Combinatory Representation of Mobile Processes*

Kohei Honda †
kohei@mt.cs.keio.ac.jp

Nobuko Yoshida
yoshida@mt.cs.keio.ac.jp

Department of Computer Science, Keio University
3-14-1, Hiyoshi, Kohoku-ku, Yokohama 223, Japan

Abstract

A possible analogue of theory of combinators in the setting of concurrent processes is formulated. The new combinators are derived from the analysis of the operation called *asynchronous name passing*, just as the analysis of *logical substitution* gave rise to the sequential combinators. A system with seven atoms and fixed interaction rules, but with no notion of prefixing, is introduced, and is shown to be capable of representing input and output prefixes over arbitrary terms in a behaviourally correct way, just as SK-combinators are closed under functional abstraction without having it as a proper syntactic construct. The basic equational correspondence between concurrent combinators and a system of asynchronous mobile processes, as well as the embedding of the finite part of π-calculus in concurrent combinators, is proved. These results will hopefully serve as a cornerstone for further investigation of the theoretical as well as pragmatic possibilities of the presented construction.

1 Introduction

The notion of combinators [23, 5, 9], which was independently discovered by Schönfinkel and Curry and has been studied extensively by the latter and his school as well as by other researchers for decades, provides a means of rigorous analysis of the procedure of logical substitution, and, as such, has turned out to be significant in theories and practices of computing. This is because many notions of programming and computation involve the treatment of substitution in essential ways. A typical example can be found in the paradigm of computing based on the idea of *functions.* While the λ-notation provides a neat way of writing down higher-order functions, and consequently becomes a basis of programming language methodologies, combinators give another way of representing the same thing in terms of finite atoms and their combination, enjoying notable conceptual/mechanical simplicity; an elementary, yet culminating, fact tells us that the only two atoms, named "**S**" and "**K**", suffice for representing arbitrarily complex applicative behaviour. This ultimately led us to the notion of *combinatory algebra* as a semantic foundation of typed and untyped λ-calculi. At the same time, the decomposition of the application-substitution process into finite dynamics in combinators has had a profound impact on the execution schemes of modern functional programming languages. Truly, parallel developments in the study of λ-calculi and that of combinators are essential to our current practice of sequential programming, both theoretical and pragmatic.

Such parallel developments, however, have not been known in the world of concurrent processes. Nor, at least until recently, has one agreed upon the existence of such an essential operation as β*-reduction* in the concurrency setting. Yet nowadays we find, especially among researchers on concurrency, growing interest in one simple yet powerful primitive, which, when coupled with basic operators like concurrent composition and name hiding, can represent quite versatile structures of concurrent computation. The operation is *name passing*, where a process passes the names of communication channels to another process in its message, and the enunciation of its power was done, for example, in a recent work by Robin Milner, where he showed that β-reduction of certain λ-calculi can be concisely represented as combination of synchronous monadic name passing [20]. He called the calculus

*Copyright 1994 ACM. Appeared in the Proceedings of the 21st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, January 1994, pp. 348–360. Slightly revised on September, 1995.

†Partially supported by JSPS Fellowships for Japanese Junior Scientists.

he used *π-calculus*, originally developed with Joachim Parrow and David Walker [19], which is essentially a non-trivial simplification of ECCS [6]. Based upon the preceding work, however, a further simplification was done by one of the present authors [10] and, independently, by Gerald Boudol [4], without sacrificing any expressive power. Now the following simple idea of reduction emerges.

$$ax.P,\ \leftarrow ab\ \longrightarrow\ P\{b/x\}\ .$$

In the above, $ax.P$ denotes a process which receives a name through a port a and then launches P after appropriate substitution, $\leftarrow ab$ is a message carrying a name b to a, and "," denotes concurrent composition. Thus, in the simplified system, which is now called *ν-calculus* [11, 13], the output prefix has been taken away. What is significant is that this asynchronous construct can encode synchronous communication quite concisely [10, 4], and thus is capable of doing all that can be done by synchronous name passing, representing versatile interaction structures in a clean framework. Perhaps more essential to the present context, however, is the observation that this operation seems so simple and tractable, that it would allow us to perform an analysis similar to what has been done for β-reduction, the analysis which, in exchange for a finite number of atoms, should make it unnecessary to have an explicit syntactic construct for any prefixing (so the input prefix would also be taken away), just as **S** and **K** make it unnecessary to have λ-abstraction in the functional setting.

We shall show in subsequent sections that such an analysis can indeed be performed; that the traditional technology for decomposing logical substitution is useful even in the concurrency setting; and the analysis actually results in seven atoms which embody the basic units of behaviour of processes, reminiscent of such elementary combinators as **S**, **K** and **I** in some aspects. Atoms, however, arise somewhat differently in this setting, since the composition of atoms is done not only at the level of terms but also by connection of channels (or by sharing of names). Thus atomic agents are formed from atoms by connecting "ports" to real "locations". For example, if $\mathcal{M}$ is an atom for a message, and if we connect its "target port" to a and its "value port" to b, we get $\mathcal{M}(ab)$, a message carrying a value b to a. Interaction rules are specified between atoms, in a way reminiscent of Lafont's Interaction Net [18], from which the reduction relation is formed. *Concurrent composition* and *name hiding* are identified as the essential "glues" to combine atomic agents, partly for simplicity and partly for their genuine expressive power (cf. [20, 10]). We shall show, in this framework, that this system of combinators is capable of representing input and output prefixes over arbitrary terms in a behaviourally correct way (as indicated in Section 6, we can even represent *replication* with a handful of additional atoms, but we shall not go that far in the present exposition). Finally, we establish the essential equivalence between the finite part of ν-calculus and concurrent combinators in the equational setting, as well as the embedding of the finite π-calculus in the latter. We hope that these basic results will provide a cornerstone for further theoretical as well as pragmatic investigation of the presented formal framework, giving a positive impact on the principles of concurrent programming such as formally founded execution schemes, semantic analysis/verification, and optimization technologies.

Outline of this paper: Section 2 introduces the asynchronous ν-calculus (without replication) and its equational theory. Section 3 introduces the system of concurrent combinators with seven atoms. Behavioural equivalences, naturally equipped with the notion of dyadic interaction, are formulated and studied. Section 4 presents the representation of input prefixing in concurrent combinators and verifies its correctness. Section 5 is devoted to the correspondence results between combinators and (finite) name passing calculi, namely the ν-calculus and monadic π-calculus. Section 6 gives comparisons with related work, and points out further topics.

Many proofs are given only in outline. Details are found in [15].

2 The Asynchronous ν-calculus

2.1 Terms and Reduction

ν-calculus, an offspring of π-calculus [19, 20], is a small formalism of concurrency using the notion of asynchronous name passing as the interaction primitive [10, 11]. The simple primitive, coupled with the capability to generate new names, gives ν-calculus enough power to construct versatile structures of interaction, just as the simple operation called *application* in λ-calculus gives it enough power to construct any imaginable applicative behaviour. Below we introduce basic constructions of the calculus without replication, since this is all we need for our present purpose.

Definition 2.1 (ν-terms) *Let* **N** *be a countable set of* names *and* **V** *be a countable set of (name)* variables,[1] *ranged over by* $a, b, c, ..$ *and by* $x, y, z, ..$,

[1] Variables for names are dissuaded by e.g. [19], but are found, especially in the present context, to be essential to distinguish two kinds of binding constructs. The technical results, however, can be carried over to systems without variables, as can be easily seen.

respectively. $u, v, w, ..$ *range over their union, the set of* identifiers. *We suppose that* $\mathbf{N}$ *and* $\mathbf{V}$ *are mutually disjoint. Then the set of* ν*-terms* $\mathbf{P}_\nu$, *ranged over by* $P, Q, ...,$ *is given by the following grammar.*

$$P \quad ::= \quad \leftharpoonup uv \quad | \quad ux.P \quad | \quad P, Q \quad | \quad a \blacktriangleright P \quad | \quad \Lambda$$

Among terms, "$\leftharpoonup uv$" denotes a *message* to a target u carrying a value v, while "$ux.P$" denotes a *receptor* which receives a message and instantiates the value in its body. In $ux.P$, the variable x binds free occurrences of x in P (like x in $\lambda x.M$). We may call the abstraction over x in $ux.P$, *u-pointed name abstraction.* Here "ux" is a *prefix* or an *input guard.* "$a \blacktriangleright P$" is a *scope restriction* of a in P, showing that a in P is local to P. Here the initial a binds its free occurrences in P. "P, Q" is a *concurrent composition* of P and Q. "Λ" (to be read "nil") is a syntactic convention used to denote nothing.

