Minimality and Separation Results on Asynchronous Mobile Processes

— Representability Theorems by Concurrent Combinators —

(Extended Abstract)[†]

Nobuko Yoshida[⋆]

Abstract. In [18, 19], we presented a theory of *concurrent combinators* for the asynchronous monadic π-calculus without match or summation operator [7, 16]. 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 [23] and input guarded summations [26]. The present paper shows that each of the five basic combinators introduced in [18] 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 weak bisimilarity. Expressive power of several interesting subsystems of the asynchronous π-calculus is also measured by using appropriate subsets of the combinators and their variants. Finally as an application of the main result, we show there is no semantically sound encoding of the calculus into its proper subsystem under a certain condition.

1. Introduction

The calculi of mobile processes [22, 23, 25] 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 [25], various variants of this calculus have been considered in many settings: a polyadic or monadic, synchronous or asynchronous π-calculus with or without match, mismatch, and summation operators. In sequential computation, the hierarchy of computable functions has been traditionally used to measure the expressive power of programming languages based on a rigid mathematical background. This notion is, however, too function-oriented to examine the whole expressiveness realisable in π-calculi. Consider the result in [22], which showed lazy and call-by-value λ-calculi can be simulated in an operationally correct way in monadic π-calculus without match or summation operator. The two λ-calculi are in the same computability hierarchy, but their encodings in π-calculus represent quite different communication protocols: computational behaviour in π-calculus is based on much finer interactions than functional one. The question then arises as to what are general methods to measure representability for π-calculi, which would also be applicable to other concurrency formalisms and programming languages.

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

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

[†] Full version available at: `http://www.cogs.susx.ac.uk/users/nobuko/index.html`.

In the absence of match operator, one remarkable separation result on expressiveness was proved by Palamidessi [28]: "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 [16, 11, 17, 7, 3] is known as a powerful formalism to represent a wide repertoire of interactive computational structures: polyadic and synchronous communications [16, 7] and even input-guarded summations [26], are embeddable within this calculus. At the practical level, this expressiveness gives rise to a useful high-level concurrent programming language Pict [33], which is basically 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 [18, 19]. 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 [18, 19]. More concretely, we show that five atoms introduced in [18, 19], which can represent all processes in this calculus, are indeed semantically indispensable: if any one of combinators is missing, we can no longer express the whole calculus up to weak bisimilarity.[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, 37], 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. Sections 3 and 4. Another technical interest would be the introduction of a simple way of measuring expressive power, *generation* and *minimality*, which does not depend on the notion of encodings.[2] In spite of its simplicity, we show that the minimality result is applicable to the establishment of several negative results on (the encodings into) proper subsystems of this calculus, cf. Sections 4 and 5. We believe 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 [16, 18, 6, 28, 26, 21] and further issues. Due to space limitation, we leave all of the detailed proofs to the full version [41].

---

[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 [29].

2. Generation Theorem

2.1. **The Asynchronous π-calculus.** The formalism presented in the following is a small fragment of the original π-calculus [22, 25] based on the notion of asynchronous communication [7, 16]. It is a succinct yet powerful calculus for concurrent computation, which can soundly embed various languages and calculi. We call this calculus *the asynchronous π-calculus*, or often simply *π-calculus* if there is no confusion. Let $\mathbf{N}$ be a countable set of *names*, ranged over by $a, b, c, .., x, y, z, v, w, ...$ The syntax of the asynchronous π-calculus is given as follows (writing $ax.P$ for $a(x).P$ in [25]):

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

We write $\mathbf{P}_\pi$ for the set of terms. The free and bound names in P are standard and denoted by $\mathsf{fn}(P)$ and $\mathsf{bn}(P)$. W.l.o.g. we assume all bound names in P are distinct and disjoint from free names. Name "a" in $\overline{a}v$ and $ax.P$ is called an *output* and *input subject*, respectively. The structural congruence $\equiv$ [5, 22] and the reduction relation $\longrightarrow$ and $\twoheadrightarrow$ ($\overset{\text{def}}{=} \longrightarrow^* \cup \equiv$) again follow the standard definitions [25, 16, 11] (see Appendix A in [41]). The following notations concerning name usage in terms are important.

- $\mathsf{Sub}_\downarrow(P)$ and $\mathsf{Sub}_\uparrow(P)$ are the sets of the *free input/ output subjects* of P, respectively. E.g. $\mathsf{Sub}_\downarrow(ax.bx.xy.\overline{c}e) = \{a, b\}$ and $\mathsf{Sub}_\uparrow(ax.bx.xy.\overline{c}e) = \{c\}$.
- The sets of *output/ input active names* are given by: $a \in \mathcal{AN}_\uparrow(P)$ iff $P \equiv (\boldsymbol{\nu}\, \tilde{c})(\overline{a}v \,|\, R)$ with $a \notin \{\tilde{c}\}$ and $a \in \mathcal{AN}_\downarrow(P)$ iff $P \equiv (\boldsymbol{\nu}\, \tilde{c})(ax.Q \,|\, R)$ with $a \notin \{\tilde{c}\}$.
- The *convergence predicate* is defined by: $P \Downarrow_{a^\uparrow}$ (resp. $P \Downarrow_{a^\downarrow}$) iff $\exists P'.\ P \twoheadrightarrow P' \wedge a \in \mathcal{AN}_\uparrow(P')$ (resp. $a \in \mathcal{AN}_\downarrow(P')$). $P \Downarrow_{a^\updownarrow}$ 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: $\sharp\langle \mathbf{0},\ a\rangle = \sharp\langle (\boldsymbol{\nu}\, a)P,\ a\rangle = \sharp\langle ba.P,\ a\rangle = 0$, $\sharp\langle \overline{a}b,\ a\rangle = \sharp\langle \overline{b}a,\ a\rangle = 1$, $\sharp\langle \overline{a}a,\ a\rangle = 2$, $\sharp\langle !P,\ a\rangle = \omega$ if $a \in \mathsf{fn}(P)$ else $\sharp\langle !P,\ a\rangle = 0$, $\sharp\langle ba.P,\ b\rangle = 1 + \sharp\langle P,\ b\rangle$, $\sharp\langle ca.P,\ b\rangle = \sharp\langle P,\ b\rangle$, $\sharp\langle P \,|\, Q,\ a\rangle = \sharp\langle P,\ a\rangle + \sharp\langle Q,\ a\rangle$, where we assume a, b, c are pairwise distinct.

