Math. Struct. in Comp. Science (2004), *vol.* 14, *pp.* 715–767.
DOI: 10.1017/S0960129504004323 Printed in the United Kingdom

On asynchrony in name-passing calculi†

MASSIMO MERRO‡ and DAVIDE SANGIORGI§

‡*University of Verona, Italy*

§*University of Bologna, Italy*

Received 26 June 2002; revised 1 September 2003

The asynchronous π-calculus has been considered as the basis of experimental programming languages (or proposals for programming languages) like Pict, Join and TyCO. However, on 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, more 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.*

1. Introduction

The *asynchronous* π*-calculus*, abbreviated π_a, is a variant of the π-calculus (Milner *et al.* 1992) in which message emission is non-blocking. Formally, the output prefix $\overline{a}b.P$ of the π-calculus is replaced by the simpler output particle $\overline{a}b$, which has no continuation. The asynchronous π-calculus has been investigated in Honda and Tokoro (1991a), and, independently, by Boudol, who in Boudol (1992) 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 the available distributed systems.

The asynchronous π-calculus has been considered as the basis of experimental concurrent and/or distributed programming languages (or proposals for programming languages) like Pict (Pierce and Turner 1997), Join (Fournet and Gonthier 1996) and TyCO (Vasconcelos 1994).

However, on closer inspection, the programming languages above are based on an even simpler calculus in which:

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

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

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

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 $(\nu a)P$ the inputs at channel a are statically determined: no further inputs at a may be created, inside or outside P. For instance, the process

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

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 a 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 (Walker 1991; Hüttel and Kleist 1996; Sangiorgi 1998; Kleist and Sangiorgi 1998; Philippou and Walker 1996; Sangiorgi 1999b; Merro *et al.* 2002).

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* (Milner and Sangiorgi 1992). 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 π_{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 says, essentially, that inputs cannot be observed in π_{a}. Although this law is useful (it is used for instance by Nestmann and Pierce to prove the correctness of an encoding of guarded choice (Nestmann and Pierce 2000)), it seems fair to say that the restriction to asynchronous contexts does not gain us much.

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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:

$$\overline{a}b = (\nu c)(\overline{a}c \mid !c(x).\overline{b}x) \quad (1)$$

$$(\nu a)(!a(x).R \mid P \mid Q) = (\nu a)(!a(x).R \mid P) \mid (\nu a)(!a(x).R \mid Q) \quad (2)$$

$$(\nu a)(!a(x).R \mid C[\overline{a}b]) = (\nu a)(!a(x).R \mid C[R\{b/x\}]) \quad (3)$$

$$(\nu c)(\overline{a}c) = (\nu c)(\overline{a}c \mid c(x).\mathbf{0}) \quad (4)$$

$$(\nu c)(\overline{a}c) = (\nu c)(\overline{a}c \mid \overline{c}b). \quad (5)$$

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, which is known as one of Milner's *replication theorems* (Milner 1991). This law is true in the π-calculus, under the hypothesis that name a is never transmitted and never used in an 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 that 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 an 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 it 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* (Sangiorgi 1992). Similarly, in π_a, barbed congruence coincides with the closure under substitutions of *asynchronous early bisimilarity* (Amadio *et al.* 1998). 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 (Fournet and Gonthier 1998).

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 in which all names emitted are private. Barbed congruence between processes of Lπ coincides, on their images, with (a slight variant of) *asynchronous ground bisimilarity* (Amadio *et al.* 1998). The second characterisation is based on a new labelled transition system (LTS),

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

which is a modification of the standard one that reveals what is observable in Lπ, that is, what an external observer that behaves like an Lπ process can see by interacting with an 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, while others are new.

Technical differences between our characterisations and those in π_a and π-calculus (Amadio *et al.* 1998; Sangiorgi 1992) 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, Merro *et al.* (2002)). 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.

Outline

In Section 2 we give the 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* (Milner 1991). 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 (Fournet and Gonthier 1996). In Subsection 9.6 we prove some non-full abstraction and full abstraction

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

results for Boreale's encoding (Boreale 1998) 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* (Thielecke 1997) is operationally sound. The paper ends with a discussion and summary of our result and a comparison with related work.

2. The calculus $L\pi$

$L\pi$ is a subset of the asynchronous π-calculus. $L\pi$ 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, that is, when it is not beneath 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 perform 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 (Castellani and Hennessy 1998)). Nestmann and Pierce have shown 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 (Nestmann and Pierce 2000).

$L\pi$, 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 (Honda and Yoshida 1994) and recursion (Milner 1991; Sangiorgi and Walker 2001); (ii) replicated input has a simpler semantics and is handy for implementations. The theory and the results presented in this paper would, however, also hold with full replication or recursion.

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 $L\pi$-processes is

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

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 $(\boldsymbol{\nu} b)P$ act 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 we use $\mathrm{n}(P, Q)$ and $\mathrm{bn}(P, Q)$, similarly. 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.

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Table 1. *Labelled Transition System for* Lπ

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

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

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

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

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 $(\nu a)(\overline{a} \mid a.\,P)$ 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 $(\nu \widetilde{a})P$ for $(\nu a_1)\ldots(\nu a_n)P$.

The *operational semantics* of Lπ is given by means of a *labelled transition system* (LTS) in the SOS style of Plotkin (1981). The LTS is the standard one, in the *late* style (Milner *et al.* 1992; Sangiorgi 1999a), 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) or $\overline{a}(b)$ (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 rest of the paper, we use the symbol $\equiv$ to denote *structural congruence*, a relation used to rearrange the structure of processes (Milner 1991).

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

— $P \mid \mathbf{0} \equiv P$, $P \mid Q \equiv Q \mid P$, $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
— $(\nu a)\mathbf{0} \equiv \mathbf{0}$, $(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$, $(\nu a)(P \mid Q) \equiv P \mid (\nu a)Q$ 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 we said in the introduction, we will focus on bisimulation-based behavioural equivalences,

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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 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 (Amadio *et al.* 1998). The reason is that in asynchronous calculi the observer has no direct way of knowing when an emitted message is received. Below, we define barbed congruence on a generic subset $\mathscr{P}$ of π-calculus processes. A *$\mathscr{P}$-context* is a process of $\mathscr{P}$ with a single hole $[\cdot]$ in it.

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

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

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

With the exception of Fournet and Gonthier (1998), 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 $\mathscr{I}$ of π-processes that is derivation closed and such that $P \in \mathscr{I}$ 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 (Sangiorgi 1992; Amadio *et al.* 1998). In asynchronous calculi *without* matching, like Lπ, early bisimilarity is a congruence, and it coincides with its simpler *ground* variant (Honda 1992; Sangiorgi 2000), which differs from the early one in that there is no universal quantification in the input clause.

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Definition 3.4 (Ground bisimilarities).

— *Synchronous ground bisimulation* is the largest $o\tau$-bisimulation $\mathcal{S}$ on processes such that $P\ \mathcal{S}\ Q$ 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 $P'\ \mathcal{S}\ Q'$.

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

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

1 $Q \stackrel{a(b)}{\Longrightarrow} Q'$ and $P'\ \mathcal{S}\ Q'$; or

2 $Q \Longrightarrow Q'$ and $P'\ \mathcal{S}\ (Q' \mid \overline{a}b)$.

Two processes P and Q are *asynchronous ground bisimilar*, written $P \approx_{\mathrm{a}} Q$, if $P\ \mathcal{S}\ Q$ 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* (Arun-Kumar and Hennessy 1992), 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 $P\ \mathcal{S}\ Q$ implies:

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

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

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

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

1 If $P \xrightarrow{\mu} P'$, μ is not an input, and $\mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \varnothing$, then there exists Q' such that $Q \stackrel{\mu}{\Longrightarrow} Q'$ and $P'\ \mathcal{S}\ Q'$.

2 If $P \xrightarrow{a(b)} P'$, $b \notin \mathrm{fn}(Q)$, there exists Q' such that either:

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

(b) $Q \stackrel{\tau}{\Longrightarrow} Q'$ and $P'\ \mathcal{S}\ (Q' \mid \overline{a}b)$.

3 If $Q \xrightarrow{\mu} Q'$, μ is not an input, and $\mathrm{bn}(\mu) \cap \mathrm{fn}(P) = \varnothing$, then there exists P' such that $P \xrightarrow{\hat{\mu}} P'$ and $P'\ \mathcal{S}\ Q'$.

4 If $Q \xrightarrow{a(b)} Q'$, $b \notin \mathrm{fn}(P)$, then there exists P' such that either:

(a) $P \xrightarrow{a(b)} P'$ and $P'\ \mathcal{S}\ Q'$; or

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

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

4. The link processes

The theory of Lπ is based on special processes called *links* that 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 (Honda and Yoshida 1995) or wires (Sangiorgi and Walker 2001)).

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

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

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 πI, a calculi where all outputs emit private names (Sangiorgi 1996b)).

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

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

Being recursively defined, the process $a \rightarrow b$ is not in Lπ. However, there exists a process in Lπ that is synchronous bisimilar to it. In the following we explain how this process can be built up. In Milner (1991), 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 that does not respect the Lπ constraint on the output capability. This problem can be avoided by rewriting the definition of dynamic links as:

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

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

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

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

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

As we 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 that is bisimilar to the recursive process $a \rightarrow b$. The encoding $\{\!|\cdot|\!\}$ is a slight variant of Milner's encoding (Milner 1991) of polyadic processes into monadic ones[†]. $\{\!|\cdot|\!\}$ is a homomorphism on all operators except input and output for which we have:

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

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

[†] For our purposes it suffices to consider the encoding in which only pairs are taken into account.

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Table 2. *The encoding of free outputs in terms of bound outputs*

$[\![P_1 \mid P_2]\!] \stackrel{\text{def}}{=} [\![P_1]\!] \mid [\![P_2]\!]$	$[\![a(x).P]\!] \stackrel{\text{def}}{=} a(x).[\![P]\!]$	$[\![!a(x).P]\!] \stackrel{\text{def}}{=} !a(x).[\![P]\!]$
$[\![(\nu a)P]\!] \stackrel{\text{def}}{=} (\nu a)[\![P]\!]$	$[\![\overline{a}b]\!] \stackrel{\text{def}}{=} (\nu c)(\overline{a}c \mid c \to b)$	$[\![\mathbf{0}]\!] \stackrel{\text{def}}{=} \mathbf{0}$

To conclude this 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 $(\nu b)(a \triangleright b \mid b \triangleright c) \gtrsim a \triangleright c$
2 $(\nu b)(a \to b \mid b \to c) \gtrsim a \to c$
3 $(\nu b)(a \triangleright b \mid b \to c) \gtrsim a \to c$
4 $(\nu b)(a \to b \mid b \triangleright c) \gtrsim a \to c$.

