reduction involving one of these marked capabilities will be principal. All the others are *auxiliary* and are written $\overset{aux}{\hookrightarrow}$.

Then, we can show that every reduction in the π_{esc}-calculus corresponds to one principal and many auxiliary reductions after encoding.

Proposition 5.1 *If $\sigma : P \longmapsto Q$, then $\{\!\{\sigma, P\}\!\} \overset{pr}{\hookrightarrow} \overset{aux*}{\hookrightarrow} \{\!\{\sigma, Q\}\!\}$.*

In the other direction, we can prove that if an encoding has a principal reduction, one can extend it with auxiliary reductions so that it corresponds to one single π_{esc}-reduction. Moreover, this single reduction is unique in some sense, up to structural congruence.

Proposition 5.2 *If $\{\!\{\sigma, P\}\!\} \overset{pr}{\hookrightarrow} Q$, then there is a process P' such that $\sigma : P \longmapsto P'$ and $Q \overset{aux*}{\hookrightarrow} \{\!\{\sigma, P'\}\!\}$. Moreover, if $\sigma : P \longmapsto P''$ and $Q \overset{aux*}{\hookrightarrow} \{\!\{\sigma, P''\}\!\}$, then $P' \equiv P''$.*

We need to explain why we had to distinguish between principal and auxiliary reductions. A counter-example, written in CCS style, is

$$P \triangleq\ !\, a \mid\ !\, \overline{a} \mid b.C \mid \overline{b}.D$$

We have $P \longrightarrow P$ and $P \longrightarrow P' =\ !\, a \mid\ !\, \overline{a} \mid C \mid D$. Considering the first reduction, the last theorem would give $\{\!\{P\}\!\} \hookrightarrow Q$, with $P \longrightarrow P$ and $Q \hookrightarrow^* \{\!\{P\}\!\}$. But we also have $P \longrightarrow P'$ and $Q \hookrightarrow^* \{\!\{P'\}\!\}$, with $P \not\equiv P'$. Thus the second assertion would be false. This would be impossible with two kinds of reductions: there must be a principal reduction between Q and $\{\!\{P'\}\!\}$.

However, Proposition 5.2 is not as strong as we would hope: we always need to reach the next encoding with auxiliary reductions before the next principal reduction. In fact, auxiliary reductions do not really matter: our encoding was designed so that a new effective step in the computation (i.e. a principal reduction) can take place as soon as possible (sometimes a few auxiliary reductions are needed before to unblock the situation). This is why we believe the following conjecture to be true. Proving it is not difficult in theory, but we face a very huge number of cases to examine, leading to a combinatorial explosion that only an automatic demonstration tool could maybe handle.

Conjecture 5.3 *$\overset{aux}{\hookrightarrow}$ is confluent with $\overset{aux}{\hookrightarrow}$ and $\overset{pr}{\hookrightarrow}$ (i.e. if $P \overset{aux}{\hookrightarrow} P_1$ and $P \overset{\alpha}{\hookrightarrow} P_2$, then there is a process P' such that $P_1 \overset{\alpha}{\hookrightarrow} P'$ and $P_2 \overset{aux}{\hookrightarrow} P'$ for $\alpha = pr$ or aux).*

6 The Final Encoding

It remains to compose the results of the two previous Sections. The encoding of a π-process P into pure ambients is simply defined by:

$$\langle\!\langle P \rangle\!\rangle \triangleq \{\!\{\varnothing, cl_{\varnothing}(P)\}\!\}$$

Using the definitions in Section 5, we can give the final encoding directly, and not via the π_{esc}-calculus (those definitions apply to processes without free names; otherwise we need to add an empty channel for each free name):

$$
\begin{aligned}
&\langle\!\langle \mathbf{0} \rangle\!\rangle \triangleq \mathbf{0}\\
&\langle\!\langle P \mid Q \rangle\!\rangle \triangleq \langle\!\langle P \rangle\!\rangle \mid \langle\!\langle Q \rangle\!\rangle\\
&\langle\!\langle !P \rangle\!\rangle \triangleq\ !\langle\!\langle P \rangle\!\rangle\\
&\langle\!\langle (\nu n)\ P \rangle\!\rangle \triangleq (\nu n)\\
&\qquad (\ n\ [\ allowIO\ n\\
&\qquad\qquad \mid server\ read\ .\ (\nu p)\\
&\qquad\qquad\qquad (\ \overline{out}\ read\ .\ read\ be\ p\ .\ \overline{in}\ p\ .\ out\ n\ .\ p\ be\ read\\
&\qquad\qquad\qquad \mid enter[\ out\ read\ .\ in\ write\ .\ \overline{open}\ enter\ .\ in\ p\ .\ \overline{open}\ write\ ])]\\
&\qquad \mid \langle\!\langle P \rangle\!\rangle)\\
&\langle\!\langle \overline{M}\langle M' \rangle.P \rangle\!\rangle \triangleq (\nu p)\ \ (\ write\ [\ request\ write\ M\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mid fwd\ M'\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mid p[\ out\ read\ .\ \overline{open}\ p\ .\langle\!\langle P \rangle\!\rangle]]\\
&\qquad\qquad\qquad\qquad \mid open\ p\ )\\
&\langle\!\langle M(x).P \rangle\!\rangle \triangleq (\nu p)\ \ (\ read\ [\ request\ read\ M\\
&\qquad\qquad\qquad\qquad\qquad \mid open\ write\ .\ \overline{out}\ read\ .\ (\nu x)\ \ read\ be\ x\ .\\
&\qquad\qquad\qquad\qquad\qquad\qquad (\ \overline{out}\ x\ .\ allowIO\ x\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mid p[\ out\ x\ .\ \overline{open}\ p\ .\langle\!\langle P \rangle\!\rangle])]\\
&\qquad\qquad\qquad \mid open\ p\ )
\end{aligned}
$$

It remains to state some operational correspondence properties. We will first define an equivalence relation $\approx$ between the π-calculus and pure ambients.

