Regular Paper

An Equational Relation for Ambient Calculus

Toru Kato†

Ambient calculus is a process algebra developed for describing mobile processes. Ambients represent the substances of movement and the fields of the ambients themselves. Having this hierarchy, it can model various kinds of mobile computation. Equational relation for ambient calculus "Contextual Equivalence" were proposed regarding the names of ambients observed from the environment. This relation is, however, not strong as "testing equivalence" so that it can identify the processes which have different properties. This paper proposes equational relations for ambient calculus by which we can distinguish processes that the existing equivalence identifies.

1. Introduction

This paper presents equational relations for *ambient calculus* [3]. Ambient calculus is a process algebra designed for describing mobile agents. Many approaches for expressing concurrent computation have been developed [1),5),8),9),13)] and some of them treat higher order processes [5),8)].

In the higher order framework, processes, the subjects of the computation, can be sent by other processes and keep computing in other locations. This framework can express the situation such that a process moves around on a network environment and keeps collecting information (mobile Web searching agent [16)]) or a process enters a field and the field itself is also a process (hierarchal mobile agent [12)]). Ambient calculus can model various kinds of those mobile computations by the hierarchy of ambients.

Reference 7) develops tools for proving equations for ambient calculus which is called *Contextual Equivalence*. Intuitively, two processes P and Q are contextually equivalent if, for any context $\mathcal{C}()$, the observable sets of the names of ambients in $\mathcal{C}(P)$ and $\mathcal{C}(Q)$ are the same. This equivalence would be finer than "may testing" because if ambient calculus had choice "+" as a primitive operator, it would distinguish $a[b[] + c[]]$ from $a[b[]] + a[c[]]$ by using the context $\mathcal{C}_e() \stackrel{def}{=} open\ a|-$. We will discus choice operators in Section 3.2.

Contextual equivalence is not, however, fine enough to distinguish simple processes which have different behaviors such as:

$$P_e \stackrel{def}{=} n[0].$$

$$Q_e \stackrel{def}{=} (\nu m)(m[n[0]]|m[0]|open\ m.0).$$

See Section 3.1 for details. Thus, this paper firstly defines an equivalence relation that is as fine as testing equivalence. We call it contextually testing equivalence.

By the way, ambient calculus dose not have the choice operator as a privative because it can be simulated using parallel and restricting operations. Reference 3) shows the example of choice macro that has several restrictions. So this paper defines another nondeterministic choice macro. Using the nondeterministic choice macro, we find there are contextually equivalent processes which have different properties that even the testing contextual equivalence identifies. Thus, this paper also introduces another equivalence relation to distinguish those processes using should testing notions [2)].

Reference 15) shows a different kind of choice example called Electoral System. Electoral System is a network of processes one of which will be chosen as a representative of the network. We will discus the similarity and the difference between our choice macro and the Electoral System in Section 5.

This paper is organized as follows: in Section 2, we review the syntax and reduction semantics of ambient calculus. Section 2 also shows several definitions for contextual equivalence. Section 3 explains the problem of contextual equivalence by introducing an external choice operator. Then, we introduce an equational relation and shows the properties of the relation in Section 4.

† Department of Informatics Kinki University

2. Ambient Calculus

This section describes the syntax and semantics of ambient calculus originally defined in Ref. 3). An expression $go(M)$ bellow is called objective moves which used to be defined as a macro expression using subjective moves (*in* M and *out* M). Reference 4) makes a little extension in syntax by adding objective moves as primitives which we utilize in this paper, so our syntax is the extended version. This section also reviews an equivalence relation of ambient calculus defined in Ref. 7).

2.1 Syntax and Operational Semantics

We assume there are infinite sets of *names* and *variables*, ranged over by m, n, p, q and x, y, z, respectively. *Expressions* and *processes* are ranged over by M, N and P, Q, R, respectively. We use the notation P_e ($_e$ means example), P_1 or P' for concrete processes in examples while we use P or Q as meta symbols in definitions.

Definition 2.1
(Expressions and Processes) [3),4)]

$M, N ::=$	expressions
x	variable
$\mid n$	name
$\mid in\ M$	can enter M
$\mid out\ M$	can exit M
$\mid open\ M$	can open M where M must be a variable or a name
$\mid go(M).P$	make a process P move along a path M where M consists of *in* n, *out* n and ϵ and P must be an ambient
$\mid \epsilon$	null
$\mid M.M'$	path
$P, Q, R ::=$	processes
$(\nu n)P$	restriction
$\mid 0$	inactivity
$\mid P\|Q$	composition
$\mid !P$	replication
$\mid M[P]$	ambient
$\mid M.P$	action
$\mid (x).P$	input
$\mid \langle M \rangle$	output

□

We use the following abbreviations:

- M for $M.0$
- $M[]$ for $M[0]$
- $(\nu\vec{p})P$ for $(\nu p_1), \ldots, (\nu p_k)P$ where $\vec{p} = p_1, \ldots, p_k$.

Let ϕ be an expression or a process, $fn(\phi)$ and $fv(\phi)$ be the sets of *free names* and *free variables* of ϕ. $\phi\{x \leftarrow M\}$ and $\phi\{n \leftarrow M\}$ are the outcomes of capture–avoiding substitutions of M for each free occurrence of the variable x and the name n respectively in ϕ.

Definition 2.2
(Structural Congruence: $P \equiv Q$ [3),4)]**)**

$P|Q \equiv Q|P$
$(P|Q)|R \equiv P(Q|R)$
$!P \equiv P|!P$
$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$
$n \notin fn(P) \Rightarrow (\nu n)(P|Q) \equiv P|(\nu n)Q$
$n \neq m \Rightarrow (\nu n)m[P] \equiv m[(\nu n)P]$
$P|0 \equiv P$
$(\nu n)0 \equiv 0$
$!0 \equiv 0$
$\epsilon.P \equiv P$
$go(\epsilon).P \equiv P$
$(M.M').P \equiv M.M'.P$

$P \equiv P$
$Q \equiv P \Rightarrow Q \equiv P$
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$
$P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$
$P \equiv Q \Rightarrow P|R \equiv Q|R$
$P \equiv Q \Rightarrow !P \equiv !Q$
$P \equiv Q \Rightarrow M[P] \equiv M[Q]$
$P \equiv Q \Rightarrow M.P \equiv M.Q$
$P \equiv Q \Rightarrow (x).P \equiv (x).Q$ □

The behavior of processes of ambient calculus is defined by the following reduction rules:

Definition 2.3
(Reduction: $P \rightarrow Q$ [3),4)]**)**

$n[in\ m.P|Q]|m[R] \rightarrow m[n[P|Q]|R]$
$go(in\ m).P|m[Q] \rightarrow m[P|Q]$
$m[n[out\ m.P|Q]|R] \rightarrow n[P|Q]|m[R]$
$m[go(out\ m).P|Q] \rightarrow P|m[Q]$
$open\ n.P|n[Q] \rightarrow P|Q$
$\langle M \rangle|(x).P \rightarrow P\{x \leftarrow M\}$

$P \rightarrow Q \Rightarrow P|R \rightarrow Q|R$
$P \rightarrow Q \Rightarrow (\nu n)P \rightarrow (\nu n)Q$
$P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$
$P' \equiv P, P \rightarrow Q, Q \equiv Q' \Rightarrow P' \rightarrow Q'$ □

2.2 Contextual Equivalence

Parallel testing equivalence [14)] is a useful equivalence relation for some process calculi (such as CCS), though Ref. 7) shows an example that indicates it is not appropriate for ambient calculus as follows:

Example 2.4 (Parallel Testing [7)]**)** Let $\Downarrow$ be the predicate defined in Definition 2.6 and let processes A and B be parallel testing equiv-

alent iff for all processes R and names n, $A|R \Downarrow n \Leftrightarrow B|R \Downarrow n$. This means R is a tester and the situation "n will be visible" is an element of the success set. This equivalence identifies following two processes:

$P_e \stackrel{def}{=} out\ p.0.$

$Q_e \stackrel{def}{=} 0.$

For any R and n, $P_e|R \Downarrow n \Leftrightarrow Q_e|R \Downarrow n$ thus P_e and Q_e are parallel testing equivalent, though when they are placed in the ambient $p[m[]]$, that is, $p[m[P_e]]$ and $p[m[Q_e]]$, the former will become the process $p[]|m[]$, while the latter will not change. □

Thus, another equivalence relation was proposed in Ref. 7) that is called context equivalence. This subsection reviews several definitions for the relation.

Definition 2.5
(P exhibits a name n: $P \downarrow n$ 7))

$P \downarrow n \stackrel{def}{=}$
there are $\vec{m}, P', P''$ with $n \notin \{\vec{m}\}$ and $P \equiv (\nu\vec{m})(n[P']|P'')$ □

Intuitively, $P \downarrow n$ means the process P has at least one ambient whose name is n in its top level. For example, see the process P appears in the Definition 2.5, we can observe the ambient $n[\cdots]$ directly.

