On asynchrony in name-passing calculi*

Massimo Merro
EPFL, Lausanne, Switzerland

Davide Sangiorgi
INRIA, Sophia-Antipolis, France

September 1, 2002

Abstract

The asynchronous π-calculus is considered the basis of experimental programming languages (or proposal of programming languages) like Pict, Join, and TyCO. However, at a closer inspection, these languages are based on an even simpler calculus, called *Localised* π (Lπ), where: (a) only the *output capability* of names may be transmitted; (b) there is no *matching* or similar constructs for testing equality between names.

We study the basic operational and algebraic theory of Lπ. We focus on bisimulation-based behavioural equivalences, precisely on *barbed congruence*. We prove two coinductive characterisations of barbed congruence in Lπ, and some basic algebraic laws. We then show applications of this theory, including: the derivability of the *delayed input*; the correctness of an optimisation of the encoding of call-by-name λ-calculus; the validity of some laws for Join; the soundness of Thielecke's axiomatic semantics of the *Continuation Passing Style calculus*.

*An extended abstract has appeared in the *Proceedings of 25th International Colloquium on Automata, Languages and Programming*, volume 1443, of *Lecture Notes in Computer Science*, Springer Verlag, July 1998.

Contents

1 Introduction

The *asynchronous π-calculus*, abbreviated $\pi_{\rm a}$, is a variant of the π-calculus [39], where message emission is non-blocking. Formally, the output prefix $\overline{a}b.\,P$ of the π-calculus is replaced with the simpler output particle $\overline{a}b$, which has no continuation. The asynchronous π-calculus has been introduced by Honda and Tokoro [23], and independently Boudol [9], who showed that it is expressive enough to encode the (synchronous) π-calculus. Asynchronous communications are interesting from the point of view of concurrent and distributed programming languages, because they are easier to implement and they are closer to the communication primitives offered by available distributed systems.

The asynchronous π-calculus is considered the basis of experimental concurrent and/or distributed programming languages (or proposal of programming languages) like Pict [47], Join [16], and TyCO [63]. However, at a closer inspection, the programming languages above are based on an even simpler calculus, where:

(a) the recipient of a name may only use it in output actions; that is, only the *output capability* of names may be transmitted;

(b) there is no *matching* construct (or similar constructs like mismatching) for testing equality between channel names. [1]

These restrictions are explicit in Join. In Pict and TyCO, (b) is explicit; (a) is not, but most programs comply with it. We call *Localised* π, abbreviated Lπ, the asynchronous π-calculus with the additional simplifications (a) and (b).

By restriction (a), the recipients of a channel are *local* to the process that has created the channel. More precisely, in a process $(\boldsymbol{\nu} a)P$ the inputs at channel a are statically determined: no further inputs at a may be created, inside or outside P. For instance, the process

$$(\boldsymbol{\nu} a)(\overline{b}a.\,P \mid a(x).\,Q) \;\mid\; b(z).\,z(y).\,R$$

is not in Lπ because, after a reduction along b, a new input at a is created. *Locality* of channels makes Lπ particularly suitable for giving semantics to, and reasoning about, concurrent or distributed object-oriented languages. For instance, locality can guarantee the fundamental property that an object has unique identity. In object-oriented languages, the name a of an object may be transmitted; the recipient may use a to access its methods, but it cannot create a new object called a. When representing objects in the π-calculus, this usually translates into the constraint that the process receiving the object name may only use it in output [64, 27, 53, 28, 45, 55, 33].

Restriction (b), that is the *absence of matching*, is an important requirement too. Indeed, name-testing, like testing equality between pointers in imperative languages, prevents many useful program optimisations and transformations. Also from a programming point of view, the usefulness of matching is questionable. For instance, Join, Pict, and TyCO do not provide any construct for testing channels.

In this paper, we study the operational and algebraic theory of Lπ. We focus on bisimulation-based behavioural equivalences, and more precisely on *barbed congruence* [40]. Barbed congruence equates processes that, very roughly, in all contexts give rise to the same set of *observable actions*. Like other contextually-defined forms of bisimulation, barbed congruence is sensitive to the set of operators of a calculus. Lπ is a sub-calculus of $\pi_{\rm a}$ and π-calculus, and therefore has fewer contexts. This allows us to gain useful process equalities. In this respect, the most important algebraic law of asynchronous π that is not in the theory of the synchronous π-calculus is the *asynchrony law*:

$$a(x).\,\overline{a}x \;=\; \mathbf{0}.$$

The asynchrony law essentially says that inputs can not be observed in $\pi_{\rm a}$. Although this law is useful (it is used for instance by Nestmann and Pierce to prove the correctness of an encoding of

[1] We may also view (b) as a consequence of (a), since testing the identity of a name requires more than the output capability.

guarded choice [41]), it seems fair to say that the restriction to asynchronous contexts does not allows us to gain much.

By contrast, asynchrony has strong semantic consequences under simplifications (a) and (b). Consider the following laws which are valid (under the specified conditions) in Lπ, but are false in π_{a} and in π-calculus:

$$\begin{aligned}
\overline{a}b &= (\boldsymbol{\nu} c)(\overline{a}c \mid !c(x).\, \overline{b}x) && (1)\\
(\boldsymbol{\nu} a)(!a(x).\, R \mid P \mid Q) &= (\boldsymbol{\nu} a)(!a(x).\, R \mid P) \mid (\boldsymbol{\nu} a)(!a(x).\, R \mid Q) && (2)\\
(\boldsymbol{\nu} a)(!a(x).\, R \mid C[\overline{a}b]) &= (\boldsymbol{\nu} a)(!a(x).\, R \mid C[R\{b/x\}]) && (3)\\
(\boldsymbol{\nu} c)(\overline{a}c) &= (\boldsymbol{\nu} c)(\overline{a}c \mid c(x).\, \mathbf{0}) && (4)\\
(\boldsymbol{\nu} c)(\overline{a}c) &= (\boldsymbol{\nu} c)(\overline{a}c \mid \overline{c}b) && (5)
\end{aligned}$$

Law 1, where $c \neq b$, equates processes that may perform syntactically different outputs: the process on the left performs the output of a global name b, whereas that on the right the output of a private name c. The forwarder process $!c(x).\, \overline{b}x$ makes the two processes indistinguishable. Law 2 is a distributivity law for replicated resources known as one of Milner's *replication theorems* [34]. This law is true in the π-calculus, under the hypothesis that name a is never transmitted and never used in input by processes P, Q and R. In Lπ, Law 2 is still valid when name a is transmitted by P, Q or R to the environment. Law 3 is reminiscent of *inline expansion*, an optimisation technique for functional languages which replaces a function call (the particle $\overline{a}b$) with an instance of the function body (the process $R\{b/x\}$). This law holds in Lπ provided that name a does not appear free in input in process R and context $C[\cdot]$. (Also, by α-conversion we assume that all bound names are different from each other.) Finally, Laws 4 and 5 represent two forms of garbage collection. Notice that, in both laws, after the initial output, the derivatives are very different.

The main difficulty when proving that two processes are barbed congruent is represented by the quantification over contexts in the definition of barbed congruence. This quantification makes very hard to prove process equalities, and makes mechanical checking impossible. Simpler proof techniques are based on *labelled bisimulations* whose definitions do not use context quantification. These bisimulations should imply, or (better) coincide with, barbed congruence. In the π-calculus barbed congruence coincides with the closure under substitutions of *synchronous early bisimilarity* [49]. Similarly, in π_{a} barbed congruence coincides with the closure under substitutions of *asynchronous early bisimilarity* [3]. In these proofs, a central role is played by the *matching* construct, for testing equality between names. If matching is removed from the language, then (the closure under substitutions of) early bisimilarity still implies barbed congruence, but the converse does not hold. Furthermore, both characterisations are given on the class of the *image-finite* processes and exploit the n-approximants of the labelled equivalences. More recently, Fournet and Gonthier showed that this requirement can be relaxed in π_{a} [17].

In this paper, we give two characterisations of barbed congruence in Lπ (as usual, on image-finite processes). The first is based on an embedding of Lπ into a subcalculus where all names emitted are private. Barbed congruence between processes of Lπ coincides, on their images, with (a slight variant of) *asynchronous ground bisimilarity* [3]. The second characterisation is based on a new labelled transition system (LTS) which modifies the standard one so to reveal what is observable in Lπ, that is, what an external observer that behaves like a Lπ process can see by interacting with a Lπ process. Barbed congruence in Lπ coincides with the standard asynchronous ground bisimilarity defined on the new LTS. We then show enhancements of the coinductive proof methods presented by means of *up–to proof techniques*, some of which are standard up-to proof techniques for π-calculus bisimilarities, others are new.

Technical differences of our characterisations with respect to those in π_{a} and π-calculus [3, 49] are: (i) the labelled bisimilarities of Lπ are congruence relations and therefore do not have to be closed under substitutions to obtain barbed congruence; (ii) the labelled bisimilarities in Lπ are ground, rather than early, which means that they do not need universal quantifications on the received names; (iii) the characterisations in Lπ are proved without the matching construct, which is essential in the proofs in π_{a} and π-calculus.

Proof techniques for Lπ can be exploited to reason about languages such as Pict, Join, and TyCO, either by directly adapting the techniques to these languages, or by means of encodings into Lπ. The theory of Lπ (for instance, its algebraic properties and labelled bisimulations) is also useful in calculi where the usage of some names goes beyond the syntax of Lπ. For instance, there could be a distinct set of synchronous names, or names that can be tested for identity (see, for instance, [33]). A type system could be used to distinguish between "Lπ names" and the other names, and the theory of Lπ can then be applied to the formers.

For simplicity we develop the theory for a *monadic* calculus (where exactly one name may be transmitted); the generalisation to the *polyadic* version (where tuple of names may be transmitted) is straightforward.

1.1 Outline

In Section 2 we give syntax and operational semantics of Lπ. In Section 3 we recall some common bisimulation-based behavioural equivalences for π-calculi. In Section 4 we present some special processes, the *link processes*, which are important in the theory of Lπ. In Section 5 we give the first proof technique for barbed congruence. In Section 6 we give the second proof technique for barbed congruence. In Section 7 we prove that these two proof techniques completely describe barbed congruence. In Section 8 we enhance the second proof technique with a new form of up-to proof technique. Section 9 is entirely devoted to applications: in subsection 9.1 we use link processes to express name substitutions; in Subsection 9.3 we prove that the *delayed input* (a form of non-blocking input prefixing) is derivable in Lπ, and present some of its algebraic properties. In Subsection 9.2 we prove a sharpened form of Milner's *replication theorems* [34]. In Subsection 9.4 we give an optimisation of the encoding of call-by-name λ-calculus and, exploiting delayed input, we derive an encoding of *strong call-by-name*. In Subsection 9.5 we prove some laws for Fournet and Gonthier's Join-calculus [16]. In Subsection 9.6 we prove some non-full abstraction and full abstraction results for Boreale's encoding [7] of *external mobility* (communication of free names) in terms of *internal mobility* (communication of private names). In Subsection 9.7 we prove that Thielecke's axiomatic semantics of the *Continuation Passing Style calculus* [61] is operationally sound. The paper ends with discussion of summary of our result and comparison with related works.

2 The calculus Lπ

Lπ is a subset of the asynchronous π-calculus. Lπ has operators of inaction, input prefix, asynchronous output, parallel composition, restriction and replicated input.

We recall that in asynchronous settings the π-calculus operator of non-deterministic choice does not make sense. The reason is that in asynchronous calculi a message is sent simply when it is unguarded, i.e., when it is not underneath an input prefix. The coincidence between "message sent" and "message unguarded" would break if the calculus had the choice. In a process $\overline{a}b + c(x).\,P$, message $\overline{a}b$, although unguarded, is not really sent because its offer can vanish at any moment. The availability of $\overline{a}b$ only depends on whether the other summand $c(x).\,P$ can performs its input at c. An asynchronous process that non-deterministically can chose to send $\overline{a}b$ or to receive at $c(x).\,P$ should be written $\tau.\,\overline{a}b + c(x).\,P$ (see also Castellani and Hennessy's asynchronous choice [13]). Nestmann and Pierce have showed that this form of choice, in which each summand is guarded by a τ or by an input prefix, can be encoded in the asynchronous π-calculus [41].

Lπ, like several other dialects of the π-calculus, adopts replicated inputs instead of *recursion*. This is because: (i) replicated input has the same expressive power as full replication [25] and recursion [34, 60]; (ii) replicated input has a simpler semantics and is handy for implementations. The theory and the results presented in this paper would however hold also with full replication or recursion.

$$\texttt{inp:}\ \frac{}{a(x).\,P \xrightarrow{a(x)} P} \qquad \texttt{out:}\ \frac{}{\overline{a}b.\,P \xrightarrow{\overline{a}b} P}$$

$$\texttt{open:}\ \frac{P \xrightarrow{\overline{a}b} P' \quad a \neq b}{(\boldsymbol{\nu} b)P \xrightarrow{\overline{a}(b)} P'} \qquad \texttt{close:}\ \frac{P \xrightarrow{\overline{a}(c)} P' \quad Q \xrightarrow{a(c)} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu} c)(P' \mid Q')}$$

$$\texttt{com:}\ \frac{P \xrightarrow{\overline{a}b} P' \quad Q \xrightarrow{a(x)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{b/x\}} \qquad \texttt{par:}\ \frac{P \xrightarrow{\mu} P' \quad \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\mu} P' \mid Q}$$

$$\texttt{new:}\ \frac{P \xrightarrow{\mu} P' \quad a \notin \mathrm{n}(\mu)}{(\boldsymbol{\nu} a)P \xrightarrow{\mu} (\boldsymbol{\nu} a)P'} \qquad \texttt{rep:}\ \frac{}{!a(x).\,P \xrightarrow{a(x)} P \mid !a(x).\,P}$$

Table 1: Labelled transition system for Lπ

Definition 2.1 *Let $\mathcal{N}$ be a countable infinite set of names, ranged over by small letters ($a, b, c, \ldots,$ x, y, z). The grammar of* Lπ*-processes is*

$$P ::= \mathbf{0} \;\big|\; a(b).\,P \;\big|\; \overline{a}b \;\big|\; P \mid P \;\big|\; (\boldsymbol{\nu} a)P \;\big|\; !a(b).\,P$$

with the syntactic constraint *that in processes $a(b).\,P$ and $!a(b).\,P$ name b may not occur free in P in input position.*

Input prefix $a(b).\,P$ and restriction $(\boldsymbol{\nu} b)P$ acts as *binders* for name b leading to the usual notions of *free* and *bound* occurrences of names, $\mathrm{fn}(\cdot)$ and $\mathrm{bn}(\cdot)$, and α-conversion, $\equiv_\alpha$. We will identify processes up to α-conversion. More formally we will view process terms as representatives of their equivalence class with respect to $\equiv_\alpha$, and these representatives will always be chosen so that bound names are distinct from free names. The *names* of a process P, written $\mathrm{n}(P)$, are given by $\mathrm{fn}(P) \cup \mathrm{bn}(P)$. Sometimes, $\mathrm{fn}(P, Q)$ is used as a shorthand for $\mathrm{fn}(P) \cup \mathrm{fn}(Q)$, and similarly for $\mathrm{n}(P, Q)$ and $\mathrm{bn}(P, Q)$. In a statement, a name declared *fresh* is supposed to be different from any other name appearing in the objects of the statement, like processes or substitutions.

Substitutions, ranged over by $\sigma, \sigma', \ldots$ are functions from $\mathcal{N}$ to $\mathcal{N}$; for any process P, we write $P\sigma$ for the process obtained by applying σ to P with renaming possibly involved to avoid capture of free names. The following order precedence when writing processes is assumed: substitution > { restriction, input prefix, replicated input } > parallel composition. We write $\overline{a}$ and $a.\,P$ when the name transmitted at a is not important. We write $\tau.\,P$ as an abbreviation for $(\boldsymbol{\nu} a)(\overline{a} \mid a.\,P)$ where $a \notin \mathrm{fn}(P)$. We write $\widetilde{a}$ to denote a tuple of names, such as $a_1, \ldots, a_n$. We write $(\boldsymbol{\nu}\widetilde{a})P$ for $(\boldsymbol{\nu} a_1)\ldots(\boldsymbol{\nu} a_n)P$.

The *operational semantics* of Lπ is given by means of *labelled transition system* (LTS) in the SOS style of [48]. The LTS is the standard one, in the *late* style [39, 54], and is presented in Table 1. Transitions are of the form $P \xrightarrow{\mu} P'$, where *action* μ can be: τ (interaction), $a(b)$ (input), $\overline{a}b$ (free output) and $\overline{a}(b)$ (bound output, that is the emission of a private name b at a). In these actions, a is the *subject* and b the *object*. Free and bound names of actions and processes are defined as usual. We write $\xrightarrow{\hat{\mu}}$ to mean $P \xrightarrow{\mu} Q$, if $\mu \neq \tau$, and either $P = Q$ or $P \xrightarrow{\tau} Q$, if $\mu = \tau$. Relation $\Longrightarrow$ is the reflexive and transitive closure of $\xrightarrow{\tau}$; moreover, $\stackrel{\mu}{\Longrightarrow}$ stands for $\Longrightarrow \xrightarrow{\mu} \Longrightarrow$, and $\stackrel{\hat{\mu}}{\Longrightarrow}$ for $\stackrel{\mu}{\Longrightarrow}$ if $\mu \neq \tau$, and for $\Longrightarrow$ if $\mu = \tau$.

Finally, in the sequel, we use the symbol $\equiv$ to denote *structural congruence*, a relation used to rearrange the structure of processes [34].

Definition 2.2 (Structural congruence) Structural congruence*, $\equiv$, is the smallest congruence relation satisfying the axioms below:*

- $P \mid \mathbf{0} \equiv P,\ \ P \mid Q \equiv Q \mid P,\ \ P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
- $(\boldsymbol{\nu} a)\mathbf{0} \equiv \mathbf{0},\ \ (\boldsymbol{\nu} a)(\boldsymbol{\nu} b)P \equiv (\boldsymbol{\nu} b)(\boldsymbol{\nu} a)P,\ \ (\boldsymbol{\nu} a)(P \mid Q) \equiv P \mid (\boldsymbol{\nu} a)Q$ *if* $a \notin \mathrm{fn}(P)$.

3 Some background on behavioural equivalences

A crucial notion in a process calculus is that of *behavioural equality* between processes. As said in the introduction, we focus on bisimulation-based behavioural equivalences, and more precisely on *barbed congruence*. Barbed congruence can be defined in any calculus possessing: (i) an *interaction relation* (the τ-steps in the π-calculus), modelling the evolution of the system; and (ii) an *observability predicate* $\downarrow_a$ for each name a, which detects the possibility of a process of accepting a communication with the environment at a. More precisely, we write $P \downarrow_a$ if P can make an output action whose subject is a, that is, if there exist P' and b such that $P \xrightarrow{\overline{a}b} P'$ or $P \xrightarrow{\overline{a}(d)} P'$. We write $P \Downarrow_a$ if $P \Longrightarrow P'$ and $P' \downarrow_a$. Unlike synchronous π-calculus, in asynchronous calculi it is natural to restrict the observation to output actions [3]. The reason is that in asynchronous calculi the observer has no direct way of knowing when a message emitted is received. Below, we define barbed congruence on a generic subset $\mathcal{P}$ of π-calculus processes. A *$\mathcal{P}$-context* is a process of $\mathcal{P}$ with a single hole $[\cdot]$ in it.

Definition 3.1 (Barbed relations) *A symmetric relation $\mathcal{S}$ on processes is a* barbed bisimulation *if $P \mathrel{\mathcal{S}} Q$ implies:*

1. *If $P \xrightarrow{\tau} P'$ then there exists Q' such that $Q \Longrightarrow Q'$ and $P' \mathrel{\mathcal{S}} Q'$.*
2. *If $P \downarrow_a$ then $Q \Downarrow_a$.*

Two processes P and Q are barbed bisimilar*, written $P \stackrel{\bullet}{\approx} Q$, if $P \mathrel{\mathcal{S}} Q$ for some barbed bisimulation $\mathcal{S}$. Let $\mathcal{P}$ be a set of π-calculus processes, and $P, Q \in \mathcal{P}$. We say that P and Q are* barbed congruent in $\mathcal{P}$*, written $P \cong_{\mathcal{P}} Q$, if for each $\mathcal{P}$-context $C[\cdot]$, it holds that $C[P] \stackrel{\bullet}{\approx} C[Q]$.*

With the exception of [17], characterisations of barbed congruence in π-calculus in terms of labelled bisimilarities are usually given on the class of *image-finite processes*, by exploiting the n-approximants of the labelled equivalence, that is a characterisation of the labelled bisimilarities as intersections of appropriate inductively-defined relations.

Definition 3.2 *The class of* image-finite processes *is the largest subset $\mathcal{I}$ of π-processes which is derivation closed and such that $P \in \mathcal{I}$ implies that, for all μ, the set $\{P' : P \stackrel{\mu}{\Longrightarrow} P'\}$, quotiented by alpha conversion, is finite.*

In π-calculi, barbed congruence coincides with the closure under substitutions of early bisimilarity [49, 3]. In asynchronous calculi *without* matching, like Lπ, early bisimilarity is a congruence and it coincides with its simpler *ground* variant [22, 56], which differs from the early one in that there is no universal quantification in the input clause.

Definition 3.3 *A symmetric relation $\mathcal{S}$ on π-terms is an $o\tau$-*bisimulation *if $P \mathrel{\mathcal{S}} Q$, $P \xrightarrow{\mu} P'$, μ is not an input and* $\mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$*, implies that there exists Q' such that $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ and $P' \mathrel{\mathcal{S}} Q'$.*

Definition 3.4 (Ground bisimilarities)

- Synchronous ground bisimulation *is the largest $o\tau$-bisimulation $\mathcal{S}$ on processes such that $P \mathrel{\mathcal{S}} Q$ and $P \xrightarrow{a(b)} P'$, with $b \notin \mathrm{fn}(Q)$, implies that there exists Q' such that $Q \stackrel{a(b)}{\Longrightarrow} Q'$ and $P' \mathrel{\mathcal{S}} Q'$.*

  Two processes P and Q are synchronous ground bisimilar*, written $P \approx Q$, if $P \mathrel{\mathcal{S}} Q$ for some synchronous ground bisimulation $\mathcal{S}$.*

- Asynchronous ground bisimulation *is the largest $o\tau$-bisimulation $\mathcal{S}$ on processes such that $P \mathrel{\mathcal{S}} Q$ and $P \xrightarrow{a(b)} P'$, with $b \notin \mathrm{fn}(Q)$, implies that there exists Q' such that:*

  1. *either $Q \stackrel{a(b)}{\Longrightarrow} Q'$ and $P' \mathrel{\mathcal{S}} Q'$*
  2. *or $Q \Longrightarrow Q'$ and $P' \mathrel{\mathcal{S}} (Q' \mid \overline{a}b)$.*

  Two processes P and Q are asynchronous ground bisimilar*, written $P \approx_{\mathrm{a}} Q$, if $P \mathrel{\mathcal{S}} Q$ for some asynchronous ground bisimulation $\mathcal{S}$.*

Sometimes, it may be useful to count the number of silent moves performed by a process. The *synchronous expansion* [5], written $\lesssim$, is an asymmetric variant of $\approx$ such that $P \lesssim Q$ holds if $P \approx Q$, and Q has at least as many τ-moves as P.

Similarly, one can define *asynchronous expansion*.

Definition 3.5 (Expansions)

- *A relation $\mathcal{S}$ on processes is a* synchronous ground expansion *if $P \mathrel{\mathcal{S}} Q$ implies:*

  1. *If $P \xrightarrow{\mu} P'$, $\mu \in \{\tau, \overline{a}b, \overline{a}(b), a(b)\}$ and $\mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$, then there exists Q' such that $Q \stackrel{\mu}{\Longrightarrow} Q'$ and $P' \mathrel{\mathcal{S}} Q'$.*
  2. *If $Q \xrightarrow{\mu} Q'$, $\mu \in \{\tau, \overline{a}b, \overline{a}(b), a(b)\}$ and $\mathrm{bn}(\mu) \cap \mathrm{fn}(P) = \emptyset$, then there exists P' such that $P \xrightarrow{\hat{\mu}} P'$ and $P' \mathrel{\mathcal{S}} Q'$.*

  We write $P \lesssim Q$ if $P \mathrel{\mathcal{S}} Q$ for some synchronous ground expansion $\mathcal{S}$.

- *A relation $\mathcal{S}$ on processes is an* asynchronous ground expansion *if $P \mathrel{\mathcal{S}} Q$ implies:*

  1. *If $P \xrightarrow{\mu} P'$, μ is not an input, and $\mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$, then there exists Q' such that $Q \stackrel{\mu}{\Longrightarrow} Q'$ and $P' \mathrel{\mathcal{S}} Q'$.*
  2. *If $P \xrightarrow{a(b)} P'$, $b \notin \mathrm{fn}(Q)$, there exists Q' such that:*

     (a) either $Q \stackrel{a(b)}{\Longrightarrow} Q'$ and $P' \mathrel{\mathcal{S}} Q'$

     (b) or $Q \stackrel{\tau}{\Longrightarrow} Q'$ and $P' \mathrel{\mathcal{S}} (Q' \mid \overline{a}b)$.
  3. *If $Q \xrightarrow{\mu} Q'$, μ is not an input, and $\mathrm{bn}(\mu) \cap \mathrm{fn}(P) = \emptyset$, then there exists P' such that $P \xrightarrow{\hat{\mu}} P'$ and $P' \mathrel{\mathcal{S}} Q'$;*
  4. *if $Q \xrightarrow{a(b)} Q'$, $b \notin \mathrm{fn}(P)$, then there exists P' such that:*

     (a) either $P \xrightarrow{a(b)} P'$ and $P' \mathrel{\mathcal{S}} Q'$

     (b) or $P \xrightarrow{\hat{\tau}} P'$ and $(P \mid \overline{a}b) \mathrel{\mathcal{S}} Q'$.

  We write $P \lesssim_{\mathrm{a}} Q$, if $P \mathrel{\mathcal{S}} Q$ for some asynchronous ground expansion $\mathcal{S}$.

4 The link processes

The theory of Lπ is based on special processes called *links* which behave as name buffers receiving names at one end-point and retransmitting them at the other end-point (in the π-calculus literature, links are sometimes called forwarders [26] or wires [60]).

Definition 4.1 (Static link) *Given any two names a and b, we call* static link *the process below:*

$$a \triangleright b \stackrel{\mathrm{def}}{=} !a(x).\,\overline{b}x.$$

We sometimes use a more sophisticated form of link $a \rightarrow b$, which does not perform free outputs: the name sent at b is not x, but a link to x (this is the definition of links in calculi where all outputs emit private names [52]).

