On the Representation of References in the Pi-Calculus

Daniel Hirschkoff
ENS de Lyon, France

Enguerrand Prebet
ENS de Lyon, France

Davide Sangiorgi
Università di Bologna, Italy
INRIA, Sophia Antipolis, 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 $\pi^{\mathtt{ref}}$, an extension of $A\pi$ with references and operators to manipulate them, and illustrate examples of the subtleties of behavioural equivalence in $\pi^{\mathtt{ref}}$. We then consider a translation of $\pi^{\mathtt{ref}}$ into $A\pi$. References of $\pi^{\mathtt{ref}}$ 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 → Semantics and reasoning

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

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.34

Related Version The proofs of most of the results in this paper are presented in a full version of this paper, available at `https://hal.archives-ouvertes.fr/hal-02895654`.

Funding Hirschkoff 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" (ASPRA, 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., [25, 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 make it unclear if and how interesting properties of imperative languages can be proved via a translation into the π-calculus.

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

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

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

$$\overline{n}\langle m\rangle \mid Q \Rightarrow Q' \qquad (*)$$

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

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

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

- we study the properties of this translation;
- we establish proof techniques on $A\pi$ to reason about references.

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

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

We define a type system in $A\pi$ to capture the intended pattern of usage of names that represent references, called *reference names*, in particular the property that there is always a unique output message available at these names. The type system has linearity features similar to π-calculus type systems for locks [13] or for receptiveness [22].

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

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

We establish an operational correspondence between the behaviour of a process in $\pi^{\mathtt{ref}}$ and its encoding in $A\pi$, and from this we establish full abstraction of the translation of $\pi^{\mathtt{ref}}$ into $A\pi$ with respect to both barbed equivalence and barbed congruence in the two calculi. We then 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. This bisimilarity is similar to, but not the same as, asynchronous bisimilarity. For instance, it is defined on "reference-closed" processes (intuitively, processes in which all references are allocated); therefore inputs on reference names from the tested processes are not visible (because such inputs are supposed to consume the unique output message at that reference that is present in the tested processes). The output clause of bisimilarity on reference names is also different, as we have to make sure that the observer respects the pattern of usage for reference names; thus the observer consuming the output message on a reference name ℓ should immediately re-install an output on ℓ.

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

Related Work. The classic encoding of references in the π-calculus [16] follows their encoding into CCS [15]: a reference is a stateful recursive process, which may be interrogated using two names, one for read operations, the other for write operations. Properties of this encoding have been explored [20], comparing the π-calculus to *Concurrent Idealised Algol* [3], an extension of Idealised Algol [19] with shared variables concurrency. The encoding has been shown to be sound but not complete.

Many works have studied the effect of type systems on behavioural equivalence, formalised using both barbed congruence and labelled bisimilarity. (See the references in the books [24, 7]). To our knowledge, no such study has been done regarding the discipline for reference names which we use in this work. This discipline bears similarities with receptiveness [22], which is also related to the results in [23, 14]. We can also remark that our notion of complete processes is reminiscent of the notion of catalysers used by Dezani et al. [5] in session types to enforce progress.

Section 5 discusses further related work.

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

The proofs of most of the results in this work are presented in a full version of this paper [9].

2 Asynchronous Processes Accessing References: $\pi^{\mathtt{ref}}$

In this section, we introduce $\pi^{\mathtt{ref}}$, the asynchronous π-calculus extended with primitives to interact with memory locations.

2.1 Syntax and Semantics

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

$$\begin{aligned} P \quad ::= \quad & \mathbf{0} \mid a(x).\,P \mid \overline{a}\langle n\rangle \mid !P \mid P_1 \mid P_2 \mid (\boldsymbol{\nu} a)P \mid [n = m]P \\ & \mid (\boldsymbol{\nu}\ell = n)P \mid \ell \triangleleft n.\,P \mid \ell \triangleright (x).\,P \mid \ell \bowtie n(x).\,P \end{aligned}$$

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

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

$$\texttt{R-Equiv:}\ \frac{P \equiv P' \qquad P' \longrightarrow Q' \qquad Q' \equiv Q}{P \longrightarrow Q} \qquad \texttt{R-Ctxt:}\ \frac{P \longrightarrow P'}{E[P] \longrightarrow E[Q]}$$

$$\texttt{R-Comm:}\ \frac{}{a(x).\,P \mid \overline{a}\langle n\rangle \longrightarrow P\{n/x\}}$$

$$\texttt{R-Read:}\ \frac{\ell, n \notin \mathrm{br}(\boldsymbol{\nu}\widetilde{L})}{(\boldsymbol{\nu}\ell = n)(\boldsymbol{\nu}\widetilde{L})(\ell \triangleright (x).\,P \mid Q) \longrightarrow (\boldsymbol{\nu}\ell = n)(\boldsymbol{\nu}\widetilde{L})(P\{n/x\} \mid Q)}$$