Definition 2.6
(Convergence to a name n: $P \Downarrow n$ 7))

$$\text{(Conv Exh)}\ \frac{P \downarrow n}{P \Downarrow n} \qquad \text{(Conv Red)}\ \frac{P \to Q \quad Q \Downarrow n}{P \Downarrow n}$$ □

Intuitively, $P \Downarrow n$ means the process P will exhibit the name n directly or after several reductions. For example, let $P_e \stackrel{def}{=} m[n[out\ n]]$ where $P_e \Downarrow n$ holds, we can observe the ambient $n[\cdots]$ after it executes the capability $out\ n$.

Definition 2.7 (Context 7)) A *context* $\mathcal{C}()$ is a process containing zero or more holes. $\mathcal{C}(P)$ is the outcome of filling each of the holes in the context $\mathcal{C}()$ with the process P. □

Definition 2.8
(Contextual Equivalence: $P \simeq Q$ 7))

$P \simeq Q \stackrel{def}{=}$
for all $n, \mathcal{C}()$. if $\mathcal{C}(P)$ and $\mathcal{C}(Q)$ are closed, then $\mathcal{C}(P) \Downarrow n \Leftrightarrow \mathcal{C}(Q) \Downarrow n$. □

Example 2.9
(Contextually different processes) 7)
Let P_e and Q_e be the processes defined in Example 2.4 and $\mathcal{C}_e$ be a context as follows: $\mathcal{C}_e \stackrel{def}{=} p[m[()]]$. As $\mathcal{C}_e(P_e) \Downarrow m$ while $\mathcal{C}_e(Q_e) \not\Downarrow m$, $P_e \not\simeq Q_e$. □

3. Problems of Contextual Equivalence

Contextual equivalence is a relation that concentrates on the possibility whether processes will exhibit the same names or not. This means the relation corresponds to "may testing". This section focuses on the two problems of contextual equivalence, one is caused by its fineness as coarse as may testing, and the other comes to light when we define the nondeterministic choice operator.

3.1 Contextually Testing Equivalence

Example 3.1 The following processes P_e and Q_e are contextually equivalent:

$P_e \stackrel{def}{=} n[0].$

$Q_e \stackrel{def}{=} (\nu m)(m[n[0]]|m[0]|open\ m.0).$

The process Q_e can behave as $n[0]$ when the capability $open\ m$ dissolves the ambient $m[]$ of $m[n[0]]$ and any context neither has effects on ambient $m[]$ nor interferes the capability $open\ m$ since the name m is restricted. Thus, for all context $\mathcal{C}_e$, if $\mathcal{C}_e(P_e) \Downarrow n$ then $\mathcal{C}_e(Q_e) \Downarrow n$ and vice versa. So, according to Definition 2.8, $P_e \simeq Q_e$. The process Q_e can, however, act as 0 when the capability $open\ m$ dissolves the ambient $m[]$ of $m[0]$ while that is impossible for the process P_e, thus P_e and Q_e must be distinguished. To solve the problem, we add another definition that corresponds to must testing and we call it "Hit". □

Definition 3.2 (Hit a name n: $P \downdownarrows n$)

$$\text{(Hit Exh)}\ \frac{P \downarrow n}{P \downdownarrows n} \qquad \text{(Hit Red)}\ \frac{\text{for any } Q \text{ st } P \to Q.\quad Q \downdownarrows n}{P \downdownarrows n}$$ □

Intuitively, $P \downdownarrows n$ iff ambient $n[]$ will be visible in every possible execution path of P.

Definition 3.3 (Contextually Testing Equivalence: $P \simeq_{test} Q$)

$P \simeq_{test} Q \stackrel{def}{=}$
for all $n, \mathcal{C}()$. if $\mathcal{C}(P)$ and $\mathcal{C}(Q)$ are closed,
then $\mathcal{C}(P) \Downarrow n \Leftrightarrow \mathcal{C}(Q) \Downarrow n$
and $\mathcal{C}(P) \downdownarrows n \Leftrightarrow \mathcal{C}(Q) \downdownarrows n$. □

Example 3.4 Let P_e and Q_e be the processes defined in Example 3.1 and the context $\mathcal{C}_e$ as follows: $\mathcal{C}_e() = 0|-$. Obviously, $\mathcal{C}_e(P_e) \downdownarrows n$ though $\mathcal{C}_e(Q_e) \not\downdownarrows n$ because $\mathcal{C}_e(Q_e)$ can reduce to $(\nu m)(m[n[0]])$ which never exhibits the name n. Consequently, $P_e \not\simeq_{test} Q_e$.

3.2 Choice Operator

Since ambient calculus does not have an ex-

ternal choice as a primitive operator, it did not seem there were the problems that have been discussed in several papers for other languages resulted from choice operation [2),6),11),13)].

We found, however, an interesting phenomenon when we defined an external choice operator using only restriction and parallel compositions. Using the choice operator, we can construct two contextual equivalent processes of ambient calculus which must be distinguished though even contextually testing equivalence identifies.

This subsection firstly presents the examples of two contextually equivalent processes which are defined using the choice operator, then we present the definition of the choice operator.

3.2.1 Problems with "+"

Let "+" be a usual choice operator defined in other process algebras such as CCS [13)] or Pi Calculus [10)] while, ambient calculus does not have it. We discus here the case ambient calculus had the choice operator as one of primitives: if ambient calculus had a choice operator, we would have two contextual equivalent processes which must be distinguished.

Example 3.5 (Contextually Equivalent Processes P_e and Q_e)
Let P_e and Q_e be the processes as follows:

$$P_e \stackrel{def}{=} a[P_1] + a[P_2].$$
$$Q_e \stackrel{def}{=} a[Q_1] + a[Q_2].$$
$$P_1 \stackrel{def}{=} a[P_2] + b[].$$
$$P_2 \stackrel{def}{=} a[P_1] + c[].$$
$$Q_1 \stackrel{def}{=} a[Q_1] + b[].$$
$$Q_2 \stackrel{def}{=} a[Q_2] + c[].$$

The behavior of P_e and Q_e are illustrated in **Fig. 1**.

The problem of these processes were originally shown in Ref. 2) for CCS processes. When the process $n[!in\ a|in\ b|in\ c]$ (we call it a traveling ambient) is running parallel to the process P_e, the traveling ambient will be able to enter the ambient $b[]$ in P_e though this may be impossible when the traveling ambient is running parallel to Q_e. Contextual equivalence, however, identifies P_e and Q_e. Intuitively, P_e and Q_e are the processes as follows:

$$P_e \doteq a[a[a[\cdots] + b[]] + c[]] + b[]] + a[a[a[\cdots] + c[]] + b[]] + c[]].$$

$$Q_e \doteq a[a[a[\cdots] + b[]] + b[]] + b[]]$$

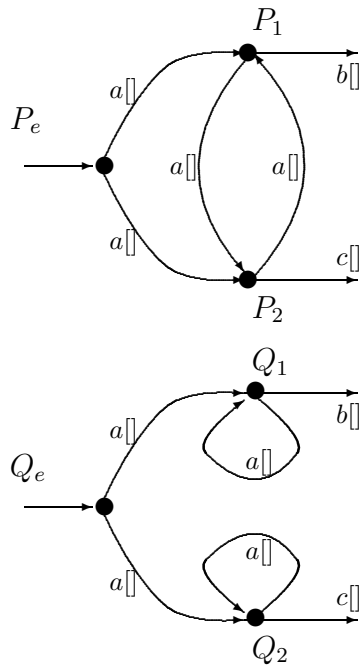

Fig. 1 Contextually equivalent processes P_e and Q_e.

$$+a[a[a[a[\cdots] + c[]] + c[]] + c[]].$$

If "+" were a primitive operator, any times of opening of ambient a, that is, $open\ a., \ldots, .open\ a$ for P_e would exhibit the pair of ambients $a[]$ and $b[]$ or $a[]$ and $c[]$. The same opening operation for Q_e would be able to exhibit the same pair of ambient. □

3.2.2 External Choice

According to Ref. 3), there is no primitives for external choice for ambient calculus in the spirit of the asynchronous Pi calculus and it can be simulated by using parallel composition and restriction primitives. Choice operation can be used for many purpose such as simulating Numerals, simulating boolean conditionals, so an example of the simulation of external choice is presented in Ref. 3) as follows:

$$n \Rightarrow P + m \Rightarrow Q \stackrel{def}{=} (\nu p\ q\ r)($$
$$p[in\ n.out\ n.q[out\ p.open\ r.P]]$$
$$\mid p[in\ m.out\ m.q[out\ p.open\ r.Q]]|open\ q|r[]).$$

This macro captures many aspects of "+" operation but has several limitations such that it has to know what name does the coming ambient have, and the process $n[in\ a]|n \Rightarrow a[] + n \Rightarrow b[]$ dose not work as "+" because the environment (such as $n[in\ a]$ or traveling ambient in Example 3.5) can not select the alternative ($a[]$ or $b[]$) but the the macro itself selects it. In

order for environments to select the alternative (for simulating processes P_e and Q_e in Example 3.5), We define another external choice which we can use for both deterministic and nondeterministic purpose as follows:

Definition 3.6 (Choice Operator "+") Let B and C be any processes of ambient calculus. Then, we define $b[B] + c[C]$ as follows:

$$
\begin{array}{l}
b[B] + c[C] \stackrel{def}{=} \\
(\nu trash\ sync)\ (\\
\ b[in\ trash \mid go(in\ n.out\ n) \\
\ .sync[out\ trash | \ trash[out\ b] \mid B] | open\ sync] \\
\ \mid c[in\ trash \mid go(in\ n.out\ n) \\
\ .sync[out\ trash | trash[out\ c] \mid C] | open\ sync]).
\end{array}
$$

□

From now on, the symbol "+" expresses the choice operator defined in Definition 3.6.