Proof. Parts 1, 3 and 4 are proved by simply exhibiting the appropriate expansion relations. Part 2 was proved in Boreale (1998) and requires up-to context proof techniques. □

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 a homomorphism on all operators except output. The output particle $\overline{a}b$ is mapped onto the process $(\nu c)(\overline{a}c \mid c \to b)$ where $c \notin \{a, b\}$. This encoding is essentially the asynchronous version of an encoding used by Boreale (Boreale 1998) to compare internal and external mobility. In Boreale's encoding, an output $\overline{a}b$ is mapped onto the synchronous process $(\nu c)(\overline{a}c.\, c \to b)$. This process is behaviourally the same as $(\nu c)(\overline{a}c \mid c \to b)$: in both cases the link $c \to b$ becomes active only when the bound output at a is consumed. Note that the process $(\nu c)(\overline{a}c \mid c \to b)$ coincides with the recursive definition out(a, b) seen in Section 4.

In Section 5.1, we present some results on $[\![\cdot]\!]$ that have already been proved by Boreale (Boreale 1998) for his 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, that is, for each substitution σ we have $[\![P\sigma]\!] = [\![P]\!]\sigma$. In Boreale (1998), Boreale proves an operational correspondence between processes P and $[\![P]\!]$; he also gives an adequacy result for $[\![\cdot]\!]$ with respect to barbed bisimulation.

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Lemma 5.1 (Boreale 1998). 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 either:

(A) $P \xrightarrow{\overline{a}b} P'$, with $c \notin \mathrm{fn}(P')$ and $P_1 \gtrsim (c \to b \mid [\![P']\!])$; or

(B) $P \xrightarrow{\overline{a}(b)} P'$, 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 this lemma relies on Proposition 4.3(2) and the technical, but important, Lemma 5.2. Roughly speaking, Lemma 5.2 says that name substitution on translated terms can be encoded, up-to expansion, using private dynamic links.

Lemma 5.2 (Boreale 1998). Let P be an Lπ-process, and a and b be two names such that $a \neq b$ and a does not occur free in P in input-subject position. Then:

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

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

Theorem 5.3 (Boreale 1998). Let P and Q be two processes in Lπ. Then:

$$P \mathrel{\dot{\approx}} Q \text{ iff } [\![P]\!] \mathrel{\dot{\approx}} [\![Q]\!].$$

5.2. *The proof technique*

In this section we present our translation-based proof technique. Roughly speaking, to verify if two Lπ-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 Lπ 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

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

with the same constraint on received names as in Lπ.

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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. The proof is by structural induction. The most interesting case is when

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

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

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 hold:

1 If $P \xrightarrow{\tau} P'$, there exists Q' such that $Q \Longrightarrow Q'$ and $P'\ \mathcal{S}\ Q'$.
2 If $P \xrightarrow{\overline{a}(b)} P'$, $b \notin \mathrm{fn}(Q)$, there exists Q' such that $Q \xRightarrow{\overline{a}(b)} Q'$ and $P'\ \mathcal{S}\ Q'$.
3 If $P \xrightarrow{a(b)} P'$, $b \notin \mathrm{fn}(Q)$, there exists Q' such that either:
   (a) $Q \xRightarrow{a(b)} Q'$ and $P'\ \mathcal{S}\ Q'$; or
   (b) $Q \Longrightarrow Q'$ 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, so 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 Sangiorgi and Milner (1992) to our new bisimilarity.

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

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

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

Proof. The proof is analogous to that in Sangiorgi and Milner (1992). If $\mathcal{S}$ is a $\asymp_{\mathrm{a}}$-bisimulation up to $\gtrsim$ and $\approx$, one shows that the relation $\approx \mathcal{S} \approx$ is a $\asymp_{\mathrm{a}}$-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 $\mathcal{S} \subseteq \approx \mathcal{S} \approx \subseteq \asymp_{\mathrm{a}}$, the proof is complete. □

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

In the rest of this paper, we often use a $\asymp_a$-*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 $L_{[\![\pi]\!]}$-processes that may perform an output action. Part 2 will be needed in Lemma 5.10(2) to prove that $\asymp_a$ is preserved by parallel composition.

Lemma 5.9. Let P be an $L_{[\![\pi]\!]}$-process.

1 If $P \xrightarrow{\overline{a}(c)} P_1$, then $P \equiv (\nu \widetilde{z})((\nu c)(\overline{a}c \mid c \to b) \mid P_2)$ and $P_1 \equiv (\nu \widetilde{z})(c \to b \mid P_2)$ for some $\widetilde{z}$, b and P_2 such that $\{a, c\} \cap \widetilde{z} = \varnothing$ and $c \notin \mathrm{fn}(P_2)$.
2 If $P \xrightarrow{\overline{a}(c)} P_1$, then $P \approx (\nu c)([\![\overline{a}c]\!] \mid P_1)$.

Proof.

1 The proof is by transition induction.
2 By part 1, $P \equiv (\nu \widetilde{z})((\nu c)(\overline{a}c \mid c \to b) \mid P_2)$ and $P_1 \equiv (\nu \widetilde{z})(c \to b \mid P_2)$ for some $\widetilde{z}$, b and P_2 such that $\{a, c\} \cap \widetilde{z} = \varnothing$ and $c \notin \mathrm{fn}(P_2)$. Picking some fresh name d, we have, using Proposition 4.3(2):

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

□

Lemma 5.10. Let P and Q be two $L_{[\![\pi]\!]}$-processes such that $P \asymp_a Q$. Then:

1 $(\nu a)P \asymp_a (\nu a)Q$.
2 $P \mid R \asymp_a Q \mid R$, for all $L_{[\![\pi]\!]}$-process R.

Proof. We will only give a sketch of the proof here; a detailed proof can be found in Appendix A.

1 We show that the relation

$$\mathscr{S} = \{((\nu a)P\,,\,(\nu a)Q) \,:\, P, Q \in L_{[\![\pi]\!]} \text{ and } P \asymp_a Q\}$$

is a $\asymp_a$-bisimulation up-to structural congruence. The proof is straightforward because the output actions performed by an $L_{[\![\pi]\!]}$-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

$$\mathscr{S} = \{((\nu \tilde{a})(P \mid R), (\nu \tilde{a})(Q \mid R)) \,:\, P, Q, R \in L_{[\![\pi]\!]} \text{ and } P \asymp_a Q\}$$

is a $\asymp_a$-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 $L_{[\![\pi]\!]}$ never perform free output actions. □

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

In Lemma 5.11 we prove that $\asymp_a$ 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 $L_{[\![\pi]\!]}$-processes. Then:

1 $P \asymp_a P$.
2 $P \asymp_a Q$ implies $Q \asymp_a P$.
3 $P \asymp_a Q$ and $Q \asymp_a R$ implies $P \asymp_a R$.

Proof. The only non-trivial property is transitivity. Essentially, we need to prove two results:

— $\asymp_a$ is preserved by injective substitutions. The proof is analogous to that of $\approx$ in the π-calculus.
— $P \asymp_a Q$ and $b \notin \mathrm{fn}(P,Q)$ implies $P \mid [\![\overline{a}b]\!] \asymp_a Q \mid [\![\overline{a}b]\!]$. 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 $\asymp_a$. □

Being in ground style, the bisimilarity $\asymp_a$ is preserved by input prefixing.

Lemma 5.12. Let P and Q be two $L_{[\![\pi]\!]}$-processes such that $P \asymp_a Q$. Then:

1 $a(x).P \asymp_a a(x).Q$.
2 $!a(x).P \asymp_a !a(x).Q$.

Proof. We will only give a sketch of the proof here; a detailed proof can be found in Appendix A.

1 We prove that the relation

$$\mathscr{S} = \{(a(x).P\,, a(x).Q) \,:\, P, Q \in L_{[\![\pi]\!]} \text{ and } P \asymp_a Q\}$$

is a $\asymp_a$-bisimulation. Since $\asymp_a$ is in ground style, it suffices to prove that $\asymp_a$ is preserved by injective substitution. The proof is analogous to that of $\approx$ in the π-calculus.
2 We prove that the relation $\mathscr{S}$ defined by

$$\{(P_1 \mid !a(x).Q_1, P_2 \mid !a(x).Q_2) : P_1, P_2, Q_1, Q_2 \in L_{[\![\pi]\!]},\ P_1 \asymp_a P_2\,,\ Q_1 \asymp_a Q_2\}$$

is a $\asymp_a$-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 $\asymp_a$ is preserved by injective substitutions, parallel composition and restriction. By transitivity of $\asymp_a$ (Lemma 5.11(3)), the conclusion then follows. □

Using Lemmas 5.10, 5.11 and 5.12, we can prove that $\asymp_a$ is a congruence.

Corollary 5.13. $\asymp_a$ 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:

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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

The following is an easy corollary of the above result.

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 $\asymp_{\mathrm{a}}$. We get the result by applying Theorem 5.14. □

Remark 5.16. Theorem 5.14 (respectively, Corollary 5.15) does not hold when replacing $\asymp_{\mathrm{a}}$ (respectively, $\approx$) and barbed congruence with their expansion variants. This is 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, the 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 based, essentially, on the following law, which relates free and bound output actions:

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

However, in Lπ, there is an even simpler law relating 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:

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

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

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

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

From Lemma 5.17, we can derive a 'translation-based' proof technique for barbed congruence that is simpler than that given by Theorem 5.14, whereby in the encoding $[\![\cdot]\!]$, static links $c \triangleright b$ are used instead of dynamic links $c \to b$. 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.

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Table 3. *A new labelled transition system for* Lπ

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

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

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 added 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, given 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 $p \triangleright b$ 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 to 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 $\overset{\mapsto}{\approx}_{\mathrm{a}}$ to denote the relation obtained by replacing arrow $\longrightarrow$ with $\longmapsto$ and arrow $\Longrightarrow$ with $\Longmapsto$ in the definition of asynchronous ground bisimulation (Definition 3.4). 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 $\overset{\mapsto}{\approx}_{\mathrm{a}}$ 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 hold:

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 either:

(a) $Q \overset{a(p)}{\Longrightarrow} Q'$ and $P' \mathcal{S} Q'$; or

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

3 If $P \xrightarrow{\overline{a}b} P'$, then either:

(a) $Q \stackrel{\overline{a}d}{\Longrightarrow} Q'$ and $\exists\, p \notin \mathrm{fn}(P,Q)$ such that $(p \triangleright b \mid P') \;\mathscr{S}\; (p \triangleright d \mid Q')$; or

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

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

(a) $Q \stackrel{\overline{a}b}{\Longrightarrow} Q'$ and $\exists\, p \notin \mathrm{fn}(P,Q)$ such that $(\boldsymbol{\nu} c)(p \triangleright c \mid P') \;\mathscr{S}\; (p \triangleright b \mid Q')$; or

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

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

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 can be seen immediately that $\stackrel{\mapsto}{\approx}_{\mathrm{a}}$ and $\approx_{\mathrm{l}}$ coincide.