The set of *free (resp. bound) names* in P is denoted by $\mathcal{FN}(P)$ (resp. $\mathcal{BN}(P)$). The set of *free (resp. bound) variables* in P is denoted by $\mathcal{FV}(P)$ (resp. $\mathcal{BV}(P)$). We also assume the usual notion of (multiple) substitutions, written $\{\tilde{v}/\tilde{u}\}$ where $\tilde{u}$ and $\tilde{v}$ are strings of identifiers (names or variables) with the same length and all identifiers in $\tilde{u}$ are distinct. In the present exposition, a substitution should always map a variable to a name, a variable to a variable, or a name to a name, but not a name to a variable. σ, σ' etc., range over the set of substitutions. $\equiv_\alpha$ denotes α-convertibility in terms of both names and variables. *Closed* and *open* terms are defined as usual (with respect to name variables). Thus $\leftharpoonup ab$ and $ax.\leftharpoonup xb$ are closed but $\leftharpoonup yc$ and $ax.\leftharpoonup cz$ are open. Then, for an open term P, σ from variables to names is a *closing substitution* if $\mathcal{FV}(P\sigma) = \emptyset$.

Some conventions: $ab \blacktriangleright P$ denotes $a \blacktriangleright (b \blacktriangleright P)$; "," is the weakest in association, e.g. $a \blacktriangleright P, Q \stackrel{\text{def}}{=} (a \blacktriangleright P), Q$, and associates to the left. In spite of these conventions, parentheses will be used freely to make syntactic structures explicit.

Henceforth, terms are often considered modulo the following structural congruence, following Milner [20] (cf. [3]).

Definition 2.2 $\equiv$ *is the smallest congruence relation over* ν*-terms generated by the following rules.*

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

(ii) $P, Q \equiv Q, P \qquad (P, Q), R \equiv P, (Q, R) \qquad P, \Lambda \equiv P$

(iii) $aa \blacktriangleright P \equiv a \blacktriangleright P \qquad ab \blacktriangleright P \equiv ba \blacktriangleright P \qquad a \blacktriangleright \Lambda \equiv \Lambda$
$a \blacktriangleright P, Q \equiv a \blacktriangleright (P, Q)$ if $a \notin \mathcal{FN}(Q)$

The reduction relation provides the basic notion of computing in the formalism. The underlying idea is a primitive notion of interaction where a piece of information is simply consumed by a computing entity which, in turn, generates a new term after substitution.

Definition 2.3 (reduction relation) *One step reduction relation over* ν*-terms,* $\longrightarrow$, *is the smallest relation generated by:*

(COM)	$ux.P, \leftharpoonup uv \longrightarrow P\{v/x\}.$
(PAR)	$P \longrightarrow Q \Rightarrow P, R \longrightarrow Q, R.$
(RES)	$P \longrightarrow Q \Rightarrow a \blacktriangleright P \longrightarrow a \blacktriangleright Q.$
(STR)	$P \equiv P' \; P' \longrightarrow Q' \; Q \equiv Q' \Rightarrow P \longrightarrow Q.$

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

2.2 Agents and their Reduction

Some important agents are listed below. Note, by (3), we gain the power of the *output prefix,* together with that of *multiple name passing.* Note also, in the same example, how private names are used to make the intermediate interactions safe.

(1) (Forwarder) The left-hand side agent "forwards" a message.[2]

$$ax.\leftharpoonup bx, \;\; \leftharpoonup ae \quad \longrightarrow \quad \leftharpoonup be$$

(2) (Switcher) The left-hand side agent sends back a value it holds to the name it receives.

$$ax.\leftharpoonup xb, \;\; \leftharpoonup ae \quad \longrightarrow \quad \leftharpoonup eb$$

(3) (Synchronous, multiple name passing) Assume that z, c, and y are all fresh.

$$a : (x_1..x_n).P \stackrel{\text{def}}{=} ay.c \blacktriangleright (\leftharpoonup yc, cx_1.c \blacktriangleright (\leftharpoonup yc, ..c \blacktriangleright (\leftharpoonup yc, cx_n.P)..))$$
$$\bar{a} : [v_1..v_n].P \stackrel{\text{def}}{=} c \blacktriangleright (\leftharpoonup ac, cz.(\leftharpoonup zv_1, cz.(.., cz.(\leftharpoonup zv_n, P)..)))$$

Then we have:

$$a : (x_1x_2).P, \; \bar{a} : [v_1v_2].Q \longrightarrow\!\!\!\!\!\rightarrow P\{v_1v_2/x_1x_2\}, Q$$

2.3 Equality over Agents

Equality over ν-terms has been studied extensively in [11, 13]. Here we employ a construction which is naturally applicable to both ν-calculus and concurrent combinators in general, inducing behaviourally

[2] The name *forwarder* comes from a similar agent in the context of Hewitt's *actors* [8].

significant equality over agents in each case with little technical overhead. The formulation is based on *reduction-closure* for equality [13] as well as *action predicate* which is a refinement of *convergence predicate* in the line of e.g. [1, 4, 22] in the scheme of dyadic interaction. Below we naturally assume that any congruence (say $\cong$) satisfies the usual proviso for open terms, i.e. if $P \cong Q$ with P or Q open, then $P\{v/x\} \cong Q\{v/x\}$ for any v and x.

We first stipulate what we regard as one of the essential criteria for meaningful equality over processes, just as the inclusion of β-convertibility is essential for any λ-theories. We say that a congruence $\cong$ over ν-terms is *reduction-closed* iff, whenever $P \cong Q$, $P \longrightarrow\!\!\!\!\rightarrow P'$ implies, for some Q', $Q \longrightarrow\!\!\!\!\rightarrow Q'$ and $P' \cong Q'$. In essence, we require that the notion of equality should be consistent with state changes induced by reduction. Since the β-equality, i.e. $P \longrightarrow\!\!\!\!\rightarrow P' \Rightarrow P = P'$, satisfies the closure property vacuously, we would say that the property generalises the β-equality into the stateful regime (the usage of $\longrightarrow\!\!\!\!\rightarrow$ instead of $\longrightarrow$ is essential in this aspect; think of $II =_\beta I$ with $I \stackrel{\text{def}}{=} \lambda x.x$). Note also that the congruent closure of the union of a family of reduction closed congruences is again reduction-closed.

As argued in [13], the notion by itself may not induce a behaviourally significant congruence. Thus we introduce another element into our equational construction, and form the notion of soundness in equality combining the two. Let $a \in \mathcal{AN}_+(P)$ iff $P \equiv \tilde{c} \blacktriangleright (\leftarrow av, R)$ with $a \notin \{\tilde{c}\}$ and $a \in \mathcal{AN}_-(P)$ iff $P \equiv \tilde{c} \blacktriangleright (ax.Q, R)$ with $a \notin \{\tilde{c}\}$ where $\{\tilde{c}\}$ is the set of names in $\tilde{c}$. Then, with θ ranging over $+$ and $-$, let us define a family of (untyped) *action predicates* over closed ν-terms, as follows.

$$P \Downarrow_{a^\theta} \stackrel{\text{def}}{\Leftrightarrow} \exists P'.\, P \longrightarrow\!\!\!\!\rightarrow P' \wedge a \in \mathcal{AN}_\theta(P')$$

The soundness criterion follows.

Definition 2.4 *We say a congruence* $\cong$ *is* $\Downarrow_{a^\theta}$-sound, *or simply* sound, *if it includes* $\equiv$, *is reduction closed, and, moreover,* $\cong$ respects $\Downarrow_{a^\theta}$, *i.e. if* $P \cong Q$ *and* $P \Downarrow_{a^\theta}$ *then* $Q \Downarrow_{a^\theta}$.

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:

Proposition 2.5 *There is a maximum sound congruence within the family of all sound congruences.*

We call this maximum sound equality $=_\nu$. We will be using $=_\nu$ as the representative notion of equality over ν-terms.[3]

3 Concurrent Combinators (1): Basic Construction

3.1 Combinators in a Concurrency Setting

Concurrent combinators are based on the notion of dyadic interaction. The basic idea is that two atoms interact via a common interaction port to generate new nodes and a new connection topology. To represent multiple interaction points and their connection topology in a general way, the notion of *names* is introduced from process calculi. As mentioned in the Introduction, this means that we have to, at least syntactically, parameterize atoms with names to denote how the internal ports are connected to real locations. But behaviour *is* constant, and the present way of representation seems inevitable, as long as we rely on syntactic formulation (cf. 6.1). Thus $\mathcal{M}(ab)$ would mean a message with its target port connected to a and a value port to b, while $\mathcal{FW}(ab)$ would be a forwarder with an input port connected to a and an output port to b. Then we may have an interaction of the form

$$\mathcal{FW}(ab), \mathcal{M}(ae) \longrightarrow \mathcal{M}(be) \quad .$$

This is actually a rule schemata (as $\mathbf{K}MN \longrightarrow M$), saying that any forwarder and any message with the common initial port would interact in this way. Here "," denotes concurrent composition, and we shall also allow "$\blacktriangleright$", the scope restriction, as another basic connective, but that is all. We begin with seven basic atoms. They arise in the course of "analysing away" the input prefix, as will become clear later.

