Math. Struct. in Comp. Science (2003), *vol.* 13, *pp.* 721–770.
DOI: 10.1017/S0960129503004079 Printed in the United Kingdom

On the expressiveness of pure safe ambients[†]

PASCAL ZIMMER

INRIA Sophia Antipolis – Mimosa, 2004 route des Lucioles, BP 93,
06902 Sophia Antipolis Cedex, France
Email: pascal.zimmer@sophia.inria.fr

Received 15 January 2001; revised 6 March 2002

We consider the *Pure Safe Ambient Calculus*, which is Levi and Sangiorgi's *Safe Ambient Calculus* (a variant of Cardelli and Gordon's *Mobile Ambient Calculus*) restricted to its mobility primitives - in particular, 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 a main result, we use these techniques to design an encoding of the synchronous π-calculus into pure ambients, and we study its correctness, thus showing that pure ambients are as expressive as the π-calculus. 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 an extension of the π-calculus in which communication and substitution are broken into simpler steps, and we show that is has the same expressive power as the π-calculus.

1. Introduction

The *ambient calculus* (Cardelli and Gordon 1997; Cardelli and Gordon 1998) was designed to model within a single framework both *mobile computing*, that is, computation in mobile devices such as a laptop, and *mobile computation*, that is, mobile code moving between different devices, like applets or agents. It also shows how the notions of administrative domains, firewalls, authorisations, and so on, can be formalised in a calculus. (For more discussion about the problems raised by mobility and computation over wide-area networks, see Cardelli (1999a;1999b).) 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 is 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 become visible at the current level, and communication between two processes can occur within an ambient (like in the π-calculus).

As a variant, the *safe ambients* were first presented in Levi and Sangiorgi (2000). 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. In contrast with this, in safe ambients both participants must agree

[†] An extended abstract of this paper appeared in Zimmer (2000).

by using matching action and coaction. In our investigations, it appeared that protocols were much simpler to implement in safe ambients than in classical 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 the same request twice. In safe ambients, the uniqueness of an answer is easier to achieve if there is only one coaction in each request.

The purpose of this paper is to study the expressive power of the subcalculus obtained by removing all communication primitives, to give the *pure safe 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 global substitution of variables in a process. Consequently, the only 'tools' allowed are the hierarchical structure of ambients, their movements and openings. The main motivation for this study is to understand what makes the ambient calculus so expressive, and which constructs are really important from a purely theoretical point of view. A similar question has been addressed in previous work in the setting of the π-calculus (Palamidessi 1997). 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.

We have not been able to show that pure ambients are as expressive as classical mobile ambients, but we have managed to encode the finite sum-free synchronous π-calculus (Milner 1991) in pure (safe) ambients. This result is also interesting: we know that the π-calculus is very expressive, and we show that pure ambients are at least as expressive. We give such an encoding in this paper and prove its correctness. The main problem we had to face was the simulation of substitution (which is precisely what is missing in pure ambients): 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.

It was shown in Cardelli and Gordon (1998) that mobile ambients without communication primitives are expressive enough to simulate Turing machines. However, Turing machines are a good model for sequential programming but are not well adapted to a concurrency framework. What we want is a 'reasonable' encoding having at least the property of compositionality (that is, such that $\langle\!\langle P \mid Q \rangle\!\rangle = \langle\!\langle P \rangle\!\rangle \mid \langle\!\langle Q \rangle\!\rangle$), which would not be the case if we use an encoding *via* Turing machines (CCS is also Turing-complete, but the π-calculus is much more powerful).

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). This is an extension of the π-calculus, with new primitives for variables and explicit channels, breaking up the communication and substitution mechanisms of the π-calculus into simpler steps. This calculus appears to be an interesting byproduct and not just a technical tool. This is because:

- it allows a better intuitive description of the mechanism underlying the encoding in pure ambients;
- and it has the same expressive power as the π-calculus. More precisely, we give translations from π to π_{esc} and *vice versa*, prove their correctness, and show a soundness result.

1.1. *Related work*

Some encodings of the π-calculus into ambients have already been proposed in the literature (Cardelli and Gordon 1998; Levi and Sangiorgi 2000), 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 (Levi and Sangiorgi 2000), no operational correspondence result has been completely proved for any of them.

For some restrictions of the π-calculus, substitution can be simulated in a different way from our approach. The *local* π ($L\pi$) (Merro and Sangiorgi 1998) is an asynchronous π-calculus (without matching) with an additional constraint on the input construct $n(x).P$: the name x may not occur free in P in input position. In this calculus, the following is a valid 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, so we could not use this approach in our case (that is, in the full synchronous π-calculus).

In the same way, an *equator* was first defined in Honda and Yoshida (1995) by

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

and it was shown in Merro (1999) that

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

($\cong_{\pi_a}$ 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, so we could not use this approach either.

Some variants of the π-calculus with explicit substitutions have also been proposed. In the $\pi\xi$-calculus (Ferrari *et al.* 1996), processes are prefixed by a global environment ξ that 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 between this approach and 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 Hirschkoff (1999), 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 point to note is that all dialects and variants of the π-calculus that 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 do not have any such operator in pure ambients.

1.2. *Outline*

In Section 2, we give the necessary background for the π-calculus and safe ambients. We also introduce a special kind of substitution. In Section 3, we present 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 are given in Appendix A.

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.

2.1.1. *Syntax* We distinguish between names of channels and names of variables. Let *Name* be a denumerably infinite set of names of channels (ranged over by n, m, p, ...), and *Var* be a denumerably infinite set of names of variables (ranged over by x, y, ...). We need to treat these two sets as distinct, because their behaviour will be different in the encoding into ambients. The syntax of the π-calculus is then defined as follows:

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

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

2.1.2. *Reduction rules* Below is the operational semantics of our π-calculus, given in the form of a one-step reduction relation, written $\longrightarrow$. The main rule is (π Red Comm), in which an input prefix and an output prefix on the 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). We write $\longrightarrow^*$ for zero or more reductions in the π-calculus, and $\longrightarrow^+$ for one or more reductions (the same conventions apply also to the other calculi that will be presented in the rest of the paper).

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

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

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

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

The last rule of 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, and so on. Its rules are given below.

$$\begin{array}{ll} P \equiv P & (\pi \text{ Struct Refl}) \\ P \equiv Q \ \Rightarrow\ Q \equiv P & (\pi \text{ Struct Symm}) \\ P \equiv Q \equiv R \ \Rightarrow\ P \equiv R & (\pi \text{ Struct Trans}) \\ P \equiv Q \ \Rightarrow\ (\nu n)\ P \equiv (\nu n)\ Q & (\pi \text{ Struct Res}) \\ P \equiv Q \ \Rightarrow\ P \mid R \equiv Q \mid R & (\pi \text{ Struct Par}) \\ P \equiv Q \ \Rightarrow\ !P \equiv\ !Q & (\pi \text{ Struct Repl}) \\ P \equiv Q \ \Rightarrow\ \overline{M}\langle M' \rangle.P \equiv \overline{M}\langle M' \rangle.Q & (\pi \text{ Struct Output}) \\ P \equiv Q \ \Rightarrow\ M(x).P \equiv M(x).Q & (\pi \text{ Struct Input}) \\ P \mid \mathbf{0} \equiv P & (\pi \text{ Struct Par Zero}) \end{array}$$

$$P \mid Q \equiv Q \mid P \qquad (\pi \text{ Struct Par Comm})$$
$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad (\pi \text{ Struct Par Assoc})$$
$$(\nu n)\ (P \mid Q) \equiv P \mid (\nu n)\ Q \quad \text{if} \quad n \notin fn(P) \qquad (\pi \text{ Struct Res Par})$$
$$(\nu n)\ (\nu m)\ P \equiv (\nu m)\ (\nu n)\ P \qquad (\pi \text{ Struct Res Res})$$
$$!P \equiv P \mid !P \qquad (\pi \text{ Struct Repl Par})$$
$$!\mathbf{0} \equiv \mathbf{0} \qquad (\pi \text{ Struct Repl Zero})$$

2.2. *Pure ambients*

In this section, we present the subcalculus of the Safe Ambient Calculus we will use. It corresponds to the original Safe Ambients from Levi and Sangiorgi (2000) 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:

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

The basic constructs of process calculi are present: restriction of names, nil process, parallel operator 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, a capability and its corresponding cocapability (that is, with the same name) must be present at the right place, as shown by the following reduction rules:

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

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):

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

$$\frac{P \ \hookrightarrow\ Q}{n[P] \ \hookrightarrow\ n[Q]}\ \text{(SA Red Amb)} \qquad \frac{P \equiv P' \quad P' \ \hookrightarrow\ Q' \quad Q' \equiv Q}{P \ \hookrightarrow\ Q}\ \text{(SA Red Struct)}$$

The main difference between this and the safe ambients of Levi and Sangiorgi (2000) is the lack of communication primitives, namely the asynchronous output $\langle M \rangle$ and the input binder $(x).P$. Another difference is the use of replication in place of recursion. Furthermore, cocapabilities are not present in the ambient calculus of Cardelli and Gordon (1998).

2.3. *Substitutions*

In this section, we introduce a special kind of substitution, which has a tree structure. This is needed because both π_{esc}-calculus and the encoding into ambients implicitly use such a mathematical structure and not a substitution of general shape.

Intuitively, to every variable, it associates either another variable in the domain of the substitution, or a channel name. And there is an additional condition: by following the 'chain' of successive images, we always end on a channel name.

More formally, in the rest of the paper, every occurrence of 'substitution' refers to the following definition.

Definition 1. A substitution is a partial function $\sigma : Var \to Var \cup Name$ such that:

- $\forall x \in dom(\sigma), x\sigma \in Name \cup dom(\sigma)$ (that is, $im(\sigma) \subseteq Name \cup dom(\sigma)$).
- $\forall x \in dom(\sigma)$, there is $k > 0$ such that $x\sigma^k \in Name$ (that is, there are no cycles) (σ^k being the composition of σ, k times).

We now 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 ($\sigma^* = \sigma^p$ where $p = \max\{k / x\sigma^k \in Name, x \in dom(\sigma)\}$).

If $x \notin dom(\sigma)$ and $M \in Name \cup dom(\sigma)$, we define $\sigma' = \{^M/_x\} \uplus \sigma$ by $x\sigma' = M$ and $y\sigma' = y\sigma$ for $y \neq x$. The resulting substitution σ' is still a substitution in the sense of Definition 1.

The empty substitution is written $\emptyset$, and we also define $fn(\sigma) \triangleq im(\sigma) \cap Name$. Moreover, we extend 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 additional constructs to handle substitutions and channels.

First, the construction $(\nu x : M)\ P$ (with $x \neq M$) represents a new variable x whose contents is M. The name x is bound in P (as n is bound in $(\nu n)\ P$). Intuitively, any 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.

Prefixes in the π-calculus have the form $M(x).P$ and $\overline{M}\langle M'\rangle.P$; we call the bodies of the prefixes, namely $(x).P$ and $\langle M'\rangle.P$, *abstraction* and *concretion*, respectively.

The construction $[n : S]$ represents an explicit channel of name n, whose contents are 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 use parallel composition for convenience in proofs). 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 to the processes $\overline{n}\langle M \rangle.P$ and $n(x).P$, respectively). Intuitively, when a process performs an output or input on n, the request is put inside the channel with that name (if there is one).

The complete syntax of π_{esc} is as follows:

$$
\begin{array}{rcll}
P & ::= & (\nu n)\ P & \text{restriction} \\
 & \mid & \mathbf{0} & \text{nil process} \\
 & \mid & P \mid Q & \text{parallel composition} \\
 & \mid & !P & \text{replication} \\
 & \mid & \overline{M}\langle M' \rangle.P & \text{output} \\
 & \mid & M(x).P & \text{input} \\
 & \mid & [n : S] & \text{explicit channel} \\
 & \mid & (\nu x : M)\ P & \text{explicit variable with } x \neq M
\end{array}
\qquad
\begin{array}{rcll}
M & ::= & n & \text{channel name} \\
 & \mid & x & \text{variable name} \\
 & & & \\
S & ::= & \varepsilon & \text{empty channel} \\
 & \mid & S \mid S' & \text{parallel composition} \\
 & \mid & \langle M \rangle.P & \text{concretion} \\
 & \mid & (x).P & \text{abstraction}
\end{array}
$$

3.2. *Reduction rules*

We now give an operational semantics for π_{esc}. Reduction rules are of the form $\sigma : P \longmapsto P'$, where P and P' are processes, and σ is a substitution that acts as an environment containing the values of free variables in P. As a side condition, we restrict the application of the rules to processes P and substitutions σ such that $fv(P) \subseteq dom(\sigma)$, in order 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 can then apply the same rule again (since in this case $M \in dom(\sigma)$ by the 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 have already been 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 $[n : S]$ refer implicitly 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 occur effectively. The two continuations P and Q are then placed outside the channel, except that a new variable x with contents 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 $fv((\nu x : M)\ P) \subseteq dom(\sigma)$, which is an instance of the implicit side condition).

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

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 means of a structural congruence rewriting.

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

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

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

3.3. *Structural congruence*

The congruence $\equiv$ 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). The complete list is given below:

$P \equiv P$ (same for S)	(π_{esc} Struct Refl)
$P \equiv Q \Rightarrow Q \equiv P$ (same for S)	(π_{esc} Struct Symm)
$P \equiv Q \equiv R \Rightarrow P \equiv R$ (same for S)	(π_{esc} Struct Trans)
$P \equiv Q \Rightarrow (\nu n)\ P \equiv (\nu n)\ Q$	(π_{esc} Struct Res)

$P \equiv Q \;\Rightarrow\; P \mid R \equiv Q \mid R$ (π_{esc} Struct Par)
$P \equiv Q \;\Rightarrow\; !P \equiv !Q$ (π_{esc} Struct Repl)
$P \equiv Q \;\Rightarrow\; \overline{M}\langle M'\rangle.P \equiv \overline{M}\langle M'\rangle.Q$ (π_{esc} Struct Output)
$P \equiv Q \;\Rightarrow\; M(x).P \equiv M(x).Q$ (π_{esc} Struct Input)
$S \equiv S' \;\Rightarrow\; [n : S] \equiv [n : S']$ (π_{esc} Struct Channel)
$P \equiv Q \;\Rightarrow\; (\nu x : M)\; P \equiv (\nu x : M)\; Q$ (π_{esc} Struct Var)
$S' \equiv S'' \;\Rightarrow\; S \mid S' \equiv S \mid S''$ (π_{esc} Struct Abs)
$P \equiv Q \;\Rightarrow\; \langle M\rangle.P \equiv \langle M\rangle.Q$ (π_{esc} Struct Out Abs)
$P \equiv Q \;\Rightarrow\; (x).P \equiv (x).Q$ (π_{esc} Struct In Abs)
$P \mid \mathbf{0} \equiv P$ (π_{esc} Struct Par Zero)
$P \mid Q \equiv Q \mid P$ (π_{esc} Struct Par Comm)
$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$ (π_{esc} Struct Par Assoc)
$S \mid \varepsilon \equiv S$ (π_{esc} Struct Abs Zero)
$S \mid S' \equiv S' \mid S$ (π_{esc} Struct Abs Comm)
$S \mid (S' \mid S'') \equiv (S \mid S') \mid S''$ (π_{esc} Struct Abs Assoc)
$(\nu n)\; (P \mid Q) \equiv P \mid (\nu n)\; Q \quad \text{if} \quad n \notin fn(P)$ (π_{esc} Struct Res Par)
$(\nu x : M)\; (P \mid Q) \equiv P \mid (\nu x : M)\; Q \quad \text{if} \quad x \notin fv(P)$ (π_{esc} Struct Var Par)
$(\nu n)\; (\nu m)\; P \equiv (\nu m)\; (\nu n)\; P$ (π_{esc} Struct Res Res)
$(\nu n)\; (\nu x : M)\; P \equiv (\nu x : M)\; (\nu n)\; P \quad \text{if} \quad n \neq M$ (π_{esc} Struct Res Var)
$(\nu x : M)\; (\nu y : M')\; P \equiv (\nu y : M')\; (\nu x : M)\; P$
$\quad \text{if} \quad x \neq y, x \neq M' \text{ and } \; y \neq M$ (π_{esc} Struct Var Var)
$!P \equiv P \mid !P$ (π_{esc} Struct Repl Par)
$!\mathbf{0} \equiv \mathbf{0}$ (π_{esc} Struct Repl Zero)

3.4. *Channel presentation*

To follow our intuition (that is, the modelling of explicit channels), we need to cut down the set of allowed processes in the π_{esc}-calculus to ensure that channels are correctly positioned 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 channels, for instance resulting in

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

and communication would never occur between P and Q.

For this reason, we need to be able to detect a channel. We define a *presentation predicate* $P \Downarrow_1 n$, which means, intuitively, that at least a channel $[n : S]$ is present in P and is not hidden by scope restriction. The formal definition of this predicate is easy: the only axiom is $[n : S] \Downarrow_1 n$ and all other rules just perform inductive calls (except for $(\nu m)\; P \Downarrow_1 n$, which checks $m \neq n$).

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

Here is the formal definition of $P \Downarrow_i n$ where $i = 1, 2$:

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

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

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

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

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

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

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

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

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

Moreover, we write $pr(P)$ for the set of channels presented by P.

Definition 2. $pr(P) \triangleq \{n \in Name / P \Downarrow_1 n\}$.

Proposition 3. $pr(P) \subseteq fn(P)$.

See Section A.2 for a proof.

3.5. *Validity*

Now that we have a way to detect the presence or absence of one or many channels, we can define exactly the set of valid processes by ensuring that all channels are correctly positioned and unique. This can be achieved by means of a small type system.

For this purpose, 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.

$$\frac{\vdash P : OK \qquad P \not\Downarrow_2 n}{\vdash (\nu n)\ P : OK}\ (\pi_{esc}\ \text{OK Res}) \qquad \frac{}{\vdash \mathbf{0} : OK}\ (\pi_{esc}\ \text{OK Zero})$$

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

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

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

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

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

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

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

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

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, that is, only under some name restrictions.

Lemma 4. 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 $n \neq n_i$.

Proof. The proof is by induction on the structure of P. □

Corollary 5. If $P \Downarrow_1 n_i$ and $\vdash P : OK$, then $P \equiv (\nu m_1) \ldots (\nu m_k)\, (\nu x_1 : M_1) \ldots (\nu x_{k'} : M_{k'})\, ([n_1 : S_1] \mid \ldots \mid [n_p : S_p] \mid P')$ with $n_i \neq m_j$.

Proof. We give the inductive step by considering a process P such that $P \Downarrow_1 n_1$ and $P \Downarrow_1 n_2$. By Lemma 4, $P \equiv (\nu m_1) \ldots (\nu m_k)\, (\nu x_1 : M_1) \ldots (\nu x_{k'} : M_{k'})\, ([n_1 : S_1] \mid P_1)$ with $m_1 \neq n_i$. From $\vdash P : OK$, we get $\vdash S_1 : OK$ and $\vdash P_1 : OK$. Consequently, we cannot have $S_1 \Downarrow_1 n_2$. Necessarily, $P_1 \Downarrow_1 n_2$ since $P \Downarrow_1 n_2$. Then apply Lemma 4 to P_1 and use scope extrusion to get the final result. □

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

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

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

Proposition 6. (Subject Reduction). If $\sigma : P \longmapsto Q$ and $\vdash P : Valid$, then $\vdash Q : Valid$.

See Section A.2 for a proof.

3.6. *Closure*

Now that we have eliminated some extra 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 takes place. We thus 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 need to take care of this case too.

Definition 7. We first take scope restrictions into account. $cl(P)$ is a homomorphism from π_{esc}-processes to π_{esc}-processes for all constructs, except for restriction:

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

Then the channel closure of a process with regard to a substitution σ consists of adding an empty channel for each free name in P or σ for which P does not present a channel. Formally:

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

where $\{n_1, \dots, n_k\} = (fn(P) \cup fn(\sigma)) \setminus pr(P)$ (by Proposition 3, we know that $pr(P) \subseteq fn(P)$).

We say that P is channel-closed with regard to σ if $cl_\sigma(P) \equiv P$ (that is if P has all channels to guarantee communication).

Note that if we take two different enumerations for $(fn(P) \cup fn(\sigma)) \setminus pr(P)$ in Definition 7, the resulting processes may not be syntactically equal; they will only be structurally congruent. This is why all our results involving $cl_\sigma(P)$ will be up to $\equiv$.

4. Relations between the π and π_{esc}-calculi

In this section, we prove a few equivalence properties between the π-calculus and the π_{esc}-calculus. The proofs mainly rely on our ability to translate a π_{esc}-process back into a π-process.

4.1. *Back to the π-calculus*

The translation from π_{esc} to π is written $[\![P]\!]$ (with a parameter name n for the content of a channel) and is defined inductively by the following rules:

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

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. In the latter case, we perform the substitution required by the variable restriction: that is, $[\![(\nu x : M)\ P]\!]$ is $[\![P]\!]$ in which every free occurrence of x is replaced by M.

4.2. *Operational correspondence*

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, $[\![P]\!]\sigma^*$ 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 proposition shows that every reduction step in the π_{esc}-calculus corresponds to zero or one step in the π-calculus.

Proposition 8. If $\sigma : P \longmapsto Q$, then $[\![P]\!]\sigma^* \overset{\equiv}{\longrightarrow} [\![Q]\!]\sigma^*$, where $\overset{\equiv}{\longrightarrow}$ is either $\equiv$ or $\longrightarrow$.