Example 3.7 Intuitively, when we have an ambient whose name is n that goes in the ambient $b[]$ such as $n[in\ b]$ in parallel to $b[B]+c[C]$ as follows:

$$n[in\ b] \mid (b[B] + c[C]),$$

then, the ambient $c[C]$ that is not chosen by $n[in\ b]$ will be captured in restricted ambient $trash[]$ so that it becomes invisible as if it disappeared as follows:

$$
\begin{array}{l}
\stackrel{t}{\Rightarrow} \\
(\nu trash\ sync)(\\
\ b[n[] \mid B] \\
\ |trash[c[go(in\ n.out\ n) \\
\ \ .sync[out\ trash | trash[out\ c] | C] | open\ sync]).
\end{array}
$$

Where $\stackrel{t}{\Rightarrow}$ above means some visible transitions (including $in\ b$) occur followed by or following 0 or more invisible actions as usual [13].

We explain the behavior of the choice operator "+" more precisely by the transitions in Example A.5.4 that shows the nondeterministic property of "+".

Our "+" operator defined in Definition 3.6 can be used like a primitive operator and when it works as a primitive one, the explanations in Example 3.5 hold even if we replace the primitive operator in Example 3.5 with the operator defined in Definition 3.6. While, our "+" is a macro defined by using parallel operator, there exists a case in which ambients $a[]$, $b[]$ and $c[]$ are exhibited at the same time for P_e. The next example shows this situation.

Example 3.8 Let P_e and Q_e be the processes defined in Example 3.5 and $\mathcal{C}_e$ be a context as follows:

$$\mathcal{C}_e() = n[!in\ a] \mid !open\ a \mid -\ .$$

We can observe ambients $a[]$, $b[]$ and $c[]$ at the same time in the following computation for $\mathcal{C}_e(P_e)$:

$$
\begin{array}{l}
\mathcal{C}_e(P_e) \xrightarrow{open\ a} \xrightarrow{open\ a} \xrightarrow{go(in\ n)} \xrightarrow{go(out\ n)} \\
\qquad \xrightarrow{go(in\ n)} \xrightarrow{go(out\ n)} \xrightarrow{open\ sync} \xrightarrow{open\ sync} \\
!open\ a \mid n[!in\ a] \\
\ \mid (\nu\ trash)(in\ trash \mid in\ trash \mid out\ trash \\
\qquad \mid out\ trash \mid trash[out\ a] \mid trash[out\ a] \\
\qquad\qquad \mid P_1 \mid P_2).
\end{array}
$$

Thus, ambients $a[]$, $b[]$ and $c[]$ in P_1 and P_2 are exhibited. But, the same context $\mathcal{C}_e$ will exhibit $a[]$, $b[]$ and $c[]$ for Q_e in the same sequence, thus, the procedure for proving contextual equivalence can not distinguish P_e from Q_e. □

In Section A.5, we give a formal proof of $P_e \simeq Q_e$ that we have explained in Example 3.5 and 3.8.

The next example shows that contextually testing equivalence is finer than usual parallel testing equivalence, so that it can distinguish P_e from Q_e.

Example 3.9 ($P_e \not\approx_{test} Q_e$) Let P_e and Q_e be the processes defined in Example 3.5 and $\mathcal{C}_e$ be the context as follows:

$$\mathcal{C}_e() \stackrel{def}{=} n[!in\ a] \mid !open\ a \mid open\ b \mid -\ .$$

In usual parallel testing scenario, P_e would not pass must testing. This is because by using even a context (tester) that have infinite opening a action such as $\mathcal{C}_e$, from P_2 position of Fig. 1, we may not open b because of the infinite path of opening the a ambients, this means $\mathcal{C}_e(P_e)$ may fail to open b, and so do $\mathcal{C}_e(Q_e)$.
In contextually testing scenario, however, we will **observe** the name b from P_2 position of Fig. 1 even in that infinite path, this means $\mathcal{C}_e(P_e)$ never fail to Hit b while $\mathcal{C}_e(Q_e)$ do from Q_2 position. Consequently, $P_e \not\approx_{test} Q_e$. □

As we have seen in Example 3.9, the testing equivalence of other process algebras and that of this paper have significant difference in the way how to test processes. The former tests processes by checking what action is possible while the latter by checking what names are visible. Thanks to this difference, contextually testing equivalence makes a distinction between P_e and Q_e that are impossible in usual testing relations. Of course this character comes from the idea of contextual equivalence. By introducing choice operation to Ambient Calculus, this difference becomes clear.

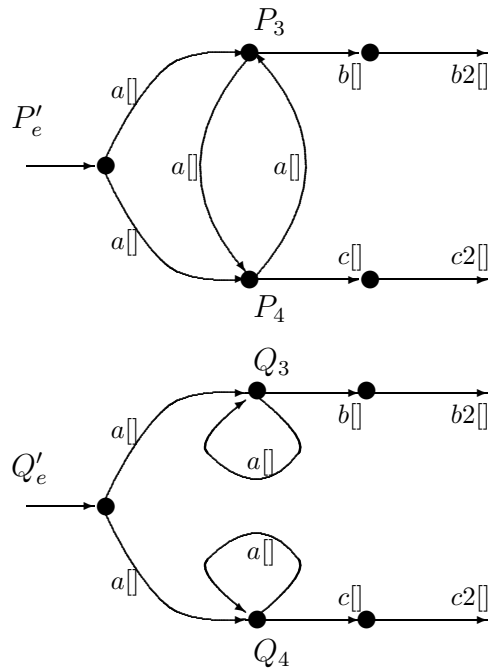

Fig. 2 Contextually testing equivalent processes P'_e and Q'_e.

We can, however, make similar processes by adding only two ambients to P_e and Q_e that contextually testing equivalence identifies as follows.

Example 3.10 ($P'_e \simeq_{test} Q'_e$) Let $\mathcal{C}_e$ be the same context in Example 3.9 ($\mathcal{C}_e() \stackrel{def}{=} n[!in\ a] \mid !open\ a \mid open\ b \mid -$) and, P'_e and Q'_e be the processes as follows:

$$P'_e \stackrel{def}{=} a[P_3] + a[P_4].$$
$$Q'_e \stackrel{def}{=} a[Q_3] + a[Q_4].$$

$$P_3 \stackrel{def}{=} a[P_4] + b[b2[]].$$
$$P_4 \stackrel{def}{=} a[P_3] + c[c2[]].$$
$$Q_3 \stackrel{def}{=} a[Q_3] + b[b2[]].$$
$$Q_4 \stackrel{def}{=} a[Q_4] + c[c2[]].$$

The behavior of P'_e and Q'_e are illustrated in **Fig. 2**.
In this case, we may not observe the name $b2$: $\mathcal{C}_e(P'_e)$ may fail to open b because of the infinite path of opening the name a. This means P'_e does not hit the name b nor does Q'_e. □

4. Extension of Contextual Equivalence

Processes P'_e and Q'_e in Example 3.10 have different properties though they are contextually (testing) equivalent. Thus, we need a finer equivalence relation.

This section introduces *Contextually Should Equivalence* which is the extension of contextual equivalence using the *should testing* idea of Ref. 2).

Definition 4.1 Let P be a process and n be a name such that $P \Downarrow n$. We define R^n_P be the set of processes as follows:

$$R^n_P \stackrel{def}{=} \{R \mid \exists \alpha.(P \stackrel{\alpha}{\Rightarrow} R) \wedge R \downarrow n\}. \quad \square$$

Intuitively, by the definition of convergence, $P \Downarrow n$ means there are some reductions from P to a process R and $R \downarrow n$. R^n_P means the set of those processes Rs.