Lemma 6.2. Let P and Q be two processes in Lπ. Then

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

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

Lemma 6.3. Let P and Q be two processes in Lπ. Then

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

Proof. We will only give a sketch of the proof here; a detailed proof can be found in Appendix B. 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

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

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

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

is a link bisimulation. □

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

Theorem 6.4 (Soundness). Let P and Q be two processes in Lπ. Then

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

Proof. Apply, in sequence, Lemmas 6.2 and 6.3, and Theorem 5.14. □

Both the 'translation-based' and '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:

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

(i) it uses simpler links $p \triangleright b$ instead of links $p \to b$;
(ii) links are not added for internal communications;
(iii) the input clause uses the particle $\overline{a}b$ instead of $[\![\overline{a}b]\!]$ (which 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 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 (Amadio *et al.* 1998; Sangiorgi 1992), 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 C. Compared to the proofs of similar results in the literature, the non-standard case is when the tested processes perform an output. □

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 Lemmas 7.1 and 6.3 in sequence. Part 2 follows from Theorem 5.14. □

Our translation-based proof technique relies on the algebraic law

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

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

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

(we recall that $c \triangleright b \stackrel{\text{def}}{=} !c(x).\overline{b}x$). 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

$$P \stackrel{\text{def}}{=} \overline{a}b$$
$$Q \stackrel{\text{def}}{=} (\nu c)(\overline{a}c \mid c \to b) = (\nu c)(\overline{a}c \mid !c(x).(\nu d)(\overline{b}d \mid d \to x)).$$

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

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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 $[\![Q]\!] \xrightarrow{\overline{a}(h)} \xrightarrow{h(s)} \xRightarrow{\overline{b}(r)}$.

Now, we prove that the link bisimilarity, that is, 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 $P \overset{\hookrightarrow}{\approx}_{\mathrm{a}} Q$, for P and Q image-finite processes.
2 $P \overset{\hookrightarrow}{\approx}_{\mathrm{a}} Q$ implies $P \cong_{\mathrm{L}\pi} Q$.

Proof. Part 1 follows by applying Lemmas 7.1 and 6.2 in sequence. 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 the presence of matching, the closure under substitutions of asynchronous early bisimilarity on the LTS $\xrightarrow{\mu}$ coincides with barbed congruence in π_{a} (Amadio *et al.* 1998). 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 the standard (*cf.* Definition 8.1) and new (*cf.* Definition 8.3) up-to proof techniques.

We start with the standard one, that is, the up-to-expansion proof technique (Sangiorgi and Milner 1992).

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

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

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

Proof. The proof is analogous to that of standard bisimilarity (Sangiorgi and Milner 1992). □

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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 $(\nu c)(\overline{a}c \mid c \triangleright b)$ are link bisimilar, we have to prove that the derivatives $p \triangleright b$ and $(\nu c)(p \triangleright c \mid c \triangleright b)$ are also link bisimilar. Using our new proof technique, we simply require that $p \triangleright b$ is related to itself (*cf.* Clause 3.*c.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 hold:

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 either:

(a) $Q \overset{a(p)}{\Longrightarrow} Q'$ and $P' \mathcal{S} Q'$; or

(b) $Q \Longrightarrow Q'$ 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 one of:

(a) $Q \overset{\overline{a}b}{\Longrightarrow} Q'$ and $P' \mathcal{S} Q'$; or

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

(c) $Q \overset{\overline{a}(c)}{\Longrightarrow} Q'$, with $c \notin \mathrm{fn}(P)$, and either

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

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

3 If $P \xrightarrow{\overline{a}(c)} P'$, with $c \notin \mathrm{fn}(Q)$ and $p \notin \mathrm{fn}(P, Q)$, then either:

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

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

(ii) $(\nu c)(p \triangleright c \mid P') \mathcal{S} (p \triangleright b \mid Q')$; or

(b) $Q \overset{\overline{a}(c)}{\Longrightarrow} Q'$ and one of

(i) $P' \mathcal{S} Q'$, or

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

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

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

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 (Sangiorgi and Milner 1992) and the up-to-context (Sangiorgi 1994) techniques (in which common contexts are factorised out). However, it cannot be reduced to them because private links may be cut from just 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 the up-to-expansion and up-to-context techniques. We recall that in up-to proof techniques we do not always need to apply a cut to reduce the size of the derivative processes. This explains why we have the subclauses marked by $i, \ldots iv$.

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Finally, we prove that the up-to-link proof technique is sound and complete. The proof of 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 will only give a sketch of the proof here; details can be found in Appendix D. In the implication from left to the right, we prove that the relation

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

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 D.1) saying that

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

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

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

is a link bisimulation. The proof requires the following results (see Lemma D.3):

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

$$(\nu c)(p \triangleright c \mid P) \approx_{\mathrm{lut}} (\nu c)(p \triangleright c \mid Q).$$

4 $P\{p/c\} \approx_{\mathrm{lut}} (p \triangleright b \mid Q)$ and $c \notin \mathrm{fn}(Q)$ implies $(\nu c)(p \triangleright c \mid P) \approx_{\mathrm{lut}} (p \triangleright b \mid Q)$. □

9. Applications of the theory of Lπ

In this section we show a 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 $(\nu u)(\overline{u}b \mid u(a).P)$. In L$\pi$, there exists an alternative way to express name substitution that presents a few advantages (see Sections 9.5 and 9.7, and Merro (2000)).

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

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

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

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

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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 Honda and Yoshida (1995) 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 use $\approx_l^e$ to denote the early variant of $\approx_l$, obtained by replacing the input clause in Definition 6.1 with:

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

(a) (*a*) $Q \stackrel{a(x)}{\Longrightarrow} Q'$ and $P'\{b/x\} \mathrel{\mathscr{S}} Q'\{b/x\}$; or
(b) $Q \Longrightarrow Q'$ and $P'\{b/x\} \mathrel{\mathscr{S}} (Q' \mid \overline{a}b)$.

Proposition 9.2. $P \approx_l^e Q$ iff $P \approx_l Q$.

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

$$\mathscr{S} = \{(P, Q) : P, Q \in \mathrm{L}\pi,\ P \approx_l Q\}$$

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 $Q \stackrel{a(x)}{\Longrightarrow} Q'$ and $P'\{b/x\} \mathrel{\mathscr{S}} Q'\{b/x\}$. The case $b = x$ is trivial, so we suppose $b \neq x$. Since $P \approx_l Q$, there exists Q' such that $Q \stackrel{a(x)}{\Longrightarrow} Q'$ and $P' \approx_l Q'$. 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, $(\nu x)(P' \mid x \triangleright b) \approx_l (\nu x)(Q' \mid x \triangleright b)$ for every name b. By Lemmas 5.2 and 6.3, we have $P'\{b/x\} \approx_l (\nu x)(P' \mid x \triangleright b)$ and $Q'\{b/x\} \approx_l (\nu x)(Q' \mid x \triangleright b)$. By transitivity, we have $P'\{b/x\} \approx_l Q'\{b/x\}$ and, therefore, $P'\{b/x\} \mathrel{\mathscr{S}} Q'\{b/x\}$. □

9.2. *The replication theorems*

The *replication theorems* (Milner 1991) express some 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 $(\nu a)\big(!a(x).R \mid P \mid Q\big) \approx (\nu x)(!a(x).R \mid P) \mid (\nu a)(!a(x).R \mid Q)$.
2 $(\nu a)(!a(x).R \mid !P) \approx !(\nu a)(!a(x).R \mid P)$.

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

the resource $a(x).R$ may change dynamically. To see why this side condition is necessary, take

$$P_1 \stackrel{\text{def}}{=} (\nu a)(!a(x).R \mid \overline{b}a \mid Q)$$
$$P_2 \stackrel{\text{def}}{=} (\nu a)(!a(x).R \mid \overline{b}a) \mid (\nu a)(!a(x).R \mid Q).$$

These processes are not, in general, 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 $(\nu a)\big(!a(x).R \mid P \mid Q\big) \cong_{\mathrm{L}\pi} (\nu x)(!a(x).R \mid P) \mid (\nu a)(!a(x).R \mid Q)$.
2 $(\nu a)(!a(x).R \mid !P) \cong_{\mathrm{L}\pi} !(\nu a)(!a(x).R \mid P)$.

Proof. We code up both members of each equation using $[\![\cdot]\!]_,$. 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. □

9.3. *The delayed input*

In an asynchronous calculus, message emission is non-blocking. Milner, Parrow, Victor, and others have also advocated non-blocking message reception (which is among the motivations for the Update and Fusion calculi (Parrow and Victor 1997; Parrow and Victor 1998) and for the Chi calculus (Fu 1997)). 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 Linear Logic proofs as π-calculus processes (Abramsky 1994; Bellin and Scott 1994). 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 for 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

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

β_b represents the binding part of bound output actions. We extend free and bound names as follows: $\mathrm{fn}((\nu b)\overline{a}b) = \{a\}$, $\mathrm{bn}((\nu b)\overline{a}b) = \{b\}$, $\mathrm{fn}(c(b)\overline{a}b) = \{c, a\}$, $\mathrm{bn}(c(b)\overline{a}b) = \{b\}$,

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Table 4. *Transition rules for delayed input*

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

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

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

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

Table 5. *The encoding* $\{\!|\cdot|\!\}$.

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

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

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

$\mathrm{fn}((\nu c)c(b)\overline{a}b) = \{a\}$, $\mathrm{bn}((\nu c)c(b)\overline{a}b) = \{c, b\}$. 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-`ν;
(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 (van Breugel 1997):

(i) actions have a simpler syntax, because only the output capability of name may be transmitted;
(ii) a restriction (νb) is added in rule `s-com` to model *self communications*, as in $a(b)(\overline{a}b \mid P) \xrightarrow{\tau} (\nu b)P$.

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 a homomorphism on all operators

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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 Yoshida (1998) 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). More 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 $\{\!|\cdot|\!\}$.

In order to prove 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, we have

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

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

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 $\{\!|[\![P]\!]|\!\} \xrightarrow{(\nu c)\overline{a}c} \succcurlyeq_{\mathrm{a}} (c \to b \mid \{\!|[\![P']\!]|\!\})$, with $c \notin \mathrm{fn}(P)$.

(c) If $\mu = (\nu b)\overline{a}b$, then $\{\!|[\![P]\!]|\!\} \xrightarrow{(\nu c)\overline{a}c} \succcurlyeq_{\mathrm{a}} (\nu b)(c \to b \mid \{\!|[\![P']\!]|\!\})$, with $c \notin \mathrm{fn}(P)$.