$$\texttt{R-Write:}\ \frac{\ell, n \notin \mathrm{br}(\boldsymbol{\nu}\widetilde{L})}{(\boldsymbol{\nu}\ell = m)(\boldsymbol{\nu}\widetilde{L})(\ell \triangleleft n.\,P \mid Q) \longrightarrow (\boldsymbol{\nu}\ell = n)(\boldsymbol{\nu}\widetilde{L})(P \mid Q)}$$

$$\texttt{R-Swap:}\ \frac{\ell, n, m \notin \mathrm{br}(\boldsymbol{\nu}\widetilde{L})}{(\boldsymbol{\nu}\ell = m)(\boldsymbol{\nu}\widetilde{L})(\ell \bowtie n(x).\,P \mid Q) \longrightarrow (\boldsymbol{\nu}\ell = n)(\boldsymbol{\nu}\widetilde{L})(P\{m/x\} \mid Q)}$$

Figure 1 $\pi^{\mathtt{ref}}$, reduction relation.

We assume the calculus is simply-typed. Any basic type system for the π-calculus would do. In this paper, we assume Milner's *sorting*: names and references are partitioned into a collection of *types* (or sorts). Name types contain names, and reference types contain references. Then a sorting function maps types onto types. If a name type s is mapped onto a type t, this means that names in s may only carry, or contain, objects in t; if s is a reference type then only objects of type t may be stored in s. We shall assume that there is a sorting system under which all processes we manipulate are well-typed. For simplicity we use simple types; e.g., the sorting is non-recursive (meaning that the graph that represents the sorting function, in which the nodes are the types, does not contain cycles). In the remainder we assume that all objects (processes, contexts, actions, etc.) respect a given sorting.

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

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

$$E \quad ::= \quad [\,] \mid E \mid P \mid (\boldsymbol{\nu}a)E \mid (\boldsymbol{\nu}\ell = n)E \ .$$

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

As usual in concurrent calculi, the reference behavioural equivalence will be barbed congruence (in its variant sometimes called *reduction-closed barbed congruence*), a form of bisimulation on reduction that uses closure under contexts and simple observables. In the

context closure, however, we make sure that all references mentioned in the tested process have been allocated. As often in π-calculi, we also consider *barbed equivalence*, that uses only active contexts.

P exhibits a barb at a (so a is in `Names`), written $P \downarrow_{\overline{a}}$, if $P \equiv (\boldsymbol{\nu}\widetilde{b})(\boldsymbol{\nu}\widetilde{L})(\overline{a}\langle m\rangle \mid P')$ with $a \notin \widetilde{b}$. We write $P \Downarrow_{\overline{a}}$ if $P \Longrightarrow P_1$ and $P_1 \downarrow_{\overline{a}}$ for some P_1.

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

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

▶ **Definition 2** (Barbed congruence and equivalence in $\pi^{\mathtt{ref}}$)**.** Barbed congruence *is the largest symmetric relation $\cong_{\mathrm{ref}}$ in $\pi^{\mathtt{ref}}$ such that whenever $P\ \mathcal{R}\ Q$ then P, Q are: closed under reductions if P, Q are reference-closed; preserved by the contexts that are closing on references for P, Q; compatible on barbs if P, Q are reference-closed.* Barbed equivalence, $\cong^{\mathrm{e}}_{\mathrm{ref}}$, *is defined in the same way, but using active contexts in place of all contexts.*

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

$$\ell \triangleright (x).\mathrm{P} \cong_{\mathrm{ref}} \mathrm{P}, \tag{1}$$

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

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

2.2 Behavioural Equivalence in $\pi^{\mathtt{ref}}$: Examples

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

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

▶ **Example 3.** For any P, we have $P_1 \cong_{\mathrm{ref}} P_2$, for

$$P_1 \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu}\ell = a)(P \mid !\ell \triangleleft a \mid !\ell \triangleleft b) \qquad P_2 \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu}\ell = b)(P \mid !\ell \triangleleft a \mid !\ell \triangleleft b)$$

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

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

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

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

▶ **Example 5.** We have $P_s \not\cong^{\mathrm{e}}_{\mathrm{ref}} Q_s$, where

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

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

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

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

$$P_1 \stackrel{def}{=} (\boldsymbol{\nu} \ell_1 = 0, \ell_2 = 0)(P \mid (\boldsymbol{\nu} t)(\bar{t} \mid !t.\, \ell_1 \triangleleft 1.\, \ell_1 \triangleleft 0.\, \ell_2 \triangleleft 1.\, \ell_2 \triangleleft 0.\, \bar{t}))$$
$$P_2 \stackrel{def}{=} (\boldsymbol{\nu} \ell_1 = 0, \ell_2 = 0)(P \mid (\boldsymbol{\nu} t)(\bar{t} \mid !t.\, \ell_1 \triangleleft 1.\, \ell_2 \triangleleft 1.\, \ell_1 \triangleleft 0.\, \ell_2 \triangleleft 0.\, \bar{t}))$$

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, we see that if P_1 loops, the content of ℓ_1 and ℓ_2 will evolve thus: $(0,0) \to (1,0) \to (0,0) \to (0,1) \to (0,0)$, while for P_2 the loop is different: $(0,0) \to (1,0) \to (1,1) \to (0,1) \to (0,0)$.