Definition 4.2 (Dynamic link) *Given two names a and b, we call* dynamic link *the process defined by the following recursive definition:*

$$a \rightarrow b \stackrel{\text{def}}{=} !a(x).\,(\boldsymbol{\nu} c)(\overline{b}c \mid c \rightarrow x).$$

Being recursively defined, the process $a \rightarrow b$ is not in Lπ. However, there exists a process in Lπ which is synchronous bisimilar to it. In the following we explain how this process can be built up. In [34], Milner shows that recursive definitions can be encoded, up to bisimilarity, in terms of replication. When applying Milner's encoding to a dynamic link we get a process which does not respect the Lπ constraint on the output capability. This problem can be avoided by rewriting the definition of dynamic links as:

$$a \rightarrow b \stackrel{\text{def}}{=} !a(x).\,\mathrm{out}(b,x)$$

where $\mathrm{out}(b,x)$ is recursively defined as:

$$\mathrm{out}(b,x) \stackrel{\text{def}}{=} (\boldsymbol{\nu} c)(\overline{b}c \mid !c(y).\,\mathrm{out}(x,y)).$$

Process $\mathrm{out}(b,x)$ can be expressed in (polyadic) Lπ in terms of replication as follows:

$$\mathrm{out}(b,x) \approx (\boldsymbol{\nu} o)(\overline{o}\langle b,x\rangle \mid !o(b,x).\,(\boldsymbol{\nu} c)(\overline{b}c \mid !c(y).\,\overline{o}\langle x,y\rangle)).$$

As already pointed out in Section 3, in asynchronous calculi, the relation $\approx$ is a congruence. So, the process $a \rightarrow b$ can be rewritten, up to $\approx$, in (polyadic) Lπ. Finally, by exploiting the encoding $\{\!|\cdot|\!\}$ defined below, we get the desired process of Lπ which is bisimilar to the recursive process $a \rightarrow b$. The encoding $\{\!|\cdot|\!\}$ is a slight variant of Milner's encoding [34] of polyadic processes into monadic ones. [2] $\{\!|\cdot|\!\}$ is an homomorphism on all operators except input and output for which we have:

- $\{\!| a(x_1,x_2).\,P |\!\} \stackrel{\text{def}}{=} a(w).\,(\boldsymbol{\nu} c_1 c_3)(\overline{w}c_1 \mid c_1(c_2).\,(\overline{c_2}c_3 \mid c_3(x_1).\,c_1(x_2).\,\{\!| P |\!\}))$
- $\{\!| \overline{a}\langle b_1,b_2\rangle |\!\} \stackrel{\text{def}}{=} (\boldsymbol{\nu} w)(\overline{a}w \mid w(c_1).\,(\boldsymbol{\nu} c_2)(\overline{c_1}c_2 \mid c_2(c_3).\,(\overline{c_3}b_1 \mid \overline{c_1}b_2)))$.

To conclude the section we show how links can be joined with each other. This result will be useful later on.

Proposition 4.3 *Let a and b be names different from c. Then:*

1. $(\boldsymbol{\nu} b)(a \triangleright b \mid b \triangleright c) \gtrsim a \triangleright c$
2. $(\boldsymbol{\nu} b)(a \rightarrow b \mid b \rightarrow c) \gtrsim a \rightarrow c$
3. $(\boldsymbol{\nu} b)(a \triangleright b \mid b \rightarrow c) \gtrsim a \rightarrow c$
4. $(\boldsymbol{\nu} b)(a \rightarrow b \mid b \triangleright c) \gtrsim a \rightarrow c$.

Proof: Parts 1, 3, 4 are proved by simply exhibiting the appropriate expansion relations. Part 2 has been already proved in [7] and requires up-to context proof techniques. $\square$

$$[\![P_1 \mid P_2]\!] \stackrel{\text{def}}{=} [\![P_1]\!] \mid [\![P_2]\!] \qquad [\![a(x).\,P]\!] \stackrel{\text{def}}{=} a(x).\,[\![P]\!] \qquad [\![!a(x).\,P]\!] \stackrel{\text{def}}{=} !a(x).\,[\![P]\!]$$

$$[\![(\boldsymbol{\nu} a)P]\!] \stackrel{\text{def}}{=} (\boldsymbol{\nu} a)[\![P]\!] \qquad [\![\overline{a}b]\!] \stackrel{\text{def}}{=} (\boldsymbol{\nu} c)(\overline{a}c \mid c \to b) \qquad [\![\mathbf{0}]\!] \stackrel{\text{def}}{=} \mathbf{0}$$

Table 2: The encoding of free outputs in terms of bound outputs

5 A "translation-based" proof technique

In this section, we give a proof technique for barbed congruence in Lπ based on an encoding of free outputs in terms of bound outputs (see Table 2). The encoding, written $[\![\cdot]\!]$, is an homomorphism on all operators except output. The output particle $\overline{a}b$ is mapped onto the process $(\boldsymbol{\nu} c)(\overline{a}c \mid c \to b)$ where $c \notin \{a, b\}$. This encoding is essentially the asynchronous version of an encoding used by Boreale [7] to compare internal and external mobility. In Boreale's encoding, an output $\overline{a}b$ is mapped onto the synchronous process $(\boldsymbol{\nu} c)(\overline{a}c.\, c \to b)$. This process is behaviourally the same as $(\boldsymbol{\nu} c)(\overline{a}c \mid c \to b)$: in both cases the link $c \to b$ becomes active only when the bound output at a is consumed. Note that the process $(\boldsymbol{\nu} c)(\overline{a}c \mid c \to b)$ coincides with the recursive definition $\mathrm{out}(a, b)$ seen in Section 4.

In Section 5.1, we present some results on $[\![\cdot]\!]$ already proved by Boreale [7] for its encoding and that trivially apply to our encoding as well. Then, in Section 5.2, we give the proof technique.

5.1 Background

The encoding $[\![\cdot]\!]$ commutes with substitutions, i.e., for each substitution σ it holds that $[\![P\sigma]\!] = [\![P]\!]\sigma$. In [7] Boreale proves an operational correspondence between processes P and $[\![P]\!]$; he also gives an adequacy result for $[\![\cdot]\!]$ with respect to barbed bisimulation.

Lemma 5.1 (Boreale [7]) *Let P be a process in* Lπ.

1. *Suppose that $P \xrightarrow{\mu} P'$. Then we have:*
   (a) *if $\mu = a(c)$ then $[\![P]\!] \xrightarrow{a(c)} \gtrsim [\![P']\!]$*
   (b) *if $\mu = \overline{a}b$ then $[\![P]\!] \xrightarrow{\overline{a}(c)} \gtrsim c \to b \mid [\![P']\!]$, with $c \notin \mathrm{fn}(P')$*
   (c) *if $\mu = \overline{a}(b)$ then $[\![P]\!] \xrightarrow{\overline{a}(c)} \gtrsim (\boldsymbol{\nu} b)(c \to b \mid [\![P']\!])$, with $c \notin \mathrm{fn}(P')$*
   (d) *if $\mu = \tau$ then $[\![P]\!] \xrightarrow{\tau} \gtrsim [\![P']\!]$.*
2. *Suppose that $[\![P]\!] \xrightarrow{\mu} P_1$. Then there exists $P' \in$ Lπ such that:*
   (a) *if $\mu = a(c)$ then $P \xrightarrow{a(c)} P'$, with $P_1 \gtrsim [\![P']\!]$*
   (b) *if $\mu = \overline{a}(c)$ then:*
      i. *either $P \xrightarrow{\overline{a}b} P'$, with $c \notin \mathrm{fn}(P')$ and $P_1 \gtrsim (c \to b \mid [\![P']\!])$*
      ii. *or $P \xrightarrow{\overline{a}(b)} P'$, with $c \notin \mathrm{fn}(P')$ and $P_1 \gtrsim (\boldsymbol{\nu} b)(c \to b \mid [\![P']\!])$*
   (c) *if $\mu = \tau$ then $P \xrightarrow{\tau} P'$ with $P_1 \gtrsim [\![P']\!]$.*

The proof of the lemma above relies on Proposition 4.3(2) and the technical but important Lemma 5.2. Roughly, Lemma 5.2 says that name substitution on translated terms can be encoded, up-to expansion, using private dynamic links.

[2] For our purposes it suffices to consider the encoding where only pairs are taken into account.

Lemma 5.2 (Boreale [7]) *Let P be an* $\mathrm{L}\pi$*-process, and a and b two names such that $a \neq b$ and a does not occur free in P in input-subject position. Then:*

$$(\boldsymbol{\nu}a)(a \rightarrow b \mid [\![P]\!]) \gtrsim [\![P]\!]\{b/a\}.$$

From Lemma 5.1 Boreale derives the following adequacy result for $[\![\cdot]\!]$.

Theorem 5.3 (Boreale [7]) *Let P and Q be two processes in* $\mathrm{L}\pi$. *Then:*

$$P \stackrel{\bullet}{\approx} Q \;\text{ iff }\; [\![P]\!] \stackrel{\bullet}{\approx} [\![Q]\!].$$

5.2 The proof technique

In this section we present our translation-based proof technique. Roughly, to verify if two $\mathrm{L}\pi$-processes are barbed congruent we translate them, by means of $[\![\cdot]\!]$, and then check if the resulting processes are related by the bisimilarity $\asymp_{\mathrm{a}}$ (cf. Definition 5.6), a variant of asynchronous ground bisimilarity. The bisimilarity $\asymp_{\mathrm{a}}$ is defined on the image of $[\![\cdot]\!]$ which is a calculus itself, denoted by $\mathrm{L}_{[\![\pi]\!]}$, whose operators are the same as those of $\mathrm{L}\pi$ except for the output construct $\overline{a}b$ which is replaced by its translation $[\![\overline{a}b]\!]$. Formally, the syntax of $\mathrm{L}_{[\![\pi]\!]}$ is the following:

$$P ::= \mathbf{0} \;\Big|\; a(x).\,P \;\Big|\; [\![\overline{a}b]\!] \;\Big|\; P \mid P \;\Big|\; (\boldsymbol{\nu}a)P \;\Big|\; !a(x).\,P$$

with the same constraint on received names as in $\mathrm{L}\pi$.

Lemma 5.4 *If P is an* $\mathrm{L}\pi$*-process then $[\![P]\!]$ is an* $\mathrm{L}_{[\![\pi]\!]}$*-process. Conversely, for each* $\mathrm{L}_{[\![\pi]\!]}$*-process Q there exists an* $\mathrm{L}\pi$*-process P such that $[\![P]\!] = Q$.*

It is easy to see that $\mathrm{L}_{[\![\pi]\!]}$ is closed under labelled transitions:

Lemma 5.5 *Let P be an* $\mathrm{L}_{[\![\pi]\!]}$*-process. If $P \xrightarrow{\mu} P_1$ then $P_1 \in \mathrm{L}_{[\![\pi]\!]}$.*

Proof: By structural induction. The most interesting case is when

$$P = [\![\overline{a}b]\!] \stackrel{\text{def}}{=} (\boldsymbol{\nu}c)(\overline{a}c \mid c \rightarrow b) = (\boldsymbol{\nu}c)(\overline{a}c \mid !c(x).\,[\![\overline{b}x]\!]).$$

In this case, $\mu = \overline{a}(c)$ and $P_1 = !c(x).\,[\![\overline{b}x]\!]$ which, by definition, is in $\mathrm{L}_{[\![\pi]\!]}$. $\square$

The bisimilarity $\asymp_{\mathrm{a}}$ differs from that of asynchronous ground bisimilarity (cf. Definition 3.4) only in clause 2, where the output particle $\overline{a}b$ is replaced by its translation $[\![\overline{a}b]\!]$.

Definition 5.6 ($\asymp_{\mathrm{a}}$-bisimilarity) *A symmetric relation $\mathcal{S}$ on processes of* $\mathrm{L}_{[\![\pi]\!]}$ *is a* $\asymp_{\mathrm{a}}$-bisimulation *if whenever $P \;\mathcal{S}\; Q$ the following holds:*

1. *If $P \xrightarrow{\tau} P'$, then there exists Q' such that $Q \Longrightarrow Q'$ and $P' \;\mathcal{S}\; Q'$.*
2. *If $P \xrightarrow{\overline{a}(b)} P'$, $b \notin \mathrm{fn}(Q)$, then there exists Q' such that $Q \stackrel{\overline{a}(b)}{\Longrightarrow} Q'$ and $P' \;\mathcal{S}\; Q'$.*
3. *If $P \xrightarrow{a(b)} P'$, $b \notin \mathrm{fn}(Q)$, then there exists Q' such that:*
   - *(a) either $Q \stackrel{a(b)}{\Longrightarrow} Q'$ and $P' \;\mathcal{S}\; Q'$*
   - *(b) or $Q \Longrightarrow Q'$ and $P' \;\mathcal{S}\; (Q' \mid [\![\overline{a}b]\!])$.*

Processes P and Q are $\asymp_{\mathrm{a}}$-bisimilar, written $P \asymp_{\mathrm{a}} Q$, if $P \,\mathcal{S}\, Q$ for some $\asymp_{\mathrm{a}}$-bisimulation $\mathcal{S}$.

This bisimilarity is designed to be used on $\mathrm{L}_{[\![\pi]\!]}$-processes and therefore its definition does not contain clauses for free output actions.

Now, we give a few technical lemmas to prove that $\asymp_{\mathrm{a}}$ is a congruence relation. In order to prove that $\asymp_{\mathrm{a}}$ is preserved by parallel composition, we adapt the up-to expansion proof technique of [59] to our new bisimilarity.

Definition 5.7 ($\precsim_{\mathrm{a}}$-bisimulation up to $\gtrsim$ and $\approx$) *A symmetric relation $\mathcal{S}$ is a* $\precsim_{\mathrm{a}}$-bisimulation up to $\gtrsim$ and $\approx$ *if whenever $P \mathcal{S} Q$ the following holds:*

1. *If $P \xrightarrow{\tau} P'$, then there exists Q' such that $Q \Longrightarrow Q'$ and $P' \gtrsim \mathcal{S} \approx Q'$.*

2. *If $P \xrightarrow{\overline{a}(b)} P'$, $b \notin \mathrm{fn}(Q)$, then there exists Q' such that $Q \stackrel{\overline{a}(b)}{\Longrightarrow} Q'$ and $P' \gtrsim \mathcal{S} \approx Q'$.*

3. *If $P \xrightarrow{ab} P'$, $b \notin \mathrm{fn}(P, Q)$, then there exists Q' such that:*

   (a) either $Q \stackrel{ab}{\Longrightarrow} Q'$ and $P' \gtrsim \mathcal{S} \approx Q'$

   (b) or $Q \Longrightarrow Q'$ and $P' \gtrsim \mathcal{S} \approx (Q' \mid [\![\overline{a}b]\!])$.

Lemma 5.8 *If $\mathcal{S}$ is a $\precsim_{\mathrm{a}}$-bisimulation up to $\gtrsim$ and $\approx$ then $\mathcal{S} \subseteq \precsim_{\mathrm{a}}$.*

Proof: The proof is analogous to that in [59]. If $\mathcal{S}$ is a $\precsim_{\mathrm{a}}$-bisimulation up to $\gtrsim$ and $\approx$, then one shows that the relation $\approx \mathcal{S} \approx$ is a $\precsim_{\mathrm{a}}$-bisimulation. This follows from the transitivity of $\approx$ and the fact that $\approx$ is preserved by parallel composition (The latter result is necessary to deal with clause 3.b of Definition 5.6). Finally, since $\mathcal{S} \subseteq \approx \mathcal{S} \approx \subseteq \precsim_{\mathrm{a}}$, the proof is complete. $\Box$

In the sequel, we often use a $\precsim_{\mathrm{a}}$-*bisimulation up to* $\equiv$ proof technique. The soundness of this technique follows from Lemma 5.8 and the fact that $\equiv$ is contained in $\gtrsim$ (and therefore also in $\approx$).

Lemma 5.9 gives us some information about the structure of the $\mathrm{L}_{[\![\pi]\!]}$-processes which may perform an output action. Part 2 will be needed in Lemma 5.10(2) to prove that $\precsim_{\mathrm{a}}$ is preserved by parallel composition.

Lemma 5.9 *Let P be an $\mathrm{L}_{[\![\pi]\!]}$-process.*

1. *If $P \xrightarrow{\overline{a}(c)} P_1$ then $P \equiv (\boldsymbol{\nu}\widetilde{z})((\boldsymbol{\nu}c)(\overline{a}c \mid c \rightarrow b) \mid P_2)$ and $P_1 \equiv (\boldsymbol{\nu}\widetilde{z})(c \rightarrow b \mid P_2)$ for some $\widetilde{z}$, b and P_2 such that $\{a, c\} \cap \widetilde{z} = \emptyset$ and $c \notin \mathrm{fn}(P_2)$.*

2. *If $P \xrightarrow{\overline{a}(c)} P_1$ then $P \approx (\boldsymbol{\nu}c)([\![\overline{a}c]\!] \mid P_1)$.*

Proof:

1. By transition induction.

2. By part 1 $P \equiv (\boldsymbol{\nu}\widetilde{z})((\boldsymbol{\nu}c)(\overline{a}c \mid c \rightarrow b) \mid P_2)$ and $P_1 \equiv (\boldsymbol{\nu}\widetilde{z})(c \rightarrow b \mid P_2)$ for some $\widetilde{z}$, b and P_2 such that $\{a, c\} \cap \widetilde{z} = \emptyset$ and $c \notin \mathrm{fn}(P_2)$. Picking some fresh name d we have, using proposition 4.3(2):

$$\begin{aligned} P &\equiv (\boldsymbol{\nu}\widetilde{z})((\boldsymbol{\nu}d)(\overline{a}d \mid d \rightarrow b) \mid P_2) \\ &\approx (\boldsymbol{\nu}\widetilde{z})((\boldsymbol{\nu}d)(\overline{a}d \mid (\boldsymbol{\nu}c)(d \rightarrow c \mid c \rightarrow b)) \mid P_2) \\ &\equiv (\boldsymbol{\nu}c)((\boldsymbol{\nu}d)(\overline{a}d \mid d \rightarrow c) \mid (\boldsymbol{\nu}\widetilde{z})(c \rightarrow b \mid P_2)) \\ &\equiv (\boldsymbol{\nu}c)([\![\overline{a}c]\!] \mid P_1). \end{aligned}$$

$\Box$

Lemma 5.10 *Let P and Q be two $\mathrm{L}_{[\![\pi]\!]}$-processes such that $P \precsim_{\mathrm{a}} Q$. Then:*

1. *$(\boldsymbol{\nu}a)P \precsim_{\mathrm{a}} (\boldsymbol{\nu}a)Q$*

2. *$P \mid R \precsim_{\mathrm{a}} Q \mid R$, for all $\mathrm{L}_{[\![\pi]\!]}$-process R .*

Proof: Here we give a sketch of the proof. A detailed proof can be found in Appendix A.1

1. We show that the relation

$$\mathcal{S}= \{((\boldsymbol{\nu} a)P\,,\,(\boldsymbol{\nu} a)Q) :\; P,Q \in \mathrm{L}_{[\![\pi]\!]} \text{ and } P \asymp_{\mathrm{a}} Q\}$$

is a $\asymp_{\mathrm{a}}$-bisimulation up-to structural congruence. The proof is straightforward because the output actions performed by an $\mathrm{L}_{[\![\pi]\!]}$-process are always bound and therefore the restriction operator cannot bind names in output object position. We work up to structural congruence when dealing with the asynchronous clause for input.

2. We prove that the relation

$$\mathcal{S} = \{((\boldsymbol{\nu}\tilde{a})(P \mid R), (\boldsymbol{\nu}\tilde{a})(Q \mid R)) :\; P,Q,R \in \mathrm{L}_{[\![\pi]\!]} \text{ and } P \asymp_{\mathrm{a}} Q\}$$

is a $\asymp_{\mathrm{a}}$-bisimulation up to $\gtrsim$ and $\approx$. The proof requires both the up-to proof technique of Definition 5.7 and Lemma 5.9. In general, proving that a ground bisimulation is preserved by parallel composition is hard. In our case the proof is simple because processes in $\mathrm{L}_{[\![\pi]\!]}$ never perform free output actions.

□

In Lemma 5.11 we prove that $\asymp_{\mathrm{a}}$ is an equivalence relation. Notice that the proof of transitivity relies on Lemma 5.10(2).

Lemma 5.11 *Let P, Q and R be $\mathrm{L}_{[\![\pi]\!]}$-processes. Then:*

1. $P \asymp_{\mathrm{a}} P$
2. $P \asymp_{\mathrm{a}} Q$ *implies* $Q \asymp_{\mathrm{a}} P$
3. $P \asymp_{\mathrm{a}} Q$ *and* $Q \asymp_{\mathrm{a}} R$ *implies* $P \asymp_{\mathrm{a}} R$.

Proof: The only nontrivial property is transitivity. We essentially need to prove two results:

- $\asymp_{\mathrm{a}}$ is preserved by injective substitutions. The proof is analogous to that of $\approx$ in the π-calculus.
- $P \asymp_{\mathrm{a}} Q$ and $b \notin \mathrm{fn}(P,Q)$ implies $P \mid [\![\overline{a}b]\!] \asymp_{\mathrm{a}} Q \mid [\![\overline{a}b]\!]$. This result is necessary to deal with clause (3b) of Definition 5.6 and is just a particular case of Lemma 5.10(2). Notice that the proof of Lemma 5.10(2) does not rely on the transitivity of $\asymp_{\mathrm{a}}$.

□

Being in ground style, the bisimilarity $\asymp_{\mathrm{a}}$ is preserved by input prefixing.

Lemma 5.12 *Let P and Q be two $\mathrm{L}_{[\![\pi]\!]}$-processes such that $P \asymp_{\mathrm{a}} Q$. Then:*

1. $a(x).\,P \asymp_{\mathrm{a}} a(x).\,Q$
2. $!a(x).\,P \asymp_{\mathrm{a}} !a(x).\,Q$.

Proof: Here we give a sketch of the proof. A detailed proof can be found in Appendix A.1

1. We prove that the relation

$$\mathcal{S}= \{(a(x).\,P\,, a(x).\,Q) :\; P,Q \in \mathrm{L}_{[\![\pi]\!]} \text{ and } P \asymp_{\mathrm{a}} Q\}$$

is a $\asymp_{\mathrm{a}}$-bisimulation. Since $\asymp_{\mathrm{a}}$ is in ground style it suffices to prove that $\asymp_{\mathrm{a}}$ is preserved by injective substitution. The proof is analogous to that of $\approx$ in the π-calculus.

2. We prove that the relation $\mathcal{S}$ defined below

$$\{(P_1 \mid !a(x).\, Q_1, P_2 \mid !a(x).\, Q_2) : P_1, P_2, Q_1, Q_2 \in \mathrm{L}_{[\![\pi]\!]}\,,\, P_1 \succeq_{\mathrm{a}} P_2\,,\, Q_1 \succeq_{\mathrm{a}} Q_2\}$$

is a $\succeq_{\mathrm{a}}$-bisimulation up to structural congruence (it is a special case of the proof technique of Definition 5.7). The proof relies on the fact that $\succeq_{\mathrm{a}}$ is preserved by injective substitutions, parallel composition, and restriction. By transitivity of $\succeq_{\mathrm{a}}$ (Lemma 5.11(3)) the conclusion follows.

□

Using Lemmas 5.10, 5.11, and 5.12 we can prove that $\succeq_{\mathrm{a}}$ is a congruence.

Corollary 5.13 $\succeq_{\mathrm{a}}$ *is a congruence relation.*

Finally, we are ready to prove the main result of the section, that is the soundness of our "translation-based" technique for barbed congruence in Lπ.

Theorem 5.14 (Soundness) *Let P and Q be two processes in* Lπ. *Then:*

$$[\![P]\!] \succeq_{\mathrm{a}} [\![Q]\!] \text{ implies } P \cong_{\mathrm{L}\pi} Q.$$

Proof: By Lemmas 5.4 and Corollary 5.13 it holds that $[\![C[P]]\!] \succeq_{\mathrm{a}} [\![C[Q]]\!]$ for any context $C[\cdot]$ in Lπ. Since $\succeq_{\mathrm{a}} \subset \dot{\approx}$, it holds that $[\![C[P]]\!] \dot{\approx} [\![C[Q]]\!]$ and therefore, by Theorem 5.3, $C[P] \dot{\approx} C[Q]$. □

An easy corollary of the result above is:

Corollary 5.15 *Let P and Q be two processes in* Lπ. *Then:*

$$[\![P]\!] \approx [\![Q]\!] \text{ implies } P \cong_{\mathrm{L}\pi} Q.$$

Proof: $\approx$ implies $\succeq_{\mathrm{a}}$. By applying Theorem 5.14 we get the result. □

Remark 5.16 *Theorem 5.14 (resp. Corollary 5.15) does not hold when replacing $\succeq_{\mathrm{a}}$ (resp. $\approx$) and barbed congruence with their expansion variants. This because, as we will see in the next section, the encoding $[\![\cdot]\!]$ may introduce a few τ-steps changing the balance of silent moves between two processes. Had such an expansion variant of Theorem 5.14 (and Corollary 5.15) been available, then proof of a key result in Section 9.3 would have been much simpler (see Remark 9.11).*

The encoding $[\![\cdot]\!]$, and therefore the proof technique of Theorem 5.14, is essentially based on the law below, relating free and bound output actions:

$$\overline{a}b \cong_{\mathrm{L}\pi} (\boldsymbol{\nu} c)(\overline{a}c \mid c \rightarrow b) \tag{6}$$

However, in Lπ, an even simpler law holds which relates free and bound output actions. We recall that $c \triangleright b$ denotes the static link of Definition 4.1.

