A Concurrent System of Multi-ported Processes with Causal Dependency

Tatsuya Abe

December 18, 2004

Abstract

The pi-calculus is a concurrent system invented by Milner et al. in which concurrent computation is expressed as interaction of processes through name-passing. Building on the concept of name-passing, we propose a new concurrent system based on multi-ports, whereas the pi-calculus is based on single-ports. Although our trial is not the first one in constructing a concurrent system based on multi-ports, ours is unique in that it is only extended in terms of multi-ports. This simplicity makes it possible to control self-communication of processes. Besides, it is an extension of the pi-calculus, because a single-port can be expressed as a restriction in our system. These suggest that the concept of multi-ports is natural. Furthermore, it is more expressive than other calculi including the original pi-calculus in spite of this simplicity. Even the strong call-by-name lambda-calculus can be embedded into our system with respect to convergence and divergence, while it has not been successfully done into the original pi-calculus.

1 Introduction

The pi-calculus was invented by Milner et al. as a model for expressing concurrent computation [16]. It has been studied actively in the field of concurrency due to its simplicity. As a result, it has turned out that it can be extended in various directions and that it is far more expressive than initially considered. In this paper we propose a new concurrent system, which is built on the notion of name-passing as in the original pi-calculus. However, it is essentially different from previous approaches for extending the pi-calculus. We claim that our system is more fundamental than the original one. The reason is that it is not only an extension of the original pi-calculus but it can express the original one as *syntax sugar*. This is different from other approaches, which tend to complicate the original pi-calculus by adding extra rules.

The most distinguishing feature of our system is that it is based on the concept of *multi-ports*. To make this point clear, we explain computation of the original pi-calculus. In the original pi-calculus, a process transits to another by consuming a prefix of the process. For instance,

$$\overline{x}y.P \xrightarrow{\overline{x}y} P \ .$$

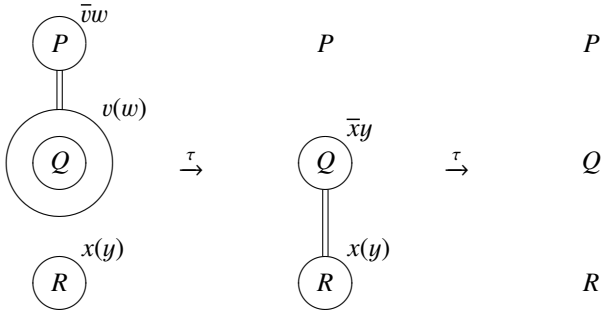
Informally, $\overline{x}y.P$ means "send the name y along the link named x, and then enact P". Here, $\overline{x}y.P$ is said to be an *output process*, and $\overline{x}y$ an *output action*. Similarly, $x(y).P$ means "receive some name z along the link named x, and then enact $[z/y]P$". Here, $[z/y]P$ denotes the result of replacing occurrences of y with z in P. We call $x(y).P$ an *input process*, and $x(y)$ an *input action*.

Computation of the original pi-calculus is realized as *interaction*. An output process and an input process which possess the same link name interact. The rule which expresses interaction is as follows:

$$\frac{P \xrightarrow{\overline{x}z} P' \qquad Q \xrightarrow{x(y)} Q'}{P|Q \xrightarrow{\tau} P'|[z/y]Q'} \quad .$$

Here, $P|Q$ stands for the parallel composition of two processes P and Q. The process $P|Q$ can transit as P or Q, and when the transition occurs in P, it is independent of Q, and vice versa. Moreover, interaction between P and Q can occur when one is an output process and the other is an input process which possess the same link name. In the case of internal interaction, we use $\xrightarrow{\tau}$, where τ means an internal action. In the above rule, P can transit to P' sending the message z along the link named x, and Q can transit to $[z/y]Q'$ receiving the message z along the link named x. Therefore, interaction can occur in $P|Q$, and $P|Q$ itself can transit to $P'|[z/y]Q'$.

Let us now go back to the input processes, in which our system essentially differs from the original pi-calculus. Milner explained the meaning of $x(y).P$ as "receive any name along the link named x, and then enact $[z/y]P$" [14]. However, one may observe that another meaning is implicitly attached to it. That is temporal precedence. In addition to Milner's explanation, it is implicitly assumed that the $x(y).P$ *does nothing* until it receives something along the link named x. In other words, in the original pi-calculus a prefix of a process can be interpreted as *capsuling* the process inside as follows:



where double lines denote interactions. The interaction along the link named v strips the outermost layer $v(w)$ of the process $v(w).\overline{x}y.Q$, and the layer $\overline{x}y$ appears. Once $\overline{x}y$ becomes observable, an interaction occurs between $\overline{x}y.Q$ and $x(y).R$. Syntactically,

$$\overline{v}w.P|v(w).\overline{x}y.Q|x(y).R \xrightarrow{\tau} P|\overline{x}y.Q|x(y).R \xrightarrow{\tau} P|Q|R \quad .$$

We claim that this idea restricts behavior of processes and interaction between processes. In the above example, the interaction along the link named x does not affect the interaction along the link named v if w is neither x nor y. There exists no a priori reason why we have to impose the restriction that the interaction of v should be ahead of that of x. However, there exist indeed some occasions where we *have to* impose an ordering on prefixes. A typical situation is when a prefix binds a name occurring in some succeeding prefixes: for instance, consider the process $x(y).\overline{y}z.0$ where the first prefix binds a name in the second prefix. In this case, it is impossible to have interaction for the second prefix first. This kind of precedence is essential.

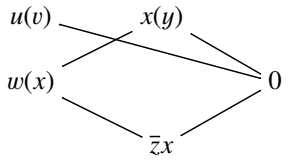
We improve this point and design a new concurrent system, where the temporal precedence in the original pi-calculus is replaced by causal dependency. In our system, interaction does

not depend on the order of prefixes; interaction may occur as long as the causality condition is satisfied. Interpreting a prefix of a process as a port along which it can interact, we regard the original pi-calculus as a concurrent system in which a process has *only one* port. On the other hand, our system allows a process to have several ports, each of which can interact. In this sense, processes in our system can be seen as multi-ported.

Let us now illustrate causality of prefixes visually. For instance, consider the process

$$u(v):w(x):x(y):\bar{z}x.0$$

in our system, where the names u, w, z, v, and x are distinct from one another. We use *colon* notation $x(y):P$ as an input prefix instead of *dot* notation $x(y).P$ to stress that our input prefix is different from that of Milner et al. It incorporates the capsuling facility intrinsically. The following illustrates the causality relation among the prefixes of this process, where causality of prefixes is denoted by edges.



As a matter of fact, both $u(v)$ and $w(x)$ are considered to be outermost prefixes while $u(v)$ is the only prefix in its counterpart of the original pi-calculus: $u(v).w(x).x(y).\bar{z}x.0$. Returning to the process $\bar{v}w.P|v(w):\bar{x}y.Q|x(y):R$, we can see that an interaction along the port x can precede one along the port v while capsuling interpretation cannot be realized.

Causal dependency might be reminiscent of the notion of *delayed input* [23, 12], which is different from ours in terms of self-communication. For instance, the delayed input process $x(y):\bar{x}z.P$ can transit to $[z/y]P$ while it cannot in our system. However, we can express any delayed input process, i.e., a process which can perform self-communication in different way as described in Section 3. As a result, our system has the advantage that we can control permission of self-communication.

In this paper, we also show that our system is more expressive than the original one. Although it is known that the call-by-name, the call-by-value, and the strong call-by-value lambda-calculi[1] can be embedded into the original pi-calculus [14], we prove the fact that not only those but also the strong call-by-name lambda-calculus can be embedded into our system naturally. In particular, it is remarkable that our embedding of the strong call-by-name lambda-calculus is complete with respect to convergence and divergence.

We use the following notation throughout this paper, for convenience. For any relation $\mathcal{R}$, the relations $\mathcal{R}^*$, $\mathcal{R}^?$, and $\mathcal{R}^+$ denote the reflexive and transitive closure, the reflexive closure, and the transitive closure of $\mathcal{R}$, respectively. For any relations $\mathcal{R}$ and $\mathcal{S}$, the relation $\mathcal{R} \cdot \mathcal{S}$ is defined as follows:

$$\{\, (P, R) \mid \text{there exists } Q \text{ such that } (P, Q) \in \mathcal{R} \text{ and } (Q, R) \in \mathcal{S} \,\} \ .$$

Furthermore, we often use an infix notation for short, e.g., $P \to P'$ denotes $(P, P') \in \to$.

This is a revised and expanded version of the work presented at the 2nd Asian Symposium on Programming Languages and Systems [2].

[1] Called the *weak* call-by-value lambda-calculus in the paper [14], in which evaluation strategy is more loose than that of the call-by-value lambda-calculus.

2 Syntax and Semantics

In this section, we introduce a concurrent system called Mpi-calculus. The Mpi-calculus is a modification of the pi-calculus, which is meant to capture the notion of multi-port. The main difference from the pi-calculus is that single-ported input prefixes of the pi-calculus are replaced by the multi-ported ones.

The Mpi-calculus is a concurrent system where processes interact through passing names just as the pi-calculus. We presuppose an infinite set of *names*, and that $x, y, \ldots$ range over it. The Mpi-calculus consists of a set of processes. The set of processes is defined as follows:

$$P, Q, \ldots ::= 0 \mid \tau.P \mid \overline{x}y.P \mid x(y):P \mid P|Q \mid (\nu x)P \mid !P$$

where τ is a special constant which is distinct from the others. Unary connectives bind more strongly than binary ones.

A process transits to another by consuming a prefix of the process. For instance,

$$\overline{x}y.P \xrightarrow{\overline{x}y} P \ .$$

Let us regard $\overline{x}y.P$ as a process which sends the name y along the link named x, and then enact P. The process $\overline{x}y.P$ is said to be an *output process*, and $\overline{x}y$ an *output prefix* of the output process. Symmetrically, let us regard $x(y):P$ as a process which receives some name z along the link named x, and then enact $[z/y]P$. Here, $[z/y]P$ denotes the result of replacing occurrences of y with z in P. We give the formal definition after we define substitution of names. We call $x(y):P$ an *input process*, and $x(y)$ an *input prefix* of the input process. Since the occurrences of y in P of $x(y):P$ only point at the locations into which a name received along x passes, they are defined to be bound in $x(y):P$.

The connective | composes two processes in parallel in the Mpi-calculus. Two processes connected by the parallel composition connective transit independently, and can interact if one is an output process and the other is an input process which possess the same link name.

We explain the other connectives below. The process $\tau.P$ transits to P by performing an unobservable action τ. The process $(\nu x)P$ is a process P whose port (or message) x is closed (or localized). Since this name x only points at the port (or message) which should be closed (or localized), the xs in P are bound just as in an input process. The process $!P$ is a process which is allowed to be copied arbitrarily many times.

The set of *actions* is defined as follows:

$$\alpha ::= \tau \mid \overline{x}y \mid \overline{x}(y) \mid x(y) \ .$$

We define *free names*, and *bound names of an action*:

$$\begin{array}{llll} \mathrm{fn}\,\tau = \emptyset & \mathrm{fn}\,\overline{x}y = \{x, y\} & \mathrm{fn}\,\overline{x}(y) = \{x\} & \mathrm{fn}\,x(y) = \{x\} \\ \mathrm{bn}\,\tau = \emptyset & \mathrm{bn}\,\overline{x}y = \emptyset & \mathrm{bn}\,\overline{x}(y) = \{y\} & \mathrm{bn}\,x(y) = \{y\} \ . \end{array}$$

The set $\mathrm{n}\,\alpha$ of names, called *names of* α, is defined as $\mathrm{fn}\,\alpha \cup \mathrm{bn}\,\alpha$. The set $\mathrm{fn}\,P$ consists of free names of P out of the scopes of input prefixes and restriction connectives. A function σ from names to names is said to be a substitution if its support, written $\mathrm{supp}\,\sigma = \{\, x \mid \sigma x \neq x \,\}$, is finite. The cosupport of a substitution, written $\mathrm{cosupp}\,\sigma$, is defined as the image of its support. The set $\mathrm{n}\,\sigma$ of names is $\mathrm{supp}\,\sigma \cup \mathrm{cosupp}\,\sigma$. Naturally, any substitution can be extended on an action. Formally,

$$\sigma(\tau) = \tau \qquad \sigma(\overline{x}y) = \overline{\sigma x}\sigma y \qquad \sigma(\overline{x}(y)) = \overline{\sigma x}(y) \qquad \sigma(x(y)) = \sigma x(y) \ .$$

In a similar way, a substitution replaces *free* occurrences in a process such that a new binding does not occur. We define a relation on processes, written $\xrightarrow{\alpha}$, which we consider to be a transition. For instance, $P \xrightarrow{\alpha} P'$ denotes that P transits to P' by invoking an action α. We define this by the notion of inference. Transition rules are as follows:

$$\tau.P \xrightarrow{\tau} P \qquad \overline{x}y.P \xrightarrow{\overline{x}y} P$$

$$\frac{P \xrightarrow{\alpha} P' \qquad x \notin \mathrm{bn}\,\alpha \qquad y \notin \mathrm{n}\,\alpha}{x(y):P \xrightarrow{\alpha} x(y):P'} \qquad \frac{z \notin (\mathrm{fn}\,P \setminus \{y\}) \cup \{x\}}{x(y):P \xrightarrow{x(z)} [z/y]P}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad x \notin \mathrm{n}\,\alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'} \qquad \frac{P \xrightarrow{\overline{x}y} P' \qquad x \neq y \qquad z \notin \mathrm{fn}\,P \setminus \{y\}}{(\nu y)P \xrightarrow{\overline{x}(z)} [z/y]P'}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad \mathrm{bn}\,\alpha \cap \mathrm{fn}\,Q = \emptyset}{P|Q \xrightarrow{\alpha} P'|Q}$$

$$\frac{P \xrightarrow{\overline{x}z} P' \qquad Q \xrightarrow{x(y)} Q'}{P|Q \xrightarrow{\tau} P'|[z/y]Q'} \qquad \frac{P \xrightarrow{\overline{x}(y)} P' \qquad Q \xrightarrow{x(y)} Q'}{P|Q \xrightarrow{\tau} (\nu y)(P'|Q')}$$

$$\frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'|!P}$$

$$\frac{P \xrightarrow{\overline{x}z} P' \qquad P \xrightarrow{x(y)} P''}{!P \xrightarrow{\tau} P'|[z/y]P''|!P} \qquad \frac{P \xrightarrow{\overline{x}(y)} P' \qquad P \xrightarrow{x(y)} P''}{!P \xrightarrow{\tau} (\nu y)(P'|P'')|!P}$$

where symmetric and transitive ones are omitted. The annotation on a transition arrow is the action which is performed. The output action $\overline{x}(y)$ is invoked by the process P when y is bound in P. The notion of output action of bound name realizes a natural interaction such that:

$$(\nu y)\overline{x}y.P|x(y):Q \xrightarrow{\tau} (\nu y)(P|Q) \ .$$

Let us consider the process $v(w):x(y):\overline{w}y.0$ where the names are distinct from one another, in order to see that computation in our system differs from that of the original pi-calculus. The two prefixes $v(w)$ and $x(y)$ in this process are independent. This process receives a name along the link v as a name of its port and another name along the link x as a message along the link not named yet. It is, therefore, natural that the order of interactions does not depend on the denotation: $v(w):x(y):\overline{w}y.0$. Indeed, the deduction:

$$\frac{\dfrac{y \notin (\{w,y\} \setminus \{y\}) \cup \{x\}}{x(y):\overline{w}y.0 \xrightarrow{x(y)} \overline{w}y.0} \qquad v \neq y \qquad w \notin \{x,y\}}{v(w):x(y):\overline{w}y.0 \xrightarrow{x(y)} v(w):\overline{w}y.0}$$

ensures a transition on the basis of causal dependency.