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

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

3 Mapping $\pi^{\mathtt{ref}}$ onto the Asynchronous π-calculus

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

3.1 The Asynchronous π-calculus

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

$$P \quad ::= \quad \mathbf{0} \mid n(x).\,P \mid !P \mid \overline{n}\langle m\rangle \mid P_1 \mid P_2 \mid (\boldsymbol{\nu} n)P \mid [n=m]P$$

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

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

- *there is Q' such that $Q \stackrel{\hat{\mu}}{\Rightarrow} Q'$ and $P'\,\mathcal{R}\,Q'$;*
- *$\mu = n\langle m\rangle$ and there is Q' such that $Q \mid \overline{n}\langle m\rangle \Rightarrow Q'$ and $P'\,\mathcal{R}\,Q'$.*

Asynchronous bisimilarity, $\approx_{\mathrm{a}}$, is the largest asynchronous bisimulation.

▶ **Theorem 8** ([1])**.** *Relations $\cong_{\mathrm{a}}^{\mathrm{e}}$ and $\approx_{\mathrm{a}}$ coincide.*

3.2 Encoding $\pi^{\mathtt{ref}}$

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

The encoding $\mathcal{E}[\![\cdot]\!]$, from $\pi^{\mathtt{ref}}$ 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:

$$\mathcal{E}[\![(\boldsymbol{\nu}\ell = m).\,P]\!] \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell)(\overline{\ell}\langle m\rangle \mid \mathcal{E}[\![P]\!]) \qquad \mathcal{E}[\![\ell \triangleleft v.\,P]\!] \stackrel{\text{def}}{=} \ell(\_).\,(\overline{\ell}\langle v\rangle \mid \mathcal{E}[\![P]\!])$$

$$\mathcal{E}[\![\ell \triangleright (x).\,P]\!] \stackrel{\text{def}}{=} \ell(x).\,(\overline{\ell}\langle x\rangle \mid \mathcal{E}[\![P]\!]) \qquad \mathcal{E}[\![\ell \bowtie n(x).\,P]\!] \stackrel{\text{def}}{=} \ell(x).\,(\overline{\ell}\langle n\rangle \mid \mathcal{E}[\![P]\!])$$

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

3.3 Types and Behavioural Equivalences with Reference Names

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

$$\texttt{TNil}\ \frac{}{\emptyset \vdash \mathbf{0}} \qquad \texttt{TOut}\ \frac{}{\emptyset \vdash \overline{a}\langle m\rangle} \qquad \texttt{TInp}\ \frac{\emptyset \vdash P}{\emptyset \vdash a(x).\,P} \qquad \texttt{TRep}\ \frac{\emptyset \vdash P}{\emptyset \vdash\, !P}$$

$$\texttt{TPar}\ \frac{\Delta_1 \vdash P \qquad \Delta_2 \vdash Q}{\Delta_1 \uplus \Delta_2 \vdash P \mid Q} \qquad \texttt{TResN}\ \frac{\Delta \vdash P}{\Delta \vdash (\boldsymbol{\nu} a)P} \qquad \texttt{TResR}\ \frac{\Delta, \ell \vdash P}{\Delta \vdash (\boldsymbol{\nu} \ell)P}$$

$$\texttt{TRefO}\ \frac{}{\ell \vdash \overline{\ell}\langle m\rangle} \qquad \texttt{TRefI}\ \frac{\ell \vdash P}{\emptyset \vdash \ell(x).\,P}$$

Figure 2 Typing conditions for reference names in $A\pi$ processes.

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

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

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

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

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

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

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

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

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

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

The following standard property relies on the standard LTS for $A\pi$, which is given in Appendix B.

▶ **Proposition 10** (Subject reduction). *If $\Delta \vdash P$ and $P \xrightarrow{\mu} P'$, then*
1. *if $\mu = \tau$, $\mu = \overline{a}\langle m\rangle$, $\mu = a\langle m\rangle$ or $\mu = (\boldsymbol{\nu}b)\overline{a}\langle b\rangle$, then $\Delta \vdash P'$.*
2. *if $\mu = (\boldsymbol{\nu}\ell)\overline{a}\langle\ell\rangle$ then $\Delta, \ell \vdash P'$.*
3. *if $\mu = \ell\langle m\rangle$ and $\ell \notin \Delta$, then $\Delta, \ell \vdash P'$*
4. *if $\ell \notin \Delta$, then $\Delta, \ell \vdash P \mid \overline{\ell}\langle m\rangle$.*
5. *if $\mu = \overline{\ell}\langle m\rangle$ or $\mu = (\boldsymbol{\nu}b)\overline{\ell}\langle b\rangle$, then $\Delta - \ell \vdash P'$.*
6. *if $\mu = (\boldsymbol{\nu}\ell')\overline{\ell}\langle\ell'\rangle$, then $(\Delta - \ell), \ell' \vdash P'$.*