Lemma 5.17 *Let a and b be two names different from c. Then:*

$$\overline{a}b \cong_{\mathrm{L}\pi} (\boldsymbol{\nu} c)(\overline{a}c \mid c \triangleright b).$$

Proof: We code up, via $[\![\cdot]\!]$, both members of the equation obtaining, respectively:

$$(\boldsymbol{\nu} d)(\overline{a}d \mid d \rightarrow b) \text{ and } (\boldsymbol{\nu} c)((\boldsymbol{\nu} d)(\overline{a}d \mid d \rightarrow c) \mid c \rightarrow b).$$

By Proposition 4.3(2) the two processes above are synchronous bisimilar. By Corollary 5.15, the conclusion follows. □

From Lemma 5.17 we can derive a "translation based" proof technique for barbed congruence simpler than that given by Theorem 5.14, whereby in the encoding $[\![\cdot]\!]$, static links $c \triangleright b$ are used instead of dynamic links $c \rightarrow b$. However, as we will prove in Section 7, this proof technique is sound but not complete, that is, it does not completely describe barbed congruence.

$$\texttt{free-out:}\ \frac{P \xrightarrow{\overline{a}b} P' \qquad p \notin \text{fn}(P)}{P \overset{\overline{a}(p)}{\longmapsto} (p \rhd b \mid P')} \qquad\qquad \texttt{bound-out:}\ \frac{P \xrightarrow{\overline{a}(b)} P' \qquad p \notin \text{fn}(P)}{P \overset{\overline{a}(p)}{\longmapsto} (\boldsymbol{\nu} b)(p \rhd b \mid P')}$$

$$\texttt{sync:}\ \frac{P \xrightarrow{\tau} P'}{P \overset{\tau}{\longmapsto} P'} \qquad\qquad \texttt{input:}\ \frac{P \xrightarrow{a(b)} P'}{P \overset{a(b)}{\longmapsto} P'}$$

Table 3: A new labelled transition system for Lπ

6 A "run-time" proof technique

The main drawback of the proof technique of Theorem 5.14 is that the encoding $[\![\cdot]\!]$ adds a dynamic link to all outputs in the source processes, slowing down computation. A process $[\![P]\!]$ will normally take more steps to simulate the computation of P. For instance, if P is

$$\overline{a}b \mid a(x).\, \overline{x}c \mid b(z).\, \overline{z}w$$

then P needs two τ-steps before producing an output at c, whereas its image

$$[\![\overline{a}b]\!] \mid a(x).\, [\![\overline{x}c]\!] \mid b(z).\, [\![\overline{z}w]\!]$$

will take 5 steps. We conjecture that in Lπ the encoding $[\![\cdot]\!]$ does not introduce divergences, that is, infinite internal computations. Notice that if we would add full replication in Lπ the encoding would not be divergence-free. As an example, take the process

$$P \stackrel{\text{def}}{=} \overline{a}b \mid a(x).\, !\overline{x}c$$

This process does not diverge, but $[\![P]\!]$ does.

In this section, we introduce a more powerful technique based on a new LTS $\overset{\mu}{\longmapsto}$ for Lπ. The new LTS, reported in Table 3, is defined on top of the original one, and transforms the output of a name b into the output of a fresh *pointer* p to b. We call p a pointer to b because a static link $p \rhd b$ is introduced through which any output along p is redirected onto b. The new LTS makes explicit the constraint that in Lπ only the output capability of names may be transmitted, by transforming the occurrence of a name in output object position with an occurrence of the same name in output subject position.

The weak transitions $\overset{\mu}{\Longmapsto}$ and $\Longmapsto$ for the new LTS are defined from the strong transitions $\overset{\mu}{\longmapsto}$ and $\overset{\tau}{\longmapsto}$ in the usual way. We write $\overset{\mapsto}{\approx}_{\text{a}}$ to denote the relation obtained by replacing in the definition of asynchronous ground bisimulation (Definition 3.4) arrow $\longrightarrow$ with $\longmapsto$ and arrow $\Longrightarrow$ with $\Longmapsto$. We prove that asynchronous ground bisimulation defined on the new LTS implies barbed congruence. To avoid reasoning with two LTSs, we reformulate the definition of $\overset{\mapsto}{\approx}_{\text{a}}$ on the original LTS.

Definition 6.1 (Link bisimilarity) *A symmetric relation $\mathcal{S}$ on* Lπ*-processes is a* link bisimulation *if whenever $P \;\mathcal{S}\; Q$ the following holds:*

1. *If $P \xrightarrow{\tau} P'$ then $Q \Longrightarrow Q'$ and $P' \;\mathcal{S}\; Q'$.*
2. *If $P \xrightarrow{a(p)} P'$, and $p \notin \text{fn}(Q)$, then*
   (a) *either $Q \overset{a(p)}{\Longrightarrow} Q'$ and $P' \;\mathcal{S}\; Q'$*
   (b) *or $Q \Longrightarrow Q'$ and $P' \mathcal{S}\, (Q' \mid \overline{a}p)$.*

3. *If* $P \xrightarrow{\overline{a}b} P'$, *then*

   (a) *either* $Q \stackrel{\overline{a}d}{\Longrightarrow} Q'$ *and* $\exists\, p \notin \mathrm{fn}(P,Q)$ *such that* $(p \rhd b \mid P')\ \mathcal{S}\ (p \rhd d \mid Q')$

   (b) *or* $Q \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$, *with* $c \notin \mathrm{fn}(P)$, *and* $\exists\, p \notin \mathrm{fn}(P,Q)$ *such that* $(p \rhd b \mid P')\ \mathcal{S}\ (\boldsymbol{\nu} c)(p \rhd c \mid Q')$.

4. *If* $P \xrightarrow{\overline{a}(c)} P'$, *with* $c \notin \mathrm{fn}(Q)$, *then*

   (a) *either* $Q \stackrel{\overline{a}b}{\Longrightarrow} Q'$ *and* $\exists\, p \notin \mathrm{fn}(P,Q)$ *such that* $(\boldsymbol{\nu} c)(p \rhd c \mid P')\ \mathcal{S}\ (p \rhd b \mid Q')$

   (b) *or* $Q \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$, $\exists\, p \notin \mathrm{fn}(P,Q)$ *such that* $(\boldsymbol{\nu} c)(p \rhd c \mid P')\ \mathcal{S}\ (\boldsymbol{\nu} c)(p \rhd c \mid Q')$.

P *and* Q *are* link bisimilar*, written* $P \approx_{\mathrm{l}} Q$*, if* $P\ \mathcal{S}\ Q$ *for some link bisimulation* $\mathcal{S}$*.*

The only difference between link bisimilarity and asynchronous ground bisimilarity is in the clauses for output actions: In link bisimilarity the name emitted by the two processes may be different. To mask this difference, links are added in the derivatives. It is immediate to see that $\overset{\mapsto}{\approx}_{\mathrm{a}}$ and $\approx_{\mathrm{l}}$ coincide.

Lemma 6.2 *Let* P *and* Q *be two processes in* $\mathrm{L}\pi$*. Then:*

$$P \overset{\mapsto}{\approx}_{\mathrm{a}} Q \ \text{ iff } \ P \approx_{\mathrm{l}} Q.$$

The following lemma relates the translation-based and the run-time proof techniques.

Lemma 6.3 *Let* P *and* Q *be two processes in* $\mathrm{L}\pi$*. Then:*

$$P \approx_{\mathrm{l}} Q \ \text{ iff } \ [\![P]\!] \asymp_{\mathrm{a}} [\![Q]\!].$$

Proof: We give a sketch of the proof. A detailed proof can be found in Appendix A.2. In the implication from left to right, we use a variant of the operational correspondence between processes P and $[\![P]\!]$ to show that the relation

$$\mathcal{S} = \{([\![P]\!], [\![Q]\!]) \mid \ P, Q \in \mathrm{L}\pi \ \text{ and } \ P \approx_{\mathrm{l}} Q\}$$

is a $\asymp_{\mathrm{a}}$-bisimulation up-to $\gtrsim$ and $\approx$. In the implication from right to left, we show that the relation

$$\mathcal{S} = \{(P, Q) \mid P, Q \in \mathrm{L}\pi \ \text{ and } \ [\![P]\!] \asymp_{\mathrm{a}} [\![Q]\!]\}$$

is a link bisimulation. $\square$

Finally, we prove the soundness of the "run-time" proof technique.

Theorem 6.4 (Soundness) *Let* P *and* Q *be two processes in* $\mathrm{L}\pi$*. Then*

$$P \overset{\mapsto}{\approx}_{\mathrm{a}} Q \ \text{ implies } \ P \cong_{\mathrm{L}\pi} Q.$$

Proof: By applying, in sequence, Lemma 6.2, Lemma 6.3, and Theorem 5.14. $\square$

Both the "translation-based" and the "run-time" proof techniques are based on the use of links. In the former, links are added statically via an encoding (at "compile-time"); in the latter, they are added dynamically in the bisimulation game (at "run-time"). The advantages of the latter proof-technique are that: (i) it uses simpler links $p \rhd b$ instead of links $p \to b$; (ii) links are not added in case of internal communications; (iii) the input clause uses the particle $\overline{a}b$ instead of $[\![\overline{a}b]\!]$ (that produces links); (iv) in the latter proof technique, the number of added links may be further reduced using the *up to link* proof technique (see Section 8).

7 Two characterisations of barbed congruence

In this section we show that the translation-based and the run-time proof techniques described in Section 5 and 6, respectively, are not only sound but also complete, that is, they completely characterise barbed congruence in Lπ. As usual when proving labelled characterisations of barbed congruence [3, 49], we prove the completeness for image-finite processes (see Definition 3.2). The *challenge* here is that, unlike similar results in the literature, our calculus does not provide any construct for testing equality between names (such as matching).

Lemma 7.1 *Let P and Q be two image-finite* Lπ*-processes. Then*

$$P \cong_{\mathrm{L}\pi} Q \ \text{ implies } \ P \approx_{\mathrm{l}} Q.$$

Proof: See Appendix A.3. Compared to the proofs of similar results in the literature, the non-standard case is when the tested process perform an output. $\Box$

By Lemma 7.1 we can derive the completeness of our two proof techniques.

Theorem 7.2 (First characterisation) *Let P and Q be two processes in* Lπ*. Then:*

1. *$P \cong_{\mathrm{L}\pi} Q$ implies $[\![P]\!] \asymp_{\mathrm{a}} [\![Q]\!]$, for P and Q image-finite;*
2. *$[\![P]\!] \asymp_{\mathrm{a}} [\![Q]\!]$ implies $P \cong_{\mathrm{L}\pi} Q$.*

Proof: Part 1 follows by applying in sequence Lemmas 7.1 and 6.3. Part 2 follows from Theorem 5.14 $\Box$

Our translation-based proof technique relies on the algebraic law

$$\overline{a}b \ \cong_{\mathrm{L}\pi} \ (\boldsymbol{\nu} c)(\overline{a}c \mid c \rightarrow b) \tag{7}$$

As already pointed out in Section 5, a simpler translation $[\![\cdot]\!]$ can be defined by replacing this law with:

$$\overline{a}b \ \cong_{\mathrm{L}\pi} \ (\boldsymbol{\nu} c)(\overline{a}c \mid c \rhd b) \tag{8}$$

(we recall that $c \rhd b \stackrel{\mathrm{def}}{=} !c(x).\, \overline{b}x$). The encoding $[\![\cdot]\!]$ obtained by using Law 8 instead of 7 is still sound but it *is not* complete. As a counterexample, take the processes:

$$\begin{array}{lcl} P & \stackrel{\mathrm{def}}{=} & \overline{a}b \\ Q & \stackrel{\mathrm{def}}{=} & (\boldsymbol{\nu} c)(\overline{a}c \mid c \rightarrow b) = (\boldsymbol{\nu} c)(\overline{a}c \mid !c(x).\, (\boldsymbol{\nu} d)(\overline{b}d \mid d \rightarrow x)). \end{array}$$

Then, $P \cong_{\mathrm{L}\pi} Q$ whereas $[\![P]\!]$ and $[\![Q]\!]$ are not related by either $\asymp_{\mathrm{a}}$ or $\approx$. Indeed,

$$[\![P]\!] \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu} h)(\overline{a}h \mid !h(x).\, \overline{b}x)$$

$$[\![Q]\!] \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu} c)((\boldsymbol{\nu} h)(\overline{a}h \mid !h(x).\, \overline{c}x) \mid !c(x).\, (\boldsymbol{\nu} d)((\boldsymbol{\nu} r)(\overline{b}r \mid !r(x).\, \overline{d}x) \mid [\![d \rightarrow x]\!])).$$

and both relations $\asymp_{\mathrm{a}}$ and $\approx$ can distinguish the two processes after performing two visible actions. More precisely, $[\![P]\!] \xrightarrow{\overline{a}(h)} \xrightarrow{h(s)} \xrightarrow{\overline{b}s}$ whereas $[\![Q]\!] \xrightarrow{\overline{a}(h)} \xrightarrow{h(s)} \stackrel{\overline{b}(r)}{\Longrightarrow}$.

Now, we prove that the link bisimilarity, i.e. the "run-time" proof technique, is a complete characterisation of barbed congruence in Lπ.

Theorem 7.3 (Second characterisation) *Let P and Q be two processes in* Lπ*. Then:*

1. *$P \cong_{\mathrm{L}\pi} Q$ implies $P \overset{\mapsto}{\approx}_{\mathrm{a}} Q$, for P and Q image-finite processes;*
2. *$P \overset{\mapsto}{\approx}_{\mathrm{a}} Q$ implies $P \cong_{\mathrm{L}\pi} Q$.*

Proof: Part 1 follows by applying in sequence Lemmas 7.1 and 6.2. Part 2 follows from Theorem 6.4. □

Theorem 7.3 says that, on image-finite processes, asynchronous ground bisimilarity, defined on the new LTS, coincides with barbed congruence in Lπ. We recall that, in presence of matching, the closure under substitutions of asynchronous early bisimilarity on the LTS $\xrightarrow{\mu}$ coincides with barbed congruence in π_{a} [3]. Therefore, somehow, the difference between the two LTSs essentially shows the difference between what is observable in Lπ and what is observable in π_{a} with matching.

8 Up-to-link proof techniques

In this section we enhance our labelled proof techniques for barbed congruence by means of both standard (cf. Definition 8.1) and new (cf. Definition 8.3) up-to proof techniques.

We start with the standard one, i.e. the up-to-expansion proof technique [59].

Definition 8.1 (Link bisimilarity up to expansion) *A symmetric relation $\mathcal{S}$ on* Lπ*-processes is a* link bisimulation up to $\gtrsim$ *if whenever $P \; \mathcal{S} \; Q$ the following holds:*

1. *If $P \xrightarrow{\tau} P'$, then $Q \Longrightarrow Q'$ and $P' \gtrsim \mathcal{S} \lesssim Q'$.*
2. *If $P \xrightarrow{a(p)} P'$, and $p \notin \mathrm{fn}(Q)$, then*
   (a) *either $Q \stackrel{a(p)}{\Longrightarrow} Q'$ and $P' \gtrsim \mathcal{S} \lesssim Q'$*
   (b) *or $Q \Longrightarrow Q'$ and $P' \gtrsim \mathcal{S} \lesssim (Q' \mid \overline{a}p)$.*
3. *If $P \xrightarrow{\overline{a}b} P'$, and $p \notin \mathrm{fn}(P, Q)$, then*
   (a) *either $Q \stackrel{\overline{a}d}{\Longrightarrow} Q'$ and $(p \rhd b \mid P') \gtrsim \mathcal{S} \lesssim (p \rhd d \mid Q')$*
   (b) *or $Q \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$, with $c \notin \mathrm{fn}(P)$, and $(p \rhd b \mid P') \gtrsim \mathcal{S} \lesssim (\boldsymbol{\nu} c)(p \rhd c \mid Q')$.*
4. *If $P \xrightarrow{\overline{a}(c)} P'$, with $c \notin \mathrm{fn}(Q)$ and $p \notin \mathrm{fn}(P, Q)$, then*
   (a) *either $Q \stackrel{\overline{a}b}{\Longrightarrow} Q'$ and $(\boldsymbol{\nu} c)(p \rhd c \mid P') \gtrsim \mathcal{S} \lesssim (p \rhd b \mid Q')$*
   (b) *or $Q \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$ and $(\boldsymbol{\nu} c)(p \rhd c \mid P') \gtrsim \mathcal{S} \lesssim (\boldsymbol{\nu} c)(p \rhd c \mid Q')$.*

Lemma 8.2 *If $\mathcal{S}$ is a link bisimilarity up to expansion then $\mathcal{S} \subseteq \approx_{\mathrm{l}}$.*

Proof: The proof is analogous to that of standard bisimilarity [59]. □

The following definition provides a new and more powerful up-to proof technique which allows us to reduce the number of links introduced in the derivatives. Roughly speaking, when comparing two processes P and Q this technique allows us to cut the *same* links from both processes, or to cut a *private* link from one process only. For instance, when proving that $\overline{a}b$ and $(\boldsymbol{\nu} c)(\overline{a}c \mid c \rhd b)$ are link bisimilar we have to prove that the derivatives $p \rhd b$ and $(\boldsymbol{\nu} c)(p \rhd c \mid c \rhd b)$ are also link bisimilar. Using our new proof technique we simply require that $p \rhd b$ is related to itself (cf. Clause 3. a. i of Definition 8.3).

Definition 8.3 (Link bisimulation up to link) *A symmetric relation $\mathcal{S}$ on* Lπ*-processes is a* link bisimulation up to link *if whenever $P \; \mathcal{S} \; Q$ the following holds:*

1. *If $P \xrightarrow{\tau} P'$ then $Q \Longrightarrow Q'$ and $P' \; \mathcal{S} \; Q'$.*
2. *If $P \xrightarrow{a(p)} P'$, and $p \notin \mathrm{fn}(Q)$, then*

(a) *either* $Q \stackrel{a(p)}{\Longrightarrow} Q'$ *and* $P' \mathcal{S}\, Q'$

(b) *or* $Q \Longrightarrow Q'$ *and* $P' \mathcal{S}\, (Q' \mid \overline{a}p)$.

3. *If* $P \xrightarrow{\overline{a}b} P'$, *and* $p \notin \mathrm{fn}(P, Q)$, *then*

   (a) *either* $Q \stackrel{\overline{a}b}{\Longrightarrow} Q'$ *and* $P' \mathcal{S}\, Q'$

   (b) *or* $Q \stackrel{\overline{a}d}{\Longrightarrow} Q'$, *with* $d \neq b$, *and* $(p \triangleright b \mid P')\, \mathcal{S}\, (p \triangleright d \mid Q')$

   (c) *or* $Q \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$, *with* $c \notin \mathrm{fn}(P)$, *and*

      i. *either* $(p \triangleright b \mid P')\, \mathcal{S}\, Q'\{p/c\}$

      ii. *or* $(p \triangleright b \mid P')\, \mathcal{S}\, (\boldsymbol{\nu} c)(p \triangleright c \mid Q')$.

4. *If* $P \xrightarrow{\overline{a}(c)} P'$, *with* $c \notin \mathrm{fn}(Q)$ *and* $p \notin \mathrm{fn}(P, Q)$, *then*

   (a) *either* $Q \stackrel{\overline{a}b}{\Longrightarrow} Q'$ *and*

      i. *either* $P'\{p/c\}\, \mathcal{S}\, (p \triangleright b \mid Q')$

      ii. *or* $(\boldsymbol{\nu} c)(p \triangleright c \mid P')\, \mathcal{S}\, (p \triangleright b \mid Q')$

   (b) *or* $Q \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$ *and*

      i. *either* $P' \mathcal{S}\, Q'$

      ii. *or* $P'\{p/c\}\, \mathcal{S}\, (\boldsymbol{\nu} c)(p \triangleright c \mid Q')$

      iii. *or* $(\boldsymbol{\nu} c)(p \triangleright c \mid P')\, \mathcal{S}\, Q'\{p/c\})$.

      iv. *or* $(\boldsymbol{\nu} c)(p \triangleright c \mid P')\, \mathcal{S}\, (\boldsymbol{\nu} c)(p \triangleright c \mid Q')$.

Two processes P *and* Q *are link bisimilar up to link, written* $P \approx_{\mathrm{lut}} Q$, *if* $P\, \mathcal{S}\, Q$ *for some link bisimulation up to link* $\mathcal{S}$.

The up-to-link technique is inspired by the up-to-expansion [59] and the up-to-context [50] techniques (in which common contexts are factorised out). However, it cannot be reduced to them because private links may be cut from only one process and not from both of them as required by the up-to-context technique. Of course, the up-to-link technique can be used in combination with up-to-expansion and up-to-context techniques. We recall that in up-to proof techniques we need not always apply a cut for reducing the size of the derivative processes. This explains why we have the subclauses marked by $i. \ldots iv.$

Finally, we prove that the up-to-link proof technique is sound and complete. The proof of the completeness is non trivial.

Theorem 8.4 *Let* P *and* Q *be two* Lπ*-processes. Then:*

$$P \approx_{\mathrm{l}} Q \text{ iff } P \approx_{\mathrm{lut}} Q.$$

Proof: We give a sketch of the proof. Details can be found in Appendix A.4. In the implication from left to the right, we prove that the relation

$$\mathcal{S} = \{(P, Q) : P \approx_{\mathrm{l}} Q\}$$

is a link bisimulation up to link. The only interesting point in the proof is when dealing with the clause in which a free output action $\overline{a}b$ is matched by the same free output action. In this case we need a result (see Lemma A.4) saying that if

$$p \notin \mathrm{fn}(P, Q) \text{ and } (p \triangleright b \mid P) \approx_{\mathrm{l}} (p \triangleright b \mid Q) \text{ then } P \approx_{\mathrm{l}} Q.$$

In the implication from right to left we prove that the relation

$$\mathcal{S} = \{(P, Q) : P \approx_{\mathrm{lut}} Q\}$$

is a link bisimulation. The proof requires the following results (see Lemma A.6):

1. $P \approx_{\text{lut}} Q$ and $p \notin \text{fn}(P, Q)$ implies $(p \triangleright b \mid P) \approx_{\text{lut}} (p \triangleright b \mid Q)$.
2. $P \approx_{\text{lut}} Q$ and $p \notin \text{fn}(P, Q)$ implies $(\boldsymbol{\nu} c)(p \triangleright c \mid P) \approx_{\text{lut}} (\boldsymbol{\nu} c)(p \triangleright c \mid Q)$.
3. $P\{p/c\} \approx_{\text{lut}} (\boldsymbol{\nu} c)(p \triangleright c \mid Q)$ and $p \notin \text{fn}(P, Q)$ implies $(\boldsymbol{\nu} c)(p \triangleright c \mid P) \approx_{\text{lut}} (\boldsymbol{\nu} c)(p \triangleright c \mid Q)$.
4. $P\{p/c\} \approx_{\text{lut}} (p \triangleright b \mid Q)$ and $c \notin \text{fn}(Q)$ implies $(\boldsymbol{\nu} c)(p \triangleright c \mid P) \approx_{\text{lut}} (p \triangleright b \mid Q)$.

□

9 Applications of the theory of Localised π

In this section we show a certain number of applications of the theory of Lπ.

9.1 Expressing substitution in Lπ

In π-calculus, the process $P\{b/a\}$ can be expressed as $(\boldsymbol{\nu} u)(\overline{u}b \mid u(a).\, P)$. In L$\pi$, there exists an alternative way to express name substitution which presents a few advantages (see Sections 9.5 and 9.7, and [32]).

Proposition 9.1 *Let P be an Lπ-process and a and b two names such that $a \neq b$ and a does not appear free in P in input position. Then:*

$$(\boldsymbol{\nu} a)(P \mid a \triangleright b) \cong_{\text{L}\pi} P\{b/a\}.$$

Proof: We code up, via $[\![\cdot]\!]$ (the encoding in Section 5), both members. By Lemma 5.2 we get

$$[\![(\boldsymbol{\nu} a)(P \mid a \triangleright b)]\!] = (\boldsymbol{\nu} a)([\![P]\!] \mid a \to b) \gtrsim [\![P]\!]\{b/a\} = [\![P\{b/a\}]\!].$$

Since $\gtrsim$ implies $\approx$, we use Corollary 5.15 to compete the proof. □

The law in Proposition 9.1 is valid in Lπ but not in π_{a} and π-calculus. A similar law, but with the double link $a \triangleright b \mid b \triangleright a$ in place of $a \triangleright b$, is given in [26] for the asynchronous π-calculus.

We exploit Proposition 9.1 to give an easy proof that early and ground link bisimilarities coincide in Lπ. We denote with $\approx_{\text{l}}^{\text{e}}$ the early variant of $\approx_{\text{l}}$, obtained by replacing input clause in Definition 6.1 with

2. $P \xrightarrow{a(x)} P'$ implies that for all b, there exists Q' such that:

   (a) either $Q \overset{a(x)}{\Longrightarrow} Q'$ and $P'\{b/x\} \;\mathcal{S}\; Q'\{b/x\}$
   (b) or $Q \Longrightarrow Q'$ and $P'\{b/x\} \;\mathcal{S}\; (Q' \mid \overline{a}b)$.

Proposition 9.2 $P \approx_{\text{l}}^{\text{e}} Q$ iff $P \approx_{\text{l}} Q$.

Proof: The implication from left to right is easy. An early bisimilarity has a universal quantification on the received names, whereas a ground bisimilarity has an existential quantification. Hence an early bisimilarity is included in its ground variant. For the implication from right to left, we prove that the relation

$$\mathcal{S} = \{(P, Q) : \; P, Q \in \text{L}\pi, \; P \approx_{\text{l}} Q\}$$

is an early link bisimulation. We focus on the input clause because this is the only difference between the two bisimilarities. Suppose $P \xrightarrow{a(x)} P'$. We want to prove that for every b there exists Q' such that $Q \overset{a(x)}{\Longrightarrow} Q'$ and $P'\{b/x\} \;\mathcal{S}\; Q'\{b/x\}$. The case $b = x$ is trivial, so we suppose

$b \neq x$. Since $P \approx_1 Q$, there exists Q' such that $Q \xRightarrow{a(x)} Q'$ and $P' \approx_1 Q'$. Notice that, since P' and Q' are Lπ-processes, x does not appear free in P' and Q' in input subject position. By Lemma 6.3, and Corollary 5.13 link bisimilarity is an equivalence relation and it is preserved by all operators of the calculus. So, $(\boldsymbol{\nu} x)(P' \mid x \triangleright b) \approx_1 (\boldsymbol{\nu} x)(Q' \mid x \triangleright b)$ for every name b. By Lemma 5.2 and Lemma 6.3 it holds that $P'\{b/x\} \approx_1 (\boldsymbol{\nu} x)(P' \mid x \triangleright b)$ and $Q'\{b/x\} \approx_1 (\boldsymbol{\nu} x)(Q' \mid x \triangleright b)$. By transitivity we have that $P'\{b/x\} \approx_1 Q'\{b/x\}$ and therefore $P'\{b/x\} \mathrel{\mathcal{S}} Q'\{b/x\}$. $\square$

9.2 The replication theorems

The *replication theorems* [34] express useful distributivity properties of private replicated processes. The assertions of the theorems can be read thus: A passive resource that is shared among a certain number of clients can be made private to each of them.