Definition 4.2
$P \leq_{should} Q \stackrel{def}{=}$

$\forall n, \mathcal{C}()$. if $\mathcal{C}(Q)$ and $\mathcal{C}(P)$ are closed, then
$(\mathcal{C}(P) \Downarrow n \Rightarrow \mathcal{C}(Q) \Downarrow n) \wedge$
$(\mathcal{C}(P) \Downarrow\!\Downarrow n \Rightarrow \mathcal{C}(Q) \Downarrow\!\Downarrow n)$
$\wedge\ \forall m, \forall R_p \in R^n_{\mathcal{C}(P)}, \exists R_q \in R^n_{\mathcal{C}(Q)}.$
$(R_p \Downarrow m \Rightarrow R_q \Downarrow m) \wedge$
$(R_p \Downarrow\!\Downarrow m \Rightarrow R_q \Downarrow\!\Downarrow m).$ □

Definition 4.3
(Contextually Should Equivalence)
$P \simeq_{should} Q \stackrel{def}{=} P \leq_{should} Q \wedge Q \leq_{should} P.$ □

We explain the intuitive notion of contextually should equivalence in the following example:

Example 4.4 Let P'_e and Q'_e be the processes defined in Example 3.10 and $\mathcal{C}_e()$ be the context as follows:

$$\mathcal{C}_e() \stackrel{def}{=} n[!in\ a] \mid !open\ a \mid open\ b \mid -.$$

Then, $\mathcal{C}_e(P'_e) \Downarrow c \wedge \forall R_p \in R^c_{\mathcal{C}_e(P'_e)}.\ R_p \Downarrow b2$ holds as follows:
$\mathcal{C}_e(P'_e) \xrightarrow{in\ a} \xrightarrow{go(in\ n)} \xrightarrow{go(out\ n)} \xrightarrow{open\ sync} \xrightarrow{out\ a}$
$\xrightarrow{go(in\ trash)} \xrightarrow{go(in\ trash)} \xrightarrow{go(out\ trash)} \xrightarrow{open\ a}$
$n[!in\ a] \mid !open\ a \mid open\ b$
$\mid (\nu trash\ sync)\ (a[P_3] + c[c_2[]] \mid trash[\cdots])$
$(=$ one of $R_p)$. (See Appendix A.6 for details)

Obviously $R_p \Downarrow c$. As $P_3 = a[P_4] + b[b_2[]]$, we can easily find that $R_p \Downarrow b$.
This means, when the ambient $c[]$ is exhibited, there is a possibility that the ambient $b2[]$ will be exhibited.

On the other hand, we have a $R_q \in R^n_{\mathcal{C}_e(Q'_e)}$ that exhibits the name c as follows:
$\mathcal{C}_e(Q'_e) \xrightarrow{in\ a} \xrightarrow{go(in\ n)} \xrightarrow{go(out\ n)} \xrightarrow{open\ sync} \xrightarrow{out\ a}$
$\xrightarrow{go(in\ trash)} \xrightarrow{go(in\ trash)} \xrightarrow{go(out\ trash)} \xrightarrow{open\ a}$
$n[!in\ a] \mid !open\ a \mid open\ b$
$\mid (\nu trash\ sync)\ (a[Q_4] + c[c_2[]] \mid trash[\cdots])$
$(=$ one of $R_q)$.
But the R_q never converges to the name b_2 and any other R_q that exhibits the name c never converges to the name b_2 (we can prove this by structural induction). So $\forall m, \forall R_p \in R^n_{\mathcal{C}_e(P'_e)}, \exists R_q \in R^n_{\mathcal{C}_e(Q'_e)}.\ (R_p \Downarrow m \Rightarrow R_q \Downarrow m)$ false, and $P'_e \not\leq_{should} Q'_e$. Consequently, contextually should equivalence can distinguish P'_e from Q'_e. □

In Definition 4.2, we use not only $\Downarrow n$ but also $\downarrow\downarrow n$ though $\downarrow\downarrow n$ dose not play any important roles in Example 4.4. The following example shows the roles of $\downarrow\downarrow n$ for contextually should equivalence.

Example 4.5 Let P_e and Q_e be processes defined in Example 3.1 and $\mathcal{C}_e() = 0|-$. The condition $\mathcal{C}_e(P_e) \downarrow\downarrow n \Rightarrow \mathcal{C}_e(Q_e) \downarrow\downarrow n$ does not hold because Q_e has a transition that will never exhibit a name n. Thus $P_e \not\leq_{should} Q_e$. □

Proposition 4.6 Contextually should equivalence is a congruence. □

Proof: We can apply the strategy of the proof for contextual equivalence [7].

(Equivalence part) Reflexivity and symmetry are trivial. We only show the proof for transitivity. Suppose $P_1 \leq_{should} P_2$ and $P_2 \leq_{should} P_3$. Let $\mathcal{C}()$ be any context such that $\mathcal{C}(P_1)$ and $\mathcal{C}(P_3)$ are closed, θ be a closing substitution for $\mathcal{C}(P_2)$ and $\mathcal{D}() \stackrel{def}{=} \mathcal{C}()\theta$. Suppose n be any name such that $\mathcal{C}(P_1) \Downarrow n \wedge \mathcal{C}(P_1) \downarrow\downarrow n$.

Since $\mathcal{C}(P_1)$ are closed, $\mathcal{C}(P_1) \equiv \mathcal{D}(P_1)$ holds, thus $\mathcal{D}(P_1) \Downarrow n \wedge \mathcal{D}(P_1) \downarrow\downarrow n$. With this condition, the fact $\mathcal{D}(P_2)$ is closed and the assumption $P_1 \leq_{should} P_2$, we are led to the following condition:

$\mathcal{D}(P_2) \Downarrow n \wedge \mathcal{D}(P_2) \downarrow\downarrow n \wedge$
$\forall m, \forall R_p \in R^n_{\mathcal{D}(P_1)}, \exists R_q \in R^n_{\mathcal{D}(P_2)}.$
$(R_p \Downarrow m \Rightarrow R_q \Downarrow m) \wedge (R_p \downarrow\downarrow m \Rightarrow R_q \downarrow\downarrow m).$

As $\mathcal{C}(P_3)$ is closed, $\mathcal{C}(P_3) \equiv \mathcal{D}(P_3)$ and $\mathcal{D}(P_3)$ is also closed. These condition and the assumption $P_2 \leq_{should} P_3$ lead us to the following condition:

$\mathcal{D}(P_3) \Downarrow n \wedge \mathcal{D}(P_3) \downarrow\downarrow n \wedge$
$\forall m, \forall R_q \in R^n_{\mathcal{D}(P_2)}, \exists R_r \in R^n_{\mathcal{D}(P_3)}.$
$(R_q \Downarrow m \Rightarrow R_r \Downarrow m) \wedge (R_q \downarrow\downarrow m \Rightarrow R_r \downarrow\downarrow m).$

As $\mathcal{C}(P_3) \equiv \mathcal{D}(P_3)$, we have
$\mathcal{C}(P_3) \Downarrow n \wedge \mathcal{C}(P_3) \downarrow\downarrow n \wedge$
$\forall m, \forall R_q \in R^n_{\mathcal{C}(P_2)}, \exists R_r \in R^n_{\mathcal{C}(P_3)}.$
$(R_q \Downarrow m \Rightarrow R_r \Downarrow m) \wedge (R_q \downarrow\downarrow m \Rightarrow R_r \downarrow\downarrow m).$

So, $P_1 \leq_{should} P_3$. The symmetric procedure proves $P_3 \leq_{should} P_1$.

(Precongruence part) Let P and Q be any contextually should equivalent processes, $\mathcal{C}()$ be any context such that $\mathcal{C}(P)$ and $\mathcal{C}(Q)$ are closed. Suppose $\mathcal{D}()$ be any contexts such that $\mathcal{D}(\mathcal{C}(P))$ and $\mathcal{D}(\mathcal{C}(Q))$ are closed.

As $\mathcal{D}(\mathcal{C}())$ is a context and $P \leq_{should} Q$, for any name n, the following conditions hold by Definition 4.2:

$(\mathcal{D}(\mathcal{C}(P)) \Downarrow n \Rightarrow \mathcal{D}(\mathcal{C}(Q)) \Downarrow n)$
$\wedge (\mathcal{D}(\mathcal{C}(P)) \downarrow\downarrow n \Rightarrow \mathcal{D}(\mathcal{C}(Q)) \downarrow\downarrow n)$
$\wedge\ \forall m, \forall R_p \in R^n_{\mathcal{D}(\mathcal{C}(P))}, \exists R_q \in R^n_{\mathcal{D}(\mathcal{C}(Q))}.$
$(R_p \Downarrow m \Rightarrow R_q \Downarrow m) \wedge (R_p \downarrow\downarrow m \Rightarrow R_q \downarrow\downarrow m).$

Consequently, we have $\mathcal{C}(P) \leq_{should} \mathcal{C}(Q)$. The symmetric way proves $\mathcal{C}(Q) \leq_{should} \mathcal{C}(P)$.

Because we have proved $\mathcal{C}(P) \simeq_{should} \mathcal{C}(Q)$ for any context $\mathcal{C}()$ and for any two processes P and Q such that $P \simeq_{should} Q$, should equivalence "$\simeq_{should}$" is a precongruence. □

Proposition 4.7 Let P and Q be processes. If $P \simeq_{should} Q$ then $P \simeq Q$. □

Proof: Obvious by Definition 4.3 and Definition 2.8.