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

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

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

▶ **Definition 11** (Open references and complete processes). *The* open references *of P such that $\Delta \vdash P$ are the names in $\mathtt{frn}(P) \setminus \Delta$; similarly the open references of processes $P_1, \ldots, P_n$ is the union of the open references of the P_i's. P is* complete *if it contains no open reference. $\mathtt{frn}(P) \subseteq \Delta$ and $\Delta \vdash P$, for some Δ.*

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

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

▶ **Lemma 12.** *P is complete iff $\emptyset \vdash (\boldsymbol{\nu}\widetilde{n})P$ where $\widetilde{n} \stackrel{def}{=} \mathtt{frn}(P)$.*

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

$$E \quad ::= \quad [\,] \;\big|\; E \mid P \;\big|\; (\boldsymbol{\nu}n)E \ .$$

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

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

▶ **Lemma 14.** *For all compatible P, Q, $P \cong^{\mathrm{e}}_{\mathrm{a}} Q$ (and hence also $P \approx_{\mathrm{a}} Q$) implies $P \cong^{\mathrm{e}}_{\mathrm{Arn}} Q$.*

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 $\pi^{\mathtt{ref}}$. And similarly for contexts.

▶ **Lemma 16.** *If $\emptyset \vdash P$, then there exists R in $\pi^{\mathtt{ref}}$ such that $\mathcal{E}[\![R]\!] \approx_{\mathrm{a}} P$.*

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

▶ **Theorem 17** (Full abstraction). *For any P, Q in $\pi^{\mathtt{ref}}$: $P \cong_{\mathrm{ref}} Q$ iff $\mathcal{E}[\![P]\!] \cong_{\mathrm{Arn}} \mathcal{E}[\![Q]\!]$; and similarly $P \cong^{\mathrm{e}}_{\mathrm{ref}} Q$ iff $\mathcal{E}[\![P]\!] \cong^{\mathrm{e}}_{\mathrm{Arn}} \mathcal{E}[\![Q]\!]$.*

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 $\overline{\ell}\langle n\rangle$, then, following the discipline on reference names, he/she has to immediately provide another such output message, say $\overline{\ell}\langle m\rangle$. This is formalised using transition notations such as $P \xrightarrow{\overline{\ell}\langle n\rangle[m]} P'$, 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 $P \mid \overline{\ell}\langle n\rangle\ \mathcal{R}\ Q \mid \overline{\ell}\langle n\rangle$ for any $\overline{\ell}\langle n\rangle$ such that $P \mid \overline{\ell}\langle n\rangle$ and $Q \mid \overline{\ell}\langle n\rangle$ are well-typed. We write $P \xrightarrow{\overline{\ell}\langle n\rangle[m]} P'$ if $P \xrightarrow{\overline{\ell}\langle n\rangle} P''$ and $P' = \overline{\ell}\langle m\rangle \mid P''$, for some P''; similarly for $P \xrightarrow{(\boldsymbol{\nu} n)\overline{\ell}\langle n\rangle[m]} P'$. Then, as usual, $P \xRightarrow{\overline{\ell}\langle n\rangle[m]} P'$ holds if $P \Rightarrow P'' \xrightarrow{\overline{\ell}\langle n\rangle[m]} P''' \Rightarrow P'$ for some P'', P''', and similarly for $P \xRightarrow{(\boldsymbol{\nu} n)\overline{\ell}\langle n\rangle[m]} P'$.

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

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*

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

Reference bisimilarity, written $\approx$, is the largest reference bisimulation.

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

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

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

▶ **Theorem 21** (Labelled characterisation). *$P \approx Q$ iff $P \cong^{\mathrm{e}}_{\mathrm{Arn}} Q$.*

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

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

$$\textit{Base}\ \frac{\begin{cases} Q \mid \overline{n}\langle m\rangle \Rightarrow Q' & \text{for } \mu = n\langle m\rangle \\ Q \stackrel{\mu}{\Rightarrow} Q' & \text{otherwise} \end{cases} \qquad P'\ \mathcal{R}\ Q'}{\mathtt{ok}(\Delta, \mathcal{R}, P', Q, \mu)}$$

$$\textit{Ext}\ \frac{\ell \notin \Delta \qquad \forall\ m\ :\ \ \mathtt{ok}((\Delta, \ell), \mathcal{R}, P' \mid \overline{\ell}\langle m\rangle, Q \mid \overline{\ell}\langle m\rangle, \mu)}{\mathtt{ok}(\Delta, \mathcal{R}, P', Q, \mu)}$$

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

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

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

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

▶ **Definition 24** ($\approx_{\text{ip}}$-bisimulation up to store). *An $\approx_{\text{ip}}$-bisimulation* up to store *is defined like $\approx_{\text{ip}}$-bisimulation (Definition 23), using a predicate* `ok'`$(\Delta\cup\Delta',\mathcal{R},P',Q,\mu)$. *This predicate is defined by a modified version of rule* `Ext` *where ok′ is used instead of ok, both in the premise and in the conclusion, and the following modified version of the* `Base` *rule:*

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

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

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

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

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

4.2 Examples

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

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

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

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

We can derive $P_1 \approx_{\text{a}} P_2$, using the singleton relation $\mathcal{R} \stackrel{def}{=} \{(P_1,P_2)\}$, and showing that $\mathcal{R}$ is an asynchronous bisimilarity up-to context and structural congruence [18] (this known 'up-to' technique allows one to remove additional processes created from the replications after a transition). We can then conclude by Lemma 14.