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

Proposition 2.1. (weak bisimilarity) (i) *$\approx$ is a congruence relation* [11]*, and* (ii) *if $P \approx Q$ then $P \Downarrow_{a^\updownarrow} \ \Leftrightarrow \ Q \Downarrow_{a^\updownarrow}$.*

2.2. **Concurrent Combinators.** Concurrent combinators are tractable and powerful self-contained proper subset of π-terms, just as $\mathbf{S}$ and $\mathbf{K}$ are such for λ-calculus. Atomic agents are formed from atoms by connecting "ports" to real "locations" (names) and their computation is based on the notion of fixed dyadic interaction: two atoms interact via a common interaction port to generate new nodes and a new connection topology. See [18, 19] (cf. [39]) for the full account of the introduction and motivations of this study. Here we begin with seven basic atoms which represent basic elements of communication behaviour in name passing.

$$\begin{array}{c}
\mathbf{m}(av) \overset{\text{def}}{=} \overline{a}v \qquad \mathbf{d}(abc) \overset{\text{def}}{=} ax.(\overline{b}x \,|\, \overline{c}x) \qquad \mathbf{k}(a) \overset{\text{def}}{=} ax.\mathbf{0} \qquad \mathbf{fw}(ab) \overset{\text{def}}{=} ax.\overline{b}x \\
\mathbf{b}_r(ab) \overset{\text{def}}{=} ax.\mathbf{fw}(bx) \qquad \mathbf{b}_l(ab) \overset{\text{def}}{=} ax.\mathbf{fw}(xb) \qquad \mathbf{s}(abc) \overset{\text{def}}{=} ax.\mathbf{fw}(bc)
\end{array}$$

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

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

We write $\mathbf{c}, \mathbf{c}', ...$ for the symbols for atoms. $\mathbf{m}(ab)$ (*message*) carries 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 and $\mathbf{s}$ is used for pure synchronisation without name passing, which is indeed necessary in interaction scenarios as seen in the main theorem later.

2.3. **Finite Generation.** We introduce the ideas of *generation* and *basis* (following the treatment in λ-theory, cf. Def 8.1.1 in [4]), as well as *subsystems*. Intuitively, from a programming viewpoint, if any program written in a language Y can be described by a composition of programs written in its "core language" X up to semantic equality, we say X is a basis for Y; if X is closed under evaluation (reduction relation), it can be used as a self-contained language. Then we call X a subsystem. These ideas would be generally applicable to both functional and concurrent calculi with adaptation.

Definition 2.2. (i) (generation) Let $X \subset \mathbf{P}_\pi$. The set of terms *generated by* X, notation X^+, is the least set Y such that: $X \subset Y$ and
(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 any injective renaming.

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

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

X is called a *basis* if X is a basis for $\mathbf{P}_\pi$.

(iii) (subsystem) Let $\mathbf{P} \subseteq \mathbf{P}_\pi$. Then we say $\mathbf{P}$ is a *subsystem* (of $\mathbf{P}_\pi$) if $\mathbf{P}^+ = \mathbf{P}$ and $P \in \mathbf{P} \,\wedge\, P \longrightarrow\!\!\!\!\!\rightarrow Q \Rightarrow\; Q \in \mathbf{P}$. We also say $\mathbf{P}_1$ is a *subsystem of* $\mathbf{P}_2$ if $\mathbf{P}_1$ and $\mathbf{P}_2$ are subsystems and $\mathbf{P}_1 \subseteq \mathbf{P}_2$.

(iv) (subbasis) Y_1 is a *subbasis* of Y_2, written $Y_1 \lesssim Y_2$, if $\mathbf{P}_i = \{P \mid P \approx Q \in Y_i^+\}$ is a subsystem with $i = 1, 2$ 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 X^+ generated by X includes inaction (b), and it is closed under structural rules (a), reduction contexts (c–e) and renaming operators (f) (cf. [12, 28])[3]. (ii) says that if X is a basis then any term in $\mathbf{P}_\pi$ should be behaviourally equivalent to some term generated by X. (iii) means a subsystem $\mathbf{P}$ should be self-contained w.r.t. reduction. In (iv), $Y_1 \simeq Y_2$ means two subsystems generated by Y_1 and Y_2 have the same expressive power. Note the relation $\lesssim$ is preorder and it can be defined even if $Y_i \not\subseteq Y_j$ with $i \neq j$ and Y_i itself is not a subsystem.

Remark 2.3. • Without "!" in (i) in Definition 2.2, the finite generation with at most 19 combinators is possible by the result in [19]. Here we include $!P$ because we are concerned with expressiveness in terms of communication behaviours.

- In (ii), we can use any weak equalities (asynchronous weak bisimilarity [16], barbed congruence [3], and the maximum sound theory [17]) instead of the synchronous bisimilarity to reach the main theorems of this paper; see [41] for more details.
- (iii,iv) can be generally extended to discuss the relationship among the family of π-calculi by considering the subsystems of the full synchronous polyadic π-calculus. E.g. the asynchronous polyadic π-calculus is a subsystem of monadic synchronous π-calculus [22] and the monadic synchronous π-calculus is that of polyadic π-calculus [23] etc. See Sections 5 and 6 for more discussions on expressiveness in π-family.

---

[3] Usage 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 they may not be closed under substitutions (cf. [12]).

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

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

The main theorem of this section states these 5 combinators can generate the whole set of terms up to the weak bisimilarity.[4]

Theorem 2.4. (finite generation) $\mathbf{C}$ *is a basis, equivalently* $\mathbf{P}_\pi \simeq \mathbf{C}$.

2.4. **Proof Outline of the Finite Generation Theorem.** We first introduce the set of combinators corresponding to subsection 2.2 (we assume a, b, c are pairwise distinct).

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

Let $\mathbf{P_{cc}} \stackrel{\text{def}}{=} \mathbf{C}_7^+$. Then clearly $\mathbf{P_{cc}}$ is a proper subsystem of $\mathbf{P}_\pi$ by checking the reduction rules for atomic agents. To prove Theorem 2.4, we first show $\mathbf{C}_7$ is a basis: any prefix can be decomposed following the idea in [18]. We define a prefix mapping $a^*x.P$ in Definition A.1 in Appendix (note the rule for replication (IV) is new, cf. [18]). As we expected, "$a^*x.P$" behaves as a prefix "$ax.P$", i.e. we have $a^*x.P \mid \mathbf{m}(av) \longrightarrow\approx P\{v/x\}$. We decompose all terms in $\mathbf{P}_\pi$ to $\mathbf{P_{cc}}$ with: $\llbracket \overline{a}b \rrbracket \stackrel{\text{def}}{=} \mathbf{m}(ab)$, $\llbracket ax.Q \rrbracket \stackrel{\text{def}}{=} a^*x.\llbracket Q \rrbracket$, $\llbracket P \mid Q \rrbracket \stackrel{\text{def}}{=} \llbracket P \rrbracket \mid \llbracket Q \rrbracket$, $\llbracket (\boldsymbol{\nu}\, a)P \rrbracket \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, a)\llbracket P \rrbracket$, $\llbracket !P \rrbracket \stackrel{\text{def}}{=} !\llbracket P \rrbracket$, and $\llbracket \mathbf{0} \rrbracket \stackrel{\text{def}}{=} \mathbf{0}$. Then we have:

Lemma 2.5. $P \approx \llbracket P \rrbracket$.