Proposition 2.1. *Suppose that* $P \xrightarrow{\alpha} P'$.

1. $\mathrm{bn}\,\alpha \cap \mathrm{fn}\,P = \emptyset$ *and* $(\mathrm{fn}\,P' \setminus \mathrm{bn}\,\alpha) \cup \mathrm{fn}\,\alpha \subseteq \mathrm{fn}\,P$.

2. $\alpha = \overline{x}(y)$ *and* $z \notin \mathrm{n}\, P$ *imply* $P \xrightarrow{\overline{x}(z)} [z/y]P'$.

3. $\alpha = x(y)$ *and* $z \notin \mathrm{n}\, P$ *imply* $P \xrightarrow{x(z)} [z/y]P'$.

4. $\mathrm{bn}\,\alpha \cap (\mathrm{fn}\,\sigma P \cup \mathrm{n}\,\sigma) = \emptyset$ *implies* $\sigma P \xrightarrow{\sigma\alpha} \sigma P'$.

Proof. By induction on $P \xrightarrow{\alpha} P'$. □

We define an equational relation between processes that we would like to identify in the concurrent system. The set of *contexts* is defined as follows:

$$C ::= [\cdot] \mid \tau.C \mid \overline{x}y.C \mid x(y):C \mid C|P \mid P|C \mid (\nu x)C \mid\ !C$$

where $[\cdot]$ is a symbol denoting a hole of the process. When C is a context and P is a process, $C[P]$ denotes the result of replacing P with the hole $[\cdot]$ in C.

Definition 2.2. *A relation $\mathcal{R}$ on processes is called* compatible *if $(P, Q) \in \mathcal{R}$ implies $(C[P], C[Q]) \in \mathcal{R}$ for every context C. An equivalence relation $\mathcal{R}$ on processes is said to be a* congruence relation *if $\mathcal{R}$ is compatible.*

The structural congruence relation, written $\equiv$, is defined as the smallest congruence relation that satisfies

1. $P|0 \equiv P$, $P|Q \equiv Q|P$, $(P|Q)|R \equiv P|(Q|R)$,

2. $(\nu x)(P|Q) \equiv P|(\nu x)Q$ if $x \notin \mathrm{fn}\, P$, $(\nu x)0 \equiv 0$, $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$,

3. $!P \equiv P|!P$.

Namely, the set of processes is a monoid with an operation | and a unit 0. In addition, the order of closing ports are ignored, and closing a port is done only where the port exists. The rule 3 means that the process $!P$ is a process P which is allowed to be copied arbitrarily many times, as already described.

3 Conservativeness

In the previous section, we constructed a concurrent system based on the notion of causality. Recall that we have adopted unguarded input processes of the form $x(y):P$ instead of guarded processes of the form $x(y).P$. The Mpi-calculus does not have guarded input processes syntactically. However, guarded processes are expressed as macros of Mpi-processes as seen in this section. In this sense, the Mpi-calculus does not abandon temporal precedence.

First, we define a concurrent system (indeed, Milner's original pi-calculus) by replacing input processes and their two rules with the ones of the form $x(y).P$ and the following rule:

$$\frac{z \notin (\mathrm{fn}\, P \setminus \{y\}) \cup \{x\}}{x(y).P \xrightarrow{x(z)} [z/y]P} \quad .$$

Second, we define a translation $\langle\!|\cdot|\!\rangle$ from pi-processes into Mpi-processes. With respect to guarded processes, it is as follows:

$$\langle\!|x(y).P|\!\rangle = (\nu u)(x(y):\overline{u}y.\langle\!|P|\!\rangle | u(y):0)$$

where u is fresh. The rest is the same as the original pi-calculus, i.e.,

$$\langle\!|0|\!\rangle = 0 \qquad \langle\!|\tau.P|\!\rangle = \tau.\langle\!|P|\!\rangle \qquad \langle\!|\overline{x}y.P|\!\rangle = \overline{x}y.\langle\!|P|\!\rangle$$

$$\langle\!| P|Q |\!\rangle = \langle\!| P |\!\rangle | \langle\!| Q |\!\rangle \qquad \langle\!| (\nu x)P |\!\rangle = (\nu x)\langle\!| P |\!\rangle \qquad \langle\!| !P |\!\rangle = !\langle\!| P |\!\rangle \ .$$

Clearly, this translation preserves the free names of a process, and commutes with substitutions, that is, $\langle\!| \sigma P |\!\rangle$ and $\sigma\langle\!| P |\!\rangle$ are alpha-equivalent: they only differ from each other in the choice of bound names. We often write $P \equiv_\alpha Q$ when P and Q are alpha-equivalent. Third, the relation $\mapsto$ is defined as the reflexive and transitive closure of the smallest compatible relation that satisfies

1. $P|0 \mapsto P$, $0|P \mapsto P$,
2. $(\nu x)P \mapsto P$ if $x \notin \mathrm{fn}\, P$.

The relation $\mapsto$ is as trivial as contained properly by the structural congruence relation $\equiv$, and commutes internal actions, that is, $P \mapsto \cdot \xrightarrow{\tau} P'$ implies $P \xrightarrow{\tau} \cdot \mapsto P'$. Finally, we give some notion. It turns out that any internal transition $\xrightarrow{\tau}$ is produced by a primitive action τ or an interaction through a port. Now we annotate τ which is produced by an interaction through a port x as in $x\tau$. Moreover, $\langle\!| P |\!\rangle\tau$ is defined as the set of fresh names produced by $\langle\!| P |\!\rangle$. Sometimes, $\langle\!| P |\!\rangle\tau$ denotes some element of $\langle\!| P |\!\rangle\tau$ for short.

Theorem 3.1. *If $P \xrightarrow{\alpha} P'$, then $\langle\!| P |\!\rangle \xrightarrow{\alpha} \cdot \xrightarrow{\langle\!| P |\!\rangle\tau}{}^{?} \cdot \mapsto \langle\!| P' |\!\rangle$.*

Proof. By induction on $P \xrightarrow{\alpha} P'$ it suffices to prove:

1. If $P \xrightarrow{\overline{x}y} P'$, then $\langle\!| P |\!\rangle \xrightarrow{\overline{x}y} \langle\!| P' |\!\rangle$.
2. If $P \xrightarrow{\alpha} P'$ where α is $x(y)$ or τ, then $\langle\!| P |\!\rangle \xrightarrow{\alpha} \cdot \xrightarrow{\langle\!| P |\!\rangle\tau}{}^{?} \cdot \mapsto \langle\!| P' |\!\rangle$.

Consider the case that $x(y){:}P_1 \xrightarrow{x(z)} [z/y]P_1$ is inferred from $z \notin (\mathrm{fn}\, P_1 \setminus \{y\}) \cup \{x\}$. Then,

$$\dfrac{\dfrac{\dfrac{z \notin (\mathrm{fn}\, \overline{u}y.\langle\!| P_1 |\!\rangle \setminus \{y\}) \cup \{x\}}{x(y){:}\overline{u}y.\langle\!| P_1 |\!\rangle \xrightarrow{x(z)} \overline{u}z.[z/y]\langle\!| P_1 |\!\rangle} \quad z \notin \{u\}}{x(y){:}\overline{u}y.\langle\!| P_1 |\!\rangle | u(y){:}0 \xrightarrow{x(z)} \overline{u}z.[z/y]\langle\!| P_1 |\!\rangle | u(y){:}0} \quad u \notin \{x, y\}}{(\nu u)(x(y){:}\overline{u}y.\langle\!| P_1 |\!\rangle | u(y){:}0) \xrightarrow{x(z)} (\nu u)(\overline{u}z.[z/y]\langle\!| P_1 |\!\rangle | u(y){:}0)}$$

and

$$\dfrac{\dfrac{\overline{u}z.[z/y]\langle\!| P_1 |\!\rangle \xrightarrow{\overline{u}z} [z/y]\langle\!| P_1 |\!\rangle \qquad \dfrac{y \notin (\mathrm{fn}\, 0 \setminus \{y\}) \cup \{u\}}{u(y).0 \xrightarrow{u(y)} 0}}{\overline{u}z.[z/y]\langle\!| P_1 |\!\rangle | u(y){:}0 \xrightarrow{u\tau} [z/y]\langle\!| P_1 |\!\rangle | 0}}{(\nu u)(\overline{u}z.[z/y]\langle\!| P_1 |\!\rangle | u(y){:}0) \xrightarrow{u\tau} (\nu u)([z/y]\langle\!| P_1 |\!\rangle | 0)}$$

suffice to show the case since

$$(\nu u)([z/y]\langle\!| P_1 |\!\rangle | 0) \mapsto [z/y]\langle\!| P_1 |\!\rangle \equiv_\alpha \langle\!| [z/y]P_1 |\!\rangle \ .$$

Next, consider the case that $P_1|Q_1 \xrightarrow{\tau} P_1'|[z/y]Q_1'$ is inferred from $P_1 \xrightarrow{\overline{x}z} P_1'$ and $Q_1 \xrightarrow{x(y)} Q_1'$. By induction hypothesis, we have $\langle\!| P_1 |\!\rangle \xrightarrow{\overline{x}z} \langle\!| P_1' |\!\rangle$ and $\langle\!| Q_1 |\!\rangle \xrightarrow{x(y)} R \xrightarrow{\langle\!| Q_1 |\!\rangle\tau}{}^{?} S \mapsto \langle\!| Q_1' |\!\rangle$ for some R and S. We therefore have:

$$\dfrac{\langle\!| P_1 |\!\rangle \xrightarrow{\overline{x}z} \langle\!| P_1' |\!\rangle \qquad \langle\!| Q_1 |\!\rangle \xrightarrow{x(y)} R}{\langle\!| P_1 |\!\rangle | \langle\!| Q_1 |\!\rangle \xrightarrow{\tau} \langle\!| P_1' |\!\rangle | [z/y]R} \ .$$

By Proposition 2.1.4, we have $[z/y]R \xrightarrow{\langle\!|Q_1|\!\rangle\tau}{}^{?} [z/y]S$. Then,

$$\frac{[z/y]R \xrightarrow{\langle\!|Q_1|\!\rangle\tau}{}^{?} [z/y]S \qquad \emptyset \cap \mathrm{fn}[z/y]R = \emptyset}{\langle\!|P_1'|\!\rangle|[z/y]R \xrightarrow{\langle\!|Q_1|\!\rangle\tau}{}^{?} \langle\!|P_1'|\!\rangle|[z/y]S} \ .$$

Since $\mapsto$ is closed under substitutions, we have:

$$\langle\!|P_1'|\!\rangle|[z/y]S \mapsto \langle\!|P_1'|\!\rangle|[z/y]\langle\!|Q_1'|\!\rangle \equiv_\alpha \langle\!|P_1'|[z/y]\langle\!|Q_1'|\!\rangle\!\rangle \ .$$

The other cases are trivial or similar to the above case. □This claims that the Mpi-calculus is an extension of the original pi-calculus. Furthermore, we can know the extension to be conservative[2] by the following theorem.

Lemma 3.2. *If $\langle\!|P|\!\rangle \xrightarrow{\alpha} P'$, then there exists P'' such that $P \xrightarrow{\alpha} P''$ and $P' \xrightarrow{\langle\!|P|\!\rangle\tau}{}^{?} \cdot \mapsto \langle\!|P''|\!\rangle$.*

Proof. By induction on $\langle\!|P|\!\rangle \xrightarrow{\alpha} P'$, it is sufficient to prove:

1. If $\langle\!|P|\!\rangle \xrightarrow{\overline{x}y} P'$, then there exists P'' such that $P \xrightarrow{\overline{x}y} P''$ and $P' = \langle\!|P''|\!\rangle$.
2. If α is $x(y)$ or τ and $\langle\!|P|\!\rangle \xrightarrow{\alpha} P'$, then there exists P'' such that $P \xrightarrow{\alpha} P''$ and $P' \xrightarrow{\langle\!|P|\!\rangle\tau}{}^{?} \cdot \mapsto \langle\!|P''|\!\rangle$.