We now consider Examples 3 and 4 from Section 2.

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

$$R_1 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell)\big(\overline{\ell}\langle a\rangle \mid \mathcal{E}[\![P]\!] \mid !\ell(\_).\,\overline{\ell}\langle a\rangle \mid !\ell(\_).\,\overline{\ell}\langle b\rangle\big)$$
$$R_2 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell)\big(\overline{\ell}\langle b\rangle \mid \mathcal{E}[\![P]\!] \mid !\ell(\_).\,\overline{\ell}\langle a\rangle \mid !\ell(\_).\,\overline{\ell}\langle b\rangle\big)$$

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

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

$$R_1 \stackrel{\text{def}}{=} \ell(\_).(\overline{\ell}\langle b\rangle \mid \mathcal{E}[\![P]\!]) \mid !\ell(\_).\overline{\ell}\langle b\rangle \mid !\ell(x).(\overline{\ell}\langle x\rangle \mid \ell(\_).\overline{\ell}\langle x\rangle)$$
$$R_2 \stackrel{\text{def}}{=} \mathcal{E}[\![P]\!] \mid !\ell(\_).\overline{\ell}\langle b\rangle \mid !\ell(x).(\overline{\ell}\langle x\rangle \mid \ell(\_).\overline{\ell}\langle x\rangle)$$

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

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

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

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

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

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

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

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

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

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

$$\begin{aligned}\mathcal{R}_1 \stackrel{\text{def}}{=}\ & \{(\ell(x).(P_0 \mid \overline{\ell}\langle x\rangle), Q_0)\}\\ & \cup \{(\ell(x).(P_0 \mid \overline{\ell}\langle x\rangle) \mid \overline{\ell}\langle\ell'\rangle \mid \overline{\ell'}\langle m\rangle, Q_0 \mid \overline{\ell}\langle\ell'\rangle \mid \overline{\ell'}\langle m\rangle) \;\big|\; \text{for any } m\}\\ & \cup \{(\ell(x).(P_0 \mid \overline{\ell}\langle x\rangle) \mid \overline{\ell}\langle\ell'\rangle \mid \overline{\ell'}\langle m\rangle \mid M_s, Q_0 \mid \overline{\ell}\langle\ell'\rangle \mid \overline{\ell'}\langle m\rangle \mid M_s) \;\big|\; \text{for any } m, M_s\}\\ & \cup \{P \mid \overline{\ell}\langle\ell'\rangle \mid \overline{\ell'}\langle m\rangle \mid M_s, Q \mid \overline{\ell}\langle\ell'\rangle \mid \overline{\ell'}\langle m\rangle \mid M_s) \;\big|\; \text{for any } m, M_s, \text{with } P\ \mathcal{R}\ Q\}\end{aligned}$$

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.

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

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

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

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

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

5 Future work

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

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

It would be interesting to see if the labelled bisimilarities we have considered, whose bisimulation clauses are different from those of ordinary bisimilarity, can be recovered in an abstract setting, e.g., using coalgebras [12, 2, 21]. This would be particularly interesting for $\approx_{\mathrm{ip}}$-bisimulation, whose definition involves a mixture of induction and coinduction.

Equivalences for higher-order languages with state are known to be hard to establish. Various approaches exist, from Kripke logical relations to trace semantics and game semantics [10, 11, 17, 4]. It would be interesting to compare the proof techniques offered by these approaches with those shown in this paper, and developments of them. More generally, more experimentation is needed to test the bisimilarities proposed in this paper and the associated proof techniques, on examples from high-level languages that include higher-order features, mutable state, and concurrency.

References

1 R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous pi-calculus. *Theor. Comput. Sci.*, 195(2):291–324, 1998.

2 F. Bonchi, D. Petrişan, D. Pous, and J. Rot. A general account of coinduction up-to. *Acta Informatica*, pages 1–64, 2016.

3 S. D. Brookes. The Essence of Parallel Algol. *Inf. Comput.*, 179(1):118–149, 2002.

4 S. Castellan, P. Clairambault, J. Hayman, and G. Winskel. Non-angelic concurrent game semantics. In *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018*, pages 3–19, 2018.

5 M. Coppo, M. Dezani-Ciancaglini, N. Yoshida, and L. Padovani. Global progress for dynamically interleaved multiparty sessions. *Math. Struct. Comput. Sci.*, 26(2):238–302, 2016.

6 M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully abstract model for the π-calculus. *Inf. Comput.*, 179(1):76–117, 2002.