(d) If $\mu = d(b)\overline{a}b$, then $\{\!|[\![P]\!]|\!\} \xrightarrow{(\nu c)\overline{a}c} \succcurlyeq_{\mathrm{a}} (\nu b)(d(b').\, b \to b' \mid c \to b \mid \{\!|[\![P']\!]|\!\})$, with

$$\{b', c\} \cap \mathrm{fn}(P) = \varnothing.$$

(e) If $\mu = (\nu d)d(b)\overline{a}b$, then

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

with $\{b', c\} \cap \mathrm{fn}(P) = \varnothing$.

(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 $\mu = (\nu c)\overline{a}c$, we have one of the following:

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

(i) $P \xrightarrow{\overline{a}b} P'$ and $P_1 \gtrdot_a (c \to b \mid \{\!|P'|\!\})$, with $c \notin \mathrm{fn}(P)$; or

(ii) $P \xrightarrow{(\nu b)\overline{a}b} P'$ and $P_1 \gtrdot_a (\nu b)(c \to b \mid \{\!|P'|\!\})$, with $c \notin \mathrm{fn}(P)$; or

(iii) $P \xrightarrow{d(b)\overline{a}b} P'$ and $P_1 \gtrdot_a (\nu b)(d(b').\, b \to b' \mid c \to b \mid \{\!|P'|\!\})$, with $\{b', c\} \cap \mathrm{fn}(P) = \varnothing$; or

(iv) $P \xrightarrow{(\nu d)d(b)\overline{a}b} P'$ and $P_1 \gtrdot_a (\nu b)(\nu d)(d(b').\, b \to b' \mid c \to b \mid \{\!|P'|\!\})$, with $\{b', c\} \cap \mathrm{fn}(P) = \varnothing$.

(c) If $\mu = \tau$, then $P \xrightarrow{\tau} P'$ with $P_1 \gtrdot_a \{\!|P'|\!\}$.

Proof. The proof is by transition induction and relies on Proposition 4.3(2) and Lemmas 5.2 and 9.5. Details can be found in Appendix E. □

From Lemma 9.6 we can derive a weak operational correspondence.

Lemma 9.7.

1 If $P \Longrightarrow P'$, then $\{\!|P|\!\} \Longrightarrow \gtrdot_a \{\!|P'|\!\}$.
2 If $\{\!|P|\!\} \Longrightarrow P_1$, then there is P' such that $P \Longrightarrow P'$ and $P_1 \gtrdot_a \{\!|P'|\!\}$.
3 $P \Downarrow_a$ iff $\{\!|P|\!\} \Downarrow_a$.

Proof. Parts 1 and 2 are proved 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

$$P \mathrel{\dot{\approx}} Q \text{ iff } \{\!|P|\!\} \mathrel{\dot{\approx}} \{\!|Q|\!\}.$$

Proof. To prove the implication from left to right, we use Lemma 9.7 to prove that the relation $\mathscr{R} = \{(\{\!|P|\!\}, \{\!|Q|\!\}) : P \mathrel{\dot{\approx}} Q\}$ is a barbed bisimulation up-to $\gtrdot_a$.

To prove the implication from right to left, we use Lemma 9.7 and the fact that $\lesssim_a \mathrel{\dot{\approx}} \gtrdot_a \subset \mathrel{\dot{\approx}}$ to prove that the relation $\mathscr{R} = \{(P, Q) : \{\!|P|\!\} \mathrel{\dot{\approx}} \{\!|Q|\!\}\}$ 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

$$P \mathrel{\dot{\approx}} Q \text{ iff } \{\!|P|\!\} \mathrel{\dot{\approx}} \{\!|Q|\!\}.$$

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

Notice that if $P \in \mathrm{L}\pi$, 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

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Proof. The soundness follows from Lemma 9.9 and the compositionality of $\{\!|\cdot|\!\}$. To prove completeness we want to prove that for every Lπ-context $C[\cdot]$ we have $C[\{\!| P |\!\}] \dot{\approx} C[\{\!| Q |\!\}]$. 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, we have, by hypothesis, $C[P] \dot{\approx} C[Q]$. By Lemma 9.9, this implies $\{\!| C[P] |\!\} \dot{\approx} \{\!| C[Q] |\!\}$. So, it follows that $C[\{\!| P |\!\}] \dot{\approx} C[\{\!| Q |\!\}]$. □

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 $\{\!|\![\cdot]\!|\!\}$), 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 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:

$$a(b)(P \mid Q) = (a(b)P) \mid Q \qquad \text{if } b \notin \mathrm{fn}(Q) \tag{9}$$

$$a(b)c(d)P = c(d)a(b)P \qquad \text{if } c \neq b \text{ and } d \neq a \tag{10}$$

$$(\boldsymbol{\nu} a)(a(x)(\overline{a}x \mid P)) = (\boldsymbol{\nu} x)P \qquad \text{if } a \notin \mathrm{fn}(P) \tag{11}$$

Laws 9 and 10 are similar to structural rules for restriction. Similar laws have been proposed in Boudol (1994). Law 11 transforms a delayed input binder into a restriction binder (it might be interesting to examine delayed input from within action calculi (Milner 1993); for instance, law 11 is reminiscent of the definition of restriction in reflexive action calculi (Milner 1994)).

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 $\overline{a}\langle b_1 \ldots b_n \rangle$ for polyadic outputs and $a(x_1, \ldots x_n).P$ for polyadic inputs. Below, we give Milner's encoding of call-by-name λ-calculus into π-calculus (more precisely, the variant given in Ostheimer and Davie (1993)):

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

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):

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

We can also exploit the delayed input operator, which is a derived operator in Lπ, to get an encoding of the *strong* call-by-name strategy, where reductions can also occur beneath

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

an abstraction (that is, 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. More precisely, we have to replace this input with a delayed input:

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

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

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

Results of operational correspondence and validity of β-reduction, similar to those in Milner (1992b) and Sangiorgi (1994) for the call-by-name λ-calculus, hold for this encoding. This allows us to 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 (Sangiorgi 1994; Sangiorgi 2000).

9.5. *Some properties of the Join calculus*

In this section we apply the theory of Lπ to prove some laws in Fournet and Gonthier's Join calculus (Fournet and Gonthier 1996), which is a calculus for distributed and concurrent programming.

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

$$P ::= a\langle b\rangle \ \mid\ P_1 \mid P_2 \ \mid\ \texttt{def}\ a\langle x\rangle \mid b\langle y\rangle = P_1\ \texttt{in}\ P_2.$$

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 $\texttt{def}\ a\langle x\rangle \mid b\langle y\rangle = P_1\ \texttt{in}\ P_2$ is a sort of amalgamation of the operators of replication, parallel composition and restriction, which allows one to model the joint reception of values from different channels.

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

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

As with 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 (Lévy 1997). With J abbreviating a join pattern $a\langle x\rangle \mid b\langle y\rangle$, the main (simplified) reduction rule is

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

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, this instance may be replaced by the corresponding instantiation $P\sigma$ of the defined continuation P.

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Table 6. *Mapping of the Join calculus into* Lπ

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

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

The construct $\texttt{def}\ a\langle x\rangle \mid b\langle y\rangle = P_1\ \texttt{in}\ P_2$ 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 in one definition only (see also Amadio (1997) and Sangiorgi (1999a)).

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

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

which can be seen as an abbreviation for $\texttt{def}\ a\langle x\rangle \mid b\langle y\rangle = P_1 \mid y\langle y\rangle\ \texttt{in}\ P_2 \mid b\langle b\rangle$, 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 (Fournet and Gonthier 1996)). 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 Fournet (1999). □

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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

$$\texttt{def}\ a\langle x\rangle = R\ \texttt{in}\ P \mid Q \ \cong_{\mathrm{J}}\ (\texttt{def}\ a\langle x\rangle = R\ \texttt{in}\ P) \mid (\texttt{def}\ a\langle x\rangle = R\ \texttt{in}\ Q) \tag{13}$$

$$\texttt{def}\ a\langle x\rangle = b\langle x\rangle\ \texttt{in}\ P \ \cong_{\mathrm{J}}\ P\{{}^{b}/_{a}\} \ \text{ if } \ a \neq b \tag{14}$$

$$\texttt{def}\ a\langle x\rangle = P\ \texttt{in}\ (Q \mid a\langle b\rangle) \ \cong_{\mathrm{J}}\ \texttt{def}\ a\langle x\rangle = P\ \texttt{in}\ (Q \mid P\{{}^{b}/_{x}\}) \tag{15}$$

$$\texttt{def}\ a\langle x\rangle = P\ \texttt{in}\ C[a\langle b\rangle] \ \cong_{\mathrm{J}}\ \texttt{def}\ a\langle x\rangle = P\ \texttt{in}\ C[P\{{}^{b}/_{x}\}] \tag{16}$$

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 Theorem 9.4(1) again.

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 $(\!|\cdot|\!)$ and the theory of π_{a} or π-calculus: the encodings of the processes in the laws are behaviourally different in both π_{a} and π-calculus.

9.6. *External versus internal mobility*

The π-calculus mobility mechanism can be divided into *internal* mobility and *external* mobility (Sangiorgi 1996b). 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:

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

In πI, recursive definitions are more appropriate than replication. This is because, when only internal mobility is allowed, recursion is strictly more expressive than replication (Sangiorgi 1996b). 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 (Milner *et al.* 1992).

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 this calculus $\pi\mathrm{I}_{\mathrm{a}}$.

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

In Boreale (1998), 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 first maps the asynchronous π-calculus into Lπ; the second (which is 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 (Sangiorgi 1996b)) 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

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

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

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

We have $A \not\Downarrow_b$, while, for all processes B such that $[\![Q]\!] \mid R \Longrightarrow B$, we have $B \Downarrow_b$. 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 refer to this variant of πI as *Localised* πI, which we abbreviate to LπI.

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

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

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 E. □

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

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Proof. The implication from right to left follows from Theorem 5.3 and the compositionality of $[\![\cdot]\!]$. To prove the implication from left to right, we want to prove that for all contexts $C[\cdot]$ in LπI we have$C[[\![P]\!]] \mathrel{\dot{\approx}} C[[\![Q]\!]]$. 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 we have $\hat{C}[[\![P]\!]] \mathrel{\dot{\approx}} \hat{C}[[\![Q]\!]]$. With 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 $C[[\![P]\!]] \approx \hat{C}[[\![P]\!]] \mathrel{\dot{\approx}} \hat{C}[[\![Q]\!]] \approx C[[\![Q]\!]]$. Since $\approx \mathrel{\dot{\approx}} \approx \subseteq \mathrel{\dot{\approx}}$ 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 (Thielecke 1997). The CPS-calculus, which is similar to the intermediate language of Appel's compiler (Appel 1992), was 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:

$$M ::= a\langle b\rangle \mid M\{a\langle b\rangle \Leftarrow M\}.$$

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 Appel (1992). We write $\mathrm{CPS} \vdash M{=}N$ to denote the fact that the equality $M = N$ can be derived by the above axiomatic semantics.