Consider the case that $\langle\!|x(y).P_1|\!\rangle = (\nu u)(x(y){:}\overline{u}y.\langle\!|P_1|\!\rangle|u(y){:}0) \xrightarrow{x(z)} (\nu u)(\overline{u}y.\langle\!|P_1|\!\rangle|u(y){:}0)$. By the definition,

$$\frac{z \notin (\mathrm{fn}\, P_1 \setminus \{y\}) \cup \{x\}}{x(y).P_1 \xrightarrow{x(z)} [z/y]P_1}$$

is inferred. Moreover,

$$\frac{\dfrac{\overline{u}z.[z/y]\langle\!|P_1|\!\rangle \xrightarrow{\overline{u}z} [z/y]\langle\!|P_1|\!\rangle \qquad \dfrac{y \notin (\mathrm{fn}\, 0 \setminus \{y\}) \cup \{u\}}{u(y){:}0 \xrightarrow{u(y)} 0}}{\overline{u}z.[z/y]\langle\!|P_1|\!\rangle|u(y){:}0 \xrightarrow{u\tau} [z/y]\langle\!|P_1|\!\rangle|0} \qquad u \notin \emptyset}{(\nu u)(\overline{u}z.[z/y]\langle\!|P_1|\!\rangle|u(y){:}0) \xrightarrow{u\tau} (\nu u)([z/y]\langle\!|P_1|\!\rangle|0)}$$

and

$$(\nu u)([z/y]\langle\!|P_1|\!\rangle|0) \mapsto [z/y]\langle\!|P_1|\!\rangle|0 \mapsto [z/y]\langle\!|P_1|\!\rangle \equiv_\alpha \langle\!|[z/y]P_1|\!\rangle$$

suffice to show the case. Consider the case that $\langle\!|!P_1|\!\rangle = !P_2 \xrightarrow{\tau} P_2^\dagger|[z/y]P_2^\ddagger|!P_2$ is inferred from $P_2 \xrightarrow{\overline{x}z} P_2^\dagger$ and $P_2 \xrightarrow{x(y)} P_2^\ddagger$. By induction hypothesis,

$$P_1 \xrightarrow{\overline{x}z} P_1^{\dagger\dagger} \qquad P_2^\dagger = \langle\!|P_1^{\dagger\dagger}|\!\rangle$$
$$P_1 \xrightarrow{x(y)} P_1^{\ddagger\ddagger} \qquad P_2^\ddagger \xrightarrow{\tau}{}^{?}\mapsto \langle\!|P_1^{\ddagger\ddagger}|\!\rangle$$

for some $P_1^{\dagger\dagger}$ and $P_1^{\ddagger\ddagger}$. Then, we have:

$$\frac{P_1 \xrightarrow{\overline{x}z} P_1^{\dagger\dagger} \qquad P_1 \xrightarrow{x(y)} P_1^{\ddagger\ddagger}}{!P_1 \xrightarrow{\tau} P_1^{\dagger\dagger}|[z/y]P_1^{\ddagger\ddagger}|!P_1}$$

[2]This terminology is proof-theoretically incorrect. It should be said that the translation $\langle\!|\cdot|\!\rangle$ is complete. However, we adopted it in order to emphasize regarding the original pi-calculus as a subsystem of the Mpi-calculus.

and

$$P_2^{\dagger}|[z/y]P_2^{\ddagger}|!P_2 = \langle\!|P_1^{\dagger\dagger}|\!\rangle|[z/y]P_2^{\ddagger}|!P_2 \xrightarrow{\tau}{}^{?} \cdot \mapsto \langle\!|P_1^{\dagger\dagger}|\!\rangle|[z/y]\langle\!|P_2^{\ddagger\ddagger}|\!\rangle|!P_2 \equiv_{\alpha} \langle\!|P_1^{\dagger\dagger}|[z/y]P_1^{\ddagger\ddagger}|!P_1|\!\rangle \ .$$

The other cases are trivial or similar to the above case. □

Theorem 3.3. *If* $\langle\!|P|\!\rangle \xrightarrow{\alpha_1} Q_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} Q_n$*, then there exist* $P \xrightarrow{\alpha_1}{}^{?} P_1 \xrightarrow{\alpha_2}{}^{?} \cdots \xrightarrow{\alpha_n}{}^{?} P_n$ *and* $u_1, \ldots, u_n \in \langle\!|P|\!\rangle\tau$ *such that* $Q_i \xrightarrow{u_i\tau}{}^{?} \cdot \mapsto \langle\!|P_i|\!\rangle$ *for any i.*

Proof. It follows from Lemma 3.2 and the commutativity of $\mapsto$ and $\xrightarrow{\tau}$. □These two theorems show that a process which has temporal precedence can be expressed in the Mpi-calculus up to internal transitions and the relation $\mapsto$.

Fu [9], Parrow and Victor [17, 18], and other researchers studied causality of processes, in particular van Breugel [23], and Merro and Sangiorgi [12] dealt with modifications of the pi-calculus. Input processes can make the same transition as ours in their system. However, their systems have a rule which corresponds to the rule:

$$\frac{P \xrightarrow{\overline{x}z} P' \qquad y \notin \{x, z\}}{x(y){:}P \xrightarrow{\tau} [z/y]P'} \ .$$

This rule amounts to admitting self-communication of processes. Such an input process is said to be a *delayed input* process. We cannot understand why they expressed causal dependency by delayed input, since we do not know why self-communication is primitive and whether the delayed input can express a process without self-communication whose outermost prefixes have the same link name. Here, we claim that our formalization of input processes is more natural than delayed input for the following reasons. First, consider the case where two identical processes on network are serving each other (as in peer-to-peer applications). In our system, they can be naturally expressed as $x(y){:}\overline{x}z.P$. Since self-communication is forbidden, the parallel composition $x(y){:}\overline{x}z.P|x(y){:}\overline{x}z.P$ can only transit to $\overline{x}z.[z/y]P|x(y){:}P$ (or the symmetric one) and then to $[z/y]P|[z/y]P$. On the other hand, in Merro and Sangiorgi's system the problem is that $x(y){:}\overline{x}z.P$ can transit to $[z/y]P$, that is, the process serves itself since self-communication is primitive. In the case where each of the processes is also allowed to serve itself, they can be expressed as $x(y){:}P|\overline{x}z.0$ in our system (and also in Merro and Sangiorgi's system). These examples suggest that our system is more expressive than Merro and Sangiorgi's, because self-communication can be controlled by the form of processes. Second, we express a process which can interact through outermost ports named x as $x(y).P|\overline{x}z.0$ in our system (using the translation $\langle\!|\cdot|\!\rangle$). This fact suggests that our system has delayed input processes. Finally, Merro and Sangiorgi discovered that a delayed input process could be expressed as syntax sugar in the localized asynchronous pi-calculus [12]. This fact claims that the localized asynchronous pi-calculus has causal dependency now that they realize causal dependency by delayed input. However, it appears that self-communication is admitted as a by-product—that a process based on the notion of a single-port is forced to be multi-ported. In contrast, our system has the advantage that we ourselves can choose whether to allow self-communication or not.[3] On the basis of these discussions, we cannot escape the conclusion that our formalization of input processes is more natural than that of delayed input.

At the last of this section, we refer to other concurrent systems except the Mpi-calculus with respect to controlling self-communication. As described above, we sometimes require a

[3]Controlling self-communication of processes is one of the reasons that we do not construct a concurrent system built on the notion of *abstraction* and *concretion*, for such a system forces $x(y).(P|Q) \equiv P|x(y).Q$ whenever $y \notin \mathrm{fn}\, P$. Then, it is natural that the process $x(y).\overline{x}z.0$ can be related to $0|0$ by the relation $\rightarrow$ mentioned in Section 5. Such a system is, therefore, not suitable for controlling self-communication of processes.

process which sends a message at one time, or receives a message at other times. We checked it impossible to make such a process to avoid self-communication on many concurrent systems without the nondeterministic choice operator + except the Mpi-calculus, for instance, the pi-calculus without +, pi-epsilon calculus [23], and localized asynchronous pi-calculus [12]. Formally, we proved:

Theorem 3.4. *In the pi-calculus without +, pi-epsilon calculus, or localized asynchronous pi-calculus, if* $P \xrightarrow{\overline{x}z} P'$ *and* $P \xrightarrow{x(y)} P''$*, then there exists* P''' *such that* $P \xrightarrow{\tau} P'''$*.*

4 Equivalence on Processes

In this section we introduce a notion of equivalence on processes. Our notion of equivalence amounts to the so-called open bisimilarity.

The notion of equivalence is very important. For instance, let us think about the case where one part of a process needs to be replaced. Although it goes without saying that we can replace it with the same one, there exist several situations where we would like to replace it with something else which is equivalent to the original one, namely which has the same capability as the original one. In order to formalize the notion of equivalence of processes properly, we must consider what it means for two processes to have the same capability. Our solution here is that two processes have the same capability when they behave in the same way. The equivalence which we define in this section is based on this intuition; it is surely an equivalence on processes (i.e., a congruence relation on processes) as we prove later.

4.1 The Open Bisimilarity

The following notion of bisimulation is well-known.

Definition 4.1. *A relation* $\mathcal{R}$ *on processes is said to be an* open simulation *if* σ *is a substitution,* $\sigma P \xrightarrow{\alpha} P'$*,* $(P, Q) \in \mathcal{R}$*, and* $\mathrm{bn}\,\alpha \cap (\mathrm{fn}\,\sigma P \cup \mathrm{fn}\,\sigma Q) = \emptyset$*, then there exists* Q' *such that* $\sigma Q \xrightarrow{\alpha} Q'$ *and* $(P', Q') \in \mathcal{R}$*.*

A relation $\mathcal{R}$ on processes is said to be an *open bisimulation* if $\mathcal{R}$ is an open simulation and so is $\mathcal{R}^{-1}$. Two processes P and Q are said to be *open bisimilar*, written $P \sim_{\mathrm{o}} Q$, if $(P, Q) \in \mathcal{R}$ for some open bisimulation $\mathcal{R}$.

On the other hand, we define the following notion of bisimulation ad hoc (in fact, easier than an open bisimulation to deal with at this time).

Definition 4.2. *A relation* $\mathcal{R}$ *on processes is a* hypersimulation *whenever*

1. $\mathcal{R}$ *is closed under substitutions,*
2. $P \xrightarrow{\alpha} P'$*,* $\mathrm{bn}\,\alpha \cap (\mathrm{fn}\,P \cup \mathrm{fn}\,Q) = \emptyset$*, and* $(P, Q) \in \mathcal{R}$ *imply that there exists* Q' *such that* $Q \xrightarrow{\alpha} Q'$ *and* $(P', Q') \in \mathcal{R}$*.*

A relation $\mathcal{R}$ on processes is said to be a *hyperbisimulation* if $\mathcal{R}$ is a hypersimulation and so is $\mathcal{R}^{-1}$. Two processes P and Q are said to be *hyperbisimilar*, written $P \sim Q$, if $(P, Q) \in \mathcal{R}$ for some hyperbisimulation $\mathcal{R}$.

The following fact ensures that the relation $\sim$ is less of a trivial one.

Proposition 4.3. *The relation* $\equiv$ *is a hyperbisimulation.*

Obviously, $\sim$ itself is a hyperbisimulation. It is, therefore, the greatest hyperbisimulation.

The notion of hyperbisimulation is different from that of open bisimulation; although any hyperbisimulation is an open bisimulation, the converse does not hold, as we have a counter-example $\{(\overline{x}y.0, \overline{x}y.0), (0,0)\}$. It is, however, not our intention to introduce a new notion of equivalence. Indeed, it turns out that the notion of the hyperbisimilarity, which is the greatest hyperbisimulation, coincides with that of the open bisimilarity:

Proposition 4.4. *$P \sim_o Q$ if and only if $P \sim Q$.*

Proof. To prove that $\sim_o$ is contained in $\sim$, it is enough to show that for every open bisimulation $\mathcal{O}$, its substitution closure $\mathcal{O}'$ defined by

$$\{\, (\sigma R, \sigma S) \mid \sigma \text{ is arbitrary and } (R,S) \in \mathcal{O} \,\}$$

is a hyperbisimulation. Let (P,Q) belong to some open bisimulation $\mathcal{O}$. It is clear that $\mathcal{O}'$ is closed under substitutions by the definition. Now suppose that $(\sigma R, \sigma S) \in \mathcal{O}'$, $\sigma R \xrightarrow{\alpha} R'$, and $\mathrm{bn}\,\alpha \cap (\mathrm{fn}\,\sigma P \cup \mathrm{fn}\,\sigma Q) = \emptyset$. By the definition of $\mathcal{O}'$, we have $(R,S) \in \mathcal{O}$. It follows that $\sigma S \xrightarrow{\alpha} S'$ and $(R', S') \in \mathcal{O}$ since $\mathcal{O}$ is an open bisimulation. We therefore have $(R', S') \in \mathcal{O}'$, which means that $\mathcal{O}'$ is a hypersimulation. Similarly, we can check that $\mathcal{O}'^{-1}$ is also a hypersimulation.

To show the converse, it is enough to show that any hyperbisimulation is an open bisimulation. Let $\mathcal{R}$ be a hyperbisimulation and suppose $(P,Q) \in \mathcal{R}$, $\sigma P \xrightarrow{\alpha} P'$, and $\mathrm{bn}\,\alpha \cap (\mathrm{fn}\,\sigma P \cup \mathrm{fn}\,\sigma Q) = \emptyset$. Since $\mathcal{R}$ is closed under substitutions, we have $(\sigma P, \sigma Q) \in \mathcal{R}$. Furthermore, there exists Q' such that $\sigma Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{R}$ since $\mathcal{R}$ is a hyperbisimulation. This shows that $\mathcal{R}$ is an open bisimulation. Similarly, we can check that $\mathcal{R}^{-1}$ is also an open bisimulation. □Now that we may discuss $\sim$ only. The following theorem is one part of main theorem in this section.

Theorem 4.5. *The relation $\sim$ is an equivalence relation.*

Proof. Reflexivity and symmetricity are trivial. To show transitivity, it is sufficient to check that whenever $\mathcal{R}$ and $\mathcal{S}$ are hyperbisimulations, $\mathcal{R} \cdot \mathcal{S}$ is also a hyperbisimulation.

It is obvious that $\mathcal{R} \cdot \mathcal{S}$ is closed under substitutions. Now, suppose that $(P,R) \in \mathcal{R} \cdot \mathcal{S}$, $P \xrightarrow{x(y)} P'$ and $y \notin \mathrm{fn}\,P \cup \mathrm{fn}\,R$. There exists Q such that $(P,Q) \in \mathcal{R}$ and $(Q,R) \in \mathcal{S}$. Without loss of generality, we may assume that $y \notin \mathrm{fn}\,Q$; otherwise, let σ be a substitution which replaces y with a fresh name y'. Note that $\sigma P = P$ and $\sigma R = R$. Since $\mathcal{R}$ and $\mathcal{S}$ are hyperbisimulations, we have $(P, \sigma Q) \in \mathcal{R}$ and $(\sigma Q, R) \in \mathcal{S}$, and clearly $y \notin \mathrm{fn}\,\sigma Q$. Now, there exists Q' such that $Q \xrightarrow{x(y)} Q'$ (or $\sigma Q \xrightarrow{x(y)} Q'$ in case $y \in \mathrm{fn}\,Q$) and $(P', Q') \in \mathcal{R}$. Similarly, there exists R' such that $R \xrightarrow{x(y)} R'$ and $(Q', R') \in \mathcal{S}$. We can therefore conclude that $(P', R') \in \mathcal{R} \cdot \mathcal{S}$.