7 M. Hennessy. *A distributed Pi-calculus*. Cambridge University Press, 2007.

8 M. P. Herlihy. Impossibility and universality results for wait-free synchronization. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, PODC '88, pages 276–290, 1988.

9 D. Hirschkoff, E. Prebet, and D. Sangiorgi. Online appendix to this paper. available from `https://hal.archives-ouvertes.fr/hal-02895654`, 2020.

10 C.-K. Hur, D. Dreyer, G. Neis, and V. Vafeiadis. The marriage of bisimulations and kripke logical relations. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*, pages 59–72, 2012.

11 G. Jaber and N. Tzevelekos. Trace semantics for polymorphic references. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16*, pages 585–594, 2016.

12 B. Jacobs. Introduction to coalgebra. towards mathematics of states and observations. Draft, 2014.

13 N. Kobayashi. A partially deadlock-free typed process calculus. *Transactions on Programming Languages and Systems*, 20(2):436–482, 1998. A preliminary version in *12th Lics Conf.* IEEE Computer Society Press 128–139, 1997.

14 M. Merro, J. Kleist, and U. Nestmann. Mobile objects as mobile processes. *Information and Computation*, 177(2):195–241, 2002.

15 R. Milner. *Communication and concurrency.* PHI Series in computer science. Prentice Hall, 1989.

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

17 A. S. Murawski and N. Tzevelekos. Full abstraction for reduced ML. *Ann. Pure Appl. Logic*, 164(11):1118–1143, 2013.

18 D. Pous and D. Sangiorgi. *Advanced Topics in Bisimulation and Coinduction (D. Sangiorgi and J. Rutten editors)*, chapter Enhancements of the coinductive proof method. Cambridge University Press, 2011.

19 J. C. Reynolds. The essence of ALGOL. In *Algorithmic Languages*, pages 345–372. North-Holland, 1981.

20 C. Röckl and D. Sangiorgi. A pi-calculus process semantics of concurrent idealised ALGOL. In *Foundations of Software Science and Computation Structure, Second International Conference, FoSSaCS'99*, volume 1578 of *Lecture Notes in Computer Science*, pages 306–321. Springer, 1999.

21 J. Rot, F. Bonchi, M. M. Bonsangue, D. Pous, J. Rutten, and A. Silva. Enhanced coalgebraic bisimulation. *Math. Struct. Comput. Sci.*, 27(7):1236–1264, 2017.

22 D. Sangiorgi. The name discipline of uniform receptiveness. *Theor. Comput. Sci.*, 221(1-2):457–493, 1999.

23 D. Sangiorgi. Typed pi-calculus at work: A Correctness Proof of Jones's Parallelisation Transformation on Concurrent Objects. *TAPOS*, 5(1):25–33, 1999.

24 D. Sangiorgi and D. Walker. *The pi-calculus: a Theory of Mobile Processes*. Cambridge university press, 2003.

25 I. Stark. A fully abstract domain model for the pi-calculus. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 36–42. IEEE Computer Society, 1996.

A Additional Material for the Examples in Section 2.2

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

1. $(\boldsymbol{\nu}\ell = z)(\boldsymbol{\nu}t)\ell \triangleleft a.\,(\overline{t} \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)))$
2. $(\boldsymbol{\nu}\ell = a)(\boldsymbol{\nu}t)(\overline{t} \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c))) \mid c^n \mid \overline{c}^n$
3. $(\boldsymbol{\nu}\ell = a)(\boldsymbol{\nu}t)(\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)) \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c))) \mid c^n \mid \overline{c}^n$
4. $(\boldsymbol{\nu}\ell = b)(\boldsymbol{\nu}t)(\ell \triangleleft a.\,(\overline{t} \mid c) \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c))) \mid c^n \mid \overline{c}^{n+1}$.

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

We now consider the following context:

$$E_0 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell = z)([\,] \mid \ell \bowtie z(x).\,[x = b]s_0.\,s_1.\,(P_{11} \mid P_{12}) \mid \overline{s_0} \mid \overline{s_1})$$

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

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

At first $\overline{s_0}$ and $\overline{s_1}$ are the only observables, meaning $E_0[P_s] \downarrow_{\overline{s_0}}$ and $E_0[P_s] \downarrow_{\overline{s_1}}$, but then $E_0[P_s] \longrightarrow\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow (\boldsymbol{\nu}\ell = z)((\boldsymbol{\nu}t)(\overline{t} \mid !t.\,\ell \triangleleft a.\,(\overline{c} \mid \ell \triangleleft b.\,(\overline{t} \mid c))) \mid s_1.\,(P_{11} \mid P_{12}) \mid \overline{s_1}) \stackrel{\text{def}}{=} P'$ where the three reductions have been derived using rules `R-Write`, `R-Swap`, and `R-Comm` respectively. Finally, we have $P' \not\downarrow_{\overline{s_0}}$, whereas $P' \downarrow_{\overline{s_1}}$.

Thus, to avoid the observable $\overline{s_0}$, process $E_0[Q_s]$ must reduce to a process with b stored in ℓ before doing the swap in E_0. This implies that the swap is executed in a state that corresponds to case 4 above. So for any Q' with $E[Q_s] \Longrightarrow Q'$ and $Q' \not\downarrow_{\overline{s_0}}$ and $Q' \Downarrow_{\overline{s_1}}$, such process Q' has one of the following forms:

1. $Q'_1 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell = a)((\boldsymbol{\nu}t)(\overline{t} \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)) \mid c^n \mid \overline{c}^n) \mid s_1.\,(P_{11} \mid P_{12}) \mid \overline{s_1})$
2. $Q'_2 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell = a)((\boldsymbol{\nu}t)(\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)) \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)) \mid c^n \mid \overline{c}^n) \mid s_1.\,(P_{11} \mid P_{12}) \mid \overline{s_1})$
3. $Q'_3 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell = b)((\boldsymbol{\nu}t)(\ell \triangleleft a.\,(\overline{t} \mid c) \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)) \mid c^n \mid \overline{c}^{n+1}) \mid s_1.\,(P_{11} \mid P_{12}) \mid \overline{s_1})$
4. $Q'_4 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell = z)((\boldsymbol{\nu}t)(\ell \triangleleft a.\,(\overline{t} \mid c) \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)) \mid c^n \mid \overline{c}^{n+1}) \mid s_1.\,(P_{11} \mid P_{12}) \mid \overline{s_1})$

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

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

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

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

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

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

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

To prove their equivalence, we introduce the following processes:

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

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

$$
\begin{aligned}
P_1 = Q_1 &\stackrel{\text{def}}{=} \bar{t} \\
P_2 &\stackrel{\text{def}}{=} \ell_1(\_).(\overline{\ell_1}\langle 1\rangle \mid \ell_1(\_).(\overline{\ell_1}\langle 0\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 1\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t})))) \\
Q_2 &\stackrel{\text{def}}{=} \ell_1(\_).(\overline{\ell_1}\langle 1\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 1\rangle \mid \ell_1(\_).(\overline{\ell_1}\langle 0\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t})))) \\
P_3 &\stackrel{\text{def}}{=} \ell_1(\_).(\overline{\ell_1}\langle 0\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 1\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t}))) \\
Q_3 &\stackrel{\text{def}}{=} \ell_2(\_).(\overline{\ell_2}\langle 1\rangle \mid \ell_1(\_).(\overline{\ell_1}\langle 0\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t}))) \\
P_4 &\stackrel{\text{def}}{=} \ell_2(\_).(\overline{\ell_2}\langle 1\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t})) \\
Q_4 &\stackrel{\text{def}}{=} \ell_1(\_).(\overline{\ell_1}\langle 0\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t})) \\
P_5 = Q_5 &\stackrel{\text{def}}{=} \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t})
\end{aligned}
$$

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

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

$$
\begin{aligned}
\mathcal{R} \stackrel{\text{def}}{=} &\left\{ \begin{array}{c} (\overline{\ell_1}\langle n_1\rangle \mid (\boldsymbol{\nu} t)(P' \mid P_i),\ \overline{\ell_1}\langle n_1'\rangle \mid (\boldsymbol{\nu} t)(Q' \mid Q_j)) \\ \text{for any } n_1, n_1' \in \{0,1\}, i, j \end{array} \right\} \\
\cup &\left\{ \begin{array}{c} (\overline{\ell_2}\langle n_2\rangle \mid (\boldsymbol{\nu} t)(P' \mid P_i),\ \overline{\ell_2}\langle n_2'\rangle \mid (\boldsymbol{\nu} t)(Q' \mid Q_j)) \\ \text{for any } n_2, n_2' \in \{0,1\}, i, j \end{array} \right\} \\
\cup &\left\{ \begin{array}{c} (\overline{\ell_1}\langle n_1\rangle \mid \overline{\ell_2}\langle n_2\rangle \mid (\boldsymbol{\nu} t)(P' \mid P_i),\ \overline{\ell_1}\langle n_1'\rangle \mid \overline{\ell_2}\langle n_2'\rangle \mid (\boldsymbol{\nu} t)(Q' \mid Q_j)) \\ \text{for any } n_1, n_1', n_2, n_2' \in \{0,1\}, i, j \end{array} \right\}
\end{aligned}
$$

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

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

- If $P \xrightarrow{\tau} P_0$, then $P_0 \mathcal{R} Q$
- If $P \xrightarrow{\ell_i\langle n\rangle} P_0$, then $P_0 \mathcal{R} Q \mid \overline{\ell_i}\langle n\rangle$
- 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 use rule `Ext` (from Definition 22) to add the other location if $\Delta \neq \ell_1, \ell_2$. Then after at most 5 internal transitions (by cycling around the P_i or Q_j), we obtain a process Q_0 that can make the required transition $Q_0 \xrightarrow{\overline{\ell_i}\langle n\rangle} Q_0'$ with $P_0 \mathcal{R} Q_0'$.