Theorem 9.3 (Standard replication theorems) *Assume that name a occurs free in processes P, Q and R only in output subject position. Then:*

1. $(\boldsymbol{\nu} a)\big(!a(x).\, R \mid P \mid Q\big) \approx (\boldsymbol{\nu} x)(!a(x).\, R \mid P) \mid (\boldsymbol{\nu} a)(!a(x).\, R \mid Q)$.
2. $(\boldsymbol{\nu} a)(!a(x).\, R \mid !P) \approx !(\boldsymbol{\nu} a)(!a(x).\, R \mid P)$.

The side condition in the theorems prevents the restricted name a from being exported. As a consequence, the theorem cannot be used in situations where the set of clients of the resource $a(x).\, R$ may change dynamically. To see why this side condition is necessary, take:

$$\begin{array}{lcl} P_1 & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} a)(!a(x).\, R \mid \bar{b}a \mid Q) \\ P_2 & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} a)(!a(x).\, R \mid \bar{b}a) \mid (\boldsymbol{\nu} a)(!a(x).\, R \mid Q) \end{array}$$

These processes are in general not equivalent in π-calculus. Intuitively, the environment external to P_1 can receive a along b and then use it in input position to interfere with an attempt by Q to activate a copy of R. This is not possible in P_2, where Q has its own private access to R. The difference between P_1 and P_2 can be observed in a context that receives a and then uses it in input; in this way, the context may steal messages that were supposed to reach the resource.

In Lπ the above side condition can be relaxed by requiring that the processes P, Q, and R only possess the output capability on a.

Theorem 9.4 (sharpened replication theorems) *Let P, Q, and R be processes in* Lπ *not containing name a in input position. Then:*

1. $(\boldsymbol{\nu} a)\big(!a(x).\, R \mid P \mid Q\big) \cong_{\mathrm{L}\pi} (\boldsymbol{\nu} x)(!a(x).\, R \mid P) \mid (\boldsymbol{\nu} a)(!a(x).\, R \mid Q)$;
2. $(\boldsymbol{\nu} a)(!a(x).\, R \mid !P) \cong_{\mathrm{L}\pi} !(\boldsymbol{\nu} a)(!a(x).\, R \mid P)$.

Proof: We code up, via $[\![\cdot]\!]$, both members of each equation. By definition of $[\![\cdot]\!]$ name a may appear free in the translated terms only in output subject position. Therefore, the translated terms comply with the hypotheses of Theorem 9.3 and we can assert that the images (of each equation) are synchronous ground bisimilar. Since $\approx$ implies $\asymp_{\mathrm{a}}$, by theorem 5.14 we can complete the proof. $\square$

9.3 The delayed input

In an asynchronous calculus message emission is non-blocking. Milner, Parrow, Victor and others have advocated also non-blocking message reception (which is among the motivations for the Update and the Fusion calculi [43, 44] and for the Chi calculus [20]). Such a *delayed input*, written $a(x)P$, should allow the continuation P to evolve underneath the input guard, except for observable actions along x. The delayed input replaces *temporal precedences*, imposed by plain input, with *causal dependencies*. This appears, for instance, in Abramsky's representation of

$$\texttt{d-in:}\ \frac{}{a(b)P \xrightarrow{a(b)} P} \qquad \texttt{s-com:}\ \frac{P \xrightarrow{\overline{a}c} P'}{a(b)P \xrightarrow{\tau} (\boldsymbol{\nu} b)(P'\{c/b\})}$$

$$\texttt{p-in:}\ \frac{P \xrightarrow{\mu} P' \quad b \notin \mathrm{n}(\mu) \quad a \notin \mathrm{bn}(\mu)}{a(b)P \xrightarrow{\mu} a(b)P'} \qquad \texttt{s-cls:}\ \frac{P \xrightarrow{\beta_c \overline{a}c} P' \quad b \notin \mathrm{n}(\beta_c \overline{a}c)}{a(b)P \xrightarrow{\tau} \beta_c(P'\{c/b\})}$$

$$\texttt{o-}\boldsymbol{\nu}\texttt{:}\ \frac{P \xrightarrow{\mu} P' \quad [\mu = \overline{a}c \vee \mu = c(b)\overline{a}b] \quad a \neq c}{(\boldsymbol{\nu} c)P \xrightarrow{(\boldsymbol{\nu} c)\mu} P'} \qquad \texttt{o-in:}\ \frac{P \xrightarrow{\overline{a}b} P' \qquad b \neq a}{c(b)P \xrightarrow{c(b)\overline{a}b} P'}$$

$$\texttt{cls:}\ \frac{P \xrightarrow{\beta_c \overline{a}c} P' \quad Q \xrightarrow{a(b)} Q' \quad \mathrm{bn}(\beta_c) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\tau} \beta_c(P' \mid Q'\{c/b\})}$$

Table 4: New transition rules, for delayed input

$$\{\!|\, P_1 \mid P_2 \,|\!\} \stackrel{\text{def}}{=} \{\!|\, P_1 \,|\!\} \mid \{\!|\, P_2 \,|\!\} \qquad \{\!|\, (\boldsymbol{\nu} a)P \,|\!\} \stackrel{\text{def}}{=} (\boldsymbol{\nu} a)\{\!|\, P \,|\!\} \qquad \{\!|\, \overline{a}b \,|\!\} \stackrel{\text{def}}{=} \overline{a}b$$

$$\{\!|\, a(b).\, P \,|\!\} \stackrel{\text{def}}{=} a(b).\, \{\!|\, P \,|\!\} \qquad \{\!|\, !a(b).\, P \,|\!\} \stackrel{\text{def}}{=} !a(b).\, \{\!|\, P \,|\!\} \qquad \{\!|\, \mathbf{0} \,|\!\} \stackrel{\text{def}}{=} \mathbf{0}$$

$$\{\!|\, a(b)P \,|\!\} \stackrel{\text{def}}{=} (\boldsymbol{\nu} b)(a(b').\, b \triangleright b' \mid \{\!|\, P \,|\!\})$$

Table 5: The encoding $\{\!|\cdot|\!\}$

Linear Logic proofs as π-calculus processes [1, 6]. In this section we prove that the delayed input is a derived operator in Lπ.

Let DLπ be the calculus obtained by adding the delayed input construct $a(b)P$ to the grammar of Lπ (with the same constraint as plain input that b may not appear free in P in input position). Actions are extended as follows:

$$\mu ::= \tau \mid a(b) \mid \overline{a}b \mid \beta_b \overline{a}b$$

where

$$\beta_b ::= (\boldsymbol{\nu} b) \mid c(b) \mid (\boldsymbol{\nu} c)c(b) \text{ for some } c.$$

β_b represents the binding part of bound output actions. We extend free and bound names as follows: $\mathrm{fn}((\boldsymbol{\nu} b)\overline{a}b) = \{a\}$, $\mathrm{bn}((\boldsymbol{\nu} b)\overline{a}b) = \{b\}$, $\mathrm{fn}(c(b)\overline{a}b) = \{c, a\}$, $\mathrm{bn}(c(b)\overline{a}b) = \{b\}$, $\mathrm{fn}((\boldsymbol{\nu} c)c(b)\overline{a}b) = \{a\}$, $\mathrm{bn}((\boldsymbol{\nu} c)c(b)\overline{a}b) = \{c, b\}$. In Table 4, we give the missing transition rules of DLπ. More precisely, we enrich the rules in Table 1 as follows: (i) rule `close` is split into rules `cls` and `s-cls`; (ii) rule `open` is split into rules `o-in` and `o-`$\boldsymbol{\nu}$; (iii) the rules `d-inp`, `s-com`, and `p-inp` are similar to the rules `inp`, `com`, and `par`, but involve the delayed input. Our labelled transition system has two main differences with respect to that of van Breugel's [62]: (i) actions have a simpler syntax, because only the output capability of name may be transmitted; (ii) a restriction $(\boldsymbol{\nu} b)$ is added in rule `s-com` to model *self communications*, as in $a(b)(\overline{a}b \mid P) \xrightarrow{\tau} (\boldsymbol{\nu} b)P$.

In Table 5 we give an encoding $\{\!|\cdot|\!\}$ of DLπ into Lπ and prove that it is fully-abstract with respect to barbed congruence. The encoding $\{\!|\cdot|\!\}$ is an homomorphism on all operators except delayed input. (A similar encoding, but with the double link $b \triangleright b' \mid b' \triangleright b$ in place of $b \triangleright b'$, has been suggested by Yoshida in [66] for the asynchronous π-calculus.)

In order to prove the full abstraction of $\{\!|\cdot|\!\}$, we first prove an adequacy result with respect to barbed bisimulation. The proof of this adequacy result requires an operational correspondence between processes P and $\{\!| P |\!\}$, up to some notion of expansion. As we will argue in Remark 9.11, such an operational correspondence is not easy to prove. Thus, for convenience, we define an auxiliary encoding $\{\![\![\cdot]\!]\!\}$ as the composition of the encodings $\{\!|\cdot|\!\}$ and $[\![\cdot]\!]$ (of Section 5). Precisely, if P is a DLπ-process

$$\{\![\![P]\!]\!\} \stackrel{\text{def}}{=} [\![\{\!| P |\!\}]\!].$$

We prove that the encoding $\{\![\![\cdot]\!]\!\}$ satisfies an operational correspondence up to $\succcurlyeq_{\mathrm{a}}$, where $\preccurlyeq_{\mathrm{a}}$ denotes the expansion variant of $\asymp_{\mathrm{a}}$ (Definition 5.6) in the same way as $\lesssim_{\mathrm{a}}$ denotes the expansion variant of $\approx_{\mathrm{a}}$ (see Definition 3.5). This operational correspondence, together with Theorem 5.3, allows us to prove the soundness of $\{\!|\cdot|\!\}$. Then, by exploiting the inclusion L$\pi \subset$ DLπ we prove the completeness of $\{\!|\cdot|\!\}$.

For proving the operational correspondence of $\{\![\![\cdot]\!]\!\}$ we need to know that $\preccurlyeq_{\mathrm{a}}$ is a precongruence in $\mathrm{L}_{[\![\pi]\!]}$ (simply adapt Corollary 5.13). We also need the following technical lemma.

Lemma 9.5 *Given an* Lπ*-process* P *and a name* a *it holds that*

$$(\boldsymbol{\nu} a)(a \to a \mid [\![P]\!]) \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} a)[\![P]\!].$$

Proof: We have $a \to a \succcurlyeq_{\mathrm{a}} \mathbf{0}$. Then the conclusion follows because $\succcurlyeq_{\mathrm{a}}$ is a precongruence. $\Box$

Lemma 9.6 (Operational correspondence of $\{\![\![\cdot]\!]\!\}$) *Let* P *be a process in* DLπ.

1. *Suppose that* $P \xrightarrow{\mu} P'$. *Then we have:*
   (a) *if* $\mu = a(b)$ *then* $\{\![\![P]\!]\!\} \xrightarrow{a(b')} \gtrsim \{\![\![P']\!]\!\}\{b'/b\}$ *and* $b' \notin \mathrm{fn}(P)$
   (b) *if* $\mu = \overline{a}b$ *then* $\{\![\![P]\!]\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}} (c \to b \mid \{\![\![P']\!]\!\})$, *with* $c \notin \mathrm{fn}(P)$
   (c) *if* $\mu = (\boldsymbol{\nu} b)\overline{a}b$ *then* $\{\![\![P]\!]\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(c \to b \mid \{\![\![P']\!]\!\})$, *with* $c \notin \mathrm{fn}(P)$
   (d) *if* $\mu = d(b)\overline{a}b$ *then* $\{\![\![P]\!]\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(d(b').\, b \to b' \mid c \to b \mid \{\![\![P']\!]\!\})$, *with* $\{b', c\} \cap \mathrm{fn}(P) = \emptyset$
   (e) *if* $\mu = (\boldsymbol{\nu} d)d(b)\overline{a}b$ *then* $\{\![\![P]\!]\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(\boldsymbol{\nu} d)(d(b').\, b \to b' \mid c \to b \mid \{\![\![P']\!]\!\})$, *with* $\{b', c\} \cap \mathrm{fn}(P) = \emptyset$
   (f) *if* $\mu = \tau$ *then* $\{\![\![P]\!]\!\} \xrightarrow{\tau} \succcurlyeq_{\mathrm{a}} \{\![\![P']\!]\!\}$.
2. *Suppose that* $\{\![\![P]\!]\!\} \xrightarrow{\mu} P_1$. *Then there exists* $P' \in$ DLπ *such that:*
   (a) *if* $\mu = a(b')$ *then* $P \xrightarrow{a(b)} P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} \{\![\![P']\!]\!\}\{b'/b\}$
   (b) *if* $\mu = (\boldsymbol{\nu} c)\overline{a}c$ *then:*
      i. *either* $P \xrightarrow{\overline{a}b} P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} (c \to b \mid \{\![\![P']\!]\!\})$, *with* $c \notin \mathrm{fn}(P)$
      ii. *or* $P \xrightarrow{(\boldsymbol{\nu} b)\overline{a}b} P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(c \to b \mid \{\![\![P']\!]\!\})$, *with* $c \notin \mathrm{fn}(P)$
      iii. *or* $P \xrightarrow{d(b)\overline{a}b} P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(d(b').\, b \to b' \mid c \to b \mid \{\![\![P']\!]\!\})$, *with* $\{b', c\} \cap \mathrm{fn}(P) = \emptyset$
      iv. *or* $P \xrightarrow{(\boldsymbol{\nu} d)d(b)\overline{a}b} P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(\boldsymbol{\nu} d)(d(b').\, b \to b' \mid c \to b \mid \{\![\![P']\!]\!\})$, *with* $\{b', c\} \cap \mathrm{fn}(P) = \emptyset$

(c) *if* $\mu = \tau$ *then* $P \xrightarrow{\tau} P'$ *with* $P_1 \succcurlyeq_{\mathrm{a}} \{\![\![P']\!]\!\}$.

Proof: By transition induction. The proof relies on Proposition 4.3(2), Lemma 5.2, and Lemma 9.5. Details can be found in Appendix A.5. □

From Lemma 9.6 we can derive a weak operational correspondence.

Lemma 9.7

1. *If* $P \Longrightarrow P'$ *then* $\{\![\![P]\!]\!\} \Longrightarrow\succcurlyeq_{\mathrm{a}} \{\![\![P']\!]\!\}$.
2. *If* $\{\![\![P]\!]\!\} \Longrightarrow P_1$ *then there is* P' *such that* $P \Longrightarrow P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} \{\![\![P']\!]\!\}$.
3. $P \Downarrow_a$ *iff* $\{\![\![P]\!]\!\} \Downarrow_a$.

Proof: Parts 1 and 2 are proven by induction on the number of τ-moves by exploiting Lemma 9.6. Part 3 is a consequence of parts 1 and 2, and Lemma 9.6. □

Lemma 9.7 allows us to prove that the encoding $\{\![\![\cdot]\!]\!\}$ is adequate with respect to barbed bisimulation.

Lemma 9.8 *Let* P *and* Q *be two processes in* DLπ. *Then:*

$$P \stackrel{\bullet}{\approx} Q \text{ iff } \{\![\![P]\!]\!\} \stackrel{\bullet}{\approx} \{\![\![Q]\!]\!\}.$$

Proof: In the implication from left to right, we use Lemma 9.7 to prove that the relation $\mathcal{R} = \{(\{\![\![P]\!]\!\}, \{\![\![Q]\!]\!\}) : P \stackrel{\bullet}{\approx} Q\}$ is a barbed bisimulation up-to $\succcurlyeq_{\mathrm{a}}$. In the implication from right to left we use Lemma 9.7 and the fact that $\lesssim_{\mathrm{a}} \stackrel{\bullet}{\approx} \succcurlyeq_{\mathrm{a}} \subset \stackrel{\bullet}{\approx}$ to prove that the relation $\mathcal{R} = \{(P, Q) : \{\![\![P]\!]\!\} \stackrel{\bullet}{\approx} \{\![\![Q]\!]\!\}\}$ is a barbed bisimulation. □

Lemma 9.8 allows us to prove that the encoding $\{\!| \cdot |\!\}$ is adequate with respect to barbed bisimulation and therefore sound with respect to barbed congruence.

Lemma 9.9 *Let* P *and* Q *be two* DLπ*-processes. Then:*

$$P \stackrel{\bullet}{\approx} Q \text{ iff } \{\!| P |\!\} \stackrel{\bullet}{\approx} \{\!| Q |\!\}.$$

Proof: By the definition of $\{\![\![\cdot]\!]\!\}$, Lemma 9.8, and Theorem 5.3. □

Notice that if $P \in$ Lπ then $\{\!| P |\!\} = P$. This will allow us to prove the completeness of $\{\!| \cdot |\!\}$.

Theorem 9.10 (Full abstraction of $\{\!| \cdot |\!\}$) *Let* P *and* Q *be two processes in* DLπ. *Then:*

$$P \cong_{\mathrm{DL}\pi} Q \text{ iff } \{\!| P |\!\} \cong_{\mathrm{L}\pi} \{\!| Q |\!\}.$$

Proof: The soundness follows from Lemma 9.9 and the compositionality of $\{\!| \cdot |\!\}$. As regards the completeness, we want to prove that for every Lπ-context $C[\cdot]$ it holds that $C[\{\!| P |\!\}] \stackrel{\bullet}{\approx} C[\{\!| Q |\!\}]$. Let $C[\cdot]$ be a such a context. By the compositionality of $\{\!| \cdot |\!\}$ and the fact that $\{\!| \cdot |\!\}$ does not affect Lπ-processes, it follows that $\{\!| C[P] |\!\} = C[\{\!| P |\!\}]$ and $\{\!| C[Q] |\!\} = C[\{\!| Q |\!\}]$. Since an L$\pi$-context is also a DL$\pi$-context, by hypothesis we have $C[P] \stackrel{\bullet}{\approx} C[Q]$. By Lemma 9.9 this implies $\{\!| C[P] |\!\} \stackrel{\bullet}{\approx} \{\!| C[Q] |\!\}$. So, it follows that $C[\{\!| P |\!\}] \stackrel{\bullet}{\approx} C[\{\!| Q |\!\}]$. □

Remark 9.11 *In order to clarify the usefulness of the auxiliary encoding* $\{\![\![\cdot]\!]\!\}$, *notice that if one wanted to prove the operational correspondence of* $\{\!| \cdot |\!\}$ *(instead of* $\{\![\![\cdot]\!]\!\}$*), then Proposition 4.3(2) and Lemma 9.5 would no longer be necessary. By contrast, an expansion variant of Proposition 9.1 would be necessary. The proof of a such a result would require an expansion variant of Theorem 5.14 (or Corollary 5.15), which, as already pointed out in Remark 5.16, does not hold.*

Using $\{\!|\cdot|\!\}$ and the theory of Lπ we can prove laws for delayed input like:

$$\begin{array}{rcll} a(b)(P \mid Q) & = & (a(b)P) \mid Q & \text{if } b \notin \mathrm{fn}(Q) \qquad (9)\\ a(b)c(d)P & = & c(d)a(b)P & \text{if } c \neq b \text{ and } d \neq a \qquad (10)\\ (\boldsymbol{\nu}a)(a(x)(\overline{a}x \mid P)) & = & (\boldsymbol{\nu}x)P & \text{if } a \notin \mathrm{fn}(P) \qquad (11) \end{array}$$

Laws 9 and 10 are similar to structural rules for restriction. Similar laws have been proposed in [10]. Law 11 transforms a delayed input binder into a restriction binder (it might be interesting to examine delayed input from within action calculi [37]; for instance, law 11 is reminiscent of the definition of restriction in reflexive action calculi [38]).

9.4 Encodings of the λ-calculus

In this example, we use polyadicity, which is straightforward to accommodate in the theory of Lπ. We write $\overline{a}\langle b_1 \dots b_n\rangle$ for polyadic outputs and $a(x_1, \dots x_n).\, P$ for polyadic inputs. Below, we give Milner's encoding of call-by-name λ-calculus into π-calculus (more precisely, the variant in [42]).

$$\begin{array}{rcl} (\!|\ \lambda x.\, M\ |\!)_p & \stackrel{\mathrm{def}}{=} & (\boldsymbol{\nu}v)(\overline{p}\langle v\rangle \mid v(x,q).\, (\!|\ M\ |\!)_q)\\ (\!|\ x\ |\!)_p & \stackrel{\mathrm{def}}{=} & \overline{x}\langle p\rangle\\ (\!|\ MN\ |\!)_p & \stackrel{\mathrm{def}}{=} & (\boldsymbol{\nu}q)\Big((\!|\ M\ |\!)_q \mid q(v).\, (\boldsymbol{\nu}x)(\overline{v}\langle x,p\rangle \mid !x(r).\, (\!|\ N\ |\!)_r)\Big) \end{array}$$

This is also an encoding into (polyadic) Lπ. By applying Proposition 9.1, we can prove the following optimisation of the definition of application in the case when the argument is a variable (a tail-call-like optimisation):

$$(\!|\ My\ |\!)_p \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu}q)\Big((\!|\ M\ |\!)_q \mid q(v).\, \overline{v}\langle y,p\rangle\Big)$$

We can also exploit the delayed input operator, that is a derived operator in Lπ, to get an encoding of the *strong* call-by-name strategy, where reductions can also occur underneath an abstraction (i.e., the `Xi` rule, saying that if $M \longrightarrow M'$ then $\lambda x.\, M \longrightarrow \lambda x.\, M'$, is allowed). For this, we have to relax, in the translation of $\lambda x.\, M$, the sequentiality imposed by the input prefix $v(x,q)$ that guards the body $(\!|\ M\ |\!)_q$ of the function. Precisely, we have to replace this input with a delayed input:

$$(\!|\ \lambda x.\, M\ |\!)_p \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu}v)(\overline{p}\langle v\rangle \mid v(x,q)(\!|\ M\ |\!)_q) \qquad (12)$$

Using (the polyadic variant of) the encoding of delayed input in Section 9.3, we get:

$$(\!|\ \lambda x.\, M\ |\!)_p \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu}vxq)\Big(\overline{p}\langle v\rangle \mid v(y,r).\, (x \triangleright y \mid q \triangleright r) \mid (\!|\ M\ |\!)_q\Big)$$

Results of operational correspondence and validity of β-reduction, similar to those in [36, 50] for the call-by-name λ-calculus, hold for this encoding. This allows us derive the soundness of the encoding. As customary when translating λ-calculi into the π-calculus, our encodings are not complete. Full abstraction can be recovered by extending the λ-calculus with operators that yield non-confluent reductions [50, 56].

9.5 Some properties of the Join calculus

We apply the theory of Lπ to prove some laws in Fournet and Gonthier's Join calculus [16], a calculus for distributed and concurrent programming.

$$\llparenthesis\, a\langle b\rangle \,\rrparenthesis \stackrel{\text{def}}{=} \overline{a}b \qquad \llparenthesis\, P \mid Q \,\rrparenthesis \stackrel{\text{def}}{=} \llparenthesis\, P \,\rrparenthesis \mid \llparenthesis\, Q \,\rrparenthesis$$

$$\llparenthesis\, \mathtt{def}\ a\langle x\rangle \,|\, b\langle y\rangle = P_1\ \mathtt{in}\ P_2 \,\rrparenthesis \stackrel{\text{def}}{=} (\boldsymbol{\nu} ab)(!a(x).\, b(y).\, \llparenthesis\, P_1 \,\rrparenthesis \mid \llparenthesis\, P_2 \,\rrparenthesis)$$

Table 6: Mapping of the Join calculus into Lπ

The Join calculus is an off-spring of the asynchronous π-calculus specifically designed to facilitate distributed implementations of channels mechanisms. The syntax of the (core) Join calculus is given by the following grammar:

$$P ::= a\langle b\rangle \;\Big|\; P_1 \mid P_2 \;\Big|\; \mathtt{def}\ a\langle x\rangle \,|\, b\langle y\rangle = P_1\ \mathtt{in}\ P_2 .$$

The particle $a\langle b\rangle$ denotes the asynchronous output of name b at channel a. $P_1 \mid P_2$ denotes two processes P_1 and P_2 running in parallel. The construct $\mathtt{def}\ a\langle x\rangle \,|\, b\langle y\rangle = P_1\ \mathtt{in}\ P_2$ is a sort of amalgamation of the operators of replication, parallel composition, and restriction, which allows to model the joint reception of values from different channels.

Free names and *bound names* of a process P, are defined as follows:

- $\mathrm{fn}(a\langle b\rangle) = \{a, b\}$
- $\mathrm{fn}(P_1 \mid P_2) = \mathrm{fn}(P_1) \cup \mathrm{fn}(P_2)$
- $\mathrm{fn}(\mathtt{def}\ a\langle x\rangle \,|\, b\langle y\rangle = P_1\ \mathtt{in}\ P_2) = ((\mathrm{fn}(P_1) \setminus \{x, y\}) \cup \mathrm{fn}(P_2)) \setminus \{a, b\}$.

As for the π-calculus, the operational semantics of the Join calculus can be given in terms of a reduction relation and a relation of structural congruence [31]. With J abbreviating a join pattern $a\langle x\rangle \mid b\langle y\rangle$, the main (simplified) reduction rule is:

$$\mathtt{def}\ J = P\ \mathtt{in}\ J\sigma \mid Q \quad \longrightarrow \quad \mathtt{def}\ J = P\ \mathtt{in}\ P\sigma \mid Q$$

where the substitution σ does not affect channels a and b.

The intuition is that if an instantiation $J\sigma$ of a definition join-patter J can be found at top-level in the scope of the definition, then this instance may be replaced by the corresponding instantiation $P\sigma$ of the defined continuation P.

The construct $\mathtt{def}\ a\langle x\rangle \,|\, b\langle y\rangle = P_1\ \mathtt{in}\ P_2$ has the following properties:

1. channels a and b are *locally defined*, that is, they can be accessed only from within P_1 and P_2;
2. channels a and b are *uniquely defined*, that is, they appear only in one definition (see also [2, 54]).

A derived construct in Join is the *single pattern definition*

$$\mathtt{def}\ a\langle x\rangle = P_1\ \mathtt{in}\ P_2$$

which can be seen as an abbreviation for $\mathtt{def}\ a\langle x\rangle \,|\, b\langle y\rangle = P_1 \,|\, y\langle y\rangle\ \mathtt{in}\ P_2 \,|\, b\langle b\rangle$, where b is not free in P_1 and P_2 and y is not free in P_1. Barbed congruence can be defined in Join in the usual manner.