Proposition 4.7 and Example 4.4 show that contextually should equivalence is the finer congruence relation than contextual equivalence enough to distinguish the process P_e from the process Q_e of Example 3.5.

5. Comparison with Electoral System

We introduced a choice macro that can be used for deterministic and no deterministic purpose using only restriction and parallel composition. Reference 15) also shows another choice mechanism without using "+" primitive, called *Electoral System*. An electoral system **Net** whose size is k is a network of processes (that is a composition of processes) as follows:

$$\mathbf{Net} \stackrel{def}{=} P_0 | \cdots | P_{k-1},$$

where $P_0, \cdots, P_{k-1}$ are processes satisfying the following conditions: for any maximal compu-

tation C of **Net**, there exists an $i < k$ such that $\mathbf{Obs}(C)=\{\omega_i\}$ where ω_i appears only in P_i. The computation is defined in the ordinal way as a sequence of reduced processes such as $\mathbf{Net}_0 \rightarrow \mathbf{Net}_1 \rightarrow \mathbf{Net}_2 \rightarrow, \cdots$, maximal means the sequence can not be extended and $\mathbf{Obs}(C)$ is a set of names exhibited in the computation that means a **Net** can interact only through the names in $\mathbf{Obs}(C)$ (See more precise definitions in Ref. 15)).

The condition of the **Net**: "there exists an $i < k$ such that $\mathbf{Obs}(C)=\{\omega_i\}$" means, only one process P_i will be elected as a winner (or a leader) of the **Net**.

Example 5.1 (Electoral system of Ambient Calculus [15])

Let **Net** be a process as follows:

$$\begin{aligned}\mathbf{Net} \stackrel{def}{=}\ & n_0[in\ n_1.\omega_0[out\ n_0.out\ n_1]] \\ & |n_1[in\ n_0.\omega_1[out\ n_1.out\ n_0]].\end{aligned}$$

After the following transition, only one observable ω_0 will be exhibited:

$$\begin{aligned}&\mathbf{Net} \xrightarrow{in\ n_1} \xrightarrow{out\ n_0} \xrightarrow{out\ n_1} \\ &\quad \omega_0[]|n_1[in\ n_0.\omega_1[out\ n_1.out\ n_0]|n_0[]].\end{aligned}$$

This means one of the component processes in **Net** (that is $n_0[in\ n_1.\omega_0[out\ n_0.out\ n_1]]$) is chosen as a winner of the **Net** and we can see a choice mechanism is realized without using choice primitive. □

The mechanism of electoral system and that of ours have several similarity: both are constructed using parallel composition, do not need central controller that would recognize which component has been chosen and makes other component not available. These are great merits for the implementation. On the other hand, there are obvious difference between them: electoral system chooses the representation of the components autonomously while our choice macro chooses the representation that is chosen by environments and other components will disappear autonomously.

6. Conclusion

The primary result of the paper is that it has pointed out there had been contextually equivalent processes of ambient calculus which must be distinguished, and defined alternative equivalence relations.

Ambient calculus does not have an external choice as primitives, it had seemed that problems on discriminating processes caused by choice operation had not existed. We found, however, the problem by defining an external choice operator using only parallel composition and restriction primitives. We have solved the problem using should testing idea [2].

Our choice operator can work as ideal choice primitives and when it works well the problem above arises. But the opening capability can destroy the structure of choice operator. Cardelli has defined typed ambient calculus in Ref. 4), and using types, we will find our choice operator works better. Thus, in future we will investigate an equivalence relation on typed ambient calculus.

References

1) Berry, G. and Boudol, G.: The Chemical Abstract Machine, *Theoretical Computer Science*, Vol.96, pp.217–248 (1992).
2) Brinksma, E., Rensink, A. and Vogler, W.: Fair Testing, *LNCS*, Vol.962, pp.313–327 (1995).
3) Cardelli, L. and Gordon, A.D.: Mobile Ambients, *LNCS*, Vol.1378, pp.140–155 (1998).
4) Cardelli, L. and Gordon, A.D.: Mobility Types for Mobile Ambients, *LNCS*, Vol.1644, pp.230–239 (1999).
5) Fournet, G. and Gonthier, G.: A Calculus of Mobile Agents, *LNCS*, Vol.1119, pp.406–421 (1996).
6) Gaifman, H., Maher, M.J. and Shapiro, E.: Reactive Behavior Semantics for Concurrent Constraint Logic Programs, *Proc. North American Conf. on Logic Programming*, pp.553–569 (1989).
7) Gordon, A.D. and Cardelli, L.: Equational Properties of Mobile Ambients, *Mathematical Structures in Computer Science*, Vol.13, No.3, pp.371–408 (2003).
8) Hennessy, M.: Higher-Order Processes and Their Models, *LNCS*, Vol.820, pp.286–303 (1994).
9) Hoare, C.: *Communicating Sequential Processes*, Prentice Hall (1985).
10) Parrow, J.: An Introduction to the π-Calculus, *HANDBOOK OF PROCESS ALGEBRA*, Bergstra, J.A., Ponse, A. and Smolka, S.A. (eds.), NORTH-HOLLAND (2001).
11) Kato, T.: The Semantics of Guarded Horn Clauses for Programs on Distributed Environments, *Trans of Information Processing Society of Japan*, Vol.40, No.1, pp.362–373 (1999).
12) Kumeno, F., Sato, H., Kato, T. and Honiden, S.: Flage: A Programming Language for Adaptive Software, *Proc. International Conference on Software Engineering '98* (1998).

13) Milner, R.: *Communication and Concurrency*, Prentice Hall (1989).
14) Nicola, R.D. and Hennessy, M.: Testing Equivalences for Processes, *TCS*, Vol.34, pp.83–133 (1984).
15) Phillips, I. and Vigliotti, M.: Electoral Systems in Ambient Calculi, *Proc. FOSSACS 2004 LNCS 2004*, Vol.2987, pp.408–422 (2004).
16) Someya, Y., Abe, H., Matubara, K., Toumura, K. and Kato, K.: Implementing Mobile Web Search Robots with the Planet Mobile Object System, *15 th Conference Proceedings*, JSSST, pp.29–36 (1998).

Appendix

A.1 Tools for Proving Contextual Equivalence

When two processes are not equivalent, we only have to indicate a context with which we can find different sets of names of ambients observed by the environment of processes and the context. On the other hand, we need to consider all contexts to show that they are contextually equivalent. This section shows tools that help us prove the equivalence introduced in Ref. 7).

A.1.1 A Hardening Relation

The hardening relation introduced in Ref. 7) takes the form

$$P > (\nu p_1 \dots p_k)\langle P'\rangle P''$$

which means P consists of subprocesses P' and P'', and P' can be the top level of subprocesses. We call the right hand side of the relation concretion. Using the hardening relation, Ref. 7) defines the labeled transition system by which we can understand the sequential behavior of any processes.

Definition A.1.1 (Concretions [7]) We define concretions ranged over by C, D as follows:

$C, D ::=$	concretions
$(\nu\vec{p})\langle M.P\rangle Q$	action, $M \in \{in\ n, out\ n, open\ n, go(N)\}$ where N is a path consists of *in* n and *out* n
$(\nu\vec{p})\langle n[P]\rangle Q$	ambient
$(\nu\vec{p})\langle (x).P\rangle Q$	input
$(\nu\vec{p})\langle\langle M\rangle\rangle Q$	output

□

We can understand how processes are harden to concretions in Definition A.1.3.

Definition A.1.2 (Restricting a Concretion $\overline{(\nu n)}C$ [7])
Let $C = (\nu\vec{p})\langle P_1\rangle P_2$ and $n \notin \{\vec{p}\}$

- If $n \in fn(P_1)$ then:
  (a) If $P_1 = m[P_1'], m \neq n, n \notin fn(P_2)$, let $\overline{(\nu n)}C \stackrel{def}{=} (\nu\vec{p})\langle m[(\nu n)P_1']\rangle P_2$.
  (b) Otherwise let $\overline{(\nu n)}C \stackrel{def}{=} (\nu n\ \vec{p})\langle P_1\rangle P_2$.
- If $n \notin fn(P_1)$ let $\overline{(\nu n)}C \stackrel{def}{=} (\nu\vec{p})\langle P_1\rangle (\nu n)P_2$.

