



HAL
open science

On the Representation of References in the pi-calculus

Daniel Hirschhoff, Enguerrand Prebet, Davide Sangiorgi

► **To cite this version:**

Daniel Hirschhoff, Enguerrand Prebet, Davide Sangiorgi. On the Representation of References in the pi-calculus. 2020. hal-02895654v2

HAL Id: hal-02895654

<https://hal.science/hal-02895654v2>

Preprint submitted on 10 Jul 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On the Representation of References in the π -calculus

Daniel Hirschhoff

ENS de Lyon, France

Enguerrand Prebet

ENS de Lyon, France

Davide Sangiorgi

Università di Bologna, Italy

INRIA, France

Abstract

The π -calculus has been advocated as a model to interpret, and give semantics to, languages with higher-order features. Often these languages make use of forms of references (and hence viewing a store as set of references). While translations of references in π -calculi (and CCS) have appeared, the precision of such translations has not been fully investigated. In this paper we address this issue.

We focus on the asynchronous π -calculus ($A\pi$), where translations of references are simpler. We first define π^{ref} , an extension of $A\pi$ with references and operators to manipulate them, and illustrate examples of the subtleties of behavioural equivalence in π^{ref} . We then consider a translation of π^{ref} into $A\pi$. References of π^{ref} are mapped onto names of $A\pi$ belonging to a dedicated "reference" type. We show how the presence of reference names affects the definition of barbed congruence. We establish full abstraction of the translation w.r.t. barbed congruence and barbed equivalence in the two calculi. We investigate proof techniques for barbed equivalence in $A\pi$, based on two forms of labelled bisimilarities. For one bisimilarity we derive both soundness and completeness; for another, more efficient and involving an inductive 'game' on reference names, we derive soundness, leaving completeness open. Finally, we discuss examples of uses of the bisimilarities.

2012 ACM Subject Classification Theory of computation \rightarrow Semantics and reasoning

Keywords and phrases Process calculus, Bisimulation, Asynchrony, Imperative programming

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.31

Funding Hirschhoff and Prebet acknowledge support from the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157), and from LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" ANR-11-IDEX-0007. Sangiorgi acknowledges support from the MIUR-PRIN project 'Analysis of Program Analyses' (ASPR, ID: 201784YSZ5_004), and from the European Research Council (ERC) Grant DLV-818616 DIAPASoN.

1 Introduction

The π -calculus has been advocated as a model to interpret, and give semantics to, languages with higher-order features. Often these languages make use of forms of references (and hence viewing a store as set of references). This therefore requires representations of references using the names of the π -calculus. There are strong similarities between the names of the π -calculus and the references of imperative languages. This is evident in the denotational semantics of these languages: the mathematical techniques employed in modelling the π -calculus (e.g., [24, 6]) were originally developed for the semantic description of references. Yet names and references behave rather differently: receiving from a name is destructive —it consumes a value —whereas reading from a reference is not; a reference has a unique location, whereas a name may be used by several processes both in input and in output; etc. These differences



© Daniel Hirschhoff, Enguerrand Prebet and Davide Sangiorgi;
licensed under Creative Commons License CC-BY

31st International Conference on Concurrency Theory (CONCUR 2020).

Editors: Igor Konnov and Laura Kovács; Article No. 31; pp. 31:1–31:26

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

31:2 On the Representation of References in the pi-calculus

45 make it unclear if and how interesting properties of imperative languages can be proved via
 46 a translation into the π -calculus.

47 A subset of the π -calculus that often appears in the literature, for its expressive power
 48 and elegant theory, is the Asynchronous π -calculus ($A\pi$). $A\pi$ allows one to provide a simpler
 49 representation of references, where a reference ℓ storing a value n is just an output message
 50 $\bar{\ell}\langle n \rangle$ (in $A\pi$ output is not a prefix, hence it has no process continuation). A process that
 51 wishes to access the reference is supposed to make an input at ℓ and then immediately emit
 52 a message at ℓ with the new content of the reference. For instance a process reading on the
 53 reference and binding its content to x in the continuation P is

$$\ell(x).(\bar{\ell}\langle x \rangle \mid P) .$$

54 Another reason that makes this representation of references in $A\pi$ interesting is the bisimilarity
 55 of $A\pi$, called *asynchronous bisimilarity*. It differs from standard bisimilarity in the input
 56 clause, in which a transition $P \xrightarrow{n\langle m \rangle} P'$ (where P is receiving m on n) can be answered by
 57 a bisimilar process Q thus:

$$\bar{n}\langle m \rangle \mid Q \Rightarrow Q' \quad (*)$$

58 (provided P' and Q' are bisimilar), where \Rightarrow stands for zero or several internal communication
 59 steps. Intuitively, Q does not necessarily perform an input on n in response to the transition
 60 done by P . To see why this clause could be interesting with references, consider a process
 61 that performs a *useless read* on a reference ℓ and then continues as P_2 ; in a language with
 62 references this would be equivalent to P_2 itself. When written in $A\pi$, the process with the
 63 useless read becomes $P_1 \stackrel{\text{def}}{=} \ell(x).(\bar{\ell}\langle x \rangle \mid P_2)$ where x does not appear in P_2 . In ordinary
 64 bisimilarity, P_1 is immediately distinguished from P_2 , as the latter cannot answer the input
 65 transition $P_1 \xrightarrow{\ell\langle n \rangle} \bar{\ell}\langle n \rangle \mid P_2$. However, the answer is possible using the clause (*), as we have

$$\bar{\ell}\langle n \rangle \mid P_2 \Rightarrow \bar{\ell}\langle n \rangle \mid P_2 .$$

67 We are not aware of studies that investigate the faithfulness of the above representation
 68 of references in $A\pi$. In this paper we address this issue. For this, we first define π^{ref} , an
 69 extension of $A\pi$ with references and operators to manipulate them. We then consider a
 70 translation of π^{ref} into $A\pi$ and:

71 we study the properties of this translation;

72 we establish proof techniques on $A\pi$ to reason about references.

73 The calculus with references, π^{ref} , has constructs for reading from a reference, writing
 74 on a reference, and a swap operation for atomically reading on a reference and placing a
 75 new value onto it. Modern computer architectures offer hardware instructions similar to
 76 swap, e.g., test-and-set, or control-and-swap constructs to atomically check and modify the
 77 content of a register. These constructs are important to tame the access to shared resources.
 78 In distributed systems, swap can be used to solve the consensus problem with two parallel
 79 processes, whereas simple registers cannot [8].

80 The swap construct is also suggested by the translation of references into $A\pi$. The pattern
 81 for accessing a reference ℓ is $\ell(x).(\bar{\ell}\langle n \rangle \mid P)$. This yields four cases, depending on whether x
 82 is used in P

83 and whether x is equal to n :

	$n \neq x$	$n = x$
x free in P	swap	read
x not free in P	write	useless read

84

85 We define a type system in $A\pi$ to capture the intended pattern of usage of names that
 86 represent references, called *reference names*, in particular the property that there is always
 87 a unique output message available at these names. The type system has linearity features
 88 similar to π -calculus type systems for locks [12] or for receptiveness [21].

89 Imposing a type system has consequences on behavioural equivalences. Since the set
 90 of legal contexts becomes smaller, the behavioural equivalence itself becomes coarser. For
 91 instance, in the case of reference names, a process P is supposed to be tested only in a
 92 context that guarantees that all references mentioned in P are ‘allocated’ (thus, an input
 93 at a reference name ℓ is never ‘stuck’, as an output message at ℓ must always exist). A
 94 consequence of these is a read in which the value read is not used is irrelevant (see formally
 95 law (1)).

96 In both calculi, as behavioural equivalence we use *barbed congruence* and *barbed equivalence*.
 97 These equivalences equate processes which, roughly, in all contexts give rise to ‘matching
 98 reductions’.

99 We establish an operational correspondence between the behaviour of a process in π^{ref} and
 100 its encoding in $A\pi$, and from this we establish full abstraction of the translation of π^{ref}
 101 into $A\pi$ with respect to both barbed equivalence and barbed congruence in the two calculi.
 102 We then investigate proof techniques for barbed equivalence in $A\pi$, based on two forms of
 103 labelled bisimilarities. For one bisimilarity we derive both soundness and completeness. This
 104 bisimilarity is similar to, but not the same as, asynchronous bisimilarity. For instance, it
 105 is defined on ‘reference-closed’ processes (intuitively, processes in which all references are
 106 allocated); therefore inputs on reference names from the tested processes are not visible
 107 (because such inputs are supposed to consume the unique output message at that reference
 108 that is present in the tested processes). The output clause of bisimilarity on reference names
 109 is also different, as we have to make sure that the observer respects the pattern of usage for
 110 reference names; thus the observer consuming the output message on a reference name ℓ
 111 should immediately re-install an output on ℓ .

112 The second bisimilarity is more efficient because it does not require processes to be
 113 ‘reference-closed’. Thus output messages on reference names consumed by the observer need
 114 not be immediately re-installed. However sometimes access to a certain reference is needed
 115 by a process in order to answer the bisimulation challenge from the other process. And
 116 depending on the content of such references, further accesses to other references may be
 117 needed. Since we wish to add only the needed references, this introduces an inductive game, in
 118 which a player requires a reference and the other player specifies the content of such reference,
 119 within the coinductive game of bisimulation. We show that the resulting bisimilarity is sound,
 120 and leave completeness as an open problem. Finally, we discuss examples of uses of the
 121 bisimilarities.

122 **Related Work.** The classic encoding of references in the π -calculus [15] follows their encoding
 123 into CCS [14]: a reference is a stateful recursive process, which may be interrogated using two
 124 names, one for read operations, the other for write operations. Properties of this encoding
 125 have been explored [19], comparing the π -calculus to *Concurrent Idealised Algol* [3], an
 126 extension of Idealised Algol [18] with shared variables concurrency. The encoding has been
 127 shown to be sound but not complete.

128 Many works have studied the effect of type systems on behavioural equivalence, formalised
 129 using both barbed congruence and labelled bisimilarity. (See the references in the books [23,
 130 7]). To our knowledge, no such study has been done regarding the discipline for reference
 131 names which we use in this work. This discipline bears similarities with receptiveness [21],

31:4 On the Representation of References in the pi-calculus

132 which is also related to the results in [22, 13]. We can also remark that our notion of complete
 133 processes is reminiscent of the notion of catalysers used by Dezani et al. [5] in session types
 134 to enforce progress.

135 Section 5 discusses further related work.

136 **Paper outline.** In Section 2, we introduce π^{ref} and discuss examples of behavioural equi-
 137 valences between π^{ref} processes. In Section 3 we present $A\pi$ with reference names, using a
 138 type system that captures the usage of such names. We show the encoding of π^{ref} into such
 139 $A\pi$ and prove its full abstraction for barbed equivalence and congruence. In Section 4 we
 140 introduce the two new labelled bisimilarities for $A\pi$, we establish soundness and completeness
 141 for one and soundness for the other (we conjecture that also completeness holds), and present
 142 a useful ‘up-to’ technique for the second one. Finally we illustrate the benefits of using the
 143 proof techniques based on the labelled bisimilarities of $A\pi$ on some examples.

2 Asynchronous Processes Accessing References: π^{ref}

145 In this section, we introduce π^{ref} , the asynchronous π -calculus extended with primitives to
 146 interact with memory locations.

2.1 Syntax and Semantics

148 We assume an infinite set **Names** of *names* and a distinct infinite set **Refs** of *references*.
 149 These sets do not contain the special symbol \star , that stands for the constant “unit”. We use
 150 $a, b, c, \dots, p, q, \dots$ to range over **Names**; ℓ, \dots to range over **Refs**; and n, m, \dots, x, y, \dots to
 151 range over $\text{All} \stackrel{\text{def}}{=} \text{Names} \cup \text{Refs} \cup \{\star\}$. The grammar for the calculus π^{ref} is the following; for
 152 simplicity, we develop our theory on the monadic calculus (one value at a time is handled).

$$153 \quad P ::= \mathbf{0} \mid a(x).P \mid \bar{a}\langle n \rangle \mid !P \mid P_1 \mid P_2 \mid (\nu a)P \mid [n = m]P \\ 154 \quad \mid (\nu \ell = n)P \mid \ell \triangleleft n. P \mid \ell \triangleright (x). P \mid \ell \bowtie n(x). P$$

156 The operators in the first line are the standard π -calculus constructs for the inactive
 157 process, input, asynchronous output, replication, parallel composition, name restriction, and
 158 matching (however matching here is defined on both names and references). In the second
 159 line, we find the operators to handle references: reference restriction, or allocation (creating
 160 a new reference ℓ with initial value n), write (setting the content of ℓ to n), read (reading in
 161 x the value of ℓ), swap (atomically reading on x and replacing the content of the reference
 162 with n).

163 As usual, we often omit $\mathbf{0}$, and abbreviate $\bar{a}\langle \star \rangle$ as \bar{a} (and similarly for inputs $a.P$). We
 164 use a tilde, $\tilde{\cdot}$, for (possibly empty) finite tuples; then $(\nu \tilde{a})$ is a sequence of restrictions; and
 165 $(\nu \tilde{L})$ a sequence of reference allocations (i.e., a piece of store), using L to represent a single
 166 allocation such as $\ell = n$. Given the binders $(\nu a)P$ and $(\nu \ell = n)P$ (for a and ℓ , respectively),
 167 $a(x).P$, $\ell \triangleright (x).P$ and $\ell \bowtie n(x)$ (for x), we define $\text{bn}(O)$, $\text{fn}(O)$ (resp. $\text{fr}(O)$, $\text{br}(O)$), for the
 168 *bound* and *free names* (resp. *references*) of some object O (process, action, etc.). The set
 169 of *names* of O is defined as the union of its free and bound names; and analogously for
 170 *references*. In $a(x).P$ or $\bar{a}\langle x \rangle$, name a is the *subject* whereas x is the *object*.

171 We assume the calculus is simply-typed. Any basic type system for the π -calculus would
 172 do. In this paper, we assume Milner’s *sorting*: names and references are partitioned into
 173 a collection of *types* (or *sorts*). Name types contain names, and reference types contain
 174 references. Then a sorting function maps types onto types. If a name type s is mapped

$$\begin{array}{c}
\text{R-Equiv: } \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q} \qquad \text{R-Ctxt: } \frac{P \longrightarrow P'}{E[P] \longrightarrow E[Q]} \\
\\
\text{R-Comm: } \frac{}{a(x).P \mid \bar{a}\langle n \rangle \longrightarrow P\{n/x\}} \\
\\
\text{R-Read: } \frac{\ell, n \notin \text{br}(\nu\tilde{L})}{(\nu\ell = n)(\nu\tilde{L})(\ell \triangleright (x).P \mid Q) \longrightarrow (\nu\ell = n)(\nu\tilde{L})(P\{n/x\} \mid Q)} \\
\\
\text{R-Write: } \frac{\ell, n \notin \text{br}(\nu\tilde{L})}{(\nu\ell = m)(\nu\tilde{L})(\ell \triangleleft n.P \mid Q) \longrightarrow (\nu\ell = n)(\nu\tilde{L})(P \mid Q)} \\
\\
\text{R-Swap: } \frac{\ell, n, m \notin \text{br}(\nu\tilde{L})}{(\nu\ell = m)(\nu\tilde{L})(\ell \bowtie n(x).P \mid Q) \longrightarrow (\nu\ell = n)(\nu\tilde{L})(P\{m/x\} \mid Q)}
\end{array}$$

■ **Figure 1** π^{ref} , reduction relation

175 onto a type t , this means that names in s may only carry, or contain, objects in t ; if s is a
 176 reference type then only objects of type t may be stored in s . We shall assume that there is a
 177 sorting system under which all processes we manipulate are well-typed. For simplicity we use
 178 simple types; e.g., the sorting is non-recursive (meaning that the graph that represents the
 179 sorting function, in which the nodes are the types, does not contain cycles). In the remainder
 180 we assume that all objects (processes, contexts, actions, etc.) respect a given sorting.