Other cases where $P \xrightarrow{\alpha} P'$ with α not an input action are similar. □

4.2 Up-to Technique

In this subsection we prove several lemmata which are useful for showing that two processes are hyperbisimilar. In general, how to prove whether two processes are bisimilar is one of the most interesting topics in concurrency field. The first approach was done by Milner against a concurrent system called Calculus of Communication System [13]. The method is called up-to $\sim$ technique, in which the closure of a bisimulation is achieved up to the bisimilarity itself. The method was spread to the weak bisimilarity version [20]. While these methods are compositional to the original relation in a sense, up-to restriction technique [16] and up-to

injection technique [7, 16] are contextual. Extending these techniques, Sangiorgi succeeded in establishing a series of lemmata to prove two processes related contextually to be bisimilar [19]. The content of this subsection is largely due to Sangiorgi's up-to context technique [19, 21], though accommodated to our notion of hyperbisimulation.

Definition 4.6. *A relation $\mathcal{R}$ progresses to a relation $\mathcal{S}$, written $\mathcal{R} \rightarrowtail \mathcal{S}$, if $\mathcal{R}$ and $\mathcal{S}$ are closed under substitutions and whenever $(P, Q) \in \mathcal{R}$ and $\mathrm{bn}\,\alpha \cap (\mathrm{fn}\,P \cup \mathrm{fn}\,Q) = \emptyset$,*

1. *$P \xrightarrow{\alpha} P'$ implies there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{S}$.*
2. *$Q \xrightarrow{\alpha} Q'$ implies there exists P' such that $P \xrightarrow{\alpha} P'$ and $(P', Q') \in \mathcal{S}$.*

We emphasize that when we write $\mathcal{R} \rightarrowtail \mathcal{S}$, it is assumed that $\mathcal{R}$ and $\mathcal{S}$ are closed under substitutions. This is an accommodation to our notion of hyperbisimulation and the only different point from existing works.

Proposition 4.7. *If $\mathcal{R}$ progresses to a hyperbisimulation, then $\mathcal{R}$ is contained in $\sim$.*

Proof. Let $\mathcal{S}$ be a hyperbisimulation such that $\mathcal{R} \rightarrowtail \mathcal{S}$. It is sufficient to show that $\mathcal{R} \cup \mathcal{S}$ is a hyperbisimulation. Notice that the hypothesis forces $\mathcal{R} \cup \mathcal{S}$ closed under substitution. Suppose that $P \xrightarrow{\alpha} P'$, $\mathrm{bn}\,\alpha \cap (\mathrm{fn}\,P \cup \mathrm{fn}\,Q) = \emptyset$, and $(P, Q) \in \mathcal{R} \cup \mathcal{S}$. In the case of $(P, Q) \in \mathcal{R}$, there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{S}$ by $\mathcal{R} \rightarrowtail \mathcal{S}$. On the other hand, in the case of $(P, Q) \in \mathcal{S}$, it also holds since $\mathcal{S}$ is a hypersimulation. Hence, $\mathcal{R} \cup \mathcal{S}$ is a hypersimulation. Similarly, $(\mathcal{R} \cup \mathcal{S})^{-1}$ is also a hypersimulation. □

Lemma 4.8. *Let $\mathcal{R}$, $\mathcal{S}$, $\mathcal{T}$, $\mathcal{R}_i$, and $\mathcal{S}_j$ be relations.*

1. *Suppose that $\mathcal{R}$ is closed under substitutions. Then $\mathcal{R} \subseteq \mathcal{S}$ and $\mathcal{S} \rightarrowtail \mathcal{T}$ imply $\mathcal{R} \rightarrowtail \mathcal{T}$.*
2. *Suppose that $\mathcal{T}$ is closed under substitutions. Then $\mathcal{R} \rightarrowtail \mathcal{S}$ and $\mathcal{S} \subseteq \mathcal{T}$ imply $\mathcal{R} \rightarrowtail \mathcal{T}$.*
3. *Suppose that $\mathcal{S}_j$ is closed under substitutions for any $j \in J$, and for each $i \in I$ there exists $j \in J$ such that $\mathcal{R}_i \rightarrowtail \mathcal{S}_j$. Then $\bigcup\{\mathcal{R}_i \mid i \in I\} \rightarrowtail \bigcup\{\mathcal{S}_j \mid j \in J\}$.*
4. *Suppose that for each $i \in I$ there exists $i' \in I$ such that $\mathcal{R}_i \rightarrowtail \mathcal{R}_{i'}$. Then $\bigcup\{\mathcal{R}_i \mid i \in I\}$ is a hyperbisimulation.*

Proof. 1. The hypothesis makes $\mathcal{T}$ closed under substitutions. Suppose that $P \xrightarrow{\alpha} P'$ and $(P, Q) \in \mathcal{R}$. We have $(P, Q) \in \mathcal{S}$ by $\mathcal{R} \subseteq \mathcal{S}$. There exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{T}$ by $\mathcal{S} \rightarrowtail \mathcal{T}$. It is similar in the case of $Q \xrightarrow{\alpha} Q'$.

2. It is easy to check.

3. It is obvious that $\bigcup\{\mathcal{S}_j \mid j \in J\}$ is closed under substitutions. Let $(P, Q) \in \bigcup\{\mathcal{R}_i \mid i \in I\}$. Then, there exists $i \in I$ such that $(P, Q) \in \mathcal{R}_i$. By the hypothesis, there exists $j \in J$ such that $\mathcal{R}_i \rightarrowtail \mathcal{S}_j$. Suppose that $P \xrightarrow{\alpha} P'$, then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{S}_j$. Hence, $(P', Q') \in \bigcup\{\mathcal{S}_j \mid j \in J\}$. It is similar in the case of $Q \xrightarrow{\alpha} Q'$.

4. This is a special case of 3. □

Definition 4.9. *A function $\mathfrak{F}$ on relations on processes is said to be* monotonic *if $\mathcal{R} \rightarrowtail \mathcal{S}$ implies $\mathfrak{F}(\mathcal{R}) \rightarrowtail \mathfrak{F}(\mathcal{S})$.*

Proposition 4.10. *A monotonic function $\mathfrak{F}$ preserves closure property under substitution, that is, if $\mathcal{R}$ is closed under substitutions, then $\mathfrak{F}(\mathcal{R})$ is closed under substitutions.*

Proof. Since $\mathcal{R}$ is closed under substitutions, we have $\emptyset \rightarrowtail \mathcal{R}$. Hence, it turns out that $\mathfrak{F}(\emptyset) \rightarrowtail \mathfrak{F}(\mathcal{R})$, especially that $\mathfrak{F}(\mathcal{R})$ is closed under substitutions. □

Proposition 4.11. *If $\mathfrak{F}$ is a monotonic function such that $\mathcal{R} \rightarrowtail \mathfrak{F}(\mathcal{R})$, then $\mathcal{R} \subseteq \sim$.*

Proof. We define the relations $\mathcal{R}_n$ $(0 \leq n \leq \omega)$ as follows:

$$\begin{aligned} \mathcal{R}_0 &= \mathcal{R} \\ \mathcal{R}_{n+1} &= \mathcal{R}_n \cup \mathfrak{F}(\mathcal{R}_n) \\ \mathcal{R}_\omega &= \bigcup \{ \mathcal{R}_i \mid 0 \leq i < \omega \} \ . \end{aligned}$$

We first show $\mathcal{R}_n \rightarrowtail \mathcal{R}_{n+1}$ for any n by induction on n. Suppose $n = 0$. The hypothesis derives $\mathcal{R}_0 \rightarrowtail \mathfrak{F}(\mathcal{R}_0) \subseteq \mathcal{R}_1$. Since $\mathcal{R}_1$ is closed under substitutions, Lemma 4.8.2 induces $\mathcal{R}_0 \rightarrowtail \mathcal{R}_1$. Suppose $n > 0$. Induction hypothesis and the monotonicity of $\mathfrak{F}$ derive $\mathfrak{F}(\mathcal{R}_{n-1}) \rightarrowtail \mathfrak{F}(\mathcal{R}_n)$. Hence $\mathcal{R}_n \rightarrowtail \mathcal{R}_{n+1}$ holds by Lemma 4.8.3.

By Lemma 4.8.4, it follows that $\mathcal{R}_\omega$ is a hyperbisimulation. Moreover, we obtain $\mathcal{R} \rightarrowtail \mathcal{R}_\omega$ by applying Lemma 4.8.2 to $\mathcal{R} \rightarrowtail \mathcal{R}_1 \subseteq \mathcal{R}_\omega$. As a consequence, we have $\mathcal{R} \subseteq \sim$ by Proposition 4.7. □Next, let us describe several ways to construct new monotonic functions from given ones. The composition $\mathfrak{F} \cdot \mathfrak{G}$ and the union $\bigcup\{ \mathfrak{F}_i \mid i \in I \}$ are defined as follows:

$$\begin{aligned} (\mathfrak{F} \cdot \mathfrak{G})(\mathcal{R}) &= \mathfrak{F}(\mathcal{R}) \cdot \mathfrak{G}(\mathcal{R}) \\ (\bigcup\{ \mathfrak{F}_i \mid i \in I \})(\mathcal{R}) &= \bigcup\{ \mathfrak{F}_i(\mathcal{R}) \mid i \in I \} \ . \end{aligned}$$

Lemma 4.12. *Let $\mathfrak{F}$, $\mathfrak{G}$, and $\mathfrak{F}_i$ be monotonic.*

1. *$\mathfrak{F} \cdot \mathfrak{G}$ is monotonic.*
2. *$\bigcup\{ \mathfrak{F}_i \mid i \in I \}$ is monotonic.*

Proof. Suppose $\mathcal{R} \rightarrowtail \mathcal{S}$.

1. Let $(P, R) \in (\mathfrak{F} \cdot \mathfrak{G})(\mathcal{R})$. Since $\mathfrak{F}(\mathcal{R})$ and $\mathfrak{G}(\mathcal{R})$ are closed under substitutions, so are $(\mathfrak{F} \cdot \mathfrak{G})(\mathcal{R})$ and $(\mathfrak{F} \cdot \mathfrak{G})(\mathcal{S})$. Let $P \xrightarrow{\alpha} P'$, $\mathrm{bn}\,\alpha \cap (\mathrm{fn}\,P \cup \mathrm{fn}\,R) = \emptyset$, and $(P, R) \in (\mathfrak{F} \cdot \mathfrak{G})(\mathcal{R})$. By the definition, there exists Q such that $(P, Q) \in \mathfrak{F}(\mathcal{R})$ and $(Q, R) \in \mathfrak{G}(\mathcal{R})$. Without loss of generality, we may assume that $\mathrm{bn}\,\alpha$ and $\mathrm{fn}\,Q$ are disjoint (otherwise, let σ be a substitution which replaces $\mathrm{bn}\,\alpha$ with a fresh name and consider σQ as in the proof of Theorem 4.5). Then, there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathfrak{F}(\mathcal{R})$. Thus, there exists R' such that $R \xrightarrow{\alpha} R'$ and $(Q', R') \in \mathfrak{G}(\mathcal{S})$. The existence of Q' ensures $(P', R') \in (\mathfrak{F} \cdot \mathfrak{G})(\mathcal{S})$. It is similar in the case of $R \xrightarrow{\alpha} R'$.

2. By Lemma 4.8.3. □

Finally, let us give some concrete examples of monotonic functions, which are useful when confirming that two processes are hyperbisimilar. First of all, let $\mathfrak{K}$ be a constant function to $\sim$, i.e., $\mathfrak{K}(\mathcal{R}) = \sim$ for any $\mathcal{R}$. Next, define $\mathfrak{N}$ as follows:

$$\mathfrak{N}(\mathcal{R}) = \{ ((\nu y)P, (\nu y)Q \mid (P, Q) \in \mathcal{R} \text{ and } y \text{ is arbitrary} \} \ .$$

Two processes $(\nu y)P$ and $(\nu y)Q$ are said to be hyperbisimilar up to restriction when P and Q are hyperbisimilar. The importance of the monotonicity of this function $\mathfrak{N}$ depends on the existence of the rule:

$$\frac{P \xrightarrow{\overline{x}(y)} P' \qquad Q \xrightarrow{x(y)} Q'}{P|Q \xrightarrow{\tau} (\nu y)(P'|Q')} \quad .$$

In general, we expect that two processes hyperbisimilar up to context are hyperbisimilar. Unfortunately, however, the following function is not monotonic:

$$\mathfrak{O}(\mathcal{R}) = \{\, (C[P], C[Q]) \mid C \text{ is a context and } (P, Q) \in \mathcal{R} \,\} \quad .$$

The failure is due to replication, i.e., copying processes. To see this, let P be $\overline{x}y.0$ and Q be $\overline{x}y.0|0$, and the relation $\mathcal{R}$ be

$$\{(0, 0|0)\} \cup \{\, (\overline{x}y.0, \overline{x}y.0|0) \mid x \text{ and } y \text{ are arbitrary} \,\} \quad .$$

It immediately turns out that $\mathcal{R} \rightarrowtail \mathcal{R}$, $(!P, !Q) \in \mathfrak{O}(\mathcal{R})$ and $!P \xrightarrow{\overline{x}y} 0|!\overline{x}y.0$. However, the only transitions from $!Q$ is the following:

$$!Q \xrightarrow{\overline{x}y} 0|0|!(\overline{x}y.0|0) =: Q' \quad .$$

Obviously, there exist no context C and $(R, S) \in \mathcal{R}$ such that $P' = C[R]$ and $Q' = C[S]$. Therefore, it follows that $(P', Q') \notin \mathfrak{O}(\mathcal{R})$, and thus $\mathfrak{O}$ is not monotonic.

It is nevertheless possible to maintain our intuition that two processes should be hyperbisimilar whenever they are hyperbisimilar up to context. The idea is to extend the previous definition of $\mathfrak{O}$ based on single-hole contexts to a more general one based on multi-hole contexts.

A multi-hole context is a process which has some indexed holes. More precisely, the set of multi-hole contexts is defined by

$$\boldsymbol{C}, \boldsymbol{\mathcal{D}}, \ldots ::= P \mid [\cdot]_i \mid \tau.\boldsymbol{C} \mid \overline{x}y.\boldsymbol{C} \mid x(y).\boldsymbol{C} \mid \boldsymbol{C}|\boldsymbol{\mathcal{D}} \mid (\nu x)\boldsymbol{C} \mid\ !\boldsymbol{C}$$

where i is a natural number. Let $\boldsymbol{C}$ be a multi-hole context with its maximal index n. Let $\boldsymbol{P}$ denote a sequence $P_1, \ldots, P_n$ of processes. Then $\boldsymbol{C}[\boldsymbol{P}]$ denotes the result of substituting P_i for $[\cdot]_i$ for every $i \leq n$. Note that a multi-hole context is allowed to have several occurrences of a hole $[\cdot]_i$ with the same index i. In such a case, $\boldsymbol{C}[\boldsymbol{P}]$ is obtained by substituting P_i for all occurrences of $[\cdot]_i$.