Proof. The proof is by induction on the derivation of $\sigma : P \longmapsto Q$ – see Section A.6 for details. □

The converse proposition is more complex. Additional hypotheses restrict the result to valid processes and appropriate environments only. The result 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 3.6 reduce in the π-calculus, but only their channel closures reduce in the π_{esc}-calculus).

Proposition 9. If $[\![P]\!]\sigma^* \longrightarrow Q$, and $\vdash P : Valid$, and $fv(P) \subseteq dom(\sigma)$, there is a process P' such that $\sigma : cl_\sigma(P) \longmapsto^+ P'$ and $[\![P']\!]\sigma^* \equiv Q$.

Proof. The proof is by induction on the derivation of $[\![P]\!]\sigma^* \longrightarrow Q$ – see Section A.7 for details. □

This proposition 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 to the channel closures of its sub-components. Sometimes, there is more than a single answer, depending on the context. See Section A.3.
- Empty channels do not mix well with structural congruence rewriting. For instance, if the first step of reduction is

$$[\![[n : \varepsilon] \mid P]\!]\sigma^* = 0 \mid [\![P]\!]\sigma^* \equiv [\![P]\!]\sigma^* \longrightarrow Q,$$

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 channel erasing can occur anywhere in a term. So we need a result to relate the channel closure of P to P' when $[\![P]\!]\sigma^* \equiv P'$ is the first step of reduction. See Section A.4.

— Channels do not mix well with parallel composition. This is the problem that 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 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 $[n : \varepsilon]$ that 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. See Section A.5.

These are technical propositions, but in the rest of this paper we restrict ourselves to valid processes, without free variables and channel-closed with regard to $\varnothing$. Since we use these processes extensively throughout the paper, we will call them *complete*.

Definition 10. A π_{esc}-process P is called *complete* if P is channel-closed with regard to $\varnothing$, $\vdash P : Valid$ and $fv(P) = \emptyset$.

In this case, the operational correspondence is much simpler.

Corollary 11.

— If $\varnothing : P \longmapsto Q$, then $[\![P]\!] \xrightarrow{\equiv} [\![Q]\!]$.

— If $[\![P]\!] \longrightarrow Q$ for a complete process P, then there is a process P' such that $\varnothing : P \longmapsto^+ P'$ and $[\![P']\!] \equiv Q$.

Proof. The result follows from Propositions 8 and 9. □

4.3. *Observational equivalence*

To complete our results, we have managed to prove an observational equivalence property. This result is not really useful for the encoding in pure ambients, but a soundness result might also be interesting for future work.

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, that is, $(\nu x : M)\ P \downarrow M$ 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 add:

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

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

Proposition 12. For a process P in the π_{esc}-calculus, $P \downarrow M \;\Leftrightarrow\; [\![P]\!] \downarrow M$.

See Section A.8 for a proof.

Corollary 13. Let P be a complete π_{esc}-process. Then, we have $[\![P]\!] \;\longrightarrow^* \downarrow M$ if and only if $\emptyset : P \;\longmapsto^* \downarrow M$.

Proof. The proof is by induction on the length of the reductions, with the help of Corollary 11 and Proposition 12. □

4.4. *Soundness*

We conclude this first set of results with a soundness theorem between the π and π_{esc}-calculi.

First of all, we need to choose a suitable equivalence between processes for both of these calculi. For convenience, we will use *barbed bisimulation.* Here is the definition for the π-calculus.

Definition 14. A relation $\mathcal{R}$ is a *barbed bisimulation* for the π-calculus if, whenever $P \;\mathcal{R}\; Q$ for two π-processes P and Q, we have:

— If $P \;\longrightarrow\; P'$, there is a process Q' such that $Q \;\longrightarrow^*\; Q'$ and $P' \;\mathcal{R}\; Q'$.
— If $Q \;\longrightarrow\; Q'$, there is a process P' such that $P \;\longrightarrow^*\; P'$ and $P' \;\mathcal{R}\; Q'$.
— $P \;\longrightarrow^* \downarrow M$ if and only if $Q \;\longrightarrow^* \downarrow M$.

Let

$$\approx_\pi = \bigcup \{\mathcal{R} / \mathcal{R} \text{ is a barbed bisimulation}\}.$$

One can check that $\approx_\pi$ is the largest barbed bisimulation and that it contains structural congruence $\equiv$ (these are classical results).

We now give a very similar definition for complete processes in the π_{esc}-calculus.

Definition 15. A relation $\mathcal{R}$ is a *barbed bisimulation* for the π_{esc}-calculus if, whenever $P \;\mathcal{R}\; Q$ for two complete π_{esc}-processes P and Q, we have:

— If $\emptyset : P \;\longmapsto\; P'$, there is a process Q' such that $\emptyset : Q \;\longmapsto^*\; Q'$ and $P' \;\mathcal{R}\; Q'$.
— If $\emptyset : Q \;\longmapsto\; Q'$, there is a process P' such that $\emptyset : P \;\longmapsto^*\; P'$ and $P' \;\mathcal{R}\; Q'$.
— $\emptyset : P \;\longmapsto^* \downarrow M$ if and only if $\emptyset : Q \;\longmapsto^* \downarrow M$.

Let

$$\approx_{esc} = \bigcup \{\mathcal{R} / \mathcal{R} \text{ is a barbed bisimulation}\}.$$

One can check that $\approx_{esc}$ is the largest barbed bisimulation.

Finally, we can state the soundness result: the encodings of two equivalent processes are equivalent, and *vice versa*.

Theorem 16 (Soundness). Let P and Q be two complete π_{esc}-processes. Then, $P \approx_{esc} Q$ if and only if $[\![P]\!] \approx_\pi [\![Q]\!]$.

Proof. We prove the two implications separately.

$\mathbf{P} \approx_{\mathbf{esc}} \mathbf{Q} \;\Rightarrow\; [\![\mathbf{P}]\!] \approx_\pi [\![\mathbf{Q}]\!]$ We define the relation $\mathscr{R}$ by $P \mathrel{\mathscr{R}} Q$ when $P \equiv [\![P_0]\!]$, $Q \equiv [\![Q_0]\!]$ and $P_0 \approx_{esc} Q_0$. We need to show that $\mathscr{R}$ is a barbed bisimulation (then $\mathscr{R} \subseteq \approx_\pi$). To do this, suppose that $P \mathrel{\mathscr{R}} Q$ (and consequently that such P_0 and Q_0 exist).

— Suppose that $P \longrightarrow P'$. Then, $[\![P_0]\!] \longrightarrow P'$. By Corollary 11, there is a process P'' such that $\varnothing : P_0 \longmapsto^+ P''$ and $[\![P'']\!] \equiv P'$. Since $P_0 \approx_{esc} Q_0$, there is a process Q' such that $\varnothing : Q_0 \longmapsto^* Q'$ and $P'' \approx_{esc} Q'$. Then, by Corollary 11, we have $Q \equiv [\![Q_0]\!] \longrightarrow^* [\![Q']\!]$. And, $P' \mathrel{\mathscr{R}} [\![Q']\!]$ using the definition of $\mathscr{R}$.

— The reasoning is similar when Q reduces.

— Using Lemma 63, Corollary 13 and Definition 15, we have the following equivalences: $P \longrightarrow^* \downarrow M$ if and only if $[\![P_0]\!] \longrightarrow^* \downarrow M$ if and only if $\varnothing : P_0 \longmapsto^* \downarrow M$ if and only if $\varnothing : Q_0 \longmapsto^* \downarrow M$ if and only if $[\![Q_0]\!] \longrightarrow^* \downarrow M$ if and only if $Q \longrightarrow^* \downarrow M$.

$[\![\mathbf{P}]\!] \approx_\pi [\![\mathbf{Q}]\!] \;\Rightarrow\; \mathbf{P} \approx_{\mathbf{esc}} \mathbf{Q}$ We define the relation $\mathscr{R}$ by $P \mathrel{\mathscr{R}} Q$ when $[\![P]\!] \approx_\pi [\![Q]\!]$. We need to show that $\mathscr{R}$ is a barbed bisimulation (then $\mathscr{R} \subseteq \approx_{esc}$). To do this, suppose that $P \mathrel{\mathscr{R}} Q$.

— Suppose that $\varnothing : P \longmapsto P'$. By Corollary 11, we have $[\![P]\!] \xrightarrow{\equiv} [\![P']\!]$. There are two cases:

– If $[\![P']\!] \equiv [\![P]\!]$, then $[\![P']\!] \approx_\pi [\![P]\!]$ and $[\![P']\!] \approx_\pi [\![Q]\!]$ by transitivity of $\approx_\pi$. Finally, $P' \mathrel{\mathscr{R}} Q$.

– If $[\![P]\!] \longrightarrow [\![P']\!]$, since $\approx_\pi$ is a bisimulation, there is a process Q' such that $[\![Q]\!] \longrightarrow^* Q'$ and $[\![P']\!] \approx_\pi Q'$. By Corollary 11, there is a process Q'' such that $\varnothing : Q \longmapsto^* Q''$ and $[\![Q'']\!] \equiv Q'$. Then, $Q' \approx_\pi [\![Q'']\!]$ and $[\![P']\!] \approx_\pi [\![Q'']\!]$ by transitivity of $\approx_\pi$. Finally, $P' \mathrel{\mathscr{R}} Q''$.

— The reasoning is similar when Q reduces.

— Using Corollary 13 and Definition 14, we have the following equivalences: $\varnothing : P \longmapsto^* \downarrow M$ if and only if $[\![P]\!] \longrightarrow^* \downarrow M$ if and only if $[\![Q]\!] \longrightarrow^* \downarrow M$ if and only if $\varnothing : Q \longmapsto^* \downarrow M$. □

4.5. *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 with regard to $\varnothing$ and has no free variables if P has none (these properties allow us to use Corollary 11).

Proposition 17. $cl_\varnothing(P)$ is channel-closed with regard to $\varnothing$ and $\vdash cl_\varnothing(P) : Valid$. Moreover, if $fv(P) = \emptyset$, $fv(cl_\varnothing(P)) = \emptyset$. Consequently, $cl_\varnothing(P)$ is complete if P has no free variables.

4.6. *On the choice of the π_{esc}-calculus*

Explicit channels and variables are similar in their structure, but we have 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 of π_{esc} in pure ambients is a kind of communication based on the request/server model. In pure ambients, a request willing to communicate with n is 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 n. Symmetrically, a server is a replicated ambient *enter* inside the destination n that tries to enter the request and take its control. The underlying protocol is that, after the ambient-request has entered the ambient-server, the request accepts the server code and lets it run inside it. This mechanism is similar to the encoding of *objective moves* of Cardelli and Gordon (1998). Let us first define some useful abbreviations:

$$
\begin{aligned}
&server\ n\ .P \triangleq !\ enter[\ in\ n\ .\ \overline{open}\ enter\ .P]\\
&request\ rw\ n\ \triangleq\ in\ n\ .\ \overline{in}\ rw\ .\ open\ enter\\
&request\ rw\ x\ \triangleq\ in\ x\ .\ \overline{in}\ rw\ .\ open\ enter\ .\ out\ x\\
&fwd\ M\ \triangleq\ server\ write\ .\ request\ write\ M\ \mid\ server\ read\ .\ request\ read\ M\\
&n\ be\ m\ .P \triangleq m[\ out\ n\ .\ \overline{in}\ m\ .(\ open\ n\ \mid P)] \mid\ \overline{out}\ n\ .\ in\ m\ .\ \overline{open}\ n\\
&allowIO\ n\ \triangleq !\ \overline{in}\ n\ \mid\ !\ \overline{out}\ n\ .
\end{aligned}
$$

For example, the general interaction between a request and a server is:

$$
\begin{aligned}
&n[\ server\ rw\ .P\ \mid\ allowIO\ n\ ] \mid rw[\ request\ rw\ n\ \mid Q]\\
\hookrightarrow^+\ \ &n[\ server\ rw\ .P\ \mid\ allowIO\ n\ \mid rw[P \mid Q]]\quad \text{for } rw = read \text{ or } write.
\end{aligned}
$$

A variable x whose value is M is simply 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:

$$\begin{array}{ll} & x[\ \mathit{fwd}\ M \ \mid\ \mathit{allowIO}\ x\ ] \mid \mathit{rw}[\ \mathit{request}\ \mathit{rw}\ x \ \mid P] \\ \hookrightarrow^{+} & x[\ \mathit{fwd}\ M \ \mid\ \mathit{allowIO}\ x\ ] \mid \mathit{rw}[\ \mathit{request}\ \mathit{rw}\ M \ \mid P] \quad \text{for } \mathit{rw} = \mathit{read} \text{ or } \mathit{write} \end{array}$$

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 (always using the same request/server mechanism). When this 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 do not give further details of the encoding as it is not very instructive. We believe the only way to understand it fully is to test it by hand and try to mimic the reductions of the π_{esc}-calculus. Some of these are given in Section A.9 in the proof of Proposition 18.

The full definition of the encoding is presented below:

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

$$\begin{array}{l}\{\!\{(x).P\}\!\}_n \triangleq (\nu q)\ \ (\ q\ \ [\ \overline{in}\ q\ .\ out\ n\ .\ q\ be\ read \\ \qquad\qquad\qquad\qquad\quad \mid open\ write\ .\ \overline{out}\ read\ .\ (\nu x)\ \ read\ be\ x\ . \\ \qquad\qquad\qquad\qquad\qquad\qquad (\ \overline{out}\ x\ .\ allowIO\ x \\ \qquad\qquad\qquad\qquad\qquad\qquad \mid p[\ out\ x\ .\ \overline{open}\ p\ .\{\!\{P\}\!\}])]\ \text{(where } p \text{ is fresh)} \\ \qquad\qquad\qquad \mid enter[\ in\ write\ .\ \overline{open}\ enter\ .\ in\ q\ .\ \overline{open}\ write\ ]).\end{array}$$

To handle substitutions, we add the following definition:

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

5.2. *Results*

Before we state some properties, we need to distinguish two kinds of reductions in safe ambients. *Principal reductions*, written $\stackrel{pr}{\hookrightarrow}$, 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 is principal. All the others are *auxiliary* and are written $\stackrel{aux}{\hookrightarrow}$.

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

Proposition 18. If $\sigma : P \longmapsto Q$, then $\{\!\{\sigma, P\}\!\} \stackrel{pr}{\hookrightarrow} \stackrel{aux^*}{\hookrightarrow} \{\!\{\sigma, Q\}\!\}$.

Proof. The proof is by induction on the derivation of $\sigma : P \longmapsto Q$. Basically, we have to check that every reduction rule in the π_{esc}-calculus is mimicked by several reductions in the encoding, which is routine. We just give one example here, refer to Section A.9 for the other cases.

(π_{esc} Red Subst Out) Suppose that $\sigma : \overline{x}\langle M'\rangle.P \longmapsto \overline{M}\langle M'\rangle.P$ with $x\sigma = M$. To simplify, we consider only $\sigma = \{{}^{M}/_{x}\}$. It is not difficult to derive the general case with an arbitrary σ. We have (assuming that p does not interfere with other names):

$$\begin{array}{l}\{\!\{\sigma, \overline{x}\langle M'\rangle.P\}\!\} \\ \quad = \left\{\begin{array}{l} \ \ x[\ fwd\ M\ \mid\ allowIO\ x\ ] \\ \mid\ (\nu p)\ \ (\ write\ \ [\ request\ write\ x \\ \qquad\qquad\qquad\quad \mid fwd\ M' \\ \qquad\qquad\qquad\quad \mid p[\ out\ read\ .\ \overline{open}\ p\ .\{\!\{P\}\!\}]] \\ \qquad\qquad \mid open\ p\ )\end{array}\right.\end{array}$$

$$
\equiv \left\{ \begin{array}{l}
\quad x[\ enter[\ in\ write\ .\ \overline{open}\ enter\ .\ request\ write\ M\ ] \\
\qquad \mid fwd\ M \ \mid\ allowIO\ x\ ] \\
\mid\ (\nu p)\ \ (\ write\ \ [\ in\ x\ .\ \overline{in}\ write\ .\ open\ enter\ .\ out\ x \\
\qquad\qquad\qquad \mid fwd\ M' \\
\qquad\qquad\qquad \mid p[\ out\ read\ .\ \overline{open}\ p\ .\{\!\!\{P\}\!\!\}]] \\
\qquad\quad \mid open\ p\ )
\end{array} \right.
$$

$$
\overset{pr}{\hookrightarrow} \left\{ \begin{array}{l}
(\nu p)\ \ (\ x\ \ [\ enter[\ in\ write\ .\ \overline{open}\ enter\ .\ request\ write\ M\ ] \\
\qquad\qquad \mid fwd\ M\ \mid\ allowIO\ x \\
\qquad\qquad \mid write\ \ [\ \overline{in}\ write\ .\ open\ enter\ .\ out\ x \\
\qquad\qquad\qquad\quad \mid fwd\ M' \\
\qquad\qquad\qquad\quad \mid p[\ out\ read\ .\ \overline{open}\ p\ .\{\!\!\{P\}\!\!\}]]] \\
\qquad \mid open\ p\ )
\end{array} \right.
$$

$$
\overset{aux}{\hookrightarrow} \left\{ \begin{array}{l}
(\nu p)\ \ (\ x\ \ [\ \ fwd\ M\ \mid\ allowIO\ x \\
\qquad\qquad \mid write\ \ [\ enter[\ \overline{open}\ enter\ .\ request\ write\ M\ ] \\
\qquad\qquad\qquad\quad \mid open\ enter\ .\ out\ x \\
\qquad\qquad\qquad\quad \mid fwd\ M' \\
\qquad\qquad\qquad\quad \mid p[\ out\ read\ .\ \overline{open}\ p\ .\{\!\!\{P\}\!\!\}]]] \\
\qquad \mid open\ p\ )
\end{array} \right.
$$

$$
\overset{aux}{\hookrightarrow} \left\{ \begin{array}{l}
(\nu p)\ \ (\ x\ \ [\ \ fwd\ M\ \mid\ allowIO\ x \\
\qquad\qquad \mid write\ \ [\ request\ write\ M \\
\qquad\qquad\qquad\quad \mid out\ x \\
\qquad\qquad\qquad\quad \mid fwd\ M' \\
\qquad\qquad\qquad\quad \mid p[\ out\ read\ .\ \overline{open}\ p\ .\{\!\!\{P\}\!\!\}]]] \\
\qquad \mid open\ p\ )
\end{array} \right.
$$

$$
\overset{aux}{\hookrightarrow} \left\{ \begin{array}{l}
\quad x[\ fwd\ M\ \mid\ allowIO\ x\ ] \\
\mid (\nu p)\ \ (\ write\ \ [\ request\ write\ M \\
\qquad\qquad\qquad \mid fwd\ M' \\
\qquad\qquad\qquad \mid p[\ out\ read\ .\ \overline{open}\ p\ .\{\!\!\{P\}\!\!\}]] \\
\qquad\quad \mid open\ p\ )
\end{array} \right.
$$

$$
= \ \{\!\!\{\sigma, \overline{M}\langle M'\rangle.P\}\!\!\} \qquad \square
$$

In the other direction, we can prove that if an encoded ambient term 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 19. If $\{\!\!\{\sigma, P\}\!\!\} \overset{pr}{\hookrightarrow} Q$, 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 $P' \equiv P''$.

Proof. Since $\{\!\!\{\sigma, P\}\!\!\} \overset{pr}{\hookrightarrow} Q$ is a principal reduction, it must involve one of the 'marked' capabilities. This capability determines a corresponding axiomatic reduction in the π_{esc}-calculus, namely:

— (π_{esc} Red Subst Out) if the capability is $in\ x$ in $request\ write\ x$.

— (π_{esc} Red Subst In) if the capability is *in x* in *request read x*.
— (π_{esc} Red Output) if the capability is *in n* in *request write n*.
— (π_{esc} Red Input) if the capability is *in n* in *request read n*.
— (π_{esc} Red Comm) if the capability is *in write* in the ambient *enter* in $\{\![n : S]\}\!$.

Starting from this axiom, it is not difficult to determine the reduction $\sigma : P \longmapsto P'$ in π_{esc}. Then, with the same arguments as in the proof of Proposition 18, we can find the auxiliary reductions in π_{esc} such that $Q \overset{aux^*}{\hookrightarrow} \{\!\{\sigma, P'\}\!\}$.

Moreover, since the first principal reduction uniquely determines the corresponding axiom (and reduction) in π_{esc}, the resulting process P' is unique modulo $\equiv$, which implies the second part of the proposition. □

We need to explain why we have 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. The problem is avoided by distinguishing the two kinds of reductions: there must be a principal reduction between Q and $\{\!\{P'\}\!\}$.

However, Proposition 19 is not as strong as we would hope: we always reach the next encoded term 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 (that is, a principal reduction) can take place as soon as possible (sometimes a few auxiliary reductions are needed first 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 huge number of cases to examine, leading to a combinatorial explosion that possibly only an automatic demonstration tool could handle.