181 The definition of structural congruence, \equiv , is the expected one from the π -calculus,
 182 treating the $(\nu\ell = n)$ operator like a restriction (see Appendix B.1).

183 *Contexts*, ranged over by C , are process expressions with a hole $[\]$ in it. We write $C[P]$
 184 for the process obtained by replacing the hole in C with P . *Active* (or *evaluation*) *contexts*,
 185 ranged over by E , are given by:

$$E ::= [\] \mid E \mid P \mid (\nu a)E \mid (\nu\ell = n)E .$$

186 The reduction relation \longrightarrow is presented in Figure 1. It uses *active contexts* to isolate the
 187 subpart of the term that is active in a reduction. We write \Longrightarrow for the ‘multistep’ version of
 188 \longrightarrow , whereby $P \Longrightarrow P'$ if P may become P' after a (possibly empty) sequence of reductions.
 189 Rules **R-Read**, **R-Write** and **R-Swap** in Figure 1 describe an interaction between the process
 190 and a reference ℓ . These rules make use of a store $(\nu\tilde{L})$; this is necessary because there
 191 might be references that depend on ℓ , and as such cannot be moved past the restriction
 192 on ℓ . An example is $(\nu\ell = a)(\nu\ell' = \ell)\ell \triangleleft b.P$: the write operation is executed by applying
 193 rule **R-Write**, with $(\nu\tilde{L}) = (\nu\ell' = \ell)$, as the restriction on ℓ' cannot be brought above the
 194 restriction on ℓ . We recall that $\text{br}(\nu\tilde{L})$ are the references bound by the ν .

195 As usual in concurrent calculi, the reference behavioural equivalence will be barbed
 196 congruence (in its variant sometimes called *reduction-closed barbed congruence*), a form of
 197 bisimulation on reduction that uses closure under contexts and simple observables. In the
 198 context closure, however, we make sure that all references mentioned in the tested process
 199 have been allocated. As often in π -calculi, we also consider *barbed equivalence*, that uses only
 200 active contexts.

31:6 On the Representation of References in the pi-calculus

201 P exhibits a barb at a (so a is in `Names`), written $P \Downarrow_{\bar{a}}$, if $P \equiv (\nu \tilde{b})(\nu \tilde{L})(\bar{a}(m) \mid P')$ with
 202 $a \notin \tilde{b}$. We write $P \Downarrow_{\bar{a}}$ if $P \Longrightarrow P_1$ and $P_1 \Downarrow_{\bar{a}}$ for some P_1 .

203 ► **Definition 1.** Given a relation \mathcal{R} on processes, and $P \mathcal{R} Q$, we say that P, Q (in \mathcal{R}) are
 204 – closed under reductions if $P \longrightarrow P'$ implies there is Q' s.t. $Q \Longrightarrow Q'$ and $P' \mathcal{R} Q'$;
 205 – preserved by a set \mathcal{C} of contexts if $C[P] \mathcal{R} C[Q]$ for all $C \in \mathcal{C}$;
 206 – compatible on barbs if $P \Downarrow_{\bar{a}}$ implies $Q \Downarrow_{\bar{a}}$, for all a .

207 A process P is *reference-closed* if $\text{fr}(P) = \emptyset$. A context C is *closing on the references* of
 208 a process P if $C[P]$ is reference-closed; similarly, C is closing on the references of P, Q if it
 209 closing on the references of both P and Q . Since reductions may only decrease the set of
 210 free names of a process, the property of being reference-closed is preserved by reductions.

211 ► **Definition 2** (Barbed congruence and equivalence in π^{ref}). Barbed congruence is the largest
 212 symmetric relation \cong_{ref} in π^{ref} such that whenever $P \mathcal{R} Q$ then P, Q are: closed under
 213 reductions if P, Q are reference-closed; preserved by the contexts that are closing on references
 214 for P, Q ; compatible on barbs if P, Q are reference-closed. Barbed equivalence, \cong_{ref}^c , is
 215 defined in the same way, but using active contexts in place of all contexts.

216 The restriction to closing contexts (as opposed to arbitrary contexts) yields laws such as

$$217 \quad \ell \triangleright (x).P \cong_{\text{ref}} P, \quad (1)$$

218 whenever $x \notin \text{fn}(P)$. Closing contexts ensure that the reading on ℓ is not blocking, and
 219 therefore possible observables in P are visible on both sides.

220 As the quantification on contexts refers to the free references of the tested processes,
 221 transitivity of barbed congruence and equivalence requires some care. As usual in the
 222 π -calculus, barbed equivalence is not preserved by the input construct, and the closure of
 223 barbed equivalence under all (well-typed) substitutions coincides with barbed congruence.

224 2.2 Behavioural Equivalence in π^{ref} : Examples

225 We present a few examples that illustrate some subtleties of behavioural equivalence in
 226 π^{ref} . These examples will be formally treated in Section 4.2 for Examples 3 and 4, and in
 227 Appendix A for Examples 5 and 6.

228 The first example shows that processes may be equivalent even though the store is public
 229 and holds different values. (In the example, the reference ℓ is actually restricted, but the
 230 process P underneath the restriction, representing an observer, is arbitrary).

231 ► **Example 3.** For any P , we have $P_1 \cong_{\text{ref}} P_2$, for

$$P_1 \stackrel{\text{def}}{=} (\nu \ell = a)(P \mid !\ell \triangleleft a \mid !\ell \triangleleft b) \quad P_2 \stackrel{\text{def}}{=} (\nu \ell = b)(P \mid !\ell \triangleleft a \mid !\ell \triangleleft b)$$

232 In the second example, the write on top of P is not blocking, provided that the same writing
 233 is anyhow possible, and provided that the current value of the store can be recorded.

► **Example 4.** We have $P_1 \cong_{\text{ref}} P_2$, for

$$P_1 \stackrel{\text{def}}{=} \ell \triangleleft b. P \mid !\ell \triangleleft b \mid !\ell \triangleright (x). \ell \triangleleft x \quad P_2 \stackrel{\text{def}}{=} P \mid !\ell \triangleleft b \mid !\ell \triangleright (x). \ell \triangleleft x$$

234 On the left, it would seem that P runs under a store in which ℓ contains b ; whereas on the
 235 right, P could also run under the initial store, where ℓ could contain a different value, say a .
 236 However the component $!\ell \triangleright (x). \ell \triangleleft x$ allows us to store a in x and then write it back later,
 237 thus overwriting b .

► **Example 5.** We have $P_s \not\cong_{\text{ref}}^e Q_s$, where

$$P_s \stackrel{\text{def}}{=} (\nu t)\ell \triangleleft b. (\bar{t} \mid !t. \ell \triangleleft a. (\bar{c} \mid \ell \triangleleft b. (\bar{t} \mid c))) \quad Q_s \stackrel{\text{def}}{=} (\nu t)\ell \triangleleft a. (\bar{t} \mid !t. \ell \triangleleft b. (\bar{c} \mid \ell \triangleleft a. (\bar{t} \mid c)))$$

238 The discriminating context being large, the formal discussion is moved in Appendix A.
 239 Intuitively, P_s and Q_s are refinements of the processes in Example 3, in that their initial
 240 writes store different values on the reference ℓ , but both processes maintain the capability
 241 of writing both values in ℓ . The difference with Example 3 are the additional inputs and
 242 outputs on name c , which are generated along the transitions. These allow an observer to
 243 distinguish P_s from Q_s by exploiting the swap construct. We informally explain the reason.
 244 If the two processes have written the same value, say a , in ℓ , then Q_s has generated the
 245 same number of inputs and outputs on c , while P_s must have generated an extra output. An
 246 observer can use swap to read the content of ℓ , so to check that the value is indeed a , and
 247 write back a fresh name, say e . Now the observer can tell that P_s has an extra output on c :
 248 process Q_s cannot add a further output, because this would require overwriting e in ℓ , which
 249 can be tested by the observer at the end.

250 We have seen in Example 3 two equivalent processes whose initial store (a single reference)
 251 is different. The equivalence holds intuitively because the values that the two processes
 252 can store are the same. Using two references, it is possible to complicate the example. In
 253 Example 6, the processes are equivalent and yet the pairs of values that may be simultaneously
 254 stored in the two references are different for the two processes. For each reference separately,
 255 the set of possible values is the same. But setting a reference to a certain value implies first
 256 having set the other reference to some specific values. (The processes could be distinguished
 257 if an observer had the possibility to simultaneously read the two references.)

258 ► **Example 6.** Consider two references ℓ_1, ℓ_2 where booleans (represented as 0,1 below) can
 259 be stored. Then for any P , we have $P_1 \cong_{\text{ref}} P_2$, where

$$260 \quad P_1 \stackrel{\text{def}}{=} (\nu \ell_1 = 0, \ell_2 = 0)(P \mid (\nu t)(\bar{t} \mid !t. \ell_1 \triangleleft 1. \ell_1 \triangleleft 0. \ell_2 \triangleleft 1. \ell_2 \triangleleft 0. \bar{t}))$$

$$261 \quad P_2 \stackrel{\text{def}}{=} (\nu \ell_1 = 0, \ell_2 = 0)(P \mid (\nu t)(\bar{t} \mid !t. \ell_1 \triangleleft 1. \ell_2 \triangleleft 1. \ell_1 \triangleleft 0. \ell_2 \triangleleft 0. \bar{t}))$$

263 P_1 and P_2 can write 0 and 1 in references ℓ_1 and ℓ_2 , but not in the same order. By doing so,
 264 we see that if P_1 loops, the content of ℓ_1 and ℓ_2 will evolve thus: $(0, 0) \rightarrow (1, 0) \rightarrow (0, 0) \rightarrow$
 265 $(0, 1) \rightarrow (0, 0)$, while for P_2 the loop is different: $(0, 0) \rightarrow (1, 0) \rightarrow (1, 1) \rightarrow (0, 1) \rightarrow (0, 0)$.

266 In particular, P_2 can always go through the state $(1, 1)$, independently of the transitions
 267 of P , while P_1 cannot, in general, reach this state.

268 The example above relies on the fact that the domain of possible values for ℓ_1 and ℓ_2 is
 269 finite. A more sophisticated example, without such assumption, is given in the Appendix A.

270 **3 Mapping π^{ref} onto the Asynchronous π -calculus**

271 We present the encoding of π^{ref} into $A\pi$, which follows the folklore encoding of references
 272 into $A\pi$.

273 **3.1 The Asynchronous π -calculus**

274 Below is the grammar of the asynchronous π -calculus, $A\pi$; we reuse all notations from π^{ref} .

$$275 \quad P ::= \mathbf{0} \mid n(x).P \mid !P \mid \bar{n}(m) \mid P_1 \mid P_2 \mid (\nu n)P \mid [n = m]P$$

277 The reduction semantics, as well as barbed equivalence and congruence (written \cong_a^e and
 278 \cong_a , respectively), are standard (defined as in π^{ref} , and recalled in Appendix B.1). We recall
 279 the standard definition of asynchronous bisimilarity, \approx_a , from [1]. To define \approx_a , as well as
 280 the other forms of bisimilarity we introduce in Section 4, we rely on the early transition
 281 system for $A\pi$. In this LTS, which is presented in Appendix B.1 labels are either free inputs
 282 of the form $n\langle m \rangle$ (reception of name m on n), output $(\bar{n}\langle m \rangle)$, bound output $((\nu m)\bar{n}\langle m \rangle)$ or
 283 internal communication (τ) .

284 ► **Definition 7.** *A symmetric relation \mathcal{R} between processes is an asynchronous bisimulation*
 285 *if whenever $P \mathcal{R} Q$ and $P \xrightarrow{\mu} P'$, one of these two clauses hold:*

- 286 – *there is Q' such that $Q \xrightarrow{\hat{\mu}} Q'$ and $P' \mathcal{R} Q'$;*
- 287 – *$\mu = n\langle m \rangle$ and there is Q' such that $Q \mid \bar{n}\langle m \rangle \Rightarrow Q'$ and $P' \mathcal{R} Q'$.*

288 *Asynchronous bisimilarity, \approx_a , is the largest asynchronous bisimulation.*

289 ► **Theorem 8** ([1]). *Relations \cong_a^e and \approx_a coincide.*

290 3.2 Encoding π^{ref}

291 In π -calculi such as $A\pi$, there are no references, only names. To make the encoding easier to
 292 read, we assume however that the set of names contains the set of references $\{\ell, \dots\}$ of π^{ref} .
 293 We call such names *reference names*, and call *plain names* the remaining names. Reference
 294 names will be used to represent the references of π^{ref} .

The encoding $\mathcal{E}[\cdot]$, from π^{ref} to $A\pi$, is a homomorphism on all operators (thus, e.g.,
 $\mathcal{E}[P_1 \mid P_2] \stackrel{\text{def}}{=} \mathcal{E}[P_1] \mid \mathcal{E}[P_2]$, and $\mathcal{E}[a(m).P] \stackrel{\text{def}}{=} a(m).\mathcal{E}[P]$), except for reference constructs
 for which we have:

$$\begin{aligned} \mathcal{E}[(\nu l = m).P] &\stackrel{\text{def}}{=} (\nu l)(\bar{l}\langle m \rangle \mid \mathcal{E}[P]) & \mathcal{E}[\ell \triangleleft v.P] &\stackrel{\text{def}}{=} \ell(_).(\bar{\ell}\langle v \rangle \mid \mathcal{E}[P]) \\ \mathcal{E}[\ell \triangleright (x).P] &\stackrel{\text{def}}{=} \ell(x).(\bar{\ell}\langle x \rangle \mid \mathcal{E}[P]) & \mathcal{E}[\ell \bowtie n(x).P] &\stackrel{\text{def}}{=} \ell(x).(\bar{\ell}\langle n \rangle \mid \mathcal{E}[P]) \end{aligned}$$

295 (We write $\ell(_).Q$ for an input whose bound name does not appear in Q .) In the encoding, an
 296 object m stored at reference ℓ is represented as a message $\bar{\ell}\langle m \rangle$. Accordingly, the encoding of
 297 a write $\ell \triangleleft v.P$ is $\ell(_).(\bar{\ell}\langle v \rangle \mid \mathcal{E}[P])$, meaning that the process acquires the current message
 298 at ℓ (which is thus not available anymore) and replaces it with an output with the new value.
 299 The encoding of a read $\ell \triangleright (x).P$ follows a similar pattern, this time however the same value
 300 is received and emitted: $\ell(x).(\bar{\ell}\langle x \rangle \mid P)$. The encoding of swap combines the two patterns.

301 3.3 Types and Behavioural Equivalences with Reference Names

302 To prove a full abstraction property for the encoding, we use types to formalise the behavioural
 303 difference between reference names and plain names in the asynchronous π -calculus. The
 304 typing discipline can be added onto any basic type system for the π -calculus. As for π^{ref} ,
 305 we follow Milner's sorting. The types of the sorting impose a partition on the two sets of
 306 names (reference names and plain names). Thus we assume such a sorting, under which
 307 all processes are well-typed. We separate the base type system (Milner's sorting) from the
 308 typing rules for reference names so as to show the essence of the latter rules. Accordingly,
 309 we only present the additional typing constraints for reference names.