Define $\mathfrak{M}$ by

$$\mathfrak{M}(\mathcal{R}) = \{\, (\boldsymbol{C}[\boldsymbol{P}], \boldsymbol{C}[\boldsymbol{Q}]) \mid \boldsymbol{C} \text{ is a multi-hole context and } (\boldsymbol{P}, \boldsymbol{Q}) \in \mathcal{R} \,\} \quad .$$

Proposition 4.13. *$\mathfrak{K}$ and $\mathfrak{M}$ are monotonic.*

Proof. First, $\mathfrak{K}$ is monotonic since it is a constant function and $\sim$ is a hyperbisimulation.

Next, we show that $\mathfrak{M}$ is monotonic. Suppose $\mathcal{R} \rightarrowtail \mathcal{S}$. Obviously, $\mathfrak{M}(\mathcal{R})$ and $\mathfrak{M}(\mathcal{S})$ are closed under substitutions. It is therefore left to check that whenever $(\boldsymbol{P}, \boldsymbol{Q}) \in \mathcal{R} \neq \emptyset$ and $\boldsymbol{C}[\boldsymbol{P}] \xrightarrow{\alpha} P'$ where $\mathrm{bn}\,\alpha \cap (\mathrm{fn}\,\boldsymbol{C}[\boldsymbol{P}] \cup \mathrm{fn}\,\boldsymbol{C}[\boldsymbol{Q}]) = \emptyset$, there exists Q' such that $\boldsymbol{C}[\boldsymbol{Q}] \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathfrak{M}(\mathcal{S})$. We prove this by induction on multi-hole context $\boldsymbol{C}$.

Case 1. It is obvious in case there exists no hole.

Case 2. Suppose that $\boldsymbol{C}$ is $[\cdot]_i$. Then $\boldsymbol{C}[\boldsymbol{P}]$ and $\boldsymbol{C}[\boldsymbol{Q}]$ are P_i and Q_i, respectively. Since $(P_i, Q_i) \in \mathcal{R} \rightarrowtail \mathcal{S}$, there exists Q' such that $Q_i \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{S} \subseteq \mathfrak{M}(\mathcal{S})$.

Case 3. Suppose that $\boldsymbol{C}$ is $\alpha.\boldsymbol{C}'$. Then the only possible transition is $\alpha.\boldsymbol{C}'[\boldsymbol{P}] \xrightarrow{\alpha} \boldsymbol{C}'[\boldsymbol{P}]$. In this case, we have $\alpha.\boldsymbol{C}'[\boldsymbol{Q}] \xrightarrow{\alpha} \boldsymbol{C}'[\boldsymbol{Q}]$. It follows that $(\boldsymbol{C}'[\boldsymbol{P}], \boldsymbol{C}'[\boldsymbol{Q}]) \in \mathfrak{M}(\mathcal{R}) \subseteq \mathfrak{M}(\mathcal{S})$.

Case 4. $\boldsymbol{C}$ is $x(y)\!:\!\boldsymbol{C}'$. Let us write $x(y)\!:\!\boldsymbol{C}'[\boldsymbol{P}]$ and $x(y)\!:\!\boldsymbol{C}'[\boldsymbol{Q}]$ to be $x(y)\!:\!P$ and $x(y)\!:\!Q$, respectively. The proof is similar to the above when $x(y)\!:\!P \xrightarrow{x(z)} [z/y]P$ since $\mathfrak{M}(\mathcal{R})$ is closed under substitution. So, suppose that $x(y)\!:\!P \xrightarrow{\alpha} x(y)\!:\!P'$ is inferred from $P \xrightarrow{\alpha} P'$, $x \notin \mathrm{bn}\,\alpha$ and $y \notin \mathrm{n}\,\alpha$. Clearly $\mathrm{bn}\,\alpha$ and $\mathrm{fn}\,P \cup \mathrm{fn}\,Q$ are disjoint. By induction hypothesis, there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathfrak{M}(\mathcal{S})$. Hence, $x(y)\!:\!Q \xrightarrow{\alpha} x(y)\!:\!Q'$ and $(x(y)\!:\!P', x(y)\!:\!Q') \in \mathfrak{M}(\mathcal{S})$, since $\mathfrak{M}(\mathcal{S})$ is closed under the input prefixes $x(y)$.

Case 5. Suppose that $\boldsymbol{C}$ is $!\boldsymbol{C}'$. Let us write $!\boldsymbol{C}'[\boldsymbol{P}]$ and $!\boldsymbol{C}'[\boldsymbol{Q}]$ to be $!P$ and $!Q$, respectively.

There are three subcases. First, suppose that $!P \xrightarrow{\alpha} P'|!P$ is inferred from $P \xrightarrow{\alpha} P'$. By induction hypothesis, there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathfrak{M}(\mathcal{S})$. Hence, it can be inferred that $!Q \xrightarrow{\alpha} Q'|!Q$. Furthermore, we have $(P'|!P, Q'|!Q) \in \mathfrak{M}(\mathcal{S})$.

Next, $!P \xrightarrow{\tau} P_1|[z/y]P_2|!P$ is inferred from $P \xrightarrow{\overline{x}z} P_1$ and $P \xrightarrow{x(y)} P_2$. Without loss of generality, we may assume that $y \notin \mathrm{fn}\,Q$ (otherwise, choose an alpha-convertible variant of Q which does not contain y as bound name). By induction hypothesis, there exist Q_1 and Q_2 such that $Q \xrightarrow{\overline{x}z} Q_1$, $(P_1, Q_1) \in \mathfrak{M}(\mathcal{S})$, $Q \xrightarrow{x(y)} Q_2$, and $(P_2, Q_2) \in \mathfrak{M}(\mathcal{S})$. Hence it can be inferred that $!Q \xrightarrow{\tau} Q_1|[z/y]Q_2|!Q$. Furthermore, we have $(P_1|[z/y]P_2|!P, Q_1|[z/y]Q_2|!Q) \in \mathfrak{M}(\mathcal{S})$. The last subcase is similar to this case.

The other cases are similar, too (apply the alpha-conversion argument used in Case 5, if necessary). □

4.3 Main Theorems

Having developed some useful lemmata, we are now ready to prove:

Theorem 4.14. *The relation $\sim$ is a congruence relation.*

Proof. First, we show that $\sim$ is closed under the non-input prefixes α. Let $\mathcal{R}$ be

$$\sim \cup \{\, (\beta.R, \beta.S) \mid R \sim S \text{ and } \beta \text{ is the same prefix form as } \alpha \,\} \ .$$

It is sufficient to show that $\mathcal{R}$ progresses to $\sim$; then it follows from Proposition 4.7 that $R \sim S$ implies $(\alpha.R, \alpha.S) \in \mathcal{R} \subseteq \sim$. Clearly $\mathcal{R}$ is closed under substitution. Let $(P, Q) \in \mathcal{R}$. When $P \sim Q$, it follows from the definition of $\sim$ that whenever $P \xrightarrow{\beta} P'$ and $\mathrm{bn}\beta \cap (\mathrm{fn}\,P \cup \mathrm{fn}\,Q) = \emptyset$, there exists Q' such that $Q \xrightarrow{\beta} Q'$ and $P' \sim Q'$. Otherwise, P and Q are of the form $\beta.R$ and $\beta.S$, respectively. Note that the only possible transition from $\beta.R$ is $\beta.R \xrightarrow{\beta} R$, and similarly the only possible transition from $\beta.S$ is $\beta.S \xrightarrow{\beta} S$. Furthermore, $R \sim S$ by the definition of $\mathcal{R}$. This (as well as the symmetric case) shows that $\mathcal{R}$ progresses to $\sim$.

Second, we show that $\sim$ is closed under the input prefixes $x(y)$. Let $\mathcal{R}$ be

$$\sim \cup \{\, (v(w)\!:\!R, v(w)\!:\!S) \mid R \sim S \text{ and } v \text{ and } w \text{ are arbitrary } \} \ .$$

It is sufficient to show that $\mathcal{R}$ is a hyperbisimulation; then $R \sim S$ implies $(x(y)\!:\!R, x(y)\!:\!S) \in \mathcal{R} \subseteq \sim$.

Clearly $\mathcal{R}$ is closed under substitutions. Let $(P, Q) \in \mathcal{R}$. The case $P \sim Q$ is similar to the above. Otherwise, P and Q are of the form $v(w)\!:\!R$ and $v(w)\!:\!S$, respectively. The proof is also similar to the above in the case of $v(w)\!:\!R \xrightarrow{v(z)} [z/w]R$. Consider

$$\frac{R \xrightarrow{\alpha} R' \qquad v \notin \mathrm{bn}\,\alpha \qquad w \notin \mathrm{n}\,\alpha}{v(w){:}R \xrightarrow{\alpha} v(w){:}R'}$$

We may suppose that $\mathrm{bn}\,\alpha$ and $\mathrm{fn}\,v(w){:}R \cup \mathrm{fn}\,v(w){:}S$ are disjoint. The definition of $\mathcal{R}$ derives $R \sim S$. Furthermore, it is clear that $\mathrm{bn}\,\alpha$ and $\mathrm{fn}\,R \cup \mathrm{fn}\,S$ are disjoint. Hence there exists S' such that $S \xrightarrow{\alpha} S'$ and $R' \sim S'$, so it can be inferred that $v(w){:}S \xrightarrow{\alpha} v(w){:}S'$ and $(v(w){:}R', v(w){:}S') \in \mathcal{R}$. This (as well as the symmetric case) shows that $\mathcal{R}$ is a hyperbisimulation.

Finally, we show that $\sim$ is closed under parallel compositions. Other cases are easier than this case. Let $\mathcal{R}$ be

$$\sim \cup \{\, (P|R, Q|R) \mid P \sim Q \text{ and } R \text{ is arbitrary } \}\ .$$

It is sufficient to show that $\mathcal{R}$ progresses to $\mathfrak{M}(\mathcal{R})$; then it follows from Proposition 4.11 that $P \sim Q$ implies $(P|R, Q|R) \in \mathcal{R} \subseteq \sim$.

Let $(P|R, Q|R) \in \mathcal{R}$ with $P \sim Q$. There are six subcases.

First, suppose that $P|R \xrightarrow{\alpha} P'|R$ is inferred from $P \xrightarrow{\alpha} P'$ and $\mathrm{bn}\,\alpha \cap \mathrm{fn}\,R = \emptyset$. We may also assume that $\mathrm{bn}\,\alpha \cap (\mathrm{fn}\,P|R \cup \mathrm{fn}\,Q|R) = \emptyset$. Since $P \sim Q$, there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \sim Q'$. Hence, $(P'|R, Q'|R) \in \mathcal{R} \subseteq \mathfrak{M}(\mathcal{R})$.

Second, suppose that $P|R \xrightarrow{\alpha} P|R'$ is inferred from $R \xrightarrow{\alpha} R'$ and $\mathrm{bn}\,\alpha \cap \mathrm{fn}\,P = \emptyset$. We stress that we may assume that $\mathrm{bn}\,\alpha \cap (\mathrm{fn}\,P|R \cup \mathrm{fn}\,Q|R) = \emptyset$. Then, it is easy to see that $Q|R \xrightarrow{\alpha} Q|R'$ and $(P|R', Q|R') \in \mathcal{R} \subseteq \mathfrak{M}(\mathcal{R})$.

Third, suppose that $P|R \xrightarrow{\tau} P'|[z/y]R'$ is inferred from $P \xrightarrow{\overline{x}z} P'$ and $R \xrightarrow{x(y)} R'$. Since $P \sim Q$, there exists Q' such that $Q \xrightarrow{\overline{x}z} Q'$ and $P' \sim Q'$. Therefore, we have $Q|R \xrightarrow{\tau} Q'|[z/y]R'$ and $(P'|[z/y]R', Q'|[z/y]R') \in \mathcal{R} \subseteq \mathfrak{M}(\mathcal{R})$.

Fourth, suppose that $P|R \xrightarrow{\tau} [z/y]P'|R'$ is inferred from $P \xrightarrow{x(y)} P'$ and $R \xrightarrow{\overline{x}z} R'$. Let w be a fresh name. Proposition 2.1.3 induces $P \xrightarrow{x(w)} [w/y]P'$. Since $P \sim Q$, there exists Q' such that $Q \xrightarrow{x(w)} Q'$ and $[w/y]P' \sim Q'$. It follows that $Q|R \xrightarrow{\tau} [z/w]Q'|R'$. Since $\sim$ is closed under substitutions, $[z/w]([w/y]P') \sim [z/w]Q'$. Therefore, $([z/y]P'|R', [z/y]Q'|R') \in \mathcal{R} \subseteq \mathfrak{M}(\mathcal{R})$.

Fifth, suppose that $P|R \xrightarrow{\tau} (\nu y)(P'|R')$ is inferred from $P \xrightarrow{\overline{x}(y)} P'$ and $R \xrightarrow{x(y)} R'$. As before, we may assume that $y \notin \mathrm{fn}\,P \cup \mathrm{fn}\,Q$ without loss of generality. Since $P \sim Q$, there exists Q' such that $Q \xrightarrow{\overline{x}(y)} Q'$ and $P' \sim Q'$. Hence, $Q|R \xrightarrow{\tau} (\nu y)(Q'|R')$ and $((\nu y)(P'|R'), (\nu y)(Q'|R')) \in \mathfrak{M}(\mathcal{R})$.

Sixth, suppose that $P|R \xrightarrow{\tau} (\nu y)(P'|R')$ is inferred from $P \xrightarrow{x(y)} P'$ and $R \xrightarrow{\overline{x}(y)} R'$. As before, we may assume that $y \notin \mathrm{fn}\,P \cup \mathrm{fn}\,Q$. Since $P \sim Q$, there exists Q' such that $Q \xrightarrow{\overline{x}(y)} Q'$ and $P' \sim Q'$. Hence, $Q|R \xrightarrow{\tau} (\nu y)(Q'|R')$ and $((\nu y)(P'|R'), (\nu y)(Q'|R')) \in \mathfrak{M}(\mathcal{R})$. □

We have succeeded in defining a notion of equivalence on processes with causal dependency. Thus $v(w){:}x(y){:}P \sim x(y){:}v(w){:}P$ claims that these two input prefixes are independent. Remarkably, two processes identified under considering causal dependency are exactly equivalent.