Definition 3.1 *(atoms)* We assume a set of atoms, $\mathbf{A} = \{\mathcal{M}, \mathcal{D}, \mathcal{FW}, \mathcal{K}, \mathcal{B}_l, \mathcal{B}_r, \mathcal{S}\}$, ranged over by $\mathcal{C}$, $\mathcal{C}', \ldots$, with two associating functions $ar : \mathbf{A} \to \mathbb{N}^+$ and $pol : \mathbf{A} \to \{+, -\}$, whose values are:

(i) $ar(\mathcal{M}) = ar(\mathcal{FW}) = ar(\mathcal{B}_l) = ar(\mathcal{B}_r) = 2$, $ar(\mathcal{D}) = ar(\mathcal{S}) = 3$, $ar(\mathcal{K}) = 1$.

(ii) $pol(\mathcal{M}) = +$, for others $pol(\mathcal{C}) = -$.

$ar(\mathcal{C})$ and $pol(\mathcal{C})$ are called the *arity* and the *polarity* of $\mathcal{C}$, respectively. $\theta, \theta', ..$ range over $\{+, -\}$.

[3] We note that, using the equality notions as presented in [13, 22], the main results in this paper still hold.

$$
\begin{array}{rcl@{\qquad}rcl}
\mathcal{D}(uww'),\, \mathcal{M}(uv) & \longrightarrow & \mathcal{M}(wv),\, \mathcal{M}(w'v) & \mathcal{B}_l(uw),\, \mathcal{M}(uv) & \longrightarrow & \mathcal{FW}(vw) \\
\mathcal{FW}(uw),\, \mathcal{M}(uv) & \longrightarrow & \mathcal{M}(wv) & \mathcal{B}_r(uw),\, \mathcal{M}(uv) & \longrightarrow & \mathcal{FW}(wv) \\
\mathcal{K}(u),\, \mathcal{M}(uv) & \longrightarrow & \Lambda & \mathcal{S}(uww'),\, \mathcal{M}(uv) & \longrightarrow & \mathcal{FW}(ww')
\end{array}
$$

Figure 1: Reduction Rules for Atomic Agents

The functionalities of polarity and arity will be explained later. Now we form terms. We use the sets **N** and **V** in the previous section, and the same symbols denote their elements.

Definition 3.2 (cc-terms) *The set of of* terms, $\mathbf{P}_{cc}$, *is defined by the following grammar. $P, Q, R, ..$ again range over the set.*

$$P \quad ::= \quad \mathcal{C}(\tilde{u}^{(n)}) \quad | \quad P, Q \quad | \quad a \blacktriangleright P \quad | \quad \Lambda$$

where $\mathcal{C} \in \mathbf{A}$, $ar(\mathcal{C}) = n$, and $\tilde{u}^{(n)}$ is a sequence of identifiers of length n.

We assume "," to be weaker in association than $\blacktriangleright$, as in ν-terms. Terms are sometimes called *agents*. An agent of the form $\mathcal{C}(\tilde{u})$ is called an *atomic agent*. The polarity of an atomic agent is the same as its associated atom. We sometimes write $\mathcal{C}^{\theta}(\tilde{w})$, $\mathcal{C}^{(n)}(\tilde{w})$, or $\mathcal{C}^{\theta(n)}(\tilde{w})$, to inform the arity and/or the polarity of atom.

The name or the variable in the leftmost position in an atomic agent, e.g. "a" in $\mathcal{M}(ab)$, is called its *subject*. Other names/variables are called *objects*. The subject is the location at which atomic agents with opposite polarities interact, thus $\mathcal{C}_1^{+}(u\tilde{v})$ and $\mathcal{C}_2^{-}(u\tilde{w})$ may interact. The only binding is by the operator $\blacktriangleright$. α-convertibility, $\equiv_\alpha$, is defined in the standard way. $\mathcal{FN}(P), \mathcal{BN}(P), \mathcal{FV}(P)$ are defined in the same way as in ν-terms (we have no bound variables). *Open terms* are those with free variables, while *closed terms* are those without free variables. The notion of *closing substitutions* is defined as in ν-terms. Finally, $\mathcal{AN}_+(P)$ (resp. $\mathcal{AN}_-(P)$) is the set of free names occurring at the subject position of atomic agents whose polarity is $+$ (resp. $-$), where we say that such occurrences are *active*.

The basic algebra over terms is defined using exactly the same set of rules as in Definition 2.2, and denoted by $\equiv$. The definition of reduction relation starts from fixed rules of interaction between atomic agents.[4]

[4] The rules for atomic agents can be formed only by using atoms in the nameless format, which has its own advantages. We believe, however, that the present formulation is much easier to grasp.

Definition 3.3 (reduction relation) *One-step reduction relation, $\longrightarrow$, is the smallest relation generated by the rules in Figure 1, together with* (PAR), (RES) *and* (STR) *in Definition 2.3. Then* multi-step reduction relation, *$\longrightarrow\!\!\!\!\rightarrow$, is defined by $\longrightarrow\!\!\!\!\rightarrow \stackrel{\mathrm{def}}{=} \longrightarrow^* \cup \equiv$.*

Note that there is one and only one rule for each pair of atoms with opposite polarities, and, moreover, that the set of free identifiers in the right-hand side term is a subset of that in the original term. These properties would naturally be considered to be part of the criteria for reduction rules of (untyped) concurrent combinators in general (note that we regard polarities as one of the essential elements of computation, cf. [18]).

Let us give some illustrations on the behaviour of each atomic agent: firstly, $\mathcal{M}(ab)$ is a *message* carrying a name b to a name a. $\mathcal{D}(abc)$ is a *duplicator*, which duplicates a received message and sends it to two different locations. Then $\mathcal{FW}(ab)$ is a *forwarder* which simply forwards a message to another location (cf. 2.2). $\mathcal{K}(a)$ is called a *killer*, which kills a message.[5] The next two agents, called *binders*, are more complex. Each binder generates a new forwarder using the name it receives in its own way. $\mathcal{B}_l(ab)$, a left binder, uses the received name at the left of a forwarder, while $\mathcal{B}_r(ab)$, a right binder, uses it at the right. The final atom is a *synchroniser*, $\mathcal{S}(abc)$, used for pure synchronisation without value passing, often necessary in interaction scenarios. Note that binders and a synchroniser change the topology of communication by generating a new "link" (a forwarder) after interaction in their own ways.

3.2 Agents and their Reduction

Some agents with interesting behaviours, are given below. Specifically, (3) shows how multiple name passing can be done only by using atomic agents and their combination. Note again how private names are used to make intermediate interaction safe.

[5] $\mathcal{FW}$ and $\mathcal{K}$ are, in fact, definable from $\mathcal{D}$, in exchange for a certain (manageable) syntactic issue in Theorem 4.4 later. See [15].

(1) Let $\mathcal{D}_3(abcd) \stackrel{\mathrm{def}}{=} e \blacktriangleright (\mathcal{D}(abe),\ \mathcal{D}(ecd))$. Then:

$$\begin{aligned}\mathcal{M}(av),\ \mathcal{D}_3(abcd) &\longrightarrow \mathcal{M}(bv),\ e \blacktriangleright (\mathcal{M}(ev),\ \mathcal{D}(ecd))\\ &\longrightarrow \mathcal{M}(bv),\ \mathcal{M}(cv),\ \mathcal{M}(dv)\end{aligned}$$

$\mathcal{D}_4$, $\mathcal{D}_5$, .. can be defined similarly.

(2) Let $\mathcal{SW}(ab) \stackrel{\mathrm{def}}{=} c \blacktriangleright (\mathcal{B}_r(ac),\ \mathcal{M}(cb))$. Then:

$$\begin{aligned}\mathcal{SW}(ab),\ \mathcal{M}(ad) &\longrightarrow c \blacktriangleright (\mathcal{FW}(cd),\ \mathcal{M}(cb))\\ &\longrightarrow \mathcal{M}(db)\end{aligned}$$

This is a *switcher*, already introduced in 2.2.

(3) Let us define

$$\begin{aligned}\mathcal{SW}'(c_1vcc_2) &\stackrel{\mathrm{def}}{=} dd' \blacktriangleright (\mathcal{D}(c_1dd'),\ \mathcal{SW}(dv),\ \mathcal{S}(d'cc_2))\\ \mathcal{R}(e_1e_2ue_1'e_2') &\stackrel{\mathrm{def}}{=} d_1d_2d_3 \blacktriangleright (\mathcal{D}(e_1e_1'd_1),\ \mathcal{D}(e_2ud_2),\\ &\qquad \mathcal{S}(d_2d_1d_3),\ \mathcal{SW}(d_3e_2'))\ .\end{aligned}$$

Assuming these notations, the following defines agents, the first of which sends n names consecutively, while the second receives them and transmits each to different locations (we write $\prod_{i=1}^{n} P_i$ for $P_1, P_2, ..., P_n$).

$$\begin{aligned}&\leftarrow a^*\colon v_1v_2..v_n\\ &\stackrel{\mathrm{def}}{=}\ cc_1..c_n \blacktriangleright (\mathcal{M}(ac), \mathcal{FW}(cc_1),\\ &\qquad \textstyle\prod_{i=1}^{n-1} \mathcal{SW}'(c_iv_icc_{i+1}),\\ &\qquad \mathcal{SW}(c_nv_n))\\ &\rightarrow a^*\colon u_1u_2..u_n\\ &\stackrel{\mathrm{def}}{=}\ e_0e_{11}..e_{n2} \blacktriangleright (\mathcal{D}(ae_0e_{11}), \mathcal{SW}(e_0e_{12}),\\ &\qquad \textstyle\prod_{i=1}^{n-1} \mathcal{R}(e_{i1}e_{i2}u_ie_{(i+1)1}e_{(i+1)2}),\\ &\qquad \mathcal{S}(e_{n1}e_{n2}u_n))\end{aligned}$$

Then, by tracing the reduction, we get

$$\begin{aligned}&\leftarrow a^*\colon v_1v_2..v_n,\ \rightarrow a^*\colon u_1u_2..u_n\\ &\qquad \longrightarrow\!\!\!\!\rightarrow\ \mathcal{M}(u_1v_1),\ \mathcal{M}(u_2v_2), .., \mathcal{M}(u_nv_n)\ .\end{aligned}$$

Thus values are successfully transmitted in the expected order.

3.3 Equality over Agents

We formulate the notion of equality over agents in the same way as we did for ν-terms. Since the formulation of a labelled transition relation itself seems far from obvious in the present setting, a construction based on the reduction relation is essential. Equality over open terms is understood as for ν-terms.

Firstly, we have the notion of *reduction closure* in exactly the same way as before. The notion of the *action predicate* is given by: $P \Downarrow_{a^\theta} \stackrel{\mathrm{def}}{\Leftrightarrow} \exists P'.\ P \longrightarrow\!\!\!\!\rightarrow P' \ \wedge\ a \in \mathcal{AN}_\theta(P')$. Then, as before, a congruence $\cong$ is $\Downarrow_{a^\theta}$*-sound*, or simply *sound*, iff it includes $\equiv$, is reduction closed, and, moreover, $P \cong Q$ and $P \Downarrow_{a^\theta}$ implies $Q \Downarrow_{a^\theta}$. The following proposition is again easily obtained.

Proposition 3.4 *There is a maximum sound congruence among the family of all sound congruences.*

We call this maximum sound equality $=_{cc}$.

The scheme of having a canonical notion of equality as the representative of a family of meaningful "small" equalities is often useful. As a tractable subset of $=_{cc}$, we introduce β-equality, which is extensively used in subsequent sections. Firstly, we need a notion called *pointedness*. A closed term P is a^θ*-pointed*, iff P satisfies the following condition ($\bar{\theta}$ is the inverse of θ).

(i) $\mathcal{AN}_\theta(P) = \{a\}$ and $\mathcal{AN}_{\bar{\theta}}(P) = \emptyset$.

(ii) Above a is the unique active occurrence in P.[6]

(iii) $P \not\longrightarrow$.

Pointedness of a term tells us that there is only one interacting point in a given term (possibly unfolding further points after interaction). An open term is u^θ-pointed iff, for each closed substitution, the above condition is satisfied by a name substituted for u. We often write $P_{\langle u^\theta \rangle}$ etc. to denote a u^θ-pointed term P. Note that atomic agents are always pointed in this sense. We also note that pointedness is a linearly decidable property for any cc-term.

We are now ready to formulate the β-equality together with related reduction relations.

Definition 3.5 *The one-step β-reduction, $\rightarrow_\beta$, is defined by the rule:*

$$(\mathrm{COM}_\beta)\quad c \blacktriangleright (P_{\langle c- \rangle},\ Q_{\langle c+ \rangle})\ \rightarrow_\beta\ c \blacktriangleright R$$

if $P_{\langle c- \rangle}, Q_{\langle c+ \rangle} \longrightarrow R$, together with (PAR), (RES) and (STR) of Definition 2.3. $\twoheadrightarrow_\beta \stackrel{\mathrm{def}}{=} \rightarrow_\beta^* \cup \equiv$, while $=_\beta$ is the symmetric and transitive closure of $\twoheadrightarrow_\beta$.

To show that $=_\beta$ is sound, the following is crucial. Underlying is the insight of Cliff Jones [16] that the essential role of name hiding in a name passing scenario lies in control of *interference*.

Lemma 3.6 (non-interference) *Suppose $P \twoheadrightarrow_\beta Q_1$ and $P \longrightarrow\!\!\!\!\rightarrow Q_2$. Then there exists Q' s.t. $Q_1 \longrightarrow\!\!\!\!\rightarrow Q'$ and $Q_2 \twoheadrightarrow_\beta Q'$.*

Proof. By argument in terms of one-step reduction, i.e. $P \rightarrow_\beta Q_1$ and $P \longrightarrow Q_2$ with $Q_1 \not\equiv Q_2$ then there exists Q' s.t. $Q_1 \longrightarrow Q'$ and $Q_2 \rightarrow_\beta Q'$, established by the syntactic analysis of residuals. $\square$

[6] Behaviourally we can change this clause as: for any $\mathcal{C}^{\bar{\theta}}$, *if* $(P, \mathcal{C}^{\bar{\theta}}(a\tilde{v})) \longrightarrow Q_1$ *and* $(P, \mathcal{C}^{\bar{\theta}}(a\tilde{v})) \longrightarrow Q_2$ *then* $Q_1 \equiv Q_2$.

Proposition 3.7 *$=_\beta$ is a sound congruence, hence $P =_\beta Q$ implies $P =_{cc} Q$.*

Proof. Suppose $P_1 =_\beta P_2$. By induction on the derivation of $=_\beta$, we can show that there exists P such that $P_1 \twoheadrightarrow_\beta P$ and $P_2 \twoheadrightarrow_\beta P$. For such P, $P_1 \longrightarrow\!\!\!\!\!\rightarrow Q_1$ implies, using Lemma 3.6, for some Q_2, $P \longrightarrow\!\!\!\!\!\rightarrow Q_2$ and $Q_1 \twoheadrightarrow_\beta Q_2$, that is, $Q_1 =_\beta Q_2$. Since $P_2 \twoheadrightarrow_\beta P \longrightarrow\!\!\!\!\!\rightarrow Q_2$, we know the reduction-closure of $=_\beta$. To prove that $=_\beta$ respects the action predicate, we first show that $P \Downarrow_{a^\theta}$ and $P \twoheadrightarrow_\beta Q$ imply $Q \Downarrow_{a^\theta}$, using Lemma 3.6. Then the property easily follows, again using Lemma 3.6. □

As an application of $=_\beta$, let us take a close look at the reduction in (3) of 3.2. Then we get:

$$\begin{aligned}&\leftarrow a^*\!: v_1 v_2..v_n,\ \rightarrow a^*\!: u_1 u_2..u_n\\ &\qquad\longrightarrow\twoheadrightarrow_\beta\quad \mathcal{M}(u_1v_1),\ \mathcal{M}(u_2v_2), .., \mathcal{M}(u_nv_n)\end{aligned}$$

which tells us that the reduction after the initial step is "safe".

4 Concurrent Combinators (2): Representing Prefix

4.1 Analysis of Prefix (1)

If we view λ in λ-abstraction as a kind of interaction point (cf. [1]), the decomposition of functional abstraction in combinatory logic may have a natural analogue in the world of concurrency. Prefix in concurrent processes, however, has an important role not eminent in the λ-abstraction. It is concerned not only with the communication of values but also with the control of *synchronisation* in the setting of concurrency. This fact makes decomposition of name passing into finite dynamics more complex, yet it is indeed possible. The meta-notation for an input guard is written $u^*x.P$, where $*$ shows that this is not a formal construct in the system of concurrent combinators, but a meta-level notation to be mapped to a cc-term (cf. [9]). The resulting term, however, is expected to act as if it had the input prefix ux — until it receives a message it should make the body inert, while, once it does, it should activate the body, after appropriate substitution, for further interaction with the outside. The mapping follows.

Definition 4.1 (name abstraction) *For P being any cc-term, we inductively form the agent denoted by $u^*x.P$, called a u-*pointed abstraction of x over P*, in Figure 2, where rules are applied from* (I) *to* (XII) *in this order, and we assume the following annotations, which denote how each name is used in the rules of interaction.*

$$\mathcal{M}(u^+v^\pm),\ \mathcal{D}(u^-v^+w^+),\ \mathcal{K}(u^-),$$
$$\mathcal{FW}(u^-v^+),\ \mathcal{B}_l(u^-v^+),\ \mathcal{B}_r(u^-v^-),\ \mathcal{S}(u^-v^-w^+)$$

Note that the annotated polarities are preserved under reduction, e.g. $\mathcal{D}(u^-v_1^+v_2^+),\ \mathcal{M}(u^+v') \longrightarrow \mathcal{M}(v_1^+v'),\ \mathcal{M}(v_2^+v')$ etc.

Some commentaries on the translation follow.

(i) In (I), we distribute the received message into two (decomposed) pointed abstractions, safely using private names. Note that not only the value but also the activation is transmitted. The construction is clearly reminiscent of the usage of **S** in combinatory logic when decomposing the abstraction of two applied terms. The next two rules, (II) and (III), may need no explanation (in (II) we assume that fresh names are found uniquely). By rules (I) to (III), we have dealt with the cases where the prefixed body is either composite or nil, as well as the origins of $\mathcal{D}$ and $\mathcal{K}$.

(ii) When the prefixed body is an atomic agent which does not contain any abstracted name variable, the prefixing actually plays functions as nothing but *the control of synchronisation*. This is the origin of $\mathcal{S}$, the synchroniser, see Rules (IV) and (V). Note that the existence of a forwarder is already assumed by $\mathcal{S}$. (IV) is straightforward. In (V), an interaction of the mapped term with a message results in $c \blacktriangleright (\mathcal{FW}(vc), \mathcal{C}^-(c\tilde{w}))$ which is not syntactically the same as $\mathcal{C}^-(v\tilde{w})$ but which has essentially the same behaviour (cf. Theorem 4.4).

(iii) When the prefixed atom contains abstracted variables, unlike in combinatory logic, i.e. $\lambda^*x.x \stackrel{\text{def}}{=} \mathbf{I}$, we should deal with abstracted variables which may occur in various places in each atomic agent, resulting in the seven rule schemes shown in the figure. In (IX) and (X), one "pushes out" the abstracted name variable using the forwarder. Take the case of $u^*x.\mathcal{M}(x^+v)$, which corresponds to rule (IX). This becomes $u^*x.c \blacktriangleright (\mathcal{FW}(cx),\ \mathcal{M}(cv))$. Thus, when a message arrives at "u", the forwarder is launched to "forward" the waiting message. (X) acts in the opposite direction. Rules (VII) and (VIII) explain the origins of the left and right binders (note that these form the base cases for transformations by (IX) and (X)). Finally, abstracted names with negative polarities but not at subject positions, (XI) and (XII), cannot be done eascomposition

(I).	$u^*x.(P,\, Q)$	$\stackrel{\text{def}}{=}$	$c_1c_2 \blacktriangleright (\mathcal{D}(uc_1c_2),\ c_1^*x.P,\ c_2^*x.Q)$	c_1, c_2 fresh.
(II).	$u^*x.c \blacktriangleright P$	$\stackrel{\text{def}}{=}$	$c' \blacktriangleright u^*x.P\{c'/c\}$	c' fresh.
(III).	$u^*x.\Lambda$	$\stackrel{\text{def}}{=}$	$\mathcal{K}(u)$	

synchronization

(IV).	$u^*x.\mathcal{C}(v^+\tilde{w})$	$\stackrel{\text{def}}{=}$	$c \blacktriangleright (\mathcal{S}(ucv),\ \mathcal{C}(c^+\tilde{w}))$	$x \notin \{v\tilde{w}\}$, c fresh.
(V).	$u^*x.\mathcal{C}(v^-\tilde{w})$	$\stackrel{\text{def}}{=}$	$c \blacktriangleright (\mathcal{S}(uvc),\ \mathcal{C}(c^-\tilde{w}))$	$x \notin \{v\tilde{w}\}$, c fresh.

binding-I

(VI).	$u^*x.\mathcal{M}(vx)$	$\stackrel{\text{def}}{=}$	$\mathcal{FW}(uv)$	$x \neq v$
(VII).	$u^*x.\mathcal{FW}(xv)$	$\stackrel{\text{def}}{=}$	$\mathcal{B}_l(uv)$	$x \neq v$
(VIII).	$u^*x.\mathcal{FW}(vx)$	$\stackrel{\text{def}}{=}$	$\mathcal{B}_r(uv)$	$x \neq v$

binding-II

(IX).	$u^*x.\mathcal{C}(\tilde{v_1}x^+\tilde{v_2})$	$\stackrel{\text{def}}{=}$	$c \blacktriangleright u^*x.(\mathcal{FW}(cx),\ \mathcal{C}(\tilde{v_1}c^+\tilde{v_2}))$	$x \notin \{\tilde{v_1}\}$, c fresh.

binding-III

(X).	$u^*x.\mathcal{C}(x^-\tilde{v})$	$\stackrel{\text{def}}{=}$	$c \blacktriangleright u^*x.(\mathcal{FW}(xc),\ \mathcal{C}(c^-\tilde{v}))$	c fresh.	
(XI).	$u^*x.\mathcal{B}_r(vx^-)$	$\stackrel{\text{def}}{=}$	$c_1c_2c_3 \blacktriangleright u^*x.(\mathcal{D}(vc_1c_2),\ \mathcal{S}(c_1xc_3),\ \mathcal{B}_r(c_2c_3))$	$x \neq v$	c_1, c_2, c_3 fresh.
(XII).	$u^*x.\mathcal{S}(vx^-w)$	$\stackrel{\text{def}}{=}$	$c_1c_2 \blacktriangleright u^*x.(\mathcal{S}(vc_1c_2),\ \mathcal{M}(c_1x),\ \mathcal{B}_l(c_2w))$	$x \neq v$	c_1, c_2 fresh.

Figure 2: Name Abstraction

ily using forwarders as in (X), due to the control of the timing of synchronisation (note that e.g. $\mathcal{B}_r(ab)$ and $c \blacktriangleright (\mathcal{B}_r(ac), \mathcal{FW}(bc))$ are essentially different[7]). Listed mappings seem justifiable only behaviourally.

While complex in several cases, (I) to (VIII) always diminish the size of the prefixed body, and all the positively abstracted variables are pushed out by (IX). For negatively abstracted variables, (X) boils down to (VII), (XI) and (XII), while (XI) is reduced to (XII), and (XII) is reduced to (IV) or (VI) (note w in $\mathcal{B}_l(c_2w)$ occurs positively), which, as a whole, tells us that the mapping is well-defined.

We also note that the mapping satisfies expected elementary syntactic properties such as $\mathcal{FV}(a^*x.P) = \mathcal{FV}(P) \setminus \{x\}$, $a^*x.P \equiv_\alpha a^*y.P\{y/x\}$ and $(a^*x.P)\{v/u\} \equiv_\alpha a\{v/u\}^*x.P\{v/u\}$ if $u \neq x$.

4.2 Analysis of Prefix (2)

In the following, we will show that the behavior of $a^*x.P$ is essentially as we expect it to be, i.e. $(a^*x.P,\ \mathcal{M}(ab))$ would reduce to the behaviour corresponding to $P\{v/x\}$, and also $a^*x.P$ "freezes" the body P. The latter property can be easily verified, using the notion of pointedness we introduced in the previous section.

Proposition 4.2 (Pointedness) *For any a, x, and P, $a^*x.P$ is a^--pointed.*

PROOF. Firstly we show, by rule induction, that a with the negative polarity is the only active occurrence in $a^*x.P$. Using this, inductive reasoning again

[7] This comes from the asymmetric character of forwarders. A symmetric alternative is possible, although this raises its own complications. See [15].

shows that no redex is formed at any stage of translation, including the bottom cases. □

The validation of whether $(a^*x.P, \mathcal{M}(ab))$ really reduces as we expect it to, however, is not straightforward. In fact, as we already saw, we have $a^*x.\mathcal{FW}(b_1b_2), \mathcal{M}(ab) \longrightarrow c\blacktriangleright(\mathcal{FW}(b_1c), \mathcal{FW}(cb_2)) \not\equiv \mathcal{FW}(b_1b_2)$. Thus, it cannot always be the case that $a^*x.P, \mathcal{M}(ab) \twoheadrightarrow P\{b/x\}$. This is due to the issue of *synchronisation* we discussed at the outset of this section. Take the above example. To "freeze" the body $\mathcal{FW}(b_1b_2)$ by a prefix, it cannot be preserved in this form since we only have concurrent composition as a mode of combining two terms (cf. $\lambda^*x.\mathbf{S} \stackrel{\mathrm{def}}{=} \mathbf{KS}$). Thus we "hide" the subject b_1 until necessary, preventing it from interaction. With this complication, however, one can easily see that, if $a^*x.P, \mathcal{M}(ab) \longrightarrow Q$ then Q behaves essentially in the same way as $P\{b/x\}$. To clarify this point, we define $\succ_i$, a series of strongly syntactic behavioural preorders, inductively as follows, where we write $P \downarrow_\beta P'$ iff $P \twoheadrightarrow_\beta P' \not\rightarrow_\beta$ (note that such P' is unique up to $\equiv$).

(i) $\succ_0 \stackrel{\mathrm{def}}{=} \equiv$.

(ii) a. $P \stackrel{\bullet}{\succ}_{n+1} \mathcal{C}^\theta(u\tilde{v})$ iff $P \downarrow_\beta P'_{\langle u^\theta \rangle}$, and, for any $\mathcal{C}^{\overline{\theta}}$, we have:

$$P', \mathcal{C}^{\overline{\theta}}(u\tilde{w}) \longrightarrow Q' \Rightarrow (\mathcal{C}^\theta(u\tilde{v}), \mathcal{C}^{\overline{\theta}}(u\tilde{w}) \longrightarrow Q \ \wedge\ Q' \succ_n Q),$$

$$\mathcal{C}^\theta(u\tilde{v}), \mathcal{C}^{\overline{\theta}}(u\tilde{w}) \longrightarrow Q \Rightarrow (P', \mathcal{C}^{\overline{\theta}}(u\tilde{w}) \longrightarrow Q' \ \wedge\ Q' \succ_n Q).$$

b. $\succ_n$ is the compatible and transitive closure of $\stackrel{\bullet}{\succ}_n \twoheadrightarrow_\beta \cup \equiv$.

Then $P \succ Q$ iff, for some n, we have $P \succ_n Q$. Intuitively, $P \succ Q$ means that any atomic agent in Q has its close "simulator" in P, whose interactive behaviour is essentially the same as the original agent. Now we have the following.

Proposition 4.3 $P \succ Q \ \Rightarrow\ P =_{cc} Q$.

PROOF. We construct, for each n, the symmetric closure of $\succ_n$, $\prec\succ_n$, and show that it is sound by rule induction on the formation of $\prec\succ_n$ (including the rules for congruence), assuming $\succ_n \subseteq =_{cc}$ for the case of $\prec\succ_{n+1}$. The proof is tedious but elementary. See [15]. □

Note that $\succ$ is much stronger (more restrictive) than $=_{cc}$, or stronger than any "weak" canonical equalities we may apply to the present formalism. The main theorem follows.

Theorem 4.4 (Reduction Theorem) *For any a, v, x, and P, we have:*

$$a^*x.P, \mathcal{M}(av) \longrightarrow\!\succ P\{v/x\}.$$

PROOF. By induction on the rules given in Figure 2, inspecting each case. Note that we can use induction on the size of cc-terms for rules (I) to (VIII), while, for the remaining rules, we safely hypothesize on decomposed abstractions and reason about the whole expression.

Case (I). Let $\lambda x.P \odot v \stackrel{\mathrm{def}}{=} c\blacktriangleright(c^*x.P, \mathcal{M}(cv))$ with c fresh. Then:

$$a^*x.(P,Q), \mathcal{M}(av) \longrightarrow \lambda x.P \odot v,\ \lambda x.Q \odot v \twoheadrightarrow_\beta P', Q'$$

but by inductive hypothesis, $P' \succ P\{v/x\}$ and $Q' \succ Q\{v/x\}$. With transitivity and compatibility of $\succ$, $P', Q' \succ (P,Q)\{v/x\}$, as required.

Case (II). Easy by inductive hypothesis on $P\{c'/c\}$.

Case (III,VI,VII,VIII). Straightforward.

Case (IV). Note that $\mathcal{C}(v^+\tilde{v})$ is a message. Then: $\mathcal{M}(av'), a^*x.\mathcal{C}(v^+\tilde{w}) \longrightarrow c\blacktriangleright(\mathcal{FW}(cv), \mathcal{C}(c^+\tilde{w})) \rightarrow_\beta \mathcal{C}(v\tilde{w})$, as required.

Case (V). We first show, with $c \notin \{e, \tilde{w}\}$,

$$c\blacktriangleright(\mathcal{FW}(e^-c^+), \mathcal{C}(c^-\tilde{w})) \succ \mathcal{C}(e^-\tilde{w}) \qquad (1)$$

Then we have $\mathcal{M}(av'), a^*x.\mathcal{C}(v^-\tilde{w}) \longrightarrow c\blacktriangleright(\mathcal{FW}(vc), \mathcal{C}(c^-\tilde{w})) \succ \mathcal{C}(v\tilde{w})$.

For the remaining cases, we use the following claim.

CLAIM. *If $c^*x.\mathcal{C}_i(\tilde{w}_i), \mathcal{M}(cv) \longrightarrow\!\succ \mathcal{C}_i(\tilde{w}_i)\{v/x\}$ with $1 \leq i \leq n$ and $n \geq 2$, and, moreover, $\tilde{c}\blacktriangleright\prod_{i=1}^n \mathcal{C}_i(\tilde{w}_i) \succ \mathcal{C}'(\tilde{w}')$, then, for any u and x, we have*

$$\tilde{c}\blacktriangleright u^*x.\prod_{i=1}^n \mathcal{C}_i(\tilde{w}_i), \mathcal{M}(uv) \longrightarrow\!\succ \mathcal{C}'(\tilde{w}')\{v/x\}.$$

This is easily established in the same way as we did for (I) and (II) above. Now the remaining rules are inspected.

Case (IX). It is clear that the symmetric property of (1) holds, i.e. with $c \notin \{e, \tilde{w}, \tilde{v}\}$, we have:

$$c\blacktriangleright(\mathcal{FW}(c^-e^+), \mathcal{C}(\tilde{v}c^+\tilde{w})) \succ \mathcal{C}(\tilde{v}e^+\tilde{w}) \qquad (2)$$

Then this case is obvious from the Claim.

Case (X). Use (1) together with the Claim.

Case (XII). By Claim, $a^*x.\mathcal{S}(v_1xv_2),\ \mathcal{M}(aw)\ \longrightarrow\!\succ\ c_1c_2 \blacktriangleright(\mathcal{S}(v_1c_1c_2),\ \mathcal{M}(c_1w),\ \mathcal{B}_l(c_2v_2'))\ \stackrel{\text{def}}{=}\ \mathcal{S}'(v_1wv_2')$ with $v_2' \stackrel{\text{def}}{=} v_2\{w/x\}$. But $(\mathcal{S}'(v_1wv_2'), \mathcal{M}(v_1w'))\ \longrightarrow\ c_1c_2 \blacktriangleright(\mathcal{FW}(c_1c_2),\ \mathcal{M}(c_1w),\ \mathcal{B}_l(c_2v_2'))\ \twoheadrightarrow_\beta\ \mathcal{FW}(wv_2')$, as required.

Case (XI). We have $a^*x.\mathcal{B}_r(vx), \mathcal{M}(aw)\ \longrightarrow\!\succ\ c_1c_2c_3 \blacktriangleright(\mathcal{D}(vc_1c_2), \mathcal{S}(c_1wc_3), \mathcal{B}_r(c_2c_3))$ by the Claim and (XII). Let the resulting term be denoted by $\mathcal{B}_r'(vw)$. Then: $\mathcal{B}_r'(vw), \mathcal{M}(vw')\ \longrightarrow\twoheadrightarrow_\beta\ c_3 \blacktriangleright(\mathcal{FW}(c_3w'), \mathcal{FW}(wc_3)) \succ \mathcal{FW}(ww')$ by (2), hence the result. □

This concludes the validation of behaviour of $a^*x.P$.

4.3 π-Prefix

In the following we show that we can easily generalise the above results to the "synchronous" prefix in the line of the polyadic π-calculus [21] (thus subsuming monadic communication), using the notation given in 3.2.

Proposition 4.5 *We define*

$$a^*\!:(x_1...x_n).P \stackrel{\text{def}}{=} c_1..c_{n+1} \blacktriangleright(\rightharpoonup a^*\!: c_1..c_{n+1},\ c_1^*x_1.c_2^*x_2....c_{n+1}^*x_{n+1}.(\mathcal{M}(x_{n+1}),\ P))$$

$$\bar{a}^*\!:[v_1v_2..v_n].P \stackrel{\text{def}}{=} c \blacktriangleright(\leftharpoondown a^*\!: v_1v_2..v_nc,\ c^*.P)$$

with $c, c_1, ..., c_{n+1}$ fresh and distinct from each other, and $\mathcal{M}(y)$ and $c^*.P$ understood as in ν-calculus. Then $a^*\!:(x_1...x_n).P$ is a^--pointed and $\bar{a}^*\!:[v_1v_2..v_n].P$ is a^+-pointed. Moreover:

$$a^*\!:(x_1x_2..x_n).P,\ \bar{a}^*\!:[v_1v_2..v_n].Q \longrightarrow\succ\ P\{v_1v_2..v_n/x_1x_2..x_n\},\ Q$$

Proof. Pointedness is straightforward. The correctness of reduction is verified by using the property of reduction between $\leftharpoondown a^*\!: v_1v_2..v_n$ and $\rightharpoonup a^*\!: u_1u_2..u_n$ mentioned at the end of 3.3, together with Theorem 4.4. □

Thus our combinators are closed under those prefixes which we usually associate with processes. Our [14] further extends the notion of prefix to embody the complex operational structures including branching, which can also be treated in the same way. See [15] for details.

5 Correspondence between Two Systems

This section establishes the basic correspondence between ν-calculus and concurrent combinators. By the result, we know, modulo equalities $=_{cc}$ and $=_\nu$, two systems are equivalent. We also mention the embedding result of the finite π-calculus in concurrent combinators at the end. To avoid ambiguity, here and henceforth we often write $\rightarrow_{cc}$ (resp. $\rightarrow_\nu$) and $\equiv_{cc}$ (resp. $\equiv_\nu$) to denote the reduction and the structural congruence over $\mathbf{P}_{cc}$ (resp. $\mathbf{P}_\nu$).

Translations between the two systems are given below. The map from cc-terms to ν-terms is quite straightforward. The reverse translation uses the mapping of $u^*x.P$ we gave in the previous section.

Definition 5.1 (translation between ν-terms and combinators) *We define two functions, $[\![\cdot]\!]_\nu : \mathbf{P}_{cc} \rightarrow \mathbf{P}_\nu$, and $[\![\cdot]\!]_{cc} : \mathbf{P}_\nu \rightarrow \mathbf{P}_{cc}$, in Figure 3.*

Note that both maps are injections. The syntactic correspondence follows.

Proposition 5.2 (syntactic correspondence of translations) *In the following, $[\![P]\!]$ denotes either $[\![P]\!]_\nu$ or $[\![P]\!]_{cc}$, and relational and other symbols are interpreted accordingly.*

(i) $\mathcal{FN}(P) = \mathcal{FN}([\![P]\!])$, $\mathcal{FV}(P) = \mathcal{FV}([\![P]\!])$, $\mathcal{AN}_\theta(P) = \mathcal{AN}_\theta([\![P]\!])$.

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

(iii) $P \equiv_{cc} Q \Rightarrow [\![P]\!]_\nu \equiv_\nu [\![Q]\!]_\nu$, *and* $P \equiv_\nu Q \Rightarrow [\![P]\!]_{cc} =_{cc} [\![Q]\!]_{cc}$.

Proof. For (i) and (ii) we use induction on the structure of terms. For (iii), we use rule induction on derivations of $\equiv$. □

We need $=_{cc}$ for the second half of (iii) in Proposition 5.2 because we have $P \equiv_\nu Q \Rightarrow ax.P \equiv_\nu ax.Q$ (see footnote 8). The next result gives us the correspondence in terms of the reduction relations.

Proposition 5.3 *(operational correspondence)*

(i) $P \rightarrow_{cc} Q \Rightarrow [\![P]\!]_\nu \rightarrow_\nu [\![Q]\!]_\nu$, *and* $[\![P]\!]_\nu \rightarrow_\nu R \Rightarrow R \equiv_\nu [\![Q]\!]_\nu$ with $P \rightarrow_{cc} Q$.

(ii) $P \rightarrow_\nu Q \Rightarrow \exists\, Q'.(Q \equiv_\nu Q' \wedge [\![P]\!]_{cc} \rightarrow_{cc}\succ [\![Q']\!]_{cc})$, *and* $[\![P]\!]_{cc} \rightarrow_{cc} R \Rightarrow \exists\, Q.\ (R \succ [\![Q]\!]_{cc}$ with $P \rightarrow_\nu Q)$.

Proof. (i) is mechanical. For the first half of (ii), we use Theorem 4.4 and the fact: $ax.P, \leftharpoondown ab \rightarrow_\nu Q \Rightarrow Q \equiv_\nu P\{b/x\}$. For the second half, one uses Theorem 4.4 and Proposition 4.2. □

From this, together with Proposition 5.2, we obtain the correspondence in terms of the *action predicate*.

(1) Translation from $\mathbf{P}_{cc}$ to $\mathbf{P}_\nu$.

$$\begin{array}{rclrclrcl}
[\![\mathcal{M}(uv)]\!]_\nu & \stackrel{\text{def}}{=} & \leftarrow uv & [\![\mathcal{D}(uvw)]\!]_\nu & \stackrel{\text{def}}{=} & ux.(\leftarrow vx,\ \leftarrow wx) & [\![\mathcal{S}(uvw)]\!]_\nu & \stackrel{\text{def}}{=} & ux.vy.\leftarrow wy \\
[\![\mathcal{FW}(uv)]\!]_\nu & \stackrel{\text{def}}{=} & ux.\leftarrow vx & [\![\mathcal{B}_r(uv)]\!]_\nu & \stackrel{\text{def}}{=} & ux.vy.\leftarrow xy & [\![\mathcal{B}_l(uv)]\!]_\nu & \stackrel{\text{def}}{=} & ux.xy.\leftarrow vy \\
[\![\mathcal{K}(u)]\!]_\nu & \stackrel{\text{def}}{=} & ux.\Lambda & & & & & & \\
[\![P,Q]\!]_\nu & \stackrel{\text{def}}{=} & [\![P]\!]_\nu, [\![Q]\!]_\nu & [\![a \blacktriangleright P]\!]_\nu & \stackrel{\text{def}}{=} & a \blacktriangleright [\![P]\!]_\nu & [\![\Lambda]\!]_\nu & \stackrel{\text{def}}{=} & \Lambda
\end{array}$$

(2) Translation from $\mathbf{P}_\nu$ to $\mathbf{P}_{cc}$.

$$[\![\leftarrow uv]\!]_{cc} \stackrel{\text{def}}{=} \mathcal{M}(uv) \quad [\![ux.Q]\!]_{cc} \stackrel{\text{def}}{=} u^*x.[\![Q]\!]_{cc} \quad [\![P,Q]\!]_{cc} \stackrel{\text{def}}{=} [\![P]\!]_{cc}, [\![Q]\!]_{cc} \quad [\![a \blacktriangleright P]\!]_{cc} \stackrel{\text{def}}{=} a \blacktriangleright [\![P]\!]_{cc} \quad [\![\Lambda]\!]_{cc} \stackrel{\text{def}}{=} \Lambda$$

Figure 3: Translation between $\mathbf{P}_{cc}$ and $\mathbf{P}_\nu$

Proposition 5.4

(i) $P \Downarrow_{a^\theta} \;\Leftrightarrow\; [\![P]\!]_\nu \Downarrow_{a^\theta}$ *for* $P \in \mathbf{P}_{cc}$.

(ii) $P \Downarrow_{a^\theta} \;\Leftrightarrow\; [\![P]\!]_{cc} \Downarrow_{a^\theta}$ *for* $P \in \mathbf{P}_\nu$.

To study equational properties of the translations, the following property of $=_{cc}$ is fundamental.[8]

Proposition 5.5 $\quad P =_{cc} Q \;\Rightarrow\; a^*x.P =_{cc} a^*x.Q.$

The property is established by constructing a congruence which includes both $=_{cc}$ and the above equations, and showing that it is reduction-closed and respects the action predicate. See [15] for details.

Now we can show that the translations are equationally "adequate", in the following sense.

Theorem 5.6

(i) $P =_{cc} Q \;\Leftarrow\; [\![P]\!]_\nu =_\nu [\![Q]\!]_\nu$ *for* $P, Q \in \mathbf{P}_{cc}$.

(ii) $P =_\nu Q \;\Leftarrow\; [\![P]\!]_{cc} =_{cc} [\![Q]\!]_{cc}$ *for* $P, Q \in \mathbf{P}_\nu$.

Proof. (i) is obtained by constructing a relation $\cong$ such that $P \cong Q$ iff $[\![P]\!]_\nu =_\nu [\![Q]\!]_\nu$, and showing that it is congruent, reduction-closed, and respects the predicate $\Downarrow_{a^\theta}$, each of which is easily inferred from Propositions 5.2, 5.3, and 5.4. The proof of (ii) is similar, using Proposition 5.5 to verify congruence. □

Based on the result, we are now ready to state the equivalence theorem for two systems (cf. Theorem 7.3.10 in [2]).

Theorem 5.7 (equivalence between two systems modulo equality)

(i) $[\![P]\!]_{\nu,cc} =_{cc} P$.

(ii) $[\![P]\!]_{cc,\nu} =_\nu P$.

(iii) $P =_{cc} Q \;\Leftrightarrow\; [\![P]\!]_\nu =_\nu [\![Q]\!]_\nu$.

(iv) $P =_\nu Q \;\Leftrightarrow\; [\![P]\!]_{cc} =_{cc} [\![Q]\!]_{cc}$.