Conjecture 20. $\overset{aux}{\hookrightarrow}$ is confluent with $\overset{aux}{\hookrightarrow}$ and $\overset{pr}{\hookrightarrow}$, that is:
— if $P \overset{aux}{\hookrightarrow} P_1$ and $P \overset{aux}{\hookrightarrow} P_2$, there is a process Q such that $P_1 \overset{aux}{\hookrightarrow} Q$ and $P_2 \overset{aux}{\hookrightarrow} Q$.
— if $P \overset{pr}{\hookrightarrow} P_1$ and $P \overset{aux}{\hookrightarrow} P_2$, there is a process Q such that $P_1 \overset{aux}{\hookrightarrow} Q$ and $P_2 \overset{pr}{\hookrightarrow} Q$.

5.3. *Observational equivalence*

As we have done for the π_{esc}-calculus, we are able to define an observation predicate, and show an adequacy result: observability is preserved by the encoding.

However, observability in pure ambients will be more difficult to define and make well-adapted to the encoding - to this end, we define evaluation contexts.

Definition 21. An *evaluation context* is a context where the hole $[\cdot]$ occurs only once and not under a guard. More precisely, they are of the form: $C[\cdot] \triangleq (\nu\vec{n})\,([\cdot] \mid P)$.

Definition 22. The *observation predicate* is defined in pure ambients by: $P \downarrow M$ if and only if either $P \equiv C[\{\!\{\overline{M}\langle M'\rangle.Q\}\!\}]$ or $P \equiv C[\{\!\{M(x).Q\}\!\}]$ or $P \equiv C[\{\!\{[M : S]\}\!\}]$ for some evaluation context $C[\cdot]$ that does not bind M, some process Q, some names M and M', and some term $S \not\equiv \varepsilon$ (depending on the case, not all of these conditions are needed).

With this definition, we can state the adequacy property.

Proposition 23. For a π_{esc}-process P, we have:

1 $P \downarrow M \;\Rightarrow\; \{\!\{P\}\!\} \hookrightarrow^* \downarrow M$

2 $\{\!\{P\}\!\} \downarrow M \;\Rightarrow\; P \downarrow M.$

Proof.

1 The proof is by induction on the derivation of $P \downarrow M$. The only non-trivial case is for (Obs Var_2): when $(\nu x : M)\ P \downarrow M$ has been derived from $P \downarrow x$. Then, by the induction hypothesis, we get $\{\!\{P\}\!\} \hookrightarrow^* Q \downarrow x$. We have:

$$\begin{array}{rcl} \{\!\{(\nu x : M)\ P\}\!\} & = & (\nu x)\ (x[\ fwd\ M \ \mid\ allowIO\ x\ ] \mid \{\!\{P\}\!\}) \\ & \hookrightarrow^* & (\nu x)\ (x[\ fwd\ M \ \mid\ allowIO\ x\ ] \mid Q). \end{array}$$

Since $Q \downarrow x$, necessarily $Q \equiv C[\{\!\{\overline{x}\langle M'\rangle.Q'\}\!\}]$ or $Q \equiv C[\{\!\{x(y).Q'\}\!\}]$ for some evaluation context $C[\cdot]$ that does not bind x nor M (since the oberved name x is a variable, we cannot be in the third case, that is, of a non-empty channel). Then (supposing we are in the first case),

$$\begin{array}{rcl} \{\!\{(\nu x : M)\ P\}\!\} & \hookrightarrow^* & (\nu x)\ (x[\ fwd\ M \ \mid\ allowIO\ x\ ] \mid C[\{\!\{\overline{x}\langle M'\rangle.Q'\}\!\}]) \\ & \equiv & C[(\nu x)\ (x[\ fwd\ M \ \mid\ allowIO\ x\ ] \mid \{\!\{\overline{x}\langle M'\rangle.Q'\}\!\})] \\ & \hookrightarrow^+ & R = C[(\nu x)\ (x[\ fwd\ M \ \mid\ allowIO\ x\ ] \mid \{\!\{\overline{M}\langle M'\rangle.Q'\}\!\})] \end{array}$$

and we finally get $R \downarrow M$.

2 This part follows by a direct and easy induction on P. □

Soundness – discussion Is it possible to define some equivalence in pure ambients (barbed bisimulation or some other) based upon the observation predicate $P \downarrow M$, and show a soundness property as we did for the encoding between the π and π_{esc}-calculi? We are not sure if such a result would be technically reachable, but in any case, we are not really interested in it. In fact, the resulting equivalence over pure ambients seems too artificial to us, since this particular observation predicate is really too specific to our encoding and very different from sensible definitions of observability in ambients (for example, observing the name n for every ambient $n[P]$). Thus, the resulting equivalence would be very poor from our point of view. However, it would be interesting to find another sensible notion of equivalence, and show a corresponding soundness result for our encoding. This we leave as an open question.

6. The final encoding

It remains for us to compose the results of the two previous sections. The encoding of a π-process P into pure ambients is 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 (this implies that, strictly speaking, the encoding is only fully compositional for processes without free names):

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

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

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

Definition 24. Let P be a π-process with no free variables and R be a pure ambient process. We say that P and R are *equivalent* (written $P \bowtie R$) if there is a complete π_{esc}-process Q such that $P \equiv [\![Q]\!]$ and $\{\!\{\varnothing, Q\}\!\} \equiv R$.

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

With this definition, we can state the final operational correspondence theorem.

Theorem 25. Suppose $P \bowtie R$.

— If $P \longrightarrow P'$, there is a process R' such that $R \hookrightarrow^{+} R'$ and $P' \bowtie R'$.

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

— $P \downarrow M \Rightarrow R \hookrightarrow^{*} \downarrow M$ and $R \downarrow M \Rightarrow P \downarrow M$.

Proof. Combine Corollary 11 and Propositions 18, 19, 12 and 23. $\square$

7. Conclusion and future work

We have given an encoding of the synchronous π-calculus into the ambient calculus with neither communication primitives nor substitutions. And we have proved an operational correspondence for our encoding, showing that pure ambients are as expressive as the π-calculus. To do this, we designed the π_{esc}-calculus in order to facilitate the proof. This calculus seems interesting in itself, since it models substitutions and channels explicitly. Independently from the encoding in pure ambients, we proved expressiveness and adequacy results between π and π_{esc}.

The first task for future work should be to use an automatic demonstration tool to prove Conjecture 20. 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 the processes P and $(\nu read)\ (\nu write)\ (\nu enter)\ \langle\!\langle P \rangle\!\rangle$ are 'equivalent' in some sense.

Furthermore, a few expressiveness questions arise from our work. Is it possible to encode the π-calculus with classical mobile ambients instead of safe ambients (we explained in the introduction why it seems difficult)? And more importantly for 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 that 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, intuitively, they should replicate themselves and scatter dynamically, even in newly created ambients, and it is not obvious how to achieve this effect.

Appendix A. Proofs

A.1. *Lemmas concerning substitutions*

These first elementary lemmas deal with our special notion of substitution. They detail how commutativity, transitive closure, structural congruence and reduction relate to each other, and will be useful in many other proofs.

Lemma 26. If $x \notin dom(\sigma)$:

1 $\{^M/_x\}\sigma = \sigma\{^{M\sigma}/_x\}$
2 $\{^M/_x\}\sigma^* = \sigma^*\{^{M\sigma^*}/_x\}$.

Proof.

1 $x\{^M/_x\}\sigma = M\sigma = x\{^{M\sigma}/_x\} = x\sigma\{^{M\sigma}/_x\}$ since $x \notin dom(\sigma)$. For $y \neq x$, $y\{^M/_x\}\sigma = y\sigma = y\sigma\{^{M\sigma}/_x\}$ since $y\sigma \neq x$ (because $x \in im(\sigma)$ implies $x \in dom(\sigma)$).
2 With $\sigma^* = \sigma^p$, we just need to apply the last result p times. □

Lemma 27. If $x \notin dom(\sigma)$ and $M \in Name \cup dom(\sigma)$, then $(\{^M/_x\} \uplus \sigma)^* = \{^M/_x\}\sigma^*$.

Proof. We cannot have $M = x$, and since $x \notin dom(\sigma)$, $x \notin im(\sigma)$. Thus, $x \notin im(\{^M/_x\}\uplus\sigma)$. Consequently, for $M' \neq x$, $M'(\{^M/_x\} \uplus \sigma)^* = M'\sigma^* = M'\{^M/_x\}\sigma^*$. And $x(\{^M/_x\} \uplus \sigma)^* = M(\{^M/_x\} \uplus \sigma)^* = M\sigma^* = x\{^M/_x\}\sigma^*$. Finally, $(\{^M/_x\} \uplus \sigma)^* = \{^M/_x\}\sigma^*$. □

Lemma 28.

1 If $P \equiv Q$, then $P\sigma \equiv Q\sigma$.
2 If $P \longrightarrow Q$, then $P\sigma \longrightarrow Q\sigma$.

Proof. The proof is by induction on the derivations of $P \equiv Q$ and $P \longrightarrow Q$. □

Lemma 29. If $P\sigma \equiv Q$, then there exists P' such that $P \equiv P'$ and $Q = P'\sigma$.

Proof. The proof is by induction on the derivation of $P\sigma \equiv Q$. □

A.2. *Elementary lemmas for the π_{esc}-calculus*

This section contains some elementary lemmas for the π_{esc}-calculus that will be useful in the final proofs.

A.2.1. *Free names and free variables*

This lemma shows how free names and variables are preserved by structural congruence and reduction.

Lemma 30.

1 If $P \equiv Q$, then $fn(P) = fn(Q)$ and $fv(P) = fv(Q)$.
2 If $\sigma : P \longmapsto Q$, then $fn(P) \cup fn(\sigma) = fn(Q) \cup fn(\sigma)$ and $fv(Q) \subseteq dom(\sigma)$.

Proof. The proof is by induction on the derivations of $P \equiv Q$ and $\sigma : P \longmapsto Q$. □

A.2.2. *Channel presentation and channel closure*

The two following results state some basic facts.

Lemma 31. $P \Downarrow_2 n \;\Rightarrow\; P \Downarrow_1 n \;\Rightarrow\; n \in fn(P)$.

Proof. The proof is by induction on the derivations of $P \Downarrow_2 n$ and $P \Downarrow_1 n$. □

Proposition 3 $pr(P) \subseteq fn(P)$.

Proof. The result follows from Lemma 31. □

Channel presentation and channel closure are preserved by structural congruence.

Lemma 32. If $P \equiv Q$:

1 $P \Downarrow_1 n \;\Leftrightarrow\; Q \Downarrow_1 n$ for all $n \in Name$
2 $P \Downarrow_2 n \;\Leftrightarrow\; Q \Downarrow_2 n$ for all $n \in Name$
3 $cl(P) \equiv cl(Q)$

Proof. The proof is by induction on the derivation of $P \equiv Q$. □

Corollary 33. If $P \equiv Q$, then $pr(P) = pr(Q)$ and $cl_\sigma(P) \equiv cl_\sigma(Q)$.

Proof. The result follows from Lemmas 30 and 32. □

Also, channel presentation and channel closure are preserved by reduction in the π_{esc}-calculus.

Lemma 34. If $\sigma : P \longmapsto Q$:

1 $P \Downarrow_1 n \;\Leftrightarrow\; Q \Downarrow_1 n$ for all $n \in Name$.

2 $P \Downarrow_2 n \iff Q \Downarrow_2 n$ for all $n \in Name$.
3 $\sigma : cl(P) \longmapsto cl(Q)$.

Proof. The proof is by induction on the derivation of $\sigma : P \longmapsto Q$, using Lemma 32. □

Corollary 35. If $\sigma : P \longmapsto Q$:

1 $pr(P) = pr(Q)$.
2 $\sigma : cl_\sigma(P) \longmapsto cl_\sigma(Q)$.
3 P channel-closed with regard to $\sigma \iff Q$ channel-closed with regard to σ.

Proof. The results follow from Lemmas 30 and 34. □

The next lemma gives a few more results about channel closure.

Lemma 36.

1 $fv(cl(P)) = fv(P)$.
2 $P \Downarrow_2 n$ if and only if $cl(P) \Downarrow_2 n$.
3 $\vdash P : Valid$ if and only if $\vdash cl(P) : Valid$.

Proof. The proof is by induction on P. □

A.2.3. *Validity*

This section gives the proof of subject reduction for the validity type system in π_{esc}-calculus.

Lemma 37. If $P \equiv Q$, then $\vdash P : Valid \iff \vdash Q : Valid$.

Proof. First prove $\vdash P : OK \iff \vdash Q : OK$ by induction on the derivation of $P \equiv Q$, using Lemma 31 and Lemma 32. Then, it is easy to prove $\vdash P : Valid \iff \vdash Q : Valid$ with Lemma 32. □

Proposition 6. If $\sigma : P \longmapsto Q$ and $\vdash P : Valid$, then $\vdash Q : Valid$.

Proof. First prove $\vdash P : OK \Rightarrow \vdash Q : OK$ by induction on the derivation of $\sigma : P \longmapsto Q$, using Lemma 34 and Lemma 37. Then, it is easy to prove $\vdash P : Valid \Rightarrow \vdash Q : Valid$ with Lemma 34. □

A.2.4. *Reductions*

The next lemma details one elementary step of substitution.

Lemma 38. If $M\sigma = M'$, then $\sigma : \overline{M}\langle M''\rangle.P \longmapsto^* \overline{M'}\langle M''\rangle.P$ (provided that $fv(\overline{M}\langle M''\rangle.P) \subseteq dom(\sigma)$) and $\sigma : M(x).P \longmapsto^* M'(x).P$ (provided that $fv(M(x).P) \subseteq dom(\sigma)$). The same holds if $M\sigma^* = M'$.

Proof. If $M \notin dom(\sigma)$, then $\overline{M}\langle M''\rangle.P = \overline{M'}\langle M''\rangle.P$ and $M(x).P = M'(x).P$ with no reduction. Otherwise, $M = x$ for some variable name x, and $x\sigma = M'$. Then, $\sigma : \overline{M}\langle M''\rangle.P \longmapsto \overline{M'}\langle M''\rangle.P$ by (π_{esc} Red Subst Out) and $\sigma : M(x).P \longmapsto M'(x).P$ by

(π_{esc} Red Subst In). If $M\sigma^* = M'$, we just apply the last result p times where p is such that $\sigma^* = \sigma^p$ (there are at most p reductions). $\square$

The next lemma shows how we can extend environments without affecting reductions.

Lemma 39. If $x \notin dom(\sigma)$, $M \in Name \cup dom(\sigma)$ and $\sigma : P \longmapsto P'$, then $\{^M/_x\} \uplus \sigma : P \longmapsto P'$.

Proof. The proof is by induction on the derivation of $\sigma : P \longmapsto P'$. $\square$

A.2.5. *Translation* $[\![.]\!]$

The following three lemmas show how the translation $[\![.]\!]$ relates to free names and variables, channel closure and structural congruence.

Lemma 40. $fn([\![P]\!]) \subseteq fn(P)$ and $fv([\![P]\!]) \subseteq fv(P)$.

Proof. The proof is by induction on the structure of P (the inequalities come from the cases $[n : \varepsilon]$ and $(\nu x : M)\ P$ with $x \notin fv([\![P]\!])$). $\square$

Lemma 41. $[\![cl(P)]\!] \equiv [\![P]\!]$

Proof. The proof is by induction on the structure of P. $\square$

Lemma 42. If $P \equiv Q$, then $[\![P]\!] \equiv [\![Q]\!]$.

Proof. The proof is by induction on the derivation of $P \equiv Q$:

(π_{esc} Struct Var) Suppose that $(\nu x : M)\ P \equiv (\nu x : M)\ Q$ was derived from $P \equiv Q$. By the induction hypothesis, $[\![P]\!] \equiv [\![Q]\!]$. Then $[\![P]\!]\{^M/_x\} \equiv [\![Q]\!]\{^M/_x\}$ by Lemma 28. Finally, $[\![(\nu x : M)\ P]\!] \equiv [\![(\nu x : M)\ Q]\!]$.

(π_{esc} Struct Res Par) Suppose that $(\nu n)\ (P \mid Q) \equiv P \mid (\nu n)\ Q$ with $n \notin fn(P)$. By Lemma 40, $n \notin fn([\![P]\!])$. Then we can derive $[\![(\nu n)\ (P \mid Q)]\!] = (\nu n)\ ([\![P]\!] \mid [\![Q]\!]) \equiv [\![P]\!] \mid (\nu n)\ [\![Q]\!] = [\![P \mid (\nu n)\ Q]\!]$ with (π Struct Res Par).

(π_{esc} Struct Var Par) Suppose that $(\nu x : M)\ (P \mid Q) \equiv P \mid (\nu x : M)\ Q$ with $x \notin fv(P)$. By Lemma 40, $x \notin fv([\![P]\!])$ and thus $[\![P]\!]\{^M/_x\} = [\![P]\!]$. Then we have $[\![(\nu x : M)\ (P \mid Q)]\!] = [\![P]\!]\{^M/_x\} \mid [\![Q]\!]\{^M/_x\} = [\![P]\!] \mid [\![(\nu x : M)\ Q]\!] = [\![P \mid (\nu x : M)\ Q]\!]$.

(π_{esc} Struct Res Var) Suppose that $(\nu n)\ (\nu x : M)\ P \equiv (\nu x : M)\ (\nu n)\ P$ with $n \neq M$. Then $[\![(\nu n)\ (\nu x : M)\ P]\!] = (\nu n)\ ([\![P]\!]\{^M/_x\}) = ((\nu n)\ [\![P]\!])\{^M/_x\} = [\![(\nu x : M)\ (\nu n)\ P]\!]$ (the second equality is correct because $n \neq M$).

(π_{esc} Struct Var Var) Suppose $(\nu x : M)\ (\nu y : M')\ P \equiv (\nu y : M')\ (\nu x : M)\ P$ with $x \neq y, x \neq M'$ and $y \neq M$. With these conditions and Lemma 26, $\{^M/_x\}\{^{M'}/_y\} = \{^{M'}/_y\}\{^{M\{^{M'}/_y\}}/_x\} = \{^{M'}/_y\}\{^M/_x\}$. Then $[\![(\nu x : M)\ (\nu y : M')\ P]\!] = [\![(\nu y : M')\ (\nu x : M)\ P]\!]$.

The other cases are trivial. $\square$

The next two results are decomposition lemmas when the translation of a process is either a parallel composition, or an input/output operator.

Lemma 43. If $[\![P]\!] = P_1 \mid P_2$ and P is not of the form $(\nu x : M)\ P'$, then either

— $P = [n : S \mid S']$ with $[\![S]\!]_n = P_1$ and $[\![S']\!]_n = P_2$
— $P = P_1' \mid P_2'$ with $[\![P_1']\!] = P_1$ and $[\![P_2']\!] = P_2$

Proof. These are the only ways to derive $[\![P]\!] = P_1 \mid P_2$. □

Lemma 44. If $[\![P]\!] = \overline{M}\langle M'\rangle.P'$ and P is not of the form $(\nu x : M'')\ P''$, then either

— $P = [n : \langle M'\rangle.P'']$ with $M = n$ and $[\![P'']\!] = P'$
— $P = \overline{M}\langle M'\rangle.P''$ with $[\![P'']\!] = P'$

A similar lemma holds if $[\![P]\!] = M(x).P'$.

Proof. These are the only ways to derive $[\![P]\!] = \overline{M}\langle M'\rangle.P'$. □

A.3. *Lemmas for channel closure*

In this section we treat our first problem: channel closure does not mix well with an inductive proof. For this reason, we need a preliminary result for almost every construct to show how channel closure behaves with them. We start here with the two easier cases: variable restriction and name restriction. The other cases need further technical development and will be treated in the following sections. Furthermore, Lemma 46 gives a similar result in a special case.

Lemma 45. $(\nu x : M)\ cl_{\{^M/_x\}\uplus\sigma}(P) \equiv cl_\sigma((\nu x : M)\ P)$ if $x \notin dom(\sigma)$ and $M \in Name \cup dom(\sigma)$.

Proof. We have $pr(P) = pr((\nu x : M)\ P)$. Then:

$$\begin{aligned}(fn(P) \cup fn(\{^M/_x\} \uplus \sigma)) \setminus pr(P) &= (fn(P) \cup fn(M) \cup fn(\sigma)) \setminus pr(P)\\ &= (fn((\nu x : M)\ P) \cup fn(\sigma)) \setminus pr((\nu x : M)\ P).\end{aligned}$$

Finally,

$$\begin{aligned}(\nu x : M)\ cl_{\{^M/_x\}\uplus\sigma}(P) &\equiv (\nu x : M)\ ([n_1 : \varepsilon] \mid \dots \mid [n_k : \varepsilon] \mid cl(P))\\ &\equiv [n_1 : \varepsilon] \mid \dots \mid [n_k : \varepsilon] \mid (\nu x : M)\ cl(P)\\ &\equiv cl_\sigma((\nu x : M)\ P),\end{aligned}$$

using (π_{esc} Struct Var Par). □

Lemma 46. $cl_\sigma(P) \equiv cl_\sigma([n_1 : \varepsilon] \mid \dots \mid [n_k : \varepsilon] \mid P)$ if $n_i \in fn(P) \setminus pr(P)$.