Immediately $\llbracket\ \rrbracket$ is a fully abstract mapping, i.e. $P \approx Q \;\Leftrightarrow\; \llbracket P \rrbracket \approx \llbracket Q \rrbracket$. Now we know $\mathbf{C}_7$ is a basis via $\llbracket\ \rrbracket$, 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 $\mathbf{k}(b) \approx (\boldsymbol{\nu}\, c)\mathbf{b}_r(bc)$. Secondly note, for each renaming, 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 any renaming σ, $\mathbf{m}(bb) \not\approx \mathbf{m}(ab)\sigma$, etc. But *substitution can be represented by forwarders up to* $\approx$: as an example, $\mathbf{m}(aa) \approx (\boldsymbol{\nu}\, c)(\mathbf{m}(ca) \mid \mathbf{fw}(ca))$.

Now 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 $\llbracket P \rrbracket \in \mathbf{P_{cc}}$. Thus by Lemma 2.5, we have $P \approx \llbracket P \rrbracket \approx Q \in \mathbf{C}^+$, hence $\mathbf{P}_\pi \lesssim \mathbf{C}$ as required. The second inclusion is by $\mathbf{P}_\pi \supseteq \mathbf{C}$.

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 there exists a program which can be described in a core-language, but not in its proper subset.

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

Note $\mathbf{fw}(ab)$ and $\mathbf{k}(a)$ can be generated by whichever of $\mathbf{d}(abc)$, $\mathbf{b}_l(ab)$, $\mathbf{b}_r(ab)$ or $\mathbf{s}(abc)$ (for $\mathbf{b}_l(ab)$ and $\mathbf{b}_r(ab)$, we also need $\mathbf{m}(ab)$). Hence we have:

Lemma 3.2. *Let* $Y \subseteq \mathbf{C}_7$ *and write* $Y\backslash\mathbf{c}$ *for* $Y\backslash\{\mathbf{c}(\tilde{v}_1), .., \mathbf{c}(\tilde{v}_n)\}$ *with* $\mathbf{c}(\tilde{v}_i) \in Y$, *i.e. all terms of the form* $\mathbf{c}(\tilde{v}_i)$ *are deleted from* Y. *Then for all* $\mathbf{c}(\tilde{v}) \in \mathbf{C}$, $(\mathbf{C}_7\backslash\mathbf{c})^+$ *is not a basis iff* $\mathbf{C}\backslash\mathbf{c}$ *is not a basis.*

---

[4] This theorem is newly formalised and proved in this paper (cf.[18]).

We omit "w.r.t. $\mathbf{C}$" when discussing essentiality and write $\mathbf{P_{cc}}\backslash\mathbf{c}$ for $(\mathbf{C}_7\backslash\mathbf{c})^+$ with $\mathbf{c}(\tilde{v}) \in \mathbf{C}$ in the following. By Lemma 3.2, to verify the essentiality of $\mathbf{c}$, we can equivalently prove $\mathbf{P_{cc}}\backslash\mathbf{c} \underset{\not\approx}{<} \mathbf{C}$. More concretely, we show that, if we assume there is $P \in \mathbf{P_{cc}}\backslash\mathbf{c}$. s.t. $P \approx \mathbf{c}(\tilde{v})$, then this gives a contradiction. Notice $\mathbf{P_{cc}}\backslash\mathbf{c}$ is always a subsystem, and if $\mathbf{c} \neq \mathbf{m}$, it is also closed under transition relation.

3.2. **Output, Duplication and Bindings.** It is clear that "the minimum output" is needed because for all $P \in \mathbf{P_{cc}}\backslash\mathbf{m}$, $\neg P \overset{\overline{a}b}{\Longrightarrow}$, while we have $\mathbf{m}(ab) \xrightarrow{\overline{a}b} \mathbf{0}$.

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

$\mathbf{d}(abc)$ is the only agent who distributes the same value to two locations. Therefore without $\mathbf{d}(abc)$, we cannot realize *sharing of names* in π-calculus, as shown in the following lemma (i) (note $\sharp\langle P,\, e\rangle$ was given in 2.1, and that $P \equiv Q \;\Rightarrow\; \sharp\langle P,\, e\rangle = \sharp\langle Q,\, e\rangle$).

Lemma 3.4. (i) *For all* $P \in \mathbf{P_{cc}}\backslash\mathbf{d}$, $P \twoheadrightarrow P'$ *implies* $\sharp\langle P,\, e\rangle \geq \sharp\langle P',\, e\rangle$.

(ii) *Suppose* $P \overset{l}{\Longrightarrow}\overset{l'}{\Longrightarrow} P'$ *where* $l = \overline{b}e$ *or* $\overline{b}(e)$ *and* $l' = \overline{c}e'$ *or* $\overline{c}(e')$ *where if* $l = \overline{b}(e)$, *then* $e \neq e'$. *Then there exists* Q' *s.t.* $P \twoheadrightarrow (\boldsymbol{\nu}\,\tilde{f})(Q' \,|\, \mathbf{m}(be) \,|\, \mathbf{m}(ce'))$.

Now suppose there exists $P \in \mathbf{P_{cc}}\backslash\mathbf{d}$ s.t. $P \approx \mathbf{d}(abc)$. By Proposition 2.1 (i), we have $P \,|\, \mathbf{m}(ae) \approx \mathbf{d}(abc) \,|\, \mathbf{m}(ae)$. Assume e is fresh. Then $\mathbf{d}(abc) \,|\, \mathbf{m}(ae) \longrightarrow\xrightarrow{\overline{b}e}\xrightarrow{\overline{c}e} \mathbf{0}$, while $P \,|\, \mathbf{m}(ae) \overset{\overline{b}e}{\Longrightarrow}\overset{\overline{c}e}{\Longrightarrow}$ is impossible since, if so, by Lemma 3.4 (ii), $P \,|\, \mathbf{m}(ae) \twoheadrightarrow (\boldsymbol{\nu}\,\tilde{f})(Q' \,|\, \mathbf{m}(be) \,|\, \mathbf{m}(ce))$, but this contradicts (i) because $\sharp\langle(P \,|\, \mathbf{m}(ae)),\, e\rangle = 1$. Therefore we have:

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

Next we consider the 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 is received (x in $ax.\mathbf{fw}(xb)$), and the latter is the only one which can create a new output-subject (x in $ax.\mathbf{fw}(bx)$).

Lemma 3.6. (i) *For all* $P \in \mathbf{P_{cc}}\backslash\mathbf{b}_l$, $P \xrightarrow{l} P'$ *implies* $\mathsf{Sub}_{\downarrow}(P) \supseteq \mathsf{Sub}_{\downarrow}(P')$.

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

The proof of the following proposition is similar to that of Proposition 3.5.

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

Lemmas 3.4 and 3.6 simply explain the roles of $\mathbf{d}$, $\mathbf{b}_l$ and $\mathbf{b}_r$ through the proof of their essentiality. However $\mathbf{d}(abc)$ also represents another functionality, namely increase of parallelism. We also note the syntax of $\mathbf{b}_l(ab)$ and $\mathbf{b}_r(ab)$ cannot be reduced even if we still keep the capability to create the new input and output subject names: $\mathbf{b}_l(ab)$ cannot be replaced with $ax.xy.\mathbf{0}$ and $\mathbf{b}_r(ab)$ cannot be replaced with $ax.\overline{x}b$, see Theorem 3.14. Section 4 gives further examination of these functionalities.