Next, we show an important property for concurrent systems. The following statement is called *the replication theorem*:

$$(\nu x)(P|Q|!x(y).R) \sim (\nu x)(P|!x(y).R)|(\nu x)(Q|!x(y).R)\ .$$

This statement is intuitively interpreted as follows. Let R be a resource which is used by other processes. It is natural to think of R as replicable since it is expected to be invoked out arbitrarily many times. On the left-hand side, P and Q share R, while on the right-hand side, P and Q have their own copies of R. The above statement thus claims that whenever a resource

is shared by two processes, the resource may be distributed to each of the processes. However, this does not hold in general. Typically it does not hold when the processes involved send x as message. Since the name x is expected to serve as a "guard" for the resource R, we restrict it used only for the purpose. This is also why we do not use colon prefixes but dot prefixes.

A process P is said to be *x-negative* if x occurs in P only in the form $\overline{x}y$. A process which is alpha-convertible to an x-negative process is also considered to be x-negative.

Theorem 4.15. *Let P, Q, and R be x-negative.*

1. $(\nu x)(P|Q|!x(y).R) \sim (\nu x)(P|!x(y).R)|(\nu x)(Q|!x(y).R)$.
2. $(\nu x)(!P|!x(y).R) \sim !(\nu x)(P|!x(y).R)$.

Theorem 4.15 is used in the later section, where we consider convergence and divergence of processes. In the section, resources are considered to be environments in which a program is evaluated.

So far, we have been mainly interested in symmetric relations. However, the following relation, which may also be considered to be a sort of "equivalence" in some weak sense, is asymmetric. More precisely, it is a hypersimulation and the inverse of it is a hypersimulation up to internal actions.

Definition 4.16. *A relation $\mathcal{R}$ is an* expansion *if $\mathcal{R}$ is closed under substitutions and whenever $(P, Q) \in \mathcal{R}$ and $\mathrm{bn}\,\alpha \cap (\mathrm{fn}\,P \cup \mathrm{fn}\,Q) = \emptyset$,*

1. *$P \xrightarrow{\alpha} P'$ implies that there exists Q' such that $Q \overset{\alpha}{\Rightarrow} Q'$ and $(P', Q') \in \mathcal{R}$,*
2. *$Q \xrightarrow{\alpha} Q'$ implies that there exists P' such that $P \xrightarrow{\widehat{\alpha}} P'$ and $(P', Q') \in \mathcal{R}$,*

where $\overset{\alpha}{\Rightarrow}$ and $\xrightarrow{\widehat{\tau}}$ denote $\xrightarrow{\tau}{}^* \cdot \xrightarrow{\alpha} \cdot \xrightarrow{\tau}{}^*$ and $\xrightarrow{\tau}{}^?$ respectively, and $\xrightarrow{\widehat{\alpha}}$ denotes $\xrightarrow{\alpha}$ for $\alpha \neq \tau$.

We say that Q *expands* P, written $Q \succeq P$, if $(P, Q) \in \mathcal{R}$ for some expansion $\mathcal{R}$.

Definition 4.17. *A reflexive and transitive relation $\mathcal{R}$ on processes is said to be a* precongruence relation *if $\mathcal{R}$ is compatible.*

Theorem 4.18. *The relation $\succeq$ is a precongruence relation.*

5 Correspondences to Functional Languages

The call-by-name and the call-by-value lambda-calculi are lambda-calculi whose evaluation strategies are fixed. It is important to gain knowledge of these calculi since strictness of their evaluation strategies makes them pragmatic. As part of it, Milner discovered that the call-by-name, the call-by-value, and the strong call-by-value lambda-calculi can be embedded into his original pi-calculus with the construction [14]. Here, it is of theoretical interest to bring up the strong call-by-name lambda-calculus whose evaluation strategy is more loose than that of the call-by-name lambda-calculus, for the following facts are well-known: (1) lambda-definable functions in the lambda-calculus with full reductions are *computable* and vice versa (*Church's thesis*); (2) the definition of lambda-definability depends on whether can be reached a normal form in the lambda-calculus with full reductions, called *normalizability*; and (3) a normalizable term is also normalizable in the strong call-by-name lambda-calculus but not in other three ones (*the standardization theorem* [8]).

In the following, our goal is that not only the three calculi as described but also the strong call-by-name lambda-calculus can be embedded into the Mpi-calculus. The achievement immediately induces:

1. name-passing: the fundamental operation on concurrent systems is so expressive as to simulate the so-called beta-reduction on lambda calculi;

2. the expressive power of the Mpi-calculus is equal to or more than that of Turing machine.

In fact, it is obvious that the call-by-name, the call-by-value, and the strong call-by-value lambda-calculi can be embedded into our system with respect to convergence and divergence between reaction and reduction, since Milner has already shown that these three calculi can be embedded into his original pi-calculus and we have already proved that our system is conservative to Milner's original pi-calculus in Section 3. Therefore, only the strong call-by-name lambda-calculus remains.

5.1 The Strong Call-by-Name Lambda-Calculus

We define lambda-terms and reduction rules in the strong call-by-name lambda-calculus as follows:

$$M, N, \ldots ::= x \mid \lambda x.M \mid MN$$

$$(\lambda x.M)N \to_\lambda [N/x]M \qquad \frac{M \to_\lambda M'}{\lambda x.M \to_\lambda \lambda x.M'} \qquad \frac{M \to_\lambda M'}{MN \to_\lambda M'N}$$

and borrow various notation and terminology of lambda-calculi from Barendregt's encyclopedic book [3]. The difference from the call-by-name lambda-calculus is the existence of the second rule. The rule permits a function to reduce to one which returns the same output as it for any input, that is, it enforces extensionality of functions. We prove the following lemma in advance.

Lemma 5.1. *If $M \to_\lambda M'$, then $[N/x]M \to_\lambda [N/x]M'$.*

Proof. By induction on $M \to_\lambda M'$. □Next, we introduce a translation from lambda-terms into Mpi-processes. This translation reflects the lambda-reduction to the relation $\xrightarrow{\tau}$ up to the structural congruence relation $\equiv$, which is called *reaction* and denoted by $\to$. To show this, we assume that the Mpi-calculus has polyadicity [15], that is, it permits a fixed number of names along one port. For instance,

$$\overline{x}yz.P|x(vw).Q \to P|[y/v, z/w]Q$$

where it is assumed that $[y/v, z/w]$ is not sensitive to the order of substitutions.

We give a translation from lambda-terms into Mpi-processes:

$$\begin{aligned}
[\![x]\!]_u &= \overline{x}u.0 \\
[\![\lambda x.M]\!]_u &= u(xv).[\![M]\!]_v \\
[\![MN]\!]_u &= (\nu v)([\![M]\!]_v | (\nu v')(\overline{v}v'u | [\![v' := N]\!])) \\
[\![v' := N]\!] &= !v'(v'').[\![N]\!]_{v''}
\end{aligned}$$

which is a modification of Milner's call-by-name translation [14]. Both translations are based on the notion that functions are programs and their arguments are environments in which they are evaluated. These ones translate a function (i.e., a body of a program) and its argument into an input process and its resource, respectively. Ours differs from Milner's only in lambda-abstraction. We adopt input processes under consideration of causal dependency while Milner did guarded input processes. The reader sees that unguardedness of our input processes to internal actions makes it to simulate extensionality of functions.

In the following, we often omit an index of translations when it is clear from the context.

Lemma 5.2. $[\![(\lambda x.M)N]\!] \to \cdot \geq [\![[N/x]M]\!]$.

Proof. We may assume $x \notin \mathrm{fv}\, N$. Since we have:

$$\begin{aligned}[\![(\lambda x.M)N]\!]_u &= (\nu v)(v(xv'').[\![M]\!]_{v''} | (\nu v')(\overline{v}v'u | [\![v' := N]\!])) \\ &\to (\nu x)([\![M]\!]_u | [\![x := N]\!]) \ ,\end{aligned}$$

it is enough to prove:

$$(\nu x)([\![M]\!]_u | [\![x := N]\!]) \geq [\![[N/x]M]\!]_u \ .$$

We show the above by induction on the structure of M.

Case 1. Suppose that M is x. We have:

$$\begin{aligned}(\mathrm{LHS}) &= (\nu x)(\overline{x}u.0 | [\![x := N]\!]) \\ &\to (\nu x)([\![N]\!]_u | [\![x := N]\!]) \\ &\equiv [\![N]\!]_u | (\nu x)[\![x := N]\!] \sim (\mathrm{RHS}) \ .\end{aligned}$$

Case 2. Suppose that M is y distinct from x. We have:

$$\begin{aligned}(\mathrm{LHS}) &= (\nu x)(\overline{y}u | [\![x := N]\!]) \\ &\equiv \overline{y}u.0 | (\nu x)[\![x := N]\!] \sim (\mathrm{RHS}) \ .\end{aligned}$$

Case 3. Suppose that M is $\lambda y.M_0$ where $y \neq x$. We have:

$$\begin{aligned}(\mathrm{LHS}) &= (\nu x)(u(yv).[\![M_0]\!]_v | [\![x := N]\!]) \\ &\sim u(yv).(\nu x)([\![M_0]\!]_v | [\![x := N]\!]) \\ &\geq u(yv).[\![[N/x]M_0]\!]_v && (\because \text{induction hypothesis}) \\ &= [\![\lambda y.[N/x]M_0]\!]_u = (\mathrm{RHS}) \ .\end{aligned}$$

Case 4. Suppose that M is $M_1 M_2$. Theorem 4.15 induces:

$$\begin{aligned}(\mathrm{LHS}) &= (\nu x)((\nu v)([\![M_1]\!]_v | (\nu v')(\overline{v}v'u | [\![v' := M_2]\!])) | [\![x := N]\!]) \\ &\sim (\nu v)((\nu x)([\![M_1]\!]_v | [\![x := N]\!]) | (\nu x)(\nu v')(\overline{v}v'u | [\![v' := M_2]\!]) | [\![x := N]\!]) \\ &\sim (\nu v)((\nu x)([\![M_1]\!]_v | [\![x := N]\!]) | (\nu v')(\overline{v}v'u | !(\nu x)(v'(v'').[\![M_2]\!]_{v''} | [\![x := N]\!]))) \\ &\sim (\nu v)((\nu x)([\![M_1]\!]_v | [\![x := N]\!]) | (\nu v')(\overline{v}v'u | !v'(v'').(\nu x)([\![M_2]\!]_{v''} | [\![x := N]\!]))) \\ &\geq (\nu v)([\![[N/x]M_1]\!]_u | (\nu v')(\overline{v}v'u | !v'(v'').[\![[N/x]M_2]\!]_{v''})) && (\because \text{induction hypothesis}) \\ &= [\![[N/x]M_1[N/x]M_2]\!]_u = (\mathrm{RHS}) \ .\end{aligned}$$

□

Theorem 5.3. *If $M \to_\lambda M'$, then $[\![M]\!] \to \cdot \geq [\![M']\!]$.*

Proof. By induction on $M \to_\lambda M'$.

Case 1. The base step has already proved by Lemma 5.2.

Case 2. Suppose that $\lambda x_0.M_0 \to_\lambda \lambda x_0.M_0'$ is inferred from $M_0 \to_\lambda M_0'$. Induction hypothesis derives $[\![M_0]\!]_v \to \cdot \geq [\![M_0']\!]_v$. Then,

$$\begin{aligned}[\![M]\!]_u &= u(x_0 v).[\![M_0]\!]_v \\ &\to \cdot \geq u(x_0 v).[\![M_0']\!]_v = [\![M']\!]_u \ .\end{aligned}$$

Case 3. Suppose $M_1 M_2 \to_\lambda M_1' M_2$ is inferred from $M_1 \to_\lambda M_1'$. By induction hypothesis, we get $[\![M_1]\!]_v \to \cdot \geq [\![M_1']\!]_v$. Then,

$$\begin{aligned}[\![M]\!]_u &= (\nu v)([\![M_1]\!]_v | (\nu v')(\overline{v} v' u | [\![v' := M_2]\!])) \\ &\to \cdot \geq (\nu v)([\![M_1']\!]_v | (\nu v')(\overline{v} v' u | [\![v' := M_2]\!])) = [\![M']\!]_u \ .\end{aligned}$$ □

For convenience, we adopt the following notation. A term M is said to be a *normal form*, written $M \not\to_\lambda$, if there exists no term to which M reduces. A term M *converges on* M', written $M \downarrow_\lambda M'$, if M reduces to M' in arbitrary finite steps and M' is a normal form. A term M *converges* if there exists a term on which M converges. A term M *diverges* if there exists an infinite reduction sequence from M. We also use similar terminology and notation for processes and reactions $\to$.

The above definitions immediately derive the following simple proposition as a corollary of Theorem 5.3.

Corollary 5.4. *If M diverges, then $[\![M]\!]$ also diverges.*

This corollary claims that the strong call-by-name lambda-calculus to be sound to the Mpi-calculus in divergence. Although this says nothing in convergence, we have:

Theorem 5.5. *If M is a normal form, so is $[\![M]\!]$.*

There exists a correspondence once it becomes a normal form. It is, therefore, natural that we introduce a relationship between reduction and reaction connected more directly. This is done in Subsection 5.3.

5.2 Explicit Environment

In the previous subsection we saw that lambda-reduction $\to_\lambda$ reflects to reaction $\to$. However, it has indeed a gap of $\geq$. For the purpose of looking for the source of this gap, we treat lambda-reduction more delicately.

A function f receives an argument a for a parameter x in f, and becomes an output itself. In more detail, when it is decided that f receives a for x, an environment that evaluate x in f as a is constructed, and f is evaluated in the environment until an output is needed. A lambda-calculus in which these are built is said having *explicit environment*, and was first considered by Abadi et al. [1]. Many lambda-calculi have been considered since the seminal work of Abadi et al. For instance, λexp, λxgc, λs, $\lambda\nu$, and $\lambda\varepsilon$, see [10, 5], [6], [11], [4], and [22] respectively.