310 We write: **RefTypes** for the the set of reference types (i.e., types that contain reference
 311 names); **Type**(n) is the type of name n ; **ObjType**(n) is the type of the objects of n (i.e., the
 312 type of the names that may be carried at n). For example in well-typed processes such as
 313 $\bar{n}\langle m \rangle$ and $n(m).P$, name m will be of type **ObjType**(n).

$$\begin{array}{c}
\text{TNil} \frac{}{\emptyset \vdash \mathbf{0}} \quad \text{TOut} \frac{}{\emptyset \vdash \bar{a}\langle m \rangle} \quad \text{TInp} \frac{\emptyset \vdash P}{\emptyset \vdash a(x).P} \quad \text{TRep} \frac{\emptyset \vdash P}{\emptyset \vdash !P} \\
\text{TPar} \frac{\Delta_1 \vdash P \quad \Delta_2 \vdash Q}{\Delta_1 \uplus \Delta_2 \vdash P \mid Q} \quad \text{TResN} \frac{\Delta \vdash P}{\Delta \vdash (\nu a)P} \quad \text{TResR} \frac{\Delta, \ell \vdash P}{\Delta \vdash (\nu \ell)P} \\
\text{TRef0} \frac{}{\ell \vdash \bar{\ell}\langle m \rangle} \quad \text{TRefI} \frac{\ell \vdash P}{\emptyset \vdash \ell(x).P}
\end{array}$$

■ **Figure 2** Typing conditions for reference names in $\mathcal{A}\pi$ processes

314 **Notations.** We use ℓ, \dots to range over reference names, a, b, \dots over plain names, n, m, \dots
315 over the set of all names. Δ ranges over finite sets of reference names. We sometimes write
316 $\Delta - x$ as abbreviation for $\Delta - \{x\}$. Moreover $\Delta_1 \uplus \Delta_2$ is defined only when $\Delta_1 \cap \Delta_2 = \emptyset$, in
317 which case it is $\Delta_1 \cup \Delta_2$; we write Δ, x for $\Delta \uplus \{x\}$.

318 The type system is presented in Figure 2. Judgements have the form $\Delta \vdash P$, where P is
319 an $\mathcal{A}\pi$ process. Rule **TRef0** along with Rule **TPar** ensures that every reference names in Δ
320 appears in subject of exactly one unguarded output. Rule **TResR** ensures that new reference
321 names are always in Δ while Rule **TRefI** ensures that Δ is constant after a communication
322 between references (by re-emitting an output after one has been consumed).

323 Intuitively, if $\Delta \vdash P$, then P must make available the names in Δ *immediately* and *exactly*
324 *once* in output subject position. We say that ℓ is *output receptive* in P if there is exactly
325 one unguarded output at ℓ , and moreover this output is not underneath a replication. Then
326 $\Delta \vdash P$ holds if

- 327 – any $\ell \in \Delta$ is output receptive in P ;
- 328 – in any subterm of P of the form $(\nu \ell')Q$ or $\ell'(m).Q$, name ℓ' is output receptive in Q .

329 This intuition is formalised in Lemma 9, and in Proposition 10 that relates types and
330 operational semantics.

331 Typing is important because it allows us to derive the required behavioural equivalences.
332 For instance, allowing parallel composition with the ill-typed process $\ell(x).\mathbf{0}$ would invalidate
333 barbed equivalence between the (translations of the) terms in law (1).

334 In the remainder of the paper, it is assumed that all processes are *well typed*, meaning
335 that each process P obeys the underlying sorting system and that there is Δ s.t. $\Delta \vdash P$
336 holds. Two processes P, Q are *type-compatible* if both $\Delta \vdash P$ and $\Delta \vdash Q$, for some Δ ; we
337 write $\Delta \vdash P, Q$ in this case. *In the remainder of the paper, all relations are on pairs of*
338 *type-compatible processes. Similarly, all compositions (i.e., of a context with processes) and*
339 *actions are well-typed.*

340 The type system satisfies standard properties, like uniqueness of typing ($\Delta \vdash P$ and
341 $\Delta' \vdash P$ imply $\Delta = \Delta'$), and preservation by structural congruence ($P \equiv Q$ and $\Delta \vdash P$ imply
342 $\Delta \vdash Q$). As claimed above, if $\Delta \vdash P$, then names in Δ are output receptive:

343 ► **Lemma 9.** *If $\Delta, \ell \vdash P$ then $P \equiv (\nu \tilde{n})(\bar{\ell}\langle m \rangle \mid Q)$, with $\ell \notin \tilde{n}$, and there is no unguarded*
344 *output at ℓ in Q .*

345 The following standard property relies on the standard LTS for $\mathcal{A}\pi$, which is given in
346 Appendix B.1.

347 ► **Proposition 10** (Subject reduction). *If $\Delta \vdash P$ and $P \xrightarrow{\mu} P'$, then*

31:10 On the Representation of References in the pi-calculus

- 348 1. if $\mu = \tau$, $\mu = \bar{a}\langle m \rangle$, $\mu = a\langle m \rangle$ or $\mu = (\nu b)\bar{a}\langle b \rangle$, then $\Delta \vdash P'$.
 349 2. if $\mu = (\nu \ell)\bar{a}\langle \ell \rangle$ then $\Delta, \ell \vdash P'$.
 350 3. if $\mu = \ell\langle m \rangle$ and $\ell \notin \Delta$, then $\Delta, \ell \vdash P'$
 351 4. if $\ell \notin \Delta$, then $\Delta, \ell \vdash P \mid \bar{\ell}\langle m \rangle$.
 352 5. if $\mu = \bar{\ell}\langle m \rangle$ or $\mu = (\nu b)\bar{\ell}\langle b \rangle$, then $\Delta - \ell \vdash P'$.
 353 6. if $\mu = (\nu \ell')\bar{\ell}\langle \ell' \rangle$, then $(\Delta - \ell), \ell' \vdash P'$.

354 We can remark that in case 3, we have $\ell \notin \Delta$, as otherwise the context would not be able
 355 to trigger an input (since, by typing, it could not generate an output on ℓ).

356 **Barbed congruence.** As usual in typed calculi, the definitions of the barbed relations take
 357 typing into account, so that the composition of a context and a process be well-typed. In the
 358 case of reference names, an additional ingredient has to be taken into account, namely the
 359 accessibility of reference names. If a process has the possibility of accessing a reference, then
 360 a context in which the process is tested should guarantee the availability of that reference.
 361 For this, we define the notion of *completing context* and *complete* process. Then, roughly,
 362 barbed congruence becomes “barbed congruence under all completing contexts”.

363 A process P is *complete* if each reference name that appears free in P is ‘allocated’ in P .
 364 We write $\text{frn}(P)$ for the set of free reference names in P .

365 ► **Definition 11** (Open references and complete processes). *The open references of P such*
 366 *that $\Delta \vdash P$ are the names in $\text{frn}(P) \setminus \Delta$; similarly the open references of processes P_1, \dots, P_n*
 367 *is the union of the open references of the P_i ’s. P is complete if it contains no open reference.*
 368 *$\text{frn}(P) \subseteq \Delta$ and $\Delta \vdash P$, for some Δ .*

369 A context C is *completing* for P if $C[P]$ is complete.

370 (Note that an $A\pi$ *complete process* might have free reference names, if these are not open
 371 references; in contrast, a π^{ref} *reference-closed process* does not have free references.)

372 ► **Lemma 12.** *P is complete iff $\emptyset \vdash (\nu \tilde{n})P$ where $\tilde{n} \stackrel{\text{def}}{=} \text{frn}(P)$.*

373 Completing contexts are the only contexts in which processes should be tested. We
 374 constrain the definitions of typed barbed congruence and equivalence accordingly. The
 375 grammar for the active contexts in $A\pi$ is as expected:

$$E ::= [] \mid E \mid P \mid (\nu n)E .$$

376 ► **Definition 13** (Barbed congruence and equivalence in $A\pi$ with reference names). *Barbed*
 377 *congruence is the largest symmetric relation \cong_{Arn} in $A\pi$ such that whenever $P \mathcal{R} Q$ then*
 378 *P, Q are: closed under reductions whenever they are complete; closed under the contexts that*
 379 *are completing for P, Q ; compatible on barbs whenever they are complete. Barbed equivalence,*
 380 *\cong_{Arn}^e , is defined analogously except that one uses active contexts in place of all contexts.*

381 This typed barbed equivalence is the behavioural equivalence we are mainly interested in.
 382 The reference name discipline weakens the requirements on names (by limiting the number of
 383 legal contexts), hence the corresponding typed barbed relation is coarser. We are not aware
 384 of existing works in the literature that study the impact of the reference name discipline on
 385 behavioural equivalence.

386 ► **Lemma 14.** *For all compatible P, Q , $P \cong_a^e Q$ (and hence also $P \approx_a Q$) implies $P \cong_{\text{Arn}}^e Q$.*

387 We show in Section 4 that the inclusion is strict.

3.4 Validating the Encoding

We now show that the two notions of barbed congruence coincide via the encoding.

► **Theorem 15** (Operational correspondence). *If $P \longrightarrow P'$, then $\mathcal{E}[[P]] \longrightarrow \mathcal{E}[[P']]$.
Conversely, if $\mathcal{E}[[P]] \longrightarrow Q$, then $P \longrightarrow P'$, with $\mathcal{E}[[P']] \equiv Q$.*

The next lemma shows that, up to asynchronous bisimilarity, we can ‘read back’ well-typed processes in $A\pi$, via the encoding, as processes in π^{ref} . And similarly for contexts.

► **Lemma 16.** *If $\emptyset \vdash P$, then there exists R in π^{ref} such that $\mathcal{E}[[R]] \approx_a P$.*

Theorem 15 and Lemma 16 are the main ingredients to derive the following theorem:

► **Theorem 17** (Full abstraction). *For any P, Q in π^{ref} : $P \cong_{\text{ref}} Q$ iff $\mathcal{E}[[P]] \cong_{\text{Arn}} \mathcal{E}[[Q]]$;
and similarly $P \cong_{\text{ref}}^c Q$ iff $\mathcal{E}[[P]] \cong_{\text{Arn}}^c \mathcal{E}[[Q]]$.*

4 Bisimulation with Reference Names

4.1 Two Labelled Bisimilarities

In this section we present proof techniques for barbed equivalence based on the labelled transition semantics of $A\pi$. For this we introduce two labelled bisimilarities.

The first form of bisimulation, *reference bisimilarity*, only relates complete processes; processes that are not complete have to be made so. Intuitively, in this bisimilarity processes are made complete by requiring a closure of the relation with respect to the (well-typed) addition of output messages at reference names (the ‘closure under allocation’ below). Moreover, when an observer consumes an output at a reference name, say $\bar{\ell}\langle n \rangle$, then, following the discipline on reference names, he/she has to immediately provide another such output message, say $\bar{\ell}\langle m \rangle$. This is formalised using transition notations such as $P \xrightarrow{\bar{\ell}\langle n \rangle[m]} P'$, which makes a swap on ℓ (reading its original content n and replacing it with m). As a consequence of the appearance of such swap transitions, ordinary outputs at reference names are not observed in the bisimulation. Similarly for inputs at reference names: an input $P \xrightarrow{\ell\langle m \rangle} P'$ from a complete process P is not observed, since it is supposed to interact with unique output at ℓ contained in P (which exists as P is complete). Finally, an observer should respect the completeness condition by the processes and should not communicate a fresh reference name — to communicate such a reference, say ℓ , an allocation for ℓ (an output message at ℓ) has first to be added.

A relation \mathcal{R} is *closed under allocation* if $P \mathcal{R} Q$ implies $P \mid \bar{\ell}\langle n \rangle \mathcal{R} Q \mid \bar{\ell}\langle n \rangle$ for any $\bar{\ell}\langle n \rangle$ such that $P \mid \bar{\ell}\langle n \rangle$ and $Q \mid \bar{\ell}\langle n \rangle$ are well-typed. We write $P \xrightarrow{\bar{\ell}\langle n \rangle[m]} P'$ if $P \xrightarrow{\bar{\ell}\langle n \rangle} P''$ and $P' = \bar{\ell}\langle m \rangle \mid P''$, for some P'' ; similarly for $P \xrightarrow{(\nu n)\bar{\ell}\langle n \rangle[m]} P'$. Then, as usual, $P \xrightarrow{\bar{\ell}\langle n \rangle[m]} P'$ holds if $P \Rightarrow P'' \xrightarrow{\bar{\ell}\langle n \rangle[m]} P''' \Rightarrow P'$ for some P'', P''' , and similarly for $P \xrightarrow{(\nu n)\bar{\ell}\langle n \rangle[m]} P'$.

We let α range over the actions μ plus the aforementioned ‘update actions’ $\bar{\ell}\langle n \rangle[m]$ and $(\nu n)\bar{\ell}\langle n \rangle[m]$.

Setting m to be the object of an update actions, we write $\Delta \vdash \alpha$ when: (i) if the object of α is a free reference name then it is in Δ , and (ii) α is not an input or an output at a reference name.

► **Definition 18** (Reference bisimilarity). *A symmetric relation \mathcal{R} closed under allocation is a reference bisimulation if whenever $P \mathcal{R} Q$ with P, Q complete, $\Delta \vdash P, Q$ and $P \xrightarrow{\alpha} P'$ with $\Delta \vdash \alpha$, then*

31:12 On the Representation of References in the pi-calculus

- 429 1. either there exists Q' such that $Q \xrightarrow{\hat{\alpha}} Q'$ and $P' \mathcal{R} Q'$ for some Q'
 430 2. or α is an input $a\langle m \rangle$ and $Q \mid \bar{a}\langle m \rangle \Rightarrow Q'$ with $P' \mathcal{R} Q'$ for some Q' .
 431 Reference bisimilarity, written \approx , is the largest reference bisimulation.

432 We now show that \approx coincides with barbed equivalence. The structure of the proof is
 433 standard, however some care has to be taken to deal with closure under parallel composition.

434 ► **Lemma 19.** *If $P \approx Q$, and $\emptyset \vdash R$, then $P \mid R \approx Q \mid R$.*

435 ► **Proposition 20** (Substitutivity for active contexts). *If $P \approx Q$, then $E[P] \approx E[Q]$ for any*
 436 *active context E .*

437 ► **Theorem 21** (Labelled characterisation). *$P \approx Q$ iff $P \cong_{\text{Arn}}^e Q$.*

438 In reference bisimilarity, the tested processes are complete: hence all their references
 439 must explicitly appear as allocated, and when a reference is accessed, an extension of the
 440 store is made so to remain with complete processes (and if such an extension introduces
 441 other new references, a further extension is needed). The goal of the bisimilarity \approx_{ip} below
 442 is to allow one to work on processes with open references, and make the extension of the
 443 store only when necessary. The definition of the bisimulation exploits an inductive predicate
 444 to accommodate finite extensions of the store, one step at a time. This predicate can be
 445 thought of as an inductive game, in which the ‘verifier’ can choose rule **Base** and close the
 446 game, or choose rule **Ext** and a reference ℓ ; in the latter case the ‘refuter’ chooses the value
 447 stored in ℓ .

448 ► **Definition 22** (Inductive predicate). *The predicate $\text{ok}(\Delta, \mathcal{R}, P, Q, \mu)$ (where Δ is a set*
 449 *of names, \mathcal{R} a process relation, P, Q processes, and μ an action) holds if it can be proved*
 450 *inductively from the following two rules:*

$$\text{Base} \frac{\begin{cases} Q \mid \bar{n}\langle m \rangle \Rightarrow Q' & \text{for } \mu = n\langle m \rangle \\ Q \xrightarrow{\mu} Q' & \text{otherwise} \end{cases} \quad P' \mathcal{R} Q'}{\text{ok}(\Delta, \mathcal{R}, P', Q, \mu)}$$

$$\text{Ext} \frac{\ell \notin \Delta \quad \forall m : \text{ok}((\Delta, \ell), \mathcal{R}, P' \mid \bar{\ell}\langle m \rangle, Q \mid \bar{\ell}\langle m \rangle, \mu)}{\text{ok}(\Delta, \mathcal{R}, P', Q, \mu)}$$

451 ► **Definition 23** (Bisimilarity with inductive predicate, \approx_{ip}). *A symmetric relation \mathcal{R} is a*
 452 *\approx_{ip} -bisimulation if whenever $P \mathcal{R} Q$ with $\Delta \vdash P, Q$, and $P \xrightarrow{\mu} P'$ with $\Delta' \vdash P'$, we can*
 453 *derive $\text{ok}(\Delta \cup \Delta', \mathcal{R}, P', Q, \mu)$. We write \approx_{ip} for the largest \approx_{ip} -bisimulation.*

454 The names in $\Delta \cup \Delta'$ are the reference names that appear in output subject position
 455 in P' or Q . Therefore, when using rule **Ext** of the inductive predicate, the condition $\ell \notin \Delta$
 456 ensures us that the message at ℓ can be added without breaking typability.