3.3. **Synchronisation.** Now we prove the most interesting and difficult part: *creating some term* (a forwarder or a message, cf. Lemma 3.13) *after synchronisation, while doing no name-instantiation*, is really essential to represent the whole behaviour of π-calculus.

Before that, we introduce the notion of *needed redex pairs*[5] to formalise causality of reductions (τ-actions). Write $\mathbf{c}(a^- \tilde{v})$ for an input atom (remember $\mathbf{c}$ ranges over atoms). Assume $P \xrightarrow{\tau} P'$ is obtained by interaction between $\mathbf{c}(x^- \tilde{v})$ and $\mathbf{m}(xy)$ in P, i.e. the

---

[5] The more detailed definitions for occurrences and needed redex pairs are found in [41]. Such a needed redex pair is uniquely determined up to $\equiv_\alpha$ because $\xrightarrow{\tau}$ is based on the labelled transition relation without $\equiv$, cf. Appendix A in [41].

derivation of $P \xrightarrow{\tau} P'$ 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 (the occurrences of) $\mathbf{c}(x\tilde{v})$ and $\mathbf{m}(xy)$ in P are *needed* for (the occurrence of) $\mathbf{c}'(\tilde{w})$ in P'. For the same P and P', if $P' \xrightarrow{\tau}^{+} P''$ and $\mathbf{c}'(\tilde{w})$ remains as $\mathbf{c}'(\tilde{w}')$ in P'' without interaction, we also say the same pair of occurrences are needed for (the occurrence of) $\mathbf{c}'(\tilde{w}')$ in P''. We now extend this notion to a sequence of τ-actions: (the occurrences of) $\mathbf{c}(x\tilde{v})$ and $\mathbf{m}(xy)$ in P are needed for (the occurrence of) $\mathbf{c}'(\tilde{w})$ in P' s.t. $P \xrightarrow{\tau}^{+} P'$ if there is a chain of needed redex pairs from $\mathbf{c}(x\tilde{v})$ and $\mathbf{m}(xy)$ to $\mathbf{c}'(\tilde{w})$. For example, take the following τ-actions.

$$(\mathbf{d}(abc) \,|\, \mathbf{m}(ae) \,|\, \mathbf{b}_r(bd)) \xrightarrow{\tau} (\mathbf{m}(be) \,|\, \mathbf{m}(ce) \,|\, \mathbf{0} \,|\, \mathbf{b}_r(bd)) \xrightarrow{\tau} (\mathbf{0} \,|\, \mathbf{m}(ce) \,|\, \mathbf{0} \,|\, \mathbf{fw}(de))$$

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 $\mathbf{fw}(de)$.

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

Definition 3.8. (general synchroniser) Let $a \neq b$. A *general synchroniser at 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) $\neg(P \,|\, \mathbf{m}(ae) \,|\, \mathbf{m}(bc)) \Downarrow_{e\updownarrow}$, where e is fresh in (2) and (3).

We can easily check none of $\mathbf{m}$, $\mathbf{d}$, $\mathbf{fw}$ and $\mathbf{b}_l$ is a general synchroniser because they cannot create a new input subject after interaction with a message $\mathbf{m}(ae)$. If $\mathbf{k}(a)$ interacts with a message $\mathbf{m}(ae)$, then a value e is thrown away. Hence it satisfies (1) and (3), but does not have the property (2). $\mathbf{b}_r(ab)$ satisfies (1) and (2), but does not have the property (3) since a value is used as an output subject. On the other hand, $\mathbf{s}(abc)$ is a general synchroniser at a to b because $\mathbf{s}(abc) \xrightarrow{ae} \mathbf{fw}(bc) \xrightarrow{bf} \mathbf{m}(cf) \xrightarrow{\overline{c}f} \mathbf{0}$ is its only possible transition.

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 weak bisimilarity?* The following lemma answers this question.

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

Outline of Proof: In the following, we present only the basic ideas of the proof. The full proof can be found in [41]. Suppose $P \in \mathbf{P_{cc}}\backslash\mathbf{s}$ is general synchroniser at a to b. Since $b \notin \mathcal{AN}_{\downarrow}(P')$ for all P' s.t. $P \longrightarrow\!\!\!\!\!\rightarrow P'$, there are only two ways for b to be created as a new active input subject:

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

Notice either $\mathbf{b}_r(db)$ or $\mathbf{b}_l(df)$ should be a subterm of P since no combinator can generate $\mathbf{b}_r$ or $\mathbf{b}_l$. Now assume $P_0 \overset{\text{def}}{=} P \,|\, \mathbf{m}(ae) \xrightarrow{\tau} P_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_{n-1} \xrightarrow{\tau} P_n$ with $b \notin \mathcal{AN}_{\downarrow}(P_i)$ $(1 \leq i \leq n-1)$ and $b \in \mathcal{AN}_{\downarrow}(P_n)$. By the above argument, we note:

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

Case **(i)** is mechanical (remember neither $\mathbf{b}_r(ab)$ nor $\mathbf{b}_l(ab)$ is a general synchroniser). Case **(ii)** is divided into three cases because we only have three input combinators $\mathbf{d}(abc)$, $\mathbf{b}_r(ab)$ and $\mathbf{b}_l(ab)$. Suppose $\mathbf{c}(a^{-}\tilde{v}) \equiv \mathbf{d}(aa_{21}a_{22})$ in P_i for some a_{21}, a_{22}. Then by the definition of needed redexes, either $\mathbf{m}(a_{21}e)$ or $\mathbf{m}(a_{22}e)$ is needed again, and there exists a needed $\mathbf{c}_2(a_{2j}\tilde{v})$ in P_l $(i < l \leq n-1)$ which should interact with $\mathbf{m}(a_{2j}e)$ $(j = 1, 2)$. If again $\mathbf{c}_2(a_{2j}\tilde{v})$ is $\mathbf{d}(a_{2j}a_{31}a_{32})$, then again $\mathbf{m}(a_{31}e)$ or $\mathbf{m}(a_{32}e)$ is needed. Because the chain of needed redexes is always finite, either $\mathbf{b}_l(a_{kj}g)$ or $\mathbf{b}_r(a_{kj}g)$ is needed for some k with $i < k \leq n-1$ since $\mathbf{m}(a_{kj}e)$ is forwarded just throughout a chain of $\mathbf{d}$ without changing the value e. So this case finally amounts to the remaining two cases $\mathbf{c}(a^-\tilde{v}) \equiv \mathbf{b}_r(ag)$ or $\mathbf{c}(a^-\tilde{v}) \equiv \mathbf{b}_l(ag)$, which are in turn similar to Case **(i)**. $\square$

As seen in the above proof outline, one of the merits of using combinators is that we only have to consider a few simple cases to prove this kind of negative results (i.e. impossibility of generation of general synchronisation).