□

Definition A.1.3 (Hardening: $P > C$ [7])

(Harden Action)

$$\frac{M \in \{in\ n, out\ n, open\ n, go(N)\}}{M.P > (\nu)\langle M.P\rangle 0}$$

(Harden ϵ) (Harden .)

$$\frac{P > C}{\epsilon.P > C} \qquad \frac{M.(N.P) > C}{(M.N).P > C}$$

(Harden Amb)

$$\overline{n[P] > (\nu)\langle n[P]\rangle 0}$$

(Harden Input)

$$\overline{(x).P > (\nu)\langle (x).P\rangle 0}$$

(Harden Output)

$$\overline{\langle M\rangle > (\nu)\langle\langle M\rangle\rangle 0}$$

(Harden Par 1)(for $\{\vec{p}\} \cap fn(Q) = \emptyset$)

$$\frac{P > (\nu\vec{p})\langle P'\rangle P''}{P|Q > (\nu\vec{p})\langle P'\rangle (P''|Q)}$$

(Harden Par2)(for $\{\vec{q}\} \cap fn(P) = \emptyset$)

$$\frac{Q > (\nu\vec{q})\langle Q'\rangle Q''}{P|Q > (\nu\vec{q})\langle Q'\rangle (P|Q'')}$$

(Harden Repl)

$$\frac{P > (\nu\vec{p})\langle P'\rangle P''}{!P > (\nu\vec{p})\langle P'\rangle (P''|!P)}$$

(Harden Res)

$$\frac{P > C}{(\nu n)P > \overline{(\nu n)}C}$$

□

A.2 A Labelled Transition System [7]

Using hardening relation, a labeled transition is defined.

Definition A.2.1 (Labels [7])

$\alpha ::=$ label
- τ internal step
- $in\ n$ enter ambient n
- $out\ n$ exit ambient n
- $go(N)$ make a process move along a path N
- $open\ n$ dissolve ambient n □

Definition A.2.2 (Labelled Transitions [7])

(Trans Cap)
$$\frac{P > (\nu\vec{p})\langle M.P'\rangle P'' \quad fn(M)\cap\{\vec{p}\}=\emptyset}{P \xrightarrow{M} (\nu\vec{p})(P'|P'')}$$

(Trans Amb)
$$\frac{P > (\nu\vec{p})\langle n[Q]\rangle P' \quad Q\xrightarrow{\tau}Q'}{P\xrightarrow{\tau}(\nu\vec{p})(n[Q']|P')}$$

(Trans In) (where $\{\vec{r}\}\cap fn(n[Q])=\emptyset$ and $\{\vec{r}\}\cap\{\vec{p}\}=\emptyset$)
$$\frac{P > (\nu\vec{p})\langle n[Q]\rangle R \quad Q \xrightarrow{in\ m} Q' \quad R > (\nu\vec{r})\langle m[R']\rangle R''}{P\xrightarrow{\tau}(\nu\vec{p}\ \vec{r})(m[n[Q']|R']|R'')}$$

(Trans Out) (where $n\notin\{\vec{q}\}$)
$$\frac{P > (\nu\vec{p})\langle n[Q]\rangle P' \quad Q > (\nu\vec{q})\langle m[R]\rangle Q' \quad R \xrightarrow{out\ n} R'}{P\xrightarrow{\tau}(\nu\vec{p})((\nu\vec{q})(m[R']|n[Q'])|P')}$$

(Trans Go in) (where $\{\vec{r}\}\cap fn(Q)=\emptyset$ and $\{\vec{r}\}\cap\{\vec{p}\}=\emptyset$)
$$\frac{P > (\nu\vec{p})\langle Q\rangle R \quad Q \xrightarrow{go(in\ m)} Q' \quad R > (\nu\vec{r})\langle m[R']\rangle R''}{P\xrightarrow{\tau}(\nu\vec{p}\ \vec{r})(m[Q'|R''])}$$

(Trans Go out)
$$\frac{P > (\nu\vec{p})\langle n[Q]\rangle P' \quad Q > (\nu\vec{q})\langle R\rangle Q' \quad R \xrightarrow{go(out\ n)} R'}{P\xrightarrow{\tau}(\nu\vec{p})((\nu\vec{q})(R'|n[Q'])|P')}$$

(Trans Open)
$$\frac{P > (\nu\vec{p})\langle n[Q]\rangle P' \quad P' \xrightarrow{open\ n} P''}{P \xrightarrow{\tau} (\nu\vec{p})(Q|P'')}$$

(Trans I/O)(where $\{\vec{q}\}\cap fn(\langle M\rangle)=\emptyset$)
$$\frac{P > (\nu\vec{p})\langle\langle M\rangle\rangle P' \quad P' > (\nu\vec{q})\langle(x).P''\rangle P'''}{P \xrightarrow{\tau} (\nu\vec{p}\ \vec{q})(P''\{x\leftarrow M\}|P''')}$$

□

A.3 A Context Lemma

This section presents the context lemma by using a notion of *harness*. Instead of considering all contexts, we can prove contextual equivalence with only a limited set of contexts.

Definition A.3.1 (Harness [7])

$H ::=$ harnesses
- $-$ process variable
- $(\nu n)H$ restriction
- $P|H$ left composition
- $H|P$ right composition
- $n[H]$ ambient

Unlike the contexts of Section 3, harnesses are identified up to consistent renaming of bound names. Names restricted in H are renamed to avoid capture of free names of P. □

Definition A.3.2 A process or a harness is *closed* if and only if it has no free variables (though it may have free names). □

Theorem A.3.3 (Context [7]) For all processes P and Q, $P\simeq Q$ if and only if for all substitutions σ with $dom(\sigma)=fv(P)\cup fv(Q)$, and for all closed harnesses H and names n, that $H\{P\sigma\}\Downarrow n \Leftrightarrow H\{Q\sigma\}\Downarrow n$. □

A.4 An Activity Lemma

This section introduces the formal way to analyze judgments of the form $H\{P\}\downarrow n$ or $H\{P\}\rightarrow Q$ to use Theorem A.3.3.

Definition A.4.1 (Extension of the structural congruence, hardening, and reduction relation for harnesses [7])

- Let $H\equiv H'$ hold if and only if $H\{P\}\equiv H'\{P\}$ for all P.
- Let $H > (\nu\vec{p})\langle n[H']\rangle Q$ hold if and only if $H\{P\} > (\nu\vec{p})\langle n[H'\{P\}]\rangle Q$ for all P such that $\{\vec{p}\}\cap fn(P)=\emptyset$.
- Let $H > (\nu\vec{p})\langle Q\rangle H'$ hold if and only if $H\{P\} > (\nu\vec{p})\langle Q\rangle(H'\{P\})$ for all P such that $\{\vec{p}\}\cap fn(P)=\emptyset$.
- Let $H\rightarrow H'$ hold if and only if $H\{P\}\rightarrow H'\{P\}$ for all P. □

Lemma A.4.2 [7] If $H\{P\} > (\nu\vec{p})\langle P_1\rangle P_2$ then either:

(1) $H > (\nu\vec{p})\langle n[H']\rangle P_2$ and $P_1 = n[H'\{P\}]$, or

(2) $H > (\nu\vec{p})\langle P_1\rangle H'$ and $P_2 = H'\{P\}$, or

(3) $P > (\nu\vec{p})\langle P_1\rangle P', H\equiv -|R, P_2\equiv P'|R$ and $\{\vec{p}\}\cap fn(R)=\emptyset$. □

Proposition A.4.3 (Exhibition Property [7])

If $H\{P\}\downarrow n$ then either (a) $H\{Q\}\downarrow n$ for all Q, or (b) both $P\downarrow n$ and also for all $Q, Q\downarrow n$

implies that $H\{Q\} \downarrow n$. □

We extend activity theorem in Ref. 7) by adding objective moves.

Theorem A.4.4 (Activity)
$H\{P\} \to R$ if and only if:

(Act Proc) there is a reduction $P \to P'$ with $R \equiv H\{P'\}$ or

(Act Har) there is a reduction $H \to H'$ with $R \equiv H'\{P\}$, or

(Act Inter) there are H' and $\vec{r}$ with $\{\vec{r}\} \cap fn(P) = \emptyset$, and one of the following holds:

(Inter In) $H \equiv (\nu\vec{r})H'\{m[-|R']|n[R'']\}$, $P \xrightarrow{in\ n} P'$ and $R \equiv (\nu\vec{r})H'\{m[P'|R']|n[R'']\}$

(Inter Out) $H \equiv (\nu\vec{r})H'\{n[m[-|R']|R'']\}$, $P \xrightarrow{out\ n} P'$ and $R \equiv (\nu\vec{r})H'\{m[P'|R']|n[R'']\}$

(Inter Go(in)) $H \equiv (\nu\vec{r})H'\{-|n[R']\}$, $P \xrightarrow{go(in\ n)} P'$ and $R \equiv (\nu\vec{r})H'\{n[P'|R']\}$

(Inter Go(out)) $H \equiv (\nu\vec{r})H'\{n[-|R']\}$, $P \xrightarrow{go(out\ n)} P'$ and $R \equiv (\nu\vec{r})H'\{P'|n[R']\}$

(Inter Open) $H \equiv (\nu\vec{r})H'\{-|n[R']\}$, $P \xrightarrow{open\ n} P'$ and $R \equiv (\nu\vec{r})H'\{P'|R'\}$