457 The following up-to technique allows us to erase common messages on reference names
 458 along the bisimulation game.

459 For this, we use the notation M_s , where s is a finite list of pairs (ℓ, m) , to describe parallel
 460 compositions of outputs on reference names (i.e., $M_s \stackrel{\text{def}}{=} \prod_{(\ell, m) \in s} \bar{\ell}\langle m \rangle$), and $\Delta_s \vdash M_s$ where
 461 Δ_s contains all first components of pairs of s . Intuitively, M_s represents a chunk of store.

462 ► **Definition 24** (\approx_{ip} -bisimulation up to store). *An \approx_{ip} -bisimulation up to store is defined like*
 463 *\approx_{ip} -bisimulation (Definition 23), using a predicate $\text{ok}'(\Delta \cup \Delta', \mathcal{R}, P', Q, \mu)$. This predicate is*

464 defined by a modified version of rule **Ext** where ok' is used instead of ok , both in the premise
 465 and in the conclusion, and the following modified version of the **Base** rule:

$$\text{Base-Up} \frac{P' \equiv P'' \mid M_s \quad \begin{cases} Q \mid \bar{n}\langle m \rangle \Rightarrow \equiv Q'' \mid M_s & \text{for } \mu = n\langle m \rangle \\ Q \stackrel{\mu}{\Rightarrow} \equiv Q'' \mid M_s & \text{otherwise} \end{cases} \quad P'' \mathcal{R} Q''}{ok'(\Delta, \mathcal{R}, P', Q, \mu)}$$

466 Rule **Base-Up** makes it possible to erase common store components before checking that the
 467 processes are related by \mathcal{R} .

468 ► **Proposition 25.** *If \mathcal{R} is a \approx_{ip} -bisimulation up to store, then $\mathcal{R} \subseteq \approx_{ip}$.*

469 ► **Proposition 26** (Soundness of \approx_{ip}). $\approx_{ip} \subseteq \approx$.

470 Intuitively, the inclusion holds because a \approx_{ip} -bisimulation is closed by parallel composition
 471 with M_s processes. We leave the opposite direction, completeness, as an open issue.

472 4.2 Examples

473 We now give examples of uses of the various forms of labelled bisimulation (\approx_a , \approx , \approx_{ip} , \approx_{ip}
 474 up to store) for $A\pi$ to establish equivalences between processes with references. In some
 475 cases, we use the ‘up-to structural congruence’ (\equiv) version of the bisimulations — a standard
 476 ‘up-to’ technique. In the examples we consider barbed equivalence; the results can be lifted
 477 to barbed congruence using closure under substitutions.

478 The first example is about a form of commutativity for the write construct.

479 ► **Example 27.** We wish to establish $!l \triangleleft a. l \triangleleft b \stackrel{e}{\cong}_{ref} !l \triangleleft b. l \triangleleft a$. For this, we prove the law
 480 $!l \triangleleft a. l \triangleleft b \stackrel{e}{\cong}_{ref} !l \triangleleft a \mid !l \triangleleft b$, which will be enough to conclude, by commutativity of parallel
 481 composition. The two given processes are mapped into $A\pi$ as

$$482 \quad P_1 \stackrel{def}{=} !l(_).(\bar{\ell}\langle a \rangle \mid \ell(_).\bar{\ell}\langle b \rangle) \quad \text{and} \quad P_2 \stackrel{def}{=} (!l(_).\bar{\ell}\langle a \rangle) \mid (!l(_).\bar{\ell}\langle b \rangle).$$

483 We can derive $P_1 \approx_a P_2$, using the singleton relation $\mathcal{R} \stackrel{def}{=} \{(P_1, P_2)\}$, and showing that \mathcal{R}
 484 is an asynchronous bisimilarity up-to context and structural congruence [17] (this known
 485 ‘up-to’ technique allows one to remove additional processes created from the replications
 486 after a transition). We can then conclude by Lemma 14.

487 We now consider Examples 3 and 4 from Section 2.

488 **Proof of Example 3.** Let R_1, R_2 be the encodings of P_1, P_2 in the example:

$$489 \quad R_1 \stackrel{def}{=} (\nu \ell)(\bar{\ell}\langle a \rangle \mid \mathcal{E}[P] \mid !l(_).\bar{\ell}\langle a \rangle \mid !l(_).\bar{\ell}\langle b \rangle)$$

$$490 \quad R_2 \stackrel{def}{=} (\nu \ell)(\bar{\ell}\langle b \rangle \mid \mathcal{E}[P] \mid !l(_).\bar{\ell}\langle a \rangle \mid !l(_).\bar{\ell}\langle b \rangle)$$

$$491$$

492 We then have $R_1 \Longrightarrow \equiv R_2$ and $R_2 \Longrightarrow \equiv R_1$, which implies $R_1 \approx_a R_2$ (where \approx_a is
 493 asynchronous bisimilarity), as $\{(R_1, R_2)\} \cup \mathcal{I}$, where $\mathcal{I} = \{(P, P)\}$ is the identity relation, is
 494 an asynchronous bisimulation up to \equiv . We can then conclude by Theorems 8 and 17. ◀

495 **Proof of Example 4.** Let R_1, R_2 be the encodings of P_1, P_2 in the example:

$$496 \quad R_1 \stackrel{def}{=} \ell(_).(\bar{\ell}\langle b \rangle \mid \mathcal{E}[P]) \mid !l(_).\bar{\ell}\langle b \rangle \mid !l(x).(\bar{\ell}\langle x \rangle \mid \ell(_).\bar{\ell}\langle x \rangle)$$

$$497 \quad R_2 \stackrel{def}{=} \mathcal{E}[P] \mid !l(_).\bar{\ell}\langle b \rangle \mid !l(x).(\bar{\ell}\langle x \rangle \mid \ell(_).\bar{\ell}\langle x \rangle)$$

$$498$$

31:14 On the Representation of References in the pi-calculus

499 Then for all m , processes $\bar{\ell}\langle m \rangle \mid R_1$ and $\bar{\ell}\langle m \rangle \mid R_2$ are complete. We define

$$\mathcal{R} \stackrel{\text{def}}{=} \{(R_1 \mid \bar{\ell}\langle m \rangle \mid B_X, R_2 \mid \bar{\ell}\langle m \rangle \mid B_X)\},$$

500 where $X \stackrel{\text{def}}{=} \{x_1, \dots, x_n\}$ is a possibly empty finite set of names, and

$$B_X \stackrel{\text{def}}{=} \ell(_).\bar{\ell}\langle x_1 \rangle \mid \dots \mid \ell(_).\bar{\ell}\langle x_n \rangle$$

501 Then $\mathcal{R} \cup \mathcal{I}$ is a \approx_{ip} -bisimulation.

502 Reusing the same notations, $\mathcal{R}' \stackrel{\text{def}}{=} \{(R_1 \mid B_X, R_2 \mid B_X)\}$ is an \approx_{ip} -bisimulation up to
503 store: this up-to technique allows us to remove the $\bar{\ell}\langle m \rangle$ particles. ◀

504 The following example shows some benefits of using \approx_{ip} and \approx_{ip} up to store in the proof of
505 a property that generalises (the $A\pi$ version of) law (1), which involves a ‘useless read’.

506 ▶ **Example 28.** Consider $\emptyset \vdash P_0 \mathcal{R} Q_0$, where \mathcal{R} is an asynchronous bisimulation, $\text{ObType}(\ell) \in$
507 RefTypes , and x is a fresh name. Then $\emptyset \vdash \ell(x).(P_0 \mid \bar{\ell}\langle x \rangle) \approx Q_0$.

508 In general, $\ell(x).(P_0 \mid \bar{\ell}\langle x \rangle)$ and Q_0 are not related by \approx_a (take $P_0 = Q_0 = \bar{a}\langle n \rangle$), thus
509 the inclusion in Lemma 14 is strict.

510 To prove $\ell(x).(P_0 \mid \bar{\ell}\langle x \rangle) \approx Q_0$ using a \approx -bisimulation, we need a relation such as

$$\begin{aligned} 511 \mathcal{R}_1 &\stackrel{\text{def}}{=} \{(\ell(x).(P_0 \mid \bar{\ell}\langle x \rangle), Q_0)\} \\ 512 &\cup \{(\ell(x).(P_0 \mid \bar{\ell}\langle x \rangle) \mid \bar{\ell}\langle \ell' \rangle \mid \bar{\ell}'\langle m \rangle, Q_0 \mid \bar{\ell}\langle \ell' \rangle \mid \bar{\ell}'\langle m \rangle) \mid \text{for any } m\} \\ 513 &\cup \{(\ell(x).(P_0 \mid \bar{\ell}\langle x \rangle) \mid \bar{\ell}\langle \ell' \rangle \mid \bar{\ell}'\langle m \rangle \mid M_s, Q_0 \mid \bar{\ell}\langle \ell' \rangle \mid \bar{\ell}'\langle m \rangle \mid M_s) \mid \text{for any } m, M_s\} \\ 514 &\cup \{P \mid \bar{\ell}\langle \ell' \rangle \mid \bar{\ell}'\langle m \rangle \mid M_s, Q \mid \bar{\ell}\langle \ell' \rangle \mid \bar{\ell}'\langle m \rangle \mid M_s\} \mid \text{for any } m, M_s, \text{ with } P \mathcal{R} Q\} \end{aligned}$$

516 and prove that $\mathcal{R}_1 \cup \mathcal{R}_1^{-1}$ (where \mathcal{R}_1^{-1} is the inverse of \mathcal{R}_1) is a \approx -bisimulation.

517 We can simplify the proof and avoid the several quantifications in \mathcal{R}_1 (in particular on
518 M_s , whose size is arbitrary), and prove that \mathcal{R}_2 is an \approx_{ip} -bisimulation, for

$$\begin{aligned} 519 \mathcal{R}_2 &\stackrel{\text{def}}{=} \mathcal{R} \cup \{(P \mid \bar{\ell}\langle m \rangle, Q \mid \bar{\ell}\langle m \rangle), \text{ for any } m, \text{ with } P \mathcal{R} Q\} \\ 520 &\cup \{(\ell(x).(P_0 \mid \bar{\ell}\langle x \rangle), Q_0), (Q_0, \ell(x).(P_0 \mid \bar{\ell}\langle x \rangle))\}. \end{aligned}$$

522 The last component of \mathcal{R}_2 is dealt with using rule **Ext** of the inductive predicate (Definition 22),
523 and this brings in the second component (the closure of \mathcal{R} under messages on ℓ).

524 We can simplify the proof further, by removing such second component, and show that
525 \mathcal{R}_3 is an \approx_{ip} -bisimulation up to store, for

$$526 \mathcal{R}_3 \stackrel{\text{def}}{=} \mathcal{R} \cup \{(\ell(x).(P_0 \mid \bar{\ell}\langle x \rangle), Q_0), (Q_0, \ell(x).(P_0 \mid \bar{\ell}\langle x \rangle))\}.$$

528 **5** Future work

529 In languages with store, which are usually sequential languages, bisimulation is commonly
530 defined on *configurations*. In π^{ref} , a configuration would be written $(\nu \tilde{n})\langle P, s \rangle$, where s is
531 an explicit store and \tilde{n} is a set of private names shared between process P and store s . We
532 could in principle read back \approx onto π^{ref} , and define a behavioural equivalence between π^{ref}
533 configurations. The LTS on configurations would then have specific actions to describe how
534 an observer may act on the visible part of the store. The labelled transition semantics for
535 π^{ref} and π^{ref} configurations would however be more complex than those for $A\pi$; for instance
536 the forms of actions, expressing external observations, would be much broader.

537 The swap operation arises naturally in the encoding into $A\pi$. We do not know if and
 538 how swap increases the discriminating power of external observers. We believe that, without
 539 swap, the two processes in Example 5 could not be distinguished. This point deserves further
 540 investigation, which we leave for future work. Similarly we leave for future work proving or
 541 disproving the completeness of the bisimilarity with an inductive predicate (Definition 23).

542 It would be interesting to see if the labelled bisimilarities we have considered, whose
 543 bisimulation clauses are different from those of ordinary bisimilarity, can be recovered in an
 544 abstract setting, e.g., using coalgebras [11, 2, 20]. This would be particularly interesting for
 545 \approx_{ip} -bisimulation, whose definition involves a mixture of induction and coinduction.

546 Equivalences for higher-order languages with state are known to be hard to establish.
 547 Various approaches exist, from Kripke logical relations to trace semantics and game se-
 548 mantics [9, 10, 16, 4]. It would be interesting to compare the proof techniques offered by
 549 these approaches with those shown in this paper, and developments of them. More generally,
 550 more experimentation is needed to test the bisimilarities proposed in this paper and the
 551 associated proof techniques, on examples from high-level languages that include higher-order
 552 features, mutable state, and concurrency.

553 ——— References ———

- 554 1 R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous
 555 pi-calculus. *Theor. Comput. Sci.*, 195(2):291–324, 1998.
- 556 2 F. Bonchi, D. Petrişan, D. Pous, and J. Rot. A general account of coinduction up-to. *Acta*
 557 *Informatica*, pages 1–64, 2016.
- 558 3 S. D. Brookes. The Essence of Parallel Algol. *Inf. Comput.*, 179(1):118–149, 2002.
- 559 4 S. Castellani, P. Clairambault, J. Hayman, and G. Winskel. Non-angelic concurrent game
 560 semantics. In *Foundations of Software Science and Computation Structures - 21st International*
 561 *Conference, FOSSACS 2018*, pages 3–19, 2018.
- 562 5 M. Coppo, M. Dezani-Ciancaglini, N. Yoshida, and L. Padovani. Global progress for dynamic-
 563 ally interleaved multiparty sessions. *Math. Struct. Comput. Sci.*, 26(2):238–302, 2016.
- 564 6 M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully abstract model for the π -calculus. *Inf.*
 565 *Comput.*, 179(1):76–117, 2002.
- 566 7 M. Hennessy. *A distributed Pi-calculus*. Cambridge University Press, 2007.
- 567 8 M. P. Herlihy. Impossibility and universality results for wait-free synchronization. In *Proceedings*
 568 *of the Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC '88*,
 569 pages 276–290, 1988.
- 570 9 C.-K. Hur, D. Dreyer, G. Neis, and V. Vafeiadis. The marriage of bisimulations and kripke
 571 logical relations. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles*
 572 *of Programming Languages, POPL*, pages 59–72, 2012.
- 573 10 G. Jaber and N. Tzevelekos. Trace semantics for polymorphic references. In *Proceedings of the*
 574 *31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16*, pages 585–594,
 575 2016.
- 576 11 B. Jacobs. Introduction to coalgebra. towards mathematics of states and observations. Draft,
 577 2014.
- 578 12 N. Kobayashi. A partially deadlock-free typed process calculus. *Transactions on Programming*
 579 *Languages and Systems*, 20(2):436–482, 1998. A preliminary version in *12th Lics Conf.* IEEE
 580 Computer Society Press 128–139, 1997.
- 581 13 M. Merro, J. Kleist, and U. Nestmann. Mobile objects as mobile processes. *Information and*
 582 *Computation*, 177(2):195–241, 2002.
- 583 14 R. Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall,
 584 1989.