Now we can show $\mathbf{s}(abc)$ cannot be represented by other atoms. First we note that if P is a general synchroniser and $P \approx Q$, then Q is a general synchroniser by Proposition 2.1. Thus if $\mathbf{s}(abc) \approx Q$, then Q should be a general synchroniser again. But such Q cannot be generated without $\mathbf{s}$ by the main lemma. Hence we have:

Proposition 3.10. *$\mathbf{s}(abc)$ is essential.*

This result says that the prefix "$ax.P$" in π-calculus plays the role not only of binding x in P but also of *synchronising at a (and then activating P).* See the next section for the study of a calculus with even less synchronisation. Now by Propositions 3.3, 3.5, 3.7 and 3.10, we reach the main theorem.

Theorem 3.11. (**Minimality**) $\mathbf{C}$ *is a minimal basis. Hence* $\mathbf{C}\backslash\mathbf{c} \lnapprox \mathbf{C}$ *for each* $\mathbf{c} \in \mathbf{C}$.

Because each combinator cannot be generated by other 4 combinators by the main theorem, 5 combinators cannot be generated by 4, 4 can not be by 3, etc. Hence we have the following hierarchy of expressiveness based on 5 combinators.

Corollary 3.12. *$Y_1 \subsetneq Y_2 \subseteq \mathbf{C}$ 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. $\lnapprox$.*

3.4. **Strong Minimality.** In programming languages, a user sometimes wishes to replace an existent primitive with another new primitive defined by him/herself without loss of expressive power. If a basis is minimal, we can automatically check the essentiality of a new primitive.

Lemma 3.13. *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.*

As an example, $\mathbf{s}(abc)$ can be replaced with a *message synchroniser* $\mathbf{s}_m(abc) \stackrel{\text{def}}{=} ax.\overline{b}c$ because $\mathbf{s}(abc) \approx (\boldsymbol{\nu}\, e)(\mathbf{s}_m(aeb) \mid \mathbf{b}_l(ec))$ by the above lemma. On the other hand, though $ax.by.\mathbf{0} \approx (\boldsymbol{\nu}\, c)\mathbf{s}(abc)$ is a synchroniser of Definition 3.8, we can *not* replace $\mathbf{s}(abc)$ with this agent: if we diminish any atom in $\mathbf{C}$ by hiding, it is no longer a basis.

Theorem 3.14. (the strong minimality) *Let $Y \stackrel{\text{def}}{=} (\mathbf{C}\backslash\mathbf{c}) \cup_{1 \leq i \leq n} \{P_i\}$ with $\mathbf{c}(\tilde{a}) \in \mathbf{C}$, $P_i \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, \tilde{b}_i)\mathbf{c}(\tilde{a})$ and $\emptyset \neq \{\tilde{b}_i\} \subseteq \{\tilde{a}\}$ for some $\tilde{b}_i$ and n. Then Y is not a basis.*

To prove this, we use Lemma 3.13 (See [41]). Thus the five atoms are not only essential but also have indeed "atomic" properties in that we cannot reduce its syntax further. We use these theorems in the following two sections.

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 *local π-calculus*

[15, 6, 2, 21] in which no value is instantiated with input-subjects. Next we examine *sharing of names* by studying *linear* and *affine π-calculi* where the number of free names is not changed or decreased during communications. Finally we consider *synchronisation* by formulating *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 calculus by the results in Section 3. The proof method shows how we can use combinators as a tractable and informative tool to analyse the concurrent communication protocols. We begin with the formulation of *separation.*

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

By the main theorem and Lemma 3.2, for each $\mathbf{c} \in \mathbf{C}$, $\mathbf{P}_{\pi\backslash\mathbf{c}} = \{P \mid P \approx Q \in (\mathbf{C}\backslash\mathbf{c})^+\}$ is a proper subsystem of $\mathbf{P}_\pi$.

4.2. **Local π-calculus.** The asynchronous π-calculus was originally considered as a simple formal system for concurrent object-based computation with asynchronous communication [16, 15], 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.[15]. 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.

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

We call this calculus *local π-calculus* (written π_l for short) and write $\mathbf{P}_l$ for the set of terms.[6] One important remark is local polyadic name passing, branching structures [16, 18], the weak call-by-value λ-calculus [41], and typical concurrent objects [16, 38, 2] can be encoded in π_l-calculus. But what is the difference between local and non-local? For the answer, we only have to check whether any $P \in \mathbf{P}_l$ can be decomposed into $\mathbf{P}_{\mathbf{cc}} \backslash \mathbf{b}_l$ by the rules in Definition A.1 except (viii), (xi), (xii) and (xiii).

Proposition 4.2. *$\mathbf{C}\backslash\mathbf{b}_l$ is a minimal basis of π_l-calculus, $\mathbf{P}_{\pi\backslash\mathbf{b}_l} \simeq \mathbf{P}_l \simeq \mathbf{C}\backslash\mathbf{b}_l \lnsim \mathbf{C}$.*

The minimality and separation are given by Theorem 3.11. Notice that the above result not only proves minimality but also shows that $\mathbf{C}_7 \backslash \mathbf{b}_l$ is a system of combinators for π_l-calculus: there is fully abstract correspondence between them. This and other observations indicate that the local π-world forms a self-contained universe, so that π_l-calculus may be worth being studied as an independent calculus like λI-calculus [4].

4.3. **Linear and Affine π-calculi.** The asynchronous π-calculus has two elements to increase non-determinism during communication – *sharing of names* and *parallelism* as found in $ax.(P \mid Q) \mid \overline{a}v \longrightarrow P\{v/x\} \mid Q\{v/x\}$. Such elements are represented by $\mathbf{d}(abc)$ in a concise way. To closely look at two elements separately, we consider two subsystems by restricting the syntax of the prefix:

---

[6] Such a subcalculus was already discussed and studied independently in [15, 16, 6, 2, 21] (cf. [9]).

(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_{Li} and π_{Af} for short) and we denote $\mathbf{P}_{\mathrm{Li}}$ and $\mathbf{P}_{\mathrm{Af}}$ for the sets of terms of π_{Li} and π_{Af}-calculi, respectively. Note $\mathbf{P}_{\mathrm{Li}} \subsetneq \mathbf{P}_{\mathrm{Af}} \subsetneq \mathbf{P}_{\pi}$.[7] Then a natural question is what expressiveness relation lies between with/without parallelism and/or sharing. In particular, is there any difference between linear and affine name-passing? Since $\mathbf{d}(abc)$ cannot be used directly to represent non-sharing communication, we here introduce a simple new combinator, called *1-distributer*, $\mathbf{d}_1(abc) \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu}\, d)ax.(\overline{b}x \,|\, \overline{c}d)$, and consider a new set of combinators $\mathbf{C}_{\mathrm{Af}} \stackrel{\mathrm{def}}{=} \mathbf{C}\setminus\mathbf{d} \cup \{\mathbf{d}_1(abc)\}$ with a, b, c pairwise distinct. $\mathbf{d}_1$ distributes two messages while forwarding only one value, hence this has the same parallelism as $\mathbf{d}$, but not sharing. The following proposition is proved by decomposing prefixes of these calculi into $\mathbf{C}_{\mathrm{Af}}$ 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 A.1. See [41] for the details.