(Inter Input) $H \equiv (\nu\vec{r})H'\{-|\langle M\rangle\}$, $P > (\nu\vec{p})\langle(x).P'\rangle P''$ and $R \equiv (\nu\vec{r})H'\{(\nu\{\vec{p}\})(P'\{x \leftarrow M\}|P'')\}$, with $\{\vec{p}\} \cap fn(M) = \emptyset$

(Inter Output) $H \equiv (\nu\vec{r})H'\{-|(x).R'\}$, $P > (\nu\vec{p})\langle\langle M\rangle\rangle P'$ and $R \equiv (\nu\vec{r})H'\{(\nu\{\vec{p}\})(P'|R'\{x \leftarrow M\})\}$, with $\{\vec{p}\} \cap fn(R') = \emptyset$

(Inter Amb) $P > (\nu\vec{p})\langle n[Q]\rangle P'$ and one of the following holds:

[1] $Q \xrightarrow{in\ m} Q'$, $H \equiv (\nu\vec{r})H'\{-|m[R']\}$, $\{\vec{p}\} \cap fn(m[R']) = \emptyset$ and $R \equiv (\nu\vec{r})H'\{(\nu\vec{p})(P'|m[n[Q']|R'])\}$

[2] $Q \xrightarrow{out\ m} Q'$, $H \equiv (\nu\vec{r})H'\{m[-|R']\}$, $m \notin \{\vec{p}\}$, and $R \equiv (\nu\vec{r})H'\{(\nu\vec{p})(n[Q']|m[P'|R'])\}$

[3] $H \equiv (\nu\vec{r})H'\{m[R'|in\ n.R''|-\}$, $\{\vec{p}\} \cap fn(n[R'|in\ n.R'']) = \emptyset$, and $R \equiv (\nu\vec{r})H'\{(\nu\{\vec{p}\})(n[Q|m[R'|R'']]|P')\}$

[4] $H \equiv (\nu\vec{r})H'\{-|open\ n.R'\}$, $n \notin \{\vec{p}\}$, and $R \equiv (\nu\vec{r})H'\{(\nu\{\vec{p}\})(Q|P')|R'\}$

[5] $H \equiv (\nu\vec{r})H'\{R'|go(in\ n).R''|-\}$, $\{\vec{p}\} \cap fn(R'']) = \emptyset$, and $R \equiv (\nu\vec{r})H'\{R'|(\nu\{\vec{p}\})(n[Q|R'']|P')\}$ □

We can prove this extended activity theorem by the same way to the original one proved in Ref. 7).

A.5 Proof of $P_e \simeq Q_e$

This section gives a formal proof of $P_e \simeq Q_e$ of Example 3.5.

Lemma A.5.1 Let P_e and Q_e be the processes in Example 3.5. For any H and m, if $H\{P_e\} \Downarrow m$ then $H\{Q_e\} \Downarrow m$. □

Proof: By induction on the structure of $H\{P_e\} \Downarrow m$, in the same style of the proofs appearing in Ref. 7). As the base of induction, we first consider the case in which m is directly exhibited (Conv Exh), then we proceed to the case in which m will be exhibited after some reductions (Conv Red).

(Conv Exh) Assume $H\{P_e\} \downarrow m$. By Proposition A.4.3, there exist two cases, one is that (a) H exhibit m for any processes, and the other is that (b) P_e exhibits m and for any Q_e such that $Q_e \downarrow n$, $H\{Q_e\} \downarrow n$.
(a) Obviously, $H\{Q_e\} \downarrow m$ holds.
(b) By the definition of P_e and Q_e in Example 3.5 and the Definition 3.6, the only ambient that P_e directly exhibits is $a[]$, and Q_e also exhibits $a[]$.

(Conv Red) As $H\{P_e\} \Downarrow m$, by the Definition 2.6, we have $\exists R.H\{P_e\} \to R$ and $R \Downarrow m$. We assume, as the hypothesis of the structural induction, for any R_{P_e} that can be derived from $H\{P_e\}$ at least one step of transition, if for any m, $R_{P_e} \Downarrow m$ then there exists R_{Q_e} that can be derived from $H\{Q_e\}$ at least one step of transition such that $R_{Q_e} \Downarrow m$.
By Theorem A.4.4, one of three cases pertains:

(Act Proc) By the definition of P_e, there is no transition such that $P_e \to P'$, so this case is impossible.

(Act Har) We have $H \to H'$ with $R \equiv H'\{P_e\}$. By Definition A.4.1, for all S, $H\{S\} \to H'\{S\}$ holds, thus we have $H\{Q_e\} \to H'\{Q_e\}$ particular.
As $H\{Q_e\} \to H'\{Q_e\}$ holds, and by induction hypothesis $H'\{Q_e\} \Downarrow m$ holds, we have $H\{Q_e\} \Downarrow m$.

(Act Inter) We have $H\{P_e\} \rightarrow R$ with $R \equiv H'\{P'\}$. By (Act Inter) of Theorem A.4.4, there are H' and $\vec{r}$ such that $\{\vec{r}\} \cap fn(P_e) = \emptyset$ and one of several conditions of (Act Inter) in the theorem must hold. Since the only hardenings of P_e are $P_e > (\nu\ trash)(\langle a[\cdots|P_1)]\rangle a[\ldots P_2])$, and $P_e > (\nu\ trash)(\langle a[\cdots|P_2)]\rangle a[\ldots P_1])$, only the rule (Inter Amb) applies. According to Theorem A.4.4, there are 5 possibilities in (Inter Amb), but by the definition of P_e, only [3], [4] and [5] are available.

[3]: The only effective capability is "*in a*" in case [3], (any other capability would not cause any transition) so $H \equiv (\nu\ \vec{r})H_1\{n[R_1|\underline{in\ a}.R_2]|-\}$ where $trash \notin fn(n[R'|in\ a.R''))$.
We have $H\{P_e\} \xrightarrow{in\ a} R(\equiv H'\{P'\})$ where $H' \equiv (\nu\ \vec{r})H_1\{-\}$ and

$$
\begin{array}{l}
R \equiv (\nu\ \vec{r})H_1\{(\nu\ trash)\ (\\
\quad a[n[R_1 \mid R_2] \mid in\ trash\ |go(in\ n.out\ n)\\
\qquad .sync\,[out\ trash \mid trash[out\ a] \mid P_1]\\
\quad\ \mid open\ sync\,]\\
\qquad \mid a[in\ trash|go(in\ n.out\ n)\\
\qquad .sync\,[out\ trash \mid trash[out\ a] \mid P_2]\\
\quad\ \mid open\ sync\,]),\\
\text{or}\\
R \equiv (\nu\ \vec{r})H_1\{(\nu\ trash)\ (\\
\quad a[in\ trash|go(in\ n.out\ n)\\
\qquad .sync\,[out\ trash \mid trash[out\ a] \mid P_1]\\
\quad\ \mid open\ sync\,]\\
\qquad \mid a[n[R_1|R_2]|in\ trashgo(in\ n.out\ n)\\
\qquad .sync\,[out\ trash \mid trash[out\ a]|P_2]\\
\quad\ \mid open\ sync\,]).
\end{array}
$$

By induction hypothesis, $H'\{Q'_e\} \Downarrow m$, that is,

$$
\begin{array}{l}
(\nu\ \vec{r})H_1\{(\nu\ trash)\ (\\
\quad a[n[R_1|R_2]|in\ trash|go(in\ n.out\ n)\\
\qquad .sync\,[out\ trash|trash[out\ a]|Q_1]|\\
\quad\ open\ sync\,]\\
\qquad \mid a[in\ trash|go(in\ n.out\ n)\\
\qquad .sync\,[out\ trash|trash[out\ a]|Q_2]\\
\quad\ |open\ sync\,]) \Downarrow m,
\end{array}
\tag{1}
$$

or

$$
\begin{array}{l}
(\nu\ \vec{r})H_1\{(\nu\ trash)\ (\\
\quad a[in\ trash\ |go(in\ n.out\ n)\\
\qquad .sync\,[out\ trash|trash[out\ a]\mid Q_1]\\
\quad\ \mid open\ sync\,]\\
\qquad \mid a[n[R_1 \mid R_2] \mid in\ trash|go(in\ n.out\ n)\\
\qquad .sync\,[out\ trash \mid trash[out\ a]|Q_2]\\
\quad\ \mid open\ sync\,]) \Downarrow m.
\end{array}
\tag{2}
$$