- 585 15 R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Inf.*
586 *Comput.*, 100:1–77, 1992.
- 587 16 A. S. Murawski and N. Tzevelekos. Full abstraction for reduced ML. *Ann. Pure Appl. Logic*,
588 164(11):1118–1143, 2013.
- 589 17 D. Pous and D. Sangiorgi. *Advanced Topics in Bisimulation and Coinduction* (D. Sangiorgi
590 and J. Rutten editors), chapter Enhancements of the coinductive proof method. Cambridge
591 University Press, 2011.
- 592 18 J. C. Reynolds. The essence of ALGOL. In *Algorithmic Languages*, pages 345–372. North-
593 Holland, 1981.
- 594 19 C. Röckl and D. Sangiorgi. A pi-calculus process semantics of concurrent idealised ALGOL. In
595 *Foundations of Software Science and Computation Structure, Second International Conference,*
596 *FoSSaCS’99*, volume 1578 of *Lecture Notes in Computer Science*, pages 306–321. Springer,
597 1999.
- 598 20 J. Rot, F. Bonchi, M. M. Bonsangue, D. Pous, J. Rutten, and A. Silva. Enhanced coalgebraic
599 bisimulation. *Math. Struct. Comput. Sci.*, 27(7):1236–1264, 2017.
- 600 21 D. Sangiorgi. The name discipline of uniform receptiveness. *Theor. Comput. Sci.*, 221(1-2):457–
601 493, 1999.
- 602 22 D. Sangiorgi. Typed pi-calculus at work: A Correctness Proof of Jones’s Parallelisation
603 Transformation on Concurrent Objects. *TAPoS*, 5(1):25–33, 1999.
- 604 23 D. Sangiorgi and D. Walker. *The pi-calculus: a Theory of Mobile Processes*. Cambridge
605 university press, 2003.
- 606 24 I. Stark. A fully abstract domain model for the pi-calculus. In *Proceedings, 11th Annual IEEE*
607 *Symposium on Logic in Computer Science*, pages 36–42. IEEE Computer Society, 1996.

A

 Additional Material for the Examples in Section 2.2

608

609 **Proof of Example 5.** To get a idea of how P_s and Q_s evolve, let us consider first $E \stackrel{\text{def}}{=} (\nu \ell = z)[\]$. Then $E[Q_s]$ can reduce to one of the following:

- 611 1. $(\nu \ell = z)(\nu t)(\bar{t} \mid !t. \ell \triangleleft b. (\bar{c} \mid \ell \triangleleft a. (\bar{t} \mid c)))$
- 612 2. $(\nu \ell = a)(\nu t)(\bar{t} \mid !t. \ell \triangleleft b. (\bar{c} \mid \ell \triangleleft a. (\bar{t} \mid c))) \mid c^n \mid \bar{c}^n$
- 613 3. $(\nu \ell = a)(\nu t)(\ell \triangleleft b. (\bar{c} \mid \ell \triangleleft a. (\bar{t} \mid c)) \mid !t. \ell \triangleleft b. (\bar{c} \mid \ell \triangleleft a. (\bar{t} \mid c))) \mid c^n \mid \bar{c}^n$
- 614 4. $(\nu \ell = b)(\nu t)(\ell \triangleleft a. (\bar{t} \mid c) \mid !t. \ell \triangleleft b. (\bar{c} \mid \ell \triangleleft a. (\bar{t} \mid c))) \mid c^n \mid \bar{c}^{n+1}$.

615 Similarly, $E[P_s]$ can reduce to those four processes but with the role of a and b swapped.
 616 Notice that when $E[Q_s] \Longrightarrow Q'$, then there is a correspondence between the value stored in
 617 ℓ (i.e a or b) and the presence of more \bar{c} processes than c processes (or the same number).

We now consider the following context:

$$E_0 \stackrel{\text{def}}{=} (\nu \ell = z)([\] \mid \ell \bowtie z(x). [x = b]_{s_0. s_1}. (P_{11} \mid P_{12}) \mid \bar{s}_0 \mid \bar{s}_1)$$

$$P_{11} \stackrel{\text{def}}{=} \ell \triangleright (x). [x = z]_{s_{11}} \mid \bar{s}_{11} \qquad P_{12} \stackrel{\text{def}}{=} c. \ell \triangleright (x). [x = z]_{s_{12}} \mid \bar{s}_{12}$$

618 with s_0, s_{11}, s_{12} fresh names.

619 At first \bar{s}_0 and \bar{s}_1 are the only observables, meaning $E_0[P_s] \downarrow_{\bar{s}_0}$ and $E_0[P_s] \downarrow_{\bar{s}_1}$, but then
 620 $E_0[P_s] \longrightarrow \longrightarrow (\nu \ell = z)((\nu t)(\bar{t} \mid !t. \ell \triangleleft a. (\bar{c} \mid \ell \triangleleft b. (\bar{t} \mid c))) \mid s_1. (P_{11} \mid P_{12}) \mid \bar{s}_1) \stackrel{\text{def}}{=} P'$
 621 where the three reductions have been derived using rules **R-Write**, **R-Swap**, and **R-Comm**
 622 respectively. Finally, we have $P' \not\downarrow_{\bar{s}_0}$, whereas $P' \downarrow_{\bar{s}_1}$.

623 Thus, to avoid the observable \bar{s}_0 , process $E_0[Q_s]$ must reduce to a process with b stored
 624 in ℓ before doing the swap in E_0 . This implies that the swap is executed in a state that
 625 corresponds to case 4 above. So for any Q' with $E[Q_s] \Longrightarrow Q'$ and $Q' \not\downarrow_{\bar{s}_0}$ and $Q' \downarrow_{\bar{s}_1}$, such
 626 process Q' has one of the following forms:

- 627 1. $Q'_1 \stackrel{\text{def}}{=} (\nu \ell = a)((\nu t)(\bar{t} \mid !t. \ell \triangleleft b. (\bar{c} \mid \ell \triangleleft a. (\bar{t} \mid c)) \mid c^n \mid \bar{c}^n) \mid s_1. (P_{11} \mid P_{12}) \mid \bar{s}_1)$
- 628 2. $Q'_2 \stackrel{\text{def}}{=} (\nu \ell = a)((\nu t)(\ell \triangleleft b. (\bar{c} \mid \ell \triangleleft a. (\bar{t} \mid c)) \mid !t. \ell \triangleleft b. (\bar{c} \mid \ell \triangleleft a. (\bar{t} \mid c)) \mid c^n \mid \bar{c}^n) \mid s_1. (P_{11} \mid P_{12}) \mid \bar{s}_1)$
- 629 3. $Q'_3 \stackrel{\text{def}}{=} (\nu \ell = b)((\nu t)(\ell \triangleleft a. (\bar{t} \mid c) \mid !t. \ell \triangleleft b. (\bar{c} \mid \ell \triangleleft a. (\bar{t} \mid c)) \mid c^n \mid \bar{c}^{n+1}) \mid s_1. (P_{11} \mid P_{12}) \mid \bar{s}_1)$
- 630 4. $Q'_4 \stackrel{\text{def}}{=} (\nu \ell = z)((\nu t)(\ell \triangleleft a. (\bar{t} \mid c) \mid !t. \ell \triangleleft b. (\bar{c} \mid \ell \triangleleft a. (\bar{t} \mid c)) \mid c^n \mid \bar{c}^{n+1}) \mid s_1. (P_{11} \mid P_{12}) \mid \bar{s}_1)$

631 Then we use either P_{11} or P_{12} depending on the form of Q' . If Q' is of the first three forms,
 632 then we use P_{11} .

634 Indeed, $P' \longrightarrow \longrightarrow (\nu \ell = z)((\nu t)(\bar{t} \mid !t. \ell \triangleleft a. (\bar{c} \mid \ell \triangleleft b. (\bar{t} \mid c))) \mid P_{12}) \stackrel{\text{def}}{=} P''$ using rules
 635 **R-Read** and **R-Comm** respectively. Notice that $P'' \not\downarrow_{\bar{s}_{11}}$. On the other hand, z does not appear
 636 anywhere else than in a matching in Q' , thus there is no reduction $Q' \Longrightarrow Q''$ with $Q'' \not\downarrow_{\bar{s}_{11}}$
 637 for any Q'' .

638 In the other case, it holds that $Q'_4 \longrightarrow \longrightarrow (\nu \ell = z)((\nu t)(\ell \triangleleft a. (\bar{t} \mid c) \mid !t. \ell \triangleleft b. (\bar{c} \mid \ell \triangleleft a. (\bar{t} \mid c)) \mid c^n \mid \bar{c}^n) \mid P_{11}) \stackrel{\text{def}}{=} Q''$ using rules **R-Comm**, **R-Read**, and **R-Comm** respectively.
 639 Then we have $Q'' \not\downarrow_{\bar{s}_{12}}$. However, the only output \bar{c} is behind a write $\ell \triangleleft a$ in P' . Thus, there
 640 is no $P' \Longrightarrow P''$ with $P'' \not\downarrow_{\bar{s}_{12}}$.

642 We can finally conclude $P_s \not\approx_{\text{ref}} Q_s$. ◀

643 **Proof of Example 6.** Recall the definitions of the two processes (we rename the processes
 644 that are given in the main text, to ease readability):

$$645 P \stackrel{\text{def}}{=} (\nu \ell_1 = 0, \ell_2 = 0)(R \mid (\nu t)(\bar{t} \mid !t. \ell_1 \triangleleft 1. \ell_1 \triangleleft 0. \ell_2 \triangleleft 1. \ell_2 \triangleleft 0. \bar{t}))$$

$$646 Q \stackrel{\text{def}}{=} (\nu \ell_1 = 0, \ell_2 = 0)(R \mid (\nu t)(\bar{t} \mid !t. \ell_1 \triangleleft 1. \ell_2 \triangleleft 1. \ell_1 \triangleleft 0. \ell_2 \triangleleft 0. \bar{t}))$$

31:18 On the Representation of References in the pi-calculus

648 To prove their equivalence, we introduce the following processes:

$$649 \quad P' \stackrel{\text{def}}{=} !t. \ell_1(_). (\overline{\ell_1}\langle 1 \rangle \mid \ell_1(_). (\overline{\ell_1}\langle 0 \rangle \mid \ell_2(_). (\overline{\ell_2}\langle 1 \rangle \mid \ell_2(_). (\overline{\ell_2}\langle 0 \rangle \mid \bar{t}))))$$

$$650 \quad Q' \stackrel{\text{def}}{=} !t. \ell_1(_). (\overline{\ell_1}\langle 1 \rangle \mid \ell_2(_). (\overline{\ell_2}\langle 1 \rangle \mid \ell_1(_). (\overline{\ell_1}\langle 0 \rangle \mid \ell_2(_). (\overline{\ell_2}\langle 0 \rangle \mid \bar{t}))))$$

652

$$653 \quad P_1 = Q_1 \stackrel{\text{def}}{=} \bar{t}$$

$$654 \quad P_2 \stackrel{\text{def}}{=} \ell_1(_). (\overline{\ell_1}\langle 1 \rangle \mid \ell_1(_). (\overline{\ell_1}\langle 0 \rangle \mid \ell_2(_). (\overline{\ell_2}\langle 1 \rangle \mid \ell_2(_). (\overline{\ell_2}\langle 0 \rangle \mid \bar{t}))))$$

$$655 \quad Q_2 \stackrel{\text{def}}{=} \ell_1(_). (\overline{\ell_1}\langle 1 \rangle \mid \ell_2(_). (\overline{\ell_2}\langle 1 \rangle \mid \ell_1(_). (\overline{\ell_1}\langle 0 \rangle \mid \ell_2(_). (\overline{\ell_2}\langle 0 \rangle \mid \bar{t}))))$$

$$656 \quad P_3 \stackrel{\text{def}}{=} \ell_1(_). (\overline{\ell_1}\langle 0 \rangle \mid \ell_2(_). (\overline{\ell_2}\langle 1 \rangle \mid \ell_2(_). (\overline{\ell_2}\langle 0 \rangle \mid \bar{t})))$$

$$657 \quad Q_3 \stackrel{\text{def}}{=} \ell_2(_). (\overline{\ell_2}\langle 1 \rangle \mid \ell_1(_). (\overline{\ell_1}\langle 0 \rangle \mid \ell_2(_). (\overline{\ell_2}\langle 0 \rangle \mid \bar{t})))$$

$$658 \quad P_4 \stackrel{\text{def}}{=} \ell_2(_). (\overline{\ell_2}\langle 1 \rangle \mid \ell_2(_). (\overline{\ell_2}\langle 0 \rangle \mid \bar{t}))$$

$$659 \quad Q_4 \stackrel{\text{def}}{=} \ell_1(_). (\overline{\ell_1}\langle 0 \rangle \mid \ell_2(_). (\overline{\ell_2}\langle 0 \rangle \mid \bar{t}))$$

$$660 \quad P_5 = Q_5 \stackrel{\text{def}}{=} \ell_2(_). (\overline{\ell_2}\langle 0 \rangle \mid \bar{t})$$

661

662 P' and Q' are the encodings of the replicated part of P and Q . Then P_i and Q_i are the
663 processes that can be reached from P' and Q' .

664 We now show that the relation $\mathcal{R} \cup \mathcal{R}^{-1}$ is an \approx_{ip} -bisimulation where we have:

$$665 \quad \mathcal{R} \stackrel{\text{def}}{=} \left\{ \begin{array}{l} (\overline{\ell_1}\langle n_1 \rangle \mid (\nu t)(P' \mid P_i), \overline{\ell_1}\langle n'_1 \rangle \mid (\nu t)(Q' \mid Q_j)) \\ \text{for any } n_1, n'_1 \in \{0, 1\}, i, j \end{array} \right\}$$

$$666 \quad \cup \left\{ \begin{array}{l} (\overline{\ell_2}\langle n_2 \rangle \mid (\nu t)(P' \mid P_i), \overline{\ell_2}\langle n'_2 \rangle \mid (\nu t)(Q' \mid Q_j)) \\ \text{for any } n_2, n'_2 \in \{0, 1\}, i, j \end{array} \right\}$$

$$667 \quad \cup \left\{ \begin{array}{l} (\overline{\ell_1}\langle n_1 \rangle \mid \overline{\ell_2}\langle n_2 \rangle \mid (\nu t)(P' \mid P_i), \overline{\ell_1}\langle n'_1 \rangle \mid \overline{\ell_2}\langle n'_2 \rangle \mid (\nu t)(Q' \mid Q_j)) \\ \text{for any } n_1, n'_1, n_2, n'_2 \in \{0, 1\}, i, j \end{array} \right\}$$

669 First, note that the only free names appearing in those processes are ℓ_1 and ℓ_2 . Thus for any
670 $P \mathcal{R} Q$, the only actions to consider are $\tau, \ell_i\langle n \rangle$ and $\overline{\ell_i}\langle n \rangle$, for $i = 1, 2$.

671 For any $P \mathcal{R} Q$, we have:

672 ■ If $P \xrightarrow{\tau} P_0$, then $P_0 \mathcal{R} Q$

673 ■ If $P \xrightarrow{\ell_i\langle n \rangle} P_0$, then $P_0 \mathcal{R} Q \mid \overline{\ell_i}\langle n \rangle$

674 ■ If $P \xrightarrow{\overline{\ell_i}\langle n \rangle} P_0$, then either $Q \xrightarrow{\overline{\ell_i}\langle n \rangle} Q_0$ and $P_0 \mathcal{R} Q_0$, or $Q \xrightarrow{\overline{\ell_i}\langle 1-n \rangle} Q_0$. In this case, we
675 use rule **Ext** (from Definition 22) to add the other location if $\Delta \neq \ell_1, \ell_2$. Then after at
676 most 5 internal transitions (by cycling around the P_i or Q_j), we obtain a process Q_0 that
677 can make the required transition $Q_0 \xrightarrow{\overline{\ell_i}\langle n \rangle} Q'_0$ with $P_0 \mathcal{R} Q'_0$.