Proposition 4.3. (i) $\mathbf{C}\setminus\mathbf{d} \lnapprox \mathbf{C}_{\mathrm{Af}} \lnapprox \mathbf{C}$. (ii) $\mathbf{C}_{\mathrm{Af}}$ *is a minimal basis of* π_{Li} *and* π_{Af}*-calculi, hence we have* $\mathbf{P}_{\pi\setminus\mathbf{d}} \lnapprox \mathbf{P}_{\mathrm{Li}} \simeq \mathbf{P}_{\mathrm{Af}} \lnapprox \mathbf{P}_{\pi}$.

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 $\mathbf{P}_{\mathbf{cc}}\setminus\mathbf{d}$ and cannot generate $\mathbf{d}_1$. Note also another proper subsystem, affine local π-calculus whose minimal basis is $\mathbf{C}_{\mathrm{A}fl} \stackrel{\mathrm{def}}{=} \{\mathbf{m}(ab), \mathbf{d}_1(abc), \mathbf{b}_r(ab), \mathbf{s}(abc)\}$ with a, b, c pairwise distinct, has an enough power (without replication) to embed full *linear* and *affine* λ-calculi [10, 1] where substitution of a term occurs only once or at most once (their embedding is given based on [22] without replication: see [41]). Deeper analysis related to linear typing systems on this local π-family, e.g.[40, 2], would be an interesting research topic.

4.4. **Commutative π-calculus.** The asynchronous π-calculus was born by deleting output synchronisation from the original synchronous π-calculus [22]. But what calculus is obtained if we further delete input synchronisation from this calculus? This subsection studies synchronisation by introducing a more asynchronous calculus separated by a synchroniser. This calculus, which is called *commutative π-calculus* (π_c for short), allows commutation of prefixes if there is no ordering by binding. We define π_c-calculus following the ideas in [8] and [24]. 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 4.4. (commutative π-calculus) We use the same syntax as in 2.1 for the syntax of π_c-calculus and add the following two laws as new structural rules.

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

$\longrightarrow$ is defined modulo $\equiv$ in the same way as in [22] (cf. [41]).

The first structural rule is found in [8], while the second one comes from [24]. Notice that in any strong/weak asynchronous/synchronous semantics, we have $ax.by.P \not\approx by.ax.P$. An example of reduction of π_c-calculus (with $x \neq a$ and $b \neq y$) is:

$$\overline{a}v \,|\, bx.ay.\overline{a}w \;\equiv\; \overline{a}v \,|\, ax.by.\overline{a}w \;\equiv\; \overline{a}v \,|\, ax.by.\mathbf{0} \,|\, \overline{a}w \;\longrightarrow\; \overline{a}v \,|\, by.\mathbf{0}$$

It seems impossible to construct a general synchroniser satisfying Definition 3.8 since we have $ax.by.\overline{c}y \;\equiv\; ax.\mathbf{0} \,|\, by.\overline{c}y$. But how can we prove this? First we observe that

---

[7] π_{Li} 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.

to represent π_c-calculus, $\mathbf{b}_r(ab)$ 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* defined by:

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

Note $\mathbf{b}_r^c(ab) \approx \mathbf{b}_r^c(ba) \in \mathbf{P_{cc}} \backslash \mathbf{s}$. Now set $\mathbf{C}_c \overset{\text{def}}{=} \{\mathbf{m}(ab), \mathbf{d}(abc), \mathbf{b}_r^c(ab), \mathbf{b}_l(ab)\}$ with a, b, c pairwise distinct. Then we can show $\mathbf{C}_c$ is a set of combinators of π_c, just as $\mathbf{C}$ is for the asynchronous π. This is proved by *commutative prefix mapping* $a^\star x.P$ in $\mathbf{P}_c$ given in Definition A.2 in Appendix A. Now we get:

Proposition 4.5. (i) *π_c-calculus has no general synchroniser which satisfies Definition 3.8.* (ii) $\mathbf{P_{cc}} \backslash \{\mathbf{b}_r, \mathbf{s}\} \lnsim \mathbf{C}_c \lnsim \mathbf{P_{cc}} \backslash \mathbf{s}$.

To prove, we use Lemmas 3.9 and 3.6 (ii). See [41] for the details. The behaviour of π_c-calculus is exactly simulated in the asynchronous π-calculus without $\mathbf{s}$ (i.e. the mapping based on $a^\star x.P$ is fully abstract). As an important remark, the monadic reflexive π-calculus studied in the framework of action calculi [24] is faithfully simulated by π_c-calculus up to the maximum sound theory [17] following the idea in Example 6.4 in [13]. Hence it has less synchronisation than π_c-calculus. See [41] for further discussion.

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. $[\![P]\!] \approx [\![Q]\!] \;\Rightarrow\; P \approx Q$) [18, 16, 26, 28, 6]. One of the most intriguing questions related to our present study in this context is: *if we miss any one of 5 combinators, i.e. in any proper subsystem of* $\mathbf{C}$*, is it absolutely impossible to construct any "good" encoding of* $\mathbf{P}_\pi$*?* This section shows the minimality theorem is applicable to derive several non-existence results of encodings. First we introduce a new formulation of measuring expressive power based on encodings, extending our view to the whole π-family. Hereafter "subsystems" etc. denote those of the full polyadic synchronous π-calculus $\mathbf{P}_{\text{full}}$ (with match and mixed summation operators [23]), defined as in Definition 2.2 (iii).

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

(1) (uniform [28]) $[\![\;]\!]$ is homomorphic, i.e. $[\![P \mid Q]\!] \overset{\text{def}}{=} [\![P]\!] \mid [\![Q]\!]$, $[\![(\boldsymbol{\nu}\, c)P]\!] \overset{\text{def}}{=} (\boldsymbol{\nu}\, c)[\![P]\!]$, $[\![!P]\!] \overset{\text{def}}{=} ![\![P]\!]$ and $[\![P\sigma]\!] \overset{\text{def}}{=} [\![P]\!]\sigma$ with σ an injective renaming, and $[\![\mathbf{0}]\!] \overset{\text{def}}{=} \mathbf{0}$.
(2) (reasonable [28, 36, 18]) $P \Downarrow_{a\updownarrow} \;\Leftrightarrow\; [\![P]\!] \Downarrow_{a\updownarrow}$.
(3) (reduction-closed, cf. [36, 17]) $P \longrightarrow\!\!\!\!\!\rightarrow P' \;\Rightarrow\; \exists Q.\, [\![P]\!] \longrightarrow\!\!\!\!\!\rightarrow Q \approx [\![P']\!]$, and $[\![P]\!] \longrightarrow\!\!\!\!\!\rightarrow Q \;\Rightarrow\; \exists R.\, (P \longrightarrow\!\!\!\!\!\rightarrow R \;\wedge\; Q \approx [\![R]\!])$.

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

Note that $\mathbf{P}_1 \subseteq \mathbf{P}_2$ implies $\mathbf{P}_1 \lesssim^{\text{e}} \mathbf{P}_2$ and that $\lesssim^{\text{e}}$ is a preorder. (1) and (2) nearly correspond to the uniform and reasonable conditions in [28] and (3) describes the standard operational closure properties found in most existing adequate encodings, cf. [22, 16, 26]. In (3), we may use other weak equivalences such as asynchronous weak bisimilarity [16], weak barbed bisimulation [3] and sound theories [17, 18] (as well as corresponding action predicates in (2)), according to purposes. Proposition 5.2 below is still valid when we consistently use these variations. See [41] for discussions.