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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 relies strongly 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 $a\langle b\rangle\{a_1\langle b_1\rangle \Leftarrow M_1\}\dots\{a_n\langle b_n\rangle \Leftarrow M_n\}$for some $n \geqslant 0$. This allows us to model the behaviour of CPS-terms by means of just one (global) reduction rule:

$$a_i\langle b\rangle\{a_1\langle b_1\rangle \Leftarrow M_1\}\dots\{a_i\langle b_i\rangle \Leftarrow M_i\}\dots\{a_n\langle b_n\rangle \Leftarrow M_n\}$$
$$\rightsquigarrow$$
$$M_i\{b/b_i\}\{a_1\langle b_1\rangle \Leftarrow M_1\}\dots\{a_i\langle b_i\rangle \Leftarrow M_i\}\dots\{a_n\langle b_n\rangle \Leftarrow M_1\}$$

where $1 \leqslant i \leqslant n$ and $a_j \notin \mathrm{fv}(M_i)$ for $1 \leqslant j < i$. 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 $b, a_1, b_1, \dots, a_n, b_n$, for some integer $n \geqslant 0$ with $a \neq a_i$ for every $1 \leqslant i \leqslant n$, such that $M = a\langle b\rangle\{a_1\langle b_1\rangle \Leftarrow M_1\}\dots\{a_n\langle b_n\rangle \Leftarrow M_n\}$. 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 use $\cong_{\mathrm{CPS}}$ to denote the barbed congruence on CPS terms. We prove the soundness of the axiomatic semantics with respect to $\cong_{\mathrm{CPS}}$ by exploiting a straightforward encoding of the CPS-calculus into π-calculus that has already appeared in Thielecke (1997):

— $(\!|\, a\langle b\rangle \,|\!) \stackrel{\mathrm{def}}{=} \overline{a}b$

— $(\!|\, M\{a\langle b\rangle \Leftarrow N\} \,|\!) \stackrel{\mathrm{def}}{=} (\nu a)((\!|\, M \,|\!) \mid\, !a(b).(\!|\, N \,|\!))$.

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 $(\!|\, M \,|\!) \xrightarrow{\tau} \equiv (\!|\, N \,|\!)$.
2 If $(\!|\, M \,|\!) \xrightarrow{\tau} P$, 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. □

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

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Proof. The proof follows from the operational correspondence in Lemma 9.17 and the compositionality of $(\!|\cdot|\!)$. □

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

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

Proof. Since $\cong_{\mathrm{L}\pi}$ 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 $(\!|M|\!) \cong_{\mathrm{L}\pi} (\!|N|\!)$. 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 (Milner 1991).
2 (GC) It suffices to show that

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

by exhibiting the appropriate bisimulation. The conclusion then follows since $\approx$ implies $\cong_{\mathrm{L}\pi}$.
3 (JMP) It suffices to show that

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

by proving that the relation

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

is a synchronous ground bisimulation up to $\equiv$. The conclusion then follows since $\approx$ implies $\cong_{\mathrm{L}\pi}$.
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. The proof is 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, which is 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 new up-to proof techniques. We have used our proof techniques to prove a number of

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Table 7. *Calculi and Equivalences*

Calculi:		
	$\mathrm{L}\pi$	Definition 2.1
	$\mathrm{L}_{[\![\pi]\!]}$	Section 5.2
	$\mathrm{DL}\pi$	Section 9.3
Relationship:		
	$\mathrm{L}_{[\![\pi]\!]} \subset^{\dagger} \mathrm{L}\pi \subset \mathrm{DL}\pi$	
Equivalences in $\mathrm{L}\pi$***:***		
	$\cong_{\mathrm{L}\pi}$	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.

algebraic laws and applications. Table 7 summarises the main calculi, with the associated equivalences, studied in the paper.

Calculi similar to Localised π are discussed in Honda and Tokoro (1991b), Fournet and Gonthier (1996), Amadio (1997), Boreale (1998) and Yoshida (1998). A number of characterisations of barbed congruence on asynchronous mobile processes exist (Amadio *et al.* 1998; Fournet and Laneve 2001; Amadio 1997). In Fournet and Laneve (2001), the authors prove Law 15 of Section 9.5 using a form of labelled bisimilarity. However, the labelled bisimilarities of Amadio *et al.* (1998), Fournet and Laneve (2001) and Amadio (1997) cannot 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 Fournet and Gonthier (1996) and Fournet (1999).

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Other studies on barbed congruence, or similar context-based bisimulations, for mobile processes have been conducted in Honda and Yoshida (1995), Yoshida (1996), Kobayashi *et al.* (1999), Hennessy and Riely (1998) and, for a coordination language, in Busi *et al.* (1998). Fournet and Gonthier (1998) showed that the hypothesis of image-finiteness can be dropped when characterising barbed congruence in asynchronous π-calculus. Boreale and Sangiorgi (1998) 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 Boreale and Sangiorgi (1998), 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 Boreale and Sangiorgi (1998) 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 in Pierce and Sangiorgi (1996) using the type system with input/output capabilities. The same result is proved in Sangiorgi (1999a) by previously translating processes (by means of an encoding very close to our $[\![\cdot]\!]$) and then proving that the images 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 have proposed easier proofs without using typed bisimulations. However, while the results in Pierce and Sangiorgi (1996) and Sangiorgi (1999a) hold 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 (Milner 1991; Pierce and Sangiorgi 1996). The same result holds in Lπ, though 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 in Boudol (1994), Bellin and Scott (1994), Fu (1997), Parrow and Victor (1998), Yoshida (1998), van Breugel (1997) and Laneve and Victor (1999). Bellin and Scott give a reduction semantics for a version of π-calculus that was proposed by Milner and in which both message emission and message reception are non-blocking (Milner 1992a); van Breugel defines a labelled transition system for such a 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 (Laneve and Victor 1999). 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 (Parrow and Victor 1998; Fu 1997).

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

In Join and Blue calculus, polymorphic type systems *à la* ML have been introduced (Fournet *et al.* 1997; Dal-Zilio 1997). 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.

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

By the time the writing of this paper was completed, the theory of Lπ had already been used in other work. In Merro (2000), we give an encoding of polyadic Lπ into monadic Lπ. Unlike Milner's encoding of polyadic π into monadic π (Milner 1991), the encoding in (Merro 2000) is fully-abstract with respect to barbed congruence. In Sangiorgi (2001), 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 (Sangiorgi 1992; Sangiorgi 1996a). In Merro *et al.* (2002) Lπ is used to give a translational semantics of (an appropriate abstraction of) Cardelli's distributed object-based programming language Obliq (Cardelli 1995). The theory of Lπ (more 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 (Sangiorgi 2002).

Appendix A. 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

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

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

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

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

(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) $R \xrightarrow{b(c)} R'$, for some R'. We can proceed as in case 2.a.i.

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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

(A) $Q \xRightarrow{b(c)} Q'$, with $P' \asymp_a Q'$, and the conclusion follows easily.

(B) $Q \Longrightarrow Q'$, with $P' \asymp_a Q' \mid [\![\overline{b}c]\!]$. This means that

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

with

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

which is enough because $\equiv$ is contained in $\approx$ and $\mathcal{S}$ is a $\asymp_a$-bisimulation up to $\gtrsim$ and $\approx$.

(c) If $(\nu\tilde{a})(P \mid R) \xrightarrow{\tau} A$, 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 $(\nu\tilde{a})(P \mid R) \xrightarrow{\tau} (\nu\tilde{a}c)(P' \mid R')$. Since $P \asymp_a Q$, there exists Q' such that $Q \xRightarrow{\overline{b}(c)} Q'$ and $P' \asymp_a Q'$. This means that $(\nu\tilde{a})(Q \mid R) \Longrightarrow (\nu\tilde{a}c)(Q' \mid R')$ and $((\nu\tilde{a}c)(P' \mid R'), (\nu\tilde{a}c)(Q' \mid R')) \in \mathcal{S}$.

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

(A) There exists Q' such that $Q \xRightarrow{b(c)} Q'$ with $P' \asymp_a Q'$, and the conclusion follows.

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

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

with

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

3 It suffices to show that the relation

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

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

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

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

is a $\asymp_a$-bisimulation up to structural congruence (this 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 $P_1 \asymp_a P_2$, 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 $Q_1 \asymp_a Q_2$, and $\asymp_a$ is preserved by

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

injective substitutions, we have $Q_1\{w/x\} \asymp_a Q_2\{w/x\}$. Lemma 5.10(2) then gives $P_1 \mid Q_1\{w/x\} \asymp_a P_2 \mid Q_1\{w/x\} \asymp_a P_2 \mid Q_2\{w/x\}$.
By transitivity (Lemma 5.11(3)), we conclude that $P_1 \mid Q_1\{w/x\} \mid !a(x).Q_1 \mathrel{\mathscr{S}} P_2 \mid Q_2\{w/x\} \mid !a(x).Q_2$.

(c) $P_1 \xrightarrow{\overline{a}(c)} P_1'$ and $P_1 \mid !a(x).Q_1 \xrightarrow{\tau} (\nu c)(P_1' \mid Q_1\{c/x\}) \mid !a(x).Q_1$. Since $P_1 \asymp_a P_2$, there exists P_2' such that $P_2 \xRightarrow{\overline{a}(c)} P_2'$ and $P_1' \asymp_a P_2'$. This means that $P_2 \mid !a(x).Q_2 \Longrightarrow (\nu c)(P_2' \mid Q_2\{c/x\}) \mid !a(x).Q_2$. 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. □

Appendix B. Proof of Lemma 6.3

We have 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 B.1 (Boreale 1998). Let P be an Lπ-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 $[\![P]\!] \xRightarrow{\overline{a}(p)} \gtrsim (\nu b)(p \to b \mid [\![P']\!])$, with $p \notin \mathrm{fn}(P')$.
   (d) If $\alpha = \tau$, then $[\![P]\!] \xRightarrow{\tau} \gtrsim [\![P']\!]$.