We define the simplest strong call-by-name lambda-calculus with explicit environment as follows:

$$M, N, \ldots ::= x \mid \lambda x.M \mid MN \mid \{N/x\}M$$

$$(\lambda x.M)N \to_\beta \{N/x\}M \qquad \frac{M \to_\beta M'}{\lambda x.M \to_\beta \lambda x.M'} \qquad \frac{M \to_\beta M'}{MN \to_\beta M'N}$$

$$\{M/x\}x \to_\varepsilon M \qquad \frac{x \notin \{y\} \cup \mathrm{fv}\, N}{\{N/y\}\lambda x.M \to_\varepsilon \lambda x.\{N/y\}M}$$

$$\{N/x\}(M_1 M_2) \to_\varepsilon \{N/x\}M_1\{N/x\}M_2 \qquad \frac{x \notin \mathrm{fv}\, M}{\{N/x\}M \to_\varepsilon M}$$

$$\frac{M \to_\varepsilon M'}{\lambda x.M \to_\varepsilon \lambda x.M'} \qquad \frac{M \to_\varepsilon M'}{MN \to_\varepsilon M'N} \qquad \frac{M \to_\varepsilon M'}{NM \to_\varepsilon NM'} \ .$$

Ordinary lambda-calculi with explicit environment have reductions for environment, e.g. for combining two environments into one environment. Here, explicit environments are used only as a buffer between lambda-reducible terms. It is, therefore, obvious that there exists no infinite sequence of $\to_\varepsilon$. Notice that $\{N/x\}M$ is not $[N/x]M$ but one expression of such a form. We extend the previous translation for a term with environment:

$$[\![\{N/x\}M]\!]_u = (\nu x)([\![M]\!]_u|[\![x \coloneqq N]\!]) \ .$$

These notions divide Theorem 5.3 into the following two theorems.

Theorem 5.6. *If $M \to_\beta M'$, then $[\![M]\!] \to [\![M']\!]$.*

Theorem 5.7. *If $M \to_\varepsilon M'$, then $[\![M]\!] \geq [\![M']\!]$.*

Since we regard a reaction as a transition performed automatically in the concurrent system, Theorem 5.6 suggests that the Mpi-calculus has a mechanism that assigns an argument to an address of a function. However, we consider $\geq$ to be a meta-relation on processes. The Mpi-calculus, therefore, distinguishes a function with an environment from the result of evaluating the function in the environment.

5.3 Full Correspondence

So far, we gave a translation from lambda-terms into Mpi-processes and extended it to one from lambda-terms with explicit environment to Mpi-processes. Notice that we can rewrite a lambda-term to some lambda-terms with explicit environment equal to it; for instance, $\lambda x.y$ can be expressed as $[y/z](\lambda x.z)$, $[\lambda x.y/z]z$, and so on. Therefore, the translation $[\![\cdot]\!]$ from lambda-terms into Mpi-processes can be extended to a multi-valued function. Formally, $[\![L]\!]_u$ is defined as the set:

$$\{\, (\nu \boldsymbol{x})([\![M]\!]_u|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \mid \mathrm{fv}\, N_i \cap \{x_1, \ldots, x_i\} = \emptyset \text{ for } 1 \leq i \leq n, \text{ and } [\boldsymbol{N}/\boldsymbol{x}]M = L \,\}$$

where $(\nu \boldsymbol{x})$, $[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]$, and $[\boldsymbol{N}/\boldsymbol{x}]M$ denote $(\nu x_1)\cdots(\nu x_n)$, $[\![x_1 \coloneqq N_1]\!]|\cdots|[\![x_n \coloneqq N_n]\!]$, and $[N_n/x_n]\cdots[N_1/x_1]M$ respectively. In the sequel, when we say $[\![L]\!]$, we refer to some element of $[\![L]\!]$.

Lemma 5.8. *If $[\![L]\!] \sim P$ and $L \to_\lambda L'$, then $P \to^+ \cdot \sim [\![L']\!]$.*

Proof. By induction on derivation of lambda-reduction.

Case 1. Suppose that L is $[\boldsymbol{N}/\boldsymbol{x}]((\lambda x_0.M_0)M_1)$ and L' is $[[\boldsymbol{N}/\boldsymbol{x}]M_1/x_0][\boldsymbol{N}/\boldsymbol{x}]M_0$. We remark that $[[\boldsymbol{N}/\boldsymbol{x}]M_1/x_0][\boldsymbol{N}/\boldsymbol{x}]M_0$ is $[\boldsymbol{N}/\boldsymbol{x}][M_1/x_0]M_0$ since we may assume $x_0 \notin \mathrm{fv}\, \boldsymbol{N}$.

$$\begin{aligned}
[\![L]\!]_u &= (\boldsymbol{\nu x})([\![(\lambda x_0.M_0)M_1]\!]_u|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&\to^+ (\boldsymbol{\nu x})((\nu x_0)([\![M_0]\!]_u|[\![x_0 \coloneqq M_1]\!])|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&= [\![[\boldsymbol{N}/\boldsymbol{x}][M_1/x_0]M_0]\!]_u = [\![L']\!]_u \ .
\end{aligned}$$

Case 2. Suppose that L is $[\boldsymbol{N}/\boldsymbol{x}](x_i M_1)$ and L' is $[[\boldsymbol{N}/\boldsymbol{x}]M_1/x_0][\boldsymbol{N}/\boldsymbol{x}]M_0$ where N_i is $\lambda x_0.M_0$. Notice that $[[\boldsymbol{N}/\boldsymbol{x}]M_1/x_0][\boldsymbol{N}/\boldsymbol{x}]M_0$ is $[\boldsymbol{N}/\boldsymbol{x}][M_1/x_0]M_0$ since $x_0 \notin \mathrm{fv}\, \boldsymbol{N}$.

$$\begin{aligned}
[\![L]\!]_u &= (\boldsymbol{\nu x})([\![x_i M_1]\!]_u|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&= (\boldsymbol{\nu x})((\nu v)(\overline{x}v|(\nu v')(\overline{v}v'u|[\![v' \coloneqq M_1]\!]))|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&\to^+ (\boldsymbol{\nu x})((\nu v)([\![\lambda x_0.M_0]\!]_v|(\nu v')(\overline{v}v'u|[\![v' \coloneqq M_1]\!]))|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!])
\end{aligned}$$

$$
\begin{aligned}
&= (\boldsymbol{\nu x})((\nu v)(v(x_0 v''):[\![M_0]\!]_{v''}|(\nu v')(\overline{v} v' u|[\![v' \coloneqq M_1]\!]))|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&\to (\boldsymbol{\nu x})(\nu x_0)(([\![M_0]\!]_u|[\![x_0 \coloneqq M_1]\!])|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&= [\![[\boldsymbol{N}/\boldsymbol{x}][M_1/x_0]M_0]\!]_u = [\![L']\!]_u \ .
\end{aligned}
$$

Case 3. Suppose that L is $[\boldsymbol{N}/\boldsymbol{x}]x_i$ and L' is

$$
[[t_{i+1}, \ldots, t_n/x_{i+1}, \ldots, x_n]M_1/x_0][t_{i+1}, \ldots, t_n/x_{i+1}, \ldots, x_n]M_0 \ .
$$

where N_i is $(\lambda x_0.M_0)M_1$. Notice that L' is $[\boldsymbol{N}/\boldsymbol{x}][M_1/x_0]M_0$ since $x_0 \notin \mathrm{fv}\,\boldsymbol{N}$ and $\{x_1, \ldots, x_i\} \cap (\mathrm{fv}\,M_0 \cup \mathrm{fn}(M_1)) = \emptyset$

$$
\begin{aligned}
[\![L]\!]_u &= (\boldsymbol{\nu x})([\![x_i]\!]_u|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&= (\boldsymbol{\nu x})(\overline{x_i} u|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&\to^+ (\boldsymbol{\nu x})([\![(\lambda x_0.M_0)M_1]\!]_u|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&= (\boldsymbol{\nu x})((\nu v)(v(x_0 v''):[\![M_0]\!]_{v''}|(\nu v')(\overline{v} v' u|[\![v' \coloneqq M_1]\!]))|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&\to (\boldsymbol{\nu x})((\nu x_0)([\![M_0]\!]_u|[\![x_0 \coloneqq M_1]\!])|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&= [\![[\boldsymbol{N}/\boldsymbol{x}][M_1/x_0]M_0]\!]_u = [\![L']\!]_u \ .
\end{aligned}
$$

Case 4. Suppose that $L \to_\lambda L'$ is inferred from $[\boldsymbol{N}/\boldsymbol{x}]M_i \to_\lambda M_i'$ where L, L', and N_i are $[\boldsymbol{N}/\boldsymbol{x}]x_i$, $\lambda x_0.M_i'$, and $\lambda x_0.M_i$, respectively.

$$
\begin{aligned}
[\![L]\!]_u &= (\boldsymbol{\nu x})([\![x_i]\!]_u|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&= (\boldsymbol{\nu x})(\overline{x_i} u|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&\to (\boldsymbol{\nu x})([\![\lambda x_0.M_i]\!]_u|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&= (\boldsymbol{\nu x})(u(x_0 v):[\![M_i]\!]_v|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&\sim u(x_0 v):(\boldsymbol{\nu x})([\![M_i]\!]_v|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&\to^+ \cdot \sim u(x_0 v):[\![M_i']\!]_v && (\because \text{induction hypothesis}) \\
&= [\![\lambda x_0.M_i']\!] = [\![L']\!]_u \ .
\end{aligned}
$$

Case 5. Suppose that $L \to_\lambda L'$ is inferred from $[\boldsymbol{N}/\boldsymbol{x}]M_1 \to_\lambda M_1'$ where L, L', and N_i are $[\boldsymbol{N}/\boldsymbol{x}]x_i$, $M_1'[\boldsymbol{N}/\boldsymbol{x}]M_2$, and $M_1 M_2$. Notice that $[\boldsymbol{N}/\boldsymbol{x}](M_1' M_2)$ is $M_1'[\boldsymbol{N}/\boldsymbol{x}]M_2$ since $\boldsymbol{x} \cap \mathrm{fv}\,M_1' = \emptyset$.

$$
\begin{aligned}
[\![L]\!]_u &= (\boldsymbol{\nu x})([\![x_i]\!]_u|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&= (\boldsymbol{\nu x})(\overline{x_i} u|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&\to (\boldsymbol{\nu x})([\![M_1 M_2]\!]_u|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&= (\boldsymbol{\nu x})((\nu v)([\![M_1]\!]_v|(\nu v')(\overline{v} v' u|[\![v' \coloneqq M_2]\!]))|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&\sim (\nu v)((\boldsymbol{\nu x})([\![M_1]\!]_v|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!])|(\boldsymbol{\nu x})((\nu v')(\overline{v} v' u|[\![v' \coloneqq M_2]\!])|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!])) \\
&\to^+ \cdot \sim (\nu v)([\![M_1']\!]_v|(\boldsymbol{\nu x})((\nu v')(\overline{v} v' u|[\![v' \coloneqq M_2]\!])|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!])) && (\because \text{induction hypothesis}) \\
&\equiv (\boldsymbol{\nu x})(\nu v)([\![M_1']\!]_v|(\nu v')((\overline{v} v' u|[\![v' \coloneqq M_2]\!])|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!])) \\
&= [\![[\boldsymbol{N}/\boldsymbol{x}](M_1' M_2)]\!] = [\![L']\!]_u \ .
\end{aligned}
$$

Case 6. Suppose that $L \to_\lambda L'$ is inferred from $[\boldsymbol{N}/\boldsymbol{x}]M_0 \to_\lambda s_0'$ where L is $[\boldsymbol{N}/\boldsymbol{x}]\lambda x_0.M_0$ and L' is $\lambda x_0.s_0'$.

$$
\begin{aligned}
[\![L]\!]_u &= (\boldsymbol{\nu x})(u(x_0 v):[\![M_0]\!]_v|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!]) \\
&\sim u(x_0 v):(\boldsymbol{\nu x})([\![M_0]\!]_v|[\![\boldsymbol{x} \coloneqq \boldsymbol{N}]\!])
\end{aligned}
$$

$$\begin{aligned}
&\to^+ \cdot \sim u(x_0 v):[\![s_0']\!]_v && (\because \text{induction hypothesis})\\
&= [\![\lambda x_0.s_0']\!] = [\![L']\!]_u \ .
\end{aligned}$$

Case 7. Suppose that $L \to_\lambda L'$ is inferred from $[\boldsymbol{N}/\boldsymbol{x}]M_1 \to_\lambda M_1'$ where L is $[\boldsymbol{N}/\boldsymbol{x}](M_1 M_2)$ and L' is $M_1'[\boldsymbol{N}/\boldsymbol{x}]M_2$. Notice that $[\boldsymbol{N}/\boldsymbol{x}](M_1' M_2)$ is $M_1'[\boldsymbol{N}/\boldsymbol{x}]M_2)$ since $\boldsymbol{x} \cap \mathrm{fv}\, M_1' = \emptyset$.

$$\begin{aligned}
[\![L]\!]_u &= (\nu \boldsymbol{x})((\nu v)([\![M_1]\!]_v|(\nu v')(\overline{v}v'u|[\![v' := M_2]\!]))|[\![\boldsymbol{x} := \boldsymbol{N}]\!])\\
&\sim (\nu v)((\nu \boldsymbol{x})([\![M_1]\!]_v|[\![\boldsymbol{x} := \boldsymbol{N}]\!])|(\nu \boldsymbol{x})((\nu v')(\overline{v}v'u|[\![v' := M_2]\!])|[\![\boldsymbol{x} := \boldsymbol{N}]\!]))\\
&\to^+ \cdot \sim (\nu v)([\![M_1']\!]_v|(\nu \boldsymbol{x})((\nu v')(\overline{v}v'u|[\![v' := M_2]\!])|[\![\boldsymbol{x} := \boldsymbol{N}]\!])) \quad (\because \text{induction hypothesis})\\
&\equiv (\nu \boldsymbol{x})(\nu v)([\![M_1']\!]_v|(\nu v')((\overline{v}v'u|[\![v' := M_2]\!])|[\![\boldsymbol{x} := \boldsymbol{N}]\!]))\\
&= [\![[\boldsymbol{N}/\boldsymbol{x}](M_1' M_2)]\!] = [\![L']\!]_u \ . \qquad \square
\end{aligned}$$

Lemma 5.9. *Let α be the first action except τ with which $[\![L]\!]_u$ transits. Then u belongs to* $\mathrm{fn}(\alpha)$*. Moreover if L is not an abstraction, then α is an output action.*

Proof. Let L be $[\boldsymbol{N}/\boldsymbol{x}]M$. By induction of the structure of L. By cases on the form of M.

Case 1. It is obvious in case M is a variable.

Case 2. Suppose that M is $\lambda x_0.M_0$. $[\![L]\!]_u$ is $(\nu \boldsymbol{x})(u(x_0 v):[\![M_0]\!]_v)|[\![\boldsymbol{x} := \boldsymbol{N}]\!])$. By induction hypothesis, $[\![M_0]\!]_v$ cannot transits without invoking an action related to v except τ. Therefore, it holds in this case.