Roughly speaking, $\mathbf{P}_1 \lesssim^{\mathrm{e}} \mathbf{P}_2$ means there is an adequate encoding from $\mathbf{P}_1$ into $\mathbf{P}_2$ up to reduction-based semantics [17, 3] closed under reduction-contexts, while $\mathbf{P}_1 \lesssim \mathbf{P}_2$ is closely related with the existence of a fully abstract encoding, cf. Lemma 2.5. We can easily show $\mathbf{P}_1 \lesssim \mathbf{P}_2$ implies $\mathbf{P}_1 \lesssim^{\mathrm{e}} \mathbf{P}_2$, but the converse does not hold (cf. Proposition 6.1 (ii)).[8] Thus proving a non-existence result on standard encodings is much harder than proving a non-generation result. We can apply, however, theorems and propositions in the previous sections to show the following negative result on encodings.

Proposition 5.2. *Suppose* $\mathbf{P_{cc}} \subseteq \mathbf{P}$ *is a subsystem of* $\mathbf{P}_{\mathrm{full}}$. *Then there is no standard mapping* $[\![\ ]\!] : \mathbf{P} \to \mathbf{P}'$, *if* either (1) (message preserving) $\mathbf{P}'$ *is any proper subsystem* $\mathbf{P}' \lnsim \mathbf{C}$ *studied in Sections 3 and 4, and, moreover, a map* $[\![\ ]\!]$ *satisfies* $[\![\overline{a}b]\!] \approx \overline{a}b$, (2) (non-output) $\mathbf{P}' \subseteq \mathbf{P}_{\pi\backslash\mathbf{m}}$, *or* (3) (non-parallelism) $\mathbf{P}' \subseteq \mathbf{P}_{\pi\backslash\mathbf{d}}$.

(2) is obvious. (3) is proved by analysing the number of active messages and causality of dependency on needed redexes. For (1), it is enough to show there is no encoding of $\mathbf{c}$ into $\mathbf{P}_{\pi\backslash\mathbf{c}}$ for each $\mathbf{c} \in \{\mathbf{b}_l, \mathbf{b}_r, \mathbf{s}\}$, and no encoding of $\mathbf{d}(abc)$ in $\mathbf{P}_{\mathrm{Af}}$. We use lemmas, propositions and theorems in Sections 3 and 4, as well as a general fact about renaming-closed mapping, cf. [13, 12]: *if* $[\![\ ]\!]$ *satisfies* (1) *in Definition 5.1, then* $\mathsf{fn}(P) \supset \mathsf{fn}([\![P]\!])$. See [41] for proofs.

(3) is significant in that it shows *increase of parallelism is indeed essential* to embed name-passing realisable in π-calculus. The condition $[\![\overline{a}b]\!] \approx \overline{a}b$ in (1) 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-calculi into the asynchronous π-calculus [16, 18, 19, 26]. However this condition is too strong for general adequate encodings. Relatedly we believe the following conjecture holds.

Conjecture 5.3. (1) (synchronisation) $\mathbf{P}_{\pi\backslash\mathbf{s}} \lnsim^{\mathrm{e}} \mathbf{P}_\pi$, (2) (sharing of names) $\mathbf{P}_{\mathrm{Af}} \lnsim^{\mathrm{e}} \mathbf{P}_\pi$, *and* (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 minimum, and sharing of names is indispensable, in order to construct various communication structures, e.g. polyadic name-passing. Together with (1) in Proposition 5.2, (3) would be proved by showing that if a standard encoding is not message-preserving, then it is not fully abstract up to $\approx$. This in turn would be extended to a more general statement in the context of encodings of polyadic message passing: there is no fully abstract standard encoding from polyadic into monadic name-passing[9], cf. [40].

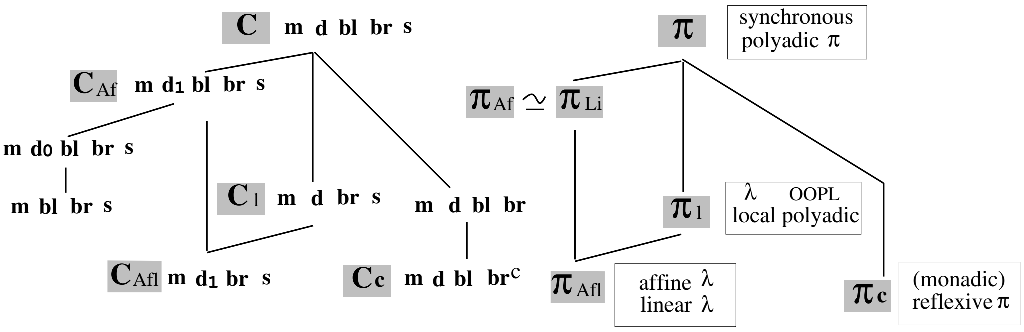
6. Discussion

6.1. **Summary of the Results.** This paper proposed a basic formal framework for representability, *generation* and *minimal basis*, and investigated that computational elements found in 5 combinators [18, 19] are essential to express the asynchronous monadic π-calculus without match or summation. Figure 1 summarises this separation result on (a) combinators and (b) subsystems of the asynchronous π-calculus, which are in one-one correspondence via a fully abstract mapping. In (b) in Figure 1, names in box depict the embeddable calculi by (congruent) adequate encodings.

---

[8] Full abstraction does not usually hold when we translate a high-level communication structure into a low level one because of difference on granularity of atomic actions, see [40].

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



(a) (b)

Figure 1. Separation for combinators and the asynchronous π-calculus.

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. We say $[\![\ ]\!]$ is *a-standard encoding* if it is defined by replacing $\Downarrow_{a\updownarrow}$ in (2) in Definition 5.1 with $\Downarrow_{a\uparrow}$ and $\approx$ in (3) in Definition 5.1 with the asynchronous bisimilarity, and write $\lesssim^{\mathrm{e}}_{a}$, $\not\lesssim^{\mathrm{e}}_{a}$ and $\simeq^{\mathrm{e}}_{a}$ for a-standard relations. Then we have the following relationship.

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

(ii) $\mathbf{P}_{\pi} \simeq^{\mathrm{e}} \mathbf{P}_{\mathrm{pol}\pi\mathrm{s}}$ *and* $\mathbf{P}_{l} \simeq^{\mathrm{e}} \mathbf{P}_{\pi}$, *hence* $\mathbf{P}_{\mathbf{cc}} \backslash \mathbf{b}_{l} \simeq^{\mathrm{e}}_{a} \mathbf{P}_{\mathrm{pol}\pi\mathrm{s}+}$ *where* $\mathbf{P}_{\mathrm{pol}\pi\mathrm{s}}$ *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* $\mathbf{P}_1 \not\lesssim^{\mathrm{e}} \mathbf{P}_2$.

(i) is proved by Theorem 2.4 and Nestmann and Pierce [26], respectively. (ii) is by Honda and Tokoro [16], Boreale [6] and (i), respectively. (iii) is by Palamidessi [28]. The result in (i) is stronger than (ii) because existing encodings are fully abstract, while in (ii), we have only adequate encodings from the right into the left. Palamidessi's result [28] is more general than (iii) because we do not need the condition (3) in Definition 5.1.

Two remarks are due for Proposition 5.2 (1). First, all encodings in (i) are message preserving. Secondly, in spite of Proposition 4.2, Boreale [6] showed that there is a "polarity-exchanging" standard encoding from (polyadic) asynchronous π-calculus into polyadic π_l-calculus, which is fully abstract up to the weak barbed bisimilarity. But this result does not contradict Conjecture 5.3 (3) since: (1) it is not fully abstract up to barbed congruence (hence not up to $\approx$ either)[10], and (2) even under the barbed bisimilarity, we do not know whether there is a fully abstract encoding from the asynchronous π-calculus into *monadic* π_l-calculus because he uses the power of polyadic name passing (hence $\mathbf{P}_l \simeq^{\mathrm{e}} \mathbf{P}_{\pi}$ is adequately related). Notice also that it is not message-preserving. See [41] for more discussion about related topics to his encodings [6, 21].

6.3. **Open Issues.** In the following, we list some of naturally arising open issues.

- As we discussed in Section 5 and the above, much still remains to be done on the study of existence or non-existence result of adequate and fully abstract encodings. For example, Boreale's result on local π-calculus [6] lets us know a possibility to construct various kinds of standard encodings. This also suggests that there is some difficulty

---

[10] For a counterexample, see [41].

to solve the negative result about encodings. Based on this observation, the most interesting but difficult open problem may be Conjecture 5.3 (1). This would reveal that the asynchronous π-calculus may be considered as a "basic π-calculus" containing sufficient power for interactive computation in a minimal tractable syntax.[11]

- 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 [22] works in π_{Af}-calculus although it includes infinite behaviour like $!ax.\overline{b}x$, cf. footnote 7. What is a minimal basis to realise universal computation power in π-calculus? Is it absolutely needed to increment the number of names during reduction and synchronise at the input prefix to represent sequential computation? Such an investigation is another important topic because it relates a basic question in functional computing to expressiveness of concurrent computing.
- In Definition 2.2, we use "!" operator for generation (i-2-e). But by the result in [19], 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" [24], or "processes for connection" [13, 14]. It may be of interest to check the essentiality of these agents to understand what computational elements are essential to express "copies" and "name restriction" in mobile processes.
- Raja and Shymasundar studied Quine combinators for the asynchronous π-calculus [34], and Parrow showed a combination of a few kinds of *trios*, which are terms in the form $\alpha.\beta.\gamma.\mathbf{0}$, can represent the synchronous polyadic π-calculus without match or summation operator up to weak bisimilarity [30]. Since the combinators in [34] are not a proper subset of π-calculus like ours, and the mapping in [30] is not given inductively (i.e. not homomorphic), the ideas of basis and generation may not be directly applicable to these systems. However to check essentiality of each combinator of [34] and each trio of [30] would also be interesting for understanding the machinery of name-passing from a different angle.
- 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 [24, 14], and for a match/summation-less Fusion calculus [31] (see [20]). Measuring expressiveness in such a calculus following the line of this paper would be possible and interesting for examination of synchronous name-passing.
- Finally match and mismatch operators are also significant from both practical and theoretical viewpoints [3, 17, 32, 27], while practical failure models [2, 35, 9] are recently studied by introducing additional operators. A systematic inquiry about the separation results on such operators would increase theoretical understanding on computation in the family of π-calculi.

---

[11] Recently the author proved an advanced negative result about encodings into a calculus which satisfies: $P_1 \overset{ab}{\Longrightarrow} P_2 \overset{cd}{\Longrightarrow} P_3 \;\Rightarrow\; \exists P_2'.\ P_1 \overset{cd}{\Longrightarrow} P_2' \overset{ab}{\Longrightarrow} P_3$ where $\{a, b\} \cap \{c, d\} = \emptyset$. We leave the proof, which uses a quite different technique from those in this paper, as well as discussions of related results, to a coming exposition.

Acknowledgement. The author thanks anonymous referees, M. Boreale, G. Boudol, M. Hennessy, K. Honda, N. Raja, C. Palamidessi, M. Merro, J. Power, I. Stark and V. Vasconcelos for comments and discussions. She was a visiting academic researcher in LFCS, University of Edinburgh during the writing of the first version of this paper, supported by JSPS Fellowship for Japanese Junior Scientists.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[17] Honda, K. and Yoshida, N., On Reduction-Based Process Semantics. *TCS*, pp.437–486, No.151, North-Holland, 1995.

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

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

[20] Full version of [18]. To appear as a technical report of Sussex University, 1998.

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

[22] Milner, R., Functions as Processes. *MSCS*, 2(2), pp.119–146, 1992.

[23] Milner, R., Polyadic π-Calculus, *Logic and Algebra of Specification*, Springer-Verlag, 1992.

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

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

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

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

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

[29] Parrow, J., The Expressive Power of Parallelism. Future Generation Computer Systems, 6:271–285, 1990.

[30] Parrow, J., Trios in Concert. Festschrift in honour of Robin Milner, MIT Press, 1998.

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

[32] 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.

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

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

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

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

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

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

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

[40] Yoshida, N., Graph Types for Monadic Mobile Processes, *FST/TCS'16*, LNCS 1180, pp. 371–386, Springer-Verlag, 1996. Full version as LFCS Technical Report, ECS-LFCS-96-350, 1996.

[41] Yoshida, N., Full version of this paper. The first version in Oct, 1997. Revised in Feb, 1998. pp.40. Available from `http://www.cogs.susx.ac.uk/users/nobuko/index.html`. To appear as a technical report of Sussex University, 1998.

Appendix A. Prefix Mappings

We assume the following annotations on polarities preserved by reduction.

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

Definition A.1. (prefix mapping) The *prefix mapping* $a^*x.P : \mathbf{N} \times \mathbf{N} \times \mathbf{P_{cc}} \to \mathbf{P_{cc}}$ is inductively defined by applying the following rules from (I) to (XIII) in this order (c, c_1, c_2 are fresh and distinct).

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

Definition A.2. (commutative prefix mapping) Let $\mathbf{P}_c \stackrel{\text{def}}{=} (\mathbf{C}_7 \backslash \{\mathbf{b}_r, \mathbf{s}\} \cup \{\mathbf{b}_r^c(ab), \mathbf{b}_r^c(aa)\})^+$. Then *commutative prefix mapping* $a^\star x.P : \mathbf{N} \times \mathbf{N} \times \mathbf{P}_c \to \mathbf{P}_c$ is given by simply changing (V,VI), (IX) and (XII) as follows, deleting (XIII) and replacing $a^*x.P$ with $a^\star x.P$ in other rules in Definition A.1.

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