Proof. Since $n_i \in fn(P) \setminus pr(P)$, we can write the set $(fn(P) \cup fn(\sigma)) \setminus pr(P)$ as $\{n_1, \dots, n_k, m_1, \dots m_p\}$. We can show $pr([n_1 : \varepsilon] \mid \dots \mid [n_k : \varepsilon] \mid P) = \{n_1, \dots, n_k\} \cup pr(P)$. Then

$$\begin{aligned}(fn([n_1 : \varepsilon] \mid \dots \mid [n_k : \varepsilon] \mid P) \cup fn(\sigma)) \setminus pr([n_1 : \varepsilon] \mid \dots \mid [n_k : \varepsilon] \mid P)&\\ = (\{n_1, \dots, n_k\} \cup fn(P) \cup fn(\sigma)) \setminus (\{n_1, \dots, n_k\} \cup pr(P))&\\ = ((fn(P) \cup fn(\sigma)) \setminus pr(P)) \setminus \{n_1, \dots, n_k\}&\\ = \{m_1, \dots m_p\}.&\end{aligned}$$

Finally,

$$\begin{aligned} cl_\sigma(P) &\equiv [m_1 : \varepsilon] \mid \ldots \mid [m_p : \varepsilon] \mid [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl(P) \\ &\equiv cl_\sigma([n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid P). \end{aligned}$$

□

Lemma 47. If $n \notin fn(\sigma)$, $cl_\sigma((\nu n)\ P) \equiv \begin{cases} (\nu n)\ cl_\sigma(P) & \text{if } n \in fn(P) \\ (\nu n)\ ([n : \varepsilon] \mid cl_\sigma(P)) & \text{if } n \notin fn(P) \end{cases}$

Proof.

— If $P \Downarrow_1 n$, then $n \in fn(P)$. In this case, $pr(P) = \{n\} \cup pr((\nu n)\ P)$ and $cl((\nu n)\ P) = (\nu n)\ cl(P)$. Then

$$\begin{aligned} (fn((\nu n)\ P) \cup fn(\sigma)) \setminus pr((\nu n)\ P) &= (fn(P) \setminus \{n\} \cup fn(\sigma)) \setminus pr((\nu n)\ P) \\ &= ((fn(P) \cup fn(\sigma)) \setminus \{n\}) \setminus pr((\nu n)\ P) \end{aligned}$$

since $n \notin fn(\sigma)$. This is equal to $(fn(P) \cup fn(\sigma)) \setminus pr(P)$. Finally,

$$\begin{aligned} cl_\sigma((\nu n)\ P) &\equiv [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid (\nu n)\ cl(P) \\ &\equiv (\nu n)\ ([n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl(P)) \\ &\equiv (\nu n)\ cl_\sigma(P), \end{aligned}$$

using (π_{esc} Struct Res Par) (we can suppose $n_i \neq n$).

— If $P \not\Downarrow_1 n$, we have $pr(P) = pr((\nu n)\ P)$ and $cl((\nu n)\ P) = (\nu n)\ ([n : \varepsilon] \mid cl(P))$. There are two cases to consider:

– If $n \notin fn(P)$, $fn((\nu n)\ P) = fn(P)$. Then $(fn((\nu n)\ P) \cup fn(\sigma)) \setminus pr((\nu n)\ P) = (fn(P) \cup fn(\sigma)) \setminus pr(P)$. Finally,

$$\begin{aligned} cl_\sigma((\nu n)\ P) &\equiv [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid (\nu n)\ ([n : \varepsilon] \mid cl(P)) \\ &\equiv (\nu n)\ ([n : \varepsilon] \mid [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl(P)) \\ &\equiv (\nu n)\ ([n : \varepsilon] \mid cl_\sigma(P)). \end{aligned}$$

– If $n \in fn(P)$, $fn((\nu n)\ P) \cup \{n\} = fn(P)$. Then

$$\begin{aligned} (fn(P) \cup fn(\sigma)) \setminus pr(P) &= (fn((\nu n)\ P) \cup \{n\} \cup fn(\sigma)) \setminus pr((\nu n)\ P) \\ &= (fn((\nu n)\ P) \cup fn(\sigma)) \setminus pr((\nu n)\ P) \cup \{n\} \end{aligned}$$

(since $n \notin pr((\nu n)\ P)$). Finally,

$$\begin{aligned} cl_\sigma((\nu n)\ P) &\equiv [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid (\nu n)\ ([n : \varepsilon] \mid cl(P)) \\ &\equiv (\nu n)\ ([n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid [n : \varepsilon] \mid cl(P)) \\ &\equiv (\nu n)\ cl_\sigma(P). \end{aligned}$$

□

A.4. *Managing empty channels*

In this section, we adress our second problem: in π_{esc}-calculus, empty channels can be 'erased' by structural congruence rewriting. To avoid this effect, we introduce an ordering relation on terms, modulo $\equiv$, written $\preccurlyeq$. Intuitively, $P \preccurlyeq P'$ means that P' is similar to P, but contains some extra empty channels. As an example, the main axiom is $\mathbf{0} \preccurlyeq [n : \varepsilon]$.

More formally, $\preccurlyeq$ is the least relation on terms satisfying the following rules:

$P \equiv Q \;\Rightarrow\; P \preccurlyeq Q$	(π_{esc} Empty Struct)
$P \preccurlyeq Q \preccurlyeq R \;\Rightarrow\; P \preccurlyeq R$	(π_{esc} Empty Trans)
$\mathbf{0} \preccurlyeq [n : \varepsilon]$	(π_{esc} Empty Axiom)
$P \preccurlyeq P' \;\Rightarrow\; (\nu n)\, P \preccurlyeq (\nu n)\, P'$	(π_{esc} Empty Res)
$\mathbf{0} \preccurlyeq \mathbf{0}$	(π_{esc} Empty Zero)
$P \preccurlyeq P'$ and $Q \preccurlyeq Q' \;\Rightarrow\; P \mid Q \preccurlyeq P' \mid Q'$	(π_{esc} Empty Par)
$P \preccurlyeq P' \;\Rightarrow\; !P \preccurlyeq !P'$	(π_{esc} Empty Repl)
$P \preccurlyeq P' \;\Rightarrow\; \overline{M}\langle M'\rangle.P \preccurlyeq \overline{M}\langle M'\rangle.P'$	(π_{esc} Empty Output)
$P \preccurlyeq P' \;\Rightarrow\; M(x).P \preccurlyeq M(x).P'$	(π_{esc} Empty Input)
$S \preccurlyeq S' \;\Rightarrow\; [n : S] \preccurlyeq [n : S']$	(π_{esc} Empty Channel)
$P \preccurlyeq P' \;\Rightarrow\; (\nu x : M)\, P \preccurlyeq (\nu x : M)\, P'$	(π_{esc} Empty Var)
$\varepsilon \preccurlyeq \varepsilon$	(π_{esc} Empty Empty)
$S \preccurlyeq S_1$ and $S' \preccurlyeq S_1' \;\Rightarrow\; S \mid S' \preccurlyeq S_1 \mid S_1'$	(π_{esc} Empty Abs)
$P \preccurlyeq P' \;\Rightarrow\; \langle M\rangle.P \preccurlyeq \langle M\rangle.P'$	(π_{esc} Empty Out Abs)
$P \preccurlyeq P' \;\Rightarrow\; (x).P \preccurlyeq (x).P'$	(π_{esc} Empty Inp Abs)

First, in order to understand why we need such a technical tool, let us explain why we could not have imagined adding the rule $\mathbf{0} \equiv [n : \varepsilon]$ to the definition of structural congruence. In that case, we would have:

$$\begin{array}{rl} & \overline{n}\langle M\rangle.P \mid n(x).Q \\ \equiv & [n : \varepsilon] \mid [n : \varepsilon] \mid \overline{n}\langle M\rangle.P \mid n(x).Q \\ \longmapsto\!\!\!\!\rightarrow & [n : \langle M\rangle.P] \mid [n : (x).Q], \end{array}$$

and no further communication could occur. This problem is very similar to the one that motivated the introduction of the presentation predicates $P \Downarrow_1 n$ and $P \Downarrow_2 n$.

Lemma 48. $\preccurlyeq$ is an ordering relation, modulo $\equiv$.

Proof. We need to check that:

— $\preccurlyeq$ is reflexive: $P \preccurlyeq P$ since $P \equiv P$.
— $\preccurlyeq$ is transitive: by definition.
— $\preccurlyeq$ is asymmetric modulo $\equiv$, that is, $P \preccurlyeq P'$ and $P' \preccurlyeq P$ implies $P \equiv P'$: by induction on the derivation of $P \preccurlyeq P'$. □

The next few lemmas show how the relation $\preccurlyeq$ relates to free names and variables, channel presentation, validity and finally channel closure.

Lemma 49. If $P \preccurlyeq P'$, then $fn(P) \subseteq fn(P')$ and $fv(P) = fv(P')$.

Proof. The proof is by induction on the derivation of $P \preccurlyeq P'$. □

Lemma 50. If $P \preccurlyeq P'$ and $P \Downarrow_i n$, then $P' \Downarrow_i n$.

Proof. The proof is by induction on the derivation of $P \preccurlyeq P'$, first for $i = 1$, then for $i = 2$, using Lemma 32. $\square$

Corollary 51. If $P \preccurlyeq P'$, then $pr(P) \subseteq pr(P')$.

Proof. The result follows using Lemma 50. $\square$

Lemma 52. If $P \preccurlyeq P'$ and $\vdash P' : Valid$, then $\vdash P : Valid$.

Proof. The proof is by induction on the derivation of $P \preccurlyeq P'$, using Lemma 50. $\square$

Lemma 53. If $P \preccurlyeq P'$ and $\vdash P' : Valid$, then $cl(P') \equiv [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl(P)$ with $\{n_1, \ldots, n_k\} = pr(P') \setminus pr(P)$.

Proof. The proof is by induction on the derivation of $P \preccurlyeq P'$, using Corollary 51:

(π_{esc} Empty Struct) Suppose $P \equiv P'$. Then, by Lemma 32, we have $cl(P) \equiv cl(P')$, and, by Corollary 33, $pr(P) = pr(P')$.

(π_{esc} Empty Trans) Suppose $P \preccurlyeq Q \preccurlyeq R$ and $\vdash R : Valid$. By Lemma 52, $\vdash Q : Valid$. By the induction hypothesis,

$$\begin{aligned} cl(R) &\equiv [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl(Q) \\ &\equiv [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid [m_1 : \varepsilon] \mid \ldots \mid [m_p : \varepsilon] \mid cl(P) \end{aligned}$$

with $\{n_1, \ldots, n_k\} = pr(R) \setminus pr(Q)$ and $\{m_1, \ldots, m_p\} = pr(Q) \setminus pr(P)$. It remains to check that $pr(R) \setminus pr(P) = \{n_1, \ldots, n_k, m_1, \ldots, m_p\}$, which is easy by Corollary 51.

(π_{esc} Empty Axiom) We have $cl([n : \varepsilon]) = [n : \varepsilon] \equiv [n : \varepsilon] \mid \mathbf{0} = [n : \varepsilon] \mid cl(\mathbf{0})$ and $pr([n : \varepsilon]) \setminus pr(\mathbf{0}) = \{n\}$.

(π_{esc} Empty Res) Suppose that $(\nu n)\ P \preccurlyeq (\nu n)\ P'$ was derived from $P \preccurlyeq P'$. Since $\vdash (\nu n)\ P' : Valid$, $\vdash P' : Valid$. By the induction hypothesis, $cl(P') \equiv [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl(P)$ with $\{n_1, \ldots, n_k\} = pr(P') \setminus pr(P)$. There are three cases to consider:

— If $P \Downarrow_1 n$, then $P' \Downarrow_1 n$ by Lemma 50, and we have $n \neq n_i$. Moreover, $pr((\nu n)\ P') \setminus pr((\nu n)\ P) = \{n_1, \ldots, n_k\}$, and

$$\begin{aligned} cl((\nu n)\ P') &= (\nu n)\ cl(P') \\ &\equiv (\nu n)\ ([n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl(P)) \\ &\equiv [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid (\nu n)\ cl(P) \\ &= [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl((\nu n)\ P). \end{aligned}$$

— If $P \not\Downarrow_1 n$ and $P' \Downarrow_1 n$, n is one of the n_i. We may assume $n = n_1$ for example. Then we have $pr((\nu n)\ P') \setminus pr((\nu n)\ P) = \{n_2, \ldots, n_k\}$, and

$$\begin{aligned} cl((\nu n)\ P') &= (\nu n)\ cl(P') \\ &\equiv (\nu n)\ ([n : \varepsilon] \mid [n_2 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl(P)) \\ &\equiv [n_2 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid (\nu n)\ ([n : \varepsilon] \mid cl(P)) \\ &= [n_2 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl((\nu n)\ P). \end{aligned}$$

— If $P \not\Downarrow_1 n$ and $P' \not\Downarrow_1 n$, we have $n \neq n_i$ and $pr((\nu n)\ P') \setminus pr((\nu n)\ P) = \{n_1, \ldots, n_k\}$. Moreover,

$$\begin{aligned} cl((\nu n)\ P') &= (\nu n)\ ([n : \varepsilon] \mid cl(P')) \\ &\equiv (\nu n)\ ([n : \varepsilon] \mid [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl(P)) \\ &\equiv [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid (\nu n)\ ([n : \varepsilon] \mid cl(P)) \\ &= [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl((\nu n)\ P). \end{aligned}$$

(π_{esc} Empty Par) Suppose that $P \mid Q \preccurlyeq P' \mid Q'$ was derived from $P \preccurlyeq P'$ and $Q \preccurlyeq Q'$. Since $\vdash P' \mid Q' : Valid$, we have $\vdash P' : Valid$ and $\vdash Q' : Valid$. By the induction hypothesis, $cl(P') \equiv [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl(P)$ with $\{n_1, \ldots, n_k\} = pr(P') \setminus pr(P)$, and $cl(Q') \equiv [m_1 : \varepsilon] \mid \ldots \mid [m_p : \varepsilon] \mid cl(Q)$ with $\{m_1, \ldots, m_p\} = pr(Q') \setminus pr(Q)$. Since $\vdash P' \mid Q' : Valid$, we have $n_i \neq m_j$ (otherwise $P' \mid Q' \Downarrow_2 n_i$). Moreover, $pr(P') \cap pr(Q) \subseteq pr(P') \cap pr(Q') = \emptyset$ and $pr(Q') \cap pr(P) = \emptyset$. Consequently, $pr(P' \mid Q') \setminus pr(P \mid Q) = pr(P') \setminus pr(P) \cup pr(Q') \setminus pr(Q) = \{n_1, \ldots, n_k, m_1, \ldots, m_p\}$. Finally, $cl(P' \mid Q') = cl(P') \mid cl(Q') \equiv [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid [m_1 : \varepsilon] \mid \ldots \mid [m_p : \varepsilon] \mid cl(P \mid Q)$.

(π_{esc} Empty Repl) Suppose that $!P \preccurlyeq !P'$ was derived from $P \preccurlyeq P'$. Since $\vdash !P' : Valid$, we have $pr(P') = \emptyset$. Then, by the induction hypothesis, $cl(P') \equiv cl(P)$. And, since $pr(!P') = \emptyset$, we have $cl(!P') \equiv cl(!P)$.

The other cases are similar or trivial. □

Finally, the following result shows that if we translate a process P in π-calculus and rewrite it by structural congruence (thus possibly erasing some empty channels), we in fact get the direct translation of a process $\preccurlyeq P$.

Lemma 54. If $[\![P]\!] \equiv Q$ (or $Q \equiv [\![P]\!]$), there exists P' such that $Q = [\![P']\!]$ and $P' \preccurlyeq P$ (and the same for S instead of P).

Proof. The proof is by induction on the derivation of $[\![P]\!] \equiv Q$, with a special treatment for $P = (\nu x : M)\ P'$ (since this part of the reasoning is common to all cases, we include it in the induction and we consider that P is not of this form in all other cases).

(ν**x** : **M**) **P** We have $[\![(\nu x : M)\ P]\!] = [\![P]\!]\{^M/_x\} \equiv Q$. By Lemma 29, there is Q' such that $[\![P]\!] \equiv Q'$ and $Q = Q'\{^M/_x\}$. By the induction hypothesis, there is P' such that $Q' = [\![P']\!]$ and $P' \preccurlyeq P$. Then $Q = [\![P']\!]\{^M/_x\} = [\![(\nu x : M)\ P']\!]$, and we can derive $(\nu x : M)\ P' \preccurlyeq (\nu x : M)\ P$.

(π Struct Refl) This case is trivial, because $\preccurlyeq$ is reflexive.

(π Struct Symm) This case follows directly by induction.

(π Struct Trans) This case is trivial, because $\preccurlyeq$ is transitive.

(π Struct Res) Suppose $[\![P]\!] = (\nu n)\ P_1 \equiv (\nu n)\ Q_1 = Q$ was derived from $P_1 \equiv Q_1$. Necessarily, $P = (\nu n)\ P_2$ with $P_1 = [\![P_2]\!]$. By the induction hypothesis, there is P_2' such that $Q_1 = [\![P_2']\!]$ and $P_2' \preccurlyeq P_2$. Let $P' = (\nu n)\ P_2'$. We have $Q = (\nu n)\ [\![P_2']\!] = [\![P']\!]$ and $P' \preccurlyeq (\nu n)\ P_2 = P$.

(π Struct Par) Suppose that $[\![P]\!] = P_0 \mid R \equiv Q \mid R$ was derived from $P_0 \equiv Q$. By Lemma 43, there are two cases to consider:

— $P = P_1 \mid P_2$ with $[\![P_1]\!] = P_0$ and $[\![P_2]\!] = R$. We have $[\![P_1]\!] \equiv Q$. By the induction hypothesis, there is P_1' such that $Q = [\![P_1']\!]$ and $P_1' \preccurlyeq P_1$. Then $Q \mid R = [\![P_1' \mid P_2]\!]$ and $P_1' \mid P_2 \preccurlyeq P_1 \mid P_2 = P$.

— $P = [n : S_1 \mid S_2]$ with $[\![S_1]\!]_n = P_0$ and $[\![S_2]\!]_n = R$. We have $[\![S_1]\!]_n \equiv Q$. By the induction hypothesis, there is S_1' such that $Q = [\![S_1']\!]_n$ and $S_1' \preccurlyeq S_1$. Then $Q \mid R = [\![S_1' \mid S_2]\!]_n = [\![[n : S_1' \mid S_2]]\!]$ and $[n : S_1' \mid S_2] \preccurlyeq [n : S_1 \mid S_2] = P$.

(π Struct Par Zero) Suppose $[\![P]\!] = Q \mid \mathbf{0} \equiv Q$. By Lemma 43, there are two cases to consider:

— $P = [n : S \mid S']$ with $[\![S]\!]_n = Q$ and $[\![S']\!]_n = \mathbf{0}$. Necessarily, $S' = \varepsilon$. We have $P = [n : S \mid \varepsilon] \equiv [n : S]$. Finally, $Q = [\![[n : S]]\!]$ and $[n : S] \preccurlyeq P$.

— $P = P_1 \mid P_2$ with $[\![P_1]\!] = Q$ and $[\![P_2]\!] = \mathbf{0}$. There are two possibilities. If $P_2 = \mathbf{0}$, then $P = P_1 \mid \mathbf{0} \equiv P_1$, so $P_1 \preccurlyeq P$. Otherwise, $P_2 = [n : \varepsilon]$, and then $P_1 \equiv P_1 \mid \mathbf{0} \preccurlyeq P_1 \mid [n : \varepsilon] = P$.

The other cases are similar. □

A.5. *Channels and parallel composition*

In this section, we adress the third problem: to complete an inductive proof, we need to show how communications involving an empty channel added by channel closure can be reported to (possibly non-empty) channels provided by a process in parallel.

As a first step, we define a syntactic operator that finds the channels n_i in a process and adds some concretions and abstractions S_i within it.

Definition 55. $P\{n_i \leftrightarrow S_i\}$ is defined by $[n_j : S']\{n_i \leftrightarrow S_i\} = [n_j : S'\{n_i \leftrightarrow S_i\} \mid S_j]$ and is a homomorphism for all other constructs.

Next, we show that this operator is preserved by structural congruence and reduction in π_{esc}.

Lemma 56.

1 If $P \equiv Q$, then $P\{n_i \leftrightarrow S_i\} \equiv Q\{n_i \leftrightarrow S_i\}$.
2 If $fv(S_i) \subseteq dom(\sigma)$ and $\sigma : P \longmapsto P'$, then $\sigma : P\{n_i \leftrightarrow S_i\} \longmapsto P'\{n_i \leftrightarrow S_i\}$.

Proof.

1 The proof is by induction on the derivation of $P \equiv Q$.
2 The proof is by induction on the derivation of $\sigma : P \longmapsto P'$. □

The next lemma shows that translating a π_{esc}-process back into π-calculus after some channel extensions is equivalent to translating the original process and the extensions separately, as if these were simple input/output instructions. Thus, we 'forget' the first step of communication (the entering of a channel).

Lemma 57. If $\vdash P : \mathit{Valid}$, then $[\![P\{n_i \leftrightarrow S_i\}]\!] \equiv [\![P]\!] \mid [\![S_{i_1}]\!]_{n_{i_1}} \mid \ldots \mid [\![S_{i_k}]\!]_{n_{i_k}}$ where $\{i_1, \ldots, i_k\} = \{i \ / \ P \Downarrow_1 n_i\}$.