In order to compare the expressivity of Join and π-calculus, Fournet and Gonthier give the encoding reported in Table 6 of the Join calculus into the π-calculus. This encoding, as an encoding of Join into π_{a} or π-calculus, is not fully-abstract because it is sound but not complete. As a counterexample take the Join processes of Law 14 which are barbed congruent in Join but their translations are not barbed congruent in π_{a}. To recover full abstraction, Fournet and Gonthier have to add a layer of "firewalls" to the encoding. However, we conjecture that the above encoding is fully-abstract with respect to barbed congruence as an encoding of Join into Lπ (a similar conjecture is made by Fournet and Gonthier [16]). It is easy to prove soundness, and this suffices for using the encoding and the theory of Lπ for proving properties of Join processes.

Theorem 9.12 (soundness of $\langle\!| \cdot |\!\rangle$) *Let P and Q be two processes of core Join. Then:*

$$\langle\!| P |\!\rangle \cong_{\mathrm{L}\pi} \langle\!| Q |\!\rangle \text{ implies } P \cong_{\mathrm{J}} Q$$

where $\cong_{\mathrm{J}}$ *is barbed congruence in core Join.*

Proof: The proof follows from the compositionality of the encoding and the operational correspondence between a Join process P and its encoding $\langle\!| P |\!\rangle$. Such an operational correspondence has already been proved in [15]. □

Using this theorem and the theory of Lπ we can prove the following laws for the Join calculus:

$$\begin{aligned}
\texttt{def}\ a\langle x\rangle = R\ \texttt{in}\ P \mid Q &\quad\cong_{\mathrm{J}}\quad (\texttt{def}\ a\langle x\rangle = R\ \texttt{in}\ P) \mid (\texttt{def}\ a\langle x\rangle = R\ \texttt{in}\ Q) && (13)\\
\texttt{def}\ a\langle x\rangle = b\langle x\rangle\ \texttt{in}\ P &\quad\cong_{\mathrm{J}}\quad P\{b/a\} \text{ if } a \neq b && (14)\\
\texttt{def}\ a\langle x\rangle = P\ \texttt{in}\ (Q \mid a\langle b\rangle) &\quad\cong_{\mathrm{J}}\quad \texttt{def}\ a\langle x\rangle = P\ \texttt{in}\ (Q \mid P\{b/x\}) && (15)\\
\texttt{def}\ a\langle x\rangle = P\ \texttt{in}\ C[a\langle b\rangle] &\quad\cong_{\mathrm{J}}\quad \texttt{def}\ a\langle x\rangle = P\ \texttt{in}\ C[P\{b/x\}] && (16)
\end{aligned}$$

where, in (16), context $C[\cdot]$ does not contain binders for a.

Law 13 is the Join calculus version of the replication theorem for parallel composition; it is proved by applying Theorem 9.4(1). Law 14 is the Join calculus version of Proposition 9.1. Law 15 shows a sort of insensitiveness to τ-actions; it is proved in three steps: (i) we apply Theorem 9.4(1), (ii) we use the law $(\boldsymbol{\nu} a)(!a(x).\,P \mid \overline{a}b) \approx (\boldsymbol{\nu} a)(!a(x).\,P \mid P\{b/x\})$, (iii) we apply again Theorem 9.4(1). Law 16 is the Join counterpart of Law 3. The proof of Law 16 requires the sharpened replication theorems of Theorem 9.4, plus the laws for pushing replications underneath input prefixes and restrictions. Note that *none* of these laws can be proved from the encoding $\langle\!| \cdot |\!\rangle$ and the theory of π_{a} or π-calculus: the encodings of the processes in the laws are behaviourally different both in π_{a} and in π-calculus.

9.6 External versus internal mobility

The mobility mechanism of π-calculus can be divided into *internal* mobility and *external* mobility [52]. The former arises when an input meets the output of a private name, the latter when an input meets the output of a free name. The πI-calculus is a subcalculus of π, where only private names may be transmitted. The syntax is given by the following grammar:

$$P ::= \mathbf{0} \;\Big|\; a(x).\,P \;\Big|\; (\boldsymbol{\nu} b)(\overline{a}b.\,P) \;\Big|\; P \mid P \;\Big|\; (\boldsymbol{\nu} a)P \;\Big|\; A\langle a_1, \ldots, a_n\rangle$$

In πI, recursive definitions are more appropriate than replication. This because, when only internal mobility is allowed, recursion is strictly more expressive than replication [52]. Each constant A has a unique defining equation of the form $A \stackrel{\text{def}}{=} (\tilde{x})P$. In constant definition $A \stackrel{\text{def}}{=} (\tilde{x})P$ and constant application $A\langle\tilde{a}\rangle$, the parameters $\tilde{x}$ and $\tilde{a}$ are tuples of all distinct names whose length equals the arity of A. In a constant definition $A \stackrel{\text{def}}{=} (\tilde{x})P$ all free occurrences of names $\tilde{x}$ in P are bound and $\mathrm{fn}(P) \subseteq \tilde{x}$. The transition rule for constants is the standard one [39].

Despite the use of only internal mobility, both λ-calculus and higher-order communications can be faithfully encoded in πI.

An asynchronous variant of πI can be defined by replacing, in the grammar, the blocking output processes $(\boldsymbol{\nu} b)(\overline{a}b.\,P)$ with $(\boldsymbol{\nu} b)(\overline{a}b \mid P)$. We call $\pi\mathrm{I}_{\mathrm{a}}$ this calculus.

In [7], Boreale gives an encoding of asynchronous π-calculus into πI. Boreale's encoding is obtained by applying two different encodings one after the other. The former maps the asynchronous π-calculus into Lπ; the latter (essentially the encoding $[\![\cdot]\!]$ of Section 5) maps Lπ into πI. Boreale shows that the whole encoding is adequate with respect to barbed bisimilarity by proving the adequacy of the two encodings separately. This result is not quite satisfactory because barbed bisimilarity is a very coarse relation, too coarse to be considered, per se, as an important behavioural equivalence (for instance, it is not even preserved by parallel composition). Boreale leaves as an open problem whether the encoding is fully-abstract for some finer behavioural equivalence. Here, we show that $[\![\cdot]\!]$ is not fully-abstract as an encoding of Lπ into πI. As a counterexample, we take

$$P = a(x).\,\overline{a}x \ \text{ and } \ Q = \mathbf{0}$$

By applying Theorem 5.14, we can show that $P \cong_{\mathrm{L}\pi} Q$. However, $[\![P]\!] \not\approx [\![Q]\!]$, and since $\approx$ and $\cong_{\pi\mathrm{I}}$ coincides (on image-finite processes [52]) we conclude that $[\![P]\!] \not\cong_{\pi\mathrm{I}} [\![Q]\!]$. This negative result is not surprising because the source language is asynchronous while the target language is synchronous. However, even if we considered as target language the asynchronous variant of πI, that is $\pi\mathrm{I}_\mathrm{a}$, the encoding would not be fully-abstract. As a counterexample, take the same processes P and Q above. Let

$$R \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu} p)(\overline{a}p \mid a(y).\,(y(z).\,(\boldsymbol{\nu} q)\overline{z}q \mid (\boldsymbol{\nu} r)(\overline{y}r \mid r(x).\,(\boldsymbol{\nu} u)\overline{b}u)))$$

be a $\pi\mathrm{I}_\mathrm{a}$-process. Since $[\![P]\!] = a(x).\,(\boldsymbol{\nu} d)(\overline{a}d \mid d \to x)$, the process $[\![P]\!] \mid R$ would evolve after four τ-steps into the process A where:

$$A \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu} pdr)(d \to p \mid (\boldsymbol{\nu} h)(\overline{p}h \mid h \to r) \mid d(z).\,(\boldsymbol{\nu} q)\overline{z}q \mid r(x).\,(\boldsymbol{\nu} u)\overline{b}u).$$

It holds that $A \not\Downarrow_b$, while, for all processes B such that $[\![Q]\!] \mid R \Longrightarrow B$, it holds that $B \Downarrow_b$. We conclude that $[\![P]\!] \not\cong_{\pi\mathrm{I}_\mathrm{a}} [\![Q]\!]$.

In the remainder of this section we prove that the encoding $[\![\cdot]\!]$ *is* fully-abstract when choosing as target calculus the asynchronous variant of πI where only output capability of names may be transmitted. We call *Localised* πI, in short LπI, this variant of πI.

The lemma below will be crucial for proving the desired full abstraction result. We recall that $\approx_\mathrm{l}$ denotes the link bisimilarity of Definition 6.1.

Lemma 9.13 *Let P be a process in* Lπ. *Then $P \approx_\mathrm{l} [\![P]\!]$.*

Proof: We prove that the relation

$$\mathcal{S} = \{(P, [\![P]\!]) : P \in \mathrm{L}\pi\}$$

is a link bisimulation up to expansion. The proof relies on Lemma 5.1 and Proposition 4.3(3). Details can be found in Appendix A.6. □

Theorem 9.14 (Full abstraction of $[\![\cdot]\!]$) *Let P and Q be two processes in* Lπ. *Then:*

$$P \cong_{\mathrm{L}\pi} Q \ \text{ iff } \ [\![P]\!] \cong_{\mathrm{L}\pi\mathrm{I}} [\![Q]\!].$$

Proof: The implication from right to left follows from Theorem 5.3 and the compositionality of $[\![\cdot]\!]$. As regards the implication from left to right, we want to prove that for all contexts $C[\cdot]$ in LπI it holds that $C[[\![P]\!]] \stackrel{\centerdot}{\approx} C[[\![Q]\!]]$. By Lemma 9.13 and transitivity of $\cong_{\mathrm{L}\pi}$, it follows that $[\![P]\!] \cong_{\mathrm{L}\pi} [\![Q]\!]$, that is, for all contexts $\hat{C}[\cdot]$ in Lπ it holds that $\hat{C}[[\![P]\!]] \stackrel{\centerdot}{\approx} \hat{C}[[\![Q]\!]]$. With a reasoning similar to that in Section 4, it is possible to show that for each process P in LπI there exists a process P' in Lπ such that $P \approx P'$. Similarly, given a context $C[\cdot]$ in LπI there exists a context $\hat{C}[\cdot]$ in Lπ such that $C[[\![P]\!]] \approx \hat{C}[[\![P]\!]] \stackrel{\centerdot}{\approx} \hat{C}[[\![Q]\!]] \approx C[[\![Q]\!]]$. Since $\approx \stackrel{\centerdot}{\approx} \approx \subseteq \stackrel{\centerdot}{\approx}$ we can complete the proof. □

9.7 Operational soundness of CPS axioms

In this section we prove the soundness of Thielecke's axiomatic semantics of the CPS-calculus [61]. The CPS-calculus, similar to the intermediate language of Appel's compiler [4], has been introduced to study the target calculi of Continuation Passing Style transforms. The CPS-calculus is very simple and low-level: only variables may be passed as arguments, moreover application is like a jump, with variables as argument. The terms of the (recursive) CPS-calculus are given by the following grammar:

$$M ::= \ a\langle b\rangle \ \Big| \ M\{a\langle b\rangle \Leftarrow M\}$$

The intended meaning is that $a\langle b\rangle$ is a jump to the continuation a with actual parameter b, while $M\{a\langle b\rangle = N\}$ binds the continuation with body N and formal parameter b to a in M. The calculus is recursive because in $M\{a\langle b\rangle \Leftarrow N\}$ the term N may refer to itself under a. For simplicity we consider a monadic variant of the CPS-calculus. The results of this sections can be straightforwardly extended to the polyadic CPS-calculus.

The set of free variables $\mathrm{fv}(M)$ of a CPS term M is defined as follows.

- $\mathrm{fv}(a\langle b\rangle) \stackrel{\mathrm{def}}{=} \{a, b\}$
- $\mathrm{fv}(M\{a\langle b\rangle \Leftarrow N\}) \stackrel{\mathrm{def}}{=} (\mathrm{fv}(M) \setminus \{a\}) \cup (\mathrm{fv}(N) \setminus \{b\})$

The *axiomatic semantics* for the CPS-calculus is defined as the congruence induced by the following four axioms:

(DISTR) $L\{a\langle b\rangle \Leftarrow M\}\{c\langle d\rangle \Leftarrow N\} = L\{c\langle d\rangle \Leftarrow N\}\{a\langle b\rangle \Leftarrow M\{c\langle d\rangle \Leftarrow N\}\}$
such that $a \neq c$ and $a, b \notin \mathrm{fv}(N)$
(GC) $a\langle b\rangle\{c\langle d\rangle \Leftarrow N\} = a\langle b\rangle,\ c \notin \mathrm{fv}(a\langle b\rangle)$
(JMP) $a\langle b\rangle\{a\langle c\rangle \Leftarrow N\} = N\{b/c\}\{a\langle c\rangle \Leftarrow N\}$
(ETA) $M\{a\langle b\rangle \Leftarrow c\langle b\rangle\} = M\{c/a\}$, $a \neq c$.

The (JMP) law is in some sense what drives the computation. By contrast, (GC) and (DISTR) are "structural" laws similar to those of the π-calculus. Most of the axioms above appear in [4]. We write $\mathrm{CPS} \vdash M{=}N$ to denote that the equality $M = N$ can be derived by the above axiomatic semantics.

Remark 9.15 *The CPS term $M_1\{a\langle b\rangle \Leftarrow M_2\}$ reminds us the construct* `def` $a\langle b\rangle = M_2$ `in` M_1 *of the Join calculus. Actually, the CPS-calculus can be seen as a name confluent subset of the Join calculus.*

In the remainder of this section we prove that the axiomatic semantics defined above is sound with respect to the operational semantics. This is the first proof of the result. Our proof strongly relies on the theory of Lπ.

We give an operational semantics for the CPS-calculus (which is a slight variant of the operational semantics given by Thielecke). It is easy to see that every CPS-term M is in the form $a\langle b\rangle\{a_1\langle b_1\rangle \Leftarrow M_1\} \ldots \{a_n\langle b_n\rangle \Leftarrow M_n\}$ for some $n \geq 0$. This allows us to model the behaviour of CPS-terms by means of just one (global) reduction rule:

$$\begin{gathered} a_i\langle b\rangle\{a_1\langle b_1\rangle \Leftarrow M_1\} \ldots \{a_i\langle b_i\rangle \Leftarrow M_i\} \ldots \{a_n\langle b_n\rangle \Leftarrow M_n\} \\ \rightsquigarrow \\ M_i\{b/b_i\}\{a_1\langle b_1\rangle \Leftarrow M_1\} \ldots \{a_i\langle b_i\rangle \Leftarrow M_i\} \ldots \{a_n\langle b_n\rangle \Leftarrow M_1\} \end{gathered}$$

where $1 \leq i \leq n$ and $a_j \notin \mathrm{fv}(M_i)$ for $1 \leq j < i$. The rule above is a "contextual" variant of the (JMP) axiom.

In the CPS-calculus the notion of observability is represented by the "external" jump that a term may perform after some internal jumps. For instance, in a jump of the form $a\langle b\rangle$, we can observe (the occurrence of a jump to) a. More generally, a free variable in the leftmost position can be observed.

Definition 9.16 (Observability in CPS-calculus) *Let M be a term of the CPS-calculus and a be a name, we write $M \downarrow_a$ if there are names $b, a_1, b_1, \ldots, a_n, b_n$, for some integer $n \geq 0$ with $a \neq a_i$ for every $1 \leq i \leq n$, such that $M = a\langle b\rangle\{a_1\langle b_1\rangle \Leftarrow M_1\} \ldots \{a_n\langle b_n\rangle \Leftarrow M_n\}$. We write $M \Downarrow_a$ if there exists a CPS-term N such that $M \rightsquigarrow^* N \downarrow_a$, where $\rightsquigarrow^*$ denotes the reflexive and transitive closure of $\rightsquigarrow$.*

We denote with $\cong_{\text{CPS}}$ the barbed congruence on CPS terms. We prove the soundness of the axiomatic semantics with respect to $\cong_{\text{CPS}}$ by exploiting a straightforward encoding of the CPS-calculus into π-calculus already appeared in [61].

- $(\!| a\langle b\rangle |\!) \stackrel{\text{def}}{=} \overline{a}b$
- $(\!| M\{a\langle b\rangle \Leftarrow N\} |\!) \stackrel{\text{def}}{=} (\boldsymbol{\nu} a)((\!| M |\!) \mid !a(b).\,(\!| N |\!))$.

There is a straightforward operational correspondence between a CPS-term M and its encoding $(\!| M |\!)$.

Lemma 9.17 (Operational correspondence of $(\!| \cdot |\!)$) *Let M be a CPS-term. Then:*

1. *If $M \rightsquigarrow N$ then $(\!| M |\!) \stackrel{\tau}{\longrightarrow} \equiv (\!| N |\!)$.*
2. *If $(\!| M |\!) \stackrel{\tau}{\longrightarrow} P$ then there is a CPS-term N such that $M \rightsquigarrow N$ and $P \equiv (\!| N |\!)$.*
3. *$M \downarrow_a$ iff $(\!| M |\!) \downarrow_a$.*
4. *If $M \rightsquigarrow^* N$ then $(\!| M |\!) \Longrightarrow \equiv (\!| N |\!)$.*
5. *If $(\!| M |\!) \Longrightarrow P$ then there is a CPS-term N s.t. $M \rightsquigarrow^* N$ and $P \equiv (\!| N |\!)$.*
6. *$M \Downarrow_a$ iff $(\!| M |\!) \Downarrow_a$.*

Proof: The proofs of parts 1, 2, and 3 are easy. Parts 4 and 5 are proved by induction on the number of silent moves. Part 6 is a consequence of previous parts. $\Box$

Lemma 9.18 *Let M and N be two CPS-terms. Then:*

$$(\!| M |\!) \cong_{\text{L}\pi} (\!| N |\!) \text{ implies } M \cong_{\text{CPS}} N.$$

Proof: It follows from the operational correspondence in Lemma 9.17 and the compositionality of $(\!| \cdot |\!)$. $\Box$

Lemma 9.19 *Let M and N be two CPS-terms. Then:*

$$\text{CPS} \vdash M{=}N \text{ implies } (\!| M |\!) \cong_{\text{L}\pi} (\!| N |\!).$$

Proof: Since $\cong_{\text{L}\pi}$ is a congruence, it suffices to prove the soundness of the four axioms (DISTR), (GC), (JMP), and (ETA). More precisely, we prove that if $M = N$ is obtained by applying one of these axioms then $(\!| M |\!) \cong_{\text{L}\pi} (\!| N |\!)$. Let us consider the four possible cases.

1. (DISTR). By applying the encoding $(\!| \cdot |\!)$ to the distributive law we get an instance of the replication theorems seen in Section 9.2. So, we prove this case by simply exploiting Theorem 9.4 and Milner's replication theorem for restriction and input prefix [34].
2. (GC). It suffices to show that

$$(\boldsymbol{\nu} c)(!c(d).\,(\!| N |\!) \mid \overline{a}b) \approx \overline{a}b$$

by exhibiting the appropriate bisimulation. Since $\approx$ implies $\cong_{\text{L}\pi}$ the conclusion follows.

3. (JMP). It suffices to show that

$$(\boldsymbol{\nu} a)(\overline{a}b \mid !a(c).\,P) \approx (\boldsymbol{\nu} a)(P\{b/c\} \mid !a(c).\,P)$$

by proving that the relation

$$\mathcal{S} = \{\big((\boldsymbol{\nu} a)(\overline{a}b \mid !a(c).\,P)\,,\,(\boldsymbol{\nu} a)(P\{b/c\} \mid !a(c).\,P)\big)\} \cup \approx$$

is a synchronous ground bisimulation up to $\equiv$. Since $\approx$ implies $\cong_{\text{L}\pi}$ the conclusion follows.

4. (ETA). This is an application of Proposition 9.1.

□

Note that in the previous proof, laws (DISTR) and (ETA) cannot be proved in π or π_{a}.

Theorem 9.20 *[Soundness of the axiomatic semantics] Let M and N be two CPS-terms. Then:*

$$\mathrm{CPS} \vdash M{=}N \text{ implies } M \cong_{\mathrm{CPS}} N.$$

Proof: By Lemmas 9.18 and 9.19. □

We do not know whether the above axiomatic semantics is complete.

10 Conclusion and related work

We have presented Lπ, an asynchronous variant of the π-calculus without name-matching where the recipients of a channel are local to the process that has created the channel. We have given two coinductive characterisations of barbed congruence and a new up-to proof techniques. We have used our proof techniques to prove a number of algebraic laws and applications. Table 10 summarises the main calculi, with the associated equivalences, studied in the paper.

Calculi similar to Localised π are discussed by Honda and Tokoro [24], Fournet and Gonthier [16], Amadio [2], Boreale [7], and Yoshida [66]. A number of characterisations of barbed congruence on asynchronous mobile processes exist [3, 18, 2]. In [18] the authors prove Law 15 of Section 9.5 using a form of labelled bisimilarity. However, the labelled bisimilarities of [3, 18, 2] can not be used to prove the Laws 1 and 2, stated in the Introduction, (and their Join counterpart Laws 13 and 14) because matching transitions of processes have the same labels. Finally, Law 14 (and its instance, Law 1) has already been studied in [16, 15].

Other studies on barbed congruence, or similar context-based bisimulations, for mobile processes have been conducted, for instance by Honda and Yoshida [26, 65], Kobayashi, Pierce, and Turner [29], Hennessy and Riely [21] and, for a coordination language, by Busi, Gorrieri, and Zavattaro [11]. Fournet and Gonthier [17] showed that the hypothesis of image-finitess can be dropped when characterising barbed congruence in asynchronous π-calculus. Boreale and Sangiorgi [8] have studied barbed congruence in synchronous π-calculus with capability types and no matching, where Lπ can be treated as a special case. Our characterisations are simpler than those in [8], but the latter are more general, in that they can be applied to several π-calculus languages (although the extension to asynchronous languages is not straightforward). The technical approaches are different: in [8] bisimilarities have a type environment (in fact, closures) whereas our bisimilarities are directly defined on processes.

The sharpened replication theorems (cf. Theorem 9.4) have already been proved by Pierce and Sangiorgi [46] using the type system with input/output capabilities. The same result is proved in [54] by previously translating processes (by means of an encoding very close to our $[\![\cdot]\!]$) and then proving that the image are bisimilar. In both cases the replication theorems are shown valid with respect to (typed) barbed congruence by proving a few barbed bisimilarities. In the current paper we propose easier proofs without using typed bisimulations. However, while the results in [46, 54] holds for the (standard) π-calculus, our results only apply to Lπ. Furthermore, we recall that the replication theorems are usually proved with respect to a *strong* form of bisimilarity which is sensitive to τ-actions [34, 46]. The same result holds in Lπ although we cannot prove it using our proof techniques. We can only prove the result for *weak* bisimilarities because the proof of Theorem 9.4 relies on the encoding $[\![\cdot]\!]$ which does not preserve the number of τ-actions performed by a process.

Non-blocking message reception has been considered by Bellin and Scott [6], Boudol [10], Fu [20], Parrow and Victor [44], Yoshida [66] van Breugel [62] and Laneve and Victor [30]. Bellin and Scott give a reduction semantics for a version of π-calculus, proposed by Milner, where both message emission and message reception are non-blocking [35]; van Breugel defines a labelled

Calculi:

Lπ	Definition 2.1
$\mathrm{L}_{[\![\pi]\!]}$	Section 5.2
DLπ	Section 9.3

Relationship:

$$\mathrm{L}_{[\![\pi]\!]} \ \subset^{\dagger} \ \mathrm{L}\pi \ \subset \ \mathrm{DL}\pi$$

Equivalences in* Lπ*:

$\cong_{\mathrm{L}\pi}$	Definition 3.1
$\approx$	Definition 3.4
$\approx_{\mathrm{a}}$	Definition 3.4
$\lesssim$	Definition 3.5
$\overset{\mapsto}{\approx}_{\mathrm{a}}$	Section 6
$\approx_{\mathrm{l}}$	Definition 6.1
$\approx_{\mathrm{lut}}$	Definition 8.3

Relationship:

$$\lesssim \ \subset \ \approx \ \subset \ \approx_{\mathrm{a}} \ \subset \ \approx_{\mathrm{lut}} \ = \ \approx_{\mathrm{l}} \ = \ \overset{\mapsto}{\approx}_{\mathrm{a}} \ =^{\ddagger} \ \cong_{\mathrm{L}\pi}$$

Equivalences in* $\mathrm{L}_{[\![\pi]\!]}$*:

$\cong_{\mathrm{L}_{[\![\pi]\!]}}$	Definition 3.1
$\asymp_{\mathrm{a}}$	Definition 5.6

Relationship:

$$\asymp_{\mathrm{a}} \ =^{\ddagger} \ \cong_{\mathrm{L}_{[\![\pi]\!]}}$$

(†) Up to $\approx$.
(‡) On image-finite processes.

Table 7: Calculi and Equivalences

transition system for such calculus and proves a correspondence with Bellin and Scott's reduction semantics. Finally, Laneve and Victor give an encoding in Fusion Calculus of blocking input in terms of delayed input and name matching [30]. We recall that both the Fusion and the Chi calculus come equipped with a form of delayed input. As a consequence, both calculi are able to encode strong call-by-name λ-calculus [44, 20].

Concerning the CPS-calculus studied in Section 9.7, we are not aware of other proofs of Theorem 9.20.

In Join and Blue calculus, polymorphic type systems à la ML have been introduced [19, 14]. In both cases, the constraint on the output capability of names is crucial. We believe that similar polymorphic type systems can be defined in Lπ.

By the time the writing of this paper has been completed, the theory of Lπ has already been used in some works. In [32], we give an encoding of polyadic Lπ into monadic Lπ. Unlike Milner's encoding of polyadic π into monadic π [34], the encoding in [32] is fully-abstract with respect to barbed congruence. In [57], we give a fully-abstract encoding of *higher-order* Lπ (where processes can be transmitted) into Lπ. The theory of Lπ allows proofs simpler than those of analogous results for other π-calculi [49, 51]. In [33] Lπ is used to give a translational semantics of (an appropriate abstraction of) Cardelli's distributed object-based programming language Obliq [12]. The theory of Lπ (precisely a typed variant of Lemma 5.17) is used to prove the correctness of *object migration*. Finally, we are currently using Lπ to study termination for mobile processes [58].

Acknowledgements

We thank the anonymous referees for their constructive remarks.

References

[1] S. Abramsky. Proofs as Processes. *Theoretical Computer Science*, 135(1):5–9, December 1994.

[2] R. Amadio. An asynchronous model of locality, failure, and process mobility. In *Proc. Coordination'97*, volume 1282 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[3] R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195:291–324, 1998. Extended abstract in *Proc. CONCUR '96*, LNCS 1119, Springer Verlag.

[4] A. Appel. *Compiling with Continuations*. Cambridge University Press, 1992.

[5] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29:737–760, 1992.

[6] G. Bellin and P. Scott. On the π-calculus and Linear Logic. *Theoretical Computer Science*, 135(1):11–65, December 1994.

