Minimality and Separation Results on Asynchronous Mobile Processes

— Representability Theorems by Concurrent Combinators —

Nobuko Yoshida*

Abstract. In [22, 23], we presented a theory of *concurrent combinators* for the asynchronous monadic π-calculus without match or summation operator [7, 19]. 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 [32] and input guarded summations [37]. The present paper shows that each of the five basic combinators introduced in [22] 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.

Key Words: mobile processes, semantics, expressive powers, combinators, the asynchronous π-calculus

1. Introduction

The calculi of mobile processes [31, 32, 35] 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 [35], many variants of this calculus have been considered for different purposes: 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 [31], 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 place in the computability hierarchy, but their encodings in π-calculus represent quite different communication protocols: since π-calculus is intended to represent interaction rather than function, the notion of expressiveness based on computable functions will be naturally too restrictive (note also that it is essentially impossible to semantically embed these λ-calculi in each other [47], suggesting how subtle semantic difference in functional computations made explicit when they are represented as processes [17, 24]). This situation is similar to the case when we consider compositional embedding between these two functional calculi, cf. [47]: however, computational behaviour in π-calculus is based on much "fine-grained" name-passing nondeterministic interactions than functional one. The question then

*Computer Science, School of Cognitive and Computing Sciences, University of Sussex, Brighton, United Kingdom, BN1 9QH. e-mail: nobuko@cogs.susx.ac.uk. Partially supported by EPSRC GR/K60701.

arises as to what are general and suitable methods to measure expressiveness 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 that 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 the world without it.

In the absence of match operator, one remarkable separation result on expressiveness was proved by Palamidessi [40]: "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 [19, 13, 21, 7, 3] is known as a powerful formalism to represent a wide repertoire of interactive computational structures: polyadic and synchronous communications [19, 7] and even input-guarded summations [37] are embeddable within this calculus. At the practical level, this expressiveness gives rise to a useful high-level concurrent programming language Pict [45] which is 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 [22, 23]. 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 [22, 23]. More concretely, we show that five atoms introduced in [22, 23], which can represent all processes in this calculus, are indeed semantically indispensable: if any one of these atoms 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, 50], it is often easier and more tractable to check representability in terms of the fixed and finite interaction of the combinators than considering interaction between arbitrary processes, cf. §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

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

[2] Closely related ideas have already been studied by Parrow to examine expressiveness of various synchronisation primitives in a non-value-passing process calculus [41]. See §6.2 for discussions.

that the minimality result is applicable to the establishment of several negative results on (the encodings into) proper subsystems of this calculus, cf. §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 [19, 22, 5, 40, 37, 29] and further issues.

This paper includes all omitted definitions and proofs of [55], and the detailed comparisons with the related work. In this full version, we newly proved that all of the main results (Theorems 2.6, 3.13, 3.17 and 5.10 and Proposition 5.6), which were formalised and proposed based on the synchronous bisimilarity in [55], also hold based on other behavioural equalities: the asynchronous bisimilarity [19] and asynchronous/synchronous reduction-based equalities [21, 49].

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; only 5 combinators, which are a 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 [31, 35] based on the notion of asynchronous name passing [19, 7]. It is a succinct yet powerful calculus for concurrent computation, into which we can soundly embed various languages and calculi, for example parallel 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 $\mathbf{N}$ be a countable set of *names* (fixed throughout the paper), ranged over by $a, b, c, ..,$ or $x, y, z, v, w, ...$ We also write $\tilde{v}$ for a sequence of names $v_1...v_n$ with $n \geq 0$.

The syntax of the calculus is given as follows.

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

$P, Q, R, \ldots$ range over the set of terms denoted by $\mathbf{P}_\pi$, which are 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. "$(\boldsymbol{\nu}\, a)P$" is a *name hiding of a in P* where the initial a binds its free occurrences in P. "$P \,|\, Q$" is a *parallel*

[3] This calculus is called *ν-calculus* in [21, 22, 23]. We call it the asynchronous π-calculus in this paper since this name is more widely in use nowadays.

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 $\mathsf{bn}(P)$. 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 $\equiv$ and the reduction relation $\longrightarrow$ and $\longrightarrow\!\!\!\!\!\rightarrow$ ($\stackrel{\text{def}}{=}\longrightarrow^* \cup \equiv$) are given in Appendix A, again following the standard definitions [35, 31, 19, 13]. We also use $\equiv_\alpha$ and $=$ 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 $P \equiv (\boldsymbol{\nu}\,\tilde{c})(\overline{a}v \mid R)$ and $a \in \mathsf{an}_\downarrow(P)$ iff $P \equiv (\boldsymbol{\nu}\,\tilde{c})(ax.Q \mid R)$ 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: $\sharp\langle \mathbf{0}, a\rangle = \sharp\langle(\boldsymbol{\nu}\,a)P,\ a\rangle = \sharp\langle ba.P,\ a\rangle = 0$ with $b \neq a$, $\sharp\langle\overline{a}v,\ a\rangle = \sharp\langle\overline{v}a,\ a\rangle = 1$ with $v \neq a$, $\sharp\langle\overline{a}a,\ a\rangle = 2$, $\sharp\langle !P,\ a\rangle = \omega$ if $a \in \mathsf{fn}(P)$ else $\sharp\langle !P,\ a\rangle = 0$, $\sharp\langle ba.P,\ b\rangle = 1 + \sharp\langle P,\ b\rangle$ with $b \neq a$, $\sharp\langle P \mid Q,\ a\rangle = \sharp\langle P,\ a\rangle + \sharp\langle Q,\ a\rangle$.

In this paper, we mainly use the *synchronous early bisimulation*, denoted by $\approx$, then we extend the results to the *asynchronous early bisimulation*, denoted by $\approx_a$. See Appendix A for the definitions of early synchronous (resp. asynchronous) transition relations denoted by $\stackrel{l}{\longrightarrow}$ (resp. $\stackrel{l}{\longrightarrow}_a$). The following fact on $\approx$ and $\approx_a$ is notable and is used throughout this paper.

Proposition 2.1. (weak bisimilarity) (i) *$\approx$ and $\approx_a$ are congruence relations* [13].
(ii) *If $P \approx Q$ then $P \Downarrow_{a\uparrow} \ \Leftrightarrow\ Q \Downarrow_{a\uparrow}$ and $P \Downarrow_{a\downarrow} \ \Leftrightarrow\ Q \Downarrow_{a\downarrow}$.*
(iii) *If $P \approx_a Q$ then $P \Downarrow_{a\uparrow} \ \Leftrightarrow\ Q \Downarrow_{a\uparrow}$.*

We will also extend the main results to more general behavioural equivalences, called *sound equalities*.[4] The sound equalities are naturally applicable to many process calculi without introducing labelled transition relations [21, 3].

Definition 2.2. (sound equality) We say a congruence $\cong$ is *reduction closed*, if it includes $\equiv$, and $P \longrightarrow\!\!\!\!\!\rightarrow P'$ implies there exists Q' such that $Q \longrightarrow\!\!\!\!\!\rightarrow Q'$ with $P' \cong Q'$. Then we say a reduction-closed congruence $\cong$ is *synchronous sound* if it *respects* $\Downarrow_{a\uparrow}$ and $\Downarrow_{a\downarrow}$, i.e. if $P \cong Q$, then $P \Downarrow_{a\uparrow} \ \Leftrightarrow\ Q \Downarrow_{a\uparrow}$ and $P \Downarrow_{a\downarrow} \ \Leftrightarrow\ Q \Downarrow_{a\downarrow}$. If a reduction-closed congruence $\cong$ respects only $\Downarrow_{a\uparrow}$, then we say $\cong$ is *asynchronous sound*.

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 the maximum synchronous (resp. asynchronous) sound congruence

[4] Fournet and Gonthier [9] recently proved that the weak asynchronous barbed congruence in [48] coincides with $=_a$.

within the family of all synchronous (resp. asynchronous) sound congruences. We denote the maximum synchronous and asynchronous sound equalities by $=_s$ and $=_a$, respectively. We summarise the relationship of these equivalence relations in the following diagram (an upward relation includes a lower one).

$$\begin{tikzcd} & =_a \arrow[dl] \arrow[dr] & \\ =_s \arrow[dr] & & \approx_a \arrow[dl] \\ & \approx & \end{tikzcd}$$

2.2. **Concurrent Combinators**

Concurrent combinators are a tractable and powerful proper subset of the asynchronous π-terms, just as **S** and **K** are of λ-terms. 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 [22, 23, 53] 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.

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

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

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

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

We write $\mathbf{c}, \mathbf{c}', ...,$ to denote these agents. $\mathbf{m}(ab)$ (*message*) carries a name b to name a, $\mathbf{d}$ (*duplicator*) distributes a message to two locations, $\mathbf{fw}$ (*forwarder*) forwards a message (thus linking two locations), $\mathbf{k}$ (*killer*) kills a message, while $\mathbf{b}_r$ (*right binder*), $\mathbf{b}_l$ (*left binder*) and $\mathbf{s}$ (*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. $\mathbf{s}$ 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. Definition 8.1.1 in [4]), as well as *subsystems*. These ideas would be generally applicable to both functional and concurrent calculi with suitable adaptation. Intuitively we say a set of terms Y is generated by its subset X if any term in Y can be written down by combination of elements in X, and Z is a basis of Y if Y is generated by Z modulo some semantic equality. A closely related idea has also been proposed and studied in [41] for a non-value passing language.

Definition 2.3. (i) (generation) Let $X \subseteq \mathbf{P}_\pi$. The set of terms *generated by* X, notation X^+, is the least set Y such that:

(1) $X \subseteq Y$.

(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) $P \in Y \;\Rightarrow\; (\boldsymbol{\nu}\, a)P \in Y$,
(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 $Y \subseteq \mathbf{P}_\pi$. Then $X \subseteq \mathbf{P}_\pi$ 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:

a (reduction-closed) system	if $P \in \mathbf{P} \;\wedge\; P \longrightarrow\!\!\!\!\!\rightarrow Q \;\;\Rightarrow\;\; Q \in \mathbf{P}$.
a system up to $\approx$	if $P \in \mathbf{P} \;\wedge\; P \longrightarrow\!\!\!\!\!\rightarrow Q \;\;\Rightarrow\;\; \exists R.\, Q \approx R \in \mathbf{P}$.
a t.c.(transition-closed)-system	if $P \in \mathbf{P} \;\wedge\; P \stackrel{\hat{l}}{\Longrightarrow} Q \;\;\Rightarrow\;\; Q \in \mathbf{P}$.
a t.c.-system 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 say $\mathbf{P}_1$ is a (*t.c.*-)*subsystem of* $\mathbf{P}_2$ (*up to* $\approx$) if $\mathbf{P}_1$ and $\mathbf{P}_2$ are (*t.c.*-)systems (*up to* $\approx$) and $\mathbf{P}_1 \subseteq \mathbf{P}_2$.

(iv) We write $Y_1 \lesssim Y_2$, if $\mathbf{P}_i = \{P \mid P \approx Q \in Y_i^{\,+}\}$ is a system ($i = 1, 2$) 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. [14, 15, 40])[5]. (iii) means a system $\mathbf{P}$ should be closed under generation and reduction. In (iv), $Y_1 \simeq Y_2$ means two systems generated by Y_1 and Y_2 have the same expressive power. Note the relation $Y_1 \lesssim Y_2$ can hold even if $Y_1 \not\subseteq Y_2$ and $Y_2 \not\subseteq Y_1$ and, moreover Y_1 and Y_2 themselves are not systems. From a programming viewpoint, if system 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 X can be used instead of Y because it is closed under evaluation.

Remark 2.4. Without "!" in (i) in Definition 2.3, the finite generation with at most 19 combinators is possible by the result in [23]. Here we include $!P$ because we are concerned with expressiveness in terms of communication behaviours. In (ii), we can use other weak equalities [19, 3, 21] instead of the synchronous bisimilarity to prove the main theorems of this paper; see Corollary 2.12 Theorems 3.22, 3.23 and 5.15. (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 [31] and the monadic synchronous π-calculus is that of polyadic π-calculus [32] etc. See §5 and §6 for more discussions on expressiveness in the π-family.

A fact related with Definition 2.3 follows.

Fact 2.5. *Let* $\mathbf{P}, \mathbf{P}_1, \mathbf{P}_2$ *be systems. Then we have:*

[5] 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 [15, 14] for details.

(i) $\lesssim$ *is a preorder and if* $\mathbf{P}_1 \subseteq \mathbf{P}_2$*, then* $\mathbf{P}_1 \lesssim \mathbf{P}_2$.
(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* $Y_1 \simeq Y_2$.
(iv) $Y \lesssim \mathbf{P}$ *implies* Y^+ *is a subsystem of* $\mathbf{P}$ *up to* $\approx$.
(v) *If* $\mathbf{P}_0$ *is a* (t.c.-)*system up to* $\approx$*, then* $\{P \mid P \approx Q \in \mathbf{P}_0\}$ *is a* (t.c.-)*system.*

Now let us define a set of five combinators as follows.

$$\mathbf{C} \stackrel{\text{def}}{=} \{\mathbf{m}(ab),\ \mathbf{d}(abc),\ \mathbf{b}_r(ab),\ \mathbf{b}_l(ab),\ \mathbf{s}(abc)\} \qquad \text{with } a, b \text{ and } c \text{ pairwise distinct.}$$

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 2.6. (finite generation) $\mathbf{C}$ *is a basis. Equivalently, we have* $\mathbf{P}_\pi \simeq \mathbf{C}$.

2.4. Proof of the Finite Generation Theorem

We first introduce the following set of atomic agents from which any combinator P is generated:

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

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.c.-subsystem of $\mathbf{P}_\pi$ by checking the transition rules of 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 [22] (the rule (IV) is newly defined in this paper). We assume the following annotations ($+$ stands for the output and $-$ stands for the input), 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.

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

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

Definition 2.7. (prefix mapping) For $P \in \mathbf{P_{cc}}$, we 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.

In (I), we distribute the received message into two decomposed prefix mappings, safely using private names. In (II) we assume that fresh names are found uniquely. In (IV), if $a^*x.!P$ interacts with $\mathbf{m}(av)$, then it creates infinite copies of Q such that $Q \approx P\{v/x\}$ (cf. Proposition 2.9). Rules (I) and (III) explain the origins of $\mathbf{d}$ and $\mathbf{k}$.

When the prefixed body is an atomic agent which does not contain any abstracted name x, the prefixing actually functions as nothing but *the control of synchronisation.* (V) is straightforward. In (VI), an interaction of the mapped term with a message results in $(\boldsymbol{\nu}\, c)(\mathbf{fw}(vc) \,|\, \mathbf{c}(c^-\tilde{w}))$ which is not syntactically the same as $\mathbf{c}(v^-\tilde{w})$ but which has essentially the same behaviour (cf. Lemma 2.11).

(I)	$a^*x.(P \mid Q)$	$\stackrel{\rm def}{=}$	$(\boldsymbol{\nu}\, c_1c_2)(\mathbf{d}(ac_1c_2) \mid c_1^*x.P \mid c_2^*x.Q)$	
(II)	$a^*x.(\boldsymbol{\nu}\, c')P$	$\stackrel{\rm def}{=}$	$(\boldsymbol{\nu}\, c)a^*x.P\{c/c'\}$	
(III)	$a^*x.\mathbf{0}$	$\stackrel{\rm def}{=}$	$\mathbf{k}(a)$	
(IV)	$a^*x.!P$	$\stackrel{\rm def}{=}$	$(\boldsymbol{\nu}\, c)(\mathbf{fw}(ac) \mid !c^*x.(P \mid \mathbf{m}(cx)))$	
(V)	$a^*x.\mathbf{c}(v^+\tilde{w})$	$\stackrel{\rm def}{=}$	$(\boldsymbol{\nu}\, c)(\mathbf{s}(acv) \mid \mathbf{c}(c^+\tilde{w}))$	$x \notin \{v\tilde{w}\}$
(VI)	$a^*x.\mathbf{c}(v^-\tilde{w})$	$\stackrel{\rm def}{=}$	$(\boldsymbol{\nu}\, c)(\mathbf{s}(avc) \mid \mathbf{c}(c^-\tilde{w}))$	$x \notin \{v\tilde{w}\}$
(VII)	$a^*x.\mathbf{m}(vx)$	$\stackrel{\rm def}{=}$	$\mathbf{fw}(av)$	$x \neq v$
(VIII)	$a^*x.\mathbf{fw}(xv)$	$\stackrel{\rm def}{=}$	$\mathbf{b}_l(av)$	$x \neq v$
(IX)	$a^*x.\mathbf{fw}(vx)$	$\stackrel{\rm def}{=}$	$\mathbf{b}_r(av)$	$x \neq v$
(X)	$a^*x.\mathbf{c}(\tilde{v_1}x^+\tilde{v_2})$	$\stackrel{\rm def}{=}$	$(\boldsymbol{\nu}\, c)a^*x.(\mathbf{fw}(cx) \mid \mathbf{c}(\tilde{v_1}c^+\tilde{v_2}))$	$x \notin \{\tilde{v_1}\}$
(XI)	$a^*x.\mathbf{c}(x^-\tilde{v})$	$\stackrel{\rm def}{=}$	$(\boldsymbol{\nu}\, c)a^*x.(\mathbf{fw}(xc) \mid \mathbf{c}(c^-\tilde{v}))$	
(XII)	$a^*x.\mathbf{b}_r(vx^-)$	$\stackrel{\rm def}{=}$	$(\boldsymbol{\nu}\, c_1c_2c_3)a^*x.(\mathbf{d}(vc_1c_2) \mid \mathbf{s}(c_1xc_3) \mid \mathbf{b}_r(c_2c_3))$	$x \neq v$
(XIII)	$a^*x.\mathbf{s}(vx^-w)$	$\stackrel{\rm def}{=}$	$(\boldsymbol{\nu}\, c_1c_2)a^*x.(\mathbf{s}(vc_1c_2) \mid \mathbf{m}(c_1x) \mid \mathbf{b}_l(c_2w))$	$x \neq v$

Table 1. Prefix Mapping

Rules (IX–XIII) are for the cases that the prefixed atom contains abstracted names. Rule (VII) is straightforward. Rules (VIII) and (XI) explain the origins of the left and right binders. In (X) and (XI), one "pushes out" the abstracted name x using the forwarder. Take the case of $a^*x.\mathbf{m}(x^+v)$, which corresponds to rule (IX). This becomes $a^*x.(\boldsymbol{\nu}\, c)(\mathbf{fw}(cx) \mid \mathbf{m}(cv))$. Thus, when a message arrives at "a", the forwarder is launched to "forward" the waiting message. (XI) acts in the opposite direction. Finally, abstracted names with negative polarities but not at subject positions, (XII) and (XIII), cannot be done easily using forwarders as in (X), due to the control of the timing of synchronisation (note that e.g. $\mathbf{b}_r(ab)$ and $(\boldsymbol{\nu}\, c)(\mathbf{b}_r(ac) \mid \mathbf{fw}(bc))$ are essentially different).

Proposition 2.8. (well-definedness) *For any a, x and P, $a^*x.P$ is well-defined and unique (up to $\equiv_\alpha$).*

Proof. We use induction on the length of P and the number of occurrences of x in P. First notice that (I) to (IX) (except (IV)) always diminish the size of the prefixed body. For (IV) and (X)–(XIII), we can unfold the right-hand side to the following terms to which the induction is applied.

(IV)	$(\boldsymbol{\nu}\, c)(\mathbf{fw}(ac) \mid !(\boldsymbol{\nu}\, c_1c_2)(\mathbf{d}(cc_1c_2) \mid c_1^*x.P \mid \mathbf{fw}(c_2c)))$
(X)	$(\boldsymbol{\nu}\, cc_1c_2)(\mathbf{d}(ac_1c_2) \mid \mathbf{b}_r(c_1c) \mid c_2^*x.\mathbf{c}(\tilde{v_1}c^+\tilde{v_2}))$
(XI)	$(\boldsymbol{\nu}\, cc_1c_2)(\mathbf{d}(ac_1c_2) \mid \mathbf{b}_l(c_1c) \mid c_2^*x.\mathbf{c}(c^-\tilde{v}))$
(XII)	$(\boldsymbol{\nu}\, c_1..c_6)(\mathbf{d}_3(ac_4c_5c_6) \mid c_4^*x.\mathbf{d}(vc_1c_2) \mid c_5^*x.\mathbf{s}(c_1xc_3) \mid c_6^*x.\mathbf{b}_r(c_2c_3))$
(XIII)	$(\boldsymbol{\nu}\, c_1..c_5)(\mathbf{d}_3(ac_3c_4c_5) \mid c_3^*x.\mathbf{s}(vc_1c_2) \mid \mathbf{fw}(c_4c_1) \mid c_5^*x.\mathbf{b}_l(c_2w))$

where $\mathbf{d}_3(ab_1b_2b_3) \stackrel{\rm def}{=} (\boldsymbol{\nu}\, c)(\mathbf{d}(acb_3) \mid \mathbf{d}(cb_1b_2))$ and we assume $c, c_1, ..., c_6$ are all fresh and distinct. In the case of (IV), we can now use the induction on P. Then in other cases, the maximum term length under prefix does not change (counting each atomic agent as having length one), but the maximum number of occurrences of an abstracted name under prefix does strictly decrease (except (X), which however is immediately well-defined if (XIII) is). This shows that the mapping $a^*x.P$ gives us a total function from $(\mathbf{N} \cup \mathbf{N}) \times \mathbf{N} \times \mathbf{P_{cc}}$ to $\mathbf{P_{cc}}$. The uniqueness is obvious since rules are applied from (I) to (XIII) by definition. □

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

Proposition 2.9. (i) $ax.P \approx a^*x.P$, (ii) $P \approx Q \;\Rightarrow\; a^*x.P \approx a^*x.Q$, *and* (iii) $a^*x.P \,|\, \mathbf{m}(av) \longrightarrow\approx P\{v/x\}$.

PROOF. (i) By rule induction in Definition 2.7. Rules except (II) and (VI) are all mechanical. For (II) and (VI), we use the (extended) β-reduction $\rightarrow_\beta$ in [22]. See Appendix B. For (ii), by Proposition 2.1 (i), we have $P \approx Q \;\Rightarrow\; ax.P \approx ax.Q$, then by (i), $a^*x.P \approx ax.P \approx ax.Q \approx a^*x.Q$, as required. (iii) is by $a^*x.P \xrightarrow{av}\approx P\{v/x\}$ which is proved as the same as (i), and Proposition 2.1 (i). □

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

$$\begin{array}{rclrclrcl} [\![\overline{a}b]\!] & \stackrel{\text{def}}{=} & \mathbf{m}(ab) & [\![ax.Q]\!] & \stackrel{\text{def}}{=} & a^*x.[\![Q]\!] & [\![P \,|\, Q]\!] & \stackrel{\text{def}}{=} & [\![P]\!] \,|\, [\![Q]\!] \\ [\![(\boldsymbol{\nu}\, a)P]\!] & \stackrel{\text{def}}{=} & (\boldsymbol{\nu}\, a)[\![P]\!] & [\![!P]\!] & \stackrel{\text{def}}{=} & ![\![P]\!] & [\![\mathbf{0}]\!] & \stackrel{\text{def}}{=} & \mathbf{0} \end{array}$$

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

Lemma 2.10. $P \approx [\![P]\!]$.

PROOF. By structural induction. Base cases, $P \equiv \mathbf{0}$ and $P \equiv \overline{a}b$, are easy. At the induction step, we use Proposition 2.9 (i) for the input prefix, while others are done by Proposition 2.1 (i). □

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. $\mathbf{fw}(ab) \approx (\boldsymbol{\nu}\, c)\mathbf{d}(abc)$ and $\mathbf{k}(b) \approx (\boldsymbol{\nu}\, c)\mathbf{b}_l(bc)$. 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* $\approx$, as proved in the following lemma.

Lemma 2.11. (substitution decomposition)

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

(2) For all $\mathbf{c}(\tilde{v}) \in \mathbf{P_{cc}}$, we have: $\mathbf{c}(\tilde{v}) \approx (\boldsymbol{\nu}\, \tilde{c})(\mathbf{c}_1(\tilde{v}_1)\sigma_1 \,|\, ... \,|\, \mathbf{c}_n(\tilde{v}_n)\sigma_n)$ for some $\mathbf{c}_i(\tilde{v}_i) \in \mathbf{C}$ and an injective renaming σ_i.

PROOF. (1) is mechanical. (2) is proved by (1). For example, we have $\mathbf{m}(aa) \approx (\boldsymbol{\nu}\, c)(\mathbf{m}(ca) \,|\, \mathbf{fw}(ca))$, $\mathbf{s}(aaa) \approx (\boldsymbol{\nu}\, c_1c_2)(\mathbf{fw}(ac_1) \,|\, \mathbf{s}(c_1ac_2) \,|\, \mathbf{fw}(c_2a))$, etc. □

Now we finish proving Theorem 2.6: 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 2.10, 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 2.5 (iii) with $\mathbf{P}_\pi \supseteq \mathbf{C}$.

To show the finite generation in the other equalities defined in § 2.1, we need to change the definition of basis and system, by replacing $\approx$ with $\approx_a$, $=_s$ and $=_a$, respectively. Then we have the following result as a corollary of Theorem 2.6 by inclusions of the relations in the diagram in §2.1.

Corollary 2.12. (finite generation) $\mathbf{C}$ *is a basis up to* $\approx_a, =_s$ *and* $=_s$.

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. Intuitively, the main theorem means there exists a program which can be described in a core-language, but not in its proper subset. In §3.7., we also show that this result can be extended even if we use weaker behavioural equalities ($\approx_a, =_s$ and $=_a$). In the following, we use $\backslash$ to denote the set difference operator.

Definition 3.1. (minimal basis) 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 w.r.t. Y.

In the following $Y\backslash\mathbf{c}$ stands 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. For example, $\mathbf{C}_7\backslash\mathbf{m}$ means $\mathbf{C}_7\backslash\{\mathbf{m}(aa), \mathbf{m}(ab)\}$.

Lemma 3.2. *Let* $Y \subseteq \mathbf{C}_7$ *For all* $\mathbf{c}(\tilde{v}) \in \mathbf{C}$, *we have:*

(i) $(\mathbf{C}_7\backslash\mathbf{c})^+$ *is a system, and* $(\mathbf{C}\backslash\mathbf{c})^+$ *is a system up to* $\approx$. *If* $\mathbf{c} \neq \mathbf{m}$, *then* $(\mathbf{C}_7\backslash\mathbf{c})^+$ *is a t.c.-system and* $(\mathbf{C}\backslash\mathbf{c})^+$ *is a t.c.-system 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 $\neg P \longrightarrow$ for all $P \in (\mathbf{C}_7\backslash\mathbf{m})^+$. Let $\mathbf{c} \neq \mathbf{m}$. Then for all l, $\mathbf{c}(\tilde{a}) \xrightarrow{l} P$ implies that P is nil, a forwarder or messages. Hence $(\mathbf{C}_7\backslash\mathbf{c})^+$ is a t.c.-system, and by Fact 2.5 (iv), $(\mathbf{C}\backslash\mathbf{c})^+$ is a t.c.-system 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. $\mathbf{fw}(ab) \approx (\boldsymbol{\nu}\, c)(\mathbf{b}_l(cb) \,|\, \mathbf{m}(ca)) \approx (\boldsymbol{\nu}\, c)(\mathbf{b}_r(ca) \,|\, \mathbf{m}(cb)))$. Then we use Fact 2.5, noting $\mathbf{C}_7\backslash\mathbf{c}$ and $\mathbf{C}\backslash\mathbf{c}$ are both bases of a system $(\mathbf{C}_7\backslash\mathbf{c})^+$. (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 $\mathbf{P}_{\mathbf{cc}}\backslash\mathbf{c}$ for $(\mathbf{C}_7\backslash\mathbf{c})^+$ with $\mathbf{c}(\tilde{v}) \in \mathbf{C}$. Note if $\mathbf{c} = \mathbf{fw}$, then $\mathbf{P}_{\mathbf{cc}}\backslash\mathbf{c}$ is not a system. By (iv) above, in order to verify the essentiality of $\mathbf{c}$, we will equivalently prove $\mathbf{P}_{\mathbf{cc}}\backslash\mathbf{c} \not\lesssim \mathbf{C}$.

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 $\forall P \in \mathbf{P_{cc}} \backslash \mathbf{m}.\ \neg P \stackrel{\overline{a}b}{\Longrightarrow}$, while we have: $\mathbf{m}(ab) \stackrel{\overline{a}b}{\longrightarrow} \mathbf{0}$. $\square$

$\mathbf{d}(abc)$ 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* $P \stackrel{l}{\Longrightarrow}\stackrel{l'}{\Longrightarrow} P'$ *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* $P \longrightarrow\!\!\!\!\rightarrow (\boldsymbol{\nu}\, \tilde{f})(Q' \,|\, \mathbf{m}(be) \,|\, \mathbf{m}(ce'))$.

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. $\square$

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

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

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 can be increased without duplicator. The answer will be given in §4.2 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* $P \in \mathbf{P_{cc}} \backslash \mathbf{b}_l$, $P \stackrel{l}{\longrightarrow} P'$ *implies* $\mathsf{fs}_\downarrow(P) \supseteq \mathsf{fs}_\downarrow(P')$.

(ii) *For all* $P \in \mathbf{P_{cc}} \backslash \mathbf{b}_r$, $P \stackrel{l}{\longrightarrow} P'$ *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 $P \in \mathbf{P_{cc}} \backslash \mathbf{b}_l$, $P \approx \mathbf{b}_l(ab)$, and $P \stackrel{ae}{\longrightarrow} P'$ with e fresh. Then by Lemma 3.2 (i), $P' \in \mathbf{P_{cc}} \backslash \mathbf{b}_l$, hence $\neg P' \stackrel{ec}{\Longrightarrow}$ by Lemma 3.6 (i). But we have $\mathbf{b}_l(ab) \stackrel{ae}{\longrightarrow} \mathbf{fw}(eb) \stackrel{ec}{\longrightarrow} \mathbf{m}(bc)$, a contradiction. The case $\mathbf{b}_r(ab)$ is just similar by changing input with output. $\square$

Lemma 3.6 simply explains the roles of $\mathbf{b}_l$ and $\mathbf{b}_r$ in terms of instantiation of subjects 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 $ax.\overline{x}b$. This will be proved in Theorem 3.17 and Proposition 4.12 (iii) later.

3.4. Synchronisation

Now we prove the most interesting and difficult part: *creating some term* (a forwarder or a message, cf. Lemma 3.15) *after synchronisation, while doing no name-instantiation*, is really essential to represent the whole behaviour of π-calculus.

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

Definition 3.8. (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) $\neg(P \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(bc)) \Downarrow_{e\updownarrow}$ with e fresh (i.e. with $e \notin \mathsf{fn}(P) \cup \{a, b\}$).

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 "$\boldsymbol{\nu}$" up to the weak bisimilarity?* The following lemma answers this question.

Lemma 3.9. (Main Lemma) $\mathbf{P_{cc}}\backslash\mathbf{s}$ *has no general synchroniser.*

3.5. Needed redex pair

To prove the main lemma, we need to track 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]). We also use this formulation to examine the parallerism of the π-calculus in §4.2.

Definition 3.10. (occurrence and needed redex)

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

$P = \mathbf{0}$ or $P = \mathbf{c}(\tilde{v}) \;\Rightarrow\; O(P) = \{\varepsilon\}$.

$P = P_1 \,|\, P_2 \;\Rightarrow\; O(P) = \{\varepsilon\} \cup \{i \cdot u \mid u \in O(P_i),\ i = 1, 2\}$.

$P = (\boldsymbol{\nu}\, a)P',\ !P' \;\Rightarrow\; O(P) = \{\varepsilon\} \cup \{1 \cdot u \mid u \in O(P')\}$.

where $=$ is the literal equality. *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.

(2) Let Δ be a pair 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 $\mathbf{c}(x^- \tilde{v}) = P/u_1$ and $\mathbf{m}(xy) = P/u_2$ 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 $\mathbf{c}'(\tilde{w})$.

(3) 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_i is directly needed for $\mathbf{c}'(\tilde{w}) = P_{i+1}/u$ for some $i \leq 0$. If $\forall j\ i \leq j \leq k.\, u \notin \Delta_j$, then we say Δ_i is *semi-directly needed* for (the occurrence of) $\mathbf{c}'(\tilde{w}) = P_k/u$ in P_k.[6]
- Write $\Delta_i \succ \Delta_j$ $(0 \leq i < j)$ if Δ_i is semi-directly 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.

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

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

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

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

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 $\mathbf{m}(ce)$. Notice there may be several needed chains for one combinator (cf. Example in §3.2). The following simple fact, however, holds because $\xrightarrow{\tau}$ is based on the labelled transition relation without $\equiv$, cf. Appendix A.

Fact 3.11. *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* $\Delta_i \stackrel{\text{def}}{=} \Delta_{i_0} \succ \cdots \succ \Delta_{i_m}$ *with* $i_m \geq 1$ *for* $\mathbf{c}(\tilde{w})$ *in* P_n. *Then we have*

(i) *for all* i_k, Δ_{i_k} *does not contains either* $\mathbf{0}$ *nor* $\mathbf{k}(\tilde{v})$.

(ii) $\mathbf{c}(\tilde{w})$ *in* P_n *does not occur under replicators.*

Various kinds of communication causality in π-calculus were defined and studied based on parametric labelled transition systems, cf. [6, 11]. Our neededness between sequences of reduction relations (τ-actions) is, however, simply defined without introducing additional information on labelled transition systems and enough for the formalisation of several causal relations between combinators in this paper (see also §4.2) .

3.6. Proofs of the Main Lemma and the Minimality Theorem

Now we prove the main lemma, Lemma 3.9. Suppose $P \in \mathbf{P_{cc}} \setminus \mathbf{s}$ is a general synchroniser at a to b. Since $b \notin \mathsf{an}_{\downarrow}(P')$ 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, with e fresh,

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

[6] I.e. $\mathbf{c}'(\tilde{w})$ in P_i remains as $\mathbf{c}'(\tilde{w})$ in P_k without interaction with any combinators.

where $\mathbf{m}(ae)$ is first consumed in the step from the i-th to the $i+1$-th process of the sequence, $b \notin \mathsf{an}_{\downarrow}(P_i)$ $(1 \leq i \leq n-1)$ and $b \in \mathsf{an}_{\downarrow}(P_n)$. 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,
- 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 $i < n-1$.

Let $\tilde{\Delta} \stackrel{\text{def}}{=} \Delta_{i_1} \succ \Delta_{i_2} \succ \cdots \succ \Delta_{i_m} (i_m \leq n)$ be a sequence of needed redex pairs for $\mathbf{fw}(bf)$ in P_n. By the second item, there should be at least one needed pair in $\tilde{\Delta}$ in which name e appears in the message, and by the first item, Δ_{i_m} is either $\langle \mathbf{b}_r(db),\ \mathbf{m}(df)\rangle$ or $\langle \mathbf{b}_l(df),\ \mathbf{m}(db)\rangle$. Suppose Δ_{i_j} is the last pair in which name e appears in the message. It cannot be that name e appears in the output subject position by the definition of the general synchroniser. So we can set $\Delta_{i_j} \stackrel{\text{def}}{=} \langle \mathbf{c}(g\tilde{c}),\ \mathbf{m}(ge)\rangle$. Then we have the following four cases.

- Case $\mathbf{c}(g\tilde{c}) = \mathbf{k}(g)$. This is impossible by Fact 3.11 (i).
- Case $\mathbf{c}(g\tilde{c}) = \mathbf{d}(ga_1a_2)$ for some a_1, a_2 (the case $\mathbf{c}(a\tilde{v}) \equiv \mathbf{fw}(ga_1)$ amounts to this case). Then by definition for needed redex pairs, at least either $\mathbf{m}(a_1e)$ or $\mathbf{m}(a_2e)$ is needed for $\mathbf{fw}(bf)$ again, hence this contradicts the assumption on Δ_{i_j}.
- Case $\mathbf{c}(g\tilde{c}) = \mathbf{b}_l(gg')$ for some g'. Then $(\mathbf{b}_l(gg') \,|\, \mathbf{m}(ge)) \longrightarrow \mathbf{fw}(eg')$, hence $(P \,|\, \mathbf{m}(ae)) \Downarrow_{e\downarrow}$, this contradicts P is a general synchroniser.
- Case $\mathbf{c}(g\tilde{c}) = \mathbf{b}_r(gg')$ for some g'. Then we have $\mathbf{b}_r(gg') \,|\, \mathbf{m}(ge) \longrightarrow \mathbf{fw}(g'e)$. If $g' = b$ (i.e. $k = i_m$), then we have:

  $$P \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(bc) \longrightarrow\!\!\!\!\!\rightarrow P_n \,|\, \mathbf{m}(bc) \equiv P'_n \,|\, \mathbf{fw}(be) \,|\, \mathbf{m}(bc) \longrightarrow P'_n \,|\, \mathbf{m}(ec) \Downarrow_{e\uparrow}$$

  which contradicts that P is a general synchroniser. If $g' \neq b$ (i.e. $k < i_m$), then by definition of needed redex pairs, $\mathbf{fw}(g'e)$ should be needed for $\mathbf{m}(df)$ or $\mathbf{m}(db)$ of Δ_{i_m} again. Thus there exists a needed message $\mathbf{m}(g'v)$ in $\tilde{\Delta}$ for some $k' < i_m$, and we get: $\mathbf{fw}(ge) \,|\, \mathbf{m}(gg') \longrightarrow \mathbf{m}(eg')$, hence $(P \,|\, \mathbf{m}(ae)) \Downarrow_{e\uparrow}$, a contradiction again. $\square$

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

Proposition 3.12. $\mathbf{s}(abc)$ *is essential.*

PROOF. First we note that if P is a general synchroniser and $P \approx Q$, then Q is also a general synchroniser by Proposition 2.1. Assume $a \neq b$ 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: $\neg(\mathbf{s}(abc) \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(bc)) \Downarrow_{e\uparrow}$, 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. $\square$

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 §4.3 for a study of the calculus with even less synchronisation. Now we reach the main theorem.

Theorem 3.13. **(Minimality)** $\mathbf{C}$ *is a minimal basis. Hence* $\mathbf{C} \backslash \mathbf{c} \lesssim_{\not\approx} \mathbf{C}$.

PROOF. By Propositions 3.3, 3.5, 3.7 and 3.12 and Fact 2.5. $\square$

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 also that we can prove the same statement of Lemma 3.2 even if we extend $\mathbf{C}\backslash\mathbf{c}$ to $\mathbf{C}\backslash\mathbf{c}_1\backslash\cdots\backslash\mathbf{c}_n$ $(0 \leq i \leq n \leq 5)$. Hence we have:

Corollary 3.14. *$Y_1 \subsetneq Y_2 \subseteq \mathbf{C}$ implies $Y_1 \lnapprox Y_2$, hence $\{Y^+ \mid Y \subseteq \mathbf{C}\}$ forms a complete lattice with $\Sigma_{0\leq n\leq 5}\ {}_5\mathrm{C}_n = 32$ elements w.r.t $\lnapprox$.*

3.7. 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 3.15. (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 $P \equiv C_n[X'\sigma_1]_1....[X'\sigma_n]_n$ for some $n \geq 1$ 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), $X \approx C_n[X'\sigma_1]_1....[X'\sigma_n]_n \approx C_n[Q\sigma_1]_1....[Q\sigma_n]_n \in (Y\backslash X)^+$, which is against the essentiality of X. $\square$

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

Proposition 3.16. (message synchroniser) *$\mathbf{C}_m \stackrel{\text{def}}{=} \mathbf{C}\backslash\mathbf{s} \cup \{\mathbf{s}_m(abc)\}$ is a minimal basis.*

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

Note $ax.by.\mathbf{0} \approx (\boldsymbol{\nu}\, c)\mathbf{s}(abc)$ is a general synchroniser in the sense of Definition 3.8. 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 3.17. (strong minimality of C) *Let $Y \stackrel{\text{def}}{=} (\mathbf{C}\setminus\mathbf{c}) \cup_{1\leq i\leq n} P_i$ with $\mathbf{c}(\tilde{a}) \in \mathbf{C}$, $P_i \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, \tilde{b}_i)\mathbf{c}(\tilde{a})$ 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 §2.3, so, by cutting off trivial cases, we only have to check the following three sets are not bases.

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

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

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

By Lemma 3.15, 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 $(\boldsymbol{\nu}\, b)\mathbf{b}_l(a'b)$ as its subterm by Lemma 3.6 (i), and we can write $P_0 \longrightarrow\!\!\!\!\rightarrow (\tilde{c})((\boldsymbol{\nu}\, b)ey.\overline{b}y \mid R)$. 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 3.9, P should include $(\boldsymbol{\nu}\, c)\mathbf{s}(dbc)$ ($\approx dx.by.\mathbf{0}$) 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. $\square$

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.[7]

3.8. **Minimality under Other Behavioural Equivalences**

In this subsection, we show that the minimality and strong minimality theorems are 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 $=_a$. The proofs for $\approx$ cannot apply to $\approx_a, =_s$ and $=_a$ directly. Remember that for the essentiality of the synchroniser $\mathbf{s}(abc)$, we used the following fact:

if $P \approx Q$ and P is a general synchroniser at a to b,
then Q is also the 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. However, first we will show *there is no general message synchroniser* (defined like a general synchroniser from $\mathbf{s}_m(abc) \stackrel{\text{def}}{=} ax.\overline{b}c$, cf. Definition 3.19 (ii)) 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 also the general message synchroniser.

Finally by Lemma 3.15, the essentiality of the message synchroniser $\mathbf{s}_m(abc)$ will be exchanged by that of $\mathbf{s}(abc)$. In the cases of sound theories, we can not observe the value of messages. But we can again prove both minimality and strong minimality using another concurrent combinator called *switcher* [21].

3.8.1. *Minimality on the Asynchronous Bisimilarity.* The essentiality of $\mathbf{m}(ab)$, $\mathbf{d}(abc)$ and $\mathbf{b}_r(ab)$ is obvious by the previous lemmas and Proposition 2.1 (i,iii). 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 3.18. *$\mathbf{b}_l(ab)$ is essential up to $\approx_a$.*

---

[7] It is uninteresting to discuss just the minimum number of π-terms rather than what combinators are indeed essential for the generation since we can easily guess that the π-calculus can be generated only from one term, for example, $P \equiv (\mathbf{m}(ab) \mid \mathbf{d}(cde) \mid \mathbf{b}_r(fg) \mid \mathbf{b}_l(hi) \mid \mathbf{s}(jkl))$. See Remark 3.19 in [55].

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. $(P \,|\, \mathbf{m}(ec)) \longrightarrow\!\!\!\!\rightarrow P'$, 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 $\mathbf{b}_l(ab) \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(ec) \approx_a P \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(ec)$. 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 since $e \notin \mathsf{fs}_{\downarrow}(P \,|\, \mathbf{m}(ae))$. $\square$

For the essentiality of $\mathbf{s}(abc)$, we show a message synchroniser $\mathbf{s}_m(abc)$ can not be generated without $\mathbf{s}(abc)$.

Definition 3.19. (i) (switcher) Let us define $\mathbf{sw}(ab) \stackrel{\text{def}}{=} ax.\overline{x}b$. We call $\mathbf{sw}(ab)$ *switcher* and write $\mathbf{sw}(a)$ for $(\boldsymbol{\nu}\, b)\mathbf{sw}(ab)$.

(ii) Let $a \neq b$. A *general message synchroniser from a to b* is a term P s.t. (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, $\mathbf{sw}(ab) \,|\, \mathbf{m}(av) \longrightarrow \mathbf{m}(vb)$, and it is generated without $\mathbf{s}(abc)$ (e.g. $\mathbf{sw}(ab) \approx (\boldsymbol{\nu}\, c)(\mathbf{b}_r(ac) \,|\, \mathbf{m}(cb)))$. 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 3.20. (i) $\mathbf{P_{cc}} \backslash \mathbf{s}$ *has no general message synchroniser up to* $\approx_a$.

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

PROOF. (i) 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. $P \,|\, \mathbf{m}(ae) \longrightarrow\!\!\!\!\rightarrow P'$ with e fresh, there are only three ways for b to be created as a new active output subject (note again 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) $\mathbf{b}_r(d_0 d) \,|\, \mathbf{m}(d_0 b) \,|\, \mathbf{m}(df) \longrightarrow \mathbf{fw}(db) \,|\, \mathbf{m}(df) \longrightarrow \mathbf{m}(bf)$ for some d_0, d and f.

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

First if $f = e$, then $P \,|\, \mathbf{m}(ae) \,|\, \mathbf{sw}(b) \longrightarrow\!\!\!\!\rightarrow P'' \,|\, \mathbf{m}(be) \,|\, \mathbf{sw}(b) \longrightarrow P'' \,|\, (\boldsymbol{\nu}\, h)\mathbf{m}(eh)$, which contradicts P is a general message synchroniser. Hence $f \neq e$. As in Lemma 3.9, let us assume $\tilde{\Delta}$ is a sequcnece of needed redex pairs for $\mathbf{m}(bf)$, and $\Delta_{i_k} \stackrel{\text{def}}{=} \langle \mathbf{c}(g\tilde{v}),\ \mathbf{m}(ge) \rangle$ is the last element in $\tilde{\Delta}$ in which name e appears in the message. Then $\mathbf{c}(g\tilde{v})$ is neither $\mathbf{k}(g), \mathbf{d}(ga_1a_2), \mathbf{fw}(ga_2)$ nor $\mathbf{b}_r(gg')$ by the same reasoning in Lemma 3.9, noting $f \neq e$. The case $\mathbf{b}_l(gg')$ is also impossible, since if so, $\Delta_{i_{k'}} = \langle \mathbf{fw}(eg'),\ \mathbf{m}(eg'') \rangle$ with $i_k < i_{k'}$ should be needed for $\mathbf{m}(bf)$, which contradicts the condition (3) of Definition 3.19.

(ii) By Proposition 3.16 and Corollary 2.12, $\mathbf{C}_m$ 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 $\mathbf{s}_m(abc)$ is a general message synchroniser. Note $\mathbf{s}_m(abc) \approx_a (\boldsymbol{\nu}\, e)(\mathbf{s}(aeb) \,|\, \mathbf{m}(ec))$ with e fresh. Then by applying Lemma 3.15 again, we now know $\mathbf{s}(abc)$ is essential w.r.t. $\mathbf{C}$. $\square$

3.8.2. *Minimality under the Synchronous/Asynchronous Maximum Sound Theories.* In all of the propositions except Proposition 3.5, we did not use observation of values for proofs. Thus all essentialities except $\mathbf{d}(abc)$ are obtained by preceeding propositions.

Proposition 3.21. $\mathbf{d}(abc)$ *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 [21], if $P =_a Q$ and $P \stackrel{\uparrow a}{\rightsquigarrow} P'$, then there exists Q', s.t. $Q \longrightarrow\!\!\!\!\rightarrow \stackrel{\uparrow a}{\rightsquigarrow} \longrightarrow\!\!\!\!\rightarrow Q'$ and $P' =_a Q'$ where $\stackrel{\uparrow a}{\rightsquigarrow}$ is defined in Appendix A.4 (an asynchronous transition relation which does not care a value of labels). Now suppose $\mathbf{d}(abc) =_a P \in \mathbf{P_{cc}} \setminus \mathbf{d}$. Then with e fresh, $\mathbf{d}(abc) \,|\, \mathbf{m}(ae) \,|\, \mathbf{sw}(b) \,|\, \mathbf{sw}(c) \longrightarrow\!\!\!\!\rightarrow \stackrel{\uparrow e}{\rightsquigarrow} \longrightarrow\!\!\!\!\rightarrow \stackrel{\uparrow e}{\rightsquigarrow} \mathbf{0}$. But two output transitions to e from $(P \,|\, \mathbf{m}(ae) \,|\, \mathbf{sw}(b) \,|\, \mathbf{sw}(c)) \in \mathbf{P_{cc}} \setminus \mathbf{d}$ are impossible by Lemma 3.4 (i). $\square$

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 3.8 (resp. Definition 3.19 (ii)) was given based on the synchronous (resp. asynchronous) convergence predicate.

Finally we have:

Theorem 3.22. (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 3.23. (strong minimality of C) *Let* $Y \stackrel{\text{def}}{=} (\mathbf{C} \setminus \mathbf{c}) \cup_{1 \le i \le n} P_i$ *with* $\mathbf{c}(\tilde{a}) \in \mathbf{C}$, $P_i \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, \tilde{b}_i)\mathbf{c}(\tilde{a})$ *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 3.17. 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. $\square$

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* [18, 5, 2, 29] in which no value is instantiated into 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 system $\mathbf{P} = \{Q \mid Q \approx R \in X^+\}$ is *separated by* P from a system $\mathbf{P}' = \{Q \mid Q \approx R \in Y^+\}$. We also say $\mathbf{P}$ is a *proper subsystem* of $\mathbf{P}'$.

By Fact 2.5 (v), Lemma 3.2 and Theorem 3.13, 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* $\mathbf{P}_\pi$*, and* $\mathbf{P}_{\backslash \mathbf{c}} \simeq \mathbf{P}_{\mathbf{cc}}\backslash\mathbf{c}$ ($\stackrel{\text{def}}{=} (\mathbf{C}_7\backslash\mathbf{c})^+$) (cf. §3.1). *Moreover with* $\mathbf{c} \neq \mathbf{m}$*,* $\mathbf{P}_{\backslash \mathbf{c}}$ *is a t.c.-system.*

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 [19, 20, 18], regarding $\overline{a}v$ as a pending message and $ax.P$ as a waiting object. But it includes a non-local feature which is prohibited in most of object-oriented languages, cf.[18]. Consider the following example.

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

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.[8] One important remark is that local polyadic name passing, branching structures [19, 22], the weak call-by-value λ-calculus [55], and typical concurrent objects [19, 52, 51, 2] can be encoded in π_l-calculus (see §4.1 in [55]). But what is the 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 2.7 without using (VIII), (XI), (XII) and (XIII), so the same statement as in Lemma 2.10 can be automatically proved. Hence the generation theorem as in Theorem 2.6 holds, and minimality and separation are given by Theorem 3.13. $\square$

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 a 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 λI-calculus [4].

[8] Such a subset was already discussed independently in [18, 19, 5, 2, 29].

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 $\sharp\langle P,\, x\rangle = 1$ and (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 $\mathbf{P}_{\mathrm{Lin}} \subsetneq \mathbf{P}_{\mathrm{Af}} \subsetneq \mathbf{P}_{\pi}$.[9] Then a natural question is what relations about expressiveness lie between parallelism and non-parallelism and between sharing and non-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*.

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

Intuitively this is similar to combinators $\mathrm{B} = \lambda xyz.x(yz)$ and $\mathrm{C} = \lambda xyz.(xz)y$ in linear and affine λ-calculi [10, 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 distributer) Let us assume a, b, c are pairwise distinct. 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) $(P \mid \mathbf{m}(ae)) \stackrel{l}{\Longrightarrow}\stackrel{l'}{\Longrightarrow}$ and $(P \mid \mathbf{m}(ae)) \stackrel{l'}{\Longrightarrow}\stackrel{l}{\Longrightarrow}$ where $l = \overline{b}e_1$ or $\overline{b}(e_1)$ and $l' = ce_2$ or $\overline{c}(e_2)$ with $\mathsf{bn}(l) \cap \mathsf{bn}(l') = \emptyset$ for some e, e_1, e_2.

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. Remind that P/u denotes a subterm of P occurs at the occurrence of u (cf. Definition 3.10).

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 $\mathbf{c}_1(\tilde{v}_1) = P_n/u_1$ and $\mathbf{c}_2(\tilde{v}_2) = P_n/u_2$ 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

$$P_0 = \mathbf{d}(abc) \mid \mathbf{m}(av) \mid \mathbf{b}_l(db) \mid \mathbf{m}(de)$$

[9] 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.

has the following τ-action sequence.

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

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 $P \xrightarrow{\tau}^{+} P' \equiv (\boldsymbol{\nu}\,\tilde{c})(\mathbf{c}_1(\tilde{v}_1)\,|\,\mathbf{c}_2(\tilde{v}_2)\,|\,R)$ 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. 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). Hence for any Δ, Δ cannot be the needed redex pair for two different occurrences of combinators $(\star)$.

Suppose that $\tilde{\Delta}_1$ and $\tilde{\Delta}_2$ are needed sequences for $\mathbf{c}_1(\tilde{v}_1)$ and $\mathbf{c}_2(\tilde{v}_2)$, respectively. Assume by contradiction there exits a needed redex pair which is in both $\tilde{\Delta}_1$ and $\tilde{\Delta}_2$. Let Δ be the last element of $\tilde{\Delta}_1$ which is also contained in $\tilde{\Delta}_2$. Since $\Delta \succ \Delta_1' \in \tilde{\Delta}_1$ and $\Delta \succ \Delta_2' \in \tilde{\Delta}_2$ imply $\Delta_1' = \Delta_2'$ by the above $(\star)$, Δ should be the last element both in $\tilde{\Delta}_1$ and $\tilde{\Delta}_2$; but this leads a contradiction by $(\star)$. $\Box$

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. $(P\,|\,\mathbf{m}(ae)) \xrightarrow{\tau}^{+} Q' \equiv (\boldsymbol{\nu}\,\tilde{c})(Q\,|\,\mathbf{m}(bf)\,|\,\mathbf{m}(cf'))$ 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 $\mathbf{m}(cf')$ in Q' by the condition of (1) in Definition 4.4, and $\mathbf{m}(bf)$ and $\mathbf{m}(cf')$ are in the different occurrences in Q' because of $b \neq c$. 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. $\Box$

Now let us define

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

Then we have:

Proposition 4.8. (i) $\mathbf{C}\backslash\mathbf{d} \lnsim \mathbf{C}_{\mathrm{Af}} \lnsim \mathbf{C}$.

(ii) $\mathbf{C}_{\mathrm{Af}}$ *is a minimal basis of π_{Lin} and π_{Af}-calculi, hence we have:*
$\mathbf{P}_{\backslash\mathbf{d}} \lnsim \mathbf{P}_{\mathrm{Lin}} \simeq \mathbf{P}_{\mathrm{Af}} \lnsim \mathbf{P}_{\pi}$.

Proof. For (i), first note that $\mathbf{d}_1(abc)$ is generated in $\mathbf{C}$ by Theorem 2.6. Hence we know $\mathbf{C}_{\mathrm{Af}} \lesssim \mathbf{C}$. Then since for all $P \in \mathbf{P}_{\mathrm{Af}}$, $P \longrightarrow\!\!\!\!\rightarrow P'$ implies $\sharp\langle P, e\rangle \geq \sharp\langle P', e\rangle$, by the same reasoning of 3.5, we have $\mathbf{C}_{\mathrm{Af}} \lnsim \mathbf{C}$. $\mathbf{C}\backslash\mathbf{d} \lesssim \mathbf{C}_{\mathrm{Af}}$ is obvious, while $\mathbf{C}\backslash\mathbf{d} \lnsim \mathbf{C}_{\mathrm{Af}}$ is obtained by Lemma 4.7 because if $P \approx Q$ and Q is a parallel distributer, then P is also a 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 2.7. Then the minimality and $\mathbf{P}_{\backslash\mathbf{d}} \lnsim \mathbf{P}_{\mathrm{Af}} \lnsim \mathbf{P}_{\pi}$ 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 2.5 (i). For the converse inclusion, we note $\mathbf{s}(abc) \notin \mathbf{P}_{\mathrm{Lin}}$ but we have $\mathbf{s}(abc) \approx ax.by.(\overline{c}y \,|\, (\boldsymbol{\nu}\, b)\overline{b}x)$. Then we use Lemma 3.15. $\square$

As a further examination of parallelism and sharing, it can be proved that 0-distributer $\mathbf{d}_0(abc) \stackrel{\mathrm{def}}{=} ax.(\boldsymbol{\nu}\, ee')(\overline{b}e \,|\, \overline{c}e')$ cannot be generated in $\mathbf{P}_{\mathbf{cc}} \backslash \mathbf{d}$ and cannot generate $\mathbf{d}_1$. More exactly, we have: $\mathbf{C} \backslash \mathbf{d} \lnapprox \mathbf{C} \backslash \mathbf{d} \cup \{\mathbf{d}_0(abc)\} \lnapprox \mathbf{C}_{\mathrm{Af}}$. Though the proper subset generated by $\mathbf{C} \backslash \mathbf{d} \cup \{\mathbf{d}_0(abc)\}$ seems to have no interest, we have the following remark about the affine local basis.

Remark 4.9. (affine local π-calculus) The finite affine local π-calculus whose minimal basis is

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

has an enough power (without replication) to embed the full *linear* and *affine* λ-calculi [10, 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.

$$(1)\ MN \quad \text{if } \mathcal{FV}(M) \cap \mathcal{FV}(N) = \emptyset \quad \text{and} \quad (2)\ \lambda x.M \quad \text{if } x \in \mathcal{FV}(M).$$

Their embedding is given based on [31] without replication.

$$\begin{array}{rcl}
[\![x]\!]u & \stackrel{\mathrm{def}}{=} & \mathbf{fw}(qx) \\
[\![\lambda x.M]\!]u & \stackrel{\mathrm{def}}{=} & (\boldsymbol{\nu}\, w)(u(z).\overline{u}[w].\mathbf{fw}(zx) \,|\, [\![M]\!]w) \\
[\![MN]\!]u & \stackrel{\mathrm{def}}{=} & (\boldsymbol{\nu}\, qr)(\overline{q}[r].q(x).\mathbf{fw}(ux) \,|\, [\![M]\!]q \,|\, [\![N]\!]r)
\end{array}$$

where $a(\tilde{v}).P$ and $\overline{a}[\tilde{v}].P$ are polyadic input and output prefixes which can be encoded in the asynchronous π-calculus (see [7, 19, 22] for the encoding). More formal analysis related to linear typing systems on this subsystem, e.g.[54], is worth studying.

4.3. Commutative π-calculus

The asynchronous π-calculus was born by deleting output synchronisation from the synchronous π-calculus [31]. 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 [8] and [33]. 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 4.10. (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.

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

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 $\stackrel{l}{\longrightarrow}$ is given in Appendix A.3. Then we write $\approx_c$ for a weak bisimilarity for π_c-calculus.[10]

The first structural rule (1) is found in [8], while the second one (2) comes from [33]. 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:

$$\begin{array}{lcll}
\overline{a}v \mid bx.ay.\overline{a}w & \equiv & \overline{a}v \mid ax.by.\overline{a}w & \text{(by (2) in Definition 4.10)} \\
& \equiv & \overline{a}v \mid ax.by.(\mathbf{0} \mid \overline{a}w) & \text{(by } P \mid \mathbf{0} \equiv P\text{)} \\
& \equiv & \overline{a}v \mid ax.(by.\mathbf{0} \mid \overline{a}w) & \text{(by (1) in Definition 4.10)} \\
& \equiv & \overline{a}v \mid ax.by.\mathbf{0} \mid \overline{a}w & \text{(by (1) in Definition 4.10)} \\
& \longrightarrow & \overline{a}v \mid by.\mathbf{0} &
\end{array}$$

We have another possible reduction from the second line as follows:

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

It seems impossible to construct a synchroniser satisfying Definition 3.8 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 $ax.by.\overline{x}y \equiv by.ax.\overline{x}y$ 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:

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

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 4.11. 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 2.7.

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

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. Now we can derive non-synchronisation of π_c-calculus using concurrent combinators again.

Proposition 4.12. (i) *π_c-calculus has no synchroniser which satisfies Definition 3.8.*

(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* $P \approx Q \;\Leftrightarrow\; [\![P]\!]_{\pi_c} \approx_c [\![Q]\!]_{\pi_c}$.

[10] $\approx_c$ is defined up to $\equiv$, as shown in Appendix A.3.

(iii) (general right binder) *Let* $a \neq b$. *A* general right binder *at* a *then at* b *is a term* P *s.t.* (1) $\neg P \Downarrow_{\bar{b}\downarrow}$, (2) $(P \,|\, \mathbf{m}(ae)) \Downarrow_{\bar{b}\downarrow}$, (3) $\neg(P \,|\, \mathbf{m}(ae)) \Downarrow_{e\updownarrow}$ *and* (4) $(P \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(bc)) \Downarrow_{e\uparrow}$ *where* e *is fresh in* (2),(3) *and* (4). *Then* $\mathbf{P}_c$ *has no general right binder at* a *then* b.

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

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 $Q \in \mathbf{P}_{\pi_c}$, $Q \Downarrow_{a\updownarrow} \;\Leftrightarrow\; [\![Q]\!]_\mathbf{c} \Downarrow_{a\updownarrow}$. Suppose towards a contradiction $P \in \mathbf{P}_{\pi_c}$ is a synchroniser from a to b. Then $\neg P \Downarrow_{\bar{b}\downarrow}$ implies $\neg[\![P]\!]_\mathbf{c} \Downarrow_{\bar{b}\downarrow}$, and $(P \,|\, \overline{a}e) \Downarrow_{\bar{b}\downarrow}$ implies $([\![P \,|\, \overline{a}e]\!]_\mathbf{c}) \stackrel{\mathrm{def}}{=} ([\![P]\!]_\mathbf{c} \,|\, \mathbf{m}(ae)) \Downarrow_{\bar{b}\downarrow}$ with e fresh. Since $[\![P]\!]_\mathbf{c} \in \mathbf{P}_c \subsetneq \mathbf{P}_{\mathbf{cc}} \backslash \mathbf{s}$, we have $[\![P]\!]_\mathbf{c} \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(bc) \Downarrow_{e\updownarrow}$ by Lemma 3.9, then $P \,|\, \overline{a}e \,|\, \overline{b}c \Downarrow_{e\updownarrow}$, which contradicts our assumption. (ii) is proved by similar reasoning as in Appendix B, see Appendix D in [55]. For (iii), we first note that $\mathbf{b}_r^c(ab)$ satisfies (2) (3) and (4), but does not satisfy (1). Suppose that $P \in \mathbf{P}_c$ has a right binder at a then b. Then the conditions (1) and (2) imply that either (a) or (b) in the proof of Lemma 3.9 should hold. But it cannot be (a) 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 (b). As the reasoning as Lemma 3.9, suppose $\tilde{\Delta}$ is a sequence of needed redex pairs of $\mathbf{fw}(bf)$ and $\Delta_{i_j} \stackrel{\mathrm{def}}{=} \langle \mathbf{c}(g\tilde{c}),\ \mathbf{m}(ge)\rangle$ is the last pair in which name e appears in the message. Then we immediately know $\mathbf{c}(g\tilde{c})$ cannot be either $\mathbf{k}(g), \mathbf{d}(ga_1a_2)$ or $\mathbf{fw}(ga_1)$. Also the cases $\mathbf{c}(g\tilde{c}) \equiv \mathbf{b}_l(gg')$ and $\mathbf{c}(g\tilde{c}) \equiv \mathbf{b}_r(gg')$ with $g' \neq b$ contradict the above assumption (3) by the same reasoning of Lemma 3.9. Hence we conclude the all cases. For (iv), the first proper inclusion is done by Lemma 3.6 (i), while the second proper inclusion is proved by (iii). $\square$

Note $\mathbf{P}_1 \stackrel{\mathrm{def}}{=} \{Q \mid Q \approx [\![P]\!]_\mathbf{c}\}$ is a t.c.-system, 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 4.13. (reflexive π-calculus) 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 [33], 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.[11] 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 2.11 in [15]. Define the encoding of reflexive π-terms by as subterms of $\mathbf{P}_\pi$ generated by:

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

[11] A similar calculus which reduces under prefixes was also discussed in [8]. Its local version is studied in [29]. Merro [28] recently showed several similar asynchronous calculi are fully abstractly embedded into the asynchronous π-calculus up to $=_a$, using the equator.

where $\mathbf{eq}(ab) \stackrel{\text{def}}{=} !\mathbf{fw}(ab) \mid !\mathbf{fw}(ba)$ is an *equator* originally introduced in [21]. 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 $(\boldsymbol{\nu}\, a)(P \mid \mathbf{eq}(ab))$, and $a(x)P$ for $(\boldsymbol{\nu}\, x)(ay.\mathbf{eq}(yx) \mid P)$ with y fresh. Then we have:

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

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

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

(cf. Proposition 4.3 in [21]), so that the same result holds as in Proposition 4.12. Thus this subset of the asynchronous π-calculus can simulate a reflexive behaviour found in [33] up to $=_a$.[12] See Remark 5.12 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 abstract (i.e. $[\![P]\!] \approx [\![Q]\!] \Leftrightarrow P \approx Q$) or adequate (i.e. $[\![P]\!] \approx [\![Q]\!] \Rightarrow P \approx Q$) [22, 19, 37, 40, 5, 28]. 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* $\mathbf{P_{cc}}$*, 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 [40], reduction-closed [21, 48] 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 at least *output* and *parallelism* can not be taken away to embed π-calculi.

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 [19, 22, 5, 40, 37]. Hereafter "systems" etc. denote subsystems of the full polyadic synchronous π-calculus $\mathbf{P}_{\text{full}}$ (with match and mixed summation operators [32]), defined as in Definition 2.3 (iii).

Definition 5.1. (standard encoding) Let $\mathbf{P}_1$ and $\mathbf{P}_2$ be systems (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. $[\![P \mid Q]\!] \stackrel{\text{def}}{=} [\![P]\!] \mid [\![Q]\!]$, $[\![(\boldsymbol{\nu}\, \tilde{c})P]\!] \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, \tilde{c})[\![P]\!]$, $[\![!P]\!] \stackrel{\text{def}}{=} ![\![P]\!]$ and $[\![P\sigma]\!] \stackrel{\text{def}}{=} [\![P]\!]\sigma$ with σ an injective renaming, and $[\![\mathbf{0}]\!] \stackrel{\text{def}}{=} \mathbf{0}$.

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

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

[12] 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 one would expect.

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

(1) and (2) nearly correspond to uniform and reasonable conditions in [40] (but we do not require divergent-sensitivity). (3) describes the standard operational closure properties found in almost existing adequate encodings, cf. [31, 19, 37]. 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 asynchronous notion of convergence is more standard in asynchronous π-family [21, 3, 37]. We will study other standard encodings based on asynchronous bisimulation and sound equalities in §5.4. Note that: (1) $\mathbf{P}_1 \subseteq \mathbf{P}_2$ implies $\mathbf{P}_1 \lesssim^{\mathrm{e}} \mathbf{P}_2$ with an identity mapping, and (2) $\lesssim^{\mathrm{e}}$ is a preorder.

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 $\mathbf{P}_1 \lesssim^{\mathrm{e}} \mathbf{P}_2$ is related to the existence of an adequate encoding from $\mathbf{P}_1$ into $\mathbf{P}_2$ up to the reduction-based equalities, as will be proved in Proposition 5.14 (ii).

Proposition 5.2. (relationship with Definition 2.3) *Let us assume a mapping* $[\![\ ]\!]$ *from a system* $\mathbf{P}_1$ *to a system* $\mathbf{P}_2$. *Then:*

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

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

(a) $P \xrightarrow{l} P' \;\Rightarrow\; \exists Q.\, [\![P]\!] \stackrel{l}{\Longrightarrow} Q \,\wedge\, Q \approx [\![P']\!]$ 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.c.-system up to* $\approx$.

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

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

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 [5]. A more important fact on the relationship between $\lesssim^{\mathrm{e}}$ and $\lesssim$ follows.

Fact 5.3. *Let* $\mathbf{P}_1$ *and* $\mathbf{P}_2$ *be systems. Then* $\mathbf{P}_1 \lesssim^{\mathrm{e}} \mathbf{P}_2 \;\not\Rightarrow\; \mathbf{P}_1 \lesssim \mathbf{P}_2$.

PROOF. By Proposition 4.3 and Proposition 6.1 (ii) in §6.2. $\Box$

However, as expected, we have:

Proposition 5.4. *Assume* $\mathbf{P}_1$ *and* $\mathbf{P}_1$ *are systems 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 $[P_1]_\approx \stackrel{\text{def}}{=} \{P_2 \mid P_1 \approx P_2,\ \ P_1 \in \mathbf{P}_1,\ P_2 \in \mathbf{P}_2\}$. We define $[\![\,]\!] : \mathbf{P}_1 \to \mathbf{P}_2$ as

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

Then $[\![\,]\!]$ 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 $[\![\ ]\!]$ satisfies $P_1 \approx Q_1 \ \Leftrightarrow\ [\![P_1]\!] \approx [\![Q_1]\!]$ because of $[\![P_1]\!] \approx P_1$. Then (2) automatically hold.

Finally we have to check $[\![\ ]\!]$ satisfies (3). We divide (a) into two cases (note $\equiv \subseteq \approx$ and $[\![\ ]\!]$ is fully abstract).

(i) the case $\longrightarrow\!\!\!\!\!\rightarrow \stackrel{\text{def}}{=} \equiv$: $P \equiv P' \ \Rightarrow\ [\![P]\!] \approx P \equiv [\![P']\!] \approx P'$, hence $[\![P]\!] \approx [\![P']\!]$. Then this satisfies $[\![P]\!] \longrightarrow\!\!\!\!\!\rightarrow^0 [\![P]\!] \approx [\![P']\!]$.

(ii) the case $\longrightarrow\!\!\!\!\!\rightarrow \stackrel{\text{def}}{=} \longrightarrow$ is by definition of $\approx$. (b) is also similarly proved. □

Notice by Fact 5.3, we have: $\mathbf{P}_1 \not\lesssim \mathbf{P}_2 \ \not\Rightarrow\ \mathbf{P}_1 \not\lesssim^{\mathrm{e}} \mathbf{P}_2$. Thus we do 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 property generally holds in any renaming closed homomorphism [15, 14].

Lemma 5.5. (name decreasing, Proposition 2.8 in [15]) *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 $[\![\ ]\!]$ satisfies (1) in Definition 5.1, then we have $\mathsf{fn}(P) \supset \mathsf{fn}([\![P]\!])$.*

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

Proposition 5.6. *Suppose $\mathbf{P}$ is a subsystem of $\mathbf{P}_{\text{full}}$ with $\mathbf{P}_\pi \subseteq \mathbf{P}$ and $\mathbf{P}'$ is any proper subsystem studied in Sections 3 and 4, i.e. $\mathbf{P}'$ is either $\mathbf{P}_{\mathbf{cc}} \setminus \mathbf{c}$, $\mathbf{P}_{\setminus\mathbf{c}}$, $\mathbf{P}_{\text{Af}}$, $\mathbf{P}_{\text{Lin}}$, or $\mathbf{P}_c$ with $\mathbf{c}(\tilde{v}) \in \mathbf{C}$. Then there is no standard mapping $[\![\ ]\!] : \mathbf{P} \to \mathbf{P}'$ which satisfies either:*

(a) (message preserving) $[\![\overline{a}b]\!] \approx \overline{a}b$, *or*

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

Proof. First note that $\mathbf{P}_{\setminus\mathbf{c}} \simeq \mathbf{P}_{\mathbf{cc}} \setminus \mathbf{c}$ and $\mathbf{P}_{\mathbf{cc}} \setminus \mathbf{c} \not\lesssim^{\mathrm{e}} \mathbf{P}_{\mathbf{cc}}$ implies $\mathbf{P}_{\setminus\mathbf{c}} \not\lesssim^{\mathrm{e}} \mathbf{P}_\pi$ by Proposition 5.4, and $\mathbf{P}_{\setminus\mathbf{d}} \not\lesssim \mathbf{P}_{\text{Lin}} \simeq \mathbf{P}_{\text{Af}}$ and $\mathbf{P}_{\setminus\mathbf{s}} \not\lesssim \mathbf{P}_c$ by Propositions 4.8 and 4.12, respectively. Hence we only have to show: (1) $\mathbf{P}_{\mathbf{cc}} \setminus \mathbf{m} \not\lesssim^{\mathrm{e}} \mathbf{P}_{\mathbf{cc}}$, (2) $\mathbf{P}_{\text{Af}} \not\lesssim^{\mathrm{e}} \mathbf{P}_{\mathbf{cc}}$, (3) $\mathbf{P}_{\mathbf{cc}} \setminus \mathbf{b}_r \not\lesssim^{\mathrm{e}} \mathbf{P}_{\mathbf{cc}}$, (4) $\mathbf{P}_{\mathbf{cc}} \setminus \mathbf{b}_l \not\lesssim^{\mathrm{e}} \mathbf{P}_{\mathbf{cc}}$, and (5) $\mathbf{P}_c \not\lesssim^{\mathrm{e}} \mathbf{P}_{\mathbf{cc}}$. In the case of (1), we will prove a stronger result in the next subsection. Below "(1)", "(2)" and "(3)" denote the conditions in Definition 5.1. We have four cases for (a).

Case (2) $\mathbf{d}(abc)$: Assume there is a mapping $[\![\mathbf{d}(abc)]\!] \in \mathbf{P}_{\mathrm{Af}}$. Then with e fresh and a, b, c distinct, $\mathbf{d}(abc) \mid \mathbf{m}(ae) \longrightarrow \mathbf{m}(be) \mid \mathbf{m}(ce)$ implies

$$[\![\mathbf{d}(abc) \mid \mathbf{m}(ae)]\!] \approx [\![\mathbf{d}(abc)]\!] \mid \overline{a}e \longrightarrow^{+} P' \approx [\![\mathbf{m}(be) \mid \mathbf{m}(ce)]\!] \approx \overline{b}e \mid \overline{c}e$$

by (1,3) and Proposition 2.1 (note $\longrightarrow^{+}$ is obtained by $[\![\mathbf{d}(abc) \mid \mathbf{m}(ae)]\!] \not\approx \overline{b}e \mid \overline{c}e$ because $[\![\mathbf{d}(abc) \mid \mathbf{m}(ae)]\!] \Downarrow_{a\updownarrow}$ but $\neg[\![\mathbf{m}(be) \mid \mathbf{m}(ce)]\!] \Downarrow_{a\updownarrow}$ by (2)). Hence P' has two output transitions: $P' \stackrel{\overline{b}e}{\Longrightarrow}\stackrel{\overline{c}e}{\Longrightarrow}$, which implies $[\![\mathbf{d}(abc)]\!] \mid \overline{a}e \longrightarrow\!\!\!\!\!\rightarrow (\boldsymbol{\nu}\, \tilde{f})(\overline{b}e \mid \overline{c}e \mid P_1)$ by Lemma 3.4 (ii) (note e occurs more than twice). But by Lemma 5.5, we have $e \notin [\![\mathbf{d}(abc)]\!]$, hence $\sharp\langle [\![\mathbf{d}(abc)]\!] \mid \overline{a}e,\ e\rangle = 1$. Since $([\![\mathbf{d}(abc)]\!] \mid \overline{a}e) \in \mathbf{P}_{\mathrm{Af}}$, for any Q s.t. $[\![\mathbf{d}(abc)]\!] \mid \overline{a}e \longrightarrow\!\!\!\!\!\rightarrow Q$, we have $\sharp\langle Q,\ e\rangle \le 1$ which is a contradiction.

Case (3) $\mathbf{b}_r$: Assume there exists $[\![\mathbf{b}_r(ab)]\!] \in \mathbf{P}_{\mathbf{cc}} \backslash \mathbf{b}_r$. Then, with a, b, c, e distinct, we have: $[\![\mathbf{b}_r(ab) \mid \mathbf{m}(ae) \mid \mathbf{m}(bc)]\!] \Downarrow_{e\uparrow}$ by (2), hence $[\![\mathbf{b}_r(ab)]\!] \mid \mathbf{m}(ae) \mid \mathbf{m}(bc) \Downarrow_{e\uparrow}$. But this contradicts to Lemma 3.6 (i) because $e \notin \mathsf{fs}_{\uparrow}([\![\mathbf{b}_r(ab)]\!]) \subseteq \mathsf{fn}([\![\mathbf{b}_r(ab)]\!])$ by Lemma 5.5.

Case (4) $\mathbf{b}_l$: The same as above.

Case (5) $\mathbf{s}$: Assume there is a mapping $[\![\mathbf{s}(abc)]\!] \in \mathbf{P}_c$. Then with e fresh and $a \neq b$, $[\![\mathbf{s}(abc)]\!]$ should satisfy:

- $\neg\mathbf{s}(abc) \Downarrow_{b\downarrow} \Rightarrow \neg[\![\mathbf{s}(abc)]\!] \Downarrow_{b\downarrow}$ by (2),
- $\mathbf{s}(abc) \mid \mathbf{m}(ae) \Downarrow_{b\downarrow} \Rightarrow [\![\mathbf{s}(abc) \mid \mathbf{m}(ae)]\!] \stackrel{\mathrm{def}}{=} [\![\mathbf{s}(abc)]\!] \mid [\![\mathbf{m}(ae)]\!] \approx [\![\mathbf{s}(abc)]\!] \mid \mathbf{m}(ae) \Downarrow_{b\downarrow}$, by (1,2) and (a), and
- $\neg\mathbf{s}(abc) \mid \mathbf{m}(ae) \mid \mathbf{m}(bc) \Downarrow_{e\updownarrow} \Rightarrow \neg[\![\mathbf{s}(abc)]\!] \mid \mathbf{m}(ae) \mid \mathbf{m}(bc) \Downarrow_{e\updownarrow}$ by the similar reasoning to the above.

Hence $[\![\mathbf{s}(abc)]\!]$ itself is a general synchroniser, which contradicts Lemma 4.12 (i).

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 $R \mid \overline{a}e \longrightarrow^{+} R'$, hence $([\![\mathbf{s}(abc)]\!] \mid \overline{a}e) \longrightarrow P_1 \longrightarrow P_2 \longrightarrow \ldots \longrightarrow P_n \stackrel{\mathrm{def}}{=} P$ with $\neg P_i \Downarrow_{e\updownarrow}$ for all $1 \le i \le n$ by (b) in 5.2. Similarly for all Q s.t. $[\![\mathbf{s}(abc)]\!] \mid \overline{a}e \mid \overline{b}e' \longrightarrow\!\!\!\!\!\rightarrow Q$, we have $\neg Q \Downarrow_{e\updownarrow}$, which contradicts Lemma 4.12 (i) 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 [19, 22, 23, 37, 28] (see also §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 systems without messages or without duplicators. The first negative result is easy as follows.

Fact 5.7. *There is no mapping* $[\![\ ]\!] : \mathbf{P}_\pi \to \mathbf{P}'$ *with* $\mathbf{P}' \subseteq \mathbf{P}_{\backslash\mathbf{m}}$ *which satisfies (1,2) in Definition 5.1. Hence we have* $\mathbf{P}_{\backslash\mathbf{m}} \lesssim^{\mathrm{e}} \!\!\!\!\!/\ \ \mathbf{P}_\pi$.

PROOF. For any $P \in \mathbf{P}_{\backslash\mathbf{m}}$ and a, $P \Downarrow_{a\uparrow}$ is impossible, while we should have $[\![\mathbf{m}(ab)]\!] \Downarrow_{a\uparrow}$ to satisfy (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.

Definition 5.8. (the number of outputs) Define $\max_{\uparrow}(P,\, a)$, the *maximum number of outputs at a from P*, as:

$$\max_{\uparrow}(P,\, a) \stackrel{\mathrm{def}}{=} \max\{\, n \mid \quad P \stackrel{\mathrm{def}}{=} P_0 \stackrel{l_1}{\Longrightarrow} P_1 \stackrel{l_2}{\Longrightarrow} P_2 \stackrel{l_3}{\Longrightarrow} \cdots P_{n-1} \stackrel{l_n}{\Longrightarrow} P_n \\ \text{with } l_i = \overline{a}e_i \text{ or } l_i = \overline{a}(e_i) \text{ for some } e_i \text{ for all } 1 \le i \le n \,\}$$

Lemma 5.9. (the number of outputs)

(i) *If $n \le \max_{\uparrow}(P,\, a)$, then $P \longrightarrow\!\!\!\!\!\rightarrow (\boldsymbol{\nu}\,\tilde{c})(\prod_n \overline{a}e_i \,|\, R)$ for some e_i and R with $a \notin \{\tilde{c}\}$ where $\prod_n Q_i$ is an abbreviation for $\prod_n Q_i \stackrel{\mathrm{def}}{=} Q_1 \,|\, Q_2 \,|\, \cdots \,|\, Q_n$ (if $n = 0$, $\prod_0 Q_i \stackrel{\mathrm{def}}{=} \mathbf{0}$).*

(ii) *$P \approx Q$ implies $\max_{\uparrow}(P,\, a) = \max_{\uparrow}(Q,\, a)$ for all a.*

(iii) *Suppose $[\![\ ]\!]$ is a standard encoding from $\mathbf{P_{cc}}$ into some system $\mathbf{P}$. Then $[\![\mathbf{m}(ae)]\!] \longrightarrow\!\!\!\!\!\rightarrow Q \;\Rightarrow\; [\![\mathbf{m}(ae)]\!] \approx Q$.*

Proof. (i) is similar to Lemma 3.4 (ii). For (ii), suppose $P \approx Q$ and $\max_{\uparrow}(P,\, a) = n \lneq \omega$ for some a. Then by (i), we have:

$$P \longrightarrow\!\!\!\!\!\rightarrow (\boldsymbol{\nu}\,\tilde{c})(\textstyle\prod_n \overline{a}e_i \,|\, P') \stackrel{\mathrm{def}}{=} P_0 \stackrel{l_1}{\longrightarrow} P_1 \stackrel{l_2}{\longrightarrow} P_2 \stackrel{l_3}{\longrightarrow} \cdots P_{n-1} \stackrel{l_n}{\longrightarrow} P_n$$

for some $\tilde{c}, e_i$ and P' with $l_i = \overline{a}e_i$ or $l_i = \overline{a}(e_i)$. Then by the definition of the bisimilarity, there exits a sequence of n times output transitions form Q such that:

$$Q \stackrel{\mathrm{def}}{=} Q_0 \stackrel{l_1}{\Longrightarrow} Q_1 \stackrel{l_2}{\Longrightarrow} \cdots Q_{n-1} \stackrel{l_n}{\Longrightarrow} Q_n$$

with $P_i \approx Q_i$ for all $1 \le i \le n$. Hence $n \le \max_{\uparrow}(Q,\, a)$. Conversely, $\max_{\uparrow}(Q,\, a) \le n$ is proved by inspecting the output transition relation from Q. We can easily prove that if $\max_{\uparrow}(P,\, a) = \omega$ and $\max_{\uparrow}(Q,\, a) = n \lneq \omega$, then we have a contradiction by the same reasoning as the above. (iii) is by Definition 5.1 (3). $\square$

Now we prove that without $\mathbf{d}(abc)$ we can not construct any standard encodings of π-calculus. We also need Lemma 4.6 (independence) studied in §4.2.

Theorem 5.10. (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 we have $\mathbf{P}_{\backslash\mathbf{d}} \lnsim^{\mathrm{e}} \mathbf{P}_\pi$.*

Proof. It is enough to show there is no standard encoding of $\mathbf{d}$ into $\mathbf{P_{cc}} \backslash \mathbf{d}$. Suppose by contradiction there exist a standard encoding $[\![\ ]\!]$ from $\mathbf{P_{cc}}$ to $\mathbf{P_{cc}} \backslash \mathbf{d}$. We have two cases. In the following, we assume a, b and c are pairwise distinct.

Case (1) $1 \le n = \max_{\uparrow}([\![\mathbf{m}(ae)]\!],\, a) \lneq \omega$. Then by Lemma 5.9 (i,iii), for some e_i, $\tilde{c}$ and Q, we have:

$$[\![\mathbf{d}(abc)]\!] \,|\, [\![\mathbf{m}(ae)]\!] \longrightarrow\!\!\!\!\!\rightarrow [\![\mathbf{d}(abc)]\!] \,|\, (\boldsymbol{\nu}\,\tilde{c})(\textstyle\prod_n \mathbf{m}(ae_i) \,|\, Q) \stackrel{\mathrm{def}}{=} R \approx [\![\mathbf{d}(abc)]\!] \,|\, [\![\mathbf{m}(ae)]\!] \tag{2}$$

Hence by Definition 5.1 (3-b), there exists R' such that

$$R \stackrel{\Delta_1}{\longrightarrow} \cdots \stackrel{\Delta_m}{\longrightarrow} R' \approx [\![\mathbf{m}(be) \,|\, \mathbf{m}(ce)]\!] \quad \text{with} \quad 1 \le m \lneq \omega \tag{3}$$

Since the standard mapping and the early transition relation $\stackrel{l}{\longrightarrow}$ are closed under injective renaming, we know $\max_{\uparrow}([\![\mathbf{m}(ae)]\!],\, a) = \max_{\uparrow}(R',\, b) = \max_{\uparrow}(R',\, c) = n$

by Lemma 5.9 (ii). So by Lemma 5.9 (i) again, for some d_i, d'_i, $\tilde{c}'$ and Q', Q'', we have:

$$R \xrightarrow{\Delta_1} \cdots \xrightarrow{\Delta_m} R' \xrightarrow{\tau}{}^{*} Q' \equiv (\boldsymbol{\nu}\, \tilde{c}')(\textstyle\prod_n(\mathbf{m}(bd_i) \,|\, \mathbf{m}(cd'_i)) \,|\, Q'') \qquad (4)$$

Note $\mathbf{m}(bd_1), ..., \mathbf{m}(bd_n), \mathbf{m}(cd'_1), ..., \mathbf{m}(cd'_n)$ are in different occurrences in Q', since if not, at least two of $\mathbf{m}(bd_1), ..., \mathbf{m}(bd_n)$ or two of $\mathbf{m}(cd'_1), ..., \mathbf{m}(cd'_n)$ are under !, which contradicts Fact 3.11 (ii) (note that there is no less than one needed redex pair to each $\mathbf{m}(bd_i)$ and $\mathbf{m}(cd'_i)$ because $\neg[\![\mathbf{d}(abc)]\!] \Downarrow_{b\uparrow}$ and $\neg[\![\mathbf{d}(abc)]\!] \Downarrow_{c\uparrow}$). Now let us define $\tilde{\Delta}_{bi}$ is a sequence of the needed redex pairs for $\mathbf{m}(bd_i)$ in Q' from R in (4) and $\tilde{\Delta}_{ci}$ is one for $\mathbf{m}(cd'_i)$ in Q' in (4) from R. Note that a message with a subject name a does not appear in a sequent of redex pairs from R' to Q' since $\neg R' \Downarrow_{a\uparrow}$. Then for all $1 \leq i \leq n$, at least one message at a in R (i.e. one of $\prod_n \mathbf{m}(ae_i)$) should be in $\tilde{\Delta}_{bi}$. Similarly for $\tilde{\Delta}_{ci}$. However by Lemma 4.6, it is impossible that n messages are needed for $2n$ different occurrences.

Case (2) $\max_{\uparrow}([\![\mathbf{m}(ae)]\!],\ a) = \omega$. By Definition 5.1 (3), there exists R such that:

$$[\![\mathbf{m}(ae)]\!] \,|\, [\![\mathbf{d}(abc)]\!] \stackrel{\text{def}}{=} R \xrightarrow{\Delta_1} \cdots \xrightarrow{\Delta_m} R' \quad \text{with} \quad R' \approx [\![\mathbf{m}(be)]\!] \,|\, [\![\mathbf{m}(ce)]\!] \qquad (5)$$

Since $[\![\mathbf{m}(ae)]\!] \Downarrow_{a\uparrow}$ but $\neg R' \Downarrow_{a\uparrow}$, $\tilde{\Delta} \stackrel{\text{def}}{=} \Delta_1, ..., \Delta_m$ contains no less than one message at a. Let us define n is the number of messages at a which appear in $\tilde{\Delta}$. Hence $n \leq m \lneq \omega$. Note that by injective renaming, $\max_{\uparrow}([\![\mathbf{m}(ae)]\!],\ a) = \max_{\uparrow}([\![\mathbf{m}(be)]\!],\ b) = \max_{\uparrow}([\![\mathbf{m}(ce)]\!],\ c) = \omega$. Hence we have the τ-transition relations from R' which satisfy (4) in **Case (1)**. Note that $\mathbf{m}(bd_1), ..., \mathbf{m}(bd_n), \mathbf{m}(cd'_1), ..., \mathbf{m}(cd'_n)$ are in different occurrences in Q' by Fact 3.11 (ii) again. Hence by similar argument, at least one of the message at a in $\tilde{\Delta}$ is needed for $\mathbf{m}(bd_i)$ and for $\mathbf{m}(cd'_i)$ for all i such that $1 \leq i \leq n$, which again contradicts Lemma 4.6. □

Finally we believe the following conjecture.

Conjecture 5.11. (1) (synchronisation) *There is no standard encoding from* $\mathbf{P}_\pi$ *to any subsystem of* $\mathbf{P}_{\backslash \mathbf{s}}$, *hence* $\mathbf{P}_{\backslash \mathbf{s}} \lnsim^{\mathrm{e}} \mathbf{P}_\pi$.

(2) (sharing of names) *There is no standard encoding from* $\mathbf{P}_\pi$ *to any subsystem of* $\mathbf{P}_{\mathrm{Af}}$, *hence* $\mathbf{P}_{\mathrm{Af}} \lnsim^{\mathrm{e}} \mathbf{P}_\pi$.[13]

(3) (full abstraction) *There is no fully abstract standard encoding* (*up to* $\approx$) *from* $\mathbf{P}_\pi$ *into any proper subsystem* $\mathbf{P} \lnsim \mathbf{C}$.

(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 π-calculus into monadic name-passing.[14]

Remark 5.12. (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

[13] One may also have a conjecture: $\mathbf{P}_{\backslash \mathbf{b}_r} \lnsim^{\mathrm{e}} \mathbf{P}_\pi$. It may be impossible to prove since $\mathbf{P}_{\backslash \mathbf{b}_l} \simeq^{\mathrm{e}} \mathbf{P}_\pi$ as shown in Proposition 6.1 (ii).

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

[19, 22, 5, 37, 29, 31] 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 4.13), we *cannot* construct any one-step standard encoding of the whole π-calculus.

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

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 5.13.
- $[\![\ ]\!]$ is *a-standard encoding* if it is defined by replacing (2) in Definition 5.1 by $P \Downarrow_{a\uparrow} \quad \Leftrightarrow \quad [\![P]\!] \Downarrow_{a\uparrow}$ and $\approx$ in (3) in Definition 5.1 by 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 by the synchronous maximum sound equality $=_s$. Similarly *sound a-standard encoding* is defined by replacing (2) by $P \Downarrow_{a\uparrow} \quad \Leftrightarrow \quad [\![P]\!] \Downarrow_{a\uparrow}$ $\approx$ 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.

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

Proposition 5.14. (relationship with Definition 2.3)

(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 $\stackrel{l}{\longrightarrow}$ with $\stackrel{l}{\longrightarrow}_a$.*

(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 $\mathcal{R} \stackrel{\text{def}}{=} \{\langle P,\ Q\rangle \mid [\![P]\!] =_a [\![Q]\!]\}$ is congruent,[15] it is adequate (up to $=_s$ and $=_a$, respectively).*

Proposition 5.15. *Let $\mathbf{P}_\pi \subseteq \mathbf{P}$ be a subsystem of $\mathbf{P}_{\text{full}}$.*

(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 $[\![\ ]\!] : \mathbf{P} \to \mathbf{P}'$ 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 $\mathbf{P}' \in \mathbf{P}_{\mathbf{cc}}\backslash\mathbf{m}$ or $\mathbf{P}' \in \mathbf{P}_{\mathbf{cc}}\backslash\mathbf{d}$. Then there is neither a-standard, sound standard, nor sound a-standard mapping. Hence we have: (1) $\mathbf{P}_{\backslash\mathbf{m}} \not\lesssim^{\mathrm{e}}_a \mathbf{P}_\pi$, $\mathbf{P}_{\backslash\mathbf{m}} \not\lesssim^{\mathrm{e}}_s \mathbf{P}_\pi$, $\mathbf{P}_{\backslash\mathbf{m}} \not\lesssim^{\mathrm{e}}_{sa} \mathbf{P}_\pi$, (2) $\mathbf{P}_{\backslash\mathbf{d}} \not\lesssim^{\mathrm{e}}_a \mathbf{P}_\pi$, $\mathbf{P}_{\backslash\mathbf{d}} \not\lesssim^{\mathrm{e}}_s \mathbf{P}_\pi$, and $\mathbf{P}_{\backslash\mathbf{d}} \not\lesssim^{\mathrm{e}}_{sa} \mathbf{P}_\pi$.*

[15] More precisely, it is enough that this relation is closed under structural rules and substitutions (cf. [54]).

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 $[\![\mathbf{d}(abc) \,|\, \mathbf{m}(ae) \,|\, \mathbf{sw}(b) \,|\, \mathbf{sw}(c)]\!]$ (e fresh) instead of $[\![\mathbf{d}(abc) \,|\, \mathbf{m}(ae)]\!]$ in the proof of Proposition 5.6. Then use the similar reasoning as in the proof of Proposition 3.21. 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 to Proposition 3.18. 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) (1) is obvious. For (2), we note that the proofs of Lemma 5.9 and Theorem 5.10 are only concerned with the number of subject of messages; to define $\max_{\uparrow}(P,\,a)$, we can use $\overset{\uparrow a}{\leadsto}$ in Appendix A.4 instead of $\xrightarrow{\overline{a}e}$, then we prove that $P =_a Q$ implies $\max_{\uparrow}(P,\,a) = \max_{\uparrow}(Q,\,a)$ by the same reasoning with the proof of Lemma 5.9 (ii). The rest is just the same as the proof of Theorem 5.10. $\square$

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 [22, 23] 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.

6.2. Related Work

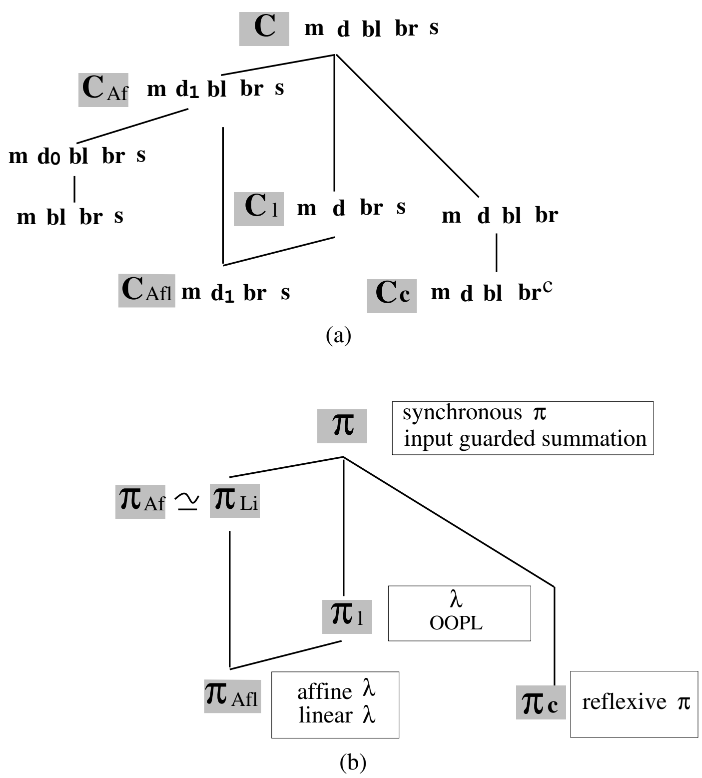
In this subsection, we summarise known results about encodings among the full π-family based on the formulation in Section 5. Then we have the following relationship.

Proposition 6.1. (i) $\mathbf{P_{cc}} \simeq^{\mathrm{e}} \mathbf{P}_{\pi} \simeq^{\mathrm{e}}_{a} \mathbf{P}_{\pi+}$ *where* $\mathbf{P}_{\pi+}$ *is the asynchronous π-calculus with input guarded summations.*

(ii) $\mathbf{P}_{\pi} \simeq^{\mathrm{e}} \mathbf{P}_{\mathrm{pol}\pi\mathrm{s}}$, $\mathbf{P}_{l} \simeq^{\mathrm{e}} \mathbf{P}_{\pi}$, and $\mathbf{P_{cc}}\backslash\mathbf{b}_l \simeq^{\mathrm{e}}_{a} \mathbf{P}_{\mathrm{pol}\pi\mathrm{s}+}$ *where* $\mathbf{P}_{\mathrm{pol}\pi\mathrm{s}}$ *is polyadic synchronous π-calculus without match or summation and* $\mathbf{P}_{\mathrm{pol}\pi\mathrm{s}+}$ *is* $\mathbf{P}_{\mathrm{pol}\pi\mathrm{s}}$ *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* $\mathbf{P}_1 \lesssim^{\mathrm{e}}_{\not\approx} \mathbf{P}_2$.

PROOF. (i) is by Theorem 2.6, Proposition 5.4 and Nestmann and Pierce [37], respectively. The first equation of (ii) is by Honda and Tokoro [19]/Boudol [7] and the second one is by Boreale [5]. We can also extend the encoding in [37] to an



In (b), boxed names indicate embeddable calculi via (congruent) adequate encodings.

Figure 1. Concurrent Combinators and the Asynchronous π-Calculus

adequate encoding up to $\approx_a$ from $\mathbf{P}_{\mathrm{pol}\pi\mathrm{s}+}$ into $\mathbf{P}_\pi$ straightforwardly. Hence we have the third equation. (iii) is by Palamidessi [40]. $\square$

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 [40] is stronger than (iii) above because the condition (3) in Definition 5.1 is not required to prove the separation between $\mathbf{P}_1$ and $\mathbf{P}_2$. See Introduction in [38] for the detailed relationship about adequacy and full abstraction between existent encodings.

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

First, in [5], Boreale recently established an interesting result which shows power of the local asynchronous (polyadic) π-calculus: there is a standard encoding from (polyadic) π-calculus to polyadic local (asynchronous) π-calculus which is fully abstract up to the weak barbed bisimilarity. But this result does not contradict Conjecture 5.11 (3) since:

(1) It is not fully abstract up to barbed congruence (hence not up to $\approx$ either). For a counterexample, we take $ax.(xy.0 \mid xy.0) \approx ax.xy.xy.0$; see Appendix E in [55] for the detailed reasoning. Note as discussed in §3.2 in [48] and §6 in [21], 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 $\mathbf{P}_l \simeq^{e} \mathbf{P}_\pi$ is only adequately related).

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

Related with (1), in the long version of [5], 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 [48]. The basic idea of this kind technique is related with the study of types of mobile processes, cf. [54]; 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 [5] from the local π-calculus to a subsystem of the local π-calculus where all objects of messages are distinct and bound by name hiding[16] is *fully abstract* up to the asynchronous weak barbed congruence [29]. This encoding uses *the link agent* in [49] to translate messages with free object names, so it is *not* message-preserving. But again this does not contradict Conjecture 5.11 (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 [49, 24], but also it may become another key agent to understand a difference between non-local and local behaviours.

6.2.2. *Expressiveness based on Combinators.* The framework for measuring expressive power which is most closely related with our idea, *generation* and *essentiality*, was formalised by Parrow in [41] 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

[16] This variant is not included in the syntax of πI-calculus [49], but it is considered as an asynchronous version of πI-calculus by changing $\overline{a}(x).P$ in πI-calculus by $(\boldsymbol{\nu}\, x)(\overline{a}x \mid P)$ where x does not occur as free objects in P.

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 [15] on the nameless processes from the more general viewpoint.

Parrow also 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 [42]. 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* $(\boldsymbol{\nu}\,\tilde{c})((!)T_1 \,|\, (!)T_2 \,|\, \cdots \,|\, (!)T_n)$ and two messages up to the strong bisimilarity. Interestingly we can observe that all our 5 combinators are also 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 satisfies a condition weaker than $P \approx [\![P]\!]$ (i.e. like standard encodings) as shown in Section 5 in our paper.

Raja and Shyamasundar also studied Quine combinators for the asynchronous π-calculus [46]. 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 [46] 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 [5] 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 5.11 (1). This would reveal that the asynchronous π-calculus may be considered as a "basic π-calculus" containing sufficient synchronisation 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 [31] works in π_{Af}-calculus although it includes infinite behaviour like $!ax.\overline{b}x$, cf. footnote 9 in §4.2. 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 2.3, we use the "!" operator for generation (i-2-e). But by the result in [23], 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" [33], or "processes for connection" [15, 16]. 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 [12] and Lafont [26] independently found the systems of combinators of *untyped* interaction nets, and Lafont [26] 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 [27] 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 [53] 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 [33, 16], and for a match/summation-less Fusion calculus [43] (see [25]). 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, 21, 44, 39]. A systematic inquiry about the separation results on such operators would increase theoretical understanding on computation in the family of π-calculi.

Acknowledgements: The question on the minimality of concurrent combinators was independently posed by Benjamin Pierce, Davide Sangiorgi and Vasco Vasconcelos. The author deeply thanks Samson Abramsky, Michele Boreale, Gerard Boudol, Simon Gay, Matthew Hennessy, Kohei Honda, Massimo Merro, Uwe Nestmann, Raja Natarajan, Catuscia Palamidessi, John Power, and Ian Stark for discussions and comments. Sincere thanks also go to anonymous referees for significant comments to revise this paper.

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

[6] Boreale, M. and Sangiorgi, D., A Fully Abstract Semantics for Causality in the π-calculus. Acta Informatica, 35, 1998.

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

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

[9] Fournet, C. and Gonthier, G, A Hierarchy of Equivalences for Asynchronous Calculi, *ICALP'98*, LNCS 1443, Springer-Verlag, 1998.

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

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

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

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

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

[15] Honda, K., Elementary Structures for Process Theory (1): Sets with Renaming, to appear *MSCS*, Available from `http://www.dcs.qmw.ac.uk/~kohei`.

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

[17] Hyland, M. and Ong, L., Pi-calculus, dialogue games and PCF, *FPCA'93*, ACM, 1995.

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

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

[20] 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.

[21] 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.

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

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

[24] Honda, K. and Yoshida, N., Game-Theoretic Analysis of Call-by-Value Computation, *Proc. ICALP'97*, LNCS 1256, pp.225–236, Springer-Verlag, July, 1997. The full version appeared in *TCS* 221 (1999), pp.393–456, Elsevier, 1999.

[25] Full version of [22]. To appear as a technical report of Queen Mary and Westfield College, 1999.

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

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

[28] Merro, M., On the expressiveness of Fusion, Chi and Pi, Proc. of *Express'98*, Vol 16, No. 2, pp.3–14, ENTCS, Elsevier Science Publishers, 1998.

[29] Merro, M. and Sangiorgi, D., On asynchrony in name-passing calculi, *ICALP'98*, LNCS 1443, pp.856–867, Springer-Verlag, 1998.

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

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

[32] Milner, R., Polyadic π-Calculus: a tutorial. *Logic and Algebra of Specification*, Springer-Verlag, 1992.
[33] Milner, R., *Action structures and the π-calculus*, Proc. of Advanced Study Institute on Proof and Computation, 60pp, Marktoberdorf, 1993.
[34] Milner, R., Calculi for Interaction. Acta Informatica, December, 1996.
[35] Milner, R., Parrow, J.G. and Walker, D.J., A Calculus of Mobile Processes, *Information and Computation* 100(1), pp.1–77, 1992.
[36] Mitchell, J., Type Systems for Programming Languages. *Handbook of Theoretical Computer Science* B, pp.367–458, MIT Press, 1990.
[37] Nestmann, U. and Pierce, B., Decoding choice encodings, *Proc. CONCUR'96*, LNCS 1119, pp.179–194, Springer-Verlag, 1996.
[38] Nestmann, U., What is a 'Good' Encoding of Guarded Choice? Proc. of *Express'97*, Vol 7, ENTCS, Elsevier Science Publishers, 1997.
[39] Odersky, M., Polarized Name Passing. *FST/TCS'15*, LNCS, Springer-Verlag, 1995.
[40] Palamidessi, C., Comparing the Expressive Power of the Synchronous and the Asynchronous π-calculus, *POPL'97*, pp. 256–265, ACM press, 1997.
[41] 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`.
[42] Parrow, J., Trios in Concert. Festschrift in honour of Robin Milner, MIT Press, 1998. Available from `http://www.sics.se/~joachim/trios.ps.Z`.
[43] Parrow, J. and Victor, B., The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes. *LICS'98*, IEEE, 1998.
[44] 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.
[45] Pierce, B. and Turner, D., Pict: A Programming Language Based on the Pi-calculus, Indiana University, CSCI Technical Report, 476, March, 1997.
[46] Raja, N. and Shyamasundar, R.K., Combinatory Formulations of Concurrent Languages, *TOPLAS*, Vol. 19, No. 6, pp.899-915, ACM Press, 1997.
[47] Riecke, J., The Analysis of Programming Structure, ACM SIGACT News, 28(2), pages 24-31, 1997.
[48] Sangiorgi, D., *Expressing Mobility in Process Algebras: First Order and Higher Order Paradigms.* Ph.D. Thesis, University of Edinburgh, 1992.
[49] Sangiorgi, D., π-calculus, internal mobility, and agent-passing calculi. *TCS*, 167(2):235–274, 1996.
[50] Turner, D.A., A New Implementation Technique for Applicative Languages., Software-Practice and Experience 9, pp.31–49, 1979.
[51] Vasconcelos, V., Typed Concurrent Objects, *ECOOP'94*, LNCS 814, pp.100–117, Springer, 1994.
[52] Walker, D., Objects in the π-calculus. *Information and Computation*, Vol. 116, pp.253–271, 1995.
[53] Yoshida, N., Graph Notation for Concurrent Combinators, *TPPP'94*, LNCS 907, pp.393–412, Springer-Verlag, 1995.
[54] 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.
[55] Yoshida, N., Minimality and Separation Results on Asynchronous Mobile Processes, CS Technical Paper, Sussex University, 98-05, September 1998. Available from `http://www.cogs.susx.ac.uk/users/nobuko/index.html`.

Appendix A. Reduction, Synchronous and Asynchronous Transition Relations

A.1. Reduction

We consider terms modulo the structural congruence. $\equiv$ is the smallest congruence relation over π-terms generated by the following rules.

(i) $P \equiv Q \;\text{ if }\; P \equiv_\alpha Q$

(ii) $P \mid Q \equiv Q \mid P \quad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \quad P \mid \mathbf{0} \equiv P \quad !P \equiv P \mid !P$
(iii) $(\boldsymbol{\nu}\, aa)P \equiv (\boldsymbol{\nu}\, a)P \quad (\boldsymbol{\nu}\, ab)P \equiv (\boldsymbol{\nu}\, ba)P \quad (\boldsymbol{\nu}\, a)\mathbf{0} \equiv \mathbf{0} \quad (\boldsymbol{\nu}\, a)P \mid Q \equiv (\boldsymbol{\nu}\, a)(P \mid Q)$ if $a \notin \mathsf{fn}(Q)$

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

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

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.

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

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

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

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

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 $\xrightarrow{l}_a$.

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

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

For the proof for the sound equalities, we often use the following (value-less) transition.

Definition A.4. *The asynchronous generic transition relation*, denoted by $\overset{\uparrow a}{\leadsto}$, is the smallest relation inferred by the following (out_g, res_g) and other rules in A.3 replacing $\overset{l}{\longrightarrow}$ with $\overset{l}{\longrightarrow}_a$.

$$(\text{out}_g) \quad \overline{a}b \overset{\uparrow a}{\leadsto} \mathbf{0} \qquad\qquad (\text{res}_g) \quad \frac{P \overset{\uparrow a}{\leadsto} P'}{(\boldsymbol{\nu}\, b)P \overset{\uparrow a}{\leadsto} (\boldsymbol{\nu}\, b)P'} \quad (a \neq b)$$

A.3. The Labelled Transition for π_c-calculus

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

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

Appendix B. Proof for Proposition 2.9

First we extend the definition of *pointedness* defined in [22] 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) $\forall \mathbf{c}(a^{+}\tilde{v}).\ (P \,|\, \mathbf{c}(a^{+}\tilde{v})) \longrightarrow Q_1 \ \wedge\ (P \,|\, \mathbf{c}(a^{+}\tilde{v})) \longrightarrow Q_2 \quad \Rightarrow \quad Q_1 \equiv Q_2$.
(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 [22].

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

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

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* $P \overset{l}{\longrightarrow} Q_2$ *with* $Q_1 \not\equiv Q_2$. *Then there exists* Q' *s.t.* $Q_1 \overset{l}{\longrightarrow} Q'$ *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 $P \overset{l}{\longrightarrow} P'$ *then, for some* Q', $Q \overset{\hat{l}}{\Longrightarrow} Q'$ *with* $P' \twoheadrightarrow_\beta \mathcal{R}_{i-1} \equiv Q'$

whenever $Q \overset{l}{\longrightarrow} Q'$ *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 [22]. For (ii), take $\mathcal{R} = \{\langle P, Q\rangle \mid P \longrightarrow_\beta Q\} \cup \equiv$ and show it is a bisimulation. (iii) is by exploiting $\mathcal{R}' \stackrel{\text{def}}{=} \cup_{i\geq 0}(\twoheadrightarrow_\beta \mathcal{R}_i \equiv)$ 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 $Q_0 \stackrel{\hat{l}}{\Longrightarrow} Q_0'$ with $P_0' \twoheadrightarrow_\beta \mathcal{R}_{i-1} \equiv Q_0'$. Since $Q \stackrel{\hat{l}}{\Longrightarrow} Q' \equiv Q_0'$, 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 $P_0 \stackrel{\hat{l}}{\Longrightarrow} P_0'$ with $P_0' \mathcal{R}_{i-1} Q_0'$. Since $P \twoheadrightarrow_\beta P_0 \stackrel{\hat{l}}{\Longrightarrow} P_0'$ implies $P \stackrel{\hat{l}}{\Longrightarrow} P' \equiv P_0'$, we have $P' \twoheadrightarrow_\beta \mathcal{R}_{i-1} \equiv Q'$, as required. □

Now we are ready to prove Proposition 2.9 (i).

We only have to consider the input transition 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 $ax.(P_1 \mid P_2) \xrightarrow{ab} (P_1\{b/x\} \mid P_2\{b/x\})$. Then

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

Since $\overline{c_i}b \mid c_ix.P_i \xrightarrow{\tau} P_i\{b/x\}$, by inductive hypothesis, $\mathbf{m}(c_ib) \mid c_i^*x.P_i \xrightarrow{\tau} P_i'$ with $P_i' \approx P_i\{b/x\}$ with $i = 1, 2$. Then $R \longrightarrow_\beta^2 (\boldsymbol{\nu}\, c_1c_2)(P_1' \mid P_2') \equiv (P_1' \mid P_2') \stackrel{\text{def}}{=} R'$. By Proposition 2.1 (i) and Proposition B.2 (iii), we have $R' \,\mathcal{R}\, (P_1\{b/x\} \mid P_2\{b/x\})$. Hence $a^*x.(P_1 \mid P_2) \approx ax.(P_1 \mid P_2)$, as required. For the rule (iv), we first define $R \stackrel{\text{def}}{=} c^*x.(P \mid \mathbf{m}(cx)) \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, cc_1c_2)(\mathbf{d}(cc_1c_2) \mid c_1^*x.P \mid \mathbf{fw}(c_2c))$. Then we have: $R \approx cx.(P \mid \mathbf{m}(cx))$ by induction on P and Proposition B.2 (iii). Next we can check: $(\boldsymbol{\nu}\, c)(!R \mid \mathbf{m}(cv)) \approx (\boldsymbol{\nu}\, c)(!cx.(\mathbf{m}(cx) \mid cx.P) \mid \mathbf{m}(cv)) \approx !P\{v/x\}$ by Propositions 2.1 (i) and B.2 (iii), respectively. Hence by Proposition 2.1 (i) again, for all v, we have:

$$a^*x.!P \quad \xrightarrow{av} \quad (\boldsymbol{\nu}\, c)(\mathbf{m}(cv) \mid !R) \approx !P\{v/x\}$$

which implies $a^*x.!P \approx ax.!P$.