2 Suppose that $[\![P]\!] \xRightarrow{\alpha} P_1$. Then there exists $P' \in \mathrm{L}\pi$ such that:
   (a) If $\alpha = a(b)$, then $P \xRightarrow{a(b)} P'$, with $P_1 \gtrsim [\![P']\!]$.
   (b) If $\alpha = \overline{a}(p)$, then either:
      (i) $P \xRightarrow{\overline{a}b} P'$, with $p \notin \mathrm{fn}(P')$ and $P_1 \gtrsim (p \to b \mid [\![P']\!])$; or
      (ii) $P \xRightarrow{\overline{a}(b)} P'$, with $p \notin \mathrm{fn}(P')$ and $P_1 \gtrsim (\nu b)(p \to b \mid [\![P']\!])$.
   (c) If $\alpha = \tau$, then $P \xRightarrow{\tau} P'$ with $P_1 \gtrsim [\![P']\!]$.

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

Lemma B.2. Let P and Q be two processes in Lπ. Then

$$P \approx_1 Q \text{ iff } [\![P]\!] \asymp_a [\![Q]\!].$$

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

$$\mathscr{S} = \{([\![P]\!], [\![Q]\!]) \mid P, Q \in \mathrm{L}\pi \text{ and } P \approx_1 Q\}$$

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

1 If $[\![P]\!] \xrightarrow{\tau} P_1$, by Lemma 5.1, there exists P' such that $P \xrightarrow{\tau} P'$ and $P_1 \gtrsim [\![P']\!]$. Since $P \approx_1 Q$, there exists Q' such that $Q \Longrightarrow Q'$ and also $P' \approx_1 Q'$. By Lemma B.1, there exists Q_1 such that $[\![Q]\!] \xRightarrow{\tau} Q_1 \gtrsim [\![Q']\!]$.

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

2 If $[\![P]\!] \xrightarrow{a(c)} P_1$, by Lemma 5.1, there exists P' such that $P \xrightarrow{a(c)} P'$ and $P_1 \gtrsim [\![P']\!]$. Since $P \approx_1 Q$ there are two possibilities:

(a) There exists Q' such that $Q \stackrel{a(c)}{\Longrightarrow} Q'$ and $P' \approx_1 Q'$. Then, by Lemma B.1, 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 $P' \approx_1 Q' \mid \overline{a}c$. Then, by Lemma B.1, 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 $[\![P]\!] \xrightarrow{\overline{a}(p)} P_1$, assuming p fresh, by Lemma 5.1, there are two possible cases:

(a) $P \xrightarrow{\overline{a}b} P'$ and $P_1 \gtrsim (p \to b \mid [\![P']\!]) = [\![p \triangleright b \mid P']\!]$. Since $P \approx_1 Q$, there are still two subcases:

(i) There exists a process Q' such that $Q \stackrel{\overline{a}d}{\Longrightarrow} Q'$ and $(p \triangleright b \mid P') \approx_1 (p \triangleright d \mid Q')$. By Lemma B.1, 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, $[\![P]\!] \xrightarrow{\overline{a}(p)} P_1 \gtrsim [\![p \triangleright b \mid P']\!]$, and $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim [\![p \triangleright d \mid Q']\!]$, and the conclusion follows since $([\![p \triangleright b \mid P']\!], [\![p \triangleright d \mid Q']\!]) \in \mathscr{S}$.

(ii) There exists a process Q' such that $Q \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$ and $(p \triangleright b \mid P') \approx_1 (\nu c)(p \triangleright c \mid Q')$. By Lemma B.1, there exists Q_1 such that $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim (\nu c)(p \to c \mid [\![Q']\!])$. So, $[\![P]\!] \xrightarrow{\overline{a}(p)} P_1 \gtrsim [\![p \triangleright b \mid P']\!]$ and $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim [\![(\nu c)(p \triangleright c \mid Q')]\!]$, and the conclusion follows since $([\![p \triangleright b \mid P']\!], [\![(\nu c)(p \triangleright c \mid Q')]\!]) \in \mathscr{S}$.

(b) $P \xrightarrow{\overline{a}(c)} P'$ and $P_1 \gtrsim (\nu c)(p \to c \mid [\![P']\!]) = [\![(\nu c)(p \triangleright c \mid P')]\!]$. Since $P \approx_1 Q$, there are two possibilities:

(i) There exists Q' such that $Q \stackrel{\overline{a}b}{\Longrightarrow} Q'$ and $(\nu c)(p \triangleright c \mid P') \approx_1 (p \triangleright b \mid Q')$. By Lemma B.1, there exists a process Q_1 such that $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim (p \to b \mid Q')$. So, $[\![P]\!] \xrightarrow{\overline{a}(p)} P_1 \gtrsim [\![(\nu c)(p \triangleright c \mid P')]\!]$ and $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim [\![p \triangleright b \mid Q']\!]$, and the conclusion follows since $([\![(\nu c)(p \triangleright c \mid P')]\!], [\![p \triangleright b \mid Q']\!]) \in \mathscr{S}$.

(ii) There exists Q' such that $Q \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$ and $(\nu c)(p \triangleright c \mid P') \approx_1 (\nu c)(p \triangleright c \mid Q')$. By Lemma B.1, there exists a process Q_1 such that $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim (\nu c)(p \to c \mid [\![Q']\!])$. So, $[\![P]\!] \xrightarrow{\overline{a}(p)} P_1 \gtrsim [\![(\nu c)(p \triangleright c \mid P')]\!]$ and $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1 \gtrsim [\![(\nu c)(p \triangleright c \mid Q')]\!]$, and since $([\![(\nu c)(p \triangleright c \mid P')]\!], [\![(\nu c)(p \triangleright c \mid Q')]\!]) \in \mathscr{S}$, the conclusion follows.

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

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

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 $[\![Q]\!] \stackrel{\tau}{\Longrightarrow} Q_1$ and $P_1 \asymp_{\mathrm{a}} Q_1$. By Lemma B.1, there exists a process Q' such that $Q \stackrel{\tau}{\Longrightarrow} Q'$ 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 $(P', Q') \in \mathscr{S}$.

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:

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

(a) There exists a processes Q_1 such that $[\![Q]\!] \stackrel{a(c)}{\Longrightarrow} Q_1$ and $P_1 \asymp_a Q_1$. By Lemma B.1, there exists a process Q' such that $Q \stackrel{a(c)}{\Longrightarrow} Q'$ and $Q_1 \gtrsim [\![Q']\!]$. So we have $[\![P']\!] \lesssim P_1 \asymp_a Q_1 \gtrsim [\![Q']\!]$. It therefore follows that $(P', Q') \in \mathscr{S}$.

(b) There exists Q_1 such that $[\![Q]\!] \Longrightarrow Q_1$ and $P_1 \asymp_a Q_1 \mid [\![\overline{a}c]\!]$. By Lemma B.1, 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

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

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

3 If $P \xrightarrow{\overline{a}b} P'$, by Lemma 5.1, there exists P_1 such that $[\![P]\!] \xrightarrow{\overline{a}(p)} P_1 \gtrsim p \to b \mid [\![P']\!] = [\![p \triangleright b \mid P']\!]$. Since $[\![P]\!] \asymp_a [\![Q]\!]$, there exists Q_1 such that $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1$ and $P_1 \asymp_a Q_1$. By Lemma B.1, we have two possibilities:

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

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

4 If $P \xrightarrow{\overline{a}(c)} P'$, by Lemma 5.1, there exists P_1 such that $[\![P]\!] \xrightarrow{\overline{a}(p)} P_1 \gtrsim (\nu c)(p \to c \mid [\![P']\!]) = [\![(\nu c)(p \triangleright c \mid P')]\!]$. Since $[\![P]\!] \asymp_a [\![Q]\!]$, there exists Q_1 such that $[\![Q]\!] \stackrel{\overline{a}(p)}{\Longrightarrow} Q_1$ and $P_1 \asymp_a Q_1$. By Lemma B.1, we have two possibilities:

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

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

Appendix C. Proof of Lemma 7.1

Proof. Let $\simeq_l$ be the variant of $\approx_l$ 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. We have $\simeq_l = \approx_l$ and $\dot{\simeq} = \dot{\approx}$. So we prove that

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

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

$$\forall R \in \mathrm{L}\pi \;\; (P \mid R \;\dot{\simeq}\; Q \mid R) \;\text{ implies that }\; P \simeq_l^\omega Q.$$

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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

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

$$\begin{aligned}
\forall\, R \in \mathrm{L}\pi\ (P \mid R \mathrel{\dot{\simeq}} Q \mid R) \text{ implies } & (\boldsymbol{\nu}\mathscr{L}')(P \mid R(n, \mathscr{L}, \mathscr{M})) \mathrel{\dot{\simeq}} (\boldsymbol{\nu}\mathscr{L}')(Q \mid R(n, \mathscr{L}, \mathscr{M})) \\
& \text{for each } n \in \omega \text{ with } \mathscr{L} = \mathrm{fn}(P, Q), \\
& \mathscr{M} = \varnothing, \text{ and } \mathscr{L}' = \varnothing. \\
\text{implies } & \forall\, n \in \omega\ \ P \simeq_{\mathrm{l}}^{n} Q \\
\text{implies } & P \simeq_{\mathrm{l}}^{\omega} Q \\
\text{implies } & P \simeq_{\mathrm{l}} Q.
\end{aligned}$$

In order to define the tests $R(n, \mathscr{L}, \mathscr{M})$, 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 $\mathscr{L}' \subseteq \mathscr{L}$, $\mathscr{M} \subseteq \mathscr{L}$, $\mathscr{L} \subseteq Ch''$. 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 $R(n, \mathscr{L}, \mathscr{M})$ is defined by induction on n below. In the definition we pick a' to be the first name in the ordered set $Ch'' \setminus \mathscr{L}$. When emitting or receiving a name that is not in $\mathscr{L}$ we work up to injective substitution to show that $P \simeq_{\mathrm{l}}^{n} Q$. The relation $R(n, \mathscr{L}, \mathscr{M})$ is defined as follows:

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

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

Notice that, unlike the characterisation proof of Amadio *et al.* (1998), 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 of distinguishing names in output object position.