[7] M. Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theoretical Computer Science*, 195:205–226, 1998.

[8] M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. In *13th LICS Conf.* IEEE Computer Society Press, 1998.

[9] G. Boudol. Asynchrony and the π-calculus. Technical Report RR-1702, INRIA-Sophia Antipolis, 1992.

[10] G. Boudol. Some Chemical Abstract Machines. In *Proc. Rex School/Symposium 1993 "A Decade of Concurrency — Reflexions and Perspectives"*, volume 803 of *Lecture Notes in Computer Science*, pages 92–123. Springer Verlag, 1994.

[11] N. Busi, R. Gorrieri, and G. Zavattaro. A process algebraic view of linda coordination primitives. *Theoretical Computer Science*, 192(2):167–199, 1988.

[12] L. Cardelli. A language with distributed scope. *Computing Systems*, 8(1):27–59, 1995. Short version in *Proc. of POPL '95*.

[13] I. Castellani and M. Hennessy. Testing theories for asynchronous languages. In *Proc. FST-TCS'98*, volume 1530 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.

[14] S. Dal-Zilio. Implicit polymorphic type system for the blue calculus. Technical Report RR-3244, Inria, Institut National de Recherche en Informatique et en Automatique, 1997.

[15] C. Fournet. *The Join calculus: a Calculus for Distributed Mobile Programming*. PhD thesis, École Polytechnique, 1999.

[16] C. Fournet and G. Gonthier. The Reflexive Chemical Abstract Machine and the Join calculus. In *Proc. 23th POPL*. ACM Press, 1996.

[17] C. Fournet and G. Gonthier. A Hierarchy of Equivalences for Asynchronous Calculi. In *Proc. 25th ICALP*, volume 1443 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.

[18] C. Fournet and C. Laneve. Bisimulations for the Join Calculus. *Theoretical Computer Science*, 266:569–603, 2001.

[19] C. Fournet, C. Laneve, L. Maranget, and D. Rémy. Implict typing à la ML for the join-calculus. In *Proc. CONCUR 97*, volume 1243 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[20] Y. Fu. A proof theoretical approach to communication. In *24th ICALP*, volume 1256 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[21] M. Hennessy and J. Riely. A typed language for distributed mobile processes. In *Proc. 25th POPL*. ACM Press, 1998.

[22] K. Honda. Two bisimilarities for the ν-calculus. Technical Report 92-002, Keio University, 1992.

[23] K. Honda and M. Tokoro. An Object Calculus for Asynchronous Communications. In *Proc. ECOOP '91*, volume 512 of *Lecture Notes in Computer Science*. Springer Verlag, 1991.

[24] K. Honda and M. Tokoro. A Small Calculus for Concurrent Objects. In *OOPS Messanger*, Association for Computing Machinery. 2(2):50-54, 1991.

[25] K. Honda and N. Yoshida. Replication in Concurrent Combinators. In *Proc. TACS'94*, volume 789 of *Lecture Notes in Computer Science*. Springer Verlag, 1994.

[26] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[27] H. Hüttel and J. Kleist. Objects as mobile processes. Technical report RR-96-38, BRICS - Basic Research in Computer Science, 1996.

[28] J. Kleist and D. Sangiorgi. Imperative objects and mobile processes. In *Proc. PROCOMET'98*. North-Holland, 1998.

[29] N. Kobayashi, B.C. Pierce, and D.N. Turner. Linearity and the pi-calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, 1993. Short version in Proc. POPL'96.

[30] C. Laneve and B. Victor. Solos in Concert. In *Proc. 26th ICALP*, volume 1644 of *Lecture Notes in Computer Science*. Springer Verlag, 1999.

[31] Jean-Jacques Lévy. Some results on the join-calculus. In *Proc. TACS '97*, volume 1281 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[32] M Merro. Locality and polyadicity in asynchronous name-passing calculi. In *Proc. FOSSACS2000*, volume 1784 of *Lecture Notes in Computer Science*. Springer Verlag, 2000.

[33] M. Merro, J. Kleist, and U. Nestmann. Mobile Objects as Mobile Processes. *Accepted for publication in Journal of Information and Computation*, 2001. An extended abstract entitled *Local π-calculus at Work: Mobile Objects as Mobile Processes*, appeared in the *Proceedings of TCS'00*, LNCS 1872, August 2000.

[34] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, LFCS, Dept. of Comp. Sci., Edinburgh Univ., October 1991. Also in *Logic and Algebra of Specification*, ed. F.L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag, 1993.

[35] R. Milner. Action structure for the π-calculus. Technical Report ECS–LFCS–93–264, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1992.

[36] R. Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[37] R.. Milner. Action calculi, or syntactic action structures. In *Proc MFCS'93*, volume 711 of *Lecture Notes in Computer Science*. Springer Verlag, 1993.

[38] R. Milner. Action calculi V: reflexive molecular forms (with appendix by Ole Jensen). Draft, 1994.

[39] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.

[40] R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. ICALP'92*, volume 623 of *Lecture Notes in Computer Science*. Springer Verlag, 1992.

[41] U. Nestmann and B. Pierce. Decoding choice encodings. *Journal of Information & Computation*, 163:1–59, November 2000.

[42] G.K. Ostheimer and A.J.T. Davie. π-calculus characterisations of some practical λ-calculus reductions strategies. Technical Report CS/93/14, St. Andrews, 1993.

[43] J. Parrow and B. Victor. The update calculus. In *Proc. AMAST '97*, volume 1349 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[44] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proc. LICS'98*, IEEE Computer Society Press, 1998.

[45] A. Philippou and D. Walker. On transformations of concurrent object programs. In *Proc. CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.

[46] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Journal of Mathematical Structures in Computer Science*, 6(5):409–454, 1996. An extended abstract in *Proc. LICS '93*, IEEE Computer Society Press.

[47] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. Technical Report CSCI 476, Indiana University, 1997. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, MIT Press, 2000".

[48] G.D. Plotkin. A structural approach to operational semantics. DAIMI-FN-19, Computer Science Department, Aarhus University, 1981.

[49] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh, 1992.

[50] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Information and Computation*, 111(1):120–153, 1994.

[51] D. Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Information and Computation*, 131(2):141–178, 1996.

[52] D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(2):235–274, 1996.

[53] D. Sangiorgi. An interpretation of typed objects into typed π-calculus. *Information and Computation*, 143(1), 1998.

[54] D. Sangiorgi. The name discipline of uniform receptiveness. *Theoretical Computer Science*, 221:457–493, 1999.

[55] D. Sangiorgi. Typed π-calculus at work: a correctness proof of Jones's parallelisation transformation on concurrent objects. *Theory and Practice of Object systems*, 5(1):25–34, 1999.

[56] D. Sangiorgi. Lazy functions and mobile processes. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.

[57] D. Sangiorgi. Asynchronous process calculi: the first-order and higher-order paradigms (tutorial). *Theoretical Computer Science*, 253(1), 2001.

[58] D. Sangiorgi. Types, or: Where's the difference between ccs and π? In *Proc. CONCUR '02*, volume 2421 of *Lecture Notes in Computer Science*. Springer Verlag, 2002.

[59] D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". In *Proc. CONCUR '92*, volume 630 of *Lecture Notes in Computer Science*. Springer Verlag, 1992.

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

[61] H. Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, 1997. Also available as technical report ECS-LFCS-97-376.

[62] F. van Breugel. A Labelled Transition System for the $\pi\epsilon$-calculus. In *Proc. TAPSOFT'97*, volume 1214 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[63] V. T. Vasconcelos. *A process-calculus approach to typed concurrent objects*. PhD thesis, Keio University, 1994.

[64] D. Walker. π-calculus semantics of object-oriented programming languages. In *Proc. TACAS'91*, volume 526 of *Lecture Notes in Computer Science*. Springer Verlag, 1991.

[65] N. Yoshida. Graph types for monadic mobile processes. In *Proc. FST & TCS*, volume 1180 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.

[66] N. Yoshida. Minimality and Separation Results on Asynchronous Mobile Processes: representability theorem by concurrent combinators. In *Proc. CONCUR'98*, volume 1466 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.

A Proofs

A.1 Proofs of Lemmas 5.10 and 5.12

For the sake of clarity we restate the result as follows.

Lemma A.1 *Let P and Q be two $\mathrm{L}_{[\![\pi]\!]}$-processes such that $P \asymp_{\mathrm{a}} Q$. Then:*

1. $(\boldsymbol{\nu} a)P \asymp_{\mathrm{a}} (\boldsymbol{\nu} a)Q$
2. $P \mid R \asymp_{\mathrm{a}} Q \mid R$, *for all* $\mathrm{L}_{[\![\pi]\!]}$*-processes* R
3. $a(x).\,P \asymp_{\mathrm{a}} a(x).\,Q$
4. $!a(x).\,P \asymp_{\mathrm{a}} !a(x).\,Q$.

Proof: 1. It suffices to show that the relation

$$\mathcal{S}=\{((\boldsymbol{\nu} a)P\,,\,(\boldsymbol{\nu} a)Q):\ P,Q\in \mathrm{L}_{[\![\pi]\!]} \text{ and } P\asymp_{\mathrm{a}} Q\}$$

is a $\asymp_{\mathrm{a}}$-bisimulation up-to structural congruence. The proof is easy because the output actions performed by an $\mathrm{L}_{[\![\pi]\!]}$-process are always bound. We work up to structural congruence when dealing with the asynchronous clause for input.

2. We prove that the relation

$$\mathcal{S}=\{((\boldsymbol{\nu}\tilde a)(P\mid R),(\boldsymbol{\nu}\tilde a)(Q\mid R)):\ P,Q,R\in \mathrm{L}_{[\![\pi]\!]} \text{ and } P\asymp_{\mathrm{a}} Q\}$$

is a $\asymp_{\mathrm{a}}$-bisimulation up to $\gtrsim$ and $\approx$. Let us consider the possible actions of $(\boldsymbol{\nu}\tilde a)(P \mid R)$:

(a) If $(\boldsymbol{\nu}\tilde a)(P\mid R)\xrightarrow{\overline{b}(c)}A$, with c fresh and $b\notin\tilde a$, then there are two cases:

i. either $R\xrightarrow{\overline{b}(c)}R'$, for some R', and $A=(\boldsymbol{\nu}\tilde a)(P\mid R')$. Process $(\boldsymbol{\nu}\tilde a)(Q\mid R)$ can match this action in the same way;

ii. or $P\xrightarrow{\overline{b}(c)}P'$ for some P'. This means $A=(\boldsymbol{\nu}\tilde a)(P'\mid R)$. Since $P\asymp_{\mathrm{a}} Q$ there exist Q' such that $Q\overset{\overline{b}(c)}{\Longrightarrow}Q'$ and $P'\asymp_{\mathrm{a}} Q'$. So, $(\boldsymbol{\nu}\tilde a)(Q\mid R)\overset{\overline{b}(c)}{\Longrightarrow}(\boldsymbol{\nu}\tilde a)(Q'\mid R)$ and $((\boldsymbol{\nu}\tilde a)(P'\mid R),(\boldsymbol{\nu}\tilde a)(Q'\mid R))\in\mathcal{S}$.

(b) If $(\boldsymbol{\nu}\tilde a)(P\mid R)\xrightarrow{b(c)}A$, with c fresh and $b\notin\tilde a$, then there are two cases:

i. either $R\xrightarrow{b(c)}R'$, for some R', and then we can proceed as in case 2(a)i;

ii. or $P\xrightarrow{b(c)}P'$ and $A=(\boldsymbol{\nu}\tilde a)(P'\mid R)$. Since $P\asymp_{\mathrm{a}} Q$, there are two possible cases:

A. either $Q\overset{b(c)}{\Longrightarrow}Q'$, with $P'\asymp_{\mathrm{a}} Q'$, and the conclusion follows easily;

B. or $Q\Longrightarrow Q'$, with $P'\asymp_{\mathrm{a}} Q'\mid[\![\overline{b}c]\!]$. This means that $(\boldsymbol{\nu}\tilde a)(Q\mid R)\Longrightarrow(\boldsymbol{\nu}\tilde a)(Q'\mid R)$ with $(\boldsymbol{\nu}\tilde a)(P'\mid R)\,\mathcal{S}\,(\boldsymbol{\nu}\tilde a)(Q'\mid[\![\overline{b}c]\!]\mid R)\equiv(\boldsymbol{\nu}\tilde a)(Q'\mid R)\mid[\![\overline{b}c]\!]$, which is enough because $\equiv$ is contained in $\approx$ and $\mathcal{S}$ is a $\asymp_{\mathrm{a}}$-bisimulation up to $\gtrsim$ and $\approx$.

(c) If $(\boldsymbol{\nu}\tilde a)(P\mid R)\xrightarrow{\tau}A$, there are four cases:

i. $P\xrightarrow{\tau}P'$. This case is easy.

ii. $R\xrightarrow{\tau}R'$. This case is easy.

iii. $P\xrightarrow{\overline{b}(c)}P'$, $R\xrightarrow{b(c)}R'$ and $(\boldsymbol{\nu}\tilde a)(P\mid R)\xrightarrow{\tau}(\boldsymbol{\nu}\tilde a c)(P'\mid R')$. Since $P\asymp_{\mathrm{a}} Q$ there exists Q' such that $Q\overset{\overline{b}(c)}{\Longrightarrow}Q'$ and $P'\asymp_{\mathrm{a}} Q'$. This means that $(\boldsymbol{\nu}\tilde a)(Q\mid R)\Longrightarrow(\boldsymbol{\nu}\tilde a c)(Q'\mid R')$ and $((\boldsymbol{\nu}\tilde a c)(P'\mid R'),(\boldsymbol{\nu}\tilde a c)(Q'\mid R'))\in\mathcal{S}$.

iv. $P \xrightarrow{b(c)} P'$, $R \xrightarrow{\overline{b}(c)} R'$ and $(\boldsymbol{\nu}\tilde{a})(P \mid R) \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{a}c)(P' \mid R')$. Since $P \precsim_{\mathrm{a}} Q$ there are two cases:

A. either there exists Q' such that $Q \xRightarrow{b(c)} Q'$ with $P' \precsim_{\mathrm{a}} Q'$, and then the conclusion follows;

B. or there exists Q' such that $Q \Longrightarrow Q'$ and $P' \precsim_{\mathrm{a}} Q' \mid [\![\overline{b}c]\!]$. In this case, by Lemma 5.9(2), $R \approx (\boldsymbol{\nu}c)([\![\overline{b}c]\!] \mid R')$ and therefore

$$(\boldsymbol{\nu}\tilde{a})(Q \mid R) \Longrightarrow (\boldsymbol{\nu}\tilde{a})(Q' \mid R)$$

with

$$(\boldsymbol{\nu}\tilde{a}c)(P' \mid R') \, \mathcal{S} \, (\boldsymbol{\nu}\tilde{a}c)(Q' \mid [\![\overline{b}c]\!] \mid R') \approx (\boldsymbol{\nu}\tilde{a})(Q' \mid R).$$

3. It suffices to show that the relation

$$\mathcal{S} = \{(a(x).\, P \,, a(x).\, Q) : \; P, Q \in \mathrm{L}_{[\![\pi]\!]} \text{ and } P \precsim_{\mathrm{a}} Q\}$$

is a $\precsim_{\mathrm{a}}$-bisimulation. The proof relies on the fact that $\precsim_{\mathrm{a}}$ is preserved by injective substitutions: since $\precsim_{\mathrm{a}}$ is ground, in the input clause only fresh names are received.

4. We prove that the relation $\mathcal{S}$ defined below

$$\{(P_1 \mid !a(x).\, Q_1, P_2 \mid !a(x).\, Q_2) : P_1, P_2, Q_1, Q_2 \in \mathrm{L}_{[\![\pi]\!]}\,,\; P_1 \precsim_{\mathrm{a}} P_2\,,\; Q_1 \precsim_{\mathrm{a}} Q_2\}$$

is a $\precsim_{\mathrm{a}}$-bisimulation up to structural congruence (it is a special case of the proof technique of Definition 5.7). If $P_1 \mid !a(x).\, Q_1 \xrightarrow{\mu} R$ for some action μ and some process R then there are three possible cases:

(a) $P_1 \xrightarrow{\mu} P_1'$ and $R = P_1' \mid !a(x).\, Q_1$. Then, since $P_1 \precsim_{\mathrm{a}} P_2$, the conclusion follows.

(b) $\mu = a(w)$ and $P_1 \mid !a(x).\, Q_1 \xrightarrow{a(w)} P_1 \mid Q_1\{{}^{w}\!/\!{}_{x}\} \mid !a(x).\, Q_1$. On the other side we have $P_2 \mid !a(x).\, Q_2 \xrightarrow{a(w)} P_2 \mid Q_2\{{}^{w}\!/\!{}_{x}\} \mid !a(x).\, Q_2$. Since $Q_1 \precsim_{\mathrm{a}} Q_2$, and $\precsim_{\mathrm{a}}$ is preserved by injective substitutions, we have $Q_1\{{}^{w}\!/\!{}_{x}\} \precsim_{\mathrm{a}} Q_2\{{}^{w}\!/\!{}_{x}\}$. By Lemma 5.10(2) we have $P_1 \mid Q_1\{{}^{w}\!/\!{}_{x}\} \precsim_{\mathrm{a}} P_2 \mid Q_1\{{}^{w}\!/\!{}_{x}\} \precsim_{\mathrm{a}} P_2 \mid Q_2\{{}^{w}\!/\!{}_{x}\}$. By transitivity (Lemma 5.11(3)) we conclude that $P_1 \mid Q_1\{{}^{w}\!/\!{}_{x}\} \mid !a(x).\, Q_1 \; \mathcal{S} \; P_2 \mid Q_2\{{}^{w}\!/\!{}_{x}\} \mid !a(x).\, Q_2$.

(c) $P_1 \xrightarrow{\overline{a}(c)} P_1'$ and $P_1 \mid !a(x).\, Q_1 \xrightarrow{\tau} (\boldsymbol{\nu}c)(P_1' \mid Q_1\{{}^{c}\!/\!{}_{x}\}) \mid !a(x).\, Q_1$. Since $P_1 \precsim_{\mathrm{a}} P_2$, there exists P_2' such that $P_2 \xRightarrow{\overline{a}(c)} P_2'$ and $P_1' \precsim_{\mathrm{a}} P_2'$. This means that $P_2 \mid !a(x).\, Q_2 \Longrightarrow (\boldsymbol{\nu}c)(P_2' \mid Q_2\{{}^{c}\!/\!{}_{x}\}) \mid !a(x).\, Q_2$. We reason as in the previous case, and by applying Lemmas 5.10(2), 5.11(3), and 5.10(1) the proof is complete.

□

A.2 Proof of Lemma 6.3

We already pointed out in Lemma 5.1 that there exists an operational correspondence on strong transitions between processes P and $[\![P]\!]$. Boreale also proved a weak operational correspondence between processes P and $[\![P]\!]$.

Lemma A.2 (Boreale [7]) *Let P be an $\mathrm{L}\pi$-process.*

1. *Suppose that $P \xRightarrow{\alpha} P'$. Then we have:*

   (a) *if $\alpha = a(b)$ then $[\![P]\!] \xRightarrow{a(b)} \gtrsim [\![P']\!]$;*

   (b) *if $\alpha = \overline{a}b$ then $[\![P]\!] \xRightarrow{\overline{a}(p)} \gtrsim (p \to b \mid [\![P']\!])$, with $p \notin \mathrm{fn}(P')$;*

   (c) *if $\alpha = \overline{a}(b)$ then $[\![P]\!] \xRightarrow{\overline{a}(p)} \gtrsim (\boldsymbol{\nu}b)(p \to b \mid [\![P']\!])$, with $p \notin \mathrm{fn}(P')$;*

(d) *if* $\alpha = \tau$ *then* $[\![P]\!] \stackrel{\tau}{\Longrightarrow} \gtrsim [\![P']\!]$.

2. *Suppose that* $[\![P]\!] \stackrel{\alpha}{\Longrightarrow} P_1$. *Then there exists* $P' \in \mathrm{L}\pi$ *such that:*

   (a) *if* $\alpha = a(b)$ *then* $P \stackrel{a(b)}{\Longrightarrow} P'$, *with* $P_1 \gtrsim [\![P']\!]$;

   (b) *if* $\alpha = \overline{a}(p)$ *then:*

      i. *either* $P \stackrel{\overline{a}b}{\Longrightarrow} P'$, *with* $p \notin \mathrm{fn}(P')$ *and* $P_1 \gtrsim (p \to b \mid [\![P']\!])$,

      ii. *or* $P \stackrel{\overline{a}(b)}{\Longrightarrow} P'$, *with* $p \notin \mathrm{fn}(P')$ *and* $P_1 \gtrsim (\boldsymbol{\nu} b)(p \to b \mid [\![P']\!])$;

   (c) *if* $\alpha = \tau$ *then* $P \stackrel{\tau}{\Longrightarrow} P'$ *with* $P_1 \gtrsim [\![P']\!]$.

This result will be used to prove Lemma 6.3 which we restate here.

Lemma A.3 *Let* P *and* Q *be two processes in* $\mathrm{L}\pi$. *Then:*

$$P \approx_{\mathrm{l}} Q \ \text{ iff } \ [\![P]\!] \asymp_{\mathrm{a}} [\![Q]\!].$$

Proof: We prove the implication *from left to right.*
We show that the relation

$$\mathcal{S} = \{([\![P]\!], [\![Q]\!]) \mid \ P, Q \in \mathrm{L}\pi \ \text{ and } \ P \approx_{\mathrm{l}} Q\}$$

is a $\asymp_{\mathrm{a}}$-bisimulation up-to $\gtrsim$. Let us consider the three possible actions of $[\![P]\!]$:

1. If $[\![P]\!] \stackrel{\tau}{\longrightarrow} P_1$, by lemma 5.1, there exists P' such that $P \stackrel{\tau}{\longrightarrow} P'$ and $P_1 \gtrsim [\![P']\!]$. Since $P \approx_{\mathrm{l}} Q$, there exists Q' such that $Q \Longrightarrow Q'$ and also $P' \approx_{\mathrm{l}} Q'$. By Lemma A.2, there exists Q_1 such that $[\![Q]\!] \stackrel{\tau}{\Longrightarrow} Q_1 \gtrsim [\![Q']\!]$.

2. If $[\![P]\!] \stackrel{a(c)}{\longrightarrow} P_1$, by Lemma 5.1, there exists P' such that $P \stackrel{a(c)}{\longrightarrow} P'$ and $P_1 \gtrsim [\![P']\!]$. Since $P \approx_{\mathrm{l}} Q$ there are two possibilities:

   (a) There exists Q' such that $Q \stackrel{a(c)}{\Longrightarrow} Q'$ and $P' \approx_{\mathrm{l}} Q'$. Then, by Lemma A.2, there exists Q_1 such that $[\![Q]\!] \stackrel{a(c)}{\Longrightarrow} Q_1 \gtrsim [\![Q']\!]$.

   (b) There exists Q' such that $Q \Longrightarrow Q'$ and $P' \approx_{\mathrm{l}} Q' \mid \overline{a}c$. Then, by Lemma A.2, there exists Q_1 such that $[\![Q]\!] \Longrightarrow Q_1 \gtrsim [\![Q']\!]$. Notice that $Q_1 \gtrsim [\![Q']\!]$ implies $Q_1 \mid [\![\overline{a}c]\!] \gtrsim [\![Q' \mid \overline{a}c]\!]$.

3. If $[\![P]\!] \stackrel{\overline{a}(p)}{\longrightarrow} P_1$, assuming p fresh, by Lemma 5.1, there are two possible cases:

   (a) $P \stackrel{\overline{a}b}{\longrightarrow} P'$ and $P_1 \gtrsim (p \to b \mid [\![P']\!]) = [\![p \triangleright b \mid P']\!]$. Since $P \approx_{\mathrm{l}} Q$, there are still two cases:

      i. There exists a process Q' such that $Q \stackrel{\overline{a}d}{\Longrightarrow} Q'$ and $(p \triangleright b \mid P') \approx_{\mathrm{l}} (p \triangleright d \mid Q')$. By Lemma A.2 there exists Q_1 such that $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim (p \to d \mid [\![Q']\!]) = [\![p \triangleright d \mid Q']\!]$. So, $[\![P]\!] \stackrel{\overline{a}(p)}{\longrightarrow} P_1 \gtrsim [\![p \triangleright b \mid P']\!]$, and $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim [\![p \triangleright d \mid Q']\!]$, and the conclusion follows since $([\![p \triangleright b \mid P']\!], [\![p \triangleright d \mid Q']\!]) \in \mathcal{S}$.

      ii. There exists a process Q' such that $Q \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$ and $(p \triangleright b \mid P') \approx_{\mathrm{l}} (\boldsymbol{\nu} c)(p \triangleright c \mid Q')$. By Lemma A.2 there exists Q_1 such that $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim (\boldsymbol{\nu} c)(p \to c \mid [\![Q']\!])$. So, $[\![P]\!] \stackrel{\overline{a}(p)}{\longrightarrow} P_1 \gtrsim [\![p \triangleright b \mid P']\!]$ and $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim [\![(\boldsymbol{\nu} c)(p \triangleright c \mid Q')]\!]$, and the conclusion follows since $([\![p \triangleright b \mid P']\!], [\![(\boldsymbol{\nu} c)(p \triangleright c \mid Q')]\!]) \in \mathcal{S}$.

   (b) $P \stackrel{\overline{a}(c)}{\longrightarrow} P'$ and $P_1 \gtrsim (\boldsymbol{\nu} c)(p \to c \mid [\![P']\!]) = [\![(\boldsymbol{\nu} c)(p \triangleright c \mid P')]\!]$. Since $P \approx_{\mathrm{l}} Q$, there are two possibilities:

i. There exists Q' such that $Q \overset{\overline{a}b}{\Longrightarrow} Q'$ and $(\boldsymbol{\nu}c)(p \triangleright c \mid P') \approx_1 (p \triangleright b \mid Q')$. By Lemma A.2, there exists a process Q_1 such that $[\![Q]\!] \overset{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim (p \rightarrow b \mid Q')$. So, $[\![P]\!] \xrightarrow{\overline{a}(p)} P_1 \gtrsim [\![(\boldsymbol{\nu}c)(p \triangleright c \mid P')]\!]$ and $[\![Q]\!] \overset{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim [\![p \triangleright b \mid Q']\!]$, and the conclusion follows since $([\![(\boldsymbol{\nu}c)(p \triangleright c \mid P')]\!], [\![p \triangleright b \mid Q']\!]) \in \mathcal{S}$.

ii. There exists Q' such that $Q \overset{\overline{a}(c)}{\Longrightarrow} Q'$ and $(\boldsymbol{\nu}c)(p \triangleright c \mid P') \approx_1 (\boldsymbol{\nu}c)(p \triangleright c \mid Q')$. By Lemma A.2, there exists a process Q_1 such that $[\![Q]\!] \overset{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim (\boldsymbol{\nu}c)(p \rightarrow c \mid [\![Q']\!])$. So, $[\![P]\!] \xrightarrow{\overline{a}(p)} P_1 \gtrsim [\![(\boldsymbol{\nu}c)(p \triangleright c \mid P')]\!]$ and $[\![Q]\!] \overset{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim [\![(\boldsymbol{\nu}c)(p \triangleright c \mid Q')]\!]$, and the conclusion follows since $([\![(\boldsymbol{\nu}c)(p \triangleright c \mid P')]\!], [\![(\boldsymbol{\nu}c)(p \triangleright c \mid Q')]\!]) \in \mathcal{S}$.