Case 3. Suppose that M is $M_1 M_2$. Then $[\![L]\!]_u$ is$(\nu \boldsymbol{x})((\nu v)([\![M_1]\!]_v|(\nu v')(\overline{v}v'u|[\![v' := M_2]\!]))|[\![\boldsymbol{x} := \boldsymbol{N}]\!])$. By induction hypothesis, $[\![M_1]\!]_v$ cannot transits without invoking an action related to v except τ. Therefore, it also holds in this case. The second assertion also depends on this proof. □

Lemma 5.10. *If $[\![L]\!] \to P$, then $L \to_\lambda^* L'$ and $[\![L']\!] = P$ for some L'.*

Proof. Let L be $[\boldsymbol{N}/\boldsymbol{x}]M$. By induction on the inference of $[\![L]\!] \to P$.

Case 1. Suppose that M is $\lambda x_0.M_0$. Then P is $(\nu \boldsymbol{x})(u(x_0 v):P_0|[\![\boldsymbol{x} := \boldsymbol{N}]\!])$ where $[\![M_0]\!]_v \to P_0$ by Lemma 5.9. By induction hypothesis, there exists L_0 such that $M_0 \to_\lambda^* L_0$ and $[\![L_0]\!]_v = P_0$. By Lemma 5.1, we have $L \to_\lambda^* [\boldsymbol{N}/\boldsymbol{x}]\lambda x_0.L_0$. It is sufficient since $[\![[\boldsymbol{N}/\boldsymbol{x}]\lambda x_0.L_0]\!]_u = P$.

Case 2. Suppose that M is $M_1 M_2$ and P is $(\nu \boldsymbol{x})((\nu v)(P_1|(\nu v')(\overline{v}v'u|[\![v' := M_2]\!]))|[\![\boldsymbol{x} := \boldsymbol{N}]\!])$ where $[\![M_1]\!]_v \to P_1$. By induction hypothesis, there exists L_1 such that $M_1 \to_\lambda^* L_1$ and $[\![L_1]\!]_v = P_1$. The rest is similar to Case 1.

Case 3. Suppose that M is $(\lambda x_0.M_0)M_2$ and P is $(\nu \boldsymbol{x})((\nu x_0)([\![M_0]\!]_u|[\![x_0 := M_2]\!])|[\![\boldsymbol{x} := \boldsymbol{N}]\!])$. We have $L \to_\lambda [\boldsymbol{t}/\boldsymbol{x}][M_2/x_0]M_0$ by applying Lemma 5.1 to $(\lambda x_0.M_0)M_2 \to_\lambda [M_2/x_0]M_0$. It is sufficient since $x_0 \notin \mathrm{fn}(\boldsymbol{N})$ implies $[\![[\boldsymbol{N}/\boldsymbol{x}][M_2/x_0]M_0]\!]_u = P$.

Case 4. Suppose that M is $x_i M_1 \cdots M_m$ $(m \geq 1)$ and P is

$$(\nu \boldsymbol{x})((\nu v_m)(\cdots([\![N_i]\!]_{v_1}|\cdots|(\nu v_m')(\overline{v_m}v_m'u|[\![v' := M_m]\!])))|[\![\boldsymbol{x} := \boldsymbol{N}]\!]) \ .$$

Since L is also $[\boldsymbol{N}/\boldsymbol{x}](N_i M_1 \cdots M_m)$, we have $[\![L]\!]_u = P$. This suffices to check this case.

Case 5. The case of $M = x$ is similar to Case 4 since this is is the one of $m = 0$ in Case 4. □

Lemma 5.11. *If $[\![L]\!] \sim P$ and $L \not\to_\lambda$, then $P \downarrow P'$ for some P'.*

Proof. Let L be $[\boldsymbol{N}/\boldsymbol{x}]M$. We prove this lemma by double induction of the structure of L and the number of substitutions.

Case 1. Suppose that M is x_i. We immediately get

$$[\![L]\!]_u \to (\boldsymbol{\nu x})([\![N_i]\!]_u | [\![\boldsymbol{x} := \boldsymbol{N}]\!]) =: Q \ .$$

Since $L = [N_n/x_n] \cdots [N_{i+1}/x_{i+1}]N_i$, there exists Q' such that $Q \downarrow Q'$ by induction hypothesis. Therefore, there exists P' such that $P \downarrow P'$.

Case 2. It is obvious when M is a variable distinct from $\boldsymbol{x}$.

Case 3. Suppose that M is $\lambda x_0.M_0$. By induction hypothesis, there exists Q such that $[\![M_0]\!]_v \downarrow Q$. Then

$$[\![L]\!]_u = (\boldsymbol{\nu x})(u(x_0 v).[\![M_0]\!]_v | [\![\boldsymbol{x} := \boldsymbol{N}]\!]) \to^* (\boldsymbol{\nu x})(u(x_0 v).Q | [\![\boldsymbol{x} := \boldsymbol{N}]\!]) =: Q' \ .$$

By Lemma 5.9, this Q' is a normal form. Therefore, there exists P' such that $P \downarrow P'$.

Case 4. Suppose that M is $M_1 M_2$. By induction hypothesis, there exists Q such that $[\![M_1]\!]_v \downarrow Q$. Then

$$\begin{aligned}[\![L]\!]_u &= (\boldsymbol{\nu x})((\nu v)([\![M_1]\!]_v | (\nu v')(\overline{v} v' u | [\![v' := M_2]\!])) | [\![\boldsymbol{x} := \boldsymbol{N}]\!]) \\ &\to^* (\boldsymbol{\nu x})((\nu v)(Q | (\nu v')(\overline{v} v' u | [\![v' := M_2]\!])) | [\![\boldsymbol{x} := \boldsymbol{N}]\!]) =: Q' \ .\end{aligned}$$

Since M is a normal form, M_1 is not an abstraction. Then Q' is a normal form by Lemma 5.9. Therefore, there exists P' such that $P \downarrow P'$. □

Theorem 5.12. *Let L and L' be lambda-terms, and P and P' be Mpi-processes.*

1. *If $L \downarrow_\lambda L'$, then for each element P of $[\![L]\!]$ there exists P' such that $P \downarrow P'$.*
2. *If $[\![L]\!] \downarrow P$, then $L \downarrow_\lambda L'$ for some L'.*
3. *L converges if and only if $[\![L]\!]$ does.*
4. *L diverges if and only if $[\![L]\!]$ does.*

Proof. First, there exists P' such that $[\![L]\!] \to^+ P'$ and $[\![L']\!] \sim P'$ by Lemma 5.8. There exists P such that $P' \downarrow P$ by Lemma 5.11. Hence, 1 holds. Secondly, there exists L'' such that $L \to_\lambda^* L''$ and $[\![L']\!] = P$ by Lemma 5.10. Since P is a normal form, L' is a normal form by Lemma 5.8. Hence, 2 holds. Finally, 3 and 4 follow from 1, 2, and the proof of Lemma 5.10. □

Notice that all the processes which belong to $[\![L]\!]$ coincide in terms of termination property, and as a result L and $[\![L]\!]$ in Subsection 5.1 converge or diverge simultaneously. It is, however, not surprising since we have extended Milner's call-by-name translation into the multi-valued function for this purpose.

The accordance in convergence and divergence claims that the Mpi-calculus contains the strong call-by-name lambda-calculus as a subsystem. Moreover, we can give an intuitive explanation to Milner's call-by-name translation, because extensionality of lambda-terms is translated into optimization of Mpi-processes, i.e.,

$$\frac{M \to_\lambda M'}{\lambda x.M \to_\lambda \lambda x.M'} \text{ corresponds to } \frac{[\![M]\!]_v \xrightarrow{\tau} [\![M']\!]_v \qquad \{u\} \cap \emptyset = \emptyset \qquad \{x, v\} \cap \emptyset = \emptyset}{u(xv):[\![M]\!]_v \xrightarrow{\tau} u(xv):[\![M']\!]_v} .$$

In addition, generality of the Mpi-calculus enables it to have various lambda-calculi: three calculi except the strong call-by-name lambda-calculus as described in Subsection 5.1. We conclude that the Mpi-calculus is not only simple and natural but also expressible and general.

Fu [9], Merro and Sangiorgi [12], Parrow and Victor [17, 18], and other researchers have given sound embeddings of the strong call-by-name lambda-calculus into their systems, and most of them have not showed their embeddings to be complete. In contrast, Parrow and Victor designed a concurrent system called *fusion calculus* using unique ideas, and showed that their embedding is not only sound but also complete to the strong call-by-name lambda-calculus [18]. Their system is based on the notion of equations of names, and properly includes the pi-calculus. Their system also has an advantage in the way that they deal with names. In their system, there exists no concept of binding names on input actions. As a result, it can avoid troublesome procedures for dealing with bound names, which can become tedious. Certainly, their system is simple and expressive. However, their study on concurrency does not directly give insights into the pi-calculus, the de facto standard concurrent system, since their system is quite unique. On the other hand, our system is directly connected to the pi-calculus with which many researchers are familiar. We could thus give them a lot of intuitive ideas. For instance, we can claim that "environments in functional languages are resources in a concurrent system" in Subsection 5.2 and that "a concurrent system can have the notion of application and extensionality of functions" in this subsection.

6 Conclusion and Future Work

We have introduced a concurrent system based on the concept of multi-ports, and have defined various equivalences on processes. Our system is an extension of the pi-calculus, in the sense that the pi-calculus can be obtained from ours by restricting the number of ports. It is noteworthy that we extend the pi-calculus to our system not by adding but by replacing a rule, and that a process with only one port can be expressed as syntax sugar for a multi-ported process. Moreover, our system has the advantage that we can control permission of self-communication as described in detail in Section 3. Although we have given examples of this, it is left as future work to find various useful applications of multi-ported processes which cannot communicate with themselves.

Our system is more expressive than the original pi-calculus. As a witness, even the strong call-by-name lambda-calculus can be embedded into our system, while Milner embedded only lambda-calculi which do not have the extensional rule into the original pi-calculus. We treated reductions of lambda-calculi carefully in this paper. As a consequence, our embedding of the strong call-by-name lambda-calculus has resulted in a complete one with respect to convergence and divergence. To our knowledge, our system is the first one that is a natural extension of the pi-calculus and that achieves completeness of the above kind.

Although we have succeeded in giving a complete embedding of the strong call-by-name lambda-calculus, it remains an open question whether it is possible to give a complete embedding of the lambda-calculus with full reductions. Our success is partly due to the fact that there exists no reduction which applies to arguments:

$$\frac{M \to_\lambda M'}{NM \to_\lambda NM'}$$

in the strong call-by-name lambda-calculus. Since our translation translates arguments into resources, it is necessary to add a rule which corresponds to optimization of resources in the pi-calculus. A candidate for such a rule would be the following:

$$\frac{P \xrightarrow{\tau} P'}{!P \xrightarrow{\tau} !P'}$$

as some researchers indicate (and write off). However, it does not seem to be easy to introduce a notion of equivalence on processes in this system since this transition, that a process of finite copying transits to an optimized one trained infinitely, is essentially different from the others. We could think behavioral equivalence up to *infinite* internal transitions as a candidate of equivalence.

Acknowledgments

I would like to express my deepest gratitude to Masami Hagiya, Daisuke Kimura, Takafumi Sakurai, Izumi Takeuti, Makoto Tatsuta, Kazushige Terui, and Mitsuharu Yamamoto, who have worked closely with me on this paper. I am also very grateful to Carl Christian Frederiksen, Masahito Hasegawa, Kohei Honda, Yoshihiko Kakutani, Akitoshi Kawamura, and Richard Potter for providing useful advice.

References

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. In *Proceedings of POPL'90: the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 31–46, 1990.

[2] T. Abe. A concurrent system of multi-ported processes with causal dependency. In *Proceedings of APLAS'04: the 2nd Asian Symposium on Programming Languages and Systems*, volume 3302 of *Lecture Notes in Computer Science*, pages 146–162. Springer-Verlag, 2004.

[3] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, revised edition, 1984.

[4] Z.-E.-A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. λv, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, 1996.

[5] R. Bloo. Preservation of strong normalisation for explicit substitution. Computer Science Report 95-08, Eindhoven University of Technology, 1995.

[6] R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *Proceedings of CSN'95: Computer Science in the Netherlands*, 1995.

[7] M. Boreale and R. De Nicola. Testing equivalence for mobile processes. *Information and Computation*, 120:279–303, 1995.

[8] H. B. Curry, R. Feys, and W. Craig. *Combinatory Logic*, volume 1. North-Holland, 1958.

[9] Y. Fu. A proof-theoretical approach to communication. In *Proceedings of ICALP'97: the 24th International Colloquium on Automata, Languages, and Programming*, volume 1256 of *Lecture Notes in Computer Science*, pages 325–335. Springer-Verlag, 1997.

[10] F. Kamareddine and R. Nederpelt. On stepwise explicit substitution. *International Journal of Foundations of Computer Science*, 4(3):197–240, 1993.

[11] F. Kamareddine and A. Rios. A lambda-calculus à la De Bruijn with explicit substitutions. In *Proceedings of PLILP'95: the 7th International Symposium on Programming Languages, Implementations, Logics, and Programs*, volume 982 of *Lecture Notes in Computer Science*, pages 45–62. Springer-Verlag, 1995.

[12] M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. In *Proceedings of ICALP'98: the 25th International Colloquium on Automata, Languages, and Programming*, 1998.

[13] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[14] R. Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[15] R. Milner. The polyadic π-calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993.

[16] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100(1):1–77, 1992.

[17] J. Parrow and B. Victor. The update calculus. In *Proceedings of AMAST'97: the 6th International Conference on Algebraic Methodology and Software Technology*, volume 1349 of *Lecture Notes in Computer Science*, pages 409–423. Springer-Verlag, 1997.

[18] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of LICS'98: the 13th IEEE Symposium on Logic in Computer Science*, 1998.

[19] D. Sangiorgi. On the bisimulation proof method. *Journal of Mathematical Structures in Computer Science*, 8(5):447–479, 1998.

[20] D. Sangiorgi and R. Milner. The problem of 'weak bisimulation up to'. In *Proceedings of CONCUR'92: the 3rd International Conference on Concurrency Theory*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, 1992.

[21] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001. This book has a list of errata: `http://www.cs.unibo.it/~sangio/BOOK/errata.html`.

[22] M. Sato, T. Sakurai, and R. Burstall. Explicit environments. In *Proceedings of TLCA'99: the 4th International Conference on Typed Lambda Calculi and Applications*, volume 1581 of *Lecture Notes in Computer Science*, pages 340–354. Springer-Verlag, 1999.

[23] F. van Breugel. A labelled transition system for pi-epsilon-calculus. In *Proceedings of TAPSOFT'97: the 7th International Joint Conference on the Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 312–336. Springer, 1997.