The proof is by induction on n. The case $n = 0$ is trivial because $\simeq_{\mathrm{l}}^{0} = \mathrm{L}\pi \times \mathrm{L}\pi$. If $n > 0$, by induction, we suppose that

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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

1 $\mu = \tau$. Then:

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

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

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

We make a further reduction on the left-hand side:

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

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

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

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

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

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

We take a further step on the left-hand side:

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

We take a further step on the left-hand side:

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

This can be matched by either:

(a) $(\boldsymbol{\nu}\mathscr{L}')(Q_1 \mid \overline{c_n^a} \oplus a(x).(a' \triangleright x \mid R(n-1, \mathscr{L} \cup \{a'\}, \mathscr{M} \cup \{a'\},))) \stackrel{\tau}{\Longrightarrow}$
$(\boldsymbol{\nu}\mathscr{L}')(Q' \mid a' \triangleright d \mid R(n-1, \mathscr{L} \cup \{a'\}, \mathscr{M} \cup \{a'\}))$,
which means that $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\overline{a}d}{\Longrightarrow} Q'$, and by the induction hypothesis, and reasoning up to structural congruence, it follows that $a' \triangleright b \mid P' \simeq_1^{n-1} a' \triangleright d \mid Q'$; or

(b) $(\boldsymbol{\nu}\mathscr{L}')(Q_1 \mid \overline{c_n^a} \oplus a(x).(a' \triangleright x \mid R(n-1, \mathscr{L} \cup \{a'\}, \mathscr{M} \cup \{a'\}))) \stackrel{\tau}{\Longrightarrow}$
$(\boldsymbol{\nu}\mathscr{L}')((\boldsymbol{\nu}c)(Q' \mid a' \triangleright c) \mid R(n-1, \mathscr{L} \cup \{a'\}, \mathscr{M} \cup \{a'\}))$,
which means that $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\overline{a}(c)}{\Longrightarrow} Q'$, and by the induction hypothesis, and reasoning up to structural congruence, it follows that $a' \triangleright b \mid P' \simeq_1^{n-1} (\boldsymbol{\nu}c)(a' \triangleright c \mid Q')$.

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

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

This has to be matched by

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

We take a further step on the left-hand side:

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

This can be matched by either:

(a) $(\nu\mathscr{L}')(Q_1 \mid \overline{c_n^a} \oplus a(x).(a' \triangleright x \mid R(n-1,\mathscr{L}\cup\{a'\},\mathscr{M}\cup\{a'\}))) \stackrel{\tau}{\Longrightarrow}$
$(\nu\mathscr{L}')(Q' \mid a' \triangleright b \mid R(n-1,\mathscr{L}\cup\{a'\},\mathscr{M}\cup\{a'\}))$,
which means that $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\overline{a}b}{\Longrightarrow} Q'$, and by the induction hypothesis, and reasoning up to structural congruence, it follows that $(\nu c)(a' \triangleright c \mid P') \simeq_1^{n-1} a' \triangleright b \mid Q'$; or

(b) $(\nu\mathscr{L}')(Q_1 \mid \overline{c_n^a} \oplus a(x).(a' \triangleright x \mid R(n-1,\mathscr{L}\cup\{a'\},\mathscr{M}\cup\{a'\}))) \stackrel{\tau}{\Longrightarrow}$
$(\nu\mathscr{L}')((\nu d)(Q' \mid a' \triangleright d) \mid R(n-1,\mathscr{L}\cup\{a'\},\mathscr{M}\cup\{a'\}))$,
which means that $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\overline{a}(d)}{\Longrightarrow} Q'$, and since $c \notin \mathrm{fn}(Q)$, we also have that $Q \stackrel{\tau}{\Longrightarrow} Q_1$ and $Q_1 \stackrel{\overline{a}(c)}{\Longrightarrow} Q'\{c/d\}$, and by the induction hypothesis, and reasoning up to structural congruence, it follows that $(\nu c)(a' \triangleright c \mid P' \simeq_1^{n-1} (\nu c)(a' \triangleright c \mid Q')$.

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

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

This has to be matched by

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

We make a further reduction on the left-hand side:

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

This is matched by

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

We have two possibilities:

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

Then $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{\tau}{\Longrightarrow} Q'$ and $P' \simeq_1^{n-1} Q' \mid \overline{a}a'$ by the induction hypothesis and up to structural congruence.

(b) $Q_1 \stackrel{aa'}{\Longrightarrow} Q'$ and $Q'' \equiv (\nu\mathscr{L}'a')(Q' \mid R(n-1,\mathscr{L}\cup\{a',\mathscr{M}\}))$. Then $Q \stackrel{\tau}{\Longrightarrow} Q_1 \stackrel{aa'}{\Longrightarrow} Q'$ and $P' \simeq_1^{n-1} Q'$ by the induction hypothesis and up to structural congruence. □

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Appendix D. Complement to the proof of Theorem 8.4

Lemma D.1. Let P and Q be two Lπ-processes and p be 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

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

is a link bisimulation up to structural congruence. The proof is straightforward and 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 D.2. Let P and Q be two Lπ-processes and b be 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

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

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

1 $P \mid \overline{a}b \xrightarrow{\alpha} P' \mid \overline{a}b$, that is, $P \xrightarrow{\alpha} P'$. In this case the process $Q \mid \overline{a}b$ can match these actions in the same way.
2 $P \mid \overline{a}b \xrightarrow{\alpha} P''$, that is, the output particle $\overline{a}b$ is consumed. Then there are two possibilities:
   (a) $P \mid \overline{a}b \xrightarrow{\overline{a}b} P$. In this case the process $Q \mid \overline{a}b$ can match these actions in the same way.
   (b) $P \mid \overline{a}b \xrightarrow{\tau} P'$. This means that $P \xrightarrow{a(b)} P'$. Since $P \approx_{\mathrm{lut}} Q$, we have two cases:
      (i) $Q \xRightarrow{a(b)} Q'$ and $P' \approx_{\mathrm{lut}} Q'$ and the process $Q \mid \overline{a}b$ can match these actions in the same way.
      (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 $(P', Q' \mid \overline{a}b) \in \mathscr{S}$. □

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

Lemma D.3. Let P and Q be two Lπ-processes and p a name such that $p \notin \mathrm{fn}(P,Q)$. Then:

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Proof.

1 It suffices to show that the relation

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

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 $p \triangleright b \mid P \xrightarrow{p(w)} p \triangleright b \mid \overline{p}w \mid P$, with w fresh, and $p \triangleright b \mid Q \xrightarrow{p(w)} p \triangleright b \mid \overline{p}w \mid Q$. By Lemma D.2, it follows that $((p \triangleright b \mid \overline{p}w \mid P), (p \triangleright b \mid \overline{p}w \mid Q)) \in \mathscr{S}$.

2 We prove that the relation

$$\mathscr{S} = \{((\nu c)(p \triangleright c \mid P)\,,\, (\nu c)(p \triangleright c \mid Q)) : P \approx_{\mathrm{lut}} Q,\ p \notin \mathrm{fn}(P \mid Q)\}$$

is a link bisimulation up to link up to $\equiv$. We use the abbreviations $A = (\nu c)(p \triangleright c \mid P)$ and $B = (\nu c)(p \triangleright c \mid Q)$. 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 $A_1 = (\nu c)(p \triangleright c \mid P')$. Since $P \approx_{\mathrm{lut}} Q$, there exists Q' such that $Q \Longrightarrow Q'$ and $P' \approx_{\mathrm{l}} Q'$. This means that there exists B_1 such that $B \Longrightarrow B_1$, $B_1 = (\nu c)(p \triangleright c \mid Q')$ and $(A_1, B_1) \in \mathscr{S}$.

(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 $A_1 = (\nu c)(p \triangleright c \mid P')$. Since $P \approx_{\mathrm{lut}} Q$, there are two possibilities:

(A) $Q \stackrel{a(b)}{\Longrightarrow} Q'$ and $P' \approx_{\mathrm{lut}} Q'$. In this case $B \stackrel{a(b)}{\Longrightarrow} B_1$ where we have $B_1 = (\nu c)(p \triangleright c \mid Q')$ and $(A_1, B_1) \in \mathscr{S}$.

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

(ii) $a = p$. Since $p \notin \mathrm{fn}(P \mid Q)$, we have $A_1 = (\nu c)(p \triangleright c \mid \overline{c}b \mid P)$. So $B \stackrel{a(b)}{\Longrightarrow} B_1 = (\nu c)(p \triangleright c \mid \overline{c}b \mid Q)$. Since $P \approx_{\mathrm{lut}} Q$ and $b \notin \mathrm{fn}(P,Q)$, by Lemma D.2, we have $(P \mid \overline{c}b) \approx_{\mathrm{lut}} (Q \mid \overline{c}b)$. It follows that $A_1 \equiv (\nu c)(p \triangleright c \mid P \mid \overline{c}b)\ \mathscr{S}\ (\nu c)(p \triangleright c \mid Q \mid \overline{c}b) \equiv B_1$.

(c) If $A \xrightarrow{\overline{a}b} A_1$, we have $P \xrightarrow{\overline{a}b} P'$, $\{a,b\} \cap \{c,p\} = \varnothing$ and $A_1 = (\nu c)(p \triangleright c \mid P')$. Since $P \approx_{\mathrm{lut}} Q$, there are three possibilities:

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

(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 more difficult one, that is, $(w \triangleright b \mid P') \approx_{\mathrm{lut}} (\nu d)(w \triangleright d \mid Q')$ for some fresh name w. We have

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

and

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

(iii) $Q \stackrel{\overline{a}h}{\Longrightarrow} Q'$, with $h \neq b$ and $(w \triangleright b \mid P') \mathrel{\mathscr{S}} (w \triangleright h \mid Q')$ 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.

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

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

for p fresh. By Proposition 4.3, we have

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

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

5 As in Part 3, the proof can be derived by Lemma D.3(2) and Proposition 4.3. □

Appendix E. Proofs of Lemma 9.6 and Lemma 9.13

We restate Lemma 9.6.

Lemma E.1 (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 $\{\![P]\!\} \xrightarrow{(\nu c)\overline{a}c} \gtrsim_{\mathrm{a}} (c \to b \mid \{\![P']\!\})$, with $c \notin \mathrm{fn}(P)$.

(c) If $\mu = (\nu b)\overline{a}b$, then $\{\![P]\!\} \xrightarrow{(\nu c)\overline{a}c} \gtrsim_{\mathrm{a}} (\nu b)(c \to b \mid \{\![P']\!\})$, with $c \notin \mathrm{fn}(P)$.

(d) If $\mu = d(b)\overline{a}b$, then $\{\![P]\!\} \xrightarrow{(\nu c)\overline{a}c} \gtrsim_{\mathrm{a}} (\nu b)(d(b').\, b \to b' \mid c \to b \mid \{\![P']\!\})$, with $\{b', c\} \cap \mathrm{fn}(P) = \varnothing$.

(e) If $\mu = (\nu d)d(b)\overline{a}b$, then $\{\![P]\!\} \xrightarrow{(\nu c)\overline{a}c} \gtrsim_{\mathrm{a}} (\nu b)(\nu d)(d(b').\, b \to b' \mid c \to b \mid \{\![P']\!\})$, with $\{b', c\} \cap \mathrm{fn}(P) = \varnothing$.

(f) If $\mu = \tau$, then $\{\![P]\!\} \xrightarrow{\tau} \gtrsim_{\mathrm{a}} \{\![P']\!\}$.

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 $P_1 \gtrsim_{\mathrm{a}} \{\![P']\!\}\{b'/b\}$.