By the way,

$$
\begin{array}{l}
H\{Q_e\} \equiv\\
\quad (\nu\ \vec{r})H_1\{n[R_1 \mid in\ a.R_2]|(\nu\ trash)(\\
\quad a[in\ trash|go(in\ n.out\ n)\\
\quad .sync\,[out\ trash|trash[out\ a]|Q_1]\\
\qquad |open\ sync\,]\\
\quad |a[in\ trash|go(in\ n.out\ n)\\
\qquad .sync\,[out\ trash|trash[out\ a]\mid Q_2]\\
\quad\ \mid open\ sync\,]),
\end{array}
$$

and

$$
\begin{array}{l}
H\{Q_e\} \xrightarrow{in\ a}\\
(\nu\ \vec{r})H_1\{(\nu\ trash)\ (\\
\quad a[n[R_1 \mid R_2] \mid in\ trash|go(in\ n.out\ n)\\
\qquad .sync\,[out\ trash|trash[out\ a]|Q_1]\\
\quad\ |open\ sync\,]\\
\quad |a[in\ trash\ |go(in\ n.out\ n)\\
\qquad .sync\,[out\ trash|trash[out\ a]|Q_2]\\
\quad |open\ sync\,]),
\end{array}
\tag{3}
$$

or

$$
\begin{array}{l}
H\{Q_e\} \xrightarrow{in\ a}\\
(\nu\ \vec{r})H_1\{(\nu\ trash)\ (\\
\quad a[in\ trash|go(in\ n.out\ n)\\
\qquad .sync\,[out\ trash|trash[out\ a]|Q_1]\\
\quad\ \mid open\ sync\,]\\
\quad |a[n[R_1 \mid R_2] \mid in\ trash|go(in\ n.out\ n)\\
\qquad .sync\,[out\ trash|trash[out\ a]|Q_2]\\
\quad\ \mid open\ sync\,]).
\end{array}
\tag{4}
$$

As we can reach (3) or (4) from $H\{Q_e\}$ with the transition '$\xrightarrow{in\ a}$' which is the sole transition of $H\{P_e\}$, and as we indicated by (1) and (2) that (3) and (4) converge to m, we proved $H\{Q_e\} \Downarrow m$ by Definition 2.6.
Proofs for the case for [4] and [5] can be shown with the similar way to the case [3]. □

Lemma A.5.2 Let P_e and Q_e be the processes in Example 3.5. For any H and m, if $H\{Q_e\} \Downarrow m$ then $H\{P_e\} \Downarrow m$. □

Proof: Quite the same method to Lemma A.5.1 is applicable. □

Lemma A.5.1 and Lemma A.5.2 prove the fol-

lowing theorem:

Theorem A.5.3 Let P_e and Q_e be the processes in Example 3.5. Then
$P_e \simeq Q_e$. □

Example A.5.4 This example precisely shows the behaviors of the external choice operator defined in Definition 3.6. When the ambient $n[in\ a.\ in\ b]$ (we call it a traveling ambient) is running parallel to $a[b[]] + a[c[]]$, that is,
$n[in\ a.in\ b] \mid (a[b[]] + a[c[]])$,
the traveling ambient can enter either the ambient $a[\cdots]$ of $a[b[]]$ or $a[\cdots]$ of $a[c[]]$. Suppose it happened to choose the former:

$\xrightarrow{in\ a}$
$(\nu\ trash\ sync)$ (
$a[n[in\ b] \mid in\ trash \mid go(in\ n.out\ n)$
$.sync[out\ trash \mid trash[out\ a]|b[]|open\ sync]$
$\mid a[in\ trash|go(in\ n.out\ n)$
$.sync[out\ trash|trash[out\ a]|c[]]|open\ sync]),$

then, the capability $go(in\ n.out\ n)$ in $a[\cdots b[]]$ is activated and consumed:

$\xrightarrow{go(in\ n,\ out\ n)}$
$(\nu\ trash\ sync)$ (
$a[n[in\ b] \mid in\ trash$
$|sync[out\ trash|trash[out\ a]|b[]]|open\ sync]$
$\mid a[in\ trash \mid go(in\ n.out\ n)$
$.sync[out\ trash|trash[out\ a]|c[]]|open\ sync]),$

then, the capability $open\ sync$ in $a[\cdots b[]]$ dissolves the ambient $sync$ in the same a ambient.

$\xrightarrow{open\ sync}$
$(\nu\ trash\ sync)$ (
$a[n[in\ b]$
$\mid in\ trash \mid out\ trash \mid trash[out\ a] \mid b[]]$
$\mid a[in\ trash|go(in\ n.out\ n)$
$.sync[out\ trash|trash[out\ a]|c[]]|open\ sync]),$

then, the ambients $trash[]$ goes out of the ambient $a[\cdots b[]]$:

$\xrightarrow{out\ a}$
$(\nu\ trash\ sync)$ (
$trash[] \mid a[n[in\ b] \mid in\ trash \mid out\ trash \mid b[]]$
$\mid a[in\ trash \mid go(in\ n.out\ n)$
$.sync[out\ trash|trash[out\ a]|c[]]|open\ sync]).$

Since the ambient $trash[]$ appeared as a sibling of the ambient $a[\cdots b[]]$ and $a[\cdots c[]]$, they can go into the ambient $trash[]$:

$\xrightarrow{in\ trash}\xrightarrow{in\ trash}$
$(\nu\ trash\ sync)$ (
$trash[a[n[in\ b] \mid out\ trash \mid b[]]$
$\mid a[go(in\ n.out\ n)$
$.sync[out\ trash|trash[out\ a]|c[]]|open\ sync]]),$

then, the only ambient which has been chosen by traveling ambient (that is, $a[\cdots b[]]$ in this case) can escape from the ambient $trash[]$:

$\xrightarrow{out\ trash}$
$(\nu\ trash\ sync)$ (
$a[n[in\ b] \mid b[]]$
$\mid trash[a[go(in\ n.out\ n)$
$.sync[out\ trash|trash[out\ a]|c[]]|open\ sync]]).$

Finally, the traveling ambient can reach the destination:

$\xrightarrow{in\ b}$
$(\nu\ trash\ sync)$ (
$a[b[n[]]]$
$\mid trash[a[go(in\ n.out\ n)$
$.sync[out\ trash|trash[out\ a]|c[]]|open\ sync]]).$

The contents of the ambient $trash[\cdots]$ are invisible from the environment, we find that our "+" can behave as an ideal choice operator.
□

A.6 Detailed Structures of Processes in Example 4.4

Here, we have the unfolded structure of P'_e and Q'_e. Using them, We explain the following transitions:

$\mathcal{C}_e(P'_e) \xrightarrow{in\ a} \xrightarrow{go(in\ n)} \xrightarrow{go(out\ n)} \xrightarrow{open\ sync} \xrightarrow{out\ a}$
$\xrightarrow{go(in\ trash)} \xrightarrow{go(in\ trash)} \xrightarrow{go(out\ trash)} \xrightarrow{open\ a}$
$n[!in\ a] \mid !open\ a \mid b$
$\mid (\nu\ trash\ sync)\ (a[P_3] + c[c_2[]] \mid trash[\cdots])$
$(=$ one of $R_p)$.

$P'_e = (\nu\ trash\ sync)($
$a[in\ trash$
$|go(in\ n.out\ n)$
$.sync[out\ trash \mid trash[out\ a] \mid P_3]$
$|open\ sync]$
$|a[in\ trash$
$|go(in\ n.out\ n)$
$.sync[out\ trash \mid trash[out\ a] \mid P_4]$
$|open\ sync]\ ),$

$Q'_e = (\nu\ trash\ sync)($
$a[in\ trash$
$|go(in\ n.out\ n)$
$.sync[out\ trash \mid trash[out\ a] \mid Q_3]$
$|open\ sync]$
$|a[in\ trash$
$|go(in\ n.out\ n)$
$.sync[out\ trash \mid trash[out\ a] \mid Q_4]$
$|open\ sync]\ ).$

We show the transition of $\mathcal{C}_e(P'_e)$. That of $\mathcal{C}_e(Q'_e)$ is similar. After $n[!in\ a]$ goes into the

lower a ambient, $go(in\ n, out\ n)$ is available and consumed as follows:

$$\begin{array}{l}\mathcal{C}_e(P'_e) \xrightarrow{in\ a} \xrightarrow{go(in\ n)} \xrightarrow{go(out\ n)} \\ !open\ a \mid !open\ b \\ \mid (\nu\ trash\ sync)(\\ \quad a[in\ trash \\ \quad\quad \mid go(in\ n.out\ n) \\ \quad\quad\quad .sync\,[out\ trash \mid trash[out\ a] \mid P_3] \\ \quad\quad \mid open\ sync] \\ \quad a[n[!in\ a] \mid in\ trash \\ \quad\quad \mid sync\,[out\ trash \mid trash[out\ a] \mid P_4] \\ \quad\quad \mid open\ sync]\ ).\end{array}$$

Then, the *sync* ambient in the lower a ambient is dissolved as follows:

$$\begin{array}{l}\xrightarrow{open\ sync} \\ !open\ a \mid !open\ b \\ \mid (\nu\ trash\ sync)(\\ \quad a[in\ trash \\ \quad\quad \mid go(in\ n.out\ n) \\ \quad\quad\quad .sync\,[out\ trash|trash[out\ a]|P_3] \\ \quad\quad \mid open\ sync] \\ \quad a[n[!in\ n] \mid in\ trash \mