Minimality and Separation Results on Asynchronous Mobile Processes

— Representability Theorems by Concurrent Combinators —

NOBUKO YOSHIDA

ABSTRACT. In [25, 26], we presented a theory of *concurrent combinators* for the asynchronous monadic π-calculus without match or summation operator [8, 22]. The system of concurrent combinators is based on a finite number of atoms and fixed interaction rules, but is as expressive as the original calculus, so that it can represent diverse interaction structures, including polyadic synchronous name passing [35] and input guarded summations [40]. The present paper shows that each of the five basic combinators introduced in [25] is indispensable to represent the whole computation, i.e. if one of the combinators is missing, we can no longer express the original calculus up to semantic equalities. Expressive power of several interesting subsystems of the asynchronous π-calculus is also measured by using appropriate subsets of the combinators and their variants. Finally as an application of the main result, we show there is no semantically sound encoding of the calculus into its proper subsystem under a certain condition.

1 Introduction

The calculi of mobile processes [34, 35, 38] have been studied as a mathematical basis of concurrent computing due to their surprising expressive power in spite of simple syntactic constructs. Since Milner, Parrow and Walker introduced the original system in [38], various variants of this calculus have been considered in many settings: a polyadic or monadic, synchronous or asynchronous π-calculus with or without match, mismatch, and summation operators. In sequential computation, the hierarchy of computable functions has been traditionally used to measure the expressive power of programming languages based on a rigid mathematical background. This notion is, however, too function-oriented to examine the whole expressiveness realisable in π-calculi. Consider the result in [34], which showed lazy and call-by-value λ-calculi can be simulated in an operationally correct way in monadic π-calculus without match or summation operator. The two λ-calculi are in the same computability hierarchy, but their encodings in π-calculus represent quite different communication protocols: computational behaviour in π-calculus is based on much finer interactions than functional one. The question then arises as to what are general methods to measure representability for π-calculi, which would also be applicable to other concurrency formalisms and programming languages.

One of the major ways to understand the expressiveness of π-calculi is to examine existence of reasonable encodings of high-level communication structures

into them. Specifically if we restrict our attention to the family of π-calculi, the problem is reducible to knowing whether an operation or a construct of some instance of π-calculi can be represented by its sub-calculus without the construct. If so, the added computational element can be regarded as just a "macro" or a "syntactic sugar". If not, then it is indispensable to describe the whole behaviour: we say that the additional construct *separates* the world with it from without it.

In the absence of match operator, one remarkable separation result on expressiveness was proved by Palamidessi [44]: "mixed summations" can*not* be embedded into any of π-calculi without them. Her result reinforces the intuitive understanding that this mechanism is very difficult to implement and quite different from other constructs in the name passing world. On the other hand, without match or summation, the output prefixless monadic π-calculus [22, 16, 24, 8, 3] is known as a powerful formalism to represent a wide repertoire of interactive computational structures: polyadic and synchronous communications [22, 8] and even input-guarded summations [40] are embeddable within this calculus. At the practical level, this expressiveness gives rise to a useful high-level concurrent programming language Pict [49] and TyCo [56], which are built on the polyadic version of this calculus with a strong typing system. At the semantic level, there exists a theory of combinators, which is derived from the analysis of the asynchronous name-passing operation [25, 26]. These and other results suggest that we may consider this asynchronous calculus as a basic syntax in the concurrency world just as λ-calculus in the function world; and that the study on expressive power of this calculus would deepen our understating of concurrent computation at the fundamental level. The basic questions which would naturally arise in this context are: How can we reduce this calculus without loss of its expressive power? What computational elements are indispensable to represent the whole behaviour realisable in this calculus?

This paper studies the expressive power of this calculus and its subsystems using the *concurrent combinators* in [25, 26]. More concretely, we show that five atoms introduced in [25, 26], which can represent all processes in this calculus, are indeed semantically indispensable: if any one of combinators is missing, we can no longer express the whole calculus up to semantic equalities.[1] Each combinator has a distinct role to separate a class of interactive behaviours realisable by the original calculus, and is essential for clarifying expressive power of its several interesting proper subsystems. Just like BCWIK-combinators of λ-calculus are useful to categorise and analyse the applicative behaviour of the family of λ-calculi [1, 4, 55], it is often easier and more tractable to check representability in terms of the fixed and finite interaction of the combinators than

[1]This question about minimality of the combinators was independently posed by B. Pierce, D. Sangiorgi and V. Vasconcelos.

considering interaction between arbitrary processes, cf. Sections 3 and 4. Another technical interest would be the introduction of a simple way of measuring expressive power, *generation* and *minimality*, which does not depend on the notion of encodings.[2] In spite of its simplicity, we show that the minimality result is applicable to the establishment of several negative results on (the encodings into) proper subsystems of this calculus, cf. Sections 4 and 5. We hope that these notions would be useful to understand expressiveness of concurrent programming languages in a formal way.

The structure of the rest of the paper follows. Section 2 introduces preliminary definitions and shows the finite generation theorem with a new quick proof. Section 3 proves the main theorem, the minimality of the concurrent combinators. The results in the next two sections are established using this theorem. Section 4 identifies expressive power of several significant proper subsystems of this asynchronous calculus, related to three important elements in name-passing: *locality*, *sharing of names* and *synchronisation*. Section 5 then shows there is no semantically sound encoding of the whole calculus into its proper subsystem under a certain condition. Finally Section 6 summarises the main results and discusses the related works [22, 25, 6, 44, 40, 32] and further issues.

This paper includes all omitted definitions and proofs of [60]. In this full version, we shall newly prove that all of the main theorems (Theorems 2.5, 3.14,3.18 and 5.9) and the main proposition (Proposition 5.6) which were formalised and proposed based on the synchronous bisimilarity in [60] can hold based on other behavioural equalities too: the asynchronous bisimilarity [22] and asynchronous/synchronous reduction based equalities [24, 53].

2 Generation Theorem

This section first introduces the asynchronous π-calculus and its combinatorial representation as far as needed, then establishes the finite generation theorem, which is that, only 5 combinators, which are small proper subset of this calculus, can represent the whole behaviour realisable in this calculus up to parallel composition, replication and name hiding.

2.1 the Asynchronous π-calculus

The formalism we are going to introduce in the following is a small fragment of the original π-calculus [34, 38] based on the notion of asynchronous name passing [22, 8]. It is a succinct yet powerful calculus for concurrent computation, into which we can soundly embed various languages and calculi, for example parallel

[2]Closely related ideas have already been studied by Parrow to examine expressiveness of various synchronisation primitives in a non-value-passing process calculus [45]. See Section 6.2 for discussions.

object-oriented programming languages and λ-calculi. We call this calculus *the asynchronous π-calculus*, or often simply *π-calculus* if there is no confusion.[3] Let **N** be a countable set of *names* (fixed throughout the paper), ranged over by $a, b, c, .., x, y, z, v, w, ...$ The syntax of the calculus is given as follows.

$$P \ ::= \ ax.P \ | \ \overline{a}v \ | \ P|Q \ | \ (\nu a)P \ | \ !P \ | \ \mathbf{0}$$

$P, Q, R, \ldots$ range over the set of terms denoted by $\mathbf{P}_\pi$ generated by the above grammar. "$\overline{a}v$" denotes a *message* which sends a value v to a port a. "$ax.P$" denotes an *input agent* which receives a name and instantiates it in free x's in P. In $ax.P$, the name x binds free occurrences of x in P. "$(\nu a)P$" is a *name hiding of a in P* where the initial a binds its free occurrences in P. "$P|Q$" is a *parallel composition* of P and Q. "$!P$" is a *replicator* which represents the copy of P. "$\mathbf{0}$" is the *inaction*. The definitions of free and bound names in P are standard and denoted by $\mathsf{fn}(P)$ and $\mathsf{bn}(P)$. W.l.o.g. we assume all bound names in P are distinct and disjoint from free names. A name "a" in $\overline{a}v$ and $ax.P$ is called an *output subject* and an *input subject*, respectively. The structural congruence $\equiv$ [5, 34] and the reduction relation $\longrightarrow$ and $\longrightarrow\!\!\!\!\rightarrow$ ($\overset{\text{def}}{=} \longrightarrow^* \cup \equiv$) are given in Appendix A again following the standard definitions [38, 34, 22, 16]. We also use $\equiv_\alpha$ and $\overset{\text{def}}{\equiv}$ for the α-conversion and the literal equality, respectively. The following notations concerning name usage in terms are important.

- $\mathsf{fs}_\uparrow(P)$ and $\mathsf{fs}_\downarrow(P)$ are the sets of the *free output/input subjects* of P, respectively. E.g. $\mathsf{fs}_\downarrow(ax.bx.xy.\overline{c}e) = \{a, b\}$ and $\mathsf{fs}_\uparrow(ax.bx.\overline{c}e) = \{c\}$.
- The sets of *output/input active names* are given by: $a \in \mathsf{an}_\uparrow(P)$ iff $P \equiv (\nu\, \tilde{c})(\overline{a}v|R)$ and $a \in \mathsf{an}_\downarrow(P)$ iff $P \equiv (\nu\, \tilde{c})(ax.Q|R)$ with $a \notin \{\tilde{c}\}$.
- The *convergence predicate* is defined by:
  - $P \Downarrow_{a\uparrow}$ iff $\exists P'.\ P \longrightarrow\!\!\!\!\rightarrow P' \ \wedge\ a \in \mathsf{an}_\uparrow(P')$
  - $P \Downarrow_{a\downarrow}$ iff $\exists P'.\ P \longrightarrow\!\!\!\!\rightarrow P' \ \wedge\ a \in \mathsf{an}_\downarrow(P')$
  - $P \Downarrow_{a\updownarrow}$ iff $P \Downarrow_{a\uparrow}$ or $P \Downarrow_{a\downarrow}$.
- The *number of free occurrences of a in P*, written $\sharp\langle P, a\rangle$, is given as: $\sharp\langle \mathbf{0}, a\rangle = \sharp\langle (\nu\, a)P, a\rangle = \sharp\langle ba.P, a\rangle = 0$, $\sharp\langle \overline{a}v, a\rangle = \sharp\langle \overline{v}a, a\rangle = 1$, $\sharp\langle \overline{a}a, a\rangle = 2$, $\sharp\langle !P, a\rangle = \omega$ if $a \in \mathsf{fn}(P)$ else $\sharp\langle !P, a\rangle = 0$, $\sharp\langle ba.P, b\rangle = 1 + \sharp\langle P, b\rangle$, $\sharp\langle P\,|\,Q, a\rangle = \sharp\langle P, a\rangle + \sharp\langle Q, a\rangle$.

We also use the standard synchronous early transition relation $\xrightarrow{l}$, and synchronous weak bisimilarity $\approx$. See Appendix A for these definitions. The following fact on $\approx$ is notable and used throughout this paper.

[3]This calculus is called ν-*calculus* in [24, 25, 26]. We call the calculus the asynchronous π-calculus since the name is more widely in use nowadays.

PROPOSITION 2.1. (weak bisimilarity) (i) $\approx$ *is a congruence relation [16], and* (ii) $P \approx Q$ *then* $P \Downarrow_{a\downarrow} \;\Leftrightarrow\; Q \Downarrow_{a\downarrow}$.

2.2 Concurrent Combinators

Concurrent combinators are tractable and powerful self-contained proper subset of the asynchronous π-terms, just as **S** and **K** are of λ-calculi. Atomic agents are formed from atoms by connecting "ports" to real "locations" (names), and their computation is based on the notion of dyadic interaction: two atoms interact via a common interaction port to generate new nodes and a new connection topology. See [25, 26, 58] for the full account of basic concepts as well as motivations of this study. Here we begin with seven basic atoms which represent basic elements of communication behaviour in name passing.

$$\mathbf{m}(av) \stackrel{\text{def}}{=} \overline{a}v \quad \mathbf{d}(abc) \stackrel{\text{def}}{=} ax.(\overline{b}x \,|\, \overline{c}x) \quad \mathbf{k}(a) \stackrel{\text{def}}{=} ax.\mathbf{0} \quad \mathbf{fw}(ab) \stackrel{\text{def}}{=} ax.\overline{b}x$$

$$\mathbf{b}_r(ab) \stackrel{\text{def}}{=} ax.\mathbf{fw}(bx) \quad \mathbf{b}_l(ab) \stackrel{\text{def}}{=} ax.\mathbf{fw}(xb) \quad \mathbf{s}(abc) \stackrel{\text{def}}{=} ax.\mathbf{fw}(bc)$$

Their interactive behaviour can be understood in terms of their reduction, as follows.

$$\begin{array}{lclclcl}
\mathbf{d}(abc) \,|\, \mathbf{m}(ae) & \longrightarrow & \mathbf{m}(be) \,|\, \mathbf{m}(ce) & \quad & \mathbf{k}(a) \,|\, \mathbf{m}(ae) & \longrightarrow & \mathbf{0} \\
\mathbf{fw}(ab) \,|\, \mathbf{m}(ae) & \longrightarrow & \mathbf{m}(be) & & \mathbf{b}_r(ab) \,|\, \mathbf{m}(ae) & \longrightarrow & \mathbf{fw}(be) \\
\mathbf{b}_l(ab) \,|\, \mathbf{m}(ae) & \longrightarrow & \mathbf{fw}(eb) & & \mathbf{s}(abc) \,|\, \mathbf{m}(ae) & \longrightarrow & \mathbf{fw}(bc)
\end{array}$$

We write $\mathbf{c}, \mathbf{c}', \ldots$, to denote these agents. $\mathbf{m}(ab)$ (*message*) carries a name b to name a, **d** (*duplicator*) distributes a message to two locations, **fw** (*forwarder*) forwards a message (thus linking two locations), **k** (*killer*) kills a message, while $\mathbf{b}_r$ (*right binder*), $\mathbf{b}_l$ (*left binder*) and **s** (*synchroniser*) generate new links. In particular $\mathbf{b}_r$ and $\mathbf{b}_l$ represent two different ways of binding names — in $\mathbf{b}_r$ one uses the received name for output, while in $\mathbf{b}_l$ one uses it for input. **s** is used for pure synchronisation without value passing, which is indeed necessary in interaction scenarios as seen in the main theorem later.

2.3 Finite Generation

We introduce the ideas of *generation* and *basis* (following the treatment in λ-theory, cf. Def 8.1.1 in [4]), as well as *subsystems*. A closely related idea has also been proposed in [45] for the non-value passing language. These ideas would be generally applicable to both functional and concurrent calculi with suitable adaptation. Then the main theorem of this section is stated and proved.

DEFINITION 2.2.

(i) (generation) Let $X \subset \mathbf{P}_\pi$. The set of terms *generated by* X, notation X^+, is the least set Y such that:

(1) $X \subset Y$.

(2) (a) $P \equiv Q$ and $P \in Y \;\Rightarrow\; Q \in Y$, (b) $\mathbf{0} \in Y$,
(c) $P, Q \in Y \;\Rightarrow\; P|Q \in Y$, (d) $P \in Y \;\Rightarrow\; (\nu a)P \in Y$,
(e) $P \in Y \;\Rightarrow\; !P \in Y$, and
(f) $P \in Y \;\Rightarrow\; P\sigma \in Y$ where σ is a injective renaming.

(ii) (basis) Let $Y \subset \mathbf{P}_\pi$. Then $X \subset \mathbf{P}_\pi$ is a *basis for* Y (up to $\approx$) iff

$$\forall P \in Y. \exists Q \in X^+ \quad P \approx Q$$

X is called a *basis* if X is a basis for the set of π-terms.

(iii) (subsystem) Let $\mathbf{P} \subseteq \mathbf{P}_\pi$ and $\mathbf{P}^+ = \mathbf{P}$. Then $\mathbf{P}$ is called:

subsystem	if $P \in \mathbf{P} \wedge P \twoheadrightarrow Q$	$\Rightarrow\; Q \in \mathbf{P}$.
subsystem up to $\approx$	if $P \in \mathbf{P} \wedge P \twoheadrightarrow Q$	$\Rightarrow\; \exists R.\, Q \approx R \in \mathbf{P}$.
t-subsystem	if $P \in \mathbf{P} \wedge P \stackrel{\hat{l}}{\Longrightarrow} Q$	$\Rightarrow\; Q \in \mathbf{P}$.
t-subsystem up to $\approx$	if $P \in \mathbf{P} \wedge P \stackrel{\hat{l}}{\Longrightarrow} Q$	$\Rightarrow\; \exists R.\, Q \approx R \in \mathbf{P}$.

We also say $\mathbf{P}_1$ is a (t-)*subsystem of* $\mathbf{P}_2$ (*up to* $\approx$) if $\mathbf{P}_1$ and $\mathbf{P}_2$ are (t-)subsystems and $\mathbf{P}_1 \subseteq \mathbf{P}_2$.

(iv) (subbasis) Y_1 is a *subbasis* of Y_2, written $Y_1 \lesssim Y_2$, if $\mathbf{P}_i = \{P \mid P \approx Q \in Y_i^+\}$ is a subsystem ($i = 1, 2$) and $\mathbf{P}_1 \subseteq \mathbf{P}_2$. We write $Y_1 \simeq Y_2$ if both $Y_1 \lesssim Y_2$ and $Y_2 \lesssim Y_1$; and $Y_1 \lnsim Y_2$ if both $Y_1 \lesssim Y_2$ and $Y_2 \not\lesssim Y_1$.

In (i), the set Y generated by X includes structural rules (a) and inaction (b), and it is closed under reduction contexts (c–e) and renaming operators (f) (cf. [17, 18, 44])[4]. (ii) says that if X is a basis then any π-term should be behaviourally equivalent to some term generated by X. (iii) means a subsystem $\mathbf{P}$ should be self-contained w.r.t reduction. In (iv), $Y_1 \simeq Y_2$ means two subsystems generated by Y_1 and Y_2 have the same expressive power. Note the relation $Y_1 \lesssim Y_2$ can be defined even if $Y_1 \not\subseteq Y_2$ and $Y_2 \not\subseteq Y_1$ and Y_1 and Y_2 themselves are not subsystems. From a programming viewpoint, if X is a basis for Y, any program written in a language Y can be described by a composition of programs written in its "core language" X, and if X is a subsystem, then X can be used as a self-contained language because it is closed under evaluation. A fact related with (iii,iv) follows.

FACT 2.3. *Let* $\mathbf{P}, \mathbf{P}_1, \mathbf{P}_2$ *be subsystems. Then we have:*

(i) $\lesssim$ *is a preorder and* $\mathbf{P}_1 \subseteq \mathbf{P}_2 \;\Rightarrow\; \mathbf{P}_1 \lesssim \mathbf{P}_2$.

(ii) Y *is a basis of* $\mathbf{P}$ *iff* $\mathbf{P} \lesssim Y$.

(iii) *If* $Y_1 \lesssim Y_2$ *and* $Y_1 \supseteq Y_2$, *then* $Y_1 \simeq Y_2$.

[4]The use of injective renaming instead of usual substitution (i.e. non-injective renaming) is preferable because equalities over processes found in the literature are usually closed under injective renaming, but may not be closed under substitutions. See [18, 17] for details.

(iv) *If Y is a subbasis, then Y^+ is a subsystem up to* $\approx$.

(v) *If* $\mathbf{P}_0$ *is a* (t-)*subsystem up to* $\approx$, *then* $\{P \mid P \approx Q \in \mathbf{P}_0\}$ *is a* (t-)*subsystem.*

REMARK 2.4.

- Without "!" in (i) in Definition 2.2, the finite generation with at most 19 combinators is possible by the result in [26]. Here we include $!P$ because we are concerned with expressiveness in terms of communication behaviours.
- In (ii), we can use any weak equalities (asynchronous weak bisimilarity [22], barbed congruence [3], and the maximum sound theory [24]) instead of the synchronous bisimilarity to reach the main theorems of this paper; see Remark 2.10, Theorems 3.24, 3.25 and 5.14 for more details.
- (iii,iv) can be generally extended to discuss the relationship among the family of π-calculi by considering the subsystems of the full synchronous polyadic π-calculus. E.g. the asynchronous π-calculus is a subsystem of monadic synchronous π-calculus [34] and the monadic synchronous π-calculus is that of polyadic π-calculus [35] etc. See Sections 5 and 6 for more discussions on expressiveness in π-family.

  Now let us define a set of five combinators as follows. We assume a, b and c are pairwise distinct.

$$\mathbf{C} \stackrel{\text{def}}{=} \{\mathbf{m}(ab),\ \mathbf{d}(abc),\ \mathbf{b}_r(ab),\ \mathbf{b}_l(ab),\ \mathbf{s}(abc)\}$$

The main theorem of this section states these 5 combinators can generate whole set of terms up to the weak bisimilarity. The next subsection shows the proof of this theorem.

THEOREM 2.5. (finite generation) $\mathbf{C}$ *is a basis and* $\mathbf{P}_\pi \simeq \mathbf{C}$.

2.4 Proof of the Finite Generation Theorem

We first introduce the set of combinators corresponding to subsection 2.2.

$$\begin{aligned}\mathbf{C}_7 \stackrel{\text{def}}{=} \{&\mathbf{m}(ab), \mathbf{m}(aa), \mathbf{d}(abc), \mathbf{d}(abb), \mathbf{d}(aab), \mathbf{d}(aaa), \mathbf{k}(a),\\ &\mathbf{fw}(ab), \mathbf{fw}(aa), \mathbf{b}_r(ab), \mathbf{b}_r(bb), \mathbf{b}_l(ab), \mathbf{b}_l(bb),\\ &\mathbf{s}(abc), \mathbf{s}(aab), \mathbf{s}(aba), \mathbf{s}(abb), \mathbf{s}(aaa)\ \}\end{aligned}$$

where a, b and c are pairwise distinct. Let $\mathbf{P_{cc}} \stackrel{\text{def}}{=} \mathbf{C}_7^+$. Then clearly $\mathbf{P_{cc}}$ is a t-subsystem of $\mathbf{P}_\pi$ by checking the reduction rules for atomic agents. To prove the main theorem, we first show $\mathbf{C}_7$ is a basis: any prefix of the asynchronous π-calculus can be represented following the idea in [25] (the rule (IV) is newly defined). We assume the following annotations, which denote how each name is used in the rules of interaction.

$$\mathbf{m}(a^+v^\pm), \mathbf{d}(a^-b^+c^+), \mathbf{k}(a^-), \mathbf{fw}(a^-b^+), \mathbf{b}_l(a^-b^+), \mathbf{b}_r(a^-b^-), \mathbf{s}(a^-b^-c^+)$$

Note the annotated polarities are preserved by reduction, e.g.

$$\mathbf{d}(a^-b^+c^+) \mid \mathbf{m}(a^+v) \longrightarrow\ \mathbf{m}(b^+v) \mid \mathbf{m}(b^+v).$$

It will be clear from the following definition that 7 atoms are in appropriate forms to decompose input prefixes.

DEFINITION 2.6. (name abstraction) For P being any agent, we inductively form the agent denoted by $a^*x.P$ in Table 1 where rules are applied from (I) to (XIII) in this order and c, c_1, c_2 are fresh and pairwise distinct.

(I).	$a^*x.(P \mid Q) \stackrel{\text{def}}{=} (\nu\, c_1c_2)(\mathbf{d}(ac_1c_2) \mid c_1^*x.P \mid c_2^*x.Q)$	
(II).	$a^*x.(\nu\, c')P \stackrel{\text{def}}{=} (\nu\, c)a^*x.P\{c/c'\}$	
(III).	$a^*x.\mathbf{0} \stackrel{\text{def}}{=} \mathbf{k}(a)$	
(IV).	$a^*x.!P \stackrel{\text{def}}{=} (\nu\, c)(\mathbf{fw}(ac) \mid !c^*x.(P \mid \mathbf{m}(cx)))$	
(V).	$a^*x.\mathbf{c}(v^+\tilde{w}) \stackrel{\text{def}}{=} (\nu\, c)(\mathbf{s}(acv) \mid \mathbf{c}(c^+\tilde{w}))$	$x \notin \{v\tilde{w}\}$
(VI).	$a^*x.\mathbf{c}(v^-\tilde{w}) \stackrel{\text{def}}{=} (\nu\, c)(\mathbf{s}(avc) \mid \mathbf{c}(c^-\tilde{w}))$	$x \notin \{v\tilde{w}\}$
(VII).	$a^*x.\mathbf{m}(vx) \stackrel{\text{def}}{=} \mathbf{fw}(av)$	$x \neq v$
(VIII).	$a^*x.\mathbf{fw}(xv) \stackrel{\text{def}}{=} \mathbf{b}_l(av)$	$x \neq v$
(IX).	$a^*x.\mathbf{fw}(vx) \stackrel{\text{def}}{=} \mathbf{b}_r(av)$	$x \neq v$
(X).	$a^*x.\mathbf{c}(\tilde{v}_1x^+\tilde{v}_2) \stackrel{\text{def}}{=} (\nu\, c)a^*x.(\mathbf{fw}(cx) \mid \mathbf{c}(\tilde{v}_1c^+\tilde{v}_2))$	$x \notin \{\tilde{v}_1\}$
(XI).	$a^*x.\mathbf{c}(x^-\tilde{v}) \stackrel{\text{def}}{=} (\nu\, c)a^*x.(\mathbf{fw}(xc) \mid \mathbf{c}(c^-\tilde{v}))$	
(XII).	$a^*x.\mathbf{b}_r(vx^-) \stackrel{\text{def}}{=} (\nu\, c_1c_2c_3)a^*x.(\mathbf{d}(vc_1c_2) \mid \mathbf{s}(c_1xc_3) \mid \mathbf{b}_r(c_2c_3))$	$x \neq v$
(XIII).	$a^*x.\mathbf{s}(vx^-w) \stackrel{\text{def}}{=} (\nu\, c_1c_2)a^*x.(\mathbf{s}(vc_1c_2) \mid \mathbf{m}(c_1x) \mid \mathbf{b}_l(c_2w))$	$x \neq v$

TABLE 1. Prefix Mapping

The following proposition shows that $a^*x.P$ behaves as we expected.

PROPOSITION 2.7.

(i) $ax.P \approx a^*x.P$.

(ii) $P \approx Q \Rightarrow a^*x.P \approx a^*x.Q$.

(iii) $a^*x.P \mid \mathbf{m}(av) \longrightarrow\approx P\{v/x\}$.

PROOF. (i) By rule induction in Definition 2.6. Rules except (II) and (VI) are all mechanical. For (II) and (VI), we use the (extended) β-reduction $\longrightarrow_\beta$ in [25]. See Appendix B. For (ii), by Proposition 2.1 (i), we have $P \approx Q \Rightarrow ux.P \approx ux.Q$, then by (i), $u^*x.P \approx ux.P \approx ux.Q \approx u^*x.Q$, as required. (iii) $u^*x.P \mid \mathbf{m}(uv) \longrightarrow\!\!\!\!\rightarrow\approx P\{v/x\}$ is from (i) and Proposition 2.1 (i). $\longrightarrow\!\!\!\!\rightarrow$ becomes $\longrightarrow$ by the proof of (i). $\square$

Now we can decompose all the asynchronous π-terms to $\mathbf{P_{cc}}$ with the following mapping.

$$[\![\overline{a}b]\!] \overset{\text{def}}{=} \mathbf{m}(ab) \qquad [\![ax.Q]\!] \overset{\text{def}}{=} a^{*}x.[\![Q]\!] \qquad [\![P\,|\,Q]\!] \overset{\text{def}}{=} [\![P]\!]\,|\,[\![Q]\!]$$
$$[\![(\nu\, a)P]\!] \overset{\text{def}}{=} (\nu\, a)[\![P]\!] \qquad [\![!P]\!] \overset{\text{def}}{=} ![\![P]\!] \qquad [\![\mathbf{0}]\!] \overset{\text{def}}{=} \mathbf{0}$$

Note that for all v, x, we have $[\![P\{v/x\}]\!] \equiv [\![P]\!]\{v/x\}$. The key lemma follows.

LEMMA 2.8. $P \approx [\![P]\!]$.

PROOF. By the structural induction. Base cases, $P \equiv \mathbf{0}$ and $P \equiv \overline{a}b$, are easy. At the induction step, we use Proposition 2.7 (i) for the input prefix, while others are done by Proposition 2.1 (i). $\square$

Notice the above lemma together with (i) in Proposition 2.1 immediately establishes that $[\![\ ]\!]$ is a fully abstract mapping, i.e. $P \approx Q \Leftrightarrow [\![P]\!] \approx [\![Q]\!]$. Now we know $\mathbf{C}_7$ is a basis via $[\![\ ]\!]$, but $\mathbf{C}_7$ is not a minimal basis: the number of atoms in $\mathbf{C}_7$ can be decreased to $\mathbf{C}$ in the following way.

First $\mathbf{fw}(ab)$ and $\mathbf{k}(a)$ can be expressed with other 5 combinators, e.g. $\mathbf{fw}(ab) \approx (\nu\, c)\mathbf{d}(abc)$ and $\mathbf{k}(b) \approx (\nu\, c)\mathbf{b}_l(bc)$. Thus the number has become 15 from 18. Secondly note, for each renaming σ, an atom with the identical arguments like $\mathbf{m}(aa)$ can not be directly generated by renaming one atom in $\mathbf{C}_7$; i.e. for all renaming σ, $\mathbf{m}(bb) \not\approx \mathbf{m}(ab)\sigma$, etc. But *substitution can be represented by forwarders up to* $\approx$, by the following lemma.

LEMMA 2.9. (substitution decomposition)

(i) $\mathbf{c}(a^{-}\tilde{b}) \approx (\nu\, c)(\mathbf{fw}(ac)\,|\,\mathbf{c}(c\tilde{b}))$ *and* $\mathbf{c}(\tilde{a}b^{+}\tilde{a}') \approx (\nu\, c)(\mathbf{fw}(cb)\,|\,\mathbf{c}(\tilde{a}c^{+}\tilde{a}'))$ *with c fresh.*

(ii) For all $\mathbf{c} \in \mathbf{C}$, we have: $\mathbf{c}(\tilde{v}) \approx (\nu\, \tilde{c})(\mathbf{c}_1(\tilde{v}_1)\,|\,...\,|\,\mathbf{c}_n(\tilde{v}_n))$ where $\mathbf{c}(\tilde{v}_i)\sigma_i \in \mathbf{C}$ and σ_i is a renaming operator.

PROOF. (i) is mechanical. Then (ii) is proved by (i) as follows:

- $\mathbf{m}(aa) \approx (\nu\, c)(\mathbf{m}(ca)\,|\,\mathbf{fw}(ca))$,
- $\mathbf{d}(aaa) \approx (\nu\, c_1c_2)(\mathbf{d}(ac_1c_2)\,|\,\mathbf{fw}(c_1a)\,|\,\mathbf{fw}(c_2a))$,
- $\mathbf{b}_r(aa) \approx (\nu\, c)(\mathbf{fw}(ac)\,|\,\mathbf{b}_r(ca))$,
- $\mathbf{b}_l(aa) \approx (\nu\, c)(\mathbf{fw}(ac)\,|\,\mathbf{b}_l(ca))$, and
- $\mathbf{s}(aaa) \approx (\nu\, c_1c_2)(\mathbf{fw}(ac_1)\,|\,\mathbf{s}(c_1ac_2)\,|\,\mathbf{fw}(c_2a))$. $\square$

Now we finish proving Theorem 2.5: by the above arguments, if $P \in \mathbf{P_{cc}}$, then $\exists Q \in \mathbf{C}^{+}.\ Q \approx P$. Moreover if $P \in \mathbf{P}_\pi$, then $[\![P]\!] \in \mathbf{P_{cc}}$. Thus by $P \approx [\![P]\!]$ for all $P \in \mathbf{P}_\pi$ in Lemma 2.8, we have $P \approx [\![P]\!] \approx Q \in \mathbf{C}^{+}$, hence $\mathbf{P}_\pi \lesssim \mathbf{C}$ as required. The second inclusion is by Fact 2.3 (iii) with $\mathbf{P}_\pi \supseteq \mathbf{C}$.

REMARK 2.10. (finite generation by other equivalences) The finite generation theorem can be established based on other known weaker equalities — (1) *the asynchronous bisimilarity* (denoted by $\approx_a$) [22, 23, 16] and (2) *the synchronous* and *asynchronous maximum sound equalities* (denoted by $=_s$ and $=_a$ [24]) which coincide with the synchronous and asynchronous barbed congruences (cf.[10]). See Appendix A for the definitions of the asynchronous bisimilarity and the sound equalities.

For the proof of the finite generation in these settings, we need to change the definition of basis, subsystem and subbasis by replacing $\approx$ with $\approx_a$, $=_s$ and $=_a$, respectively. Then all we have to prove is Lemma 2.8 for each equality, which is immediate from $\approx \subset \approx_a \subset =_a$ and $\approx \subset =_s \subset =_a$.

3 Minimality Theorem

3.1 Minimal Basis

This section establishes the main result of this paper — the minimality of $\mathbf{C}$, i.e. any proper subset of $\mathbf{C}$ cannot become a basis. We also show that this result can be extended even if we use more weak behavioural equalities in Subsection 3.7. Intuitively, the main theorem means there exists a program which can be described in a core-language, but not in its proper subset.

DEFINITION 3.1. (minimal basis) Let $\backslash$ be the set difference operator. Assume Y is a basis and $P \in Y$. Then we say P is *essential w.r.t.* Y if $Y\backslash\{P\}$ is not a basis. We call Y a *minimal basis* if all elements of Y are essential.

LEMMA 3.2. *Let $Y \subseteq \mathbf{C}_7$ and write $Y\backslash\mathbf{c}$ for $Y\backslash\{\mathbf{c}(\tilde{v}_1),..,\mathbf{c}(\tilde{v}_n)\}$ with $\mathbf{c}(\tilde{v}_i) \in Y$, i.e. all terms of the form $\mathbf{c}(\tilde{v}_i)$ are deleted from Y. Then for all $\mathbf{c}(\tilde{v}) \in \mathbf{C}$, we have:*

(i) $(\mathbf{C}_7\backslash\mathbf{c})^{+}$ *is a subsystem, and* $(\mathbf{C}\backslash\mathbf{c})^{+}$ *is a subsystem up to* $\approx$. *If* $\mathbf{c} \neq \mathbf{m}$, *then* $(\mathbf{C}_7\backslash\mathbf{c})^{+}$ *is a t-subsystem and* $(\mathbf{C}\backslash\mathbf{c})^{+}$ *is a t-subsystem up to* $\approx$.

(ii) $\mathbf{C}\backslash\mathbf{c} \lesssim \mathbf{C}_7$, *and* $\mathbf{C}\backslash\mathbf{c} \simeq \mathbf{C}_7\backslash\mathbf{c} \simeq (\mathbf{C}_7\backslash\mathbf{c})^{+}$.

(iii) $(\mathbf{C}_7\backslash\mathbf{c})^{+}$ *is not a basis iff* $\mathbf{C}\backslash\mathbf{c}$ *is not a basis.*

PROOF. (i) The case $\mathbf{c} = \mathbf{m}$ is obvious because $P \overset{l}{\not\longrightarrow}$ with l an output for all $P \in (\mathbf{C}_7\backslash\mathbf{m})^{+}$. Let $\mathbf{c} \neq \mathbf{m}$. Then if $\mathbf{c}(\tilde{a}) \xrightarrow{l} P$, then for all l, P is a forwarder or messages. Hence $(\mathbf{C}_7\backslash\mathbf{c})^{+}$ is a t-subsystem, and by Fact 2.3 (iv), $(\mathbf{C}\backslash\mathbf{c})^{+}$ is a t-subsystem up to $\approx$. (ii) $\mathbf{C}\backslash\mathbf{c}$ is a basis of $(\mathbf{C}_7\backslash\mathbf{c})^{+}$ since $\mathbf{fw}(ab)$ and $\mathbf{k}(a)$ can be represented by whichever $\mathbf{d}(abc)$, $\mathbf{b}_l(ab)$, $\mathbf{b}_r(ab)$ or $\mathbf{s}(abc)$ (for cases $\mathbf{b}_l$ and $\mathbf{b}_r$, we need $\mathbf{m}$, e.g. $\mathbf{fw}(ab) \approx (\nu\, c)(\mathbf{b}_l(cb)\,|\,\mathbf{m}(ca)) \approx (\nu\, c)(\mathbf{b}_r(ca)\,|\,\mathbf{m}(cb))$). Then we use Fact 2.3, noting $\mathbf{C}_7\backslash\mathbf{c}$ and $\mathbf{C}\backslash\mathbf{c}$ are both basses of a subsystem $(\mathbf{C}_7\backslash\mathbf{c})^+$. (iii) is an easy corollary of (ii). □

We often simply say "P is essential" to mean "P is essential w.r.t. $\mathbf{C}$" and write $\mathbf{P_{cc}}\backslash\mathbf{c}$ for $(\mathbf{C}_7\backslash\mathbf{c})^+$ with $\mathbf{c}(\tilde{v}) \in \mathbf{C}$ from the following. Note if $\mathbf{c} = \mathbf{fw}$, then $\mathbf{P_{cc}}\backslash\mathbf{c}$ is not a subsystem. By (iv) above, to verify the essentiality of $\mathbf{c}$ we will equivalently prove $\mathbf{P_{cc}}\backslash\mathbf{c} \lnsim \mathbf{C}$.

3.2 Output and Duplication

It is clear that "the minimum output", i.e. a message, is needed.

PROPOSITION 3.3. *$\mathbf{m}(ab)$ is essential.*

PROOF. Clearly $\forall P \in \mathbf{P_{cc}}\backslash\mathbf{m}.\ \neg P \overset{\overline{a}b}{\Longrightarrow}$, while we have: $\mathbf{m}(ab) \xrightarrow{\overline{a}b} \mathbf{0}$. □

$\mathbf{d}(abc)$ is the only agent who distributes the same value to two locations. Therefore, without $\mathbf{d}(abc)$, we can not realize *sharing of names* in π-calculus, as formally proved in the following lemma ($\sharp\langle P, e\rangle$ was given in 2.1.).

LEMMA 3.4.

(i) *For all $P \in \mathbf{P_{cc}}\backslash\mathbf{d}$, $P \longrightarrow\!\!\!\!\!\rightarrow P'$ implies $\sharp\langle P, e\rangle \geq \sharp\langle P', e\rangle$.*

(ii) *Suppose $P \in \mathbf{P}_\pi$ and $P \overset{l}{\Longrightarrow}\overset{l'}{\Longrightarrow} P'$ where $l = \overline{b}e$ or $\overline{b}(e)$ and $l' = \overline{c}e'$ or $\overline{c}(e')$ with $\mathsf{bn}(l) \cap \mathsf{bn}(l') = \emptyset$. Then there exists Q' such that $P \longrightarrow\!\!\!\!\!\rightarrow (\nu\,\tilde{f})(Q' \,|\, \mathbf{m}(be) \,|\, \mathbf{m}(ce'))$.*

PROOF. (i) Note if e appears under replication in P, then $\sharp\langle P, e\rangle = \omega$. Hence we have $P \equiv Q \;\Rightarrow\; \sharp\langle P, e\rangle = \sharp\langle Q, e\rangle$. The rest is straightforward by checking reduction rules in 2.2. (ii) is because of asynchrony of combinators. □

PROPOSITION 3.5. *$\mathbf{d}(abc)$ is essential.*

PROOF. Suppose $P \in \mathbf{P_{cc}}\backslash\mathbf{d}$ and $P \approx \mathbf{d}(abc)$ as a contradiction with e fresh. By Proposition 2.1 (i), we have $(P \,|\, \mathbf{m}(ae)) \approx (\mathbf{d}(abc) \,|\, \mathbf{m}(ae))$. Then we know: $(\mathbf{d}(abc) \,|\, \mathbf{m}(ae)) \longrightarrow \xrightarrow{\overline{b}e} \xrightarrow{\overline{c}e} \mathbf{0}$ while $(P \,|\, \mathbf{m}(ae)) \overset{\overline{b}e}{\Longrightarrow}\overset{\overline{c}e}{\Longrightarrow}$ is impossible because if so, $(P \,|\, \mathbf{m}(ae)) \longrightarrow\!\!\!\!\!\rightarrow (\nu\,\tilde{c})(\mathbf{m}(ae) \,|\, \mathbf{m}(be) \,|\, Q)$ for some $\tilde{c}$ and Q by Lemma 3.4 (ii), but this contradicts Lemma 3.4 (i) because of $\sharp\langle (P \,|\, \mathbf{m}(ae)), e\rangle = 1$. □

The duplicator $\mathbf{d}(abc)$ has two functionalities: duplication of the same value and distribution of two messages. There is a further point which needs to be clarified: whether parallelism is increased without duplicator. The answer will be given in Section 4 later using a pure distributor which just increments parallelism.

3.3 Binders

Next we consider two link generators $\mathbf{b}_l(ab)$ and $\mathbf{b}_r(ab)$. The former is the only agent which can create a new input-subject by a value which it receives (x in $ax.\mathbf{fw}(xb)$), and the latter is the only one which can create a new output-subject (x in $ax.\mathbf{fw}(bx)$).

LEMMA 3.6.

(i) *For all $P \in \mathbf{P_{cc}}\backslash\mathbf{b}_l$, $P \xrightarrow{l} P'$ implies $\mathsf{fs}_{\downarrow}(P) \supseteq \mathsf{fs}_{\downarrow}(P')$.*

(ii) *For all $P \in \mathbf{P_{cc}}\backslash\mathbf{b}_r$, $P \xrightarrow{l} P'$ implies $\mathsf{fs}_{\uparrow}(P) \supseteq \mathsf{fs}_{\uparrow}(P')$.*

PROPOSITION 3.7. *Both $\mathbf{b}_l(ab)$ and $\mathbf{b}_r(ab)$ are essential.*

PROOF. Assume $P \in \mathbf{P_{cc}}\backslash\mathbf{b}_l$ and $P \xrightarrow{ae} P'$ with e fresh. Then by Lemma 3.2 (i), $P' \in \mathbf{P_{cc}}\backslash\mathbf{b}_l$, hence $\neg P' \overset{ec}{\Longrightarrow}$ by Lemma 3.6 (i). But we have $\mathbf{b}_l(ab) \xrightarrow{ae} \mathbf{fw}(eb) \xrightarrow{ec} \mathbf{m}(bc)$, a contradiction. The case $\mathbf{b}_r(ab)$ is just similar by changing input with output. □

Lemma 3.6 simplily explains roles of $\mathbf{b}_l$ and $\mathbf{b}_r$ through the proof of their essentiality. We can not reduce, however, the syntax of $\mathbf{b}_l(ab)$ and $\mathbf{b}_r(ab)$ even if we still keep the capability to create the new input and output subject names in the following sense: $\mathbf{b}_l(ab)$ can not be replaced with $ax.xy.\mathbf{0}$ and $\mathbf{b}_r(ab)$ cannot be replaced with $ax.\overline{x}b$. These results will be proved in Theorem 3.18 and Proposition D.4. The next section also shows that $\mathbf{b}_l$ separates locality from non-locality of name-passing in π-calculus, and a commutative version of $\mathbf{b}_r$ characterises a more asynchronous calculus than this asynchronous π-calculus.

3.4 Needed redex pair

To formalise the causality of interaction between combinators, we here introduce the notions of an *occurrence* and a *needed redex pair*, following the notion of λ-calculus (cf. [4]).

DEFINITION 3.8. (occurrence and subterms) Let ε be a empty sequence. Then the set of *occurrences of term P*, denoted by $O(P)$ and ranged over by $u, u', \ldots$, is inductively defined as:

- $P \overset{\text{def}}{\equiv} \mathbf{0}$ or $P \overset{\text{def}}{\equiv} \mathbf{c}(\tilde{v}) \;\Rightarrow\; O(P) = \{\varepsilon\}$.
- $P \overset{\text{def}}{\equiv} P_1 \,|\, P_2 \;\Rightarrow\; O(P) = \{\varepsilon\} \cup \{i \cdot u \mid u \in O(P_i),\ i = 1, 2\}$.
- $P \overset{\text{def}}{\equiv} (\nu\,a)P',\ !P' \;\Rightarrow\; O(P) = \{\varepsilon\} \cup \{1 \cdot u \mid u \in O(P')\}$.

The subterm of P at $u \in O(P)$ is denoted by P/u. We represent the occurrence of a subterm of P by the corresponding subterm if there is no ambiguity.

For the simple example, let $R \overset{\text{def}}{\equiv} (\nu a)(\mathbf{m}(ab) \,|\, (\mathbf{b}_l(ab) \,|\, \mathbf{m}(ac)))$. Then $R/1 \overset{\text{def}}{\equiv} \mathbf{m}(ab) \,|\, (\mathbf{b}_l(ab) \,|\, \mathbf{m}(ac))$ and $R/1 \cdot 2 \cdot 1 \overset{\text{def}}{\equiv} \mathbf{b}_l(ab)$. In the following, we define which pair of combinators are *needed* to create a new combinator.

DEFINITION 3.9. (a needed redex pair)

(1) Let Δ be a tuple of occurrences, say $\Delta = \langle u_1,\ u_2 \rangle$, and write $P \xrightarrow{\Delta} P'$ if $P \xrightarrow{\tau} P'$ is obtained by interaction between $\mathbf{c}(x^- \tilde{v}) \overset{\text{def}}{\equiv} P/u_1$ and $\mathbf{m}(xy) \overset{\text{def}}{\equiv} P/u_2$ in P. Assume the derivation of $P \xrightarrow{\Delta} P'$ includes either

$\mathbf{c}(x^- \tilde{v}) \xrightarrow{xy} \mathbf{c}'(\tilde{w})$,
$\mathbf{c}(x^- \tilde{v}) \xrightarrow{xy} \mathbf{c}'(\tilde{w}) \,|\, \mathbf{c}_0(\tilde{v})$, or
$\mathbf{c}(x^- \tilde{v}) \xrightarrow{xy} \mathbf{c}_0(\tilde{v}) \,|\, \mathbf{c}'(\tilde{w})$

in its proof. Then we say Δ in P is *directly needed* for $\mathbf{c}'(\tilde{w})$.

(2) Assume a finite and infinite τ-action sequence

$$P_0 \xrightarrow{\Delta_0} P_1 \xrightarrow{\Delta_1} P_2 \xrightarrow{\Delta_2} \cdots \xrightarrow{\Delta_{n-1}} P_n \cdots.$$

- Suppose Δ_0 is directly needed for $\mathbf{c}'(\tilde{w}) \overset{\text{def}}{\equiv} P_1/u$. If $\forall j\, 0 \leq j \leq i.\, u \notin \Delta_j$, then we say Δ_0 is *needed* for (the occurrence of) $\mathbf{c}'(\tilde{w}') \overset{\text{def}}{\equiv} P_i/u$ in P_i.[5]
- Write $\Delta_i \succ \Delta_j$ $(0 \leq i < j)$ if Δ_i is needed for one of the combinators at Δ_j in P_j. We say Δ_i in P_i is *needed* for (the occurrence of) $\mathbf{c}'(\tilde{w})$ in P_n if there is a chain of needed pairs s.t.

$$\Delta_i \overset{\text{def}}{\equiv} \Delta_{i_0} \succ \Delta_{i_1} \succ \Delta_{i_2} \succ \cdots \succ \Delta_{i_m} \quad \text{with} \quad i \leq i_k < i_{k+1} \leq n-1$$

and Δ_{i_m} is needed for $\mathbf{c}'(\tilde{w})$ in P_n.

For example, assume a given reduction sequence in the following.

$$\begin{aligned}(\nu c)(\mathbf{d}(abc) \,|\, \mathbf{m}(ae) \,|\, \mathbf{b}_r(bd)) &\xrightarrow{\tau} (\nu c)(\mathbf{m}(be) \,|\, \mathbf{m}(ce) \,|\, \mathbf{0} \,|\, \mathbf{b}_r(bd)) \\ &\xrightarrow{\tau} (\nu c')(\mathbf{0} \,|\, \mathbf{m}(c'e) \,|\, \mathbf{0} \,|\, \mathbf{fw}(de))\end{aligned}$$

Note $\mathbf{0}$ comes from τ-actions. Here $\mathbf{d}(abc)$ and $\mathbf{m}(ae)$ are needed for $\mathbf{m}(be)$ and $\mathbf{m}(ce)$, and $\mathbf{m}(be)$ and $\mathbf{b}_r(bd)$ are needed for $\mathbf{fw}(de)$, hence $\mathbf{d}(abc)$ and $\mathbf{m}(ae)$ are needed for $\mathbf{fw}(de)$. $\mathbf{d}(abc)$ and $\mathbf{m}(ae)$ are also needed for $\mathbf{m}(c'e)$. Notice there may be several needed chains for one combinator (cf. Example in Subsection 3.2). The following simple fact, however, holds because $\xrightarrow{\tau}$ is based on the labelled transition relation without $\equiv$, cf. Appendix A.

[5] I.e. $\mathbf{c}'(\tilde{w})$ in P_1 remains as $\mathbf{c}'(\tilde{w}')$ in P_i without interaction with any combinators. Note $\mathbf{c}'(\tilde{w})$ may be changed to $\mathbf{c}'(\tilde{w}')$ by α-conversions during τ-actions.

FACT 3.10. *Suppose $P_0 \xrightarrow{\Delta_0} P_1 \xrightarrow{\Delta_1} P_2 \xrightarrow{\Delta_2} \cdots \xrightarrow{\Delta_{n-1}} P_n$ and there is a sequence of needed redex pairs $\Delta_i \overset{\text{def}}{=} \Delta_{i_0} \succ \cdots \succ \Delta_{i_m}$ with $i_m \geq 1$ for $\mathbf{c}(\tilde{w})$ in P_n. Then:*

(i) *Δ_{i_m} and a set of needed chains to $\mathbf{c}(\tilde{v})$ are unique.*
(ii) *for all i_k, Δ_{i_k} does not contains either $\mathbf{0}$ nor $\mathbf{k}(\tilde{v})$.*
(iii) *$\mathbf{c}(\tilde{w})$ in P_n does not occur under replicators.*

3.5 Synchronisation

Now we prove the most interesting and difficult part: *creating some term* (a forwarder or a message, cf. Lemma 3.16) *after synchronisation, while doing no name-instantiation*, is really essential to represent the whole behaviour of π-calculus.

To prove the key proposition, we formalise the idea of *general synchroniser* in name-passing. The following definition says that interaction with a message $\mathbf{m}(ae)$ is needed to create a new interaction point at b, and at the same time a value e is not used for that purpose.

DEFINITION 3.11. (general synchroniser) Let $a \neq b$. A *general synchroniser from a to b* is a term P such that (1) $\neg P \Downarrow_{b\downarrow}$, (2) $(P \,|\, \mathbf{m}(ae)) \Downarrow_{b\downarrow}$, and (3) $\neg(P \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(bc)) \Downarrow_{e\downarrow}$ where e is fresh in (2) and (3).

We can easily check none of $\mathbf{m}$, $\mathbf{d}$, $\mathbf{fw}$ and $\mathbf{b}_l$ is a general synchroniser because they cannot create a new input subject after interaction with a message $\mathbf{m}(ae)$. If $\mathbf{k}(a)$ interacts with a message $\mathbf{m}(ae)$, then a value e is thrown away. Hence it satisfies (1) and (3), but does not have the property (2). $\mathbf{b}_r(ab)$ satisfies (1) and (2), but does not have the property (3) since a value is used as an output subject. On the other hand, $\mathbf{s}(abc)$ is a general synchroniser at a to b as shown later.

In this way, it is easy to check every atom except $\mathbf{s}(abc)$ is not a general synchroniser. But *is it indeed impossible to represent a general synchroniser by any composition of six combinators except $\mathbf{s}(abc)$ using operators $|$, ! and ν up to weak bisimilarity?* The following lemma answers this question.

LEMMA 3.12. **(Main Lemma)** $\mathbf{P_{cc}} \backslash \mathbf{s}$ *has no general synchroniser.*

PROOF. Suppose $P \in \mathbf{P_{cc}} \backslash \mathbf{s}$ is a general synchroniser at a to b. Since $b \notin \mathtt{an}_{\downarrow}(P')$ for all P' s.t. $P \longrightarrow\!\!\!\!\rightarrow P'$, there are only two ways for b to be created as a new active input subject:

(a) $\mathbf{b}_r(db)$ and $\mathbf{m}(df)$ interact,
i.e. $\mathbf{b}_r(db) \,|\, \mathbf{m}(df) \longrightarrow \mathbf{fw}(bf)$ for some d and f.

(b) $\mathbf{b}_l(df)$ and $\mathbf{m}(db)$ interact,
i.e. $\mathbf{b}_l(df) \,|\, \mathbf{m}(db) \longrightarrow \mathbf{fw}(bf)$ for some d and f.

Notice either $\mathbf{b}_r(db)$ or $\mathbf{b}_l(df)$ should be a subterm of P since no combinator can generate $\mathbf{b}_r$ or $\mathbf{b}_l$. Now assume

$$P_0 \stackrel{\text{def}}{=} P \,|\, \mathbf{m}(ae) \xrightarrow{\tau} P'_1 \,|\, \mathbf{m}(ae) \cdots \xrightarrow{\tau} P'_i \,|\, \mathbf{m}(ae) \xrightarrow{\tau} P_{i+1} \cdots \xrightarrow{\tau} P_n$$

with $b \notin \mathtt{an}_\downarrow(P_i)$ $(1 \le i \le n-1)$ and $b \in \mathtt{an}_\downarrow(P_n)$. By the above argument, we note:

- either $\langle \mathbf{b}_r(db),\ \mathbf{m}(df)\rangle$ or $\langle \mathbf{b}_l(df),\ \mathbf{m}(db)\rangle$ in P_{n-1} is needed for $\mathbf{fw}(bf)$ in P_n, and
- either $i = n-1$ or $\langle \mathbf{c}(a^-\tilde{v}),\ \mathbf{m}(ae)\rangle$ in $P'_i \,|\, \mathbf{m}(ae)$ is needed for $\mathbf{m}(df)$ or $\mathbf{m}(db)$ in P_{n-1} with $i < n-1$.

Case (a) $i = n-1$: We have either (1) $\langle \mathbf{c}(a^-\tilde{v}),\ \mathbf{m}(ae)\rangle = \langle \mathbf{b}_r(db),\ \mathbf{m}(df)\rangle$ or (2) $\langle \mathbf{c}(a^-\tilde{v}),\ \mathbf{m}(ae)\rangle = \langle \mathbf{b}_l(df),\ \mathbf{m}(db)\rangle$. For the case (1), with $d = a$ and $f = e$, we have $P \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(bc) \equiv P'_0 \,|\, \mathbf{b}_r(ab) \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(bc) \longrightarrow (P'_0 \,|\, \mathbf{fw}(be) \,|\, \mathbf{m}(bc))$ $\longrightarrow (\mathbf{m}(ec) \,|\, Q) \Downarrow_{e^\uparrow}$, which is a contradiction. The case (2) is impossible because e is chosen fresh.

Case (b) $1 \le i < n-1$: There are three cases. **(i)** Suppose $\mathbf{c}(a^-\tilde{v}) \equiv \mathbf{d}(aa_{21}a_{22})$ for some a_{21}, a_{22} (the case $\mathbf{c}(a\tilde{v}) \equiv \mathbf{fw}(aa_{21})$ amounts to this case, and $\mathbf{k}(a)$ is never needed by Fact 3.10 (i)). Then by definition for needed redex pairs, at least either $\mathbf{m}(a_{21}e)$ or $\mathbf{m}(a_{22}e)$ is needed again, and there exists a needed $\mathbf{c}_2(a_{2j}\tilde{v})$ in P_l $(i < l \le n-1)$ which should interact with $\mathbf{m}(a_{2j}e)$ $(j = 1,2)$. If again $\mathbf{c}_2(a_{2j}\tilde{v})$ is $\mathbf{d}(a_{2j}a_{31}a_{32})$, then again $\mathbf{m}(a_{31}e)$ or $\mathbf{m}(a_{32}e)$ is needed. Because the chain of needed redex pairs is always finite, either $\mathbf{b}_l(a_{kj}g)$ or $\mathbf{b}_r(a_{kj}g)$ is needed for some $i < k \le n-1$ since $\mathbf{m}(a_{kj}e)$ is forwarded just throughout a chain of $\mathbf{d}$ without changing the value e. The case $k = n-1$ is just the same with the case $i = n-1$ by replacing a with a_{kj}. So assume $k < n-1$. Then there are the two cases.

(1) Case $\mathbf{b}_l(a_{kj}g)$ is needed for some g: then $(\mathbf{b}_l(a_{kj}g) \,|\, \mathbf{m}(a_{kj}e)) \longrightarrow \mathbf{fw}(eg)$, hence $(P \,|\, \mathbf{m}(ae)) \Downarrow_{e^\downarrow}$, contradicting our assumption.

(2) Case $\mathbf{b}_r(a_{kj}g)$ is needed for some g: Then $\mathbf{b}_r(a_{kj}g) \,|\, \mathbf{m}(a_{kj}e) \longrightarrow \mathbf{fw}(ge)$. By definition of needed redex pairs, $\mathbf{fw}(ge)$ should be needed for $\mathbf{m}(df)$ or $\mathbf{m}(db)$ again. Thus there exists a needed $\mathbf{m}(gg')$ in $P_{k'}$ for some $k' < n-1$, and we get: $\mathbf{fw}(ge) \,|\, \mathbf{m}(gg') \longrightarrow \mathbf{m}(eg')$, hence $(P \,|\, \mathbf{m}(ae)) \Downarrow_{e^\uparrow}$, a contradiction again.

(ii,iii) Cases $\mathbf{c}(a^-\tilde{v}) \equiv \mathbf{b}_r(ag)$ and $\mathbf{c}(a^-\tilde{v}) \equiv \mathbf{b}_l(ag)$ for some g are just the same with the above cases (i-1) and (i-2), respectively. □

Now we can show $\mathbf{s}(abc)$ can not be represented by other atoms.

PROPOSITION 3.13. *$\mathbf{s}(abc)$ is essential.*

PROOF. First we note that P is a general synchroniser and $P \approx Q$ implies Q is a general synchroniser by Proposition 2.1. Assume $a \ne b$ and e is fresh. Then we know $\neg \mathbf{s}(abc) \Downarrow_{b^\downarrow}$ and $(\mathbf{s}(abc) \,|\, \mathbf{m}(ae)) \Downarrow_{b^\downarrow}$. Since $\mathbf{s}(abc) \xrightarrow{ae} \mathbf{fw}(bc) \xrightarrow{bf}$ $\mathbf{m}(cf) \xrightarrow{cf} \mathbf{0}$ is the only possible transition $\mathbf{s}(abc)$ can have, we get $\neg(\mathbf{s}(abc) \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(bc)) \Downarrow_{e^\downarrow}$, hence $\mathbf{s}(abc)$ is a general synchroniser at a from b. But by the previous lemma, we know there is no $P \in \mathbf{P_{cc}} \backslash \mathbf{s}$ such that $P \approx \mathbf{s}(abc)$, as desired. □

This result says that the prefix "$ax.P$" in π-calculus plays the role not only of binding x in P but also of *synchronising at a (and then activating P)*. See the next section for a study of the calculus with even less synchronisation. Now we reach the main theorem.

THEOREM 3.14. (**Minimality**) *$\mathbf{C}$ is a minimal basis. Hence $\mathbf{C}\backslash\mathbf{c} \lnsim \mathbf{C}$.*

PROOF. By Propositions 3.3, 3.5, 3.7 and 3.13 and Fact 2.3. □

Because $\mathbf{c}(\tilde{v}) \in \mathbf{C}$ can not be generated by other four combinators, it is not possible to be generated by other three, two, one and zero combinators in $\mathbf{C}$. Notice that we can prove the same statement of Lemma 3.2 even if we extend $\mathbf{C}\backslash\mathbf{c}$ to $\mathbf{C}\backslash\mathbf{c}_1\backslash\cdots\backslash\mathbf{c}_n$ $(0 \le i \le n \le 5)$. Hence we have:

COROLLARY 3.15. *$Y_1 \subsetneq Y_2 \subseteq \mathbf{C}$ implies $Y_1 \lnsim Y_2$, hence $\{Y^+ \mid Y \subseteq \mathbf{C}\}$ forms a complete lattice with $\Sigma_{0 \le n \le 5}\, {}_5\mathrm{C}_n = 32$ elements w.r.t $\lnsim$.*

3.6 Strong Minimality

In programming languages, a user sometimes wants to replace an existent primitive with another new primitive defined by him/herself, and delete the previous one without loss of expressive power. If a basis is minimal, we can automatically check the essentiality of a new primitive.

LEMMA 3.16. (exchange) *Suppose Y is a minimal basis and $Z \stackrel{\text{def}}{=} Y\backslash\{X\} \cup \{X'\}$ with $X \in Y$. Then there exists P s.t. $X \approx P \in Z^+$ iff Z is a minimal basis.*

PROOF. Suppose $X \approx P \in Z^+$. Since X is essential, there is no R s.t. $X \approx R \in (Y\backslash X)^+$. Therefore P should include $X'\sigma$ as its subterm with some renaming σ. Set $P \equiv C_n[X'\sigma_1]_1 \ldots [X'\sigma_n]_n$ for some $n \ge 1$ where C_n is a n-hole context with $C_n \in (Y\backslash X)^+$, and σ_i is a renaming function. Assume contradictorily we have $X' \approx Q \in (Y\backslash X)^+$. Then by Proposition 2.1 (i), $X \approx C_n[X'\sigma_1]_1 \ldots [X'\sigma_n]_n \approx C_n[Q\sigma_1]_1 \ldots [Q\sigma_n]_n \in (Y\backslash X)^+$, which is against the essentiality of X. □

We can replace $\mathbf{s}(abc)$ with a *message synchroniser* $\mathbf{s}_m(abc) \stackrel{\text{def}}{=} ax.\overline{b}c$.

PROPOSITION 3.17. (message synchroniser) $\mathbf{C}_m \stackrel{\text{def}}{=} \mathbf{C}\backslash\mathbf{s} \cup \{\mathbf{s}_m(abc)\}$ *is a minimal basis.*

PROOF. By $\mathbf{s}(abc) \approx (\nu e)(\mathbf{s}_m(aeb) \mid \mathbf{b}_l(ec))$ and the previous lemma. □

Note $ax.by.\mathbf{0} \approx (\nu c)\mathbf{s}(abc)$ is a general synchroniser in the sense of Definition 3.11. However, this process together with 4 atoms except $\mathbf{s}(abc)$ can not generate the whole π-calculus: if we diminish any atom in $\mathbf{C}$ by name-hiding, it is no longer a basis for $\mathbf{P}_\pi$.

THEOREM 3.18. **(strong minimality of C)** *Let* $Y \stackrel{\text{def}}{=} (\mathbf{C}\backslash\mathbf{c}) \cup_{1\le i\le n} P_i$ *with* $\mathbf{c}(\tilde{a}) \in \mathbf{C}$, $P_i \stackrel{\text{def}}{=} (\nu\tilde{b}_i)\mathbf{c}(\tilde{a})$ *and* $\emptyset \neq \{\tilde{b}_i\} \subseteq \{\tilde{a}\}$ *for some* $\tilde{b}_i$ *and* n. *Then* Y *is not a basis.*

PROOF. We already know that $\mathbf{0}$, $\mathbf{k}(ab)$ and $\mathbf{fw}(ab)$ are not essential by subsection 2.3, so, by cutting off trivial cases, we only have to check the following three sets are not basses.

(1) $Y_1 \stackrel{\text{def}}{=} \mathbf{C}\backslash\mathbf{m}(ab) \cup \{(\nu b)\mathbf{m}(ab)\}$ with $a \neq b$.

(2) $Y_2 \stackrel{\text{def}}{=} \mathbf{C}\backslash\mathbf{b}_l(ab) \cup \{(\nu b)\mathbf{b}_l(ab)\}$ with $a \neq b$ (note $ax.xy.\mathbf{0} \approx (\nu b)\mathbf{b}_l(ab)$).

(3) $Y_3 \stackrel{\text{def}}{=} \mathbf{C}\backslash\mathbf{s}(abc) \cup \{(\nu c)\mathbf{s}(abc)\}$ with $a \neq b$ (note $ax.by.\mathbf{0} \approx (\nu c)\mathbf{s}(abc)$).

By Lemma 3.16, we show $Y_{1,2,3}$ can not generate $\mathbf{m}(ab)$, $\mathbf{b}_l(ab)$ and $\mathbf{s}(abc)$, respectively. (1) Suppose $P \approx Q \in Y_1^+$. Then for all P' s.t. $P \longrightarrow\!\!\!\!\!\rightarrow P'$, P' can not include free names as objects of messages. Hence $P \stackrel{\overline{a}b}{\Longrightarrow} P'$ is not possible for all a and b, while $\mathbf{m}(ab) \xrightarrow{\overline{a}b} \mathbf{0}$.

(2) Suppose $\mathbf{b}_l(ab) \approx P \in Y_2^+$. Then if $P \stackrel{ae}{\Longrightarrow} P_0 \stackrel{ee'}{\Longrightarrow} P_1$ with e, e' fresh, then P should include $(\nu b)\mathbf{b}_l(a'b)$ as its subterm by Lemma 3.6 (i), and we can write $P_0 \longrightarrow\!\!\!\!\!\rightarrow (\tilde{c})((\nu b)ey.\overline{b}y \mid R)$. By the form of the term, we know, for any f, $P_1 \stackrel{\overline{f}e'}{\Longrightarrow} P'$ is impossible, while $\mathbf{b}_l(ab) \xrightarrow{ae}\xrightarrow{ee'}\xrightarrow{\overline{b}e'} \mathbf{0}$.

(3) We assume $P \in Y_3^+$ and $P \approx \mathbf{s}(abc)$. Then by the same reasoning in the proof of Lemma 3.12, P should include $(\nu c)\mathbf{s}(dbc)$ $(\approx dx.by.\mathbf{0})$ as its subterm. But this time, with e, e' fresh, $\mathbf{s}(abc) \xrightarrow{ae}\xrightarrow{be'}\xrightarrow{\overline{c}e'} \mathbf{0}$, while the third transition of $P \stackrel{ae}{\Longrightarrow}\stackrel{be'}{\Longrightarrow}\stackrel{\overline{c}e'}{\Longrightarrow} P'$ is clearly impossible. □

Thus five atoms are not only essential in the sense of Definition 3.1, but also have indeed "atomic" properties in that we can not reduce its syntax further.

REMARK 3.19. (minimum number) It is uninteresting to discuss just what is the minimum number of concurrent combinators rather than what combinators are indeed essential. Consider the following term.

$$\mathbf{Big}(abcdefghijkl) \stackrel{\text{def}}{=} \mathbf{m}(ab) \mid \mathbf{d}(cde) \mid \mathbf{b}_r(fg) \mid \mathbf{b}_l(hi) \mid \mathbf{s}(jkl)$$

Then we can easily guess that the whole asynchronous π-calculus can be generated only from this term since we can generate any atom from it by using name hiding, e.g. $\mathbf{m}(ab) \approx (\nu cfhj)\mathbf{Big}(abcdefghijkl)$. However it can not be said that this term is a combinatorial representation because it has several active names, and then a reduction rule between them is not fixed. In addition, this is not minimum in the sense of Theorem 3.18.

3.7 Minimality under Other Behavioural Equivalences

In the following subsection, we shall show the minimality theorem is also proved even if we replace the synchronous bisimulation $\approx$ with (1) the asynchronous bisimulation $\approx_a$, (2) the synchronous maximum sound theory $=_s$ and (3) the asynchronous maximum sound theory $=_a$. In the case of (1) and (3), the most difficult case is to prove the essentiality of $\mathbf{s}(abc)$. For the proof in the case of $\approx$, we used the following fact:

if $P \approx Q$ and P is a general synchroniser at a to b,
then Q is the same general synchroniser.

But it is not satisfied if we replace $\approx$ with either $\approx_a$ or $=_a$ since the input action cannot be observed in the asynchronous equivalences. Hence Main Lemma is not directly applicable. However the same proof technique based on the concurrent combinators will be used in the following way; first we will show *there is no general message synchroniser* in $\mathbf{P_{cc}}\backslash\mathbf{s}$ with a similar proof reasoning as Main Lemma, then the following fact will be used instead of the above.

if $P \approx_a Q$ and P is a general message synchroniser at a to b,
then Q is the same general message synchroniser.

Finally by Lemma 3.16, the essentiality of $\mathbf{s}_m$ will be exchanged by that of $\mathbf{s}$.

On the other hand, in the cases of (2) and (3), we can not observe the value of messages. But we can again prove both minimality and strong minimality using another concurrent combinator called *switcher* [24].

Minimality on the Asynchronous Bisimilarity

First $\approx_a$ is a congruent relation [16], and if $P \approx_a Q$ then $P \Downarrow_{a\uparrow} \;\Leftrightarrow\; Q \Downarrow_{a\uparrow}$. Then essentiality of $\mathbf{m}(ab)$, $\mathbf{d}(abc)$ and $\mathbf{b}_r(ab)$ are obvious by the previous lemmas. Note, for the essentiality of $\mathbf{b}_l(ab)$, we used increment of input observation (Lemma 3.6 (i)), which does not seem to work for the case of $\approx_a$. But we can again use this property to prove the following result.

PROPOSITION 3.20. $\mathbf{b}_l(ab)$ *is essential.*

PROOF. By Lemma 3.6 (ii), we first note if $P \in \mathbf{P_{cc}}\backslash\mathbf{b}_l$ and $e \notin \mathsf{fs}_\downarrow(P)$, then for all P' s.t. $(P \mid \mathbf{m}(ec)) \longrightarrow\!\!\!\!\!\rightarrow P'$, we have $P' \Downarrow_{e\uparrow}$. Suppose $\mathbf{b}_l(ab) \approx_a P \in \mathbf{P_{cc}}\backslash\mathbf{b}_l$ and e fresh. Then we have $\mathbf{b}_l(ab) \mid \mathbf{m}(ae) \mid \mathbf{m}(ec) \approx_a P \mid \mathbf{m}(ae) \mid \mathbf{m}(ec)$. Now we know $\mathbf{b}_l(ab)\,|\,\mathbf{m}(ae)\,|\,\mathbf{m}(ec) \longrightarrow^2 \mathbf{m}(bc)$ and $\neg\mathbf{m}(bc) \Downarrow_{e\uparrow}$. By assumption, there exists P' such that $P\,|\,\mathbf{m}(ae)\,|\,\mathbf{m}(ec) \longrightarrow\!\!\!\!\rightarrow P'$ with $\mathbf{m}(bc) \approx_a P'$, hence $\neg P' \Downarrow_{e\uparrow}$. But this is a contradiction. □

The most difficult case in this asynchronous bisimulation is the essentiality of $\mathbf{s}(abc)$ again. Instead of proving this directly, we show a message synchroniser $\mathbf{s}_m(abc) \stackrel{\text{def}}{=} ax.\overline{b}e$ can not generated without $\mathbf{s}(abc)$.

DEFINITION 3.21.

(i) (switcher) Let us define $\mathbf{sw}(ab) \stackrel{\text{def}}{=} ax.\overline{x}b$. Then $\mathbf{sw}(ab)\,|\,\mathbf{m}(av) \longrightarrow \mathbf{m}(vb)$. We call $\mathbf{sw}(ab)$ *switcher*. We also denote $\mathbf{sw}(a)$ for $(\nu\, b)\mathbf{sw}(ab)$.

(ii) Let $a \neq b$. A *general message synchroniser from a to b* is a term P such that (1) $\neg P \Downarrow_{b\uparrow}$, (2) $P\,|\,\mathbf{m}(ae) \Downarrow_{b\uparrow}$, (3) $\neg(P\,|\,\mathbf{m}(ae)\,|\,\mathbf{sw}(b)) \Downarrow_{e\uparrow}$ where e is fresh in (2) and (3).

A switcher switches a received value to a subject of message, and it is generated without $\mathbf{s}(abc)$ (e.g. $\mathbf{sw}(ab) \approx (\nu\, c)(\mathbf{b}_r(ac)\,|\,\mathbf{m}(cb))$). We can easily check none of $\mathbf{m}(ab)$, $\mathbf{d}(abc)$, $\mathbf{b}_l(ab)$ and $\mathbf{b}_r(ab)$ is a general message synchroniser.

PROPOSITION 3.22.

(i) **(main lemma)** $\mathbf{P_{cc}}\backslash\mathbf{s}$ *has no general message synchroniser up to* $\approx_a$.

(ii) $\mathbf{s}(abc)$ *is essential up to* $\approx_a$.

PROOF. (i) See Appendix C. (ii) By Proposition 3.17 and Remark 2.10, $\mathbf{C}_m$ is a basis up to $\approx_a$, and by (i), $\mathbf{s}_m(abc)$ is essential (up to $\approx_a$) w.r.t. $\mathbf{C}_m$ [since: if P is a general message synchroniser from a to b and $P \approx_a Q$, then Q is the same general message synchroniser, and obviously $\mathbf{s}_m(abc)$ is a general message synchroniser]. Note $\mathbf{s}_m(abc) \approx_a (\nu\, e)(\mathbf{s}(aeb)\,|\,\mathbf{m}(ec))$ with e fresh. Then by applying Lemma 3.16 again, we now know $\mathbf{s}(abc)$ is essential w.r.t. $\mathbf{C}$. □

Minimality under the Synchronous/Asynchronous Maximum Sound Theories

First all propositions except Proposition 3.5, we did not use observation of values for proofs. Thus all essentialities except $\mathbf{d}(abc)$ are obviously obtained by preceeding propositions.

PROPOSITION 3.23. $\mathbf{d}(abc)$ *is essential up to* $=_s$ *and* $=_a$.

PROOF. We prove the esseintiality up to $=_a$. The case of $=_s$ is just the same. First by Theorem 3.19 (Observability Theorem) in [24], if $P =_a Q$ and $P \overset{\uparrow a}{\rightsquigarrow} P'$, then there exists Q', s.t. $Q \longrightarrow\!\!\!\!\rightarrow \overset{\uparrow a}{\rightsquigarrow} \longrightarrow\!\!\!\!\rightarrow Q'$ and $P' =_a Q'$ where $\overset{\uparrow a}{\rightsquigarrow}$ is an asynchronous transition relation which does not care a value of labels (cf.[24]). Now suppose $\mathbf{d}(abc) =_a P \in \mathbf{P_{cc}}\backslash\mathbf{d}$. Then with e fresh, $\mathbf{d}(abc)\,|\,\mathbf{m}(ae)\,|\,\mathbf{sw}(b)\,|\,\mathbf{s}(c) \longrightarrow\!\!\!\!\rightarrow \overset{\uparrow e}{\rightsquigarrow} \longrightarrow\!\!\!\!\rightarrow \overset{\uparrow e}{\rightsquigarrow} \mathbf{0}$. But two output transitions to e from $(P\,|\,\mathbf{m}(ae)\,|\,\mathbf{sw}(b)\,|\,\mathbf{s}(c)) \in \mathbf{P_{cc}}\backslash\mathbf{d}$ is impossible by Lemma 3.4 (i). □

Note for the essentiality of $\mathbf{s}(abc)$ up to $=_s$ (resp. $=_a$), we can directly use the proofs of the non-existence of a general synchroniser (resp. message synchroniser) because Definition 3.11 (resp. Definition 3.21 (ii)) was given based on the synchronous (resp. asynchronous) convergence predicate.

Finally we have:

THEOREM 3.24. **(Minimality)** $\mathbf{C}$ *is a minimal basis up to* $\approx_a$, $=_s$ *and* $=_a$.

This result makes the essentiality of each combinator stronger: in both asynchronous and synchronous, and both labelled transition-based and reduction-based semantics, we can not miss any one of 5 combinators to generate the whole π-calculus. Further, we have:

THEOREM 3.25. **(strong minimality of C)** *Let* $Y \stackrel{\text{def}}{=} (\mathbf{C}\backslash\mathbf{c}) \cup_{1\leq i\leq n} P_i$ *with* $\mathbf{c}(\tilde{a}) \in \mathbf{C}$, $P_i \stackrel{\text{def}}{=} (\nu\, \tilde{b}_i)\mathbf{c}(\tilde{a})$ *and* $\emptyset \neq \{\tilde{b}_i\} \subseteq \{\tilde{a}\}$ *for some* $\tilde{b}_i$ *and* n. *Then* Y *is not a basis up to either* $\approx_a$, $=_s$ or $=_a$.

PROOF. We only show the case for $=_a$. Hereafter (1), (2) and (3) stand for the same cases (1), (2) and (3) in the proof of Theorem 3.18. For (1), if $P \in Y_1^+$, we have the same property with Lemma 3.6 (ii) because there is no free values of messages. Hence done. For (2), we use the context $C[\;] \stackrel{\text{def}}{=} [\;]\,|\,\mathbf{m}(ae)\,|\,\mathbf{m}(ee')\,|\,\mathbf{sw}(b)$ with e, e' fresh, while for (3), we use the context $C[\;] \stackrel{\text{def}}{=} [\;]\,|\,\mathbf{m}(ae)\,|\,\mathbf{m}(be')\,|\,\mathbf{sw}(c)$ with e, e' fresh. The detailed reasoning is left to the reader. □

4 Measuring Expressiveness of Subsystems of π-calculus (1)

This section measures expressive power of interesting subsystems of the asynchronous π-calculus by concurrent combinators, focusing our attention on three key elements of name-passing computation. First we study *locality* by introducing the *local* π*-calculus* [21, 6, 2, 32] in which no value is instantiated with input-subjects. Next we examine *sharing of names* by studying the *linear* and *affine* π*-calculi* where the number of free names is not changed or decreased during communications. Finally we consider *synchronisation* by formulating the *commutative* π*-calculus* which has more asynchrony than the asynchronous π-calculus. To examine their expressive power, we first decompose their computational behaviours (i.e. prefixes) into the corresponding systems of combinators. They are generated by a proper subset of $\mathbf{C}$ (in some case with refinement), hence have strictly less power than the whole asynchronous π-calculus by the results in Section 3. The proof method shows how we can use combinators as a tractable and informative tool to analyse the concurrent communication protocols. We begin with the formulation of *separation.*

DEFINITION 4.1. (separation) Assume P is essential w.r.t. Y and $X \lesssim Y\backslash\{P\}$. Then we say a subsystem $\mathbf{P} = \{Q \mid Q \approx R \in X^+\}$ is *separated by* P from a subsystem $\mathbf{P}' = \{Q \mid Q \approx R \in Y^+\}$. We also say $\mathbf{P}$ is a *proper subsystem* of $\mathbf{P}'$.

By the main theorem and lemma 3.2, we have

LEMMA 4.2. (separation) *The maximum set separated by* $\mathbf{c}$ *from* $\mathbf{P}_\pi$, *denoted by* $\mathbf{P}_{\backslash\mathbf{c}} = \{P \mid P \approx Q \in (\mathbf{C}\backslash\mathbf{c})^+\}$, *is a proper subsystem of* $\mathbf{P}_\pi$. *Moreover with* $\mathbf{c} \neq \mathbf{m}$, $\mathbf{P}_{\backslash\mathbf{c}}$ *is a t-subsystem.*

4.1 Local π-calculus

The asynchronous π-calculus was originally considered as a simple formal system for concurrent object-based computation with asynchronous communication [22, 23, 21], regarding $\overline{a}v$ as a pending message and $ax.P$ as a waiting object. But it includes a non-local future which is prohibited in most of object-oriented languages, cf.[21]. Consider the following example.

$$(\nu b)(\overline{a}b \mid bx.P) \mid ax.xy.Q \longrightarrow (\nu b)(bx.P \mid by.Q)$$

The left hand-side process represents an object which will send the object id b to another object. After communication, the other object with the same id b is created, violating the standard manner of the uniqueness of object id. To avoid such a situation in a simple way, we restrict the grammar of receptors as follows.

$$ax.P \qquad (x \notin \mathsf{fs}_\downarrow(P))$$

We call this calculus *local* π*-calculus* (written π_l for short) and write $\mathbf{P}_l$ for the set of terms.[6]

Here we briefly observe that this system can be regarded as an independent powerful subsystem. First we note that it is a t-subsystem. Next by the same way in [22, 25], local polyadic input agent $a(\tilde{v}).P$ and output agent $\overline{a}[\tilde{v}].P$ can be encoded in (monadic) π_l-calculus.

$$\begin{aligned}\overline{a}[v_1..v_n].P &\stackrel{\text{def}}{=} (\nu c)(\overline{a}c \mid cz.(\overline{z}v_1 \mid cz.(.. \mid cz.(\overline{z}v_n \mid P)..)))\\ a(x_1..x_n).P &\stackrel{\text{def}}{=} ay.(\nu c)(\overline{y}c \mid cx_1.(\overline{y}c \mid ..(\overline{y}c \mid cx_n.P))))\end{aligned}$$

with z, c, and y all fresh. Local branching structures are embedded without instantiation of input subject following the technique of [22] again. One important remark here is that, using these encodings, the weak call-by-value λ-calculus can be simulated in this subsystem by slightly changing the encoding in [34] as

[6]Such a subset was already discussed independently in [21, 22, 6, 2, 32] (cf. [51]).

follows.

$$\begin{aligned}[\![x]\!]u &\stackrel{\text{def}}{=} (\nu q)(\overline{u}q \mid !\mathbf{fw}(qx))\\ [\![\lambda x.M]\!]u &\stackrel{\text{def}}{=} (\nu q)(\overline{u}q \mid !q(xw).[\![M]\!]w)\\ [\![MN]\!]u &\stackrel{\text{def}}{=} (\nu qr)(qy.rz.\overline{v}[zu] \mid [\![M]\!]q \mid [\![N]\!]r)\end{aligned}$$

Thus π_l-calculus enjoys the universal computational power up to ≈. More complex structures (OOPL, cf. [22, 57]) can be translated in this subsystem. But what is the exact difference between π and local-π? The next proposition gives us a simple answer.

PROPOSITION 4.3. $\mathbf{C}\backslash\mathbf{b}_l$ *is a minimal basis of* π_l*-calculus, hence we have* $\mathbf{P}_{\backslash\mathbf{b}_l} \simeq \mathbf{P}_l \simeq \mathbf{C}\backslash\mathbf{b}_l \lnsim \mathbf{C}$.

PROOF. Any $P \in \mathbf{P}_l$ can be decomposed to $\mathbf{C}_7\backslash\mathbf{b}_l$ by the same rules in Definition 2.6 without using (VIII), (XI), (XII) and (XIII), so the same statement as in Lemma 2.8 can be automatically proved. Hence the generation theorem as in Theorem 2.5 holds, and minimality and separation are given by Theorem 3.14. □

Thus, just by having the essentiality of a simple combinator $\mathbf{b}_l$, we know for sure that π-calculus includes non-local elements which can*not* be represented by any element in the local world (though the direct proof is not difficult either). Notice that the above not only proves minimality but also shows that $\mathbf{C}_7\backslash\mathbf{b}_l$ is a system of combinators for π_l-calculus: there is fully abstract correspondence between them. This and other observations indicate the local π-world forms a self-contained universe, so that π_l-calculus would be worth being studied as an independent calculus like λ*I*-calculus [4].

4.2 Linear and Affine π-calculi

The π-calculus has two elements to increase non-determinism during communication – *sharing of names* and *parallelism* as $ax.(P \mid Q) \mid \overline{a}v \longrightarrow P\{v/x\} \mid Q\{v/x\}$. Such elements are represented by $\mathbf{d}(abc)$ in a concise way. To closely look at two elements separately, we examine the following communication which gains only parallelism.

$$ax.(P \mid Q) \mid \overline{a}v \longrightarrow P\{v/x\} \mid Q$$

We introduce two subsystems of the asynchronous π-calculus by restricting the syntax of the prefix:

(a) $ax.P$ if $\sharp\langle P, x\rangle = 1$

(b) $ax.P$ if $\sharp\langle P, x\rangle = 0$ or 1.

These two subsystems are called *linear* and *affine* π-calculi (π_{Lin} and π_{Af} for short) and we denote $\mathbf{P}_{\mathrm{Lin}}$ and $\mathbf{P}_{\mathrm{Af}}$ for the sets of terms of π_{Lin} and π_{Af}-calculi, respectively. Note $\mathbf{P}_{\mathrm{Lin}} \subsetneq \mathbf{P}_{\mathrm{Af}} \subsetneq \mathbf{P}_{\pi}$.[7] Then a natural question is what expressiveness relation lies between with/without parallelism and/or sharing. In particular, is there any difference between linear and affine name-passing? For answering these questions, we also decompose prefixes of these calculi into a system of combinators. Since $\mathbf{d}(abc)$ cannot be used directly to represent non-sharing communication, we here introduce the following simple new combinator, called *1-distributer.*

$$\mathbf{d}_1(abc) \stackrel{\mathrm{def}}{=} (\nu\, d)ax.(\overline{b}x \,|\, \overline{c}d)$$

Intuitively this is similar with combinators $\mathrm{B} = \lambda xyz.x(yz)$ and $\mathrm{C} = \lambda xyz.(xz)y$ in linear and affine λ-calculi [12, 1]. $\mathbf{d}_1$ distributes two messages while forwarding only one value, hence this has the same parallelism as $\mathbf{d}$, but not sharing.

In the following, we first clarify the difference between parallelism and non-parallelism, introducing the notion of *parallel distributer.*

DEFINITION 4.4. *(parallel name passing)* Let us assume $a \neq b, c$. We say P is a *parallel distributer at a to b and c* if (1) $\neg P \Downarrow_{f\uparrow}$ for all f and (2) $(P \,|\, \mathbf{m}(ae)) \stackrel{l}{\Longrightarrow}\stackrel{l'}{\Longrightarrow}$ and $(P \,|\, \mathbf{m}(ae)) \stackrel{l'}{\Longrightarrow}\stackrel{l}{\Longrightarrow}$ where $l = \overline{b}e$ or $\overline{b}(e)$ and $l' = ce'$ or $\overline{c}(e')$ with $\mathsf{bn}(l) \cap \mathsf{bn}(l') = \emptyset$.

It is clear that $\mathbf{d}(abc)$ and $\mathbf{d}_1(abc)$ are parallel distributors at a to b and c.

Now we formulate causality of dependency on reduction relations by a sequence of needed redex pairs.

DEFINITION 4.5. (independence) Assume $P_0 \xrightarrow{\Delta_0} P_1 \xrightarrow{\Delta_1} P_2 \xrightarrow{\Delta_2} \cdots \xrightarrow{\Delta_{n-1}} P_n$ where $n \geq 1$ and $\mathbf{c}_{1,2}(\tilde{v}_{1,2}) \stackrel{\mathrm{def}}{\equiv} P_n/u_{1,2}$ with $u_1 \neq u_2$. We say a sequence of needed redex pairs $\Delta_{i_0} \succ \Delta_{i_1} \succ \cdots \succ \Delta_{i_m}$ for $\mathbf{c}_1(\tilde{v}_1)$ is *independent* from a sequence of needed redex pairs $\Delta_{j_0} \succ \Delta_{j_1} \succ \cdots \succ \Delta_{j_{m'}}$ for $\mathbf{c}_2(\tilde{v}_2)$ if, for all i_k and j_l, we have $i_k \neq j_l$ (i.e. $\Delta_{i_k} \not\succ \Delta_{j_l}$ and $\Delta_{j_l} \not\succ \Delta_{i_k}$).

In a word, two reduction sequences are independent when neither of needed sequences has any effect on computation by the other. As an example, suppose

$$P_0 \stackrel{\mathrm{def}}{\equiv} \mathbf{d}(abc) \,|\, \mathbf{m}(av) \,|\, \mathbf{b}_l(db) \,|\, \mathbf{m}(de)$$

has the following τ-action sequence.

$$\begin{array}{llll} P_0 & \xrightarrow{\Delta_0} & \mathbf{m}(bv) \,|\, \mathbf{m}(cv) \,|\, \mathbf{0} \,|\, \mathbf{b}_l(db) \,|\, \mathbf{m}(de) & \stackrel{\mathrm{def}}{\equiv} P_1 \\ & \xrightarrow{\Delta_1} & \mathbf{m}(bv) \,|\, \mathbf{m}(cv) \,|\, \mathbf{0} \,|\, \mathbf{fw}(be) \,|\, \mathbf{0} & \stackrel{\mathrm{def}}{\equiv} P_2 \\ & \xrightarrow{\Delta_2} & \mathbf{0} \,|\, \mathbf{m}(cv) \,|\, \mathbf{0} \,|\, \mathbf{m}(ev) \,|\, \mathbf{0} & \stackrel{\mathrm{def}}{\equiv} P_3 \end{array}$$

[7] π_{Lin} and π_{Af}-calculi include infinite behaviour like $!ax.\overline{b}x$ and $!\overline{a}e$, but do not include replication under prefix $ax.!P$ if $x \in \mathsf{fn}(P)$ (e.g. $ax.!\overline{b}x$) by definition.

Then the needed sequence Δ_0 for $\mathbf{m}(cv)$ is independent from the needed sequence Δ_1 for $\mathbf{fw}(be)$ in P_2 but it is not so from the needed sequence $\Delta_0 \succ \Delta_2$ for $\mathbf{m}(ev)$ in P_3.

LEMMA 4.6. *Let* $P \in \mathbf{P_{cc}} \backslash \mathbf{d}$ *and* $P \xrightarrow{\tau}{}^{+} P' \equiv (\nu\, \tilde{c})(\mathbf{c}_1(\tilde{v}_1) \,|\, \mathbf{c}_2(\tilde{v}_2) \,|\, R)$ *with* $\mathbf{c}_1(\tilde{v}_1)$ *and* $\mathbf{c}_2(\tilde{v}_2)$ *are in different occurrences in* P'*. Then any sequence of needed redex pairs for* $\mathbf{c}_1(\tilde{v}_1)$ *is independent from any of that for* $\mathbf{c}_2(\tilde{v}_2)$.

PROOF. First we note if all needed sequence for $\mathbf{c}_1(\tilde{v}_1)$ or $\mathbf{c}_2(\tilde{w}_2)$ is empty, then the lemma obviously holds. Suppose $\tilde{\Delta}_i \stackrel{\mathrm{def}}{=} \Delta_{i_0} \succ \Delta_{i_1} \succ \cdots \succ \Delta_{i_m}$ with $1 \leq i_0 < i_m \leq n-1$ is a needed sequence for $\mathbf{c}_1(\tilde{v}_1)$. In $\mathbf{P_{cc}} \backslash \mathbf{d}$, all reduction rules are in a form either $\mathbf{c}(a\tilde{v}) \,|\, \mathbf{m}(aw) \longrightarrow \mathbf{c}'(\tilde{w})$ or $\mathbf{c}(a\tilde{v}) \,|\, \mathbf{m}(aw) \longrightarrow \mathbf{0}$ (the latter is not needed reduction). Then if $\mathbf{c}(a\tilde{v})$ and $\mathbf{m}(aw)$ in Δ_{i_0} are needed for $\mathbf{c}'(\tilde{w})$ in P_{i_1}, then it is no longer possible that Δ_{i_0} becomes needed for another combinator $\mathbf{c}''(\tilde{w}')$ which is different from $\mathbf{c}'(\tilde{w})$. Hence either there is unique Δ_{i_l} s.t. $\Delta_{i_0} \succ \Delta_{i_l}$ with $l < k \leq n-1$, or $\Delta_{i_1} = \Delta_{i_m}$ by definition. Repeating this from 1 to m, we prove that $\tilde{\Delta}_i$ is independent from any needed chain of $\mathbf{c}_2(\tilde{v}_2)$, hence we are done. $\square$

LEMMA 4.7. $\mathbf{P_{cc}} \backslash \mathbf{d}$ *has no parallel distributer.*

PROOF. Suppose $P \in \mathbf{P_{cc}} \backslash \mathbf{d}$ has a parallel distributer at a to b and c. Then by Lemma 3.4 (ii), there exists Q' s.t. $(P \,|\, \mathbf{m}(ae)) \xrightarrow{\tau}{}^{+} Q' \equiv (\nu\, \tilde{c})(Q \,|\, \mathbf{m}(bf) \,|\, \mathbf{m}(cf'))$ for some f, f'. Assume $\tilde{\Delta}_i$ is a needed sequence for $\mathbf{m}(bf)$ and $\tilde{\Delta}_j$ is that for $\mathbf{m}(cf')$. Note $\mathbf{m}(ae)$ is needed for both $\mathbf{m}(bf)$ and $\mathbf{m}(cf')$ and the occurrence of $\mathbf{m}(ae)$ is unique because e is fresh. Hence we have $\Delta_{i_k} = \Delta_{j_l}$ for some Δ_{i_k} in $\tilde{\Delta}_i$ and Δ_{j_l} in $\tilde{\Delta}_j$, which contradicts Lemma 4.6. $\square$

Now we have

PROPOSITION 4.8.

(i) $\mathbf{C} \backslash \mathbf{d} \lesssim_{\not\approx} \mathbf{C}_{Af} \lesssim_{\not\approx} \mathbf{C}$ *with* $\mathbf{C}_{Af} \stackrel{\mathrm{def}}{=} \{\mathbf{d}_1(abc), \mathbf{m}(ab), \mathbf{b}_r(ab), \mathbf{b}_l(ab), \mathbf{s}(abc)\}$ *with* a, b, c *pairwise distinct.*

(ii) $\mathbf{C}_{Af}$ *is a minimal basis of* π_{Lin} *and* π_{Af}*-calculi, hence we have:* $\mathbf{P}_{\backslash \mathbf{d}} \lesssim_{\not\approx} \mathbf{P}_{Lin} \simeq \mathbf{P}_{Af} \lesssim_{\not\approx} \mathbf{P}_{\pi}$.

PROOF. For (i), first note that $\mathbf{d}_1(abc)$ is generated in $\mathbf{C}$ by Theorem 2.5. Hence we know $\mathbf{C}_{\mathrm{Af}} \lesssim \mathbf{C}$. Then by Lemma 3.4 (ii), we have $\mathbf{C}_{\mathrm{Af}} \lesssim_{\not\approx} \mathbf{C}$. $\mathbf{C} \backslash \mathbf{d} \lesssim \mathbf{C}_{\mathrm{Af}}$ is obvious, while $\mathbf{C} \backslash \mathbf{d} \lesssim_{\not\approx} \mathbf{C}_{\mathrm{Af}}$ is obtained by Lemma 4.7 because if $P \approx Q$ and Q is a parallel distributer, then P is also parallel distributer.

For (ii), we first observe that $\mathbf{C}_{\mathrm{Af}}$ is a basis for π_{Af}-calculus by checking that any π_{Af}-prefix is decomposed by replacing $\mathbf{d}$ with $\mathbf{d}_1$ in (I) and (XII) and adding the side condition $x \notin \mathsf{fn}(P)$ for (IV) in Definition 2.6. Then the minimality and $\mathbf{P}_{\backslash\mathbf{d}} \lnapprox \mathbf{P}_{\mathrm{Af}} \lnapprox \mathbf{P}_{\pi}$ are given by (i). For $\mathbf{P}_{\mathrm{Lin}} \simeq \mathbf{P}_{\mathrm{Af}}$, we have $\mathbf{P}_{\mathrm{Lin}} \lesssim \mathbf{P}_{\mathrm{Af}}$ by Fact 2.3 (i). For the converse inclusion, we note $\mathbf{s}(abc) \notin \mathbf{P}_{\mathrm{Lin}}$ but we have $\mathbf{s}(abc) \approx ax.by.(\overline{c}y \mid (\nu b)\overline{b}x)$. Then we use Lemma 3.16. □

REMARK 4.9.

- We have observed that $\mathbf{d}(abc)$ represents two roles in a concise way: sharing of names and increment of parallelism, and extraction of parallelism from it gives rise to two proper π-calculi. For further examination of parallelism, it is proved that 0-distributer $\mathbf{d}_0(abc) \stackrel{\text{def}}{=} ax.(\nu ee')(\overline{b}e \mid \overline{c}e')$ can not be generated in $\mathbf{P}_{\mathbf{cc}}\backslash\mathbf{d}$ and can not generate $\mathbf{d}_1$ by Proposition 4.7. More exactly, we have: $\mathbf{C}\backslash\mathbf{d} \lnapprox \mathbf{C}\backslash\mathbf{d} \cup \{\mathbf{d}_0(abc)\} \lnapprox \mathbf{C}_{\mathrm{Af}}$, but a proper subset generated by $\mathbf{C}\backslash\mathbf{d} \cup \{\mathbf{d}_0(abc)\}$ seems to have no interest.
- Causality of communication in π-calculus was studied based on parametric labelled transition systems in [13, 54, 7] from more general viewpoints. On the other hand, neededness and independence between sequences of reduction relations (τ-actions) in our concurrent combinators are simply defined without introducing additional information on labelled transition systems because the form of terms generated after one-step reduction is always fixed. A general study on true concurrency and non-interleaving semantics based on our combinators would be an interesting research topic.

REMARK 4.10. (affine local π-calculus) The finite affine local π-calculus whose minimal basis is

$$\mathbf{C}_{\mathrm{Af}l} \stackrel{\text{def}}{=} \{\mathbf{m}(ab), \mathbf{d}_1(abc), \mathbf{b}_r(ab), \mathbf{s}(abc)\} \text{ with } a,b,c \text{ pairwise distinct}$$

has an enough power (without replication) to embed *linear* and *affine* λ-calculi [12, 1] where substitution of a term is occurred only once or at most once. The former is inductively defined by the following rules (1) and (2), while the latter is only by (1) (drops restriction of abstraction) where $\mathcal{FV}(M)$ is a set of free variables in M.

(i) MN if $\mathcal{FV}(M) \cap \mathcal{FV}(N) = \emptyset$.
(ii) $\lambda x.M$ if $x \in \mathcal{FV}(M)$.

Their embedding is given based on [34] without replication.[8]

$$\begin{aligned}
[\![x]\!]u &\stackrel{\text{def}}{=} \mathbf{fw}(qx)\\
[\![\lambda x.M]\!]u &\stackrel{\text{def}}{=} (\nu w)u(x).\overline{u}[w].[\![M]\!]w\\
[\![MN]\!]u &\stackrel{\text{def}}{=} (\nu qr)(\overline{q}[r].q(x).\mathbf{fw}(ux) \mid [\![M]\!]q \mid [\![N]\!]r)
\end{aligned}$$

[8]This version does not evaluate under λ-body as usual encodings of λ-calculi in π-calculus but with $[\![\lambda x.M]\!]u \stackrel{\text{def}}{=} (\nu w)(u(z).\overline{u}[w].\mathbf{fw}(zx) \mid [\![M]\!]w)$, we can simulate the full linear and affine λ-calculi (this forwarder technique does not work if a variable appears more than twice in a λ-body).

More formal analysis related to linear typing systems on this subsystem, e.g.[59], is worth studying.

4.3 Commutative π-calculus

The asynchronous π-calculus was born by deleting output synchronisation from the synchronous π-calculus [34]. But what calculus is obtained if we further delete input synchronisation from the asynchronous π-calculus? This subsection studies synchronisation of π-calculus by introducing a more asynchronous π-calculus separated by a synchroniser. This calculus, which is called *commutative π-calculus* (π_c for short), allows communication by a process under prefix if there is no binding. We define π_c-calculus following the ideas in [9] and [36]. It is notable that π_c-calculus is not a subsystem of π-calculus because of additional structural rules; this makes direct comparison of its expressiveness difficult. But we can prove that π_c-calculus has less power than the asynchronous π-calculus by using the combinators again.

DEFINITION 4.11. (commutative π-calculus) We use the same syntax as in 2.1. for the syntax of π_c-calculus. Then the following two rules are added to the structural rules.

(1) $ax.(P \mid Q) \equiv ax.P \mid Q \quad (x \notin \mathsf{fn}(Q))$
(2) $ax.by.P \equiv by.ax.P \quad (x \neq b,\ y \neq a)$

We denote $\mathbf{P}_{\pi_c}$ for the set of π_c-terms. $\longrightarrow$ is defined in the same way as in the asynchronous π-calculus and $\xrightarrow{l}$ is given in Appendix C. Then we write $\approx_c$ for a weak bisimilarity for π_c-calculus.[9]

The first structural rule (1) is found in [9], while the second one (2) comes from [36]. Notice that in any strong and weak semantics, we have $ax.by.P \not\approx by.ax.P$ in π-calculus. An example of reduction of π_c-calculus (with $x \neq a$ and $b \neq y$) is:

$$\begin{aligned}
\overline{a}v \mid bx.ay.\overline{a}w &\equiv \overline{a}v \mid ax.by.\overline{a}w && \text{(by (2) in Def.4.11)}\\
&\equiv \overline{a}v \mid ax.by.(\mathbf{0} \mid \overline{a}w) && \text{(by } P \mid \mathbf{0} \equiv P\text{)}\\
&\equiv \overline{a}v \mid ax.(by.\mathbf{0} \mid \overline{a}w) && \text{(by (1) in Def.4.11)}\\
&\equiv \overline{a}v \mid ax.by.\mathbf{0} \mid \overline{a}w && \text{(by (1) in Def.4.11)}\\
&\longrightarrow \overline{a}v \mid by.\mathbf{0}
\end{aligned}$$

We have another possible reduction from the second line like:

$$\overline{a}v \mid ax.by.\overline{a}w \longrightarrow by.\overline{a}w \equiv by.\mathbf{0} \mid \overline{a}w$$

It seems impossible to construct a synchroniser satisfying Definition 3.11 since we have $ax.by.\overline{c}y \equiv ax.\mathbf{0} \mid by.\overline{c}y$. But how can we prove this? First we observe that to represent π_c-calculus by combinators, $\mathbf{b}_r(ab)$ cannot be directly used because

[9]$\approx_c$ is defined up to $\equiv$, as shown in Appendix C.