Proof. The proof is by induction on the structure of P:

— $[\![((\nu n)\ P)\{n_i \leftrightarrow S_i\}]\!] = (\nu n)\ [\![P\{n_i \leftrightarrow S_i\}]\!]$ (we can always suppose $n \notin fn(\{n_i \leftrightarrow S_i\})$. Since $\vdash (\nu n)\ P : Valid$ implies $\vdash P : Valid$, we have $[\![P\{n_i \leftrightarrow S_i\}]\!] \equiv [\![P]\!] \mid [\![S_{i_1}]\!]_{n_{i_1}} \mid \dots \mid [\![S_{i_k}]\!]_{n_{i_k}}$ with $\{i_1, \dots, i_k\} = \{i \ / \ P \Downarrow_1 n_i\}$ by the induction hypothesis. Since $n \neq n_i$, we have $\{i \ / \ (\nu n)\ P \Downarrow_1 n_i\} = \{i \ / \ P \Downarrow_1 n_i\}$. Finally,

$$\begin{aligned} [\![((\nu n)\ P)\{n_i \leftrightarrow S_i\}]\!] &\equiv (\nu n)\ ([\![P]\!] \mid [\![S_{i_1}]\!]_{n_{i_1}} \mid \dots \mid [\![S_{i_k}]\!]_{n_{i_k}}) \\ &\equiv [\![(\nu n)\ P]\!] \mid [\![S_{i_1}]\!]_{n_{i_1}} \mid \dots \mid [\![S_{i_k}]\!]_{n_{i_k}}. \end{aligned}$$

— $[\![\mathbf{0}\{n_i \leftrightarrow S_i\}]\!] = [\![\mathbf{0}]\!]$ and $\{i \ / \ \mathbf{0} \Downarrow_1 n_i\} = \emptyset$ trivially.

— $[\![(P \mid Q)\{n_i \leftrightarrow S_i\}]\!] = [\![P\{n_i \leftrightarrow S_i\}]\!] \mid [\![Q\{n_i \leftrightarrow S_i\}]\!]$. Since $\vdash P \mid Q : Valid$, we have $\vdash P : Valid$ and $\vdash Q : Valid$. Moreover, we cannot have $P \Downarrow_1 n_i$ and $Q \Downarrow_1 n_i$ at the same time, otherwise we would have $P \mid Q \Downarrow_2 n_i$. Thus, $\{i \ / \ P \mid Q \Downarrow_1 n_i\} = \{i \ / \ P \Downarrow_1 n_i\} \cup \{i \ / \ Q \Downarrow_1 n_i\}$ with a disjunctive union. By the induction hypothesis, $[\![P\{n_i \leftrightarrow S_i\}]\!] \equiv [\![P]\!] \mid [\![S_{i_1}]\!]_{n_{i_1}} \mid \dots \mid [\![S_{i_k}]\!]_{n_{i_k}}$ where $\{i_1, \dots, i_k\} = \{i \ / \ P \Downarrow_1 n_i\}$, and $[\![Q\{n_i \leftrightarrow S_i\}]\!] \equiv [\![Q]\!] \mid [\![S_{j_1}]\!]_{n_{j_1}} \mid \dots \mid [\![S_{j_p}]\!]_{n_{j_p}}$ where $\{j_1, \dots, j_p\} = \{i \ / \ Q \Downarrow_1 n_i\}$. Then $[\![(P \mid Q)\{n_i \leftrightarrow S_i\}]\!] \equiv [\![P \mid Q]\!] \mid [\![S_{i_1}]\!]_{n_{i_1}} \mid \dots \mid [\![S_{i_k}]\!]_{n_{i_k}} \mid [\![S_{j_1}]\!]_{n_{j_1}} \mid \dots \mid [\![S_{j_p}]\!]_{n_{j_p}}$ with $\{i_1, \dots, i_k, j_1, \dots . j_p\} = \{i \ / \ P \mid Q \Downarrow_1 n_i\}$.

— Since $\vdash !P : Valid$, we have $P \not\Downarrow_1 n$ for all $n \in Name$. Moreover, $\vdash P : Valid$ and $[\![P\{n_i \leftrightarrow S_i\}]\!] = [\![P]\!]$ by the induction hypothesis. Then $[\![(!P)\{n_i \leftrightarrow S_i\}]\!] = ![\![P\{n_i \leftrightarrow S_i\}]\!] = ![\![P]\!] = [\![!P]\!]$, and $\{i \ / \ !P \Downarrow_1 n_i\} = \{i \ / \ P \Downarrow_1 n_i\} = \emptyset$.

— $[\![[n_j : S]\{n_i \leftrightarrow S_i\}]\!] = [\![[n_j : S\{n_i \leftrightarrow S_i\} \mid S_j]]\!] = [\![S\{n_i \leftrightarrow S_i\}]\!]_{n_j} \mid [\![S_j]\!]_{n_j}$. Since $\vdash [n_j : S] : Valid$, $\vdash S : Valid$. By the induction hypothesis, $[\![S\{n_i \leftrightarrow S_i\}]\!]_{n_j} \equiv [\![S]\!]_{n_j} \mid [\![S_{i_1}]\!]_{n_{i_1}} \mid \dots \mid [\![S_{i_k}]\!]_{n_{i_k}}$ where $\{i_1, \dots, i_k\} = \{i \ / \ S \Downarrow_1 n_i\}$. Then $[\![[n_j : S]\{n_i \leftrightarrow S_i\}]\!] \equiv [\![[n_j : S]]\!] \mid [\![S_{i_1}]\!]_{n_{i_1}} \mid \dots \mid [\![S_{i_k}]\!]_{n_{i_k}} \mid [\![S_j]\!]_{n_j}$. And $\{i \ / \ [n_j : S] \Downarrow_1 n_i\} = \{j\} \cup \{i \ / \ S \Downarrow_1 n_i\}$ with a disjunctive union (otherwise $[n_j : S] \Downarrow_2 n_j$, which is impossible since $\vdash [n_j : S] : Valid$).

The other cases are very similar. □

The next lemma shows that we can increase the contents of channels placed in parallel without affecting the possible reductions.

Lemma 58. If $\vdash P : Valid$, $P \not\Downarrow_1 n_i$, $fv(S_i) \subseteq dom(\sigma)$ and $\sigma : [n_1 : \varepsilon] \mid \dots \mid [n_k : \varepsilon] \mid P \longmapsto^* P'$, then $\sigma : [n_1 : S_1] \mid \dots \mid [n_k : S_k] \mid P \longmapsto^* P''$ with the condition $[\![P'']\!] \equiv [\![P']\!] \mid [\![S_1]\!]_{n_1} \mid \dots \mid [\![S_k]\!]_{n_k}$.

Proof. Let $Q = [n_1 : \varepsilon] \mid \dots \mid [n_k : \varepsilon] \mid P$. By Lemma 56, $\sigma : Q\{n_i \leftrightarrow S_i\} \longmapsto^* P'\{n_i \leftrightarrow S_i\}$. Since $P \not\Downarrow_1 n_i$, we have $P\{n_i \leftrightarrow S_i\} = P$ and thus $Q\{n_i \leftrightarrow S_i\} = [n_1 : S_1] \mid \dots \mid [n_k : S_k] \mid P$. Moreover, since $P \not\Downarrow_1 n_i$, we have $Q \Downarrow_1 n_i$ but $Q \not\Downarrow_2 n_i$. It follows easily that $\vdash Q : Valid$. By Proposition 6., $\vdash P' : Valid$. Using Lemma 57, we get $[\![P'\{n_i \leftrightarrow S_i\}]\!] = [\![P']\!] \mid [\![S_1]\!]_{n_1} \mid \dots \mid [\![S_k]\!]_{n_k}$ ($P' \Downarrow_1 n_i$ for all i because of Lemma 34 and $Q \Downarrow_1 n_i$). □

The next lemma generalises the previous one by showing that we can use the channels of another process in parallel if they are available, instead of empty channels.

Lemma 59. If $\vdash P : Valid$, $\vdash Q : Valid$, $P \not\Downarrow_1 n_i$, $Q \Downarrow_1 n_i$, $fv(Q) \subseteq dom(\sigma)$ and $\sigma : [n_1 : \varepsilon] \mid \dots \mid [n_k : \varepsilon] \mid P \longmapsto^* P'$, then $\sigma : P \mid Q \longmapsto^* P''$ with $[\![P'']\!] \equiv [\![P' \mid Q]\!]$.

Proof. From Corollary 5, we get $Q \equiv (\nu m_1) \ldots (\nu m_p)\ (\nu x_1 : M_1) \ldots (\nu x_{p'} : M_{p'})\ ([n_1 : S_1] \mid \ldots \mid [n_k : S_k] \mid Q')$ with $n_i \neq m_j$. We also suppose $n_i \notin fn(P) \cup fn([\![P']\!])$, $x_i \notin fv(P) \cup fv([\![P']\!]) \cup dom(\sigma)$ and $x_i \neq x_j$ for $i \neq j$. Then $P \mid Q \equiv (\nu m_1) \ldots (\nu m_p)\ (\nu x_1 : M_1) \ldots (\nu x_{p'} : M_{p'})\ ([n_1 : S_1] \mid \ldots \mid [n_k : S_k] \mid P \mid Q')$. By Lemma 39, we have $\{{}^{M_{p'}}/_{x_{p'}}\} \uplus \ldots \uplus \{{}^{M_1}/_{x_1}\} \uplus \sigma : [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid P \longmapsto^* P'$. By Lemma 58, there exists P_1 such that $\{{}^{M_{p'}}/_{x_{p'}}\} \uplus \ldots \uplus \{{}^{M_1}/_{x_1}\} \uplus \sigma : [n_1 : S_1] \mid \ldots \mid [n_k : S_k] \mid P \longmapsto^* P_1$ and $[\![P_1]\!] \equiv [\![P']\!] \mid [\![S_1]\!]_{n_1} \mid \ldots \mid [\![S_k]\!]_{n_k}$. Then we can derive $\sigma : P \mid Q \longmapsto^* P'' = (\nu m_1) \ldots (\nu m_p)\ (\nu x_1 : M_1) \ldots (\nu x_{p'} : M_{p'})\ (P_1 \mid Q')$. Finally,

$$\begin{aligned}
[\![P'']\!] &= (\nu m_1) \ldots (\nu m_p)\ ([\![P_1]\!] \mid [\![Q']\!])\{{}^{M_{p'}}/_{x_{p'}}\} \ldots \{{}^{M_1}/_{x_1}\} \\
&\equiv (\nu m_1) \ldots (\nu m_p)\ ([\![P']\!] \mid [\![S_1]\!]_{n_1} \mid \ldots \mid [\![S_k]\!]_{n_k} \mid [\![Q']\!])\{{}^{M_{p'}}/_{x_{p'}}\} \ldots \{{}^{M_1}/_{x_1}\} \\
&\equiv [\![P']\!] \mid (\nu m_1) \ldots (\nu m_p)\ ([\![S_1]\!]_{n_1} \mid \ldots \mid [\![S_k]\!]_{n_k} \mid [\![Q']\!])\{{}^{M_{p'}}/_{x_{p'}}\} \ldots \{{}^{M_1}/_{x_1}\} \\
&\equiv [\![P']\!] \mid [\![Q]\!] \\
&= [\![P' \mid Q]\!].
\end{aligned}$$

□

Finally, the next two lemmas are the missing lemmas from Section A.3, that is, the inductive steps for parallel composition and channels.

Lemma 60. If $\vdash P \mid Q : Valid$, $fv(Q) \subseteq dom(\sigma)$ and $\sigma : cl_\sigma(P) \longmapsto^* P'$, then $\sigma : cl_\sigma(P \mid Q) \longmapsto^* P''$ with $[\![P'']\!] \equiv [\![P' \mid Q]\!]$.

Proof. Let us write $cl_\sigma(P) = [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid [m_1 : \varepsilon] \mid \ldots \mid [m_{k'} : \varepsilon] \mid cl(P)$ with $(fn(P) \cup fn(\sigma)) \setminus pr(P) = \{n_1, \ldots, n_k, m_1, \ldots, m_{k'}\}$ divided so that $Q \Downarrow_1 n_i$ and $Q \not\Downarrow_1 m_j$. Let $P_0 = [m_1 : \varepsilon] \mid \ldots \mid [m_{k'} : \varepsilon] \mid cl(P)$. We know that $\sigma : [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid P_0 \longmapsto^* P'$. Since $\vdash P \mid Q : Valid$, $\vdash P : Valid$. By definition of m_j, we have $P \not\Downarrow_1 m_j$, so $\vdash P_0 : Valid$. Moreover, $P_0 \not\Downarrow_1 n_i$, $\vdash cl(Q) : Valid$ and $cl(Q) \Downarrow_1 n_i$ by Lemma 36. Using Lemma 59, there exists P_1 such that $\sigma : P_0 \mid cl(Q) \longmapsto^* P_1$ and $[\![P_1]\!] \equiv [\![P' \mid cl(Q)]\!] \equiv [\![P' \mid Q]\!]$ (using Lemma 41). We can write:

$$\begin{aligned}
(fn(P \mid Q) \cup fn(\sigma)) \setminus pr(P \mid Q) &= (fn(P) \cup fn(Q) \cup fn(\sigma)) \setminus (pr(P) \cup pr(Q)) \\
&= ((fn(P) \cup fn(\sigma)) \setminus pr(P)) \setminus pr(Q) \\
&\cup\ ((fn(Q) \cup fn(\sigma)) \setminus pr(Q)) \setminus pr(P).
\end{aligned}$$

Because $m_j \in (fn(P) \cup fn(\sigma)) \setminus pr(P)$ but $m_j \notin pr(Q)$, we have $m_j \in (fn(P \mid Q) \cup fn(\sigma)) \setminus pr(P \mid Q)$. This allows us to write $cl_\sigma(P \mid Q) = [p_1 : \varepsilon] \mid \ldots [p_l : \varepsilon] \mid [m_1 : \varepsilon] \mid \ldots \mid [m_{k'} : \varepsilon] \mid cl(P) \mid cl(Q) = [p_1 : \varepsilon] \mid \ldots [p_l : \varepsilon] \mid P_0 \mid cl(Q)$. And we can derive $\sigma : cl_\sigma(P \mid Q) \longmapsto^* P'' = [p_1 : \varepsilon] \mid \ldots [p_l : \varepsilon] \mid P_1$. Finally, $[\![P'']\!] \equiv [\![P_1]\!] \equiv [\![P' \mid Q]\!]$. □

Lemma 61. If $\vdash [n : S \mid S'] : Valid$, $fv(S') \subseteq dom(\sigma)$ and $\sigma : cl_\sigma([n : S]) \longmapsto^* P$, then $\sigma : cl_\sigma([n : S \mid S']) \longmapsto^* P'$ with $[\![P']\!] \equiv [\![P]\!] \mid [\![S']\!]_n$.

Proof. Let us write $cl_\sigma([n : S]) = [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid [n : cl(S)]$ with $(fn([n : S]) \cup fn(\sigma)) \setminus pr([n : S]) = \{n_1, \ldots, n_k\}$. By Lemma 56, $\sigma : cl_\sigma([n : S])\{n \leftrightarrow cl(S')\} = [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid [n : cl(S) \mid cl(S')] \longmapsto^* P\{n \leftrightarrow cl(S')\}$. Moreover, $pr([n : S]) = pr([n : S \mid S']) = \{n\}$. Then

$$\begin{aligned}(fn([n:S \mid S']) \cup fn(\sigma)) \setminus pr([n:S \mid S'])&\\ &= (\{n\} \cup fn(S) \cup fn(S') \cup fn(\sigma)) \setminus \{n\}\\ &= \{n_1,\ldots,n_k\} \cup (fn(S') \setminus \{n\}) \supseteq \{n_1,\ldots,n_k\}.\end{aligned}$$

We can derive $\sigma : cl_\sigma([n : S \mid S']) \longmapsto^* P' = [m_1 : \varepsilon] \mid \ldots \mid [m_p : \varepsilon] \mid P\{n \leftrightarrow cl(S')\}$. Since $\vdash [n : S] : Valid$, $\vdash P : Valid$ by Lemma 36 and Proposition 6. Moreover, by Corollary 35 and Lemma 36, $pr(P) = pr([n : S]) = \{n\}$. Finally, using Lemma 57, $[\![P']\!] \equiv [\![P\{n \leftrightarrow cl(S')\}]\!] \equiv [\![P]\!] \mid [\![cl(S')]\!]_n \equiv [\![P]\!] \mid [\![S']\!]_n$ (using Lemma 41). □

A.6. *Proof of Proposition 8*

Proposition 8. If $\sigma : P \longmapsto Q$, then $[\![P]\!]\sigma^* \stackrel{\equiv}{\longrightarrow} [\![Q]\!]\sigma^*$, where $\stackrel{\equiv}{\longrightarrow}$ is either $\equiv$ or $\longrightarrow$.

Proof. The proof is by induction on the derivation of $\sigma : P \longmapsto Q$:

(π_{esc} Red Subst Out) Suppose that $\sigma : \overline{x}\langle M'\rangle.P \longmapsto \overline{M}\langle M'\rangle.P$ with $x\sigma = M$. We have, obviously, $x\sigma^* = M\sigma^*$. Then $[\![\overline{x}\langle M'\rangle.P]\!]\sigma^* = \overline{(x\sigma^*)}\langle M'\sigma^*\rangle.([\![P]\!]\sigma^*) = \overline{(M\sigma^*)}\langle M'\sigma^*\rangle.([\![P]\!]\sigma^*) = [\![\overline{M}\langle M'\rangle.P]\!]\sigma^*$.

(π_{esc} Red Subst In) This case is similar to (π_{esc} Red Subst Out).

(π_{esc} Red Output) Suppose that $\sigma : [n : S] \mid \overline{n}\langle M\rangle.P \longmapsto [n : S \mid \langle M\rangle.P]$. We have $[\![[n : S] \mid \overline{n}\langle M\rangle.P]\!] = [\![S]\!]_n \mid \overline{n}\langle M\rangle.[\![P]\!] = [\![S \mid \langle M\rangle.P]\!]_n = [\![[n : S \mid \langle M\rangle.P]]\!]$. Then $[\![[n : S] \mid \overline{n}\langle M\rangle.P]\!]\sigma^* = [\![[n : S \cup \{\langle M\rangle.P\}]]\!]\sigma^*$.

(π_{esc} Red Input) This case is similar to (π_{esc} Red Output).

(π_{esc} Red Comm) Suppose that $\sigma : [n : S \mid (\langle M\rangle.P \mid (x).Q)] \longmapsto [n : S] \mid (P \mid (\nu x : M)\ Q)$ with $x \neq M$ (we also suppose $x \notin dom(\sigma)$). We have:

$$\begin{aligned}&[\![[n : S \mid (\langle M\rangle.P \mid (x).Q)]]\!]\sigma^*\\ &\qquad = [\![S]\!]_n\sigma^* \mid (\overline{n}\langle M\sigma^*\rangle.([\![P]\!]\sigma^*) \mid n(x).([\![Q]\!]\sigma^*))\\ &\qquad \longrightarrow [\![S]\!]_n\sigma^* \mid ([\![P]\!]\sigma^* \mid ([\![Q]\!]\sigma^*)\{^{M\sigma^*}/_x\}) \quad \text{(by (π Red Comm))}\\ &\qquad = [\![S]\!]_n\sigma^* \mid ([\![P]\!]\sigma^* \mid ([\![Q]\!]\{^M/_x\})\sigma^*) \quad \text{(using Lemma 26)}\\ &\qquad = [\![[n : S] \mid (P \mid (\nu x : M)\ Q)]\!]\sigma^*\end{aligned}$$

(π_{esc} Red Par) Suppose that $\sigma : P \mid Q \longmapsto P' \mid Q$ was derived from $\sigma : P \longmapsto P'$. By the induction hypothesis, $[\![P]\!]\sigma^* \stackrel{\equiv}{\longrightarrow} [\![P']\!]\sigma^*$. Then $[\![P \mid Q]\!]\sigma^* = [\![P]\!]\sigma^* \mid [\![Q]\!]\sigma^* \stackrel{\equiv}{\longrightarrow} [\![P']\!]\sigma^* \mid [\![Q]\!]\sigma^* = [\![P' \mid Q]\!]\sigma^*$ by (π Struct Par) or (π Red Par).

(π_{esc} Red Res) Suppose that $\sigma : (\nu n)\ P \longmapsto (\nu n)\ P'$ was derived from $\sigma : P \longmapsto P'$. By the induction hypothesis, $[\![P]\!]\sigma^* \stackrel{\equiv}{\longrightarrow} [\![P']\!]\sigma^*$. Then $[\![(\nu n)\ P]\!]\sigma^* = (\nu n)\ ([\![P]\!]\sigma^*) \stackrel{\equiv}{\longrightarrow} (\nu n)\ ([\![P']\!]\sigma^*) = [\![(\nu n)\ P]\!]\sigma^*$ by (π Struct Res) or (π Red Res) (we can always suppose $n \notin im(\sigma)$).

(π_{esc} Red Var) Suppose that $\sigma : (\nu x : M)\ P \longmapsto (\nu x : M)\ P'$ was derived from $\{^M/_x\} \uplus \sigma : P \longmapsto P'$ with $x \notin dom(\sigma)$. By the induction hypothesis, we have $[\![P]\!](\{^M/_x\} \uplus \sigma)^* \stackrel{\equiv}{\longrightarrow} [\![Q]\!](\{^M/_x\} \uplus \sigma)^*$. Then $[\![(\nu x : M)\ P]\!]\sigma^* = [\![P]\!]\{^M/_x\}\sigma^* = [\![P]\!](\{^M/_x\} \uplus \sigma)^* \stackrel{\equiv}{\longrightarrow} [\![Q]\!](\{^M/_x\} \uplus \sigma)^* = [\![Q]\!]\{^M/_x\}\sigma^* = [\![(\nu x : M)\ Q]\!]\sigma^*$, using Lemma 27.