We prove the implication *from right to left.* We show that the relation

$$\mathcal{S} = \{(P, Q) \mid P, Q \in \mathrm{L}\pi \text{ and } [\![P]\!] \asymp_{\mathrm{a}} [\![Q]\!]\}$$

is a link bisimulation. Let us consider the four possible actions of P.

1. If $P \xrightarrow{\tau} P'$, by Lemma 5.1, there exists P_1 such that $[\![P]\!] \xrightarrow{\tau} P_1 \gtrsim [\![P']\!]$. Since $[\![P]\!] \asymp_{\mathrm{a}} [\![Q]\!]$, there exists Q_1 such that $[\![Q]\!] \overset{\tau}{\Longrightarrow} Q_1$ and $P_1 \asymp_{\mathrm{a}} Q_1$. By Lemma A.2, there exists a process Q' such that $Q \overset{\tau}{\Longrightarrow} Q'$ and $Q_1 \gtrsim [\![Q']\!]$. So, we have $[\![P']\!] \lesssim P_1 \asymp_{\mathrm{a}} Q_1 \gtrsim [\![Q']\!]$. Since $\lesssim$ and $\gtrsim$ are included in $\asymp_{\mathrm{a}}$, and $\asymp_{\mathrm{a}}$ is transitive, we conclude that $(P', Q') \in \mathcal{S}$.

2. If $P \xrightarrow{a(c)} P'$, by Lemma 5.1, there exists P_1 such that $[\![P]\!] \xrightarrow{a(c)} P_1 \gtrsim [\![P']\!]$. Since $[\![P]\!] \asymp_{\mathrm{a}} [\![Q]\!]$ we have two cases:

   (a) There exists a processes Q_1 such that $[\![Q]\!] \overset{a(c)}{\Longrightarrow} Q_1$ and $P_1 \asymp_{\mathrm{a}} Q_1$. By Lemma A.2 there exists a process Q' such that $Q \overset{a(c)}{\Longrightarrow} Q'$ and $Q_1 \gtrsim [\![Q']\!]$. So, we have $[\![P']\!] \lesssim P_1 \asymp_{\mathrm{a}} Q_1 \gtrsim [\![Q']\!]$. It therefore follows that $(P', Q') \in \mathcal{S}$.

   (b) There exists Q_1 such that $[\![Q]\!] \Longrightarrow Q_1$ and $P_1 \asymp_{\mathrm{a}} Q_1 \mid [\![\overline{a}c]\!]$. By Lemma A.2 there exists a process Q' such that $Q \Longrightarrow Q'$ and $Q_1 \gtrsim [\![Q']\!]$. Notice that $Q_1 \mid [\![\overline{a}c]\!] \gtrsim [\![Q' \mid \overline{a}c]\!]$. So, we have

   $$[\![P']\!] \lesssim P_1 \asymp_{\mathrm{a}} Q_1 \mid [\![\overline{a}c]\!] \gtrsim [\![Q' \mid \overline{a}c]\!].$$

   It therefore follows that $(P' , Q' \mid \overline{a}c) \in \mathcal{S}$.

3. If $P \xrightarrow{\overline{a}b} P'$, by Lemma 5.1, there exists P_1 such that $[\![P]\!] \xrightarrow{\overline{a}(p)} P_1 \gtrsim p \rightarrow b \mid [\![P']\!] = [\![p \triangleright b \mid P']\!]$. Since $[\![P]\!] \asymp_{\mathrm{a}} [\![Q]\!]$ there exists Q_1 such that $[\![Q]\!] \overset{\overline{a}(p)}{\Longrightarrow} Q_1$ and $P_1 \asymp_{\mathrm{a}} Q_1$. By Lemma A.2, we have two possibilities:

   (a) There exists Q' such that $Q \overset{\overline{a}d}{\Longrightarrow} Q'$ and $Q_1 \gtrsim p \rightarrow d \mid [\![Q']\!] = [\![p \triangleright d \mid Q']\!]$. So, we have $[\![p \triangleright b \mid P']\!] \lesssim P_1 \asymp_{\mathrm{a}} Q_1 \gtrsim [\![p \triangleright d \mid Q']\!]$. It therefore follows that $((p \triangleright b \mid P') , (p \triangleright d \mid Q')) \in \mathcal{S}$.

   (b) There exists Q' such that $Q \overset{\overline{a}(c)}{\Longrightarrow} Q'$ and $Q_1 \gtrsim (\boldsymbol{\nu}c)(p \rightarrow c \mid [\![Q']\!]) = [\![(\boldsymbol{\nu}c)(p \rightarrow c \mid Q')]\!]$. So, we have $[\![p \triangleright b \mid P']\!] \lesssim P_1 \asymp_{\mathrm{a}} Q_1 \gtrsim [\![(\boldsymbol{\nu}c)(p \triangleright c \mid Q')]\!]$. It therefore follows that $((p \triangleright b \mid P') , (\boldsymbol{\nu}c)(p \triangleright c \mid Q')) \in \mathcal{S}$.

4. $P \xrightarrow{\overline{a}(c)} P'$, by Lemma 5.1 there exists P_1 such that $[\![P]\!] \xrightarrow{\overline{a}(p)} P_1 \gtrsim (\boldsymbol{\nu}c)(p \rightarrow c \mid [\![P']\!]) = [\![(\boldsymbol{\nu}c)(p \triangleright c \mid P')]\!]$. Since $[\![P]\!] \asymp_{\mathrm{a}} [\![Q]\!]$ there exists Q_1 such that $[\![Q]\!] \overset{\overline{a}(p)}{\Longrightarrow} Q_1$ and $P_1 \asymp_{\mathrm{a}} Q_1$. By Lemma A.2 we have two possibilities:

   (a) There exists Q' such that $Q \overset{\overline{a}b}{\Longrightarrow} Q'$ and $Q_1 \gtrsim p \rightarrow b \mid [\![Q']\!] = [\![p \triangleright b \mid Q']\!]$. So, we have $[\![(\boldsymbol{\nu}c)(p \triangleright c \mid P')]\!] \lesssim P_1 \asymp_{\mathrm{a}} Q_1 \gtrsim [\![p \triangleright b \mid Q']\!]$. It therefore follows that $((\boldsymbol{\nu}c)(p \triangleright c \mid P') , (p \triangleright b \mid Q')) \in \mathcal{S}$.

(b) There exists Q' such that $Q \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$ and $Q_1 \gtrsim (\boldsymbol{\nu}c)(p \rightarrow c \mid [\![Q']\!])$. So, we have $[\![(\boldsymbol{\nu}c)(p \triangleright c \mid P')]\!] \lesssim P_1 \succsim_{\mathrm{a}} Q_1 \gtrsim [\![(\boldsymbol{\nu}c)(p \triangleright c \mid Q')]\!]$. It therefore follows that $((\boldsymbol{\nu}c)(p \triangleright c \mid P')\,,\,(\boldsymbol{\nu}c)(p \triangleright c \mid Q')) \in \mathcal{S}$.

$\square$

A.3 Proof of Lemma 7.1

Proof: Let $\simeq_{\mathrm{l}}$ be the variant of $\approx_{\mathrm{l}}$ obtained by replacing $\xrightarrow{\mu}$ with $\stackrel{\mu}{\Longrightarrow}$ in the hypothesis of each clause in Definition 5.6. Let $\dot{\simeq}$ be the variant of $\dot{\approx}$ obtained by replacing $\xrightarrow{\tau}$ with $\stackrel{\tau}{\Longrightarrow}$ and $\downarrow_a$ with $\Downarrow_a$ in the hypothesis of each clause in Definition 3.1. It holds that $\simeq_{\mathrm{l}} = \approx_{\mathrm{l}}$ and $\dot{\simeq} = \dot{\approx}$. So, we prove that

$$(\forall\, R \in \mathrm{L}\pi \;\; P \mid R \;\dot{\simeq}\; Q \mid R) \implies P \simeq_{\mathrm{l}} Q.$$

Let $\mathcal{F}$ be the monotone operator over $\mathcal{P}(\mathrm{L}\pi \times \mathrm{L}\pi)$ associated with the definition of $\simeq_{\mathrm{l}}$. Suppose $\simeq_{\mathrm{l}}^{0} = \mathrm{L}\pi \times \mathrm{L}\pi$, $\simeq_{\mathrm{l}}^{k+1} = \mathcal{F}(\simeq_{\mathrm{l}}^{k})$, and $\simeq_{\mathrm{l}}^{\omega} = \bigcap_{k<\omega} \simeq_{\mathrm{l}}^{k}$. On image-finite LTS the operator $\mathcal{F}$ preserves co-directed sets (the dual of directed sets). In particular, $\mathcal{F}(\simeq_{\mathrm{l}}^{\omega}) = \simeq_{\mathrm{l}}^{\omega}$. This means that on image-finite processes $\simeq_{\mathrm{l}} = \simeq_{\mathrm{l}}^{\omega}$. Therefore, we are left with proving:

$$(\forall\, R \in \mathrm{L}\pi \;\; P \mid R \;\dot{\simeq}\; Q \mid R) \text{ imply that } P \simeq_{\mathrm{l}}^{\omega} Q.$$

We define a collection of tests $R(n, \mathcal{L}, \mathcal{M})$ depending on the integer n and the finite sets of channel names $\mathcal{L}$, $\mathcal{M}$. Intuitively, $\mathcal{L}$ contains the free names along which the processes P and Q may perform some observable actions which have to be tested by $R(n, \mathcal{L}, \mathcal{M})$; $\mathcal{M}$ contains the names in $\mathcal{L}$ which cannot be used in input subject position by the test process. We show by induction on n that there exist $\mathcal{L}, \mathcal{M}, \mathcal{L}'$ such that $\mathcal{L} \supseteq \mathrm{fn}(P, Q)$, $\mathcal{L}' \subseteq \mathcal{L}$, $\mathcal{M} \subseteq \mathcal{L}$ and $(\boldsymbol{\nu}\mathcal{L}')(P \mid R(n, \mathcal{L}, \mathcal{M})) \;\dot{\simeq}\; (\boldsymbol{\nu}\mathcal{L}')(P \mid R(n, \mathcal{L}, \mathcal{M}))$ implies $P \simeq_{\mathrm{l}}^{n} Q$.

If the property above holds then we can complete the proof by observing that:

$$\begin{array}{ll}
\forall\, R \in \mathrm{L}\pi \; (P \mid R \;\dot{\simeq}\; Q \mid R) & \text{implies } (\boldsymbol{\nu}\mathcal{L}')(P \mid R(n, \mathcal{L}, \mathcal{M})) \;\dot{\simeq}\; (\boldsymbol{\nu}\mathcal{L}')(Q \mid R(n, \mathcal{L}, \mathcal{M})) \\
& \qquad \text{for each } n \in \omega \text{ with } \mathcal{L} = \mathrm{fn}(P, Q), \\
& \qquad \mathcal{M} = \emptyset, \text{ and } \mathcal{L}' = \emptyset. \\
& \text{implies } \forall\, n \in \omega \;\; P \simeq_{\mathrm{l}}^{n} Q \\
& \text{implies } P \simeq_{\mathrm{l}}^{\omega} Q \\
& \text{implies } P \simeq_{\mathrm{l}} Q.
\end{array}$$

For defining the tests $R(n, \mathcal{L}, \mathcal{M})$ we introduce an internal choice operator $\oplus$. This is a derived operator defined as follows:

$$P_1 \oplus \ldots \oplus P_n \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu}a)(a.\, P_1 \mid \ldots \mid a.\, P_n \mid \overline{a}) \text{ with } a \notin \mathrm{fn}(P_1, \ldots, P_n)$$

We suppose that the collection of channel names Ch has been partitioned in two infinite ordered sets Ch' and Ch''. In the following we have $\mathcal{L}' \subseteq \mathcal{L}$, $\mathcal{M} \subseteq \mathcal{L}$, $\mathcal{L} \subseteq Ch''$. We also use the sequences

$$\{b_n, b'_n : n \in \omega\} \cup \{c_n^{\beta} : n \in \omega,\; \beta \in \{\tau, a, \overline{a} : a \in Ch''\}\}$$

of distinct names in Ch'. The test $R(n, \mathcal{L}, \mathcal{M})$ is defined by induction on n as follows, where we pick a' to be the first name in the ordered set $Ch'' \setminus \mathcal{L}$. When emitting or receiving a name which is not in $\mathcal{L}$ we work up to injective substitution to show that $P \simeq_{\mathrm{l}}^{n} Q$. The relation $R(n, \mathcal{L}, \mathcal{M})$ is defined as follows:

$$R(0, \mathcal{L}, \mathcal{M}) = \overline{b_0} \oplus \overline{b'_0}$$

$$\begin{array}{rl}
R(n, \mathcal{L}) = & \overline{b_n} \oplus \overline{b'_n} \text{ (for } n > 0) \\
& \oplus(\overline{c_n^{\tau}} \oplus R(n-1, \mathcal{L}, \mathcal{M})) \\
& \oplus\{\overline{c_n^{\overline{a}}} \oplus ((\boldsymbol{\nu}a')(\overline{a}a' \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M}))) \mid a \in \mathcal{L}\} \\
& \oplus\{\overline{c_n^{a}} \oplus a(x).\, (a' \triangleright x \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\})) \mid a \in \mathcal{L}, a \notin \mathcal{M}\}.
\end{array}$$

Notice that, unlike the characterisation proof of [3] for π_{a}, the above relation does not contain name matching in the input branch. More precisely, in our test relation name matching is replaced by a static link to model the inability to distinguish names in output object position.
The proof is by induction on n. The case $n = 0$ is trivial because $\simeq_1^0 = \mathrm{L}\pi \times \mathrm{L}\pi$. If $n > 0$, by induction, we suppose that

$$(\boldsymbol{\nu}\mathcal{L}')(P \mid R(n, L)) \stackrel{\bullet}{\simeq} (\boldsymbol{\nu}\mathcal{L}')(Q \mid R(n, L)) \text{ and } P \stackrel{\mu}{\Longrightarrow} P'.$$

We proceed by case analysis on the action μ to show that Q can match the action μ.

1. $\mu = \tau$. Then:

$$(\boldsymbol{\nu}\mathcal{L}')(P \mid R(n, \mathcal{L}, \mathcal{M})) \stackrel{\tau}{\Longrightarrow} (\boldsymbol{\nu}\mathcal{L}')(P \mid (\overline{c_n^\tau} \oplus R(n-1, \mathcal{L}, \mathcal{M})))$$

   To match this reduction up to barbed bisimulation we must have:

$$(\boldsymbol{\nu}\mathcal{L}')(Q \mid R(n, \mathcal{L}, \mathcal{M})) \stackrel{\tau}{\Longrightarrow} (\boldsymbol{\nu}\mathcal{L}')(Q_1 \mid (\overline{c_n^\tau} \oplus R(n-1, \mathcal{L}, \mathcal{M})))$$

   We make a further reduction on the left handside

$$(\boldsymbol{\nu}\mathcal{L}')(P \mid (\overline{c_n^\tau} \oplus R(n-1, \mathcal{L}, \mathcal{M}))) \stackrel{\tau}{\Longrightarrow} (\boldsymbol{\nu}\mathcal{L}')(P' \mid R(n-1, \mathcal{L}, \mathcal{M}))$$

   Again this has to be matched by (note that we cannot run the process $R(n-1, \mathcal{L}, \mathcal{M})$ without losing a commitment $\overline{b_n}$ or $\overline{b'_n}$):

$$(\boldsymbol{\nu}\mathcal{L}')(Q_1 \mid (\overline{c_n^\tau} \oplus R(n-1, \mathcal{L}, \mathcal{M}))) \stackrel{\tau}{\Longrightarrow} (\boldsymbol{\nu}\mathcal{L}')(Q' \mid R(n-1, \mathcal{L}, \mathcal{M}))$$

   We observe that $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\tau}{\Longrightarrow} Q'$ and the conclusion follows by applying the inductive hypothesis.

2. $\mu = \overline{a}b$. We may suppose $b \in \mathcal{L}$. Then

$$\begin{array}{rl} (\boldsymbol{\nu}\mathcal{L}')(Q \mid R(n, \mathcal{L}, \mathcal{M})) & \stackrel{\tau}{\Longrightarrow} \\ (\boldsymbol{\nu}\mathcal{L}')(Q_1 \mid \overline{c_n^a} \oplus a(x).\,(a' \triangleright x \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\}))) & \end{array}$$

   We take a further step on the lhs:

$$\begin{array}{rl} (\boldsymbol{\nu}\mathcal{L}')(P \mid \overline{c_n^a} \oplus a(x).\,(a' \triangleright x \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\},))) & \stackrel{\tau}{\Longrightarrow} \\ (\boldsymbol{\nu}\mathcal{L}')(P' \mid a' \triangleright b \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\})) & \end{array}$$

   We take a further step on the lhs:

$$\begin{array}{rl} (\boldsymbol{\nu}\mathcal{L}')(P \mid \overline{c_n^a} \oplus a(x).\,(a' \triangleright x \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\},))) & \stackrel{\tau}{\Longrightarrow} \\ (\boldsymbol{\nu}\mathcal{L}')(P' \mid a' \triangleright b \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\})) & \end{array}$$

   This can be matched:

   (a) either by

$$\begin{array}{rl} (\boldsymbol{\nu}\mathcal{L}')(Q_1 \mid \overline{c_n^a} \oplus a(x).\,(a' \triangleright x \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\},))) & \stackrel{\tau}{\Longrightarrow} \\ (\boldsymbol{\nu}\mathcal{L}')(Q' \mid a' \triangleright d \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\})) & \end{array}$$

   This means that $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\overline{a}d}{\Longrightarrow} Q'$. By inductive hypothesis, and reasoning up to structural congruence, it follows that $a' \triangleright b \mid P' \simeq_1^{n-1} a' \triangleright d \mid Q'$.

(b) or by

$$(\boldsymbol{\nu}\mathcal{L}')(Q_1 \mid \overline{c^a_n} \oplus a(x).\,(a' \triangleright x \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\}))) \quad \stackrel{\tau}{\Longrightarrow}$$
$$(\boldsymbol{\nu}\mathcal{L}')((\boldsymbol{\nu}c)(Q' \mid a' \triangleright c) \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\}))$$

This means that that $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$. By inductive hypothesis, and reasoning up to structural congruence, it follows that that $a' \triangleright b \mid P' \simeq^{n-1}_{\mathrm{l}} (\boldsymbol{\nu}c)(a' \triangleright c \mid Q')$.

3. $\mu = \overline{a}(c)$. We may suppose $c \notin \mathrm{fn}(Q)$. Then

$$(\boldsymbol{\nu}\mathcal{L}')(P \mid R(n, \mathcal{L}, \mathcal{M})) \quad \stackrel{\tau}{\Longrightarrow}$$
$$(\boldsymbol{\nu}\mathcal{L}')(P \mid \overline{c^a_n} \oplus a(x).\,(a' \triangleright x \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\})))$$

This has to be matched by:

$$(\boldsymbol{\nu}\mathcal{L}')(Q \mid R(n, \mathcal{L}, \mathcal{M})) \quad \stackrel{\tau}{\Longrightarrow}$$
$$(\boldsymbol{\nu}\mathcal{L}')(Q_1 \mid \overline{c^a_n} \oplus a(x).\,(a' \triangleright x \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\})))$$

We take a further step on the lhs:

$$(\boldsymbol{\nu}\mathcal{L}')(P \mid \overline{c^a_n} \oplus a(x).\,(a' \triangleright x \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\}))) \quad \stackrel{\tau}{\Longrightarrow}$$
$$(\boldsymbol{\nu}\mathcal{L}')((\boldsymbol{\nu}c)(P' \mid a' \triangleright c) \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\}))$$

This can be matched:

(a) either by

$$(\boldsymbol{\nu}\mathcal{L}')(Q_1 \mid \overline{c^a_n} \oplus a(x).\,(a' \triangleright x \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\}))) \quad \stackrel{\tau}{\Longrightarrow}$$
$$(\boldsymbol{\nu}\mathcal{L}')(Q' \mid a' \triangleright b \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\}))$$

This means that $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\overline{a}b}{\Longrightarrow} Q'$. By inductive hypothesis, and reasoning up to structural congruence, it follows that $(\boldsymbol{\nu}c)(a' \triangleright c \mid P') \simeq^{n-1}_{\mathrm{l}} a' \triangleright b \mid Q'$.

(b) or by

$$(\boldsymbol{\nu}\mathcal{L}')(Q_1 \mid \overline{c^a_n} \oplus a(x).\,(a' \triangleright x \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\}))) \quad \stackrel{\tau}{\Longrightarrow}$$
$$(\boldsymbol{\nu}\mathcal{L}')((\boldsymbol{\nu}d)(Q' \mid a' \triangleright d) \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M} \cup \{a'\}))$$

This means that $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\overline{a}(d)}{\Longrightarrow} Q'$. Since $c \notin \mathrm{fn}(Q)$ we also have that $Q \stackrel{\tau}{\Longrightarrow} Q_1$ and $Q_1 \stackrel{\overline{a}(c)}{\Longrightarrow} Q'\{c/d\}$. By inductive hypothesis, and reasoning up to structural congruence, it follows that that $(\boldsymbol{\nu}c)(a' \triangleright c \mid P' \simeq^{n-1}_{\mathrm{l}} (\boldsymbol{\nu}c)(a' \triangleright c \mid Q')$.

4. $\mu = aa'$. We may suppose a' is the first element in $Ch'' \setminus \mathcal{L}$ (otherwise we rename and use injective substitution). Then

$$(\boldsymbol{\nu}\mathcal{L}')(P \mid R(n, \mathcal{L}, \mathcal{M})) \stackrel{\tau}{\Longrightarrow} (\boldsymbol{\nu}\mathcal{L}')(P \mid (\overline{c^{\overline{a}}_n} \oplus ((\boldsymbol{\nu}a')(\overline{a}a' \mid R(n-1, \mathcal{L} \cup \{a', \mathcal{M}\})))))$$

This has to be matched by

$$(\boldsymbol{\nu}\mathcal{L}')(Q \mid R(n, \mathcal{L}, \mathcal{M})) \stackrel{\tau}{\Longrightarrow} (\boldsymbol{\nu}\mathcal{L}')(Q_1 \mid (\overline{c^{\overline{a}}_n} \oplus ((\boldsymbol{\nu}a')(\overline{a}a' \mid R(n-1, \mathcal{L} \cup \{a', \mathcal{M}\})))))$$

We make a further reduction on the lhs:

$$(\boldsymbol{\nu}\mathcal{L}')(P \mid (\overline{c_n^{\overline{a}}} \oplus ((\boldsymbol{\nu}a')(\overline{a}a' \mid R(n-1, \mathcal{L} \cup \{a', \mathcal{M}\}))))) \quad \stackrel{\tau}{\Longrightarrow} \\ (\boldsymbol{\nu}\mathcal{L}'a')(P' \mid R(n-1, \mathcal{L} \cup \{a', \mathcal{M}\}))$$

This is matched by:

$$(\boldsymbol{\nu}\mathcal{L}')(Q_1 \mid (\overline{c_n^{\overline{a}}} \oplus ((\boldsymbol{\nu}a')(\overline{a}a' \mid R(n-1, \mathcal{L} \cup \{a', \mathcal{M}\}))))) \stackrel{\tau}{\Longrightarrow} Q''$$

We have two possibilities:

(a) $Q_1 \stackrel{\tau}{\Longrightarrow} Q'$ and $Q'' \equiv (\boldsymbol{\nu}\mathcal{L}'a')(Q' \mid \overline{a}a' \mid R(n-1, \mathcal{L} \cup \{a'\}, \mathcal{M}))$.

Then $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\tau}{\Longrightarrow} Q'$ and $P' \simeq_{\mathrm{l}}^{n-1} Q' \mid \overline{a}a'$ by inductive hypothesis and up to structural congruence.

(b) $Q_1 \stackrel{aa'}{\Longrightarrow} Q'$ and $Q'' \equiv (\boldsymbol{\nu}\mathcal{L}'a')(Q' \mid R(n-1, \mathcal{L} \cup \{a', \mathcal{M}\}))$. Then $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{aa'}{\Longrightarrow} Q'$ and $P' \simeq_{\mathrm{l}}^{n-1} Q'$ by inductive hypothesis and up to structural congruence.

