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

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

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

2.2 Concurrent Combinators

Concurrent combinators are tractable and powerful self-contained proper subset of the asynchronous π-terms, just as **S** and **K** are of λ-calculi. Atomic agents are formed from atoms by connecting "ports" to real "locations" (names), and their

(iv) *If Y*

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

DEFINITION 3.9. (a needed redex pair)

(1) Let Δ be a tuple of occurrences, say $\Delta = \langle u_1,\ u_2 \rangle$, and write $P \xrightarrow{\Delta} P'$ if $P \xrightarrow{\tau} P'$ is obtained by interaction between $\mathbf{c}(x^{-}\tilde{v}) \stackrel{\mathrm{def}}{\equiv} P/u$

Notice either $\mathbf{b}_r$

PROOF. By $\mathbf{s}(abc) \approx (\nu\, e)(\mathbf{s}_m(aeb) \,|\, \mathbf{b}_l(ec))$

begin with the formulation of *separation*.

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

By the main theorem and lemma 3.2, we have

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

4.1 Local π-calculus

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

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

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

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

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

Here we briefly observe that this system can be regarded as an independent powerful subsystem. First we note that it is a t-subsystem. Next by the same way in [22, 25], local polyadic input agent a

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

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

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

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

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

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

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

DEFINITION 4.5. (independence) Assume $P_0 \xrightarrow{\Delta_0} P_1 \xrightarrow{\Delta_1} P_2 \xrightarrow{\Delta_2} \cdots \xrightarrow{\Delta_{n-1}} P_n$ where $n \geq 1$ and $\mathbf{c}_{1,2}(\tilde{v}_{1,2}) \stackrel{\mathrm{def}}{\equiv} P_n/u_{1,2}$ with $u_1 \neq u_2$. We say a sequence of needed 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 2.3 (i). For the converse inclusion, we note $\mathbf{s}(abc) \notin \mathbf{P}_{\mathrm{Lin}}$ but we have $\mathbf{s}(abc) \approx ax.by.(\overline{c}y \mid (\nu\, b)\overline{b}x)$. Then we use Lemma 3.16. $\square$

REMARK 4.9.

- We have observed that $\mathbf{d}(abc)$ represents two roles in a concise way: sharing of names and increment of parallelism, and extraction of parallelism from it gives rise to two proper π-calculi. For further examination of parallelism, it is proved that 0-distributer $\mathbf{d}_0(abc) \stackrel{\mathrm{def}}{=} ax.(\nu\, ee')(\overline{b}e \mid \overline{c}e')$ can not be generated in $\mathbf{P}_{\mathbf{cc}} \backslash \mathbf{d}$ and can not generate $\mathbf{d}_1$ by Proposition 4.7. More exactly, we have: $\mathbf{C} \backslash \mathbf{d} \lnsim \mathbf{C} \backslash \mathbf{d} \cup \{\mathbf{d}_0(abc)\} \lnsim \mathbf{C}_{\mathrm{Af}}$, but a proper subset generated by $\mathbf{C} \backslash \mathbf{d} \cup \{\mathbf{d}_0(abc)\}$ seems to have no interest.
- Causality of communication in π-calculus was studied based on parametric labelled transition systems in [13, 54, 7] from more general viewpoints. On the other hand, neededness and independence between sequences of reduction relations (τ-actions) in our concurrentxaduc-
titt

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

First we introduce a new formulation of measuring expressive power based

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

PROOF

(2) (sharing of names) *There is no standard encoding from* $\mathbf{P}_{\pi}$ *to any subsystem of* $\mathbf{P}_{Af}$*, hence* $\mathbf{P}_{Af} \lnsim^{e} \mathbf{P}_{\pi}$.[12]

6 Discussion

6.1 Summary of the Results

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

Local π*-calculus*

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

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

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

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

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

Related with (1), in the long version of [6], he showed that his encoding is closed under

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

- Related with this, our result in Section 4 tells us that all computable functions can be expressed in the local π-calculus. More interestingly, the encoding of neither call-by-value nor lazy λ-calculus in [34] works in π_{Af}-calculus

rently partially supported by EPSRC GR/K60701.

References

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

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

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

[51] Riely, J. and Hennessy, M., A Typed Language for Distributed Mobile Processes. *POPL'98*, pp.378–390, ACM Pres390, s

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

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

Then $\stackrel{l}{=}$

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

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

Since

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

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

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

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

This proposition is important.

PROPOSITION D.2. .

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

(ii) *For any substitution* σ, $[\![P\sigma]\!] =$ R8950Td(an)Tj/R370.16968T8T071123128