(π_{esc} Red Struct) Suppose that $\sigma : P \longmapsto Q$ was derived from $P \equiv P'$, $\sigma : P' \longmapsto Q'$ and $Q' \equiv Q$. By Lemma 42, $[\![P]\!] \equiv [\![P']\!]$ and by Lemma 28, $[\![P]\!]\sigma^* \equiv [\![P']\!]\sigma^*$. By the induction hypothesis, $[\![P']\!]\sigma^* \overset{\equiv}{\longrightarrow} [\![Q']\!]\sigma^*$. Using Lemmas 42 and 28, $[\![Q']\!]\sigma^* \equiv [\![Q]\!]\sigma^*$. Finally, we can derive $[\![P]\!]\sigma^* \overset{\equiv}{\longrightarrow} [\![Q]\!]\sigma^*$ by (π Struct Trans) or (π Red Struct). $\square$

A.7. *Proof of Proposition 9*

Proposition 9. If $[\![P]\!]\sigma^* \longrightarrow Q$, $\vdash P : \mathit{Valid}$ and $fv(P) \subseteq dom(\sigma)$, then there is a process P' such that $\sigma : cl_\sigma(P) \longmapsto^+ P'$ and $[\![P']\!]\sigma^* \equiv Q$.

Proof. The proof is by induction on the derivation of $[\![P]\!]\sigma^* \longrightarrow Q$, with a special treatment for $P = (\nu x : M)\ P'$ (since this part of the reasoning is common to all cases, we include it in the induction and we consider that P is not of this form in all other cases).

(ν**x** : **M**) **P** Suppose that $[\![(\nu x : M)\ P]\!]\sigma^* \longrightarrow Q$, $\vdash (\nu x : M)\ P : \mathit{Valid}$ and $fv((\nu x : M)\ P) \subseteq dom(\sigma)$. We also suppose $x \notin dom(\sigma)$. We have $[\![P]\!]\{^M/_x\}\sigma^* \longrightarrow Q$. Let $\sigma' = \{^M/_x\} \uplus \sigma$. By Lemma 27, $\sigma'^* = \{^M/_x\}\sigma^*$. So we have $[\![P]\!]\sigma'^* \longrightarrow Q$, $\vdash P : \mathit{Valid}$ (from $\vdash (\nu x : M)\ P : \mathit{Valid}$) and $fv(P) \subseteq dom(\sigma')$. By the induction hypothesis, there exists P' such that $\sigma' : cl_{\sigma'}(P) \longmapsto^+ P'$ and $[\![P']\!]\sigma'^* \equiv Q$. Then we can derive $\sigma : (\nu x : M)\ cl_{\sigma'}(P) \longmapsto^+ (\nu x : M)\ P'$ by (π_{esc} Red Var). By Lemma 45, $(\nu x : M)\ cl_{\sigma'}(P) \equiv cl_\sigma((\nu x : M)\ P)$, so we can derive $\sigma : cl_\sigma((\nu x : M)\ P) \longmapsto^+ (\nu x : M)\ P'$. And $[\![(\nu x : M)\ P']\!]\sigma^* = [\![P']\!]\{^M/_x\}\sigma^* = [\![P']\!]\sigma'^* \equiv Q$.

(π Red Comm) Suppose that $[\![P]\!]\sigma^* = \overline{m}\langle M\rangle.P_1 \mid n(x).P_2 \longrightarrow P_1 \mid P_2\{^m/_x\} = Q$, $\vdash P : \mathit{Valid}$ and $fv(P) \subseteq dom(\sigma)$. Necessarily, $[\![P]\!] = \overline{M_1}\langle M_2\rangle.P_3 \mid M_3(x).P_4$ with $M_1\sigma^* = n$, $M_2\sigma^* = m$, $P_3\sigma^* = P_1$, $M_3\sigma^* = n$ and $P_4\sigma^* = P_2$ (we can always suppose $x \neq M_2$ and $x \notin dom(\sigma)$). By Lemma 43, there are two cases to consider:

— $P = [p : S \mid S']$ with $[\![S]\!]_p = \overline{M_1}\langle M_2\rangle.P_3$ and $[\![S']\!]_p = M_3(x).P_4$. The first assertion implies $M_1 = p$ and $S = \langle M_2\rangle.P_5$ with $[\![P_5]\!] = P_3$. The second assertion implies $M_3 = p$ and $S' = (x).P_6$ with $[\![P_6]\!] = P_4$. Then we have $n = M_1\sigma^* = p\sigma^* = p$. To summarise, $P = [n : \langle M_2\rangle.P_5 \mid (x).P_6]$. Since $x \neq M_2$, we can derive $\sigma : P \longmapsto [n : \varepsilon] \mid (P_5 \mid (\nu x : M_2)\ P_6)$ by (π_{esc} Red Comm). Then $\sigma : cl_\sigma(P) \longmapsto P' = [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid [n : \varepsilon] \mid (cl(P_5) \mid (\nu x : M_2)\ cl(P_6))$. We have $[\![P']\!] \equiv [\![cl(P_5)]\!] \mid [\![cl(P_6)]\!]\{^{M_2}/_x\} \equiv [\![P_5]\!] \mid [\![P_6]\!]\{^{M_2}/_x\} = P_3 \mid P_4\{^{M_2}/_x\}$. Then, by Lemma 28, $[\![P']\!]\sigma^* \equiv P_3\sigma^* \mid P_4\{^{M_2}/_x\}\sigma^*$. Using Lemma 26, $[\![P']\!]\sigma^* \equiv P_3\sigma^* \mid P_4\sigma^*\{^{M_2\sigma^*}/_x\} = P_1 \mid P_2\{^m/_x\} = Q$.

— $P = P_5 \mid P_6$ with $[\![P_5]\!] = \overline{M_1}\langle M_2\rangle.P_3$ and $[\![P_6]\!] = M_3(x).P_4$. There can be some variable restrictions in P_5 and P_6. We take all of them into account: $P_5 = (\nu x_1 : N_1)\ \ldots\ (\nu x_k : N_k)\ P_5'$ and $P_6 = (\nu y_1 : N_1')\ \ldots\ (\nu y_{k'} : N_{k'}')\ P_6'$. We can always rename the variables in order to have the following assumptions: $x_i \notin dom(\sigma) \cup fv(P_4) \cup fv(P_6) \cup \{M_3, y_1, \ldots, y_{k'}, N_1', \ldots, N_{k'}'\}$ and $y_j \notin dom(\sigma) \cup fv(P_3) \cup fv(P_5') \cup \{x, M_2, x_1, \ldots, x_k, N_1, \ldots, N_k\}$. Let $\sigma_x = \{^{N_k}/_{x_k}\} \ldots \{^{N_1}/_{x_1}\}$, $\sigma_x' = \{^{N_k}/_{x_k}\} \uplus \ldots \uplus \{^{N_1}/_{x_1}\}$, $\sigma_y = \{^{N_{k'}'}/_{y_{k'}}\} \ldots \{^{N_1'}/_{y_1}\}$ and $\sigma_y' = \{^{N_{k'}'}/_{y_{k'}}\} \uplus \ldots \uplus \{^{N_1'}/_{y_1}\}$. Since $fv(P) \subseteq dom(\sigma)$, we have $N_i \in \mathit{Name} \cup dom(\sigma) \cup \{x_1, \ldots, x_{i-1}\}$ and $N_j' \in \mathit{Name} \cup dom(\sigma) \cup \{y_1, \ldots, y_{j-1}\}$. With the above conditions, we can prove by Lemma 27 that $(\sigma_y' \uplus \sigma_x' \uplus \sigma)^* = \sigma_y\sigma_x\sigma^*$.

Moreover, $\sigma_y\sigma_x = \sigma_x\sigma_y$ easily. Then we have $[\![P_5']\!]\sigma_x = [\![P_5]\!] = \overline{M_1}\langle M_2\rangle.P_3$ and $[\![P_6']\!]\sigma_y = [\![P_6]\!] = M_3(x).P_4$. Then $[\![P_5']\!] = \overline{M_1'}\langle M_2'\rangle.P_3'$ with $M_1'\sigma_x = M_1$, $M_2'\sigma_x = M_2$ and $P_3'\sigma_x = P_3$, and $[\![P_6']\!] = M_3'(x).P_4'$ with $M_3'\sigma_y = M_3$ and $P_4'\sigma_y = P_4$. Using Lemma 44, there are four cases to consider:

- $P_5' = [p : \langle M_2'\rangle.P_3'']$ with $M_1' = p$ and $[\![P_3'']\!] = P_3'$, and $P_6' = [p' : (x).P_4'']$ with $M_3' = p'$ and $[\![P_4'']\!] = P_4'$. In this case, $n = M_1\sigma^* = M_1'\sigma_x\sigma^* = p\sigma_x\sigma^* = p$ and $n = M_3\sigma^* = M_3'\sigma_y\sigma^* = p'\sigma_y\sigma^* = p'$. Then $P_5' \Downarrow_1 n$ and $P_6' \Downarrow_1 n$. We can derive $P_5 \Downarrow_1 n$ and $P_6 \Downarrow_1 n$. Finally, $P \Downarrow_2 n$, but this is in contradiction with $\vdash P : \mathit{Valid}$. Consequently, this case is impossible.
- $P_5' = [p : \langle M_2'\rangle.P_3'']$ with $M_1' = p$ and $[\![P_3'']\!] = P_3'$, and $P_6' = M_3'(x).P_4''$ with $[\![P_4'']\!] = P_4'$. In this case, $n = M_1\sigma^* = M_1'\sigma_x\sigma^* = p\sigma_x\sigma^* = p$. We have $fv(P_6') \subseteq \{y_1,\ldots,y_{k'}\} \cup fv(P_6) \subseteq \{y_1,\ldots,y_{k'}\} \cup fv(P) \subseteq dom(\sigma_x' \uplus \sigma_y' \uplus \sigma)$. Moreover, $M_3'(\sigma_y' \uplus \sigma_x' \uplus \sigma)^* = M_3'\sigma_y\sigma_x\sigma^* = M_3\sigma_x\sigma^* = M_3\sigma^* = n$. Thus, by Lemma 38, we can derive $\sigma_y' \uplus \sigma_x' \uplus \sigma : P_6' \longmapsto^* n(x).P_4''$. With similar reasoning, $fv(P_5') \subseteq dom(\sigma_x' \uplus \sigma_y' \uplus \sigma)$. We can derive

$$\begin{aligned}
&\sigma_y' \uplus \sigma_x' \uplus \sigma : P_5' \mid P_6' \\
&\qquad\longmapsto^* \quad [n : \langle M_2'\rangle.P_3''] \mid n(x).P_4'' \quad (\text{by } (\pi_{esc} \text{ Red Par})) \\
&\qquad\longmapsto \quad [n : \langle M_2'\rangle.P_3'' \mid (x).P_4''] \quad (\text{by } (\pi_{esc} \text{ Red Input})) \\
&\qquad\longmapsto \quad [n : \varepsilon] \mid (P_3'' \mid (\nu x : M_2')\ P_4'') \quad (\text{by } (\pi_{esc} \text{ Red Comm}))
\end{aligned}$$

  With the above conditions, we have $P = P_5 \mid P_6 \equiv (\nu x_1 : N_1) \ldots (\nu x_k : N_k)\ (P_5' \mid P_6) \equiv (\nu x_1 : N_1) \ldots (\nu x_k : N_k)\ (\nu y_1 : N_1') \ldots (\nu y_{k'} : N_{k'}')\ (P_5' \mid P_6')$. Using ($\pi_{esc}$ Red Var) and (π_{esc} Red Struct), we can derive $\sigma : P \longmapsto^+ (\nu x_1 : N_1) \ldots (\nu x_k : N_k)\ (\nu y_1 : N_1') \ldots (\nu y_{k'} : N_{k'}')\ ([n : \varepsilon] \mid P_3'' \mid (\nu x : M_2')\ P_4'')$. Finally, $\sigma : cl_\sigma(P) \longmapsto^+ P' = [n_1 : \varepsilon] \mid \ldots \mid [n_r : \varepsilon] \mid (\nu x_1 : N_1) \ldots (\nu x_k : N_k)\ (\nu y_1 : N_1') \ldots (\nu y_{k'} : N_{k'}')\ ([n : \varepsilon] \mid cl(P_3'') \mid (\nu x : M_2')\ cl(P_4''))$. Then,

$$\begin{aligned}
[\![P']\!] &\equiv ([\![cl(P_3'')]\!] \mid [\![cl(P_4'')]\!]\{^{M_2'}/_x\})\sigma_y\sigma_x \\
&\equiv ([\![P_3'']\!] \mid [\![P_4'']\!]\{^{M_2'}/_x\})\sigma_y\sigma_x \\
&\equiv P_3'\sigma_y\sigma_x \mid P_4'\{^{M_2'}/_x\}\sigma_y\sigma_x \\
&= P_3'\sigma_x\sigma_y \mid P_4'\sigma_y\sigma_x\{^{M_2'\sigma_x\sigma_y}/_x\} \\
&= P_3\sigma_y \mid P_4\sigma_x\{^{M_2\sigma_y}/_x\} \\
&= P_3 \mid P_4\{^{M_2}/_x\} \\
[\![P']\!]\sigma^* &\equiv P_3\sigma^* \mid P_4\sigma^*\{^{M_2\sigma^*}/_x\} \\
&= P_1 \mid P_2\{^{m}/_x\} \\
&= Q.
\end{aligned}$$

- $P_5' = \overline{M_1'}\langle M_2'\rangle.P_3''$ with $[\![P_3'']\!] = P_3'$, and $P_6' = [p : (x).P_4'']$ with $M_3' = p$ and $[\![P_4'']\!] = P_4'$. This case is very similar to the last one: we just need to exchange the roles of P_5 and P_6.
- $P_5' = \overline{M_1'}\langle M_2'\rangle.P_3''$ with $[\![P_3'']\!] = P_3'$, and $P_6' = M_3'(x).P_4''$ with $[\![P_4'']\!] = P_4'$. The reasoning is very similar, except that we have to introduce the channel n

explicitly. We will not give details of the side conditions. From $M_1'(\sigma_y' \uplus \sigma_x' \uplus \sigma)^* = n$ and Lemma 38, we derive $\sigma_y' \uplus \sigma_x' \uplus \sigma : P_5' \longmapsto \overline{n}\langle M_2'\rangle.P_3''$. From $M_3'(\sigma_y' \uplus \sigma_x' \uplus \sigma)^* = n$ and Lemma 38, we derive $\sigma_y' \uplus \sigma_x' \uplus \sigma : P_6' \longmapsto n(x).P_4''$. Then,

$$\begin{aligned}\sigma_y' \uplus \sigma_x' \uplus \sigma : [n : \varepsilon] \mid P_5' \mid P_6' &\longmapsto^* [n : \varepsilon] \mid \overline{n}\langle M_2'\rangle.P_3'' \mid n(x).P_4'' \\ &\longmapsto [n : \varepsilon \mid \langle M_2'\rangle.P_3''] \mid n(x).P_4'' \\ &\longmapsto [n : \varepsilon \mid \langle M_2'\rangle.P_3'' \mid (x).P_4''] \\ &\longmapsto [n : \varepsilon] \mid P_3'' \mid (\nu x : M_2')\ P_4''.\end{aligned}$$

Then $\sigma : [n : \varepsilon] \mid P \longmapsto^+ (\nu x_1 : N_1) \ldots (\nu x_k : N_k)\ (\nu y_1 : N_1') \ldots (\nu y_{k'} : N_{k'}')\ ([n : \varepsilon] \mid P_3'' \mid (\nu x : M_2')\ P_4'')$. Since $\vdash P : Valid$, we have $P_3'' \not\downarrow_1 n$ and $P_4'' \not\downarrow_1 n$. Consequently, $P \not\downarrow_1 n$ and $n \in (fn(P) \cup fn(\sigma)) \setminus pr(P)$. Finally, $\sigma : cl_\sigma(P) = [n_1 : \varepsilon] \mid \ldots \mid [n_r : \varepsilon] \mid [n : \varepsilon] \mid cl(P) \longmapsto^+ P' = [n_1 : \varepsilon] \mid \ldots \mid [n_r : \varepsilon] \mid (\nu x_1 : N_1) \ldots (\nu x_k : N_k)\ (\nu y_1 : N_1') \ldots (\nu y_{k'} : N_{k'}')\ ([n : \varepsilon] \mid cl(P_3'') \mid (\nu x : M_2')\ cl(P_4''))$. Checking the condition $[\![P]\!]\sigma^* \equiv Q$ is the same as above.

(π **Red Par)** Suppose that $[\![P]\!]\sigma^* = P_1 \mid Q \longrightarrow P_2 \mid Q$ was derived from $P_1 \longrightarrow P_2$, and that $\vdash P : Valid$ and $fv(P) \subseteq dom(\sigma)$. Necessarily, $[\![P]\!] = P_3 \mid Q_1$ with $P_3\sigma^* = P_1$ and $Q_1\sigma^* = Q$. By Lemma 43, there are two cases to consider:

— $P = [n : S \mid S']$ with $[\![S]\!]_n = P_3$ and $[\![S']\!]_n = Q_1$. Then $P_1 = [\![S]\!]_n\sigma^* = [\![[n : S]]\!]\sigma^*$ and $P_1 \longrightarrow P_2$. From $\vdash P : Valid$, we get $\vdash [n : S] : Valid$, and from $fv(P) \subseteq dom(\sigma)$, we get $fv([n : S]) \subseteq dom(\sigma)$. By the induction hypothesis, there is a process P'' such that $\sigma : cl_\sigma([n : S]) \longmapsto^+ P''$ and $[\![P'']\!]\sigma^* \equiv P_2$. By Lemma 61, since $\vdash P : Valid$ and $fv(S') \subseteq dom(\sigma)$, there is a process P' such that $\sigma : cl_\sigma(P) \longmapsto^+ P'$ and $[\![P']\!] \equiv [\![P'']\!] \mid [\![S']\!]_n = [\![P'']\!] \mid Q_1$. Finally, $[\![P']\!]\sigma^* \equiv [\![P'']\!]\sigma^* \mid Q_1\sigma^* \equiv P_2 \mid Q$.

— $P = P_4 \mid Q_2$ with $[\![P_4]\!] = P_3$ and $[\![Q_2]\!] = Q_1$. Then $P_1 = [\![P_4]\!]\sigma^* \longrightarrow P_2$. Since $\vdash P : Valid$, we get $\vdash P_4 : Valid$. By the induction hypothesis, there is a process P_4' such that $\sigma : cl_\sigma(P_4) \longmapsto^+ P_4'$ and $[\![P_4']\!]\sigma^* \equiv P_2$. By Lemma 60, there is P' such that $\sigma : cl_\sigma(P_4 \mid Q_2) \longmapsto^+ P'$ and $[\![P']\!] \equiv [\![P_4' \mid Q_2]\!]$. Finally, $\sigma : cl_\sigma(P) \longmapsto^+ P'$ and $[\![P']\!]\sigma^* \equiv [\![P_4']\!]\sigma^* \mid [\![Q_2]\!]\sigma^* \equiv P_2 \mid Q_1\sigma^* = P_2 \mid Q$.

(π **Red Res)** Suppose that $[\![P]\!]\sigma^* = (\nu n)\ P_0 \longrightarrow (\nu n)\ Q_0 = Q$ was derived from $P_0 \longrightarrow Q_0$, and that $\vdash P : Valid$ and $fv(P) \subseteq dom(\sigma)$. Necessarily, $P = (\nu n)\ P_1$ with $[\![P_1]\!]\sigma^* = P_0$ and $n \notin fn(\sigma)$. We easily get that $\vdash P_1 : Valid$. From $[\![P_1]\!]\sigma^* \longrightarrow Q_0$, we deduce by the induction hypothesis that there exists P_1' such that $\sigma : cl_\sigma(P_1) \longmapsto^+ P_1'$ and $[\![P_1']\!]\sigma^* \equiv Q_0$. We consider two cases:

— Suppose $n \in fn(P)$. We can derive $\sigma : (\nu n)\ cl_\sigma(P_1) \longmapsto^+ P' = (\nu n)\ P_1'$ by (π_{esc} Red Res). By Lemma 47, $(\nu n)\ cl_\sigma(P_1) \equiv cl_\sigma(P)$, so we can derive $\sigma : cl_\sigma(P) \longmapsto^+ P'$. Finally, $[\![P']\!]\sigma^* = (\nu n)\ [\![P_1']\!]\sigma^* \equiv (\nu n)\ Q_0 = Q$.

— Suppose $n \notin fn(P)$. We can derive $\sigma : (\nu n)\ ([n : \varepsilon] \mid cl_\sigma(P_1)) \longmapsto^+ P' = (\nu n)\ ([n : \varepsilon] \mid P_1')$ by (π_{esc} Red Par) and (π_{esc} Red Res). By Lemma 47, we have $(\nu n)\ ([n : \varepsilon] \mid cl_\sigma(P_1)) \equiv cl_\sigma(P)$, so we can derive $\sigma : cl_\sigma(P) \longmapsto^+ P'$. Finally, $[\![P']\!]\sigma^* \equiv (\nu n)\ [\![P_1']\!]\sigma^* \equiv (\nu n)\ Q_0 = Q$.

(π Red Struct) Suppose that $[\![P]\!]\sigma^* \longrightarrow Q$ was derived from $[\![P]\!]\sigma^* \equiv P_1$, $P_1 \longrightarrow Q_1$ and $Q_1 \equiv Q$, and that $\vdash P : Valid$ and $fv(P) \subseteq dom(\sigma)$. By Lemma 29, there exists P_2 such that $[\![P]\!] \equiv P_2$ and $P_1 = P_2\sigma^*$. Then, by Lemma 54, there exists P_2' such that $P_2 = [\![P_2']\!]$ and $P_2' \preccurlyeq P$. By Lemma 53, we have $cl(P) \equiv [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl(P_2')$ with $\{n_1, \ldots, n_k\} = pr(P) \setminus pr(P_2')$. Let $(fn(P_2') \cup fn(\sigma)) \setminus pr(P_2') = \{n_{i_1}, \ldots, n_{i_{k'}}, m_1, \ldots, m_l\}$ with $n_i \neq m_j$. Then we have:

$$\begin{aligned}
&(fn(P) \cup fn(\sigma)) \setminus pr(P)\\
&\quad \supseteq (fn(P_2') \cup fn(\sigma)) \setminus pr(P) \qquad \text{using Lemma 49}\\
&\quad = (fn(P_2') \cup fn(\sigma)) \setminus (pr(P_2') \cup \{n_1, \ldots, n_k\}) \qquad \text{using Corollary 51}\\
&\quad = ((fn(P_2') \cup fn(\sigma)) \setminus pr(P_2')) \setminus \{n_1, \ldots, n_k\}\\
&\quad = \{m_1, \ldots, m_l\}.
\end{aligned}$$

Let us write $(fn(P) \cup fn(\sigma)) \setminus pr(P) = \{m_1, \ldots, m_l, p_1, \ldots, p_{l'}\}$ (we have $p_j \neq n_i$ because $n_i \in pr(P)$ and $p_j \notin pr(P)$). We have:

$$\begin{aligned}
cl_\sigma(P) &\equiv [p_1 : \varepsilon] \mid \ldots \mid [p_{l'} : \varepsilon] \mid [m_1 : \varepsilon] \mid \ldots \mid [m_l : \varepsilon] \mid cl(P)\\
&\equiv [p_1 : \varepsilon] \mid \ldots \mid [p_{l'} : \varepsilon] \mid [m_1 : \varepsilon] \mid \ldots \mid [m_l : \varepsilon]\\
&\qquad \mid [n_1 : \varepsilon] \mid \ldots \mid [n_k : \varepsilon] \mid cl(P_2') \qquad \text{using Lemma 53}\\
&\equiv [p_1 : \varepsilon] \mid \ldots \mid [p_{l'} : \varepsilon] \mid [n_{j_1} : \varepsilon] \mid \ldots \mid [n_{j_{k-k'}} : \varepsilon] \mid cl_\sigma(P_2')
\end{aligned}$$

where $\{i_1, \ldots, i_{k'}\}$ and $\{j_1, \ldots, j_{k-k'}\}$ form a partition of $\{1, \ldots, k\}$. Since $\vdash P : Valid$, $\vdash P_2' : Valid$ by Lemma 52; and $fv(P_2') = fv(P) \subseteq dom(\sigma)$ by Lemma 49. By the induction hypothesis, there is a process P_3 such that $\sigma : P_1 = [\![P_2']\!]\sigma^* \longmapsto^+ P_3$ and $[\![P_3]\!]\sigma^* \equiv Q_1$. We can derive $\sigma : cl_\sigma(P) \longmapsto^+ P' = [p_1 : \varepsilon] \mid \ldots \mid [p_{l'} : \varepsilon] \mid [n_{j_1} : \varepsilon] \mid \ldots \mid [n_{j_{k-k'}} : \varepsilon] \mid P_3$. And we have $[\![P']\!]\sigma^* \equiv [\![P_3]\!]\sigma^* \equiv Q_1 \equiv Q$. □

A.8. *Observational equivalence*

This section gives the proof for the full adequacy result of Proposition 12.

A.8.1. *Preliminary lemmas*

Lemma 62. In the π-calculus,

— If $P \downarrow n$, then $n \in fn(P)$.
— If $P \downarrow x$, then $x \in fv(P)$.

Proof. The proof is by induction on the derivations of $P \downarrow n$ and $P \downarrow x$. □

Lemma 63. In the π-calculus, if $P \equiv Q$, then $P \downarrow M \Leftrightarrow Q \downarrow M$.

Proof. The proof is by induction on the derivation of $P \equiv Q$, using Lemma 62. □

Lemma 64. In the π-calculus,

1 If $P \downarrow M$, then $P\sigma \downarrow M\sigma$.
2 If $P\sigma \downarrow M$, then there exists M' such that $P \downarrow M'$ and $M'\sigma = M$.

Proof. The proof is by induction on the derivations of $P \downarrow M$ and $P\sigma \downarrow M$. $\square$

A.8.2. *Proof of Proposition 12*

Proposition 12. For a process P in the π_{esc}-calculus, $P \downarrow M \;\Leftrightarrow\; [\![P]\!] \downarrow M$.

Proof. We prove the two implications separately.

$\mathbf{P \downarrow M \;\Rightarrow\; [\![P]\!] \downarrow M}$ We use induction on the derivation of $P \downarrow M$:

- **(Obs Res)** Suppose that $(\nu n)\ P \downarrow M$ was derived from $P \downarrow M$ with $n \neq M$. By the induction hypothesis, $[\![P]\!] \downarrow M$. We can derive $(\nu n)\ [\![P]\!] \downarrow M$ by (Obs Res), that is to say, $[\![(\nu n)\ P]\!] \downarrow M$.
- **(Obs ParL)** Suppose that $P \mid Q \downarrow M$ was derived from $P \downarrow M$. By the induction hypothesis, $[\![P]\!] \downarrow M$. We can derive $[\![P]\!] \mid [\![Q]\!] \downarrow M$ by (Obs ParL), that is to say, $[\![P \mid Q]\!] \downarrow M$.
- **(Obs ParR)** This case is similar to (Obs ParL).
- **(Obs Repl)** Suppose that $!P \downarrow M$ was derived from $P \downarrow M$. By the induction, hypothesis, $[\![P]\!] \downarrow M$. We can derive $![\![P]\!] \downarrow M$ by (Obs Repl), that is to say, $[\![!P]\!] \downarrow M$.
- **(Obs Output)** Suppose that $\overline{M}\langle M'\rangle.P \downarrow M$. We can always derive $\overline{M}\langle M'\rangle.[\![P]\!] \downarrow M$ by (Obs Output), that is to say, $[\![\overline{M}\langle M'\rangle.P]\!] \downarrow M$.
- **(Obs Input)** This case is similar to (Obs Output).
- **(Obs Channel)** Suppose that $[n : S] \downarrow n$ was derived with the condition $S \not\equiv \varepsilon$. S must contain an abstraction of the form $\langle M\rangle.P$ or $(x).P$. Then $[\![[n : S]]\!] = [\![S]\!]_n$ must contain a process of the form $\overline{n}\langle M\rangle.[\![P]\!]$ or $n(x).[\![P]\!]$. We can derive $[\![[n : S]]\!] \downarrow n$ by one application of (Obs Output) or (Obs Input), and many applications of (Obs ParL) or (Obs ParR).
- **(Obs Var$_1$)** Suppose that $(\nu x : M')\ P \downarrow M$ was derived from $P \downarrow M$ with $x \neq M$. By the induction hypothesis, $[\![P]\!] \downarrow M$. By Lemma 64, $[\![P]\!]\{^{M'}/_x\} \downarrow M\{^{M'}/_x\}$, that is to say, $[\![(\nu x : M')\ P]\!] \downarrow M$ since $x \neq M$.
- **(Obs Var$_2$)** Suppose that $(\nu x : M)\ P \downarrow M$ was derived from $P \downarrow x$. By the induction hypothesis, $[\![P]\!] \downarrow x$. By Lemma 64, $[\![P]\!]\{^{M}/_x\} \downarrow x\{^{M}/_x\}$, that is to say, $[\![(\nu x : M)\ P]\!] \downarrow M$.

$\mathbf{[\![P]\!] \downarrow M \;\Rightarrow\; P \downarrow M}$ By the induction on the structure of P:

- Suppose that $[\![(\nu n)\ P]\!] \downarrow M$, that is to say, $(\nu n)\ [\![P]\!] \downarrow M$. This must have been derived from $[\![P]\!] \downarrow M$ by (Obs Res) with $n \neq M$. By the induction hypothesis, $P \downarrow M$, and we can derive $(\nu n)\ P \downarrow M$ by (Obs Res).
- It is impossible to derive $\mathbf{0} \downarrow M$ for any M. Thus, the case $[\![\mathbf{0}]\!] \downarrow M$ cannot happen.
- Suppose that $[\![P \mid Q]\!] \downarrow M$, that is to say, $[\![P]\!] \mid [\![Q]\!] \downarrow M$. This must have been derived from $[\![P]\!] \downarrow M$ or $[\![Q]\!] \downarrow M$ by (Obs ParL) or (Obs ParR). In the former case, $P \downarrow M$ by the induction hypothesis, and $P \mid Q \downarrow M$ by (Obs ParL). The latter case is similar.

— Suppose that $[\![!P]\!] \downarrow M$, that is to say, $![\![P]\!] \downarrow M$. This must have been derived from $[\![P]\!] \downarrow M$ by (Obs Repl). By the induction hypothesis, $P \downarrow M$, and we can derive $!P \downarrow M$ by (Obs Repl).

— Suppose that $[\![\overline{M'}\langle M''\rangle.P]\!] \downarrow M$, that is to say, $\overline{M'}\langle M''\rangle.[\![P]\!] \downarrow M$. This must have been derived by (Obs Output) and, necessarily, $M' = M$. Then we can derive $\overline{M'}\langle M''\rangle.P \downarrow M$ by (Obs Output).

— The case of $[\![M'(x).P]\!] \downarrow M$ is very similar to that of $[\![\overline{M'}\langle M''\rangle.P]\!] \downarrow M$.

— Suppose that $[\![[n : S]]\!] \downarrow M$, that is to say, $[\![S]\!]_n \downarrow M$. If $S \equiv \varepsilon$, this implies $\mathbf{0} \downarrow M$ by Lemma 63, which is impossible. Thus, we must have $S \not\equiv \varepsilon$. Since $[\![S]\!]_n \downarrow M$ must have been derived by applications of the rules (Obs ParL) and (Obs ParR), and one application of the rule (Obs Output) or (Obs Input), $[\![S]\!]_n$ must contain a process of the form $\overline{M}\langle M'\rangle.P$ or $M(x).P$, and, necessarily, $n = M$. Then $[n : S] \downarrow M$ by (Obs Channel).

— Suppose that $[\![(\nu x : M')\ P]\!] \downarrow M$, that is to say $[\![P]\!]\{^{M'}/_{x}\} \downarrow M$. Note that we can always suppose $x \neq M$. By Lemma 64, there is M'' such that $[\![P]\!] \downarrow M''$ and $M''\{^{M'}/_{x}\} = M$. By the induction hypothesis, $P \downarrow M''$. There are two cases to consider. If $M'' = M$, we can derive $(\nu x : M')\ P \downarrow M$ by (Obs Var_1) since $M \neq x$. Otherwise, we must have $M'' = x$ and $M' = M$. Then we can derive $(\nu x : M')\ P \downarrow M$ by (Obs Var_2). □

A.9. *Encoding in pure ambients*

This section gives the correctness proofs for the encoding of π_{esc} into pure ambients. These are facilitated by the similarity between the π_{esc}-calculus and the encoding mechanism.

Lemma 65. $fn(P) = fn(\{\!\!\{P\}\!\!\}) \setminus \{read, write, enter\}$, and $fv(P) = fv(\{\!\!\{P\}\!\!\})$ (we always implicitly suppose that P does not contain the special names *read*, *write* and *enter*).

Proof. The proof is by induction on the structure of P. □

Lemma 66. If $P \equiv Q$, then $\{\!\!\{P\}\!\!\} \equiv \{\!\!\{Q\}\!\!\}$.

Proof. The proof is by induction on the derivation of $P \equiv Q$, using Lemma 65. □

A reduction in π_{esc} is simulated by exactly one principal reduction and many auxiliary reductions in pure ambients.

Proposition 18. If $\sigma : P \longmapsto Q$, then $\{\!\!\{\sigma, P\}\!\!\} \overset{pr}{\hookrightarrow} \overset{aux^*}{\hookrightarrow} \{\!\!\{\sigma, Q\}\!\!\}$.

Proof. We use induction on the derivation of $\sigma : P \longmapsto Q$:

(π_{esc} Red Subst Out) See the main text.
(π_{esc} Red Output) Suppose that $\sigma : [n : S] \mid \overline{n}\langle M\rangle.P \longmapsto [n : S \mid \langle M\rangle.P]$. If we name $p_1, \ldots, p_k$ the fresh names in S, and if we choose them as well as p to avoid interferences,

we have:

$$
\{\![n:S] \mid \overline{n}\langle M\rangle.P\}\!\} \;=\; \left\{
\begin{array}{l}
(\nu p_1)\ldots(\nu p_k)\\
\quad (\; n \quad [\; allowIO\ n\\
\qquad\qquad \mid server\ read\ \ldots.\\
\qquad\qquad \mid \{\!\{S\}\!\}_n]\\
\quad \mid open\ p_1\ \mid\ldots\mid\ open\ p_k\ )\\
\mid (\nu p)\ \ (\; write\ [\; request\ write\ n\\
\qquad\qquad\qquad\qquad \mid fwd\ M\\
\qquad\qquad\qquad\qquad \mid p[\; out\ read\ .\ \overline{open}\ p\ .\{\!\{P\}\!\}]]\\
\qquad\qquad \mid open\ p\ )
\end{array}\right.
$$

$$
\overset{pr}{\hookrightarrow} \left\{
\begin{array}{l}
(\nu p_1)\ldots(\nu p_k)\ (\nu p)\\
(\; n\ \ [\; allowIO\ n\\
\qquad \mid server\ read\ \ldots.\\
\qquad \mid \{\!\{S\}\!\}_n\\
\qquad \mid write\ \ [\; \overline{in}\ write\ .\ open\ enter\\
\qquad\qquad\qquad \mid fwd\ M\\
\qquad\qquad\qquad \mid p[\; out\ read\ .\ \overline{open}\ p\ .\{\!\{P\}\!\}]]]\\
\mid open\ p_1\ \mid\ldots\mid\ open\ p_k\ \mid\ open\ p\ )
\end{array}\right.
$$

$$
= \;\{\!\{[n:S \mid \langle M\rangle.P]\}\!\}
$$

Then we can easily derive $\{\!\{\sigma,[n:S] \mid \overline{n}\langle M\rangle.P\}\!\} \overset{pr}{\hookrightarrow} \overset{aux^*}{\hookrightarrow} \{\!\{\sigma,[n:S \mid \langle M\rangle.P]\}\!\}$ for any σ.

(π_{esc} Red Input) Suppose that $\sigma : [n:S] \mid n(x).P \;\longmapsto\; [n:S \mid (x).P]$. If we name $p_1,\ldots,p_k$ the fresh names in S, and if we choose them as well as p to avoid interferences, we have:

$$
\{\!\{[n:S] \mid n(x).P\}\!\}
$$

$$
= \left\{
\begin{array}{l}
(\nu p_1)\ldots(\nu p_k)\\
\qquad (\; n \quad [\; allowIO\ n\\
\qquad\qquad\quad \mid server\ read\ .\ (\nu q)\\
\qquad\qquad\quad\ \ (\; \overline{out}\ read\ .\ read\ be\ q\ .\ \overline{in}\ q\ .\ out\ n\ .\ q\ be\ read\\
\qquad\qquad\quad\ \ \mid enter[\quad out\ read\ .\ in\ write\ .\ \overline{open}\ enter\ .\\
\qquad\qquad\qquad\qquad\quad in\ q\ .\ \overline{open}\ write\ ])\\
\qquad\qquad\quad \mid \{\!\{S\}\!\}_n]\\
\qquad \mid open\ p_1\ \mid\ldots\mid\ open\ p_k\ )\\
\mid (\nu p)\ \ (\; read\ \ [\; request\ read\ n\\
\qquad\qquad\quad \mid open\ write\ .\ \overline{out}\ read\ .\ (\nu x)\ \ read\ be\ x\ .\\
\qquad\qquad\quad\ \ (\; \overline{out}\ x\ .\ allowIO\ x\\
\qquad\qquad\quad\ \ \mid p[\; out\ x\ .\ \overline{open}\ p\ .\{\!\{P\}\!\}])]\\
\qquad \mid open\ p\ )
\end{array}\right.
$$

$$
\stackrel{pr}{\hookrightarrow}
\left\{
\begin{array}{l}
(\nu p_1) \ldots (\nu p_k)\ (\nu p) \\
(\ n\ \ [\ allowIO\ n \\
\quad\ \mid enter[\ in\ read\ .\ \overline{open}\ enter\ .\ (\nu q) \\
\qquad\qquad (\ \overline{out}\ read\ .\ read\ be\ q\ .\ \overline{in}\ q\ .\ out\ n\ .\ q\ be\ read \\
\qquad\qquad \mid enter[\ \ out\ read\ .\ in\ write\ .\ \overline{open}\ enter\ .\ in\ q\ . \\
\qquad\qquad\qquad\quad \overline{open}\ write\ ])] \\
\quad\ \mid server\ read\ \ldots. \\
\quad\ \mid \{\!\!\{ S \}\!\!\}_n \\
\quad\ \mid read\ \ [\ \overline{in}\ read\ .\ open\ enter \\
\qquad\qquad \mid open\ write\ .\ \overline{out}\ read\ .\ (\nu x)\ \ read\ be\ x\ . \\
\qquad\qquad\qquad (\ \overline{out}\ x\ .\ allowIO\ x \\
\qquad\qquad\qquad \mid p[\ out\ x\ .\ \overline{open}\ p\ .\{\!\!\{ P \}\!\!\}])]] \\
\mid open\ p_1\ \ \mid \ldots \mid\ \ open\ p_k\ \ \mid\ \ open\ p\ )
\end{array}
\right.
$$

$$
\stackrel{aux}{\hookrightarrow}
\left\{
\begin{array}{l}
(\nu p_1) \ldots (\nu p_k)\ (\nu p) \\
(\ n\ \ [\ allowIO\ n \\
\quad\ \mid server\ read\ \ldots. \\
\quad\ \mid \{\!\!\{ S \}\!\!\}_n \\
\quad\ \mid read\ \ [\ open\ enter \\
\qquad\qquad \mid open\ write\ .\ \overline{out}\ read\ .\ (\nu x)\ \ read\ be\ x\ . \\
\qquad\qquad\ \ (\ \overline{out}\ x\ .\ allowIO\ x \\
\qquad\qquad\ \ \mid p[\ out\ x\ .\ \overline{open}\ p\ .\{\!\!\{ P \}\!\!\}]) \\
\qquad\qquad \mid enter[\ \overline{open}\ enter\ .\ (\nu q) \\
\qquad\qquad\ \ (\ \overline{out}\ read\ .\ read\ be\ q\ .\ \overline{in}\ q\ .\ out\ n\ .\ q\ be\ read \\
\qquad\qquad\ \ \mid enter[\ \ out\ read\ .\ in\ write\ .\ \overline{open}\ enter\ .\ in\ q\ . \\
\qquad\qquad\qquad\quad\ \ \overline{open}\ write\ ])]]] \\
\mid open\ p_1\ \ \mid \ldots \mid\ \ open\ p_k\ \ \mid\ \ open\ p\ )
\end{array}
\right.
$$

$$
\stackrel{aux}{\hookrightarrow}
\left\{
\begin{array}{l}
(\nu p_1) \ldots (\nu p_k)\ (\nu p) \\
(\ n\ \ [\ allowIO\ n \\
\quad\ \mid server\ read\ \ldots. \\
\quad\ \mid \{\!\!\{ S \}\!\!\}_n \\
\quad\ \mid read\ \ [\ open\ write\ .\ \overline{out}\ read\ .\ (\nu x)\ \ read\ be\ x\ . \\
\qquad\qquad\ \ (\ \overline{out}\ x\ .\ allowIO\ x \\
\qquad\qquad\ \ \mid p[\ out\ x\ .\ \overline{open}\ p\ .\{\!\!\{ P \}\!\!\}]) \\
\qquad\qquad \mid (\nu q) \\
\qquad\qquad\ \ (\ \overline{out}\ read\ .\ read\ be\ q\ .\ \overline{in}\ q\ .\ out\ n\ .\ q\ be\ read \\
\qquad\qquad\ \ \mid enter[\ \ out\ read\ .\ in\ write\ .\ \overline{open}\ enter\ .\ in\ q\ . \\
\qquad\qquad\qquad\quad\ \ \overline{open}\ write\ ])]] \\
\mid open\ p_1\ \ \mid \ldots \mid\ \ open\ p_k\ \ \mid\ \ open\ p\ )
\end{array}
\right.
$$