Proof. (outline) For (i) and (ii), we prove, by induction on the structure of terms, the corresponding stronger statements; $[\![P]\!]_{\nu,cc} \succ P$ and $[\![P]\!]_{cc,\nu} \succ' P$ where $\succ'$ is defined for $\mathbf{P}_\nu$ just as $\succ$, regarding messages and receptors as atomic agents. The "only if" direction of (iii) follows from Theorem 5.6 (ii) together with (i) above, in the way:

$$\begin{array}{lll}
P =_{cc} Q & \Rightarrow & [\![P]\!]_{\nu,cc} =_{cc} P =_{cc} Q =_{cc} [\![Q]\!]_{\nu,cc} \text{ (i)} \\
 & \Rightarrow & [\![P]\!]_\nu =_\nu [\![Q]\!]_\nu \qquad \text{(Theorem 5.6 ii)}
\end{array}$$

Similarly for (iv), using (ii) above and (i) of Theorem 5.6 instead. □

We finally refer to the result concerning the translation of a system of monadic synchronous communication, which is essentially the monadic π-calculus given in [20], but without replication, into our combinators. The set of terms of the calculus, denoted by $\mathbf{P}_\pi$, is given by:

$$P \;::=\; \bar{u}v.P \;\mid\; ux.P \;\mid\; P,Q \;\mid\; a \blacktriangleright P \;\mid\; \Lambda$$

Translation from cc-terms to π-terms is equally straightforward as $[\![\cdot]\!]_\nu$ except that we use $\bar{u}v.\Lambda$ instead of $\leftarrow uv$. The reverse translation is given using the mapping in Proposition 4.5, in exactly the same way as $[\![\ ]\!]_{cc}$. If we write the maximum sound congruence in the π-calculus $=_\pi$, the similar path as we had before lets us prove:

$$P =_{cc} Q \quad\Leftrightarrow\quad \langle P\rangle_\pi =_\pi \langle Q\rangle_\pi$$

[8] Contrary to possible expectations, $\equiv$ is not closed in the way, e.g. $(P,\ \Lambda) \equiv P$ but $a^*x.(P,\Lambda) \not\equiv a^*x.P$. For further discussions, see [15].

where $\langle \cdot \rangle_\pi$ is the mapping from $\mathbf{P}_{cc}$ to $\mathbf{P}_\pi$. Similarly, if we write the translation from π-terms to concurrent combinators $\langle\ \rangle_{cc}$, we get:

$$P =_\pi Q \quad \Leftarrow \quad \langle P \rangle_{cc} =_{cc} \langle Q \rangle_{cc}$$

Interestingly, the converse of the above statement does not hold, e.g. $ax.ax.\Lambda =_\pi ax.\Lambda,\ ax.\Lambda$ but $\langle ax.ax.\Lambda \rangle_{cc} \neq_{cc} \langle ax.\Lambda,\ ax.\Lambda \rangle_{cc}$. This is essentially because terms with "illegal" protocols can discriminate finer differences than those in the original world (cf. [20]). An interesting open question is whether the addition of new atoms with new behaviours, e.g. "synchronous output," can solve the issue, while retaining the closure property under prefix, as we have in the present system.

6 Discussion

6.1 Lafont's construction

Lafont's Interaction Net [18] is a generalisation of Girard's proof nets [7]. It provides a graph-based formalism for a computation scheme similar to ours. While independent in inception, Lafont's construction can be considered to be a precursor to our work. Below we list some similarities and differences.

(1) The computational frameworks of both systems are quite similar. Lafont's *polarity* corresponds to our *polarity*, his *principal ports* correspond to our notion of *subjects*, and his *port connection* corresponds to *name connection with hiding* (so our operation is finer in this aspect). Specifically, "cut" becomes our forwarder. In fact there is a clean embedding of an important family of his systems in concurrent combinators with the replication mentioned in 6.3, for details of which the reader may refer to the authors' coming exposition. Such an embedding also suggests how our construction can be given a graph-based formulation.

(2) In Lafont's system, if agents are connected to one location, one positive agent and one negative agent always share it, and, after interaction occurs, the location is never used again (hence rewriting becomes confluent). Such a restriction would be useful for some purposes, but there are situations where we need more general descriptive power (cf. [17]). We also note that, as far as we know, there is no "calculus" counterpart to Lafont's system, and that, in this connection, no proposal for basic combinators is done in [18]. Incorporation of the typing framework developed in [18] into our formalism would be an interesting topic for further research.

6.2 Combinatory Logic

The classical combinators were born from logics, while the present construction is derived from the theoretical investigation of concurrency. Whether we can in fact call the present construction an analogue of the original one is a difficult question. It is thus instructive to pick out similarities and differences between the two notions.

(1) Both are based on decomposition of simple, yet powerful, reduction schemes in their respective settings, into finite dynamics. We expect, specifically in this aspect, that ours would have the same application to the pragmatics of computing as the sequential combinators, namely, the execution scheme of programming languages, but now in a concurrency setting.

(2) As we discussed in Section 3, one essential difference is found in the notion of *atoms*. Here atoms should be "connected" to locations to become active as agents. This seems inevitable, even if we do not use names to denote locations (cf. [18]), as far as we want to represent *multiple interaction points* and their *connection topology*, which we believe to be essential elements of concurrency.

(3) Our understanding of the present construction is still in its infancy, when compared to that of classical combinators. Further investigation is needed to clarify the basic constructs of the formalism, e.g. essential connectives (how about summation?), the necessity for polarities, the type-theoretic foundations, as well as the general semantic framework. These will provide important subjects for future study.

6.3 Replication

The present work only gives a combinatory representation of a finite part of ν-calculus. Though it is possible to add the construct $!P$ to the system to gain unbounded behaviour (and hence full computational power) with some addition to the mapping shown in Figure 2, it would be more satisfactory to obtain the same effect simply by adding finite atoms. Thus a natural next step is the representation of full ν-calculus by finite atoms, or, more concretely, for any combinator P, to represent $!a^*x.P$ such that

$$!a^*x.P, \leftarrow ab \ \longrightarrow_{\gtrsim}\ !a^*x.P, P\{b/x\}$$

by utilising a finite number of atoms and their combinations (the usage of a lazy replicator results in no loss of expressive power; $\gtrsim$ is a certain extension of $\succ$). Perhaps surprisingly, this is indeed possible, by adding a handful of atoms, which are systematically derived from the seven basic atoms. There are two essential points.

(1) Introduction of the *name generator*, which creates a new name each time it interacts.

(2) Introduction of *templates*, which make a new atomic agent (with different instantiations) each time it is requested.

Using these and other persistent atoms, a system with (at most) 28 atoms is closed under usual and replicated pointed name abstraction. The representation has a non-trivial consequence in that, by this, we gain the representability of not only the full ν-calculus and Milner's π-calculus with replication in [20], but also *any* (untyped) systems of concurrent combinators with finite atoms. Full explication of the topic is left to [14].

Acknowledgements

The authors wish to thank Makoto Kubo for his suggestions and advice. They also thank Professor Mario Tokoro for his encouragement and support.

References

[1] Abramsky, S., The Lazy Lambda Calculus. D. Turner, editor, *Research topics in Functional Programming*, Addison Wesley, 1990.

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

[3] Berry, G. and Boudol, G., The Chemical Abstract Machine. *Theoretical Computer Science*, 96, pp.217–248, 1992.

[4] Boudol, G., *Asynchrony and π-calculus*. INRIA Report 1702, INRIA, Sophia Antipolis, 1991.

[5] Curry, H. B. and Feys, R., *Combinatory Logic, Vol. I.* North Holland, 1958.

[6] Engberg, U. and Nielsen, M., *A Calculus of Communicating Systems with Label Passing.* Research Report DAIMI PB-208, Computer Science Department, University of Aarhus, 1986.

[7] Girard, J.-Y., Linear Logic. *Theoretical Computer Science*, 50, pp.1–102, 1987.

[8] Hewitt, C., Bishop, P., and Steiger, R., A Universal Modular ACTOR Formalism for Artificial Intelligence. *Proc. of the 3rd International Joint Conference on Artificial Intelligence*, pp.235–245, 1973.

[9] Hindley, J. R. and Seldin J. P., *Introduction to Combinators and λ-Calculus.* Cambridge University Press, 1986.

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

[11] Honda, K., *Two bisimilarities in ν-calculus.* Keio CS report 92-002, Department of Computer Science, Keio University, 1992.

[12] Honda, K., Types for Dyadic Interaction. *CONCUR'93*, LNCS 612, pp.504–523, Springer-Verlag, April 1992.

[13] Honda, K. and Yoshida, N., On Reduction-Based Process Semantics. *FSTTCS'13*, LNCS 761, Springer-Verlag, 1993.

[14] Honda, K. and Yoshida, N., *Replication in Concurrent Combinators.* To appear in *TACS'94*, LNCS, Springer-Verlag, 1994.

[15] Honda, K. and Yoshida, N., *Combinatory Representation of Mobile Processes: Part I.* The full version of this paper. Under preparation.

[16] Jones, C.B., Constraining interference in an object-based design method. *TAPSOFT'93*, LNCS 668, pp.136–150, Springer Verlag, 1993.

[17] Jones, C.B., *Process-Algebraic Foundations for an Object-Based Design Notation.* UMCS-93-10-1, Computer Science Department, Manchester University, 1993.

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

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

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

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

[22] Milner, R. and Sangiorgi, D., Barbed Bisimulation. *ICALP'92*, LNCS 623, pp.685–695, Springer-Verlag, 1992.

[23] Schönfinkel, M., Über die Bausteine der mathematischen Logik. *Math. Annalen* 92, pp.305–316, 1924. Engl. trans. in J. van Heijenoort, editor, *From Frege to Gödel*, pp.355–366, Harvard Univ. Press, 1967.