□

A.4 Complement to the proof of Theorem 8.4

Lemma A.4 *Let P and Q be two $\mathrm{L}\pi$-processes and p a name such that $p \notin \mathrm{fn}(P, Q)$. Then $(p \triangleright b \mid P) \approx_{\mathrm{l}} (p \triangleright b \mid Q)$ implies $P \approx_{\mathrm{l}} Q$.*

Proof: We show that the relation

$$\mathcal{S} = \{(P, Q) : P, Q \in \mathrm{L}\pi \ \text{ and } \ p \notin \mathrm{fn}(P, Q) \ \text{ and } \ (p \triangleright b \mid P) \approx_{\mathrm{l}} (p \triangleright b \mid Q)\}$$

is a link bisimulation up to structural congruence. The proof is straightforward and it is based on the fact that p is a fresh name and therefore there cannot be any interaction between the link $p \triangleright b$ and the processes P and Q. □

Lemma A.5 *Let P and Q be two $\mathrm{L}\pi$-processes and b a name such that $b \notin \mathrm{fn}(P, Q)$. Then $P \approx_{\mathrm{lut}} Q$ implies $P \mid \overline{a}b \approx_{\mathrm{lut}} Q \mid \overline{a}b$.*

Proof: We show that the relation

$$\mathcal{S} = \{(P \mid \overline{a}b, Q \mid \overline{a}b) : \ P, Q \in \mathrm{L}\pi, \ b \notin \mathrm{fn}(P, Q), \ P \approx_{\mathrm{lut}} Q\} \cup \approx_{\mathrm{lut}}$$

is a link bisimulation up to link up to $\equiv$. It is enough to consider two cases:

1. $P \mid \overline{a}b \stackrel{\alpha}{\longrightarrow} P' \mid \overline{a}b$, that is, $P \stackrel{\alpha}{\longrightarrow} P'$. Then the process $Q \mid \overline{a}b$ can match this actions in the same way.
2. $P \mid \overline{a}b \stackrel{\alpha}{\longrightarrow} P''$, that is, the output particle $\overline{a}b$ is consumed. Then there are two possibilities:
   (a) $P \mid \overline{a}b \stackrel{\overline{a}b}{\longrightarrow} P$. Then the process $Q \mid \overline{a}b$ can match this actions in the same way.
   (b) $P \mid \overline{a}b \stackrel{\tau}{\longrightarrow} P'$. This means that $P \stackrel{a(b)}{\longrightarrow} P'$. Since $P \approx_{\mathrm{lut}} Q$ we have two cases.
      i. $Q \stackrel{a(b)}{\Longrightarrow} Q'$ and $P' \approx_{\mathrm{lut}} Q'$ and the process $Q \mid \overline{a}b$ can match this actions in the same way, or
      ii. $Q \Longrightarrow Q'$ and $P' \approx_{\mathrm{lut}} Q' \mid \overline{a}b$. Then $Q \mid \overline{a}b \Longrightarrow Q' \mid \overline{a}b$ and since $P' \approx_{\mathrm{lut}} Q' \mid \overline{a}b$ it follows that $(P', Q' \mid \overline{a}b) \in \mathcal{S}$.

□

The following lemma shows that $\approx_{\rm lut}$ is preserved by the introduction of links.

Lemma A.6 *Let P and Q be two* Lπ*-processes and p a name such that $p \notin \mathrm{fn}(P, Q)$. Then*

1. $P \approx_{\rm lut} Q$ *implies* $(p \rhd b \mid P) \approx_{\rm lut} (p \rhd b \mid Q)$.
2. $P \approx_{\rm lut} Q$ *implies* $(\boldsymbol{\nu} c)(p \rhd c \mid P) \approx_{\rm lut} (\boldsymbol{\nu} c)(p \rhd c \mid Q)$.
3. $P\{p/c\} \approx_{\rm lut} (\boldsymbol{\nu} c)(p \rhd c \mid Q)$ *implies* $(\boldsymbol{\nu} c)(p \rhd c \mid P) \approx_{\rm lut} (\boldsymbol{\nu} c)(p \rhd c \mid Q)$.
4. $P\{p/c\} \approx_{\rm lut} (p \rhd b \mid Q)$, $c \notin \mathrm{fn}(Q)$, *implies* $(\boldsymbol{\nu} c)(p \rhd c \mid P) \approx_{\rm lut} (p \rhd b \mid Q)$.

Proof: 1. It suffices to show that the relation

$$\mathcal{S} = \{(p \rhd b \mid P \,,\, p \rhd b \mid Q) : P, Q \in \mathrm{L}\pi,\ P \approx_{\rm lut} Q,\ p \notin \mathrm{fn}(P, Q)\}$$

is a link bisimulation up to link up to $\equiv$. Since $p \notin \mathrm{fn}(P, Q)$ there cannot be an interaction between the link and the processes. The most interesting case is when $p \rhd b \mid P \xrightarrow{p(w)} p \rhd b \mid \overline{p}w \mid P$, with w fresh, and $p \rhd b \mid Q \xrightarrow{p(w)} p \rhd b \mid \overline{p}w \mid Q$. By Lemma A.5 it follows that $((p \rhd b \mid \overline{p}w \mid P), (p \rhd b \mid \overline{p}w \mid Q)) \in \mathcal{S}$.

2. We prove that the relation

$$\mathcal{S} = \{((\boldsymbol{\nu} c)(p \rhd c \mid P) \,,\, (\boldsymbol{\nu} c)(p \rhd c \mid Q)) :\ P \approx_{\rm lut} Q,\ p \notin \mathrm{fn}(P \mid Q)\}$$

is a link bisimulation up to link up to $\equiv$. We use the following abbreviation: $A = (\boldsymbol{\nu} c)(p \rhd c \mid P)$ and $B = (\boldsymbol{\nu} c)(p \rhd c \mid Q)$. We consider the possible actions of A.

(a) If $A \xrightarrow{\tau} A_1$, since $p \notin \mathrm{fn}(P, Q)$, there exists P' such that $P \xrightarrow{\tau} P'$ and $A_1 = (\boldsymbol{\nu} c)(p \rhd c \mid P')$. Since $P \approx_{\rm lut} Q$ there exists Q' such that $Q \Longrightarrow Q'$ and $P' \approx_{\rm l} Q'$. This means that there exists B_1 such that $B \Longrightarrow B_1$, $B_1 = (\boldsymbol{\nu} c)(p \rhd c \mid Q')$ and $(A_1, B_1) \in \mathcal{S}$.

(b) If $A \xrightarrow{a(b)} A_1$, with $a \neq c$ and $b \notin \mathrm{fn}(A, B)$. There are two cases:

  i. $a \neq p$. This means that $P \xrightarrow{a(b)} P'$ and $A_1 = (\boldsymbol{\nu} c)(p \rhd c \mid P')$. Since $P \approx_{\rm lut} Q$ there are two possibilities:

    A. $Q \stackrel{a(b)}{\Longrightarrow} Q'$ and $P' \approx_{\rm lut} Q'$. In this case $B \stackrel{a(b)}{\Longrightarrow} B_1$ where $B_1 = (\boldsymbol{\nu} c)(p \rhd c \mid Q')$ and $(A_1, B_1) \in \mathcal{S}$.

    B. $Q \Longrightarrow Q'$ and $P' \approx_{\rm lut} Q' \mid \overline{a}b$. In this case $B \Longrightarrow B_1 = (\boldsymbol{\nu} c)(p \rhd c \mid Q')$ with $A_1\ \mathcal{S}\ (\boldsymbol{\nu} c)(p \rhd c \mid (Q' \mid \overline{a}b)) \equiv B_1 \mid \overline{a}b$.

  ii. $a = p$. Since $p \notin \mathrm{fn}(P \mid Q)$, it holds that $A_1 = (\boldsymbol{\nu} c)(p \rhd c \mid \overline{c}b \mid P)$. So, $B \stackrel{a(b)}{\Longrightarrow} B_1 = (\boldsymbol{\nu} c)(p \rhd c \mid \overline{c}b \mid Q)$. Since $P \approx_{\rm lut} Q$ and $b \notin \mathrm{fn}(P, Q)$, by Lemma A.5, we have $(P \mid \overline{c}b) \approx_{\rm lut} (Q \mid \overline{c}b)$. It follows that $A_1 \equiv (\boldsymbol{\nu} c)(p \rhd c \mid P \mid \overline{c}b)\ \mathcal{S}\ (\boldsymbol{\nu} c)(p \rhd c \mid Q \mid \overline{c}b) \equiv B_1$

(c) If $A \xrightarrow{\overline{a}b} A_1$. This means that $P \xrightarrow{\overline{a}b} P'$, $\{a, b\} \cap \{c, p\} = \emptyset$ and $A_1 = (\boldsymbol{\nu} c)(p \rhd c \mid P')$. Since $P \approx_{\rm lut} Q$, there are three possibilities:

  i. $Q \stackrel{\overline{a}b}{\Longrightarrow} Q'$ and $P' \approx_{\rm lut} Q'$. This implies $B \stackrel{\overline{a}b}{\Longrightarrow} B_1 = (\boldsymbol{\nu} c)(p \rhd c \mid Q')$ and $(A_1, B_1) \in \mathcal{S}$.

  ii. $Q \stackrel{\overline{a}(d)}{\Longrightarrow} Q'$ with $d \notin \mathrm{fn}(P \mid Q)$. We have two possible requirements for the derivatives. We consider the most difficult, that is, $(w \rhd b \mid P') \approx_{\rm lut} (\boldsymbol{\nu} d)(w \rhd d \mid Q')$ for some fresh name w. We have,

$$B \stackrel{\overline{a}(d)}{\Longrightarrow} B_1 = (\boldsymbol{\nu} c)(p \rhd c \mid Q')$$

and

$$\begin{array}{rl} (w \triangleright b \mid A_1) & \equiv \\ (\boldsymbol{\nu} c)(p \triangleright c \mid (w \triangleright b \mid P')) & \mathcal{S} \\ (\boldsymbol{\nu} c)(p \triangleright c \mid ((\boldsymbol{\nu} d)(w \triangleright d \mid Q'))) & \equiv \\ (\boldsymbol{\nu} d)(w \triangleright d \mid B_1). & \end{array}$$

iii. $Q \stackrel{\overline{a}h}{\Longrightarrow} Q'$, with $h \neq b$ and $(w \triangleright b \mid P') \;\mathcal{S}\; (w \triangleright h \mid Q')$ for some fresh name w. We reason as in the previous case.

(d) If $A \xrightarrow{\overline{a}(b)} A_1$ the reasoning is similar to that for case 3.

3. The proof follows from Lemma A.6(2) and Proposition 4.3. Let r be a fresh name, by hypothesis we know that $P\{^r\!/\!_c\} \approx_{\mathrm{lut}} (\boldsymbol{\nu} c)(r \triangleright c \mid Q)$. By Lemma A.6(2), we have

$$(\boldsymbol{\nu} r)(p \triangleright r \mid P\{^r\!/\!_c\}) \approx_{\mathrm{lut}} (\boldsymbol{\nu} r)(p \triangleright r \mid (\boldsymbol{\nu} c)(r \triangleright c \mid Q))$$

for p fresh. By Proposition 4.3 we have

$$(\boldsymbol{\nu} r)(p \triangleright r \mid (\boldsymbol{\nu} c)(r \triangleright c \mid Q)) \equiv (\boldsymbol{\nu} c)((\boldsymbol{\nu} r)(p \triangleright r \mid r \triangleright c) \mid Q) \gtrsim (\boldsymbol{\nu} c)(p \triangleright c \mid Q).$$

Since $(\boldsymbol{\nu} c)(p \triangleright c \mid P) \equiv (\boldsymbol{\nu} r)(p \triangleright r \mid P\{^r\!/\!_c\})$ and $\equiv \approx_{\mathrm{lut}} \equiv \gtrsim \subset \approx_{\mathrm{lut}}$ it follows that $(\boldsymbol{\nu} c)(p \triangleright c \mid P) \approx_{\mathrm{lut}} (\boldsymbol{\nu} c)(p \triangleright c \mid Q)$.

4. As in Part 3, the proof can be derived by Lemma A.6(2) and Proposition 4.3.

$\square$

A.5 Proof of Lemma 9.6

We restate Lemma 9.6.

Lemma A.7 (Operational correspondence of $\{\![\cdot]\!\}$) *Let P be a process in* DLπ.

1. *Suppose that $P \xrightarrow{\mu} P'$. Then we have:*
   (a) *if $\mu = a(b)$ then $\{\![P]\!\} \xrightarrow{a(b')} \gtrsim \{\![P']\!\}\{^{b'}\!/\!_b\}$ and $b' \notin \mathrm{fn}(P)$*
   (b) *if $\mu = \overline{a}b$ then $\{\![P]\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}} (c \to b \mid \{\![P']\!\})$, with $c \notin \mathrm{fn}(P)$*
   (c) *if $\mu = (\boldsymbol{\nu} b)\overline{a}b$ then $\{\![P]\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(c \to b \mid \{\![P']\!\})$, with $c \notin \mathrm{fn}(P)$*
   (d) *if $\mu = d(b)\overline{a}b$ then $\{\![P]\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(d(b').\, b \to b' \mid c \to b \mid \{\![P']\!\})$, with $\{b', c\} \cap \mathrm{fn}(P) = \emptyset$*
   (e) *if $\mu = (\boldsymbol{\nu} d)d(b)\overline{a}b$ then $\{\![P]\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(\boldsymbol{\nu} d)(d(b').\, b \to b' \mid c \to b \mid \{\![P']\!\})$, with $\{b', c\} \cap \mathrm{fn}(P) = \emptyset$*
   (f) *if $\mu = \tau$ then $\{\![P]\!\} \xrightarrow{\tau} \succcurlyeq_{\mathrm{a}} \{\![P']\!\}$.*

2. *Suppose that $\{\![P]\!\} \xrightarrow{\mu} P_1$. Then there exists $P' \in$ DLπ such that:*
   (a) *if $\mu = a(b')$ then $P \xrightarrow{a(b)} P'$ and $P_1 \succcurlyeq_{\mathrm{a}} \{\![P']\!\}\{^{b'}\!/\!_b\}$*
   (b) *if $\mu = (\boldsymbol{\nu} c)\overline{a}c$ then:*
      i. *either $P \xrightarrow{\overline{a}b} P'$ and $P_1 \succcurlyeq_{\mathrm{a}} (c \to b \mid \{\![P']\!\})$, with $c \notin \mathrm{fn}(P)$*
      ii. *or $P \xrightarrow{(\boldsymbol{\nu} b)\overline{a}b} P'$ and $P_1 \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(c \to b \mid \{\![P']\!\})$, with $c \notin \mathrm{fn}(P)$*

iii. or $P \xrightarrow{d(b)\overline{a}b} P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(d(b').\, b \to b' \mid c \to b \mid \{\![P']\!\})$,
with $\{b', c\} \cap \mathrm{fn}(P) = \emptyset$

iv. or $P \xrightarrow{(\boldsymbol{\nu} d)d(b)\overline{a}b} P'$ *and* $P_1 \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(\boldsymbol{\nu} d)(d(b').\, b \to b' \mid c \to b \mid \{\![P']\!\})$,
with $\{b', c\} \cap \mathrm{fn}(P) = \emptyset$

(c) if $\mu = \tau$ *then* $P \xrightarrow{\tau} P'$ *with* $P_1 \succcurlyeq_{\mathrm{a}} \{\![P']\!\}$.

Proof: The proof is by transition induction. We prove Part 1. The proof of Part 2 is similar.

1. $\mu = a(b)$. The interesting case is when the last rule applied to get $P \xrightarrow{a(b)} P'$ is `d-in`. By Lemma 5.2(1), since $\gtrsim$ implies $\succcurlyeq_{\mathrm{a}}$, it holds that

$$\{\!|\, a(b)P \,|\!\} = (\boldsymbol{\nu} b)(a(b').\, b \to b' \mid \{\!|\, P \,|\!\})$$

and

$$(\boldsymbol{\nu} b)(a(b').\, b \to b' \mid \{\!|\, P \,|\!\}) \xrightarrow{a(b')} (\boldsymbol{\nu} b)(b \to b' \mid \{\!|\, P \,|\!\}) \succcurlyeq_{\mathrm{a}} \{\!|\, P \,|\!\}\{b'\!/b\}.$$

2. $\mu = \overline{a}b$. Straightforward.

3. $\mu = (\boldsymbol{\nu} b)\overline{a}b$. Straightforward.

4. $\mu = d(b)\overline{a}b$. The interesting case is when P' is derived by applying rule `o-in`. The result follows from the definition of the encoding.

5. $\mu = (\boldsymbol{\nu} d)d(b)\overline{a}b$. The only significant case is when the last rule applied to get P' is `o-ν`. As in the previous case, the result follows from the definition of the encoding.

6. $\mu = \tau$. The interesting cases are when P' is derived by applying one of the rules `com`, `s-com`, `cls`, or `s-cls`. We consider each of these below.

   (a) Suppose `com` is the last rule applied for deriving $P \xrightarrow{\tau} P'$.

$$\texttt{com:}\ \frac{P \xrightarrow{\overline{a}b} P' \qquad Q \xrightarrow{a(x)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{b\!/x\}}$$

   By induction hypothesis we have:

   - $\{\!|\, P \,|\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}} c \to b \mid \{\!|\, P' \,|\!\}$
   - $\{\!|\, Q \,|\!\} \xrightarrow{a(c)} \succcurlyeq_{\mathrm{a}} \{\!|\, Q' \,|\!\}\{c\!/x\}$

   with $c \notin \mathrm{fn}(P)$. By Lemma 5.2 we have:

$$\begin{aligned}\{\!|\, P \mid Q \,|\!\} = \{\!|\, P \,|\!\} \mid \{\!|\, Q \,|\!\} \xrightarrow{\tau} &\succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} c)(c \to b \mid \{\!|\, P' \,|\!\} \mid \{\!|\, Q' \,|\!\}\{c\!/x\}) \\ &\succcurlyeq_{\mathrm{a}} \{\!|\, P' \,|\!\} \mid \{\!|\, Q' \,|\!\}\{b\!/x\} \\ &= \{\!|\, P' \mid Q'\{b\!/x\} \,|\!\}.\end{aligned}$$

   (b) Suppose `s-com` is the last rule applied for deriving $P \xrightarrow{\tau} P'$.

$$\texttt{s-com:}\ \frac{P \xrightarrow{\overline{a}c} P'}{a(b)P \xrightarrow{\tau} (\boldsymbol{\nu} b)(P'\{c\!/b\})}.$$

   There are two possibilities: either $c \neq b$ or $c = b$.

i. If $c \neq b$ then $(\boldsymbol{\nu} b)(P'\{c/b\}) = P'\{c/b\}$. By induction hypothesis it holds that

$$\{\!|\, P \,|\!\} \xrightarrow{(\boldsymbol{\nu} d)\overline{a}d} \succcurlyeq_{\mathrm{a}} d \to c \mid \{\!|\, P' \,|\!\}$$

with d fresh. Since $\{\!|\, a(b)P \,|\!\} \stackrel{\text{def}}{=} (\boldsymbol{\nu} b)(a(b').\, b \to b' \mid \{\!|\, P \,|\!\})$, by Proposition 4.3(2) and Lemma 5.2 it holds that:

$$\begin{aligned}\{\!|\, a(b)P \,|\!\} \xrightarrow{\tau} &\succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)((\boldsymbol{\nu} d)(b \to d \mid d \to c) \mid \{\!|\, P' \,|\!\}) \\ &\succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(b \to c \mid \{\!|\, P' \,|\!\}) \\ &\succcurlyeq_{\mathrm{a}} \{\!|\, P' \,|\!\}\{c/b\} \\ &= \{\!|\, P\{c/b\} \,|\!\}.\end{aligned}$$

ii. If $c = b$ then $(\boldsymbol{\nu} b)(P'\{c/b\}) = (\boldsymbol{\nu} b)P'$. By induction hypothesis it holds that

$$\{\!|\, P \,|\!\} \xrightarrow{(\boldsymbol{\nu} d)\overline{a}d} \succcurlyeq_{\mathrm{a}} d \to c \mid \{\!|\, P' \,|\!\}$$

with d fresh. Since $\{\!|\, a(b)P \,|\!\} \stackrel{\text{def}}{=} (\boldsymbol{\nu} b)(a(b').\, b \to b' \mid \{\!|\, P \,|\!\})$, by Proposition 4.3(2) and Lemma 9.5 it holds that:

$$\begin{aligned}\{\!|\, a(b)P \,|\!\} \xrightarrow{\tau} &\succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)((\boldsymbol{\nu} d)(b \to d \mid d \to b) \mid \{\!|\, P' \,|\!\}) \\ &\succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(b \to b \mid \{\!|\, P' \,|\!\}) \\ &\succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)\{\!|\, P' \,|\!\}.\end{aligned}$$

(c) Suppose `cls` is the last rule applied for deriving $P \xrightarrow{\tau} P'$.

$$\texttt{cls:}\frac{P \xrightarrow{\beta_b \overline{a}b} P' \quad Q \xrightarrow{a(b')} Q' \quad \mathrm{bn}(\beta_b) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\tau} \beta_b(P' \mid Q'\{b/b'\})}$$

We can suppose $\beta_b = d(b)$ for some d. The case when $\beta_b = (\boldsymbol{\nu} d)d(b)$ is similar. By induction hypothesis it holds that:

- $\{\!|\, P \,|\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(d(b').\, b \to b' \mid c \to b \mid \{\!|\, P' \,|\!\})$, with $\{b', c\} \cap fn(P) = \emptyset$.
- $\{\!|\, Q \,|\!\} \xrightarrow{a(c)} \succcurlyeq_{\mathrm{a}} \{\!|\, Q' \,|\!\}\{c/b'\}$.

We recall that $\{\!|\, P \mid Q \,|\!\} = \{\!|\, P \,|\!\} \mid \{\!|\, Q \,|\!\}$. Then, by Lemma 5.2, we have:

$$\begin{aligned}\{\!|\, P \,|\!\} \mid \{\!|\, Q \,|\!\} \xrightarrow{\tau} &\succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} c)(\boldsymbol{\nu} b)(d(b').\, b \to b' \mid c \to b \mid \{\!|\, P' \,|\!\} \mid \{\!|\, Q' \,|\!\}\{c/b'\}) \\ &\equiv (\boldsymbol{\nu} b)(d(b').\, b \to b' \mid (\boldsymbol{\nu} c)(c \to b \mid \{\!|\, P' \,|\!\} \mid \{\!|\, Q' \,|\!\}\{c/b'\})) \\ &\succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(d(b').\, b \to b' \mid \{\!|\, P' \,|\!\} \mid \{\!|\, Q' \,|\!\}\{b/b'\}) \\ &= \{\!|\, d(b)(\{\!|\, P' \,|\!\} \mid \{\!|\, Q' \,|\!\}\{b/b'\}) \,|\!\}.\end{aligned}$$

(d) Suppose `s-cls` is the last rule applied for deriving $P \xrightarrow{\tau} P'$.

$$\texttt{s-cls:}\frac{P \xrightarrow{\beta_c \overline{a}c} P' \qquad b \notin \mathrm{n}(\beta_c \overline{a}c)}{a(b)P \xrightarrow{\tau} \beta_c(P'\{c/b\})}$$

We can suppose $\beta_c = d(c)$ for some d. The case when $\beta_c = (\boldsymbol{\nu} d)d(c)$ is similar. By induction hypothesis it holds that:

$$\{\!|\, P \,|\!\} \xrightarrow{(\boldsymbol{\nu} h)\overline{a}h} \succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} c)(d(c').\, c \to c' \mid h \to c \mid \{\!|\, P' \,|\!\})$$

with $\{c', h\} \cap fn(P) = \emptyset$.
Since $\{\!|\, a(b)P \,|\!\} \stackrel{\text{def}}{=} (\boldsymbol{\nu} b)(a(b').\, b \to b' \mid \{\!|\, P \,|\!\})$, by Proposition 4.3 and Lemma 5.2 we have:

$$\begin{aligned}\{\!|\, a(b)P \,|\!\} \xrightarrow{\tau} &\succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} h)(\boldsymbol{\nu} b)(b \to h \mid d(c').\, c \to c' \mid h \to c \mid \{\!|\, P' \,|\!\}) \\ &\equiv (\boldsymbol{\nu} b)((\boldsymbol{\nu} h)(b \to h \mid h \to c) \mid d(c').\, c \to c' \mid \{\!|\, P' \,|\!\}) \\ &\succcurlyeq_{\mathrm{a}} (\boldsymbol{\nu} b)(b \to c \mid d(c').\, c \to c' \mid \{\!|\, P' \,|\!\}) \\ &\succcurlyeq_{\mathrm{a}} d(c').\, c \to c' \mid \{\!|\, P' \,|\!\}\{c/b\} \\ &= \{\!|\, d(c)P'\{c/b\} \,|\!\}.\end{aligned}$$

□

A.6 Proof of Lemma 9.13

Proof: We prove that the relation

$$\mathcal{S} = \{(P, [\![P]\!]) : P \in \mathrm{L}\pi\}$$

is a link bisimulation up to expansion. The proof relies on Lemma 5.1 and Proposition 4.3(3). Let us consider the possible actions of P (we reason in a similar manner when dealing with the possible actions of $[\![P]\!]$).

1. If $P \xrightarrow{\tau} P'$, by Lemma 5.1, there exists P_1 such that $[\![P]\!] \xrightarrow{\tau} P_1$ and $P_1 \gtrsim [\![P']\!]$. So, $P' \gtrsim P' \, \mathcal{S} \, [\![P']\!] \lesssim P_1$.

2. If $P \xrightarrow{a(b)} P'$ we reason as in the previous case.

3. If $P \xrightarrow{\overline{a}b} P'$, by Lemma 5.1, P_1 exists such that $[\![P]\!] \xrightarrow{\overline{a}(c)} P_1$ and $P_1 \gtrsim (c \to b \mid [\![P']\!])$, with $c \notin \mathrm{fn}(P')$. Since $\gtrsim$ is a congruence, it holds that

$$(\boldsymbol{\nu} c)(p \rhd c \mid P_1) \gtrsim (\boldsymbol{\nu} c)(p \rhd c \mid c \to b \mid [\![P']\!])$$

for p fresh. By Proposition 4.3(3) it holds that

$$p \rhd b \mid P' \, \mathcal{S} \, [\![p \rhd b \mid P']\!] = p \to b \mid [\![P']\!] \lesssim (\boldsymbol{\nu} c)(p \rhd c \mid P_1).$$

4. If $P \xrightarrow{\overline{a}(b)} P'$, by Lemma 5.1, P_1 exists such that $[\![P]\!] \xrightarrow{\overline{a}(c)} P_1 \gtrsim (\boldsymbol{\nu} b)(c \to b \mid [\![P']\!])$ with $c \notin \mathrm{fn}(P')$. By α-conversion there exists P_2 such that

$$[\![P]\!] \xrightarrow{\overline{a}(b)} P_2 \gtrsim (\boldsymbol{\nu} c)(b \to c \mid [\![P']\!]\{c/b\}).$$

This implies that

$$(\boldsymbol{\nu} b)(p \rhd b \mid P_2) \gtrsim (\boldsymbol{\nu} b)(p \rhd b \mid (\boldsymbol{\nu} c)(b \to c \mid [\![P']\!]\{c/b\})).$$

By Proposition 4.3(3) we have

$$(\boldsymbol{\nu} b)(p \rhd b \mid (\boldsymbol{\nu} c)(b \to c \mid [\![P']\!]\{c/b\})) \gtrsim (\boldsymbol{\nu} c)(p \to c \mid [\![P']\!]\{c/b\}) = (\boldsymbol{\nu} b)(p \to b \mid [\![P']\!]).$$

We conclude by observing that

$$(\boldsymbol{\nu} b)(p \rhd b \mid P') \, \mathcal{S} \, (\boldsymbol{\nu} b)(p \to b \mid [\![P']\!]) \lesssim (\boldsymbol{\nu} b)(p \rhd b \mid P_2).$$

□