Definition 6.1 Let P be a π-process with no free variables and R a pure ambient process. We will say that P and R are equivalent (written $P \approx R$)

if there is a π_{esc}-process Q such that Q is valid, channel-closed w.r.t. $\varnothing$, with no free variables, $P \equiv [\![Q]\!]$ and $\{\!\!\{\varnothing, Q\}\!\!\} \equiv R$.

It is routine to check that $P \approx \langle\!\langle P \rangle\!\rangle$ for every π-process P with no free variables.

With this definition, we can state the final operational correspondence theorem, which validates our encoding. It is obtained by composing Corollary 4.3 and Propositions 5.1 and 5.2.

Theorem 6.2 *Suppose $P \approx R$.*

- *If $P \longrightarrow P'$, then there is a process R' such that $R \hookrightarrow^{+} R'$ and $P' \approx R'$.*
- *If $R \overset{pr}{\hookrightarrow} R'$, then there is a process R'' such that $R' \overset{aux*}{\hookrightarrow} R''$ and either $P \approx R''$, or $P \longrightarrow P' \approx R''$.*

7 Conclusion and Future Work

We gave an encoding of the synchronous π-calculus into the ambient calculus with neither communication primitives nor substitutions. We also proved an operational correspondence for our encoding. To do this, we designed the π_{esc}-calculus in order to facilitate the proof. However, this calculus seems interesting in itself, due to the equivalence results with the π-calculus.

The first future work should be to use an automatic demonstration tool to prove Conjecture 5.3. If it succeeds, we could state a much stronger final theorem for our operational correspondence (namely that only principal reductions do really matter). Moreover, our encoding was also designed to avoid all interferences with other processes (if we restrict internal names for the request/server mechanism). Thus, we would like to show that no attack against the protocol is possible by proving that P and $(\nu read)$ $(\nu write)$ $(\nu enter)$ $\langle\!\langle P \rangle\!\rangle$ are equivalent in every context.

We could also extend our encoding so that it applies to the *polyadic* π-calculus (i.e. in which communicable values can be tuples of arbitrary length). It does not seem difficult to us to switch from the monadic calculus to its polyadic version: we just have to create many ambients-variables after a communication (one for each variable-value in the tuple). The only difficulty would be to check that the number of values in the tuple and the number of variables are the same (this can be verified statically by a type system in π-calculus [11]). The protocol in ambients would be more complicate, but should not raise major theoretical problems. Indeed, in some specific cases, a few encodings of the polyadic π-calculus into its monadic version have already been proposed.

Furthermore, a few theoretical questions arise from our work. Is it possible to encode the π-calculus with classical mobile ambients instead of safe ambients (we already explained in the introduction why it seems difficult)? And more important to us: is it possible to encode the full ambient calculus (safe or not) with its communication primitives into the same calculus without communication primitives (in fact, this is the question which led us to do this work)? The main difference with the encoding of the π-calculus is that variables should now be present at every level in the hierarchy of ambients and not only at the global level. Thus, they should replicate themselves and scatter dynamically, even in newly created ambients!

Acknowledgement

I benefited from many discussions with Davide Sangiorgi for this work. Thanks also to Ilaria Castellani and Gérard Boudol, as well as the other members of the Mimosa and Tick teams.

References

[1] L. Cardelli. Abstractions for Mobile Computation. In *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, volume 1603 of *Lecture Notes in Computer Science*, pages 51–94. Springer Verlag, 1999.

[2] L. Cardelli. Wide Area Computation. In *Proceedings of ICALP'99*, volume 1644 of *Lecture Notes in Computer Science*, pages 10–24. Springer Verlag, April 1999.

[3] L. Cardelli and A. D. Gordon. A Calculus of mobile Ambients. 1997. Slides.

[4] L. Cardelli and A. D. Gordon. Mobile Ambients. In *Proceedings of FoSSaCS'98*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer Verlag, 1998.

[5] G. Ferrari, U. Montanari, and P. Quaglia. A π-Calculus with Explicit Substitutions. *Theoretical Computer Science*, 168 (1):53–103, November 1996.

[6] D. Hirschkoff. Handling Substitutions Explicitly in the π-Calculus. In *Proceedings of the Floc Workshop WESTAPP 99*, 1999.

[7] K. Honda and N. Yoshida. On Reduction-Based Process Semantics. *Theoretical Computer Science*, 152:437–486, 1995.

[8] F. Levi and D. Sangiorgi. Controlling Interference in Ambients. In *Proceedings of POPL'00*. ACM Press, 2000.

[9] M. Merro. On Equators in Asynchronous Name-Passing Calculi without Matching. In *Proceedings of EXPRESS'99*, volume 27 of *Electronic Notes in Theoretical Science*. Elsevier, 1999.

[10] M. Merro and D. Sangiorgi. On Asynchrony in Name-Passing Calculi. In *Proceedings of ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, pages 856–867. Springer Verlag, 1998.

[11] R. Milner. The Polyadic π-Calculus: a Tutorial. Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991.

[12] C. Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous π-Calculus. In *Proceedings of POPL'97*, pages 256–265. ACM, 1997.

[13] P. Zimmer. On the Expressiveness of Pure Mobile Ambients. Technical report, INRIA, 2000. forthcoming.