$ax.by.\overline{x}y \equiv by.ax.\overline{x}y$ in π_c-calculus but $\mathbf{b}_r(ab) \not\approx \mathbf{b}_r(ba)$. This commutation on π_c-prefixes, however, is faithfully represented by a *commutative version of a right binder* of the asynchronous π-calculus, defined by:

$$\mathbf{b}^c_r(ab) \stackrel{\text{def}}{=} (\nu\, c_1c_2)(\mathbf{fw}(ac_1) \,|\, \mathbf{fw}(bc_2) \,|\, \mathbf{b}_r(c_1c_2))$$

Note $\mathbf{b}^c_r(ab) \approx \mathbf{b}^c_r(ba) \in \mathbf{P_{cc}} \backslash \mathbf{s}$. Now set $\mathbf{C}_c \stackrel{\text{def}}{=} \{\mathbf{m}(ab), \mathbf{d}(abc), \mathbf{b}^c_r(ab), \mathbf{b}_l(ab)\}$ with a,b,c pairwise distinct. Then we can show $\mathbf{C}_c$ is a set of combinators of π_c-calculus, just as $\mathbf{C}$ is for the asynchronous π-calculus. This is proved by *commutative prefix mapping* $a^\star x.P$ in $\mathbf{P}_c$ defined in the following.

DEFINITION 4.12. Set $\mathbf{P}_c \stackrel{\text{def}}{=} (\mathbf{C}_7 \backslash \{\mathbf{b}_r, \mathbf{s}\} \cup \{\mathbf{b}^c_r(ab), \mathbf{b}^c_r(aa)\})^+$. Then the *commutative prefix mapping* $u^\star x.P : \mathbf{N} \times \mathbf{N} \times \mathbf{P}_c \to \mathbf{P}_c$ is given by simply changing (V,VI), (IX) and (XII) as follows, deleting (XIII) and replacing $u^* x.P$ with $u^\star x.P$ in other rules in Definition 2.6.

(V,VI).	$a^\star x.\mathbf{c}(\tilde{w}) \stackrel{\text{def}}{=} (\nu\, c)(\mathbf{k}(a) \,|\, \mathbf{c}(\tilde{w}))$	$x \notin \{\tilde{w}\}$
(IX).	$a^\star x.\mathbf{fw}(bx) \stackrel{\text{def}}{=} \mathbf{b}^c_r(ab)$	$x \neq b$
(XII).	$a^\star x.\mathbf{b}^c_r(bx^-) \stackrel{\text{def}}{=} (\nu\, c)(\mathbf{fw}(ac) \,|\, \mathbf{b}^c_r(bc))$	$x \neq b$

Define $[\![\;]\!]_\mathbf{c} : \mathbf{P}_{\pi_c} \to \mathbf{P}_c$ with $[\![ax.Q]\!]_\mathbf{c} \stackrel{\text{def}}{=} a^\star x.[\![Q]\!]_\mathbf{c}$, $[\![\;]\!]_{\pi_c} : \mathbf{P}_c \to \mathbf{P}_{\pi_c}$ with $[\![\mathbf{b}^c_r(ab)]\!]_{\pi_c} \stackrel{\text{def}}{=} ax.by.\overline{x}y$, plus homomorphic mappings (see Appendix C). Now we can derive non-synchronisation of π_c-calculus throughout concurrent combinators.

PROPOSITION 4.13.

(i) *π_c-calculus has no synchroniser which satisfies Definition 3.11.*

(ii) (full abstraction) (1) $[\![[\![P]\!]_\mathbf{c}]\!]_{\pi_c} \approx_c P$ and $[\![[\![P]\!]_{\pi_c}]\!]_\mathbf{c} \approx P$.
(2) $P \approx_c Q \;\Leftrightarrow\; [\![P]\!]_\mathbf{c} \approx [\![Q]\!]_\mathbf{c}$ and $P \approx Q \;\Leftrightarrow\; [\![P]\!]_{\pi_c} \approx_c [\![Q]\!]_{\pi_c}$.

(iii) $\mathbf{P}_{\backslash\{\mathbf{b}_r,\mathbf{s}\}} \lnapprox \mathbf{C}_c \simeq \mathbf{P}_c \lnapprox \mathbf{P}_{\backslash\mathbf{s}}$.

PROOF. First we check that $a^\star x.[\![P \,|\, Q]\!]_\mathbf{c} \approx a^\star x.[\![P]\!]_\mathbf{c} \,|\, [\![Q]\!]_\mathbf{c}$ with $x \notin \mathsf{fn}(Q)$ and $a^\star x.b^\star y.[\![Q]\!]_\mathbf{c} \approx b^\star y.a^\star x.[\![Q]\!]_\mathbf{c}$ with $x \neq b$ and $y \neq a$ by induction on terms. Then we can prove, for all $Q \in \mathbf{P}_{\pi_c}$, $Q \Downarrow_{a\downarrow} \;\Leftrightarrow\; [\![Q]\!]_\mathbf{c} \Downarrow_{a\downarrow}$ by Proposition D.2. Suppose towards a contradiction $P \in \mathbf{P}_{\pi_c}$ is a synchroniser from a to b. Then $\neg P \Downarrow_{b\downarrow}$ implies $\neg [\![P]\!]_\mathbf{c} \Downarrow_{b\downarrow}$, and $(P \,|\, \overline{a}e) \Downarrow_{b\downarrow}$ implies $([\![P \,|\, \overline{a}e]\!]_\mathbf{c}) \stackrel{\text{def}}{=} ([\![P]\!]_\mathbf{c} \,|\, \mathbf{m}(ae)) \Downarrow_{b\downarrow}$ with e fresh. Since $[\![P]\!]_\mathbf{c} \in \mathbf{P}_c \subsetneq \mathbf{P_{cc}} \backslash \mathbf{s}$, we have $[\![P]\!]_\mathbf{c} \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(bc) \Downarrow_{e\downarrow}$ by Lemma 3.12, then $P \,|\, \overline{a}e \,|\, \overline{b}c \Downarrow_{e\downarrow}$, which contradicts our assumption. (ii) is left to Appendix C. The first proper inclusion is done by Lemma 3.6 (i), while the second proper inclusion is proved by showing that $\mathbf{b}_r(ab)$ can not be generated in $\mathbf{C}_c$, whose proof we leave to Appendix C. □

Note $\mathbf{P}_1 \stackrel{\text{def}}{=} \{Q \mid Q \approx [\![P]\!]_\mathbf{c}\}$ is a t-subsystem, and $\mathbf{P}_1 \simeq \mathbf{C}_c$. Hence the behaviour of π_c-calculus is exactly simulated in the asynchronous π-calculus without $\mathbf{s}$. In addition, π_c-calculus has a combinatorial representation with $\{\mathbf{m}(ab), \mathbf{d}(abc), \mathbf{b}_r(ab), \mathbf{b}_l(ab)\}$ $(a,b,c$ pairwise distinct) as its subsystem.

REMARK 4.14. Two additional structural rules in π_c-calculus represent more asynchronous computation we can obtain from the asynchronous π-calculus without synchroniser in a simple way. On the other hand, in the framework of action structures [36], a more asynchronous calculus called *reflexive π-calculus* is studied and a family of π-calculi is defined by adding appropriate control structures one by one based on it.[10] Roughly the essential π-calculus does not impose *any* sequencing, but only identification between two names. Such a general connection itself is difficult to be represented directly as π-syntax, but we can define (a version of) this system as a subset of the asynchronous π-calculus, following the idea found in Example 6.4 in [18]. Define the encoding of reflexive π-terms by as subterms of $\mathbf{P}_\pi$ generated by:

$$P \;::=\; \overline{a}b \mid\; ab.\mathbf{eq}(cb) \mid P \,|\, Q \mid\; (\nu\, a)P \mid \mathbf{eq}(ab) \mid \mathbf{0}$$

where $\mathbf{eq}(ab) \stackrel{\text{def}}{=} !\mathbf{fw}(ab) \,|\, !\mathbf{fw}(ba)$ is an *equator* originally introduced in [24]. Note $ab.\mathbf{eq}(cb)$ is generated by $\mathbf{d}(abc)$, $\mathbf{b}_r(ab)$ and $\mathbf{b}_l(ab)$, hence it seems evident this system can not have a machinery of synchroniser. Now we write $P[b/a]$ for $(\nu\, a)(P \,|\, \mathbf{eq}(ab))$, and $a(x)P$ for $(\nu\, x)(ay.\mathbf{eq}(yx) \,|\, P)$ with y fresh. Then we have:

$$a(x)P \,|\, \overline{a}b \;\longrightarrow\; P[b/x] \tag{1}$$

P above can interact with outside processes even before substitution, hence synchronisation is even less than π_c-calculus (e.g. consider a term $a(x)(\overline{x}y \,|\, x(y)))$. Note in the maximum sound equality, we have

$$P[b/a] =_a P\{b/a\}$$

(cf. Proposition 4.3 in [24]), so that the same result holds as in Proposition 4.13. Thus this subset of the asynchronous π-calculus can simulate a reflexive behaviour found in [36] up to $=_a$.[11] See Remark 5.11 for the examination of its expressive power related to encodings.

5 Measuring Expressiveness of Subsystems of π-calculus (2)

In the family of both synchronous and asynchronous π-calculi, the expressive power is often measured by *encoding* between two systems, which is either fully

[10] A similar calculus which reduces under prefixes was also discussed in [9]. Its local version is studied in [32].

[11] We can use $\mathbf{b}^c_r(ab)$ instead of $\mathbf{b}_r(ab)$ to simulate (1) up to $=_a$. Hence π_c-calculus can simulate this essential π-calculus as we expect.

abstract (i.e. $[\![P]\!] \approx [\![Q]\!] \Leftrightarrow P \approx Q$) or adequate (i.e. $[\![P]\!] \approx [\![Q]\!] \Rightarrow P \approx Q$) [25, 22, 40, 44, 6]. One of the most intriguing questions related to our present study in this context is: *if we miss any one of 5 combinators, i.e. in any proper subsystem of* $\mathbf{C}$, *is it absolutely impossible to construct any "good" encoding of* $\mathbf{P}_\pi$*?* This section shows the minimality theorem is applicable to derive several non-existence results of encodings: there is no uniform, reasonable [44], reduction-closed [24, 52] encodings of the whole asynchronous π-calculus into (1) any proper subsystem of the asynchronous π-calculus studied in Sections 3 and 4, assuming the message/transition preserving conditions, and (2) a proper subsystem without a message or without a duplicator (without any additional condition). (2) shows that *parallelism* can not be taken away to embed π-calculi.

First we introduce a new formulation of measuring expressive power based on encodings, extending our view to the whole π-family. Hereafter "subsystems" etc. denote those of the full polyadic synchronous π-calculus $\mathbf{P}_{\text{full}}$ (with match and mixed summation operators [35]), defined as in Definition 2.2 (iii).

5.1 Standard Encodings

We formulate the notion of *standard encoding* extending our view to the whole π-family then summarise the known results about encodings between π-family [22, 25, 6, 44, 40]. Hereafter "subsystems" etc. denote those of the full polyadic π-calculus $\mathbf{P}_{\text{full}}$ (with match and mixed summation operators [35]), defined as in Definition 2.2 (iii).

DEFINITION 5.1. (standard encoding) Let $\mathbf{P}_1$ and $\mathbf{P}_2$ be subsystems (of $\mathbf{P}_{\text{full}}$). A mapping $[\![\;]\!]$ from $\mathbf{P}_1$ to $\mathbf{P}_2$ is *standard* if it satisfies the following conditions.

(1) $[\![\;]\!]$ is homomorphic, i.e. $[\![P|Q]\!] \stackrel{\text{def}}{=} [\![P]\!]\,|\,[\![Q]\!]$, $[\![(\nu\tilde{c})P]\!] \stackrel{\text{def}}{=} (\nu\tilde{c})[\![P]\!]$, $[\![!P]\!] \stackrel{\text{def}}{=} ![\![P]\!]$ and $[\![P\sigma]\!] \stackrel{\text{def}}{=} [\![P]\!]\sigma$ with σ an injective renaming, and $[\![\mathbf{0}]\!] \stackrel{\text{def}}{=} \mathbf{0}$.

(2) $P \Downarrow_{a\updownarrow} \;\Leftrightarrow\; [\![P]\!] \Downarrow_{a\updownarrow}$.

(3) (a) $P \longrightarrow\!\!\!\!\rightarrow P' \;\Rightarrow\; [\![P]\!] \longrightarrow\!\!\!\!\rightarrow Q \approx [\![P']\!]$, and
(b) $[\![P]\!] \longrightarrow\!\!\!\!\rightarrow Q \;\Rightarrow\; \exists R.(P \longrightarrow\!\!\!\!\rightarrow R \wedge Q \approx [\![R]\!])$.

We say $\mathbf{P}_2$ *can embed* $\mathbf{P}_1$, written $\mathbf{P}_1 \lesssim^{\text{e}} \mathbf{P}_2$ if there is a standard encoding from $\mathbf{P}_1$ into $\mathbf{P}_2$, and $\mathbf{P}_2$ *properly embed* $\mathbf{P}_1$, written $\mathbf{P}_1 \lnsim^{\text{e}} \mathbf{P}_2$ if both $\mathbf{P}_1 \lesssim^{\text{e}} \mathbf{P}_2$ and $\mathbf{P}_2 \not\lesssim^{\text{e}} \mathbf{P}_1$. We also denote $\simeq^{\text{e}}$ for $\lesssim^{\text{e}} \cap (\lesssim^{\text{e}})^{-1}$.

(1) and (2) nearly correspond to uniform and reasonable conditions in [44] (but we do not require divergent-sensitivity). (3) describes the standard operational closure properties found in almost existing adequate encodings, cf. [34, 22, 40]. These conditions are general enough for reduction-based equivalences (e.g. sound theories) [24, 25, 3]. The usage of the synchronous action predicate and synchronous bisimilarity in (2) and (3) are natural when we consider the whole family of π-calculi because the full π-calculus is synchronous, while the asyn-

chronous notion of convergence now becomes more standard in asynchronous π-family [24, 3, 40]. We will study other standard encodings based on asynchronous bisimulation and sound equalities in Subsection 5.4. Note that: (1) $\mathbf{P}_1 \subseteq \mathbf{P}_2$ implies $\mathbf{P}_1 \lesssim^{\text{e}} \mathbf{P}_2$ with an identity mapping, and (2) $\lesssim^{\text{e}}$ is a preoder.

In a word, generation of a basis indicates how to *span* a core set of terms to represent the whole set up to semantic equality, while standard embedding formalises how to *bridge* two sets by a *homomorphic* mapping. More technically, if $\mathbf{P}_1 \lesssim \mathbf{P}_2$, then there is a fully abstract standard encoding as will be shown in Proposition 5.4, and $\mathbf{P}_1 \lesssim^{\text{e}} \mathbf{P}_2$ is related with an existence of an adequate encoding from $\mathbf{P}_1$ into $\mathbf{P}_2$ up to the reduction-based equalities (see Proposition 5.13 (ii)).

PROPOSITION 5.2. (relationship with Definition 2.2) *Let us assume a mapping* $[\![\;]\!]$ *from a subsystem* $\mathbf{P}_1$ *to a subsystem* $\mathbf{P}_2$. *Then:*

(i) *Suppose* $[\![\;]\!]$ *satisfies (1) and (3) in Definition 5.1. Then* $\{[\![P]\!] \mid P \in \mathbf{P}_1\}$ *is a subsystem up to* $\approx$.

(ii) *Suppose* $[\![\;]\!]$ *satisfies (1) in Definition 5.1 and the following condition.*

(a) $P \xrightarrow{l} P' \;\Rightarrow\; [\![P]\!] \stackrel{l}{\Longrightarrow} Q \approx [\![P']\!]$ and

(b) $[\![P]\!] \stackrel{\hat{l}}{\Longrightarrow} Q \;\Rightarrow\; \exists P'.(P \stackrel{\hat{l}}{\Longrightarrow} P' \wedge Q \approx [\![P']\!])$.

Then $[\![\;]\!]$ *is standard and adequate, i.e.* $[\![P]\!] \approx [\![Q]\!] \Rightarrow P \approx Q$. *Moreover* $\{[\![P]\!] \mid P \in \mathbf{P}_1\}$ *is a t-subsystem up to* $\approx$.

PROOF. All are mechanical except the adequacy of (ii). For the adequacy, we construct a relation $\mathcal{R}$ such that $P \simeq Q$ if $[\![P]\!] \approx [\![Q]\!]$ and show it is a weak bisimulation. Assume $[\![P]\!] \approx [\![Q]\!]$. Then:

$$\begin{array}{rlll}
P \xrightarrow{l} P' & \Rightarrow & [\![P]\!] \stackrel{l}{\Longrightarrow} P'' \wedge P'' \approx [\![P']\!] & \text{((a) in (ii))} \\
& \Rightarrow & [\![Q]\!] \stackrel{\hat{l}}{\Longrightarrow} Q'' \wedge Q'' \approx P'' \approx [\![P']\!] & (\text{by } [\![P]\!] \approx [\![Q]\!]) \\
& \Rightarrow & \exists Q'.(Q'' \approx [\![Q']\!] \text{ and } Q \stackrel{\hat{l}}{\Longrightarrow} Q') & \text{((b) in (ii))} \\
& \Rightarrow & P' \,\mathcal{R}\, Q' \quad \Box &
\end{array}$$

Notice we can not replace $[\![P]\!] \stackrel{l}{\Longrightarrow} Q$ in (ii-a) by $[\![P]\!] \stackrel{\hat{l}}{\Longrightarrow} Q$ as a similar reasoning found in the proofs in [6]. A more important fact on the relationship between $\lesssim^{\text{e}}$ and $\lesssim$ follows.

FACT 5.3. *Let* $\mathbf{P}_{1,2}$ *be subsystems. Then* $\mathbf{P}_1 \lesssim^{\text{e}} \mathbf{P}_2 \not\Rightarrow \mathbf{P}_1 \lesssim \mathbf{P}_2$.

PROOF. By Proposition 4.3 and Proposition 6.1 (ii) (see Section 6.2). $\Box$

However, as expected, we have:

PROPOSITION 5.4. *Assume* $\mathbf{P}_1$ *and* $\mathbf{P}_1$ *are subsystems and* $\mathbf{P}_1 \lesssim \mathbf{P}_2$*. Then there is a fully abstract standard mapping from* $\mathbf{P}_1$ *into* $\mathbf{P}_2$*. Hence we have* $\mathbf{P}_1 \lesssim^{\mathrm{e}} \mathbf{P}_2$*.*

PROOF. Here we write "(1,2,3)" to denote (1,2,3) in Definition 5.1. First by $\mathbf{P}_1 \lesssim \mathbf{P}_2$ and $\mathbf{P}_2^+ = \mathbf{P}_2$, for all $P_1 \in \mathbf{P}_1$, there exists $P_2 \in \mathbf{P}_2$ such that $P_1 \approx P_2$. Let us define $[P_1]_\approx \overset{\mathrm{def}}{=} \{P_2 \mid P_1 \approx P_2 \ \wedge\ P_i \in \mathbf{P}_i \ (i = 1,2)\}$. We define $\llbracket\ \rrbracket : \mathbf{P}_1 \to \mathbf{P}_2$ as

- $\llbracket \mathbf{0} \rrbracket \overset{\mathrm{def}}{=} \mathbf{0}$,
- If $P_1 \in \mathbf{P}_1$ is an input/output prefixed term (including a form of messages), then $\llbracket P_1 \rrbracket \overset{\mathrm{def}}{=} P_2 \in [P_1]_\approx$ for some P_2.
- $\llbracket P_1 \,|\, Q_1 \rrbracket \overset{\mathrm{def}}{=} \llbracket P_1 \rrbracket \,|\, \llbracket Q_1 \rrbracket$, $\llbracket (\nu\, a) P_1 \rrbracket \overset{\mathrm{def}}{=} (\nu\, a) \llbracket P_1 \rrbracket$, and $\llbracket !P_1 \rrbracket \overset{\mathrm{def}}{=} !\llbracket P_1 \rrbracket$.

Then $\llbracket\ \rrbracket$ is a total function from $\mathbf{P}_1$ to $\mathbf{P}_2$, and satisfies (1) because $\approx$ is closed under any injective renaming. Immediately we know $\llbracket\ \rrbracket$ satisfies $P_1 \approx Q_1 \ \Leftrightarrow\ \llbracket P_1 \rrbracket \approx \llbracket Q_1 \rrbracket$ because of $\llbracket P_1 \rrbracket \approx P_1$. Then (2) automatically hold.

Finally we have to check $\llbracket\ \rrbracket$ satisfies (3). We divide (a) into two cases (note $\equiv \subset \approx$ and $\llbracket\ \rrbracket$ is fully abstract).

(i) the case $\twoheadrightarrow \overset{\mathrm{def}}{=} \equiv$: $P \equiv P' \ \Rightarrow\ \llbracket P \rrbracket \approx P \equiv \llbracket P' \rrbracket \approx P'$, hence $\llbracket P \rrbracket \approx \llbracket P' \rrbracket$. Then this satisfies $\llbracket P \rrbracket \twoheadrightarrow^0 \llbracket P \rrbracket \approx \llbracket P' \rrbracket$.

(ii) the case $\twoheadrightarrow \overset{\mathrm{def}}{=} \longrightarrow$ is by definition of $\approx$.

(b) is also divided into two cases.

(i) the case $\twoheadrightarrow \overset{\mathrm{def}}{=} \equiv$: $\llbracket P \rrbracket \equiv P' \ \Rightarrow\ P' \twoheadrightarrow^0 \llbracket P \rrbracket \equiv P'$, hence $P' \approx \llbracket P \rrbracket$ with $P \twoheadrightarrow^0 P$.

(ii) the case $\twoheadrightarrow \overset{\mathrm{def}}{=} \longrightarrow$ by definition of bisimulation again. □

Notice by Fact 5.3, we have: $\mathbf{P}_1 \lnsim \mathbf{P}_2 \ \not\Rightarrow\ \mathbf{P}_1 \lnsim^{\mathrm{e}} \mathbf{P}_2$. Thus we can not know *when* and *under what condition* the negative result based on generation can be extended to that based on standard embedding. The rest of this section investigates these points.

5.2 The Negative Results (1): Message Preserving

Before proving the non-existence result, we need the following lemma about names. We note this is generally satisfied on any renaming closed homomorphism [18, 17].

LEMMA 5.5. (name decreasing, Proposition 2.12 in [18]) *Let* $\mathcal{F} : \mathbf{P}_1 \to \mathbf{P}_2$ *be a map closed under injective renaming. If* Q *is in its image, then* $\mathsf{fn}(Q) = \bigcap_{Q=\mathcal{F}(P')} \mathsf{fn}(P')$*. In particular if* $\llbracket\ \rrbracket$ *satisfies* (1) *in Definition 5.1, then we have* $\mathsf{fn}(P) \supset \mathsf{fn}(\llbracket P \rrbracket)$*.*

The following non-existence result is derived based on the properties of concurrent combinators in Sections 3,4 using the above lemmas.

PROPOSITION 5.6. *Suppose* $\mathbf{P}_\pi \subseteq \mathbf{P}$ *is a subsystem of* $\mathbf{P}_{full}$ *and* $\mathbf{P}'$ *is any proper subsystem studied in Sections 3 and 4. Then there is no standard mapping* $\llbracket\ \rrbracket : \mathbf{P} \to \mathbf{P}'$ *which satisfies either:*

(a) (message-preserving) $\llbracket \overline{a}b \rrbracket \approx \overline{a}b$*, or*

(b) (transition-preserving) *(a,b) in Proposition 5.2.*

PROOF. It is enough to show there is no encoding of $\mathbf{c}$ into $\mathbf{P}_{\backslash\mathbf{c}}$ with each $\mathbf{c} \in \{\mathbf{b}_r, \mathbf{b}_l, \mathbf{s}\}$, and $\mathbf{d}$ into $\mathbf{P}_{\mathrm{Af}}$. Below "(1)", "(2)" and "(3)" denote the conditions in Definition 5.1. We have four cases.

Case $\mathbf{d}(abc)$: Assume there is a mapping $\llbracket \mathbf{d}(abc) \rrbracket \in \mathbf{P}_{\mathrm{Af}}$. Then with e fresh and a, b, c distinct, $\mathbf{d}(abc) \,|\, \mathbf{m}(ae) \longrightarrow \mathbf{m}(be) \,|\, \mathbf{m}(ce)$ implies

$$\llbracket \mathbf{d}(abc) \,|\, \mathbf{m}(ae) \rrbracket \overset{\mathrm{def}}{=} \llbracket \mathbf{d}(abc) \rrbracket \,|\, \overline{a}e \longrightarrow^+ P' \approx \llbracket \mathbf{m}(be) \,|\, \mathbf{m}(ce) \rrbracket \approx \overline{b}e \,|\, \overline{c}e$$

by (1,3) and Proposition 2.1 (note $\longrightarrow^+$ is obtained by $\llbracket \mathbf{d}(abc) \,|\, \mathbf{m}(ae) \rrbracket \not\approx \overline{b}e \,|\, \overline{c}e$ because $\llbracket \mathbf{d}(abc) \,|\, \mathbf{m}(ae) \rrbracket \Downarrow_{a\downarrow}$ by (2)). Hence P' has two output transitions: $P' \overset{\overline{b}e}{\Longrightarrow} \overset{\overline{c}e}{\Longrightarrow}$, which implies $\llbracket \mathbf{d}(abc) \rrbracket \,|\, \overline{a}e \twoheadrightarrow (\nu\, \tilde{f})(\overline{b}e \,|\, \overline{c}e \,|\, P_1)$ by Lemma 3.4 (ii) (note e occurs more than twice). But by Lemma 5.5, we have $e \notin \llbracket \mathbf{d}(abc) \rrbracket$, hence $\sharp\langle \llbracket \mathbf{d}(abc) \rrbracket \,|\, \overline{a}e,\ e \rangle = 1$. Then since $(\llbracket \mathbf{d}(abc) \rrbracket \,|\, \overline{a}e) \in \mathbf{P}_{\mathrm{Af}}$, for any Q s.t. $\llbracket \mathbf{d}(abc) \rrbracket \,|\, \mathbf{m}(ae) \twoheadrightarrow Q$, we have $\sharp\langle Q,\ e \rangle \leq 1$ by Lemma 3.4 (i), which is a contradiction.

Case $\mathbf{b}_r$: Assume there exists $\llbracket \mathbf{b}_r(ab) \rrbracket \in \mathbf{P}_{\backslash\mathbf{b}_r}$. Then with a, b, c, e distinct, $\llbracket \mathbf{b}_r(ab) \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(bc) \rrbracket \Downarrow_{e\uparrow}$ by (2), hence $\llbracket \mathbf{b}_r(ab) \rrbracket \,|\, \overline{a}e \,|\, \overline{b}c \Downarrow_{e\uparrow}$. But this contradicts to Lemma 3.6 (i) because $e \notin \mathsf{fs}_\uparrow(\llbracket \mathbf{b}_r(ab) \rrbracket) \subseteq \mathsf{fn}(\llbracket \mathbf{b}_r(ab) \rrbracket)$ by Lemma 5.5.

Case $\mathbf{b}_l$: The same as above.

Case $\mathbf{s}$: Assume there is a mapping $\llbracket \mathbf{s}(abc) \rrbracket \in \mathbf{P}_{\backslash\mathbf{s}}$. Then with e fresh and $a \neq b$, $\llbracket \mathbf{s}(abc) \rrbracket$ should satisfy: we have

- $\neg \mathbf{s}(abc) \Downarrow_{b\downarrow} \Rightarrow \neg \llbracket \mathbf{s}(abc) \rrbracket \Downarrow_{b\downarrow}$ by (2),
- $\mathbf{s}(abc) \,|\, \mathbf{m}(ae) \Downarrow_{b\downarrow} \Rightarrow \llbracket \mathbf{s}(abc) \,|\, \mathbf{m}(ae) \rrbracket \overset{\mathrm{def}}{=} \llbracket \mathbf{s}(abc) \rrbracket \,|\, \llbracket \mathbf{m}(ae) \rrbracket \approx \llbracket \mathbf{s}(abc) \rrbracket \,|\, \overline{a}e \Downarrow_{b\downarrow}$, by (1,2) and (a), and
- $\neg \mathbf{s}(abc) \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(bc) \Downarrow_{e\downarrow} \Rightarrow \neg \llbracket \mathbf{s}(abc) \rrbracket \,|\, \overline{a}e \,|\, \overline{b}c \Downarrow_{e\downarrow}$ by the similar reasoning as above.

Hence $\llbracket \mathbf{s}(abc) \rrbracket$ itself is a synchroniser, which contradicts Lemma 3.12 since $\llbracket \mathbf{s}(abc) \rrbracket \in \mathbf{P}_{\backslash\mathbf{s}}$ by the assumption.

For (b), we prove the only case of $\mathbf{s}$. Others are similar. With e fresh and $a \neq b$, $\neg[\![\mathbf{s}(abc)]\!] \Downarrow_{b\downarrow}$ and $[\![\mathbf{s}(abc)]\!] \stackrel{ae}{\Longrightarrow} P \approx [\![\mathbf{fw}(bc)]\!]$, hence $P \Downarrow_{b\downarrow}$ and $\neg P \Downarrow_{e\updownarrow}$ by Proposition 2.1 (ii). Note if $R \stackrel{ae}{\Longrightarrow} R'$, then $R\,|\,\overline{a}e \longrightarrow^{+} R'$, hence $([\![\mathbf{s}(abc)]\!]\,|\,\overline{a}e) \longrightarrow P_1 \longrightarrow P_2 \longrightarrow \ldots \longrightarrow P_n \stackrel{\text{def}}{=} P$ with $\neg P_i \Downarrow_{e\updownarrow}$ for all $1 \leq i \leq n$ by (b) in 5.2. Similarly for all Q s.t. $[\![\mathbf{s}(abc)]\!]\,|\,\overline{a}e\,|\,\overline{b}e' \twoheadrightarrow Q$, we have $\neg Q \Downarrow_{e\updownarrow}$, which contradicts Lemma 3.12 again. □

The condition $[\![\overline{a}b]\!] \approx \overline{a}b$ above means that we do not change the basic meaning of behaviour by translations and is indeed satisfied in the known fully abstract translations of π-calculus into the asynchronous π-calculus [22, 25, 26, 40] (see also Section 6.2 for more discussion).

5.3 The Negative Result (2) without m *or without* d

In the following, we show there does exist no standard encodings from the whole asynchronous π-calculus into subsystems without messages or without duplicators. The first negative result is easy as follows.

FACT 5.7. *There is no mapping* $[\![\ ]\!] : \mathbf{P} \to \mathbf{P}' \subseteq \mathbf{P}_{\backslash\mathbf{m}}$ *which satisfies (1,2) in Definition 5.1.*

PROOF. Because any $P \in \mathbf{P}_{\backslash\mathbf{m}}$ and a, $P \Downarrow_{a\uparrow}$ is impossible while we should have $[\![\mathbf{m}(ab)]\!] \Downarrow_{a\uparrow}$ by (2) in Definition 5.1. □

The second negative result requires the following lemma, which suggests it is simpler to analyse output transitions in the asynchronous communication rather than in the synchronous one.

LEMMA 5.8. (the number of outputs) *Set* $\sharp_{\uparrow}\langle P, a\rangle$ *as the* number of outputs at a from P, *i.e.* $\sharp_{\uparrow}\langle P, a\rangle = n$ *if:*

$$P \stackrel{\text{def}}{=} P_0 \stackrel{l_1}{\Longrightarrow} P_1 \stackrel{l_2}{\Longrightarrow} P_2 \stackrel{l_2}{\Longrightarrow} \cdots P_{n-1} \stackrel{l_n}{\Longrightarrow} P_n$$

with $l_i = \overline{a}e_i$ or $l_i = \overline{a}(e_i)$ *for some* e_i *for all* $1 \leq i \leq n-1$. *We denote* $\max(\sharp_{\uparrow}\langle P, a\rangle)$ *for the maximum number of* $\sharp_{\uparrow}\langle P, a\rangle$. *Then we have:*

(i) *If* $\sharp_{\uparrow}\langle P, a\rangle = n \lnsim \omega$, *then* $P \twoheadrightarrow (\nu\,\tilde{c})(\prod_n \overline{a}e_i\,|\,R)$ *for some* e_i *and* R *with* $a \notin \{\tilde{c}\}$ *where* $\prod_n Q_i$ *is an abbreviation for* $\prod_n Q_i \stackrel{\text{def}}{=} Q_1\,|\,Q_2\,|\cdots|\,Q_n$ *(if* $n = 0$, $\prod_0 Q_i \stackrel{\text{def}}{=} \mathbf{0}$).

(ii) *Suppose* $\max(\sharp_{\uparrow}\langle P, a\rangle) = \omega$. *Then for all* R *and* P' *such that* $P\,|\,R \twoheadrightarrow P'$, *we have* $P' \Downarrow_{a\uparrow}$.

(iii) *Suppose* $\neg P \Downarrow_{b\downarrow}$ *for any* b. *Then we have:* $P \approx Q$ *implies* $\max(\sharp_{\uparrow}\langle P, a\rangle) = \max(\sharp_{\uparrow}\langle Q, a\rangle)$ *for all* a.

(iv) $1 \leq \max(\sharp_{\uparrow}\langle[\![\overline{a}e_i]\!], a\rangle) \lnsim \omega$.

PROOF. (i) is similar to Lemma 3.4 (ii). (ii) is straightforward by induction of R. For (iii), suppose $P \approx Q$ and $\max(\sharp_{\uparrow}\langle P, a\rangle) = n \lnsim \omega$ for some a. Then by (i), we have: $P \twoheadrightarrow (\nu\,\tilde{c})(\prod_n \overline{a}e_i\,|\,P') \stackrel{\text{def}}{=} R$ for some $\tilde{c}$ and P' with $\neg P' \Downarrow_{a\updownarrow}$. Then we have:
$(R\,|\,\prod_n \mathbf{k}(a)) \equiv (\nu\,\tilde{c})(\prod_n(\overline{a}e_i\,|\,\mathbf{k}(a))\,|\,P') \longrightarrow (\nu\,\tilde{c})(\prod_{n-1}(\overline{a}e_i\,|\,\mathbf{k}(a))\,|\,P') \longrightarrow^{n-1} (\nu\,\tilde{c})P'$ with $\neg(\nu\,\tilde{c})P' \Downarrow_{a\updownarrow}$. Now by $P \approx Q$, there exists Q' such that $Q \twoheadrightarrow Q' \approx R$, and $(Q'\,|\,\prod_n \mathbf{k}(a)) \approx (R\,|\,\prod_n \mathbf{k}(a))$. Hence there exist Q'' such that $(Q'\,|\,\prod_n \mathbf{k}(a)) \twoheadrightarrow Q''$ with $\neg Q'' \Downarrow_{a\updownarrow}$. Thus $Q' \twoheadrightarrow (\nu\,\tilde{c}')(\prod_n \overline{a}e_i\,|\,T)$ with $\neg T \Downarrow_{a\updownarrow}$, which proves $\max(\sharp_{\uparrow}\langle Q, a\rangle) = n$. The case $\max(\sharp_{\uparrow}\langle P, a\rangle) = \omega$ is direct from (ii).

For (iv), $1 \leq \max(\sharp_{\uparrow}\langle[\![\overline{a}e_i]\!], a\rangle)$ is obvious because $[\![\overline{a}e]\!] \Downarrow_{a\uparrow}$. By (3) in Definition 5.1, we have $[\![\mathbf{m}(ae)]\!]\,|\,[\![\mathbf{k}(a)]\!] \twoheadrightarrow P' \approx [\![\mathbf{0}]\!]$, hence $\neg P' \Downarrow_{a\uparrow}$. But by (ii) it is impossible if we assume $\max(\sharp_{\uparrow}\langle[\![\mathbf{m}(ae_i)]\!], a\rangle) = \omega$. □

Now we prove that without $\mathbf{d}(abc)$ we can not construct any standard encodings of π-calculus by the dependency studied in Section 4.2 together with (iii) of the above lemma.

THEOREM 5.9. (the negative result without $\mathbf{d}$) *There is no standard encoding from* $\mathbf{P}_{\pi}$ *to any subsystem of* $\mathbf{P}_{\backslash\mathbf{d}}$. *Hence* $\mathbf{P}_{\backslash\mathbf{d}} \lnsim^{e} \mathbf{P}_{\pi}$.

PROOF. It is enough to show there is no standard encoding of $\mathbf{d}$ into $\mathbf{P}_{\backslash\mathbf{d}}$. Suppose $[\![\mathbf{d}(abc)]\!]$, $[\![\mathbf{m}(ae)]\!] \in \mathbf{P}_{\backslash\mathbf{d}}$ and $\max(\sharp_{\uparrow}\langle[\![\mathbf{m}(ae)]\!], a\rangle) = n$. Note $n \lnsim \omega$ by Lemma 5.8 (iii). Then we have:

$$\begin{aligned}
[\![\mathbf{d}(abc)]\!]\,|\,[\![\mathbf{m}(ae)]\!] &\twoheadrightarrow [\![\mathbf{d}(abc)]\!]\,|\,(\nu\,\tilde{c})(\textstyle\prod_n \overline{a}e_i\,|\,P) && \text{(Lemma 5.8 (i))}\\
&\twoheadrightarrow P' \approx ([\![\mathbf{m}(be)]\!]\,|\,[\![\mathbf{m}(ce)]\!]) && \text{(Def. 5.1 (3))}\\
&\twoheadrightarrow (\nu\,\tilde{c}')(\textstyle\prod_n(\overline{b}e_i\,|\,\overline{c}e_i)\,|\,R) && \text{(Lemma 5.8 (iii))}
\end{aligned}$$

Now suppose $\overline{b}e_i$ needs none of $\prod_n \overline{a}e_i$, i.e. any $\overline{a}e_j$ is not needed for $\overline{b}e_i$. Then a needed pair for $\overline{b}e_i$ should be included in P or $[\![\mathbf{d}(abc)]\!]$. Hence we have $[\![\mathbf{d}(abc)]\!] \Downarrow_{b\uparrow}$ or $P \Downarrow_{b\uparrow}$ (hence $[\![\mathbf{m}(ae)]\!] \Downarrow_{b\uparrow}$), which does not satisfy (2) of Def.5.1. So one of $\prod_n \overline{a}e_i$ is needed for $\overline{b}e_i$. Similarly for $\overline{c}e_i$. Hence all n massages at a are needed for n messages at b, and at the same time all of them are needed for n messages at c. But this contradicts Lemma 4.6. □

Finally we believe the following conjecture.

CONJECTURE 5.10.

(1) (synchronisation) *There is no standard encoding from* $\mathbf{P}_{\pi}$ *to any subsystem of* $\mathbf{P}_{\backslash\mathbf{s}}$, *hence* $\mathbf{P}_{\backslash\mathbf{s}} \lnsim^{e} \mathbf{P}_{\pi}$.

(2) (sharing of names) *There is no standard encoding from* $\mathbf{P}_\pi$ *to any subsystem of* $\mathbf{P}_{Af}$*, hence* $\mathbf{P}_{Af} \not\lesssim^{\mathrm{e}} \mathbf{P}_\pi$.[12]

(3) (full abstraction) *There is no fully abstract standard encoding* (*up to* $\approx$) *from* $\mathbf{P}_\pi$ *into any proper subsystem* $\mathbf{P} \not\lesssim \mathbf{C}$.

(1) and (2) would make sure that the synchronisation in the asynchronous π-calculus is indeed a minimum one and sharing of names is inevitable to construct various communication structures, e.g. polyadic name-passing. Together with (1) in Proposition 5.6, (3) would be proved by showing that if a standard encoding from the asynchronous π-calculus is not message-preserving, then it is not fully abstract up to $\approx$. This would be extended to a more general statement: there is no fully abstract standard encoding from polyadic into monadic name-passing.[13]

REMARK 5.11. (synchronisation) First we replace $\longrightarrow\!\!\!\!\rightarrow$ in Definition 5.1 (3) with $\longrightarrow$ and call this encoding *one-step standard encoding*. Note all known encodings [22, 25, 6, 40, 32, 34] are one-step standard. Recently I have proved that in the system which satisfies the following commutative law (which roughly corresponds the essential π-calculus discussed in 4.14), we *cannot* construct any one-step standard encoding of the whole π-calculus.

$$P_1 \overset{ab}{\Longrightarrow} P_2 \overset{cd}{\Longrightarrow} P_3 \quad \text{with } \{a,b\}\cap\{c,d\}=\emptyset \quad \Rightarrow \quad \exists P_2'.\ P_1 \overset{cd}{\Longrightarrow} P_2' \overset{ab}{\Longrightarrow} P_3.$$

We notice that π_c-calculus does not satisfy this commutative law. We leave the proof, which uses a quite different technique from one in this paper to a coming exposition.

5.4 Standard Encodings based on Other Equivalences

In this subsection, we show all main (negative) results in this section can be preserved even if we replace the synchronous bisimilarity with the asynchronous bisimilarity $\approx_a$ or reduction-based semantics $=_s$ and $=_a$.

DEFINITION 5.12.

- $[\![\ ]\!]$ is *a-standard encoding* if it is defined by replacing $\Downarrow_{a\downarrow}$ in (2) in Definition 5.1 with $\Downarrow_{a\uparrow}$ and $\approx$ in (3) in Definition 5.1 with the asynchronous bisimilarity $\approx_a$, and write $\lesssim^{\mathrm{e}}_a$, $\not\lesssim^{\mathrm{e}}_a$ and $\simeq^{\mathrm{e}}_a$ for *a*-standard relations.
- $[\![\ ]\!]$ is *sound standard encoding* if it is defined by replacing $\approx$ in (3) in Definition 5.1 with the synchronous maximum sound equality $=_s$. Similarly *sound a-standard encoding* is defined by replacing $\Downarrow_{a\downarrow}$ in (2) with $\Downarrow_{a\uparrow}$ and $\approx$ in (3) with the asynchronous maximum sound equality $=_a$. We write $\lesssim^{\mathrm{e}}_s$, $\not\lesssim^{\mathrm{e}}_s$ and $\simeq^{\mathrm{e}}_s$ for sound standard relations and $\lesssim^{\mathrm{e}}_{sa}$, $\not\lesssim^{\mathrm{e}}_{sa}$ and $\simeq^{\mathrm{e}}_{sa}$ for sound *a*-standard relations.

[12]One may also have a conjecture: $\mathbf{P}_{\backslash\mathbf{b}_r} \not\lesssim^{\mathrm{e}} \mathbf{P}_\pi$. But we have $\mathbf{P}_{\backslash\mathbf{b}_r} \simeq^{\mathrm{e}} \mathbf{P}_\pi$ as shown in Proposition 6.1 (ii).

[13]This open question was posed to the author by D. Sangiorgi.

Note that $\lesssim^{\mathrm{e}} \subset \lesssim^{\mathrm{e}}_a \subset \lesssim^{\mathrm{e}}_{sa}$ and $\lesssim^{\mathrm{e}} \subset \lesssim^{\mathrm{e}}_s \subset \lesssim^{\mathrm{e}}_{sa}$, but $\lesssim^{\mathrm{e}}_a$ and $\lesssim^{\mathrm{e}}_s$ are incompatible. The characterisation related with Proposition 5.2 follows.

PROPOSITION 5.13. (relationship with Definition 2.2)

(i) *Let us assume* $[\![\ ]\!]$ *is a-standard. Then we have the same results as (i) and (ii) in Proposition 5.2 by replacing* $\approx$ *with* $\approx_a$ *and* $\xrightarrow{l}$ *with* $\xrightarrow{l}_a$.

(ii) *Suppose* $[\![\ ]\!]$ *is (asynchronous) sound standard. Then we have the same result as (i) in Proposition 5.2 by replacing* $\approx$ *with* $=_s$ *and* $=_a$*, respectively. Moreover if the relation* $\mathcal{R} \overset{\mathrm{def}}{=} \{\langle P, Q\rangle \mid [\![P]\!] =_a [\![Q]\!]\}$ *is congruent,*[14] *it is adequate (up to* $=_s$ *and* $=_a$*, respectively).*

PROPOSITION 5.14. *Let* $\mathbf{P}_\pi \subseteq \mathbf{P}$ *be a subsystem of* $\mathbf{P}_{full}$.

(i) (message preserving) *Assume* $\mathbf{P}'$ *is any proper subsystem studied in Sections 3 and 4. Then there is neither a-standard, sound standard, nor sound a-standard mapping* $[\![\ ]\!] : \mathbf{P} \rightarrow \mathbf{P}'$ *which satisfies* $[\![\overline{a}b]\!] \approx_a \overline{a}b$, $[\![\overline{a}b]\!] =_s \overline{a}b$, *and* $[\![\overline{a}b]\!] =_a \overline{a}b$*, respectively.*

(ii) *Assume* $\mathbf{P}' \in \mathbf{P_{cc}} \backslash \mathbf{m}$ *or* $\mathbf{P}' \in \mathbf{P_{cc}} \backslash \mathbf{d}$*. Then there is neither a-standard, sound standard, nor sound a-standard mapping.*

PROOF. We only have to show the case of the sound *a*-standard encoding. Note we can use neither the input convergence predicate nor observation of values of messages.

(i) For the case $\mathbf{d}(abc)$, we consider $[\![\mathbf{d}(abc) \mid \mathbf{m}(ae) \mid \mathbf{sw}(b) \mid \mathbf{sw}(c)]\!]$ (e fresh) instead of $[\![\mathbf{d}(abc) \mid \mathbf{m}(ae)]\!]$ in the proof of Proposition 5.6. Then use the similar reasoning as in the proof of Proposition 3.23. The case $\mathbf{b}_r(ab)$ is the same as the proof of Proposition 5.6 and the case $\mathbf{b}_l(ab)$ is similar with Proposition 3.20. For the case $\mathbf{s}(abc)$, we prove that there is no sound *a*-standard mapping of $\mathbf{s}_m(abc)$ into $\mathbf{P_{cc}} \backslash \mathbf{s}$ by similar reasoning as in the proof of Proposition 5.6.

(ii) The former is obvious. For the latter, we note that the proofs of Lemma 5.8 and Theorem 5.9 are only concerned with the number of subject of messages. Hence we are done. □

[14]More precisely, it is enough that this relation is closed under structural rules and substitutions (cf. [59]).

6 Discussion

6.1 Summary of the Results

This paper proposed the basic formal framework for representability, *generation* and *minimal basis*, and investigated that computational elements found in 5 combinators [25, 26] are essential to express the asynchronous monadic π-calculus without summation or match operators. 5 combinators can generate the whole behaviour of the calculus, and any of them should not be missing for the full expressiveness. This minimality result clarifies basic nature of our combinators. We also studied several interesting proper subsystems of the asynchronous π-calculus which are separated by combinators. All main results hold based on any of synchronous and asynchronous bisimilarities and synchronous and asynchronous reduction-based equalities. Figure 1 summarises this separation result on (a) systems of combinators and (b) the asynchronous π-calculi, which are in one-one correspondence via a fully abstract mapping. In (b) in Figure 1, names in box depict the embeddable calculi by (congruent) adequate encodings.

6.2 Related Work

In this subsection, we summarise known results about encodings among the full π-family based on the formulation in Section 5. Then we have the following relationship.

PROPOSITION 6.1.

(i) $\mathbf{P}_{\mathbf{cc}} \simeq^{\mathrm{e}} \mathbf{P}_\pi \simeq^{\mathrm{e}}_a \mathbf{P}_{\pi+}$ *where* $\mathbf{P}_{\pi+}$ *is the asynchronous* π*-calculus with input guarded summations.*

(ii) $\mathbf{P}_\pi \simeq^{\mathrm{e}} \mathbf{P}_{\mathrm{pol}\pi\mathrm{s}}$ *and* $\mathbf{P}_l \simeq^{\mathrm{e}} \mathbf{P}_\pi$, *hence* $\mathbf{P}_{\mathbf{cc}}\backslash\mathbf{b}_l \simeq^{\mathrm{e}}_a \mathbf{P}_{\mathrm{pol}\pi\mathrm{s}+}$ *where* $\mathbf{P}_{pol\pi s}$ *is polyadic synchronous* π*-calculus without match or summation and* $\mathbf{P}_{pol\pi s+}$ *is* $\mathbf{P}_{pol\pi s}$ *plus input guarded summations.*

(iii) *Let us suppose* $\mathbf{P}_1 \subseteq \mathbf{P}_2$ *and both are subsystems without match operators. Assume* $\mathbf{P}_2$ *has mixed summation operators, while* $\mathbf{P}_1$ *does not. Then* $\mathbf{P}_1 \lnsim^{\mathrm{e}} \mathbf{P}_2$*.*

PROOF. (i) is by Theorem 2.5 and Nestmann and Pierce [40], respectively. (ii) is by Honda and Tokoro [22], Boreale [6] and (i), respectively. (iii) is by Palamidessi [44]. □

The result in (i) is stronger than (ii) because existent encodings are fully abstract (up to congruence), while in (ii), we have only adequate encodings from the right calculus to the left one. Palamidessi's result [44] is more general than (iii) above because we do not need the condition (3) in Definition 5.1. See Introduction in [41] for the detailed relationship about adequacy and full abstraction between existent encodings.

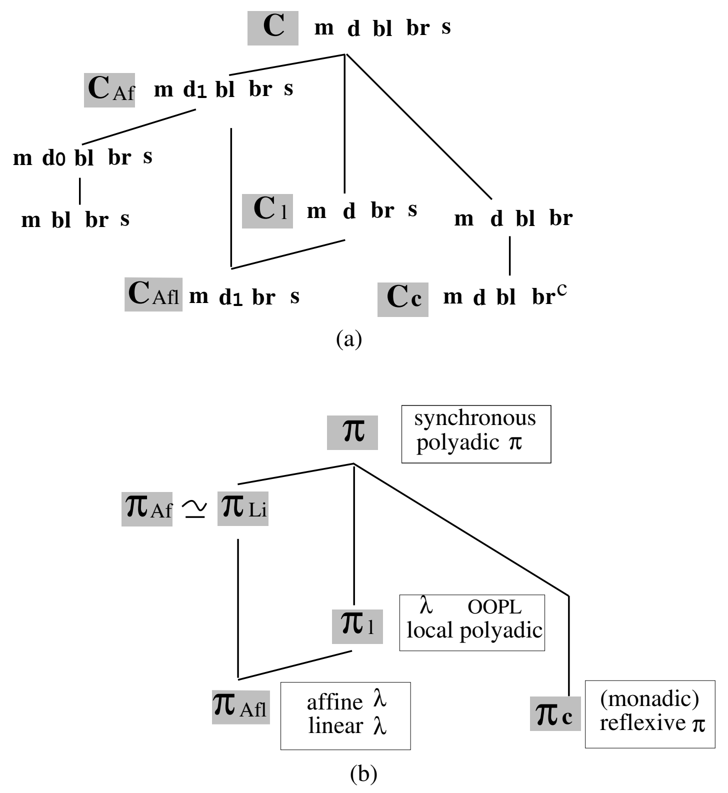

FIGURE 1. Concurrent Combinators and the Asynchronous π-Calculus

Local π-calculus

Two remarks are due for Proposition 5.6 (1) concerning with local π-calculus.

First, in [6], Boreale recently established an interesting result which shows power of the local asynchronous (polyadic) π-calculus: there is an encoding from (polyadic) π-calculus to polyadic local (asynchronous) π-calculus which satisfies the stronger property than (3) in Definition 5.1 and which is fully abstract up to the weak barbed bisimilarity. But this result does not contradict Conjecture 5.10 (3) since:

(1) It is not fully abstract up to barbed congruence (hence not up to $\approx$ either). See Appendix E for a counterexample. Note as discussed in 3.2 in [52] and Sec 6 in [24], barbed bisimulation itself is weak as a canonical equality, e.g $bx.\mathbf{0}$ is equated to $bx.\overline{a}v$ in it.

(2) Even under the barbed bisimilarity, we do not know whether there is a fully abstract encoding from the asynchronous π-calculus into *monadic* π_l-calculus because he uses the power of polyadic name passing (hence $\mathbf{P}_l \simeq^{\mathrm{e}} \mathbf{P}_\pi$ is only adequately related).

(3) It is *not* message-preserving, while all fully abstract encodings in (i) in Proposition 6.1 are all message-preserving.

Related with (1), in the long version of [6], he showed that his encoding is closed under *translation contexts*, i.e. we only consider the world of translations as the whole environment. The similar approach was also suggested in 6.6.1 in [52]. The basic idea of this kind technique is related with the study of types of mobile processes, cf. [59]; in order to get the full abstraction embedding from a high-level communication into a low-level one, we may need to restrict the environment in the low-level one.

Secondly, Merro and Sangiorgi recently proved another interesting result: an encoding based on the second Boreale's encoding in [6] from local π-calculus to a variant of local π-calculus where all objects of messages are distinct and bound by name hiding[15] is *fully abstract* up to the asynchronous weak barbed congruence [32]. This encoding uses *the link agent* in [53] to translate messages with free object names, so it is *not* message-preserving. But again this does not contradict Conjecture 5.10 (3) because we consider encodings of the whole (i.e. non-local asynchronous) π-calculus. Their result reveals that not only the link agent is enough to describe all local communication behaviour as shown in [53, 27], but also it becomes another key agent to understand a difference between non-local and local worlds.

[15]This variant is not included in the syntax of πI-calculus [53], but it is considered as an asynchronous version of πI-calculus by changing $\overline{a}(x).P$ in πI-calculus by $(\nu x)(\overline{a}x\,|\,P)$ where x does not occur as free objects in P.

Expressiveness based on Combinators

The framework of measuring expressive power which is most closely related with our idea, *generation* and *essentiality*, was formalised by Parrow in [45] in the context of non-value name passing calculi. First he proposed a simple general algebraic language for describing a fixed number of processing units with disjoint parallel and linking operators. Then he showed (1) every term which has a finite state behaviour is generated by three units and each of them is essential (he used the term “independent”), and (2) various kinds of operators in non-value passing process calculi are examined by introducing the idea of definable operators. The significant differences between his minimality result and ours are (1) we analysed expressiveness of π-calculus (i.e. name-passing), including both finite and infinite state behaviours, and (2) we use the concrete combinators while he used labelled transition based analysis to show the essentiality. The interesting further research is to extend our proof technique to the transition-relation based analysis as his, and examine not only the power of terms but also that of operators (i.e. parallel composition and name hiding). A related line of study has been done in [18] on the nameless processes from the more general viewpoint.

Parrow recently showed a combination of a few kinds of *trios*, which are polyadic synchronous π-terms in the form $T = \alpha_1.\alpha_2.\alpha_3.\mathbf{0}$ where α_i denotes input, output or τ prefix, can represent the synchronous polyadic π-calculus without match or summation operator up to weak bisimilarity [46]. More precisely, he showed there is a mapping which satisfies $P \approx [\![P]\!]$ and is translated into the normal form called *a concert of trios* $(\nu\,\tilde{c})((!)T_1\,|\,(!)T_2\,|\cdots|\,(!)T_n)$ and two messages up to the strong bisimilarity. Interestingly we can observe that all our 5 combinators are trios; hence his study and our strong minimality theorem showed that three times synchronisation made by prefixes is indeed essential to realise the causality of name-passing interaction of π-calculus. On the other hand, since his mapping is *not* homomorphic as ours, it may be difficult to apply directly his trios and mapping to examine the existence/non-existence of general homomorphic encodings which is in a weaker condition than $P \approx [\![P]\!]$ (i.e. like standard encodings) as shown in Section 5 in our paper.

Raja and Shymasundar also studied Quine combinators for the asynchronous π-calculus [50]. Since their combinators are not a proper subset of π-calculus like ours, the ideas of basis and generation may not be directly applicable to this system. However to check essentiality of each combinator of [50] would also be interesting for understanding the basic machinery of name-passing from a different angle.

6.3 Open Issues

In the following, we list some of naturally arising open issues.

- As we discussed in Section 5 and the above, much still remains to be done on the study of existence or non-existence result of adequate and fully abstract encodings. For example, Boreale's result on local π-calculus [6] lets us know a possibility to construct various kinds of standard encodings. This also suggests that there is some difficulty to solve the negative result about encodings. Based on this observation, the most interesting but difficult open problem may be Conjecture 5.10 (1). This would reveal that the asynchronous π-calculus may be considered as a "basic π-calculus" containing sufficient power for interactive computation in a minimal tractable syntax.

- Related with this, our result in Section 4 tells us that all computable functions can be expressed in the local π-calculus. More interestingly, the encoding of neither call-by-value nor lazy λ-calculus in [34] works in π_{Af}-calculus although it includes infinite behaviour like $!ax.\overline{b}x$, cf. footnote 7. What is a minimal basis to realise universal computation power in π-calculus? Is it absolutely needed to increment the number of names during reduction and synchronise at the input prefix to represent sequential computation? Such an investigation is another important topic because it relates a basic question in functional computing to expressiveness of concurrent computing.

- In Definition 2.2, we use "!" operator for generation (i-2-e). But by the result in [26], from a basis of at most 19 combinators we can generate the asynchronous π-calculus with replication without using replication as an operator. We also remark that the binding nature of restriction is representable using "naming action" [36], or "processes for connection" [18, 19]. It may be interesting to check the essentiality of these agents to understand what computational elements are essential to express "copies" and "name restriction" in mobile processes.

- Gay and Lafont independently found the systems of combinators of *untyped* interaction nets and the later also proved essentiality of each combinator by graphical analysis. In interaction nets, the idea of named ports is not explicitly present, not because it has been abstracted away but because arbitrary connection among agents is not used. Since they do not develop algebraic structures underlying their construction, the direct comparison with their combinators and ours is difficult. Recently Fernandéz and Mackie [31] proposed a formal way to translate *typed* interaction nets to term rewriting systems. Though we do not yet know a system of combinators for typed interaction nets yet, if it is discovered, by using their translation and [58] we may be able to understand exactly what subclass of communication in π-calculus is related with (typed) interaction nets based on the measuring framework proposed in this paper.

- We examined the expressiveness of the asynchronous monadic π-calculi using concurrent combinators, which gave us basic understanding on the computational elements of name-passing. A similar analysis may be more difficult in the setting of polyadic name-passing even if we do have its combinatory representation. For example, take a polyadic π-term $a(xy).b(z).\overline{y}[z]$. This process is regarded as a general synchroniser because the first value x is thrown away. At the same time, the second value y is used as an output subject. Such phenomena lead to difficulty in the analysis and decomposition of prefixes. On the other hand, in the polyadic synchronous setting, there is a system of combinators for π-calculus in action structures [36, 19], and for a match/summation-less Fusion calculus [47] (see [28]). Measuring expressiveness in such a calculus following the line of this paper would be possible and interesting for examination of the expressiveness in the world of synchronous name-passing.

- Finally match and mismatch operators are also significant from both practical and theoretical viewpoints [3, 24, 48, 43], while practical failure models [2, 51, 11] are recently studied by introducing additional operators. A systematic inquiry about the separation results on such operators would increase theoretical understanding on computation in the family of π-calculi.

Acknowledgements: This full version has been revised from the original version (October 1997) based on the anonymous referees' comments for the extended abstract of this paper [60]. The question on the minimality of concurrent combinators was independently posed by Benjamin Pierce, Davide Sangiorgi and Vasco Vasconcelos. The part of the basic idea of Subsection 4.2 came from Samson Abramsky's short course lecture in Edinburgh University and the notion of standard encodings in Section 5 owes to Kohei Honda's short course lectures delivered at Lisbon and Edinburgh Universities. Raja Natarajan's comments on the first version of this paper are helpful to revise this version. Simon Gay, Catuscia Palamidessi and Vasco Vasconcelos also gave her useful comments on the early versions. Several propositions on relationships between $\lesssim$ and $\lesssim^{\mathrm{e}}$ in Section 5 are newly added in this version following Matthew Hennessy's suggestion. Michele Boreale sent me the full version of [6] and discussed with her about his encodings. Gerard Boudol pointed out a relationship between his extension of π-cham in [9] and π_c-calculus. Massimo Merro also sent his paper about local π-calculus and had discussions by a series of e-mails. John Power and Ian Stark had discussions with her and Peter Mosses gave her a related question. She deeply thanks these people and all who discussed with her about concurrent combinators and related topics. She was a visiting academic researcher in LFCS, University of Edinburgh during the writing of the first version of this paper, supported by JSPS Fellowship for Japanese Junior Scientists. She is cur-

rently partially supported by EPSRC GR/K60701.

References

[1] Abramsky, S., Interaction, Combinators, and Complexity, LFCS short courses 1997, April, Edinburgh University, 1997.

[2] Amadio, R., An asynchronous model of locality, failure, and process mobility. INRIA Research Report 3109, 1997.

[3] Amadio, R., Casellani, I. and Sangiorgi, D., On Bisimulations for the Asynchronous π-calculus, *Proc. CONCUR'96*, LNCS 1119, pp.147–162, Springer-Verlag, 1996.

[4] Barendregt, H., *The Lambda Calculus: Its Syntax and Semantics*. North Holland, 1984.

[5] Berry, G. and Boudol, G., The Chemical Abstract Machine. TCS, vol 96, pp. 217–248, 1992.

[6] Boreale, M., On the Expressiveness of Internal Mobility in Name-Passing Calculi, *Proc. CONCUR'96*, LNCS 1119, pp.163–178, Springer-Verlag, 1996.

[7] Boreale, M. and Sangiorgi, D., A Fully Abstract Semantics for Causality in the π-calculus. LFCS report, ECS-LFCS-94-297, University of Edinburgh, 1994.

[8] Boudol, G., *Asynchrony and π-calculus*. INRIA Report 1702, INRIA, Sophia Antipolis, 1992.

[9] Boudol, G., Some chemical abstract machines, Proceedings of the REX School/Workshop "A Decade of Concurrency", LNCS 803, pp.92–123, 1994.

[10] Fournet, C. and Gonthier, G, A Hierarchy of Equivalences for Asynchronous Calculi, *ICALP'98*, LNCS, Springer-Verlag, 1998.

[11] Fournet, C. et al., A Calculus for Mobile Agents, *CONCUR'96*, LNCS 1119, pp.406–421, Springer-Verlag, 1996.

[12] Danos, V., Herbelin, H. and Regnier, L., Games Semantics and Abstract Machines. *LICS'96*, IEEE, 1996.

[13] Degano, P. and Priami, C., Non-interleaving Semantics for Mobile Processes, *Proc. of ICALP'95*, LNCS 994, pp.660–671, Springer-Verlag, 1995.

[14] Gay, S., Combinators for Interaction Nets, *Workshop on Theory and Formal Methods*, Imperial College Press, 1995.

[15] Hennessy, M., *An Algebraic Theory of Processes*, MIT Press, 1988.

[16] Honda, K., *Two Bisimilarities in ν-calculus*, Keio Technical Report, 92-002, 1992.

[17] Honda, K., *Notes on P-Algebra (1): Process Structure*. *Proc. TPPP'94*, LNCS 907, pp.25–44, Springer-Verlag, 1995.

[18] Honda, K., *Process Structure*, 49 pp, a typescript. Submitted for publication, March, 1997. Available from `http://www.dcs.ed.ac.uk/home/kohei`.

[19] Honda, K., *Notes on Undirected Action Structure*, a typescript, March, 1997. Available from `http://www.dcs.ed.ac.uk/home/kohei`.

[20] Honda, K., A Short Course on Mobile Processes, LFCS short courses 1997, April, Edinburgh University, 1997.

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

[22] Honda, K. and Tokoro, M., An Object Calculus for Asynchronous Communication. *ECOOP'91*, LNCS 512, pp.133–147, Springer-Verlag 1991.

[23] Honda, K. and Tokoro, M., On Asynchronous Communication Semantics, Object-Based Concurrent Computing, Lecture Notes in Computer Science, Vol. 612, pp. 21-51, Springer-Verlag, 1992.

[24] Honda, K. and Yoshida, N., On Reduction-Based Process Semantics. *FSTTCS'13*, LNCS 761, pp. 373–387, Springer-Verlag, December 1993. Full version appeared in *TCS*, pp.437–486, No.151, North-Holland, December, 1995.

[25] Honda, K. and Yoshida, N., Combinatory Representation of Mobile Processes. *POPL'94*, pp.348–360, ACM Press, 1994.

[26] Honda, K. and Yoshida, N., Replication in Concurrent Combinators, *TACS'94*, LNCS 789, pp.786–805, Springer, 1994.

[27] Honda, K. and Yoshida, N., Game-Theoretic Analysis of Call-by-Value Computation, *Proc. ICALP'97, Proceedings of 24th International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science 1256, pp.225–236, Springer-Verlag, July, 1997. The full version:

[28] Full version of [25]. To appear as a technical report of Sussex University, 1998.

[29] Lafont, Y., Interaction Nets, *POPL'90*, pp. 95–108, ACM press, 1990.

[30] Lafont, Y., Interaction Combinators, *Info. & Comp.*, 137:69–101, 1996.

[31] Fernandéz, M. and Mackie, I., Interaction Nets and Term Rewriting Systems, *CAAP'96*, LFCS, pp.149–164, Springer-Verlag, 1996.

[32] Merro, M. and Sangiorgi, D., On asynchrony in name-passing calculi, To appear in *ICALP'98*, 1998.

[33] Milner, R., Fully abstract models of typed lambda calculi. *TCS*, Vol.4, 1–22, North-Holland, 1977.

[34] Milner, R., Functions as Processes. *Mathematical Structure in Computer Science*, 2(2), pp.119–146, 1992.

[35] Milner, R., Polyadic π-Calculus: a tutorial. *Logic and Algebra of Specification*, Springer-Verlag, 1992.

[36] Milner, R., *Action structures and the π-calculus*, Proc. of Advanced Study Institute on Proof and Computation, 60pp, Marktoberdorf, 1993.

[37] Milner, R., Calculi for Interaction. Acta Informatica, December, 1996.

[38] Milner, R., Parrow, J.G. and Walker, D.J., A Calculus of Mobile Processes, *Information and Computation* 100(1), pp.1–77, 1992.

[39] Mitchell, J., Type Systems for Programming Languages. *Handbook of Theoretical Computer Science* B, pp.367–458, MIT Press, 1990.

[40] Nestmann, U. and Pierce, B., Decoding choice encodings, *Proc. CONCUR'96*, LNCS 1119, pp.179–194, Springer-Verlag, 1996.

[41] Nestmann, U., What is a 'Good' Encoding of Guarded Choice? A typescript, 19pp, May, 1997.

[42] Odersky, M., Applying π: Towards a basis for concurrent imperative programming, *2nd. SIPL*, pp.95–108, 1995.

[43] Odersky, M., Polarized Name Passing. *FST/TCS'15*, LNCS, Springer-Verlag, 1995.

[44] Palamidessi, C., Comparing the Expressive Power of the Synchronous and the Asynchronous π-calculus, *POPL'97*, pp. 256–265, ACM press, 1997.

[45] Parrow, J., The Expressive Power of Parallelism. Future Generation Computer Systems, 6:271–285, 1990. Available from `http://www.sics.se/ joachim/expr.ps.Z`.

[46] Parrow, J., Trios in Concert. Festschrift in honour of Robin Milner, MIT Press, 1998. Available from `http://www.sics.se/ joachim/trios.ps.Z`.

[47] Parrow, J. and Victor, B., The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes. *LICS'98*, IEEE, 1998.

[48] Parrow, J. and Sangiorgi, D., Algebraic Theories for Name-Passing Calculi, Research Report

ECS-LFCS-93-262, Department of Computer Science, University of Edinburgh 1993.

[49] Pierce, B. and Turner, D., Pict: A Programming Language Based on the Pi-calculus, Indiana University, CSCI Technical Report, 476, March, 1997.

[50] Raja, N. and Shyamasundar, R.K., Combinatory Formulations of Concurrent Languages, *TOPLAS*, Vol. 19, No. 6, pp.899-915, ACM Press, 1997.

[51] Riely, J. and Hennessy, M., A Typed Language for Distributed Mobile Processes. *POPL'98*, pp.378–390, ACM Press, 1998.

[52] Sangiorgi, D., *Expressing Mobility in Process Algebras: First Order and Higher Order Paradigms*. Ph.D. Thesis, University of Edinburgh, 1992.

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

[54] Sangiorgi, D., Locality and Non-interleaving Semantics in Calculi for Mobile Processes, LFCS report, ECS-LFCS-94-282, University of Edinburgh, 1994. An extract appeared in TACS'94, LNCS 789, Springer, 1994.

[55] Turner, D.A., A New Implementation Technique for Applicative Languages., Software-Practice and Experience 9, pp.31–49, 1979.

[56] Vasconcelos, V., Typed Concurrent Objects, *ECOOP'94*, LNCS 814, pp.100–117, Springer, 1994.

[57] Walker, D., Objects in the π-calculus. *Information and Computation*, Vol. 116, pp.253–271, 1995.

[58] Yoshida, N., Graph Notation for Concurrent Combinators, *TPPP'94*, LNCS 907, pp.393–412, Springer-Verlag, 1995.

[59] Yoshida, N., Graph Types for Mobile Process Calculi, *FST/TCS'16*, LNCS 1180, pp. 371–386, Springer-Verlag, 1996. The full version appears as LFCS Technical Report, ECS-LFCS-96-350, 1996.

[60] Yoshida, N., Extended Abstract of this paper. Available from `http://www.cogs.susx.ac.uk/users/nobuko/index.html`. The first version in Oct, 1997. Revised in June, 1998. To appear in *CONCUR'98*, LNCS, Springer, September, 1998.

A Reduction, Synchronous and Asynchronous Transition Relations and the Sound Equality

A.1 Reduction

We consider terms modulo the structural congruence following [34, 5]: $\equiv$ is the smallest congruence relation over π-terms generated by the following rules.

(i) $P \equiv Q$ if $P \equiv_\alpha Q$

(ii) $P|Q \equiv Q|P \quad (P|Q)|R \equiv P|(Q|R) \quad P|\mathbf{0} \equiv P \quad !P \equiv P|!P$

(iii) $(\nu aa)P \equiv (\nu a)P \quad (\nu ab)P \equiv (\nu ba)P \quad (\nu a)\mathbf{0} \equiv \mathbf{0} \quad (\nu a)P|Q \equiv (\nu a)(P|Q)$ if $a \notin \mathsf{fn}(Q)$

Definition A.1. (reduction) The one-step reduction relation $\longrightarrow$ is generated by the following rule.

$$\begin{array}{ll}
\text{(COM)} & ax.P \mid \overline{a}v \longrightarrow P\{v/x\} \\
\text{(PAR)} & P \longrightarrow Q \Rightarrow P \mid R \longrightarrow Q \mid R. \\
\text{(RES)} & P \longrightarrow Q \Rightarrow (\nu a)P \longrightarrow (\nu a)Q. \\
\text{(STR)} & P \equiv P' \;\; P' \longrightarrow Q' \;\; Q \equiv Q' \;\Rightarrow\; P \longrightarrow Q.
\end{array}$$

The multi-step reduction relation, $\longrightarrow\!\!\!\!\rightarrow$, is defined by $\longrightarrow\!\!\!\!\rightarrow \overset{\text{def}}{=} \longrightarrow^* \cup \equiv$.

A.2 Bisimilarities

The set of labels, ranged over by $l, l', ..$, is given by:

$$l \;=\; \tau \mid ab \mid \overline{a}b \mid \overline{a}(b)$$

where "(b)" in "$\overline{a}(b)$" is the *bound* occurrence of the label. We write $\mathsf{bn}(l)$ and $\mathsf{fn}(l)$ for the sets of bound and free names in l. A label l is *relevant to P* if $\mathsf{bn}(l) \cap \mathsf{fn}(P) = \emptyset$.

Definition A.2. The (synchronous early) transition relation, denoted by $\xrightarrow{l}$, is the smallest relation inferred by the following rules.

$$\text{(alh): } \frac{P' \equiv_\alpha P \;\; P \xrightarrow{l} Q \;\; Q \equiv_\alpha Q'}{P' \xrightarrow{l} Q'} \qquad (\text{in}_s)\text{: } ax.P \xrightarrow{ab} P\{b/x\} \qquad \text{(out): } \overline{a}b \xrightarrow{\overline{a}b} \mathbf{0}$$

$$\text{(com): } \frac{P \xrightarrow{\overline{x}y} P' \;\; Q \xrightarrow{xy} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \qquad \text{(close): } \frac{P \xrightarrow{\overline{x}(y)} P' \;\; Q \xrightarrow{xy} Q'}{P|Q \xrightarrow{\tau} (\nu y)(P'|Q')} \; (y \notin \mathsf{fn}(Q))$$

$$\text{(rep): } \frac{P|!P \xrightarrow{l} P'}{!P \xrightarrow{l} P'} \qquad \text{(par): } \frac{P \xrightarrow{l} P'}{R|P \xrightarrow{l} R|P'} \; (\mathsf{bn}(l) \cap \mathsf{fn}(R) = \emptyset)$$

$$\text{(res): } \frac{P \xrightarrow{l} P'}{(\nu a)P \xrightarrow{l} (\nu a)P'} \; (a \notin \mathsf{fn}(l) \cup \mathsf{bn}(l)) \qquad \text{(open): } \frac{P \xrightarrow{\overline{a}b} P'}{(\nu b)P \xrightarrow{\overline{a}(b)} P'} \; (a \neq b)$$

We omit the symmetric cases for (com), (close), (rep) and (par). Then $\Longrightarrow$ stands for the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\overset{\hat{l}}{\Longrightarrow}$ stands for $\Longrightarrow$ if $l = \tau$, else for $\Longrightarrow\xrightarrow{l}\Longrightarrow$. The weak bisimilarity is defined in the standard way: A *weak bisimulation* is any symmetric relation $\mathcal{R}$ such that, if $P \;\mathcal{R}\; Q$, whenever $P \xrightarrow{l} P'$ with l relevant to Q, there exists Q' such that $Q \overset{\hat{l}}{\Longrightarrow} Q'$ and $P' \;\mathcal{R}\; Q'$. By the standard argument, there exists the maximum bisimulation which is the union of all the bisimulations. The maximum weak bisimulation is called *bisimilarity* written $\approx$.

Definition A.3. (asynchronous bisimilarity) The (asynchronous early) transition relation, denoted by $\xrightarrow{l}_a$, is the smallest relation inferred by the following

(in_a, τ) and (alh,out,rep,par,res,open) in Definition A.2 replacing $\xrightarrow{l}$ with $\xrightarrow{l}_a$.

$$(\mathrm{in}_a) \quad \mathbf{0} \xrightarrow{ab}_a \overline{a}b \qquad\qquad (\tau): \quad \frac{P \xrightarrow{\tau} P'}{P \xrightarrow{\tau}_a P'}$$

Then $\overset{l}{\Longrightarrow}_a$ and the asynchronous weak bisimulation are defined similarly. We denotes $\approx_a$ for *the asynchronous weak bisimilarity*.

A.3 Sound Equalities

Equality over the asynchronous π-calculus has been studied extensively in [16, 24, 10]. Here we defined a construction which is naturally applicable to many process calculus without introducing labelled transition relations [24, 3].

DEFINITION A.4. (sound equality) We say a congruence $\cong$ is *synchronous* (resp. *asynchronous*) *sound* if it includes $\equiv$, is *reduction closed*, i.e. $P \cong Q$ and $P \longrightarrow\!\!\!\!\!\rightarrow P'$ implies there exists Q' such that $Q \longrightarrow\!\!\!\!\!\rightarrow Q'$ with $P' \cong Q'$, and, moreover, $\cong$ *respects* $\Downarrow_{a^{\updownarrow}}$, i.e. if $P \cong Q$ and $P \Downarrow_{a^{\updownarrow}}$ (resp. $P \Downarrow_{a^{\updownarrow}}$) then $Q \Downarrow_{a^{\updownarrow}}$ (resp. $Q \Downarrow_{a^{\uparrow}}$).

The first condition tells us that we are essentially working with the terms modulo $\equiv$. According to this and the last condition, a sound congruence is automatically non-trivial (i.e. neither universal nor empty). Moreover we can easily verify that the congruent closure of a family of sound congruences is again sound. Then, by taking the congruent closure of the whole family of sound congruences, we immediately know there is maximum synchronous (resp. asynchronous) sound congruence within the family of all synchronous (resp. asynchronous) sound congruences. We denote this maximum sound equality $=_s$ (resp. $=_a$).[16]

B Proof for Proposition 2.7

First we extend the definition of *pointedness* defined in [25] to incorporate with the infinite copy $!P$. $P \in \mathbf{P_{cc}}$ is $a^{\downarrow}$*-pointed*, iff P satisfies the following condition:

(1) $\mathsf{an}_{\downarrow}(P) = \{a\}$ and $\mathsf{an}_{\uparrow}(P) = \emptyset$

(2) $\forall \mathbf{c}(a^{+}\tilde{v}).\ (P|\mathbf{c}(a^{+}\tilde{v})) \longrightarrow Q_1 \ \wedge\ (P|\mathbf{c}(a^{+}\tilde{v})) \longrightarrow Q_2 \ \Rightarrow\ Q_1 \equiv Q_2$.

(3) $P \not\longrightarrow$.

$a^{\uparrow}$*-pointed* is defined with changing $\uparrow$ by $\downarrow$ and $+$ by $-$ respectively. Pointedness of a term tells us that there is only one interacting name in a given term. We write $P_{\langle a^{\updownarrow}\rangle}$ etc. to denote a $a^{\updownarrow}$-pointed term P. Note $a^{*}x.P$ is $a^{\downarrow}$-pointed. We define β-reduction/equality $\rightarrow_\beta/\!=_\beta$ as in Definition 3.5 in [25].

[16]These equalities coincide with barbed congruences with action predicate $\Downarrow_{a^{\updownarrow}}$ and $\Downarrow_{a^{\uparrow}}$, respectively [10].

DEFINITION B.1. The one-step β-reduction, $\rightarrow_\beta$, is defined by the rule:

$$(\mathrm{COM}_\beta)\ (\nu\, c)(P_{\langle c^{\downarrow}\rangle} \,|\, Q_{\langle c^{\uparrow}\rangle}) \ \rightarrow_\beta\ (\nu\, c)R$$

if $P_{\langle c^{\downarrow}\rangle} \,|\, Q_{\langle c^{\uparrow}\rangle} \longrightarrow R$, together with (PAR), (RES) and (STR) in Definition A.1. $\twoheadrightarrow_\beta \overset{\text{def}}{=} \rightarrow_\beta^{*} \cup \equiv$, while $=_\beta$ is the symmetric closure of $\twoheadrightarrow_\beta$.

This β-reduction/equality satisfies the following properties.

PROPOSITION B.2.

(i) (non-interference) *Suppose* $P \twoheadrightarrow_\beta Q_1$ *and* $P \xrightarrow{l} Q_2$ *with* $Q_1 \not\equiv Q_2$. *Then there exists* Q' *s.t.* $Q_1 \xrightarrow{l} Q'$ *and* $Q_2 \twoheadrightarrow_\beta Q'$.

(ii) $=_\beta$ *is a weak bisimulation, hence* $P =_\beta Q$ *implies* $P \approx Q$.

(iii) *Let us define* $\mathcal{R}_0 \overset{\text{def}}{=} \approx$, *and given* $\mathcal{R}_{i-1}$ $(i \geq 1)$, *we define* $\mathcal{R}_i$ *as the maximum relation such that, for all* $P\,\mathcal{R}_i\,Q$ *and* l *with* $i \geq 1$ *and* l *relevant to* Q,

whenever $P \xrightarrow{l} P'$ *then, for some* Q', $Q \overset{\hat{l}}{\Longrightarrow} Q'$ *with* $P' \twoheadrightarrow_\beta \mathcal{R}_{i-1} \equiv Q'$

whenever $Q \xrightarrow{l} Q'$ *then, for some* P', $P \overset{\hat{l}}{\Longrightarrow} P'$ *with* $P' \twoheadrightarrow_\beta \mathcal{R}_{i-1} \equiv Q'$

Then $\mathcal{R} \overset{\text{def}}{=} \cup_{i\geq 0}\mathcal{R}_i \subseteq \approx$, *indeed* $\mathcal{R}_i = \approx$ *for all* i.

PROOF. (i) is proved as in Lemma 3.6 in [25]. For (ii), take $\mathcal{R} = \{\langle P, Q\rangle \mid P \rightarrow_\beta Q\} \cup \equiv$ and show it is a bisimulation. (iii) is by exploiting $\mathcal{R}' \overset{\text{def}}{=} \cup_{i\geq 0}(\twoheadrightarrow_\beta \mathcal{R}_i \equiv)$ is a weak bisimulation. Case $i = 0$ is obvious. Assume $i \geq 1$ and $P \twoheadrightarrow_\beta P_0\,\mathcal{R}_i\,Q_0 \equiv Q$. Then $P \xrightarrow{l} P'$ implies $P_0 \xrightarrow{l} P_0'$ with $P' \twoheadrightarrow_\beta P_0'$ by (i). By $P_0\,\mathcal{R}_i\,Q_0$, we have $Q_0 \overset{\hat{l}}{\Longrightarrow} Q_0'$ with $P_0' \twoheadrightarrow_\beta \mathcal{R}_{i-1} \equiv Q_0'$. Since $Q \overset{\hat{l}}{\Longrightarrow} Q' \equiv Q_0'$, we have $P' \twoheadrightarrow_\beta P_0 \twoheadrightarrow_\beta \mathcal{R}_{i-1} \equiv Q_0' \equiv Q'$. For the symmetric case, suppose $Q \xrightarrow{l} Q'$. Then we have $Q_0 \xrightarrow{l} Q_0' \equiv Q'$. Hence $P_0 \overset{\hat{l}}{\Longrightarrow} P_0'$ with $P_0'\,\mathcal{R}_{i-1}\,Q_0'$. Since $P \twoheadrightarrow_\beta P_0 \overset{\hat{l}}{\Longrightarrow} P_0'$ implies $P \overset{\hat{l}}{\Longrightarrow} P' \equiv P_0'$, we have $P' \twoheadrightarrow_\beta \mathcal{R}_{i-1} \equiv Q'$, as required. $\square$

Now we are ready to prove Proposition 2.7 (i).

We only have to consider the input case because $a^{*}x.P$ is $a^{\downarrow}$-pointed. Rules (II,III,VII,VIII,IX) are clear because $a^{*}x.P \equiv ax.P$. Rules (V,VI,X–XIII) are also mechanically checked.

For the rule (II), we use a relation $\mathcal{R}$ of (iii) in Proposition B.2. Suppose $ax.(P_1 \,|\, P_2) \xrightarrow{ab} (P_1\{b/x\} \,|\, P_2\{b/x\})$. Then

$$a^*x.(P_1 \,|\, P_2) \xrightarrow{ab}\equiv (\nu\, c_1c_2)(\mathbf{m}(c_1b) \,|\, c_1^*x.P_1 \,|\, \mathbf{m}(c_2b) \,|\, c_2^*x.P_2) \stackrel{\text{def}}{=} R$$

Since $\overline{c_i}b \,|\, c_ix.P_i \xrightarrow{\tau} P_i\{b/x\}$, by inductive hypothesis, $\mathbf{m}(c_ib) \,|\, c_i^*x.P_i \xrightarrow{\tau} P_i'$ with $P_i' \approx P_i\{b/x\}$ with $i = 1,2$. Then $R \rightarrow_\beta^2 (\nu\, c_1c_2)(P_1' \,|\, P_2') \equiv (P_1' \,|\, P_2') \stackrel{\text{def}}{=} R'$. By Proposition 2.1 (i) and Proposition B.2 (iii), we have $R' \,\mathcal{R}\, (P_1\{b/x\} \,|\, P_2\{b/x\})$. Hence $a^*x.(P_1 \,|\, P_2) \approx ax.(P_1 \,|\, P_2)$, as required. For the rule (IV), set $!c^*x.(P \,|\, \mathbf{m}(cx)) \stackrel{\text{def}}{=} R$. Then:

$$\begin{array}{rl} a^*x.!P \xrightarrow{av} & (\nu\, c)(\mathbf{m}(cv) \,|\, R) \\ \stackrel{\text{def}}{=} & (\nu\, c)(\mathbf{m}(cv) \,|\, c^*x.(P \,|\, \mathbf{m}(cx)) \,|\, R) \\ \stackrel{\text{def}}{=} & (\nu\, cc_1c_2)(\mathbf{m}(cv) \,|\, \mathbf{d}(cc_1c_2) \,|\, c_1^*x.P \,|\, \mathbf{fw}(c_2c) \,|\, R) \\ \rightarrow_\beta^3\approx & P\{v/x\} \,|\, (\nu c)(\mathbf{m}(cv) \,|\, R) \end{array}$$

by inductive hypothesis on P. By (iii) in Proposition B.2, we can check $!P\{v/x\} \approx (\nu\, c)(\mathbf{m}(cv) \,|\, !cx.(P \,|\, \mathbf{m}(cx)))$. By Proposition 2.1 (i), we know $R \approx !cx.(P \,|\, \mathbf{m}(cx))$, hence done with using (iii) in B.2 again.

C Essentiality of the Message Synchroniser

In this appendix, we prove the following lemma.

Proposition 3.22 (i) **(main lemma)** $\mathbf{P_{cc}} \backslash \mathbf{s}$ *has no general message synchroniser (up to* $\approx_a$*).*

PROOF. Suppose $P \in \mathbf{P_{cc}} \backslash \mathbf{s}$ is a general message synchroniser at a to b. Since $b \notin \mathsf{an}_\uparrow(P')$ for all P' s.t. $P \longrightarrow\!\!\!\!\rightarrow P'$, there are only three ways for b to be created as a new active output subject (note the case $\mathbf{fw}(db)$ amounts to $\mathbf{d}(abc)$):

(a) $\mathbf{d}(dbc) \,|\, \mathbf{m}(df) \longrightarrow \mathbf{m}(bf) \,|\, \mathbf{m}(cf)$ for some d and f.

(b) $\mathbf{b}_r(d_0f) \,|\, \mathbf{m}(d_0b) \,|\, \mathbf{m}(df) \longrightarrow \mathbf{fw}(db) \,|\, \mathbf{m}(df) \longrightarrow \mathbf{m}(bf)$ for some d_0, d and f.

(c) $\mathbf{b}_l(d_0b) \,|\, \mathbf{m}(d_0d) \,|\, \mathbf{m}(df) \longrightarrow \mathbf{fw}(db) \,|\, \mathbf{m}(df) \longrightarrow \mathbf{m}(bf)$ for some d_0, d and f.

Now assume

$$P_0 \stackrel{\text{def}}{=} P \,|\, \mathbf{m}(ae) \xrightarrow{\tau} P_1' \,|\, \mathbf{m}(ae) \cdots \xrightarrow{\tau} P_i' \,|\, \mathbf{m}(ae) \xrightarrow{\tau} P_{i+1} \cdots \xrightarrow{\tau} P_n$$

with $b \notin \mathsf{an}_\uparrow(P_i)$ $(1 \leq i \leq n-1)$ and $b \in \mathsf{an}_\uparrow(P_n)$. By the above argument, we note:

(1) either (a) $\langle \mathbf{d}(dbc),\, \mathbf{m}(df) \rangle$ is needed for for $\mathbf{m}(bf)$ in P_n, or

(b) $\langle \mathbf{b}_r(d_0f),\, \mathbf{m}(d_0d) \rangle$ or $\langle \mathbf{b}_l(d_0b),\, \mathbf{m}(d_0d) \rangle$ in P_i $(i \leq n-2)$ is needed for $\mathbf{fw}(db)$, and $\langle \mathbf{fw}(db),\, \mathbf{m}(df) \rangle$ is needed for $\mathbf{m}(bf)$ in P_n.

(2) either (a) $\langle \mathbf{c}(a^-\tilde{v}),\, \mathbf{m}(ae) \rangle \stackrel{\text{def}}{=} \langle \mathbf{d}(abc),\, \mathbf{m}(ae) \rangle$ is directly needed for $\mathbf{m}(bf)$ in P_n or (b) $\langle \mathbf{c}(a^-\tilde{v}),\, \mathbf{m}(ae) \rangle$ in $P_i' \,|\, \mathbf{m}(ae)$ is needed for $\mathbf{fw}(db)$ or $\mathbf{m}(df)$ in P_{n-1} with $i < n-1$.

Case (a): $i = n-1$. We have $P \,|\, \mathbf{m}(ae) \,|\, \mathbf{sw}(b) \equiv P_0' \,|\, \mathbf{d}(abc) \,|\, \mathbf{m}(ae) \,|\, \mathbf{sw}(b) \longrightarrow (P_0' \,|\, \mathbf{m}(be) \,|\, \mathbf{m}(ce) \,|\, \mathbf{sw}(b)) \longrightarrow (P_0' \,|\, \mathbf{m}(ce) \,|\, (\nu\, c')\mathbf{m}(ec')) \Downarrow_{e\uparrow}$, which is a contradiction.

Case (b) $1 \leq i < n-1$: There are three cases. **(i)** Suppose $\mathbf{c}(a^-\tilde{v}) \equiv \mathbf{d}(aa_{21}a_{22})$ for some a_{21}, a_{22}. As in the proof of Lemma 3.12, this case finally amounts to the following two cases (remember e is just forwarded, and if finally $\mathbf{m}(a_{ij}e)$ meets $\mathbf{d}(a_{ij}bc)$, then this is a contradiction by **Case (a)** above). Assume $k < n-1$.

(1) Case $\mathbf{b}_l(a_{kj}g)$ is needed for some g: then $(\mathbf{b}_l(a_{kj}g) \,|\, \mathbf{m}(a_{kj}e)) \longrightarrow \mathbf{fw}(eg)$, by the definition of needed redex, $\mathbf{fw}(eg)$ and $\mathbf{m}(ev)$ for some v should be needed for $\mathbf{m}(d_0d)$ or $\mathbf{fw}(db)$ or $\mathbf{m}(df)$ again. Thus there exists Q such that $(P \,|\, \mathbf{m}(ae)) \longrightarrow\!\!\!\!\rightarrow (\nu\, \tilde{c})(Q \,|\, \mathbf{fw}(eg) \,|\, \mathbf{m}(ev))$ which contradicts our assumption $\neg(P \,|\, \mathbf{m}(ae)) \Downarrow_{e\uparrow}$.

(2) Case $\mathbf{b}_r(a_{kj}g)$ is needed for some g: it is just the same as the proof in Lemma 3.12.

Cases $\mathbf{c}(a^-\tilde{v}) \equiv \mathbf{b}_r(ag)$ and $\mathbf{c}(a^-\tilde{v}) \equiv \mathbf{b}_l(ag)$ for some g are just the same with the above cases (b-1) and (b-2), respectively. □

D Proof for Proposition 4.13

First the labelled transition relation for π_c-calculus is defined by replacing (alp) to the following (str) rule and deleting (rep) rule.

$$\text{(str):}\ \frac{P{\equiv}P' \quad P' \xrightarrow{l} Q' \quad Q'{\equiv}Q}{P' \xrightarrow{l} Q'}$$

We define two functions, $\llbracket\ \rrbracket_\mathbf{c} : \mathbf{P}_{\pi_c} \rightarrow \mathbf{P}_c$, and $\llbracket\ \rrbracket_{\pi_c} : \mathbf{P}_c \rightarrow \mathbf{P}_{\pi_c}$ in the following (we write $\llbracket\ \rrbracket$ for either $\llbracket\ \rrbracket_\mathbf{c}$ or $\llbracket\ \rrbracket_{\pi_c}$).

$$\begin{array}{lll} \llbracket \overline{a}b \rrbracket_\mathbf{c} \stackrel{\text{def}}{=} \mathbf{m}(ab) & \llbracket ax.Q \rrbracket_\mathbf{c} \stackrel{\text{def}}{=} a^\star x.\llbracket Q \rrbracket_\mathbf{c} & \\ \llbracket \mathbf{m}(ab) \rrbracket_{\pi_c} \stackrel{\text{def}}{=} \overline{a}b & \llbracket \mathbf{d}(abc) \rrbracket_{\pi_c} \stackrel{\text{def}}{=} ax.(\overline{b}x \,|\, \overline{c}x) & \llbracket \mathbf{fw}(ab) \rrbracket_{\pi_c} \stackrel{\text{def}}{=} ax.\overline{b}x \\ \llbracket \mathbf{b}_r^c(ab) \rrbracket_{\pi_c} \stackrel{\text{def}}{=} ax.by.\overline{x}y & \llbracket \mathbf{b}_l(ab) \rrbracket_{\pi_c} \stackrel{\text{def}}{=} ax.xy.\overline{b}y & \llbracket \mathbf{k}(a) \rrbracket_{\pi_c} \stackrel{\text{def}}{=} ax.\mathbf{0} \\ \llbracket P \,|\, Q \rrbracket \stackrel{\text{def}}{=} \llbracket P \rrbracket \,|\, \llbracket Q \rrbracket & \llbracket (\nu\, a)P \rrbracket \stackrel{\text{def}}{=} (\nu\, a)\llbracket P \rrbracket & \llbracket \mathbf{0} \rrbracket \stackrel{\text{def}}{=} \mathbf{0} \end{array}$$

Then we have:

LEMMA D.1. *For all $P, Q \in \mathbf{P}_c$, we have:*

(i) (a) $a^\star x.(P \mid Q) \approx a^\star x.P \mid Q$ *with* $x \notin \mathsf{fn}(Q)$ *and* (b) $a^\star x.b^\star y.P \approx a^\star x.b^\star y.P$ *with* $x \neq b,\ y \neq a$

(ii) $a^\star x.P \mid \mathbf{m}(av) \longrightarrow\approx P\{v/x\}$.

(iii) $P \approx Q \ \Rightarrow\ a^\star x.P \approx a^\star x.Q$.

PROOF. (i) is by induction on P. For (a), we first prove $a^\star x.P \approx \mathbf{k}(a) \mid P$ with $a \notin \mathsf{fn}(P)$. (b) is done with (a). (ii) is proved by rule induction on $a^\star x.P$. For (iii), we only have to think the input case. Suppose $P_1 \approx P_2$. Then by (ii) above, $a^\star x.P_i \xrightarrow{av} P_i' \approx P_i\{v/x\}$, but by Proposition 2.1 (i), we have $P_1\{v/x\} \approx P_2\{v/x\}$, hence $P_1' \approx P_2'$, as desired. □

This proposition is important.

PROPOSITION D.2. .

(i) $\mathsf{fn}(P) = \mathsf{fn}([\![P]\!])$, $\mathsf{fs}_\downarrow(P) = \mathsf{fs}_\uparrow([\![P]\!])$, *and* $\mathsf{an}_\uparrow(P) = \mathsf{an}_\downarrow([\![P]\!])$

(ii) *For any substitution* σ, $[\![P\sigma]\!] \equiv_\alpha [\![P]\!]\sigma$.

(iii) $P \equiv Q \ \Rightarrow\ [\![P]\!]_{\pi_c} \equiv [\![Q]\!]_{\pi_c}$ *and* $P \equiv Q \ \Rightarrow\ [\![P]\!]_{\mathbf{c}} \approx [\![Q]\!]_{\mathbf{c}}$

(iv) (a) $P \xrightarrow{l} Q \ \Rightarrow\ [\![P]\!]_{\pi_c} \xrightarrow{l}_c [\![Q]\!]_{\pi_c}$

(b) $[\![P]\!]_{\pi_c} \xrightarrow{l}_c R \ \Rightarrow\ R \equiv [\![Q]\!]_{\pi_c}$ *with* $P \xrightarrow{l} Q$.

(v) (a) $P \xrightarrow{l}_c Q \ \Rightarrow\ \exists Q'.(Q \equiv Q' \wedge [\![P]\!]_{\mathbf{c}} \xrightarrow{l}\approx [\![Q']\!]_{\mathbf{c}})$

(b) $[\![P]\!]_{\mathbf{c}} \xrightarrow{l} R \ \Rightarrow\ \exists Q.\ (R \approx [\![Q]\!]_{\mathbf{c}}$ *with* $P \xrightarrow{l}_c Q)$.

PROOF. (i,ii,iv) are mechanical. The first cause of (iii) is evident. For the second cause, we use Proposition D.1 (ii). (v) is done by D.1 (iii). □

Now we can prove $P \Downarrow_{a\downarrow} \ \Leftrightarrow\ [\![P]\!] \Downarrow_{a\downarrow}$ by (i) and τ-case in (iv,v) above. Then Proposition 4.13 (ii) is proved similarly with Theorem 5.7 in [25]. To prove (iii), we extend the proof of Lemma 3.12.

DEFINITION D.3. Let $a \neq b$. A *right binder at a then at b* is a term P s.t. (1) $\neg P \Downarrow_{b\downarrow}$, (2) $(P \mid \mathbf{m}(ae)) \Downarrow_{b\downarrow}$, (3) $\neg(P \mid \mathbf{m}(ae)) \Downarrow_{e\downarrow}$ and (4) $(P \mid \mathbf{m}(ae) \mid \mathbf{m}(bc)) \Downarrow_{e\uparrow}$ where e is fresh in (2),(3) and (4).

Note $\mathbf{b}_r^c(ab)$ satisfies (2) (3) and (4), but does not do (1).

LEMMA D.4. *Let* $a \neq b$. $\mathbf{P}_c$ *has no right binder at* a *then* b.

PROOF. Suppose that $P \in \mathbf{P}_c$ has a right binder at a then b. Then the condition (1) and (2) imply that either (a) or (b) in the proof of Lemma 3.12 should hold. But $\mathbf{b}_r(db)$ is not included as a subterm of P, because if $\mathbf{b}_r(ff') \in \mathbf{P}_c$, then f and f' are *not* free by the form of $\mathbf{b}_r^c(hh')$. So there is only the case (a) and $i < n-1$. Suppose that $\mathbf{c}(a^- \tilde{v}) \equiv \mathbf{d}(aa_1a_2)$. Then by the same reasoning, $\mathbf{m}(ae)$ is duplicated with preserving e as a value during the interaction with a chain of $\mathbf{d}$ (or $\mathbf{fw}$), and either $\mathbf{b}_l(a_{ij}g)$ or $\mathbf{b}_l(a_{ij}g)$ is needed for $\mathbf{m}(db)$, which also contradicts (3) as same as the proof in Lemma 3.12. The case $\mathbf{c}(a^- \tilde{v}) \equiv \mathbf{b}_l(ag)$ is also similar. □

E A Counter Example on π_l-calculus

We here show that an encoding from the asynchronous π-calculus to polyadic π_l-calculus $\{\!|\ |\!\}$ defined in Table 2 in [6] is *not* fully abstract up to *barbed congruence* (hence it is not up to $\approx$ either). We write $\approx_b$ for a barbed congruence. A following counter example also works to show non-full abstraction from polyadic into monadic name passing, cf. [59]. We write $a(\tilde{x}).P$ for polyadic input and $\overline{a}\langle\tilde{v}\rangle$ for polyadic output. Let's consider the following the asynchronous π-terms.

$$x(y).\mathbf{0} \mid x(w).\mathbf{0} \ \approx_b\ x(y).x(w).\mathbf{0}$$

Hence we have:

$$ax.(x(y).\mathbf{0} \mid x(w).\mathbf{0}) \ \approx_b\ ax.x(y).x(w).\mathbf{0}$$

Then the translations are defined as:

- $\{\!|a(x).(x(y).\mathbf{0} \mid x(w).\mathbf{0})|\!\} \stackrel{\text{def}}{=} a(x,z).(\nu hh')(\overline{z}\langle h\rangle \mid h(x,y).\mathbf{0} \mid \overline{z}\langle h'\rangle \mid h'(x,y).\mathbf{0})$.
- $\{\!|a(x).x(y).x(w).\mathbf{0}|\!\} \stackrel{\text{def}}{=} a(x,z).(\nu h)(\overline{z}\langle h\rangle \mid h(x,y).((\nu h')(z\langle h'\rangle \mid h'(x,y).\mathbf{0}))$

Next let us consider the following context in π_l.

$$C[\,] \stackrel{\text{def}}{=} \overline{a}\langle x,z\rangle \mid z(y).z(y').\overline{c} \mid [\,] \quad \text{(with c fresh)}$$

$C[\{\!|a(x).(x(y).\mathbf{0}|x(w).\mathbf{0})|\!\}]$ has a visible action at c, while $C[\{\!|a(x).x(y).x(w).\mathbf{0}|\!\}]$ has no action at c because $\overline{z}\langle h'\rangle$ is under prefix "$h(x,y)$". Hence we have: $\{\!|a(x).(x(y).\mathbf{0}|x(w).\mathbf{0})|\!\} \not\approx_b \{\!|a(x).x(y).x(w).\mathbf{0}|\!\}$.