$$\overset{aux}{\hookrightarrow} \left\{ \begin{array}{l} (\nu p_1) \ldots (\nu p_k)\ (\nu p) \\ (\ n\ \ [\ \mathit{allowIO}\ n \\ \quad\ \mid \mathit{server\ read} \ldots. \\ \quad\ \mid \{\!\{S\}\!\}_n \\ \quad\ \mid (\nu q) \quad (\ \mathit{read}\ \ [\ \mathit{open\ write}\ .\ \overline{\mathit{out}}\ \mathit{read}\ .\ (\nu x)\ \ \mathit{read\ be}\ x\ . \\ \qquad\qquad\qquad\qquad\qquad (\ \overline{\mathit{out}}\ x\ .\ \mathit{allowIO}\ x \\ \qquad\qquad\qquad\qquad\qquad \mid p[\ \mathit{out}\ x\ .\ \overline{\mathit{open}}\ p\ .\{\!\{P\}\!\}]) \\ \qquad\qquad\qquad\qquad\quad \mid \mathit{read\ be}\ q\ .\ \overline{\mathit{in}}\ q\ .\ \mathit{out}\ n\ .\ q\ \mathit{be\ read}\ ] \\ \qquad\qquad\quad \mid \mathit{enter}[\ \mathit{in\ write}\ .\ \overline{\mathit{open}}\ \mathit{enter}\ .\ \mathit{in}\ q\ .\ \overline{\mathit{open}}\ \mathit{write}\ ])] \\ \mid \mathit{open}\ p_1\ \ \mid \ldots \mid\ \ \mathit{open}\ p_k\ \ \mid\ \ \mathit{open}\ p\ ) \end{array} \right.$$

$$\overset{aux^*}{\hookrightarrow} \left\{ \begin{array}{l} (\nu p_1) \ldots (\nu p_k)\ (\nu p) \\ (\ n\ \ [\ \mathit{allowIO}\ n \\ \quad\ \mid \mathit{server\ read} \ldots. \\ \quad\ \mid \{\!\{S\}\!\}_n \\ \quad\ \mid (\nu q) \quad (\ q\ \ [\ \mathit{open\ write}\ .\ \overline{\mathit{out}}\ \mathit{read}\ .\ (\nu x)\ \ \mathit{read\ be}\ x\ . \\ \qquad\qquad\qquad\qquad (\ \overline{\mathit{out}}\ x\ .\ \mathit{allowIO}\ x \\ \qquad\qquad\qquad\qquad \mid p[\ \mathit{out}\ x\ .\ \overline{\mathit{open}}\ p\ .\{\!\{P\}\!\}]) \\ \qquad\qquad\qquad\quad \mid \overline{\mathit{in}}\ q\ .\ \mathit{out}\ n\ .\ q\ \mathit{be\ read}\ ] \\ \qquad\qquad\quad \mid \mathit{enter}[\ \mathit{in\ write}\ .\ \overline{\mathit{open}}\ \mathit{enter}\ .\ \mathit{in}\ q\ .\ \overline{\mathit{open}}\ \mathit{write}\ ])] \\ \mid \mathit{open}\ p_1\ \ \mid \ldots \mid\ \ \mathit{open}\ p_k\ \ \mid\ \ \mathit{open}\ p\ ) \end{array} \right.$$

$$\equiv \quad \{\!\{[n : S \mid (x).P]\}\!\}$$

Then we can easily derive $\{\!\{\sigma, [n : S] \mid n(x).P\}\!\} \overset{pr}{\hookrightarrow} \overset{aux^*}{\hookrightarrow} \{\!\{\sigma, [n : S \mid (x).P]\}\!\}$ for any σ.

(π_{esc} Red Comm) Suppose that $\sigma : [n : S \mid \langle M\rangle.P \mid (x).Q] \longmapsto [n : S] \mid P \mid (\nu x : M)\ Q$ with $x \neq M$. If we name $p_1, \ldots, p_k$ the fresh names in S, and if we choose them as well as r_1 and r_2 to avoid interferences, we have:

$$\{\!\{[n : S \mid \langle M\rangle.P \mid (x).Q]\}\!\}$$

$$\equiv \left\{ \begin{array}{l} (\nu p_1) \ldots (\nu p_k)\ (\nu r_1)\ (\nu r_2) \\ (\ n\ \ [\ \mathit{allowIO}\ n \\ \quad\ \mid \mathit{server\ read} \ldots. \\ \quad\ \mid \{\!\{S\}\!\}_n \\ \quad\ \mid \mathit{write}\ \ [\ \overline{\mathit{in}}\ \mathit{write}\ .\ \mathit{open\ enter} \\ \qquad\qquad\quad \mid \mathit{fwd}\ M \\ \qquad\qquad\quad \mid r_1[\ \mathit{out\ read}\ .\ \overline{\mathit{open}}\ r_1\ .\{\!\{P\}\!\}]] \\ \quad\ \mid (\nu q) \quad (\ q\ \ [\ \mathit{open\ write}\ .\ \overline{\mathit{out}}\ \mathit{read}\ .\ (\nu x)\ \ \mathit{read\ be}\ x\ . \\ \qquad\qquad\qquad\qquad (\ \overline{\mathit{out}}\ x\ .\ \mathit{allowIO}\ x \\ \qquad\qquad\qquad\qquad \mid r_2[\ \mathit{out}\ x\ .\ \overline{\mathit{open}}\ r_2\ .\{\!\{Q\}\!\}]) \\ \qquad\qquad\qquad\quad \mid \overline{\mathit{in}}\ q\ .\ \mathit{out}\ n\ .\ q\ \mathit{be\ read}\ ] \\ \qquad\qquad\quad \mid \mathit{enter}[\ \mathit{in\ write}\ .\ \overline{\mathit{open}}\ \mathit{enter}\ .\ \mathit{in}\ q\ .\ \overline{\mathit{open}}\ \mathit{write}\ ])] \\ \mid \mathit{open}\ p_1\ \ \mid \ldots \mid\ \ \mathit{open}\ p_k\ \ \mid\ \ \mathit{open}\ r_1\ \ \mid\ \ \mathit{open}\ r_2\ ) \end{array} \right.$$

$$
\overset{pr}{\hookrightarrow}
\left\{
\begin{array}{l}
(\nu p_1) \ldots (\nu p_k)\ (\nu r_1)\ (\nu r_2) \\
(\ n\ \ [\ allowIO\ n \\
\qquad |\ server\ read\ \ldots. \\
\qquad |\ \{\!\!\{S\}\!\!\}_n \\
\qquad |\ (\nu q)\quad (\ write\ \ [\ open\ enter \\
\qquad\qquad\qquad\qquad\qquad |\ fwd\ M \\
\qquad\qquad\qquad\qquad\qquad |\ r_1[\ out\ read\ .\ \overline{open}\ r_1\ .\{\!\!\{P\}\!\!\}] \\
\qquad\qquad\qquad\qquad\qquad |\ enter[\ \overline{open}\ enter\ .\ in\ q\ .\ \overline{open}\ write\ ]] \\
\qquad\qquad\qquad |\ q \qquad [\ open\ write\ .\ \overline{out}\ read\ .\ (\nu x)\ \ read\ be\ x\ . \\
\qquad\qquad\qquad\qquad\qquad\ \ (\ \overline{out}\ x\ .\ allowIO\ x \\
\qquad\qquad\qquad\qquad\qquad\ \ |\ r_2[\ out\ x\ .\ \overline{open}\ r_2\ .\{\!\!\{Q\}\!\!\}]) \\
\qquad\qquad\qquad\qquad\qquad |\ \overline{in}\ q\ .\ out\ n\ .\ q\ be\ read\ ])] \\
|\ open\ p_1\ \ |\ \ldots\ |\ \ open\ p_k\ \ |\ \ open\ r_1\ \ |\ \ open\ r_2\ )
\end{array}
\right.
$$

$$
\overset{aux}{\hookrightarrow}
\left\{
\begin{array}{l}
(\nu p_1) \ldots (\nu p_k)\ (\nu r_1)\ (\nu r_2) \\
(\ n\ \ [\ allowIO\ n \\
\qquad |\ server\ read\ \ldots. \\
\qquad |\ \{\!\!\{S\}\!\!\}_n \\
\qquad |\ (\nu q)\quad (\ write\ \ [\ fwd\ M \\
\qquad\qquad\qquad\qquad\qquad |\ r_1[\ out\ read\ .\ \overline{open}\ r_1\ .\{\!\!\{P\}\!\!\}] \\
\qquad\qquad\qquad\qquad\qquad |\ in\ q\ .\ \overline{open}\ write\ ] \\
\qquad\qquad\qquad |\ q \qquad [\ open\ write\ .\ \overline{out}\ read\ .\ (\nu x)\ \ read\ be\ x\ . \\
\qquad\qquad\qquad\qquad\qquad\ \ (\ \overline{out}\ x\ .\ allowIO\ x \\
\qquad\qquad\qquad\qquad\qquad\ \ |\ r_2[\ out\ x\ .\ \overline{open}\ r_2\ .\{\!\!\{Q\}\!\!\}]) \\
\qquad\qquad\qquad\qquad\qquad |\ \overline{in}\ q\ .\ out\ n\ .\ q\ be\ read\ ])] \\
|\ open\ p_1\ \ |\ \ldots\ |\ \ open\ p_k\ \ |\ \ open\ r_1\ \ |\ \ open\ r_2\ )
\end{array}
\right.
$$

$$
\overset{aux}{\hookrightarrow}
\left\{
\begin{array}{l}
(\nu p_1) \ldots (\nu p_k)\ (\nu r_1)\ (\nu r_2) \\
(\ n\ \ [\ allowIO\ n \\
\qquad |\ server\ read\ \ldots. \\
\qquad |\ \{\!\!\{S\}\!\!\}_n \\
\qquad |\ (\nu q)\ q\ \ [\ open\ write\ .\ \overline{out}\ read\ .\ (\nu x)\ \ read\ be\ x\ . \\
\qquad\qquad\qquad\qquad\qquad\ \ (\ \overline{out}\ x\ .\ allowIO\ x \\
\qquad\qquad\qquad\qquad\qquad\ \ |\ r_2[\ out\ x\ .\ \overline{open}\ r_2\ .\{\!\!\{Q\}\!\!\}]) \\
\qquad\qquad\qquad\quad |\ out\ n\ .\ q\ be\ read \\
\qquad\qquad\qquad\quad |\ write\ \ [\ fwd\ M \\
\qquad\qquad\qquad\qquad\qquad\ \ |\ r_1[\ out\ read\ .\ \overline{open}\ r_1\ .\{\!\!\{P\}\!\!\}] \\
\qquad\qquad\qquad\qquad\qquad\ \ |\ \overline{open}\ write\ ]]] \\
|\ open\ p_1\ \ |\ \ldots\ |\ \ open\ p_k\ \ |\ \ open\ r_1\ \ |\ \ open\ r_2\ )
\end{array}
\right.
$$

$$
\overset{aux}{\hookrightarrow}
\left\{
\begin{array}{l}
(\nu p_1) \ldots (\nu p_k)\ (\nu r_1)\ (\nu r_2) \\
(\ n\ \ [\ \mathit{allowIO}\ n \\
\qquad |\ \mathit{server\ read}\ \ldots. \\
\qquad |\ \{\!\!\{S\}\!\!\}_n \\
\qquad |\ (\nu q)\ q\ \ [\ \overline{\mathit{out}}\ \mathit{read}\ .\ (\nu x)\ \ \mathit{read\ be}\ x\ . \\
\qquad\qquad\qquad (\ \overline{\mathit{out}}\ x\ .\ \mathit{allowIO}\ x \\
\qquad\qquad\qquad |\ r_2[\ \mathit{out}\ x\ .\ \overline{\mathit{open}}\ r_2\ .\{\!\!\{Q\}\!\!\}]) \\
\qquad\qquad\quad |\ \mathit{out}\ n\ .\ q\ \mathit{be\ read} \\
\qquad\qquad\quad |\ \mathit{fwd}\ M \\
\qquad\qquad\quad |\ r_1[\ \mathit{out\ read}\ .\ \overline{\mathit{open}}\ r_1\ .\{\!\!\{P\}\!\!\}]]] \\
|\ \mathit{open}\ p_1\ \ |\ \ldots\ |\ \ \mathit{open}\ p_k\ \ |\ \ \mathit{open}\ r_1\ \ |\ \ \mathit{open}\ r_2\ )
\end{array}
\right.
$$

$$
\overset{aux}{\hookrightarrow}
\left\{
\begin{array}{l}
(\nu p_1) \ldots (\nu p_k)\ (\nu r_1)\ (\nu r_2) \\
(\ n\qquad [\ \mathit{allowIO}\ n \\
\qquad\quad |\ \mathit{server\ read}\ \ldots. \\
\qquad\quad |\ \{\!\!\{S\}\!\!\}_n] \\
|\ (\nu q)\ q\ \ [\ \overline{\mathit{out}}\ \mathit{read}\ .\ (\nu x)\ \ \mathit{read\ be}\ x\ . \\
\qquad\qquad (\ \overline{\mathit{out}}\ x\ .\ \mathit{allowIO}\ x \\
\qquad\qquad |\ r_2[\ \mathit{out}\ x\ .\ \overline{\mathit{open}}\ r_2\ .\{\!\!\{Q\}\!\!\}]) \\
\qquad\quad |\ q\ \mathit{be\ read} \\
\qquad\quad |\ \mathit{fwd}\ M \\
\qquad\quad |\ r_1[\ \mathit{out\ read}\ .\ \overline{\mathit{open}}\ r_1\ .\{\!\!\{P\}\!\!\}]] \\
|\ \mathit{open}\ p_1\ \ |\ \ldots\ |\ \ \mathit{open}\ p_k\ \ |\ \ \mathit{open}\ r_1\ \ |\ \ \mathit{open}\ r_2\ )
\end{array}
\right.
$$

$$
\overset{aux^*}{\hookrightarrow}
\left\{
\begin{array}{l}
(\nu p_1) \ldots (\nu p_k)\ (\nu r_1)\ (\nu r_2) \\
(\ n\qquad [\ \mathit{allowIO}\ n \\
\qquad\quad |\ \mathit{server\ read}\ \ldots. \\
\qquad\quad |\ \{\!\!\{S\}\!\!\}_n] \\
|\ \mathit{read}\ \ [\ \overline{\mathit{out}}\ \mathit{read}\ .\ (\nu x)\ \ \mathit{read\ be}\ x\ . \\
\qquad\qquad (\ \overline{\mathit{out}}\ x\ .\ \mathit{allowIO}\ x \\
\qquad\qquad |\ r_2[\ \mathit{out}\ x\ .\ \overline{\mathit{open}}\ r_2\ .\{\!\!\{Q\}\!\!\}]) \\
\qquad\quad |\ \mathit{fwd}\ M \\
\qquad\quad |\ r_1[\ \mathit{out\ read}\ .\ \overline{\mathit{open}}\ r_1\ .\{\!\!\{P\}\!\!\}]] \\
|\ \mathit{open}\ p_1\ \ |\ \ldots\ |\ \ \mathit{open}\ p_k\ \ |\ \ \mathit{open}\ r_1\ \ |\ \ \mathit{open}\ r_2\ )
\end{array}
\right.
$$

$$
\overset{aux}{\hookrightarrow}
\left\{
\begin{array}{l}
(\nu p_1) \ldots (\nu p_k)\ (\nu r_1)\ (\nu r_2) \\
(\ n\qquad [\ \mathit{allowIO}\ n \\
\qquad\quad |\ \mathit{server\ read}\ \ldots. \\
\qquad\quad |\ \{\!\!\{S\}\!\!\}_n] \\
|\ \mathit{read}\ \ [\ (\nu x)\ \ \mathit{read\ be}\ x\ . \\
\qquad\qquad (\ \overline{\mathit{out}}\ x\ .\ \mathit{allowIO}\ x \\
\qquad\qquad |\ r_2[\ \mathit{out}\ x\ .\ \overline{\mathit{open}}\ r_2\ .\{\!\!\{Q\}\!\!\}]) \\
\qquad\quad |\ \mathit{fwd}\ M\ ] \\
|\ r_1[\ \overline{\mathit{open}}\ r_1\ .\{\!\!\{P\}\!\!\}] \\
|\ \mathit{open}\ p_1\ \ |\ \ldots\ |\ \ \mathit{open}\ p_k\ \ |\ \ \mathit{open}\ r_1\ \ |\ \ \mathit{open}\ r_2\ )
\end{array}
\right.
$$

$$
\stackrel{aux^*}{\hookrightarrow}
\left\{
\begin{array}{lll}
(\nu p_1) \ldots (\nu p_k)\ (\nu r_1)\ (\nu r_2) & & \\
(\ n & [\ allowIO\ n & \\
 & \mid server\ read\ \ldots. & \\
 & \mid \{\!\{S\}\!\}_n] & \\
\mid (\nu x)\ x & [\ \overline{out}\ x\ .\ allowIO\ x & \\
 & \mid r_2[\ out\ x\ .\ \overline{open}\ r_2\ .\{\!\{Q\}\!\}] & \\
 & \mid fwd\ M\ ] & \\
\multicolumn{3}{l}{\mid r_1[\ \overline{open}\ r_1\ .\{\!\{P\}\!\}]} \\
\multicolumn{3}{l}{\mid open\ p_1\ \ \mid \ldots \mid\ \ open\ p_k\ \ \mid\ \ open\ r_1\ \ \mid\ \ open\ r_2\ )}
\end{array}
\right.
$$

$$
\stackrel{aux}{\hookrightarrow}
\left\{
\begin{array}{ll}
(\nu p_1) \ldots (\nu p_k)\ (\nu r_1)\ (\nu r_2) & \\
(\ n & [\ allowIO\ n \\
 & \mid server\ read\ \ldots. \\
 & \mid \{\!\{S\}\!\}_n] \\
\mid (\nu x) & (\ x[\ allowIO\ x\ \ \mid\ \ fwd\ M\ ] \\
 & \mid r_2[\ \overline{open}\ r_2\ .\{\!\{Q\}\!\}]) \\
\multicolumn{2}{l}{\mid r_1[\ \overline{open}\ r_1\ .\{\!\{P\}\!\}]} \\
\multicolumn{2}{l}{\mid open\ p_1\ \ \mid \ldots \mid\ \ open\ p_k\ \ \mid\ \ open\ r_1\ \ \mid\ \ open\ r_2\ )}
\end{array}
\right.
$$

$$
\stackrel{aux^*}{\hookrightarrow}
\left\{
\begin{array}{l}
(\nu p_1) \ldots (\nu p_k) \\
\quad (\ n\ [\ allowIO\ n \\
\qquad\quad \mid server\ read\ \ldots. \\
\qquad\quad \mid \{\!\{S\}\!\}_n] \\
\quad \mid open\ p_1\ \ \mid \ldots \mid\ \ open\ p_k\ ) \\
\mid (\nu x)\ (\ x[\ allowIO\ x\ \ \mid\ \ fwd\ M\ ] \mid \{\!\{Q\}\!\}) \\
\mid \{\!\{P\}\!\}
\end{array}
\right.
$$

$$
\equiv \quad \{\!\{[n : S] \mid P \mid (\nu x : M)\ Q\}\!\}
$$

Then we can easily derive $\{\!\{\sigma, [n : S \mid \langle M\rangle.P \mid (x).Q]\}\!\} \stackrel{pr}{\hookrightarrow} \stackrel{aux^*}{\hookrightarrow} \{\!\{\sigma, [n : S] \mid P \mid (\nu x : M)\ Q\}\!\}$ for any σ.

The other cases are similar or trivial. □

Acknowledgements

I have benefitted from many discussions with Davide Sangiorgi about this work. Thanks are also due to Ilaria Castellani, Gérard Boudol, Kohei Honda, Xudong Guan and the anonymous referees.

References

Cardelli, L. (1999a) Abstractions for Mobile Computation. In: Vitek, J. and Jensen, C. (eds.) Secure Internet Programming: Security Issues for Distributed and Mobile Objects. *Springer-Verlag Lecture Notes in Computer Science* **1603** 51–94.

Cardelli, L. (1999b) Wide Area Computation. In: Proceedings of ICALP'99. *Springer-Verlag Lecture Notes in Computer Science* **1644** 10–24.

Cardelli, L. and Gordon, A. D. (1997) A Calculus of mobile Ambients. Slides.

Cardelli, L. and Gordon, A. D. (1998) Mobile Ambients. In: Proceedings FoSSaCS'98. *Springer-Verlag Lecture Notes in Computer Science* **1378** 140–155.

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

Hirschkoff, D. (1999) Handling Substitutions Explicitly in the π-Calculus. In: *Proceedings of the Floc Workshop WESTAPP 99*.

Honda, K. and Yoshida, N. (1995) On Reduction-Based Process Semantics. *Theoretical Computer Science* **152** 437–486.

Levi, F. and Sangiorgi, D. (2000) Controlling Interference in Ambients. In: *Proceedings of POPL'00*, ACM Press.

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

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

Milner, R. (1991) The Polyadic π-Calculus: a Tutorial. Technical Report ECS-LFCS-91-180, University of Edinburgh.

Palamidessi, C. (1997) Comparing the Expressive Power of the Synchronous and the Asynchronous π-Calculus. In: *Proceedings of POPL'97*, ACM 256–265.

Zimmer, P. (2000) On the Expressiveness of Pure Mobile Ambients. In: 7^{th} International Workshop on Expressiveness in Concurrency. *Electronic Notes in Theoretical Science* **39**, Elsevier.