678 As $\mathcal{R} \cup \mathcal{R}^{-1}$ is an \approx_{ip} -bisimulation, we have $\mathcal{R} \subseteq \approx$. Moreover, $(\nu \ell_1, \ell_2)(\mathcal{E}[[R]] \mid [])$ is
679 an active context, so this implies $\mathcal{E}[[P]] \approx \mathcal{E}[[Q]]$. By Theorems 21 and 17, we can conclude
680 $P \cong_{\text{ref}}^e Q$.

681 To extend this result to barbed congruence, we notice that for all σ ,

682 1. either $P\sigma = (\nu \ell_1 = 0, \ell_2 = 0)(R\sigma \mid (\nu t)(\bar{t} \mid !t. \ell_1 \triangleleft 1. \ell_1 \triangleleft 0. \ell_2 \triangleleft 1. \ell_2 \triangleleft 0. \bar{t}))$

683 2. or $P\sigma = (\nu \ell_1 = 0, \ell_2 = 0)(R\sigma \mid (\nu t)(\bar{t} \mid !t. \ell_1 \triangleleft 0. \ell_1 \triangleleft 0. \ell_2 \triangleleft 0. \ell_2 \triangleleft 0. \bar{t}))$

684 3. or $P\sigma = (\nu \ell_1 = 1, \ell_2 = 1)(R\sigma \mid (\nu t)(\bar{t} \mid !t. \ell_1 \triangleleft 1. \ell_1 \triangleleft 1. \ell_2 \triangleleft 1. \ell_2 \triangleleft 1. \bar{t}))$

685 As $P \cong_{\text{ref}}^e Q$ holds for any R , it also holds for any $R\sigma$, which prove the first case. Moreover,
 686 the proof never uses the fact that 0 and 1 are distinct, so we can prove in the same way that
 687 cases 2 and 3 hold.

688 We conclude $P \cong_{\text{ref}} Q$. ◀

689 We now present an additional example, which corresponds to a generalisation of Example 6.

690 ▶ **Example 29.** Here we remove the assumption that the two references can only hold values
 691 0 and 1. This enables the context to store fresh names in references. If used with the original
 692 processes, these are distinguished by using those fresh values to block transition along the
 693 lines of Example 5. To make these processes equivalent again, we could add in parallel a
 694 buffer as in Example 4. However, by making these additions, we would also enable P_1 to
 695 desynchronise the content in ℓ_1 and ℓ_2 and have $(1, 1)$. The solution is to prevent those
 696 buffers from writing at a different ‘time’ than the ‘time’ they have read. For this we introduce
 697 a more complex buffer B_i^j . Consider the following processes:

$$\begin{aligned}
 698 \quad B_i^j &\stackrel{\text{def}}{=} r(x^j). \mathbf{0} \mid !r(x^j). t_i. \ell^j \bowtie x^j(y^j). (\bar{r}(y^j) \mid \bar{t}_i) \\
 699 \quad S_i^j &\stackrel{\text{def}}{=} !t_i. \ell^j \triangleright (x^j). (\bar{t}_i \mid (\nu r)(\bar{r}(x^j) \mid B_i^j)) \\
 700 \quad P &\stackrel{\text{def}}{=} (\nu t_1, t_2, t_3, t_4) \left(\bar{t}_1 \mid !t_1. \ell^1 \triangleleft 1. \bar{t}_2 \mid S_1^1 \mid S_1^2 \mid !t_2. \ell^1 \triangleleft 0. \bar{t}_3 \mid S_2^1 \mid S_2^2 \right. \\
 701 \quad &\quad \left. \mid !t_3. \ell^2 \triangleleft 1. \bar{t}_4 \mid S_3^1 \mid S_3^2 \mid !t_4. \ell^2 \triangleleft 0. \bar{t}_1 \mid S_4^1 \mid S_4^2 \right) \\
 702 \quad Q &\stackrel{\text{def}}{=} (\nu t_1, t_2, t_3, t_4) \left(\bar{t}_1 \mid !t_1. \ell^1 \triangleleft 1. \bar{t}_2 \mid S_1^1 \mid S_1^2 \mid !t_2. \ell^2 \triangleleft 1. \bar{t}_3 \mid S_2^1 \mid S_2^2 \right. \\
 703 \quad &\quad \left. \mid !t_3. \ell^1 \triangleleft 0. \bar{t}_4 \mid S_3^1 \mid S_3^2 \mid !t_4. \ell^2 \triangleleft 0. \bar{t}_1 \mid S_4^1 \mid S_4^2 \right)
 \end{aligned}$$

704 We have $P \cong_{\text{ref}} Q$. If we take $E \stackrel{\text{def}}{=} (\nu \ell^1 = 0)(\nu \ell^2 = 0)[\]$, we have
 705 $E[Q] \longrightarrow (\nu \ell^1 = 1)(\nu \ell^2 = 1)Q'$ for some Q' . However, there is no sequence of reductions
 706 such that $E[P] \Longrightarrow (\nu \ell^1 = 1)(\nu \ell^2 = 1)P'$ for any P' .

710 If we forget all S_i^j 's, then these processes are similar to the ‘loop’ used in the previous
 711 example but split into multiple replications. Those S_i^j 's help to equate the two processes
 712 even if the context can write any value in ℓ_1, ℓ_2 .

713 Process S_i^j can only be activated when \bar{t}_i is available. It then reads the content of ℓ_j to
 714 initialise a new buffer B_i^j .

715 Process B_i^j contains value x_i^j that is the object of $\bar{r}(x_i^j)$. Process B_i^j can be stopped by
 716 making the communication with the first input on r , or can be used to swap its content with
 717 the content of ℓ^j . Note that this swap can only be done when \bar{t}_i is available, so it cannot be
 718 used to desynchronise the content in ℓ_1 , and ℓ_2 .

719 **B** Definitions and Results about $A\pi$ with references

720 **B.1** Operational Semantics of $A\pi$: Reduction and Labelled Transitions

721 **Reduction**

Structural congruence is defined as the smallest congruence that satisfies the following axioms:

$$\begin{aligned}
 P \mid \mathbf{0} &\equiv P & P \mid Q &\equiv Q \mid P & P \mid (Q \mid R) &\equiv (P \mid Q) \mid R & !P &\equiv P \\
 P \mid (\nu n)Q &\equiv (\nu n)P \mid Q \text{ if } n \notin \text{fn}(P) & (\nu n)(\nu m)P &\equiv (\nu m)(\nu n)P & (\nu n)\mathbf{0} &\equiv \mathbf{0} \\
 [x = x]P &\equiv P
 \end{aligned}$$

31:20 On the Representation of References in the pi-calculus

$$\begin{array}{c}
\text{Inp: } \frac{}{n(x).P \xrightarrow{n\langle m \rangle} P\{m/x\}} \qquad \text{Out: } \frac{}{\bar{n}\langle m \rangle \xrightarrow{} \mathbf{0}} \\
\text{Open: } \frac{P \xrightarrow{\bar{n}\langle m \rangle} P'}{(\nu m)P \xrightarrow{(\nu m)\bar{n}\langle m \rangle} P'} \text{ if } m \neq n \qquad \text{Rep: } \frac{P \mid !P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'} \\
\text{Res: } \frac{P \xrightarrow{\mu} P'}{(\nu n)P \xrightarrow{\mu} (\nu n)P'} \text{ if } n \notin \mu \qquad \text{Par: } \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} \text{ if } \text{bn}(\mu) \cap \text{fn}(Q) = \emptyset \\
\text{Comm: } \frac{P \xrightarrow{n\langle m \rangle} P' \quad Q \xrightarrow{\bar{n}\langle m \rangle} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \\
\text{Close: } \frac{P \xrightarrow{n\langle m \rangle} P' \quad Q \xrightarrow{(\nu m)\bar{n}\langle m \rangle} Q'}{P \mid Q \xrightarrow{\tau} (\nu m)(P' \mid Q')} \text{ if } m \notin \text{fn}(P) \qquad \text{Match: } \frac{P \xrightarrow{\mu} P'}{[n = n]P \xrightarrow{\mu} P'}
\end{array}$$

■ **Figure 3** Labelled Transition Semantics for $A\pi$

Active contexts in $A\pi$ are defined by:

$$E ::= [] \mid E \mid P \mid (\nu n)E .$$

Reduction is defined by the following rules:

$$\frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q} \qquad \frac{P \longrightarrow P'}{E[P] \longrightarrow E[Q]} \qquad \frac{}{n(x).P \mid \bar{n}\langle m \rangle \longrightarrow P\{m/x\}}$$

722 Labelled Transition Semantics

Actions of the LTS are defined as follows:

$$\mu ::= n\langle m \rangle \mid \bar{n}\langle m \rangle \mid (\nu m)\bar{n}\langle m \rangle \mid \tau .$$

723 Transitions are defined in Figure 3. The symmetric versions of rules PAR, COM and CLOSE
724 are omitted. Weak transitions are defined by $\Rightarrow \stackrel{\text{def}}{=} \tau^*$, $\xRightarrow{\mu} \stackrel{\text{def}}{=} \Rightarrow \xrightarrow{\mu} \Rightarrow$, and $\xRightarrow{\bar{\mu}} \stackrel{\text{def}}{=} \xRightarrow{\mu}$ if
725 $\mu \neq \tau$ and \Rightarrow otherwise.

726 B.2 Type System for Output Receptiveness: Proof of Subject 727 Reduction

728 We prove subject reduction, which we first recall:

729 ► **Proposition 10** (Subject reduction). *If $\Delta \vdash P$ and $P \xrightarrow{\mu} P'$, then*

- 730 1. *if $\mu = \tau$, $\mu = \bar{a}\langle m \rangle$, $\mu = a\langle m \rangle$ or $\mu = (\nu b)\bar{a}\langle b \rangle$, then $\Delta \vdash P'$.*
- 731 2. *if $\mu = (\nu \ell)\bar{a}\langle \ell \rangle$ then $\Delta, \ell \vdash P'$.*
- 732 3. *if $\mu = \ell\langle m \rangle$ and $\ell \notin \Delta$, then $\Delta, \ell \vdash P'$.*
- 733 4. *if $\ell \notin \Delta$, then $\Delta, \ell \vdash P \mid \bar{\ell}\langle m \rangle$.*
- 734 5. *if $\mu = \bar{\ell}\langle m \rangle$ or $\mu = (\nu b)\bar{\ell}\langle b \rangle$, then $\Delta - \ell \vdash P'$.*

735 6. if $\mu = (\nu\ell')\bar{\ell}\langle\ell'\rangle$, then $(\Delta - \ell), \ell' \vdash P'$.

736 **Proof.** We note the type of P' as Δ' .

737 ■ For $\mu = n\langle m \rangle$, we have $P \equiv (\nu\tilde{a}, \tilde{\ell})(n(x). P_1 \mid P_2)$ and $P' \equiv (\nu\tilde{a}, \tilde{\ell})(P_1\{m/x\} \mid P_2)$ for
738 some $\tilde{a}, \tilde{\ell}, P_1, P_2$ with $m \notin \tilde{a} \cup \tilde{\ell}$.

739 We take $\Delta_1 \vdash P_1$ and $\Delta_2 \vdash P_2$. This means $\Delta_2 = \Delta \uplus \tilde{\ell}$. Depending on whether n is
740 a reference name or not, we have that $\Delta_1 = n$ or $\Delta_1 = \emptyset$ respectively. In both cases,
741 $\Delta_1 \vdash P_1\{m/x\}$ and $\Delta_1 \uplus \Delta_2 \vdash P_1\{m/x\} \mid P_2$. Thus $\Delta' = \Delta_1 \uplus \Delta$, meaning that $\Delta' = \Delta, n$
742 if n is a reference name and $\Delta' = \Delta$ otherwise.

743 ■ For $\mu = \bar{n}\langle m \rangle$, we have $P \equiv (\nu\tilde{a}, \tilde{\ell})(\bar{n}\langle m \rangle \mid P_2)$ and $P' \equiv (\nu\tilde{a}, \tilde{\ell})P_2$ for some $\tilde{a}, \tilde{\ell}, P_1, P_2$
744 with $n, m \notin \tilde{a} \cup \tilde{\ell}$.

745 We take $\Delta_1 \vdash \bar{n}\langle m \rangle$ and $\Delta_2 \vdash P_2$. This means $\Delta_1 \uplus \Delta_2 = \Delta \uplus \tilde{\ell}$, and $\Delta_2 = \Delta' \uplus \tilde{\ell}$. As
746 $n \notin \tilde{\ell}$, $\Delta' = \Delta \setminus \Delta_1$. Thus $\Delta' = \Delta - \ell$ if n is a reference name and $\Delta' = \Delta$ otherwise.

747 ■ For $\mu = (\nu m)\bar{n}\langle m \rangle$, we have $P \equiv (\nu\tilde{a}, \tilde{\ell}, m)(\bar{n}\langle m \rangle \mid P_2)$ and $P' \equiv (\nu\tilde{a}, \tilde{\ell})P_2$ for some
748 $\tilde{a}, \tilde{\ell}, P_1, P_2$ with $n \notin \tilde{a} \cup \tilde{\ell} \cup \{m\}$. With the same notation, we have that $\Delta_2 = \Delta' \uplus \tilde{\ell}$, and
749 if m is a plain name then $\Delta_1 \uplus \Delta_2 = \Delta \uplus \tilde{\ell}$ and $\Delta_1 \uplus \Delta_2 = \Delta \uplus \tilde{\ell}, m$ otherwise. Thus we
750 have four cases for Δ' shown in the table below:

$n \setminus m$	plain	reference
plain	Δ	Δ, m
reference	$\Delta \setminus n$	$\Delta, m \setminus n$

752 ■ For $\mu = \tau$, we look at the interaction that has occurred. This can be mimicked using two
753 transitions, one for the output and one for the input for which we have already proven
754 the resulting typing.

755 ■ $P \xrightarrow{\bar{a}\langle m \rangle} \xrightarrow{a\langle m \rangle} P'$, it is straightforward.

756 ■ $P \xrightarrow{(\nu b)\bar{a}\langle b \rangle} \xrightarrow{a\langle b \rangle} P''$ with $P' = (\nu b)P''$. We have $\Delta \vdash P''$ then $\Delta \vdash (\nu b)P''$.

757 ■ $P \xrightarrow{(\nu\ell)\bar{a}\langle \ell \rangle} \xrightarrow{a\langle \ell \rangle} P''$ with $P' = (\nu\ell)P''$. We have $\Delta, \ell \vdash P''$ then $\Delta \vdash (\nu\ell)P''$

758 ■ $P \xrightarrow{\bar{\ell}\langle m \rangle} \xrightarrow{\ell\langle m \rangle} P'$. We have $\ell \notin \Delta$ after the output, so we can subject reduction for the
759 input transition.

760 ■ $P \xrightarrow{(\nu b)\bar{\ell}\langle b \rangle} \xrightarrow{\ell\langle b \rangle} P''$ with $P' = (\nu b)P''$. We have $\Delta \vdash P''$ then $\Delta \vdash (\nu b)P''$

761 ■ $P \xrightarrow{(\nu\ell')\bar{\ell}\langle \ell' \rangle} \xrightarrow{\ell\langle \ell' \rangle} P''$ with $P' = (\nu\ell')P''$. We have $\Delta, \ell' \vdash P''$ then $\Delta \vdash (\nu\ell')P''$

762 ◀

763 B.3 Properties of the encoding

764 ▶ **Lemma 16.** If $\emptyset \vdash P$, then there exists R in π^{ref} such that $\mathcal{E}[[R]] \approx_a P$.