(b) If $\mu = (\nu c)\overline{a}c$, then we have one of:

(i) $P \xrightarrow{\overline{a}b} P'$ and $P_1 \gtrsim_{\mathrm{a}} (c \to b \mid \{\![P']\!\})$, with $c \notin \mathrm{fn}(P)$.

(ii) $P \xrightarrow{(\nu b)\overline{a}b} P'$ and $P_1 \gtrsim_{\mathrm{a}} (\nu b)(c \to b \mid \{\![P']\!\})$, with $c \notin \mathrm{fn}(P)$.

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

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

(c) If $\mu = \tau$, then $P \xrightarrow{\tau} P'$ with $P_1 \gtrsim_{\mathrm{a}} \{\![P']\!\}$.

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Proof. The proof is by transition induction. We will only 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 $\gtrsim_{\mathrm{a}}$, we have

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

and

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

2 $\mu = \overline{a}b$. This case is straightforward.

3 $\mu = (\boldsymbol{\nu} b)\overline{a}b$. This case is 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-`$\boldsymbol{\nu}$. 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 in 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 the induction hypothesis, we have:

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

— $\{\!|\, Q \,|\!\} \xrightarrow{a(c)} \gtrsim_{\mathrm{a}} \{\!|\, Q' \,|\!\}\{c/x\}$

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

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

(b) Suppose `s-com` is the last rule applied in 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 the induction hypothesis,

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

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

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

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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

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

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, we have

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

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

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

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

— $\{\!|\, P \,|\!\} \xrightarrow{(\boldsymbol{\nu} c)\overline{a}c} \gtrsim_{\mathrm{a}} (\boldsymbol{\nu} b)(d(b').\, b \to b' \mid c \to b \mid \{\!|\, P' \,|\!\})$, with $\{b', c\} \cap fn(P) = \varnothing$.

— $\{\!|\, Q \,|\!\} \xrightarrow{a(c)} \gtrsim_{\mathrm{a}} \{\!|\, Q' \,|\!\}\{c/b'\}$.

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

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

(d) Suppose `s-cls` is the last rule applied in 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 the induction hypothesis,

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

with $\{c', h\} \cap fn(P) = \varnothing$.
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

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

□

Finally, we prove Lemma 9.13.

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

Proof. We prove that the relation

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

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 $P' \gtrsim P' \mathscr{S} [\![P']\!] \lesssim P_1$.

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,

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

for p fresh. By Proposition 4.3(3),

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

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

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

By Proposition 4.3(3),

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

We conclude by observing that

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

□

Acknowledgements

We thank the anonymous referees for their constructive remarks.

References

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

Amadio, R. (1997) An asynchronous model of locality, failure, and process mobility. In: Proc. Coordination'97. *Springer-Verlag Lecture Notes in Computer Science* **1282** 374–391.

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

Appel, A. (1992) *Compiling with Continuations*, Cambridge University Press.

Arun-Kumar, S. and Hennessy, M. (1992) An efficiency preorder for processes. *Acta Informatica* **29** 737–760.

https://doi.org/10.1017/S0960129504004323 Published online by Cambridge University Press

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

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

Boreale, M. and Sangiorgi, D. (1998) Bisimulation in name-passing calculi without matching. In: *Proc. 13th LICS Conf.*, IEEE Computer Society Press 165–175.

Boudol, G. (1992) Asynchrony and the π-calculus. Technical Report RR-1702, INRIA-Sophia Antipolis.

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

van Breugel, F. (1997) A Labelled Transition System for the $\pi\epsilon$-calculus. In: Proc. TAPSOFT'97. *Springer-Verlag Lecture Notes in Computer Science* **1214** 321–332.

Busi, N., Gorrieri, R. and Zavattaro, G. (1998) A process algebraic view of linda coordination primitives. *Theoretical Computer Science* **192** (2) 167–199.

Cardelli, L. (1995) A language with distributed scope. *Computing Systems* **8** (1) 27–59. (An extended abstract appeared in: *Conference Record of POPL '95*.)

Castellani, I. and Hennessy, M. (1998) Testing theories for asynchronous languages. In: Proc. FST-TCS'98. *Springer-Verlag Lecture Notes in Computer Science* **1530** 90–101.

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

Fournet, C. (1999) *The Join calculus: a Calculus for Distributed Mobile Programming*, Ph.D. thesis, École Polytechnique.

Fournet, C. and Gonthier, G. (1996) The Reflexive Chemical Abstract Machine and the Join calculus. In: *Conference Record of 23th POPL*, ACM Press 372–385.

Fournet, C. and Gonthier, G. (1998) A Hierarchy of Equivalences for Asynchronous Calculi. In: Proc. 25th ICALP. *Springer-Verlag Lecture Notes in Computer Science* **1443** 844–855.

Fournet, C. and Laneve, C. (2001) Bisimulations for the Join Calculus. *Theoretical Computer Science* **266** 569–603.

Fournet, C., Laneve, C., Maranget, L. and Rémy, D. (1997) Implict typing à la ML for the join-calculus. In: Proc. CONCUR 97. *Springer-Verlag Lecture Notes in Computer Science* **1243** 196–212.

Fu, Y. (1997) A proof theoretical approach to communication. In: 24th ICALP. *Springer-Verlag Lecture Notes in Computer Science* **1256** 325–335.

Hennessy, M. and Riely, J. (1998) A typed language for distributed mobile processes. In: *Conference Record of 25th POPL*, ACM Press 378–390.

Honda, K. (1992) Two bisimilarities for the ν-calculus. Technical Report 92-002, Keio University.

Honda, K. and Tokoro, M. (1991a) An Object Calculus for Asynchronous Communications. In: Proc. ECOOP '91. *Springer-Verlag Lecture Notes in Computer Science* **512** 133–147.

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

Honda, K. and Yoshida, N. (1994) Replication in Concurrent Combinators. In: Proc. TACS'94. *Springer-Verlag Lecture Notes in Computer Science* **789** 786–805.

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

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

Kleist, J. and Sangiorgi, D. (1998) Imperative objects and mobile processes. In: *Proc. PROCOMET'98*, North-Holland 285–303.

Kobayashi, N., Pierce, B. C. and Turner, D. N. (1999) Linearity and the pi-calculus. *ACM Transactions on Programming Languages and Systems* **21** (5) 914–947. (An extended abstract appeared in: *Conference Record of POPL'96*.)

Laneve, C. and Victor, B. (1999) Solos in Concert. In: Proc. 26th ICALP. *Springer-Verlag Lecture Notes in Computer Science* **1644** 513–523.

Lévy, J.-J. (1997) Some results on the join-calculus. In: Proc. TACS '97. *Springer-Verlag Lecture Notes in Computer Science* **1281**.

Merro, M. (2000) Locality and polyadicity in asynchronous name-passing calculi. In: Proc. FOSSACS2000. *Springer-Verlag Lecture Notes in Computer Science* **1784** 239–251.

Merro, M., Kleist, J. and Nestmann, U. (2002) Mobile Objects as Mobile Processes. Journal of Information and Computation **177** (2) 195–241. (An extended abstract entitled 'Local π-calculus at Work: Mobile Objects as Mobile Processes' appeared in the Proceedings of TCS'00. *Springer-Verlag Lecture Notes in Computer Science* **1872**.)

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

Milner, R. (1992a) Action structure for the π-calculus. Technical Report ECS-LFCS-93-264, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University.

Milner, R. (1992b) Functions as processes. *Mathematical Structures in Computer Science* **2** (2) 119–141.

Milner, R. (1993) Action calculi, or syntactic action structures. In: Proc MFCS'93. *Springer-Verlag Lecture Notes in Computer Science* **711** 105–121.

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

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

Milner, R. and Sangiorgi, D. (1992) Barbed bisimulation. In: Proc. ICALP'92. *Springer-Verlag Lecture Notes in Computer Science* **623** 685–695.

Nestmann, U. and Pierce, B. (2000) Decoding choice encodings. *Journal of Information & Computation* **163** 1–59.

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

Parrow, J. and Victor, B. (1997) The update calculus. In: Proc. AMAST '97. *Springer-Verlag Lecture Notes in Computer Science* **1349** 409–423.

Parrow, J. and Victor, B. (1998) The fusion calculus: Expressiveness and symmetry in mobile processes. In: *Proc. LICS'98*, IEEE Computer Society Press 176–185.

Philippou, A. and Walker, D. (1996) On transformations of concurrent object programs. In: Proc. CONCUR '96. *Springer-Verlag Lecture Notes in Computer Science* **1119** 131–146.

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

Pierce, B. C. and Turner, D. N. (1997) Pict: A programming language based on the pi-calculus. Technical Report CSCI 476, Indiana University. (Also in: Plotkin, G., Stirling, C. and Tofte, M. (eds.) *Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press, 2000.)

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

Sangiorgi, D. (1992) *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*, Ph.D. thesis, CST-99-93, Department of Computer Science, University of Edinburgh.

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

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

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

Sangiorgi, D. (1998) An interpretation of typed objects into typed π-calculus. *Information and Computation* **143** (1) 34–73.

Sangiorgi, D. (1999a) The name discipline of uniform receptiveness. *Theoretical Computer Science* **221** 457–493.

Sangiorgi, D. (1999b) 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.

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

Sangiorgi, D. (2001) Asynchronous process calculi: the first-order and higher-order paradigms (tutorial). *Theoretical Computer Science* **253** (2) 311–350.

Sangiorgi, D. (2002) Types, or: Where's the difference between ccs and π? In: Proc. CONCUR '02. *Springer-Verlag Lecture Notes in Computer Science* **2421** 76–97.

Sangiorgi, D. and Milner, R. (1992) The problem of "Weak Bisimulation up to". In: Proc. CONCUR '92. *Springer-Verlag Lecture Notes in Computer Science* **630** 32–46.

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

Thielecke, H. (1997) *Categorical Structure of Continuation Passing Style*, Ph.D. thesis, University of Edinburgh. (Also available as technical report ECS-LFCS-97-376.)

Vasconcelos, V. T. (1994) *A process-calculus approach to typed concurrent objects*, Ph.D. thesis, Keio University.

Walker, D. (1991) π-calculus semantics of object-oriented programming languages. In: Proc. TACAS'91. *Springer-Verlag Lecture Notes in Computer Science* **526** 523–547.

Yoshida, N. (1996) Graph types for monadic mobile processes. In: Proc. FST & TCS. *Springer-Verlag Lecture Notes in Computer Science* **1180** 371–386.

Yoshida, N. (1998) Minimality and Separation Results on Asynchronous Mobile Processes: representability theorem by concurrent combinators. In: Proc. CONCUR'98. *Springer-Verlag Lecture Notes in Computer Science* **1466** 131–146.