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

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

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

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

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

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

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

$$B_i^j \stackrel{\text{def}}{=} r(x^j).\,\mathbf{0} \mid !r(x^j).\,t_i.\,\ell^j \bowtie x^j(y^j).\,(\overline{r}\langle y^j\rangle \mid \overline{t_i})$$
$$S_i^j \stackrel{\text{def}}{=} !t_i.\,\ell^j \triangleright (x^j).\,(\overline{t_i} \mid (\boldsymbol{\nu} r)(\overline{r}\langle x^j\rangle \mid B_i^j))$$

$$P \stackrel{\text{def}}{=} (\boldsymbol{\nu} t_1,t_2,t_3,t_4)\Big(\overline{t_1} \mid !t_1.\,\ell^1 \triangleleft 1.\,\overline{t_2} \mid S_1^1 \mid S_1^2 \mid !t_2.\,\ell^1 \triangleleft 0.\,\overline{t_3} \mid S_2^1 \mid S_2^2 \mid !t_3.\,\ell^2 \triangleleft 1.\,\overline{t_4} \mid S_3^1 \mid S_3^2 \mid !t_4.\,\ell^2 \triangleleft 0.\,\overline{t_1} \mid S_4^1 \mid S_4^2\Big)$$
$$Q \stackrel{\text{def}}{=} (\boldsymbol{\nu} t_1,t_2,t_3,t_4)\Big(\overline{t_1} \mid !t_1.\,\ell^1 \triangleleft 1.\,\overline{t_2} \mid S_1^1 \mid S_1^2 \mid !t_2.\,\ell^2 \triangleleft 1.\,\overline{t_3} \mid S_2^1 \mid S_2^2 \mid !t_3.\,\ell^1 \triangleleft 0.\,\overline{t_4} \mid S_3^1 \mid S_3^2 \mid !t_4.\,\ell^2 \triangleleft 0.\,\overline{t_1} \mid S_4^1 \mid S_4^2\Big)$$

We have $P \cong_{\text{ref}} Q$. If we take $E \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell^1 = 0)(\boldsymbol{\nu}\ell_2 = 0)[\,]$, we have $E[Q] \longrightarrow\!\!\!\!\longrightarrow (\boldsymbol{\nu}\ell^1 = 1)(\boldsymbol{\nu}\ell^2 = 1)Q'$ for some Q'. However, there is no sequence of reductions such that $E[P] \Longrightarrow (\boldsymbol{\nu}\ell^1 = 1)(\boldsymbol{\nu}\ell^2 = 1)P'$ for any P'.

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

Process S_i^j can only be activated when $\overline{t_i}$ is available. It then reads the content of ℓ_j to initialise a new buffer B_i^j.

Process B_i^j contains value x_i^j that is the object of $\overline{r}\langle x_i^j\rangle$. Process B_i^j can be stopped by making the communication with the first input on r, or can be used to swap its content with the content of ℓ^j. Note that this swap can only be done when $\overline{t_i}$ is available, so it cannot be used to desynchronise the content in ℓ_1, and ℓ_2.

B Operational Semantics of $A\pi$: Reduction and Labelled Transitions

Reduction

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

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

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

$$E \quad ::= \quad [\,] \mid E \mid P \mid (\boldsymbol{\nu} n)E \ .$$

Reduction is defined by the following rules:

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

$$\texttt{Inp:}\ \frac{}{n(x).\,P \xrightarrow{n\langle m\rangle} P\{m/x\}} \qquad \texttt{Out:}\ \frac{}{\overline{n}\langle m\rangle \xrightarrow{\overline{n}\langle m\rangle} \mathbf{0}}$$

$$\texttt{Open:}\ \frac{P \xrightarrow{\overline{n}\langle m\rangle} P'}{(\boldsymbol{\nu} m)P \xrightarrow{(\boldsymbol{\nu} m)\overline{n}\langle m\rangle} P'}\ \text{if } m \neq n \qquad \texttt{Rep:}\ \frac{P \mid !P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}$$

$$\texttt{Res:}\ \frac{P \xrightarrow{\mu} P'}{(\boldsymbol{\nu} n)P \xrightarrow{\mu} (\boldsymbol{\nu} n)P'}\ \text{if } n \notin \mu \qquad \texttt{Par:}\ \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ \text{if } \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$$

$$\texttt{Comm:}\ \frac{P \xrightarrow{n\langle m\rangle} P' \qquad Q \xrightarrow{\overline{n}\langle m\rangle} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\texttt{Close:}\ \frac{P \xrightarrow{n\langle m\rangle} P' \qquad Q \xrightarrow{(\boldsymbol{\nu} m)\overline{n}\langle m\rangle} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu} m)(P' \mid Q')}\ \text{if } m \notin \mathrm{fn}(P) \qquad \texttt{Match:}\ \frac{P \xrightarrow{\mu} P'}{[n = n]P \xrightarrow{\mu} P'}$$

Figure 3 Labelled Transition Semantics for $A\pi$.

Labelled Transition Semantics

Actions of the LTS are defined as follows:

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

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