765 **Proof.** We construct R by induction on the structure of P , we only discuss the two cases
766 below, the other cases are immediate.

767 ■ For $\emptyset \vdash (\nu\ell)P$, we know that $\ell \vdash P$ so we have two cases according to Lemma 9:

768 ■ $P \equiv \bar{\ell}\langle m \rangle \mid P'$. Thus $(\nu\ell)P \equiv (\nu\ell)(\bar{\ell}\langle m \rangle \mid P')$ with $\emptyset \vdash P'$. By induction, we have Q'
769 with $\mathcal{E}[[Q']] \approx_a P'$. Therefore we have $\mathcal{E}[(\nu\ell = m)Q'] \approx_a (\nu\ell)P$.

770 ■ $P \equiv (\nu m)(\bar{\ell}\langle m \rangle \mid P')$. We reason by induction on the type of ℓ . If m is a plain name, we
771 can conclude as above with $\mathcal{E}[(\nu m)(\nu\ell = m)Q']$. Otherwise, m is reference name and
772 there exists R such that $(\nu m)(\nu\ell)(\bar{\ell}\langle m \rangle \mid P') \equiv \mathcal{E}[[R]]$. As $(\nu\ell)P \equiv (\nu m)(\nu\ell)(\bar{\ell}\langle m \rangle \mid$
773 $P')$, we are done.

774 ■ For $\emptyset \vdash \ell(x).P$, we know that $\ell \vdash P$ then

775 ■ either $P \equiv \bar{\ell}\langle m \rangle \mid P'$ with $\emptyset \vdash P'$. By induction, we have $\mathcal{E}[[Q']] \approx_a P'$ in which case
776 we take $\ell \triangleright (x).Q'$ or $\ell \bowtie m(x).Q'$ depending on whether $m = x$ or not,

31:22 On the Representation of References in the pi-calculus

777 ■ or $P \equiv (\nu m)(\bar{\ell}\langle m \rangle \mid P')$ and then $\ell(x).P \approx_a (\nu m)\ell(x).(\bar{\ell}\langle m \rangle \mid P')$ and we can refer
778 to the first case. ◀

779

780 B.4 Characterisation of \cong_{Arn}^e using \approx

781 B.4.1 Soundness

782 Reference Bisimulation up to \equiv .

783 Up-to techniques ease the task of proving bisimilarity between processes. Informally, the
784 general idea is to use an extra relation (for instance \equiv), and when we need to prove that
785 $P \mathcal{R} Q$, instead of proving that $P' \mathcal{R} Q'$ (for some P', Q' that satisfy the required conditions),
786 we show $P' \equiv \mathcal{R} \equiv Q'$. This often leads to smaller relations, which are easier to check.

787 We say that a relation \mathcal{R} is \equiv -closed under allocation if $P \mathcal{R} Q$ implies $P \mid \bar{\ell}\langle n \rangle \equiv \mathcal{R} \equiv$
788 $Q \mid \bar{\ell}\langle n \rangle$ for any $\bar{\ell}\langle n \rangle$ such that $P \mid \bar{\ell}\langle n \rangle$ and $Q \mid \bar{\ell}\langle n \rangle$ are well-typed.

789 ▶ **Definition 30** (Reference Bisimulation up to \equiv). *A symmetric relation \mathcal{R} that is \equiv -closed*
790 *under allocation is a reference bisimulation up to \equiv if whenever $P \mathcal{R} Q$ with P, Q complete,*
791 *$\Delta \vdash P, Q$ and $P \xrightarrow{\alpha} P'$ with $\Delta \vdash \alpha$, we have*

- 792 1. *either there exists Q' such that $Q \xrightarrow{\hat{\alpha}} Q'$ and $P' \equiv \mathcal{R} \equiv Q'$ for some Q'*
- 793 2. *or α is an input $a\langle m \rangle$ and $Q \mid \bar{a}\langle m \rangle \Rightarrow Q'$ with $P' \equiv \mathcal{R} \equiv Q'$ for some Q' .*

794 ▶ **Proposition 31.** *If \mathcal{R} is a reference bisimulation up to \equiv , then $\mathcal{R} \subseteq \approx$.*

795 **Proof.** $\equiv \mathcal{R} \equiv$ is a reference bisimulation and $\mathcal{R} \subseteq \equiv \mathcal{R} \equiv$. ◀

796 ▶ **Lemma 32.** *If $P \approx Q$, then $(\nu n)P \approx (\nu n)Q$.*

797 **Proof.** $\mathcal{R} \stackrel{\text{def}}{=} \{((\nu n)P, (\nu n)Q) \text{ s.t. } P \approx Q\} \cup \approx$ is a reference bisimulation up to \equiv . ◀

798 ▶ **Definition 33** (Bisimulation up to restriction and up to \equiv). *A symmetric relation \mathcal{R} \equiv -closed*
799 *under allocation is a reference bisimulation up to restriction and up to \equiv if whenever $P \mathcal{R} Q$*
800 *with P, Q complete, $\Delta \vdash P, Q$ and $P \xrightarrow{\alpha} P'$ with $\Delta \vdash \alpha$, then*

- 801 1. *either there exists Q' such that $Q \xrightarrow{\hat{\alpha}} Q'$, $P' \equiv (\nu \tilde{n})P''$, $Q' \equiv (\nu \tilde{n})Q''$ with $P'' \mathcal{R} Q''$ for*
802 *some P'', Q', Q'', \tilde{n}*
- 803 2. *or α is an input $a\langle m \rangle$ and $Q \mid \bar{a}\langle m \rangle \Rightarrow Q'$ with $P' \equiv (\nu \tilde{n})P''$, $Q' \equiv (\nu \tilde{n})Q''$ with $P'' \mathcal{R} Q''$*
804 *for some P'', Q', Q'', \tilde{n} .*

805 ▶ **Lemma 34.** *If \mathcal{R} is a reference bisimulation up to restriction and up to \equiv , then $\mathcal{R} \subseteq \approx$.*

806 **Proof.** $\mathcal{R}' \stackrel{\text{def}}{=} \{((\nu \tilde{n})P, (\nu \tilde{n})Q) \text{ s.t. } P \mathcal{R} Q\}$ is a reference bisimulation up to \equiv . ◀

807 The following lemma uses notation M_s , which has been introduced before Definition 24.

808 ▶ **Lemma 35** (Extractable store). *Let $\Delta \vdash P$, then $P \equiv (\nu \tilde{n})(M_s \mid P')$ with $\emptyset \vdash P'$ for some*
809 *M_s .*

810 **Proof.** We reason by induction on the structure of P . There are two cases depending on the
811 size of Δ .

- 812 ■ If $\Delta = \emptyset$, then nothing has to be done.
- 813 ■ If $\Delta = \Delta', \ell$, then by Lemma 9, $P \equiv (\nu \tilde{n})(\bar{\ell}\langle m \rangle \mid Q)$ with $\Delta', \Delta'' \vdash Q$. By induction,
814 $Q \equiv (\nu \tilde{n}')(M_s \mid Q')$ with $\emptyset \vdash Q'$. Therefore, $P \equiv (\nu \tilde{n}, \tilde{n}')(M_{s'} \mid Q')$ with
815 $M_{s'} = \bar{\ell}\langle m \rangle \mid M_s$.

816

817 For $\ell \vdash P$, this lemma can be strengthened to $P \equiv (\nu \tilde{n})(\bar{\ell}\langle m \rangle \mid M_s \mid P')$ with $\emptyset \vdash P'$.

818 We can now prove substitutivity for \approx under parallel composition.

819 ► **Lemma 19.** *If $P \approx Q$, and $\emptyset \vdash R$, then $P \mid R \approx Q \mid R$.*

Proof. We show that \mathcal{R} is a bisimulation up to restriction, with

$$\mathcal{R} \stackrel{\text{def}}{=} \{(P \mid R, Q \mid R) \text{ s.t. } P \approx Q, \emptyset \vdash R\}$$

820 ■ \mathcal{R} is closed by allocation.

821 ■ Suppose $P \mid R$ and $Q \mid R$ are complete, and $P \mid R \xrightarrow{\alpha} \tilde{P}$ with $\Delta \vdash \alpha$ we distinguish according to the last rule used (Par, Comm or Close)

822 ■ If $P \mid R \xrightarrow{\alpha} P' \mid R$, first note that P, Q are complete, so either $Q \xrightarrow{\alpha} Q'$ and $P' \approx Q'$ or
823 $\alpha = a\langle m \rangle$ and $Q \mid \bar{a}\langle m \rangle \Rightarrow Q'$ and $P' \approx Q'$. In both cases, we have $P' \mid R \approx Q' \mid R$.

824 ■ If $P \mid R \xrightarrow{\alpha} P \mid R'$, then $Q \mid R \xrightarrow{\alpha} Q \mid R'$. For $\alpha = \tau, a\langle m \rangle, \bar{a}\langle m \rangle, (\nu b)\bar{a}\langle b \rangle$, we have
825 $\emptyset \vdash R'$. The only remaining case is when $\alpha = (\nu \ell)\bar{a}\langle \ell \rangle$ (by typing, R cannot perform
826 an output on a reference). In that case, $\ell \vdash R'$. Thus $R' \equiv (\nu \tilde{n})(M_s \mid R'')$ with $\emptyset \vdash R''$.
827 By definition $P \mid M_s \approx Q \mid M_s$ hence $P \mid M_s \mid R'' \mathcal{R} Q \mid M_s \mid R''$, which is sufficient
828 as $P \mid R' \equiv (\nu \tilde{n})(P \mid M_s \mid R'')$ and $Q \mid R' \equiv (\nu \tilde{n})(Q \mid M_s \mid R'')$.

829 ■ If $P \mid R \xrightarrow{\tau} P' \mid R'$, we distinguish according to the action performed by P :

830 * For $P \xrightarrow{\bar{a}\langle n \rangle} P'$ or $P \xrightarrow{a\langle n \rangle} P'$, then $R \xrightarrow{a\langle n \rangle} R'$ and $R \xrightarrow{\bar{a}\langle n \rangle} R'$ respectively, so
831 $\emptyset \vdash R'$. The remaining part of the proof is standard π -calculus reasoning.

832 * For $P \xrightarrow{\bar{\ell}\langle n \rangle} P'$, then $R \xrightarrow{\ell\langle n \rangle} R'$ with $\ell \vdash R'$. By Lemma 35,
833 $R' \equiv (\nu \tilde{n})(\bar{\ell}\langle m \rangle \mid M_s \mid R'')$ with $\emptyset \vdash R''$. By definition, $P \mid M_s \approx Q \mid M_s$.

834 Moreover both processes are complete and $P \mid M_s \xrightarrow{\bar{\ell}\langle n \rangle[m]} P' \mid \bar{\ell}\langle m \rangle \mid M_s$. So

835 $Q \mid M_s \xrightarrow{\bar{\ell}\langle n \rangle[m]} Q'$ and $P' \mid \bar{\ell}\langle m \rangle \mid M_s \approx Q'$. As all names in subject position
836 in M_s are fresh for Q , we have $Q \mid R \xrightarrow{\tau} \equiv (\nu \tilde{n})(Q' \mid R'')$. Moreover we have
837 $P' \mid R' \equiv (\nu \tilde{n})(P' \mid \bar{\ell}\langle m \rangle \mid R'')$, thus we are done.

838 ■ If $P \mid R \xrightarrow{\tau} (\nu n)(P' \mid R')$, then the reasoning is similar.

840

841 B.4.2 Completeness

842 We prove completeness. For this, we need the following lemmas.

843 ► **Lemma 36.** *If $(\nu n)(P \mid \bar{s}\langle n \rangle) \cong_{\text{Arn}}^e (\nu n)(Q \mid \bar{s}\langle n \rangle)$ with s fresh for P and Q , then*
844 $P \cong_{\text{Arn}}^e Q$.

Proof. We show that the following relation \mathcal{R} is included in barbed equivalence.

$$\mathcal{R} = \{(P, Q) \mid (\nu n)(P \mid \bar{s}\langle n \rangle) \cong_{\text{Arn}}^e (\nu n)(Q \mid \bar{s}\langle n \rangle) \text{ with } s \text{ fresh}\}$$

845 We will note $P_1 = (\nu n)(P \mid \bar{s}\langle n \rangle)$ and $Q_1 = (\nu n)(Q \mid \bar{s}\langle n \rangle)$

846 ■ If $P \longrightarrow P'$, then $P_1 \longrightarrow (\nu n)(P' \mid \bar{s}\langle n \rangle)$ so $Q_1 \Longrightarrow Q_2$ with $(\nu n)(P' \mid \bar{s}\langle n \rangle) \cong_{\text{Arn}}^e Q_2$.
847 But we have $Q_2 \equiv (\nu n)(Q' \mid \bar{s}\langle n \rangle)$ and $Q \Longrightarrow Q'$.

848 ■ If $P \downarrow_{\bar{a}}$, then we have two cases:

849 ■ $a \neq n$, then $P_1 \downarrow_{\bar{a}}$ so $Q_1 \downarrow_a$ meaning that $Q \downarrow_a$ as $a \neq s$.

31:24 On the Representation of References in the pi-calculus

850 ■ $a = n$, then we consider $E \stackrel{\text{def}}{=} [] \mid s(x).x(_).\bar{s}'$ for a fresh s' . $E[P_1] \longrightarrow \longrightarrow (\nu n)(P \mid \bar{s}')$
851 so $(\nu n)(P \mid \bar{s}') \Downarrow_{\bar{s}'}$. Therefore, $E[Q_1] \Longrightarrow \Longrightarrow \Downarrow_{\bar{s}'}$ which just means that $E[Q_1] \Downarrow_{\bar{s}'}$.
852 However this can only be done by doing a communication on n , thus we must have
853 $Q \Downarrow_{\bar{n}}$.

854 ■ Take an active context E completing for P and Q , we assume s is fresh for E , then
855 $E' \stackrel{\text{def}}{=} E \mid s(x).\bar{s}'\langle x \rangle$ with s' fresh is also completing for P_1 and Q_1 , so $E'[P_1] \cong_{\text{Arn}}^e$
856 $E'[Q_1]$. We then have $E'[P_1] \longrightarrow (\nu n)(E[P] \mid \bar{s}'\langle n \rangle)$, so $E'[Q_1] \Longrightarrow Q'$ with $(\nu n)(E[P] \mid$
857 $\bar{s}'\langle n \rangle) \cong_{\text{Arn}}^e Q'$, meaning in particular that $Q' \not\Downarrow_{\bar{s}}$ and $Q' \Downarrow_{\bar{s}'}$ which is only possible is $Q' \Downarrow_{\bar{s}'}$.
858 Moreover, we have that $E'[Q_1] \longrightarrow (\nu n)(E[Q] \mid s(x).\bar{s}'\langle x \rangle) \Longrightarrow Q'$. The same also apply
859 symmetrically for $E'[Q_1] \longrightarrow (\nu n)(E[Q] \mid s'(x).\bar{s}''\langle x \rangle) \cong_{\text{Arn}}^e P'$ for some P' . Thus we
860 have $(\nu n)(E[P] \mid \bar{s}''\langle x \rangle) \Longrightarrow P' \cong_{\text{Arn}}^e (\nu n)(E[Q] \mid \bar{s}''\langle x \rangle) \Longrightarrow Q' \cong_{\text{Arn}}^e (\nu n)(E[P] \mid \bar{s}'\langle x \rangle)$
861 which implies $(\nu n)(E[P] \mid \bar{s}'\langle x \rangle) \cong_{\text{Arn}}^e (\nu n)(E[Q] \mid \bar{s}'\langle x \rangle)$.

862

863 ► **Lemma 37.** *If $P \mid [x = y]\bar{s} \cong_{\text{Arn}}^e Q \mid [x = y]\bar{s}$ with $x \neq y$, then $P \cong_{\text{Arn}}^e Q$.*

864 **Proof.** We have $[x = y]\bar{s} \approx_a \mathbf{0}$ so $P \mid [x = y]\bar{s} \approx_a P$ and similarly for Q . Thus by Lemma 14,
865 $P \cong_{\text{Arn}}^e P \mid [x = y]\bar{s} \cong_{\text{Arn}}^e Q \mid [x = y]\bar{s} \cong_{\text{Arn}}^e Q$. ◀

866 This result can be extended to an arbitrary number of $[x = y]\bar{s}$ in parallel.

867 **Proof of Completeness.** We show that \cong_{Arn}^e is a reference bisimulation:

868 ■ It is closed by allocation

869 ■ Take P, Q complete with $F \stackrel{\text{def}}{=} \text{fn}(P) \cup \text{fn}(Q)$, $P \cong_{\text{Arn}}^e Q$ and $P \xrightarrow{\alpha} P'$

- 870 1. When $\alpha = \tau$, we take $E \stackrel{\text{def}}{=} []$. Then $E[P] \longrightarrow P'$. So we have $Q \Longrightarrow Q'$ with
871 $P' \cong_{\text{Arn}}^e Q'$.
- 872 2. When $\alpha = a(n)$, we take $E \stackrel{\text{def}}{=} [] \mid \bar{a}\langle n \rangle$. Then $E[P] \longrightarrow P'$. So we have $Q \mid \bar{a}\langle n \rangle \Longrightarrow Q'$
873 with $P' \cong_{\text{Arn}}^e Q'$.
- 874 3. When $\alpha = \bar{a}\langle n \rangle$, we take $E \stackrel{\text{def}}{=} [] \mid a(x).[x = n]s \mid \bar{s}$ with s fresh. Then $E[P] \longrightarrow \longrightarrow P'$
875 with $E[P] \Downarrow_{\bar{s}}$ and $P' \not\Downarrow_{\bar{s}}$. This implies that $E[Q] \Longrightarrow Q'$ with $P' \cong_{\text{Arn}}^e Q'$. So we have
876 $Q' \not\Downarrow_{\bar{s}}$, which is only possible if $Q \xrightarrow{\bar{a}\langle n \rangle} Q'$.
- 877 4. When $\alpha = (\nu n)\bar{a}\langle n \rangle$, we take $E \stackrel{\text{def}}{=} [] \mid a(x).(s \mid \bar{s}'\langle x \rangle \mid \prod_{m \in F}[x = m]\bar{s}) \mid \bar{s}$ with
878 s, s' fresh. Then $E[P] \longrightarrow \longrightarrow (\nu n)(P' \mid \prod_{m \in F}[n = m]\bar{s} \mid \bar{s}'\langle n \rangle)$. This implies
879 that $E[Q] \Longrightarrow Q''$ with $(\nu n)(P' \mid \prod_{m \in F}[x = m]\bar{s} \mid \bar{s}'\langle n \rangle) \cong_{\text{Arn}}^e Q''$. As $Q'' \not\Downarrow_{\bar{s}}$, we
880 necessarily have $Q'' \equiv (\nu n)(Q' \mid \prod_{m \in F}[n = m]\bar{s} \mid \bar{s}'\langle n \rangle)$. By Lemmas 36 and 37, this
881 means that $P' \cong_{\text{Arn}}^e Q'$. But then $Q \xrightarrow{(\nu n)\bar{a}\langle n \rangle} Q'$ so we can conclude.
- 882 5. When $\alpha = \bar{\ell}\langle n \rangle[m]$, we take $E \stackrel{\text{def}}{=} [] \mid \ell(x).(\bar{\ell}\langle m \rangle \mid [x = n]s) \mid \bar{s}$. Then $E[P] \longrightarrow \longrightarrow P'$
883 with $P' \not\Downarrow_{\bar{s}}$. This implies that $E[Q] \Longrightarrow Q'$ with $P' \cong_{\text{Arn}}^e Q'$. As $Q' \not\Downarrow_{\bar{s}}$ we have
884 $Q \xrightarrow{\bar{\ell}\langle n \rangle[m]} Q'$.
- 885 6. When $\alpha = (\nu n)\bar{\ell}\langle n \rangle[m]$, we take
886 $E \stackrel{\text{def}}{=} [] \mid \ell(x).(\bar{\ell}\langle m \rangle \mid s \mid \bar{s}'\langle x \rangle \mid \prod_{m \in F}[x = m]\bar{s}) \mid \bar{s}$. Then
887 $E[P] \longrightarrow \longrightarrow (\nu n)(P' \mid \prod_{m \in F}[n = m]\bar{s} \mid \bar{s}'\langle n \rangle)$. This implies $E[Q] \Longrightarrow Q''$ with
888 $(\nu n)(P' \mid \prod_{m \in F}[n = m]\bar{s} \mid \bar{s}'\langle n \rangle) \cong_{\text{Arn}}^e Q''$. As $Q'' \not\Downarrow_{\bar{s}}$, we necessarily have
889 $Q'' \equiv (\nu n)(Q' \mid \prod_{m \in F}[n = m]\bar{s} \mid \bar{s}'\langle n \rangle)$. By Lemmas 36 and 37, this means $P' \cong_{\text{Arn}}^e Q'$.
890 But then $Q \xrightarrow{(\nu n)\bar{\ell}\langle n \rangle[m]} Q'$ so we are done.

891

892 **B.5 Proofs about \approx_{ip}**

893 We show soundness of \approx_{ip} -bisimulation up to store with respect to \approx_{ip} -bisimilarity, and of
894 \approx_{ip} -bisimilarity with respect to reference bisimilarity.

Proof of Proposition 25. We show that

$$\mathcal{R}' \stackrel{\text{def}}{=} \{P \mid M_s, Q \mid M_s \mid P \mathcal{R} Q \text{ for any } M_s\}$$

895 is an \approx_{ip} -bisimulation.

896 If $P \mid M_s \mathcal{R} Q \mid M_s$ and $P \mid M_s \xrightarrow{\mu} \tilde{P}$, we distinguish the sub-processes of \tilde{P} that have
897 changed:

- 898 1. If $P \mid M_s \xrightarrow{\mu} P' \mid M_s$, then $P \xrightarrow{\mu} P'$, and $\text{ok}'(\Delta \cup \Delta', \mathcal{R}, P', Q, \mu)$. We show by induction on
899 the proof of $\text{ok}'(\Delta \cup \Delta', \mathcal{R}, P', Q, \mu)$ that $\text{ok}((\Delta \uplus \Delta_s) \cup (\Delta' \uplus \Delta_s), \mathcal{R}', P' \mid M_s, Q \mid M_s, \mu)$.
900 First note that $(\Delta \uplus \Delta_s) \cup (\Delta' \uplus \Delta_s) = (\Delta \cup \Delta') \uplus \Delta_s$. In short, we prove that
901 $\text{ok}'(\Delta, \mathcal{R}, P', Q, \mu)$ implies $\text{ok}(\Delta \uplus \Delta_s, \mathcal{R}', P' \mid M_s, Q \mid M_s, \mu)$.
 - 902 – (Base-Up) $P' = P'' \mid M_t, Q \xrightarrow{\mu} Q'' \mid M_t$ (or $Q \mid \bar{n}\langle m \rangle \Rightarrow Q'' \mid M_t$ for $\mu = n\langle m \rangle$) and
903 $P'' \mathcal{R} Q''$. Then $P'' \mid M_t \mid M_s \mathcal{R}' Q'' \mid M_t \mid M_s$ and $Q \mid M_s \xrightarrow{\mu} Q'' \mid M_t \mid M_s$, so we can
904 conclude with rule **Base**.
 - 905 – (Ext) We use an induction on the size of s .
 - 906 – If s is empty, then $\ell \notin \Delta \uplus \Delta_s$, and we can apply rule **Ext**.
 - 907 – If $\ell \notin \Delta \uplus \Delta_s$, we can apply rule **Ext** as before. Otherwise, $M_s = \bar{\ell}\langle m \rangle \mid M_{s'}$ for
908 some m, s' . Moreover, we know that $\text{ok}'((\Delta, \ell), \mathcal{R}, P' \mid \bar{\ell}\langle m \rangle, Q \mid \bar{\ell}\langle m \rangle, \mu)$. Thus, by
909 induction, $\text{ok}((\Delta, \ell \uplus \Delta_{s'}), \mathcal{R}', P' \mid \bar{\ell}\langle m \rangle \mid M_{s'}, Q \mid \bar{\ell}\langle m \rangle \mid M_{s'}, \mu)$.
- 910 2. If $P \mid M_s \xrightarrow{\tau} P' \mid M_{s'}$, then there exists an input action $\mu' = \ell\langle m \rangle$ such that $P \xrightarrow{\mu'} P'$, and
911 $\text{ok}'(\Delta \cup \Delta', \mathcal{R}, P', Q, \mu)$. We show by induction on the proof of $\text{ok}'(\Delta \cup \Delta', \mathcal{R}, P', Q, \mu)$,
912 that $\text{ok}((\Delta \uplus \Delta_s) \cup (\Delta' \uplus \Delta_{s'}), \mathcal{R}', P' \mid M_{s'}, Q \mid M_s, \mu)$. First note that $M_s \equiv \bar{\ell}\langle m \rangle \mid M_{s'}$
913 and $\Delta \uplus \Delta_s = \Delta' \uplus \Delta_{s'} = (\Delta \cup \Delta') \uplus \Delta_{s'}$. In short, we prove that $\text{ok}'(\Delta, \mathcal{R}, P', Q, \mu)$
914 implies $\text{ok}(\Delta \uplus \Delta_{s'}, \mathcal{R}', P' \mid M_{s'}, Q \mid M_s, \mu)$.
 - 915 – (Base-Up) $P' = P'' \mid M_t$ and $Q \mid \bar{\ell}\langle m \rangle \Rightarrow Q'' \mid M_t$, and $P'' \mathcal{R} Q''$.
916 Then $P'' \mid M_t \mid M_{s'} \mathcal{R}' Q'' \mid M_t \mid M_{s'}$ and $Q \mid M_s \equiv Q \mid \bar{\ell}\langle m \rangle \mid M_{s'} \xrightarrow{\tau} Q'' \mid M_t \mid M_{s'}$,
917 so we can conclude with rule **Base**.
 - 918 – (Ext) We use ℓ' for the name used in that rule here. We use an induction on the size
919 of s' .
 - 920 – If s' is empty, then $\ell \notin \Delta \uplus \Delta_{s'}$, and we can apply rule **Ext**.
 - 921 – If $\ell \notin \Delta \uplus \Delta_{s'}$, we can apply rule **Ext** as before. Otherwise, $M_{s'} = \bar{\ell'}\langle m' \rangle \mid M_{t'}$ for
922 some m, t' . Moreover, we know that $\text{ok}'((\Delta, \ell'), \mathcal{R}, P' \mid \bar{\ell'}\langle m' \rangle, Q \mid \bar{\ell'}\langle m' \rangle, \mu)$. Thus,
923 by induction, $\text{ok}((\Delta, \ell' \uplus \Delta_{t'}), \mathcal{R}, P' \mid \bar{\ell'}\langle m' \rangle \mid M_{t'}, Q \mid \bar{\ell'}\langle m' \rangle \mid M_{t'}, \mu)$.
- 924 3. If $P \mid M_s \xrightarrow{\mu} P \mid M_{s'}$, then μ is an output and $Q \mid M_s \xrightarrow{\mu} Q \mid M_{s'}$ so we can apply rule
925 **Base**.

926 ◀

927 ▶ **Corollary 38.** *As \approx is an \approx -bisimulation up to store, it is closed by parallel composition*
928 *of M_s .*

929 ▶ **Lemma 39.** *For any $\Delta \vdash P, Q$ and $\ell \notin \text{frn}(P) \cup \text{frn}(Q)$, and for all $m, P \mid \bar{\ell}\langle m \rangle \approx_{\text{ip}} Q \mid$
930 $\bar{\ell}\langle m \rangle$ implies $P \approx_{\text{ip}} Q$.*

931 This is true in particular for complete processes P, Q and any $\ell \notin \Delta$.

31:26 On the Representation of References in the pi-calculus

932 **Proof.** First notice that $P \mid \bar{\ell}\langle m \rangle \approx_{\text{ip}} Q \mid \bar{\ell}\langle m \rangle$ iff $P \mid \bar{\ell}'\langle m \rangle \approx_{\text{ip}} Q \mid \bar{\ell}'\langle m \rangle$ for any ℓ' fresh.
 933 We show that $\{(P, Q) \text{ s.t. } P \mid \bar{\ell}\langle m \rangle \approx_{\text{ip}} Q \mid \bar{\ell}\langle m \rangle \text{ for any fresh } \ell \text{ and any } m\}$ is an \approx_{ip} -
 934 bisimulation.
 935 When $P \xrightarrow{\mu} P'$, we distinguish if ℓ appears in μ :
 936 ■ If $\ell \notin \mu$, then $P \mid \bar{\ell}\langle m \rangle \xrightarrow{\mu} P' \mid \bar{\ell}\langle m \rangle$ and $\text{ok}((\Delta \cup \Delta', \ell), \approx_{\text{ip}}, P' \mid \bar{\ell}\langle m \rangle, Q \mid \bar{\ell}\langle m \rangle, \mu)$. We
 937 reason by induction on this predicate.
 938 ■ (Base) Then $Q \mid \bar{\ell}\langle m \rangle \xrightarrow{\mu} Q' \mid \bar{\ell}\langle m \rangle$ and $Q \xrightarrow{\mu} Q'$. Thus we conclude with rule **Base**.
 939 ■ (Ext) If $\ell' \notin \Delta, \ell$, then we can apply rule **Ext**.
 940 ■ If $\ell \in \mu$, then we consider $P \mid \bar{\ell}'\langle m \rangle$ and $Q \mid \bar{\ell}'\langle m \rangle$ with ℓ' fresh and $\ell' \neq \ell$, and do the
 941 same proof.

942

943 A consequence of this lemma is that to prove $P \approx_{\text{ip}} Q$, we may assume that rule **Ext** is never
 944 used with ℓ fresh.

945 **Proof of Proposition 26.** \approx is closed by allocation by Corollary 38.

946 For any P, Q complete:

947 ■ If $P \approx_{\text{ip}} Q$ and $P \xrightarrow{\mu} P'$, then by Lemma 39, we know $\text{ok}(\Delta, \approx_{\text{ip}}, P', Q, \mu)$ using rule **Base**,
 948 so $Q \xrightarrow{\mu} Q'$ and $P' \approx_{\text{ip}} Q'$.
 949 ■ If $P \xrightarrow{\bar{\ell}\langle n \rangle[m]} P'$ (resp. $(\nu n)\bar{\ell}\langle n \rangle[m]$), then as before but for $\mu = \bar{\ell}\langle n \rangle$ (resp. $\mu = (\nu n)\bar{\ell}\langle n \rangle$),
 950 we have $P \xrightarrow{\mu} P''$ and $Q \xrightarrow{\mu} Q''$ with $P'' \approx_{\text{ip}} Q''$, and $P' = P'' \mid \bar{\ell}\langle m \rangle$. But then we have
 951 $Q \xrightarrow{\bar{\ell}\langle n \rangle[m]} Q'$ (resp. $(\nu n)\bar{\ell}\langle n \rangle[m]$) with $Q' = Q'' \mid \bar{\ell}\langle m \rangle$ and $P' \approx_{\text{ip}} Q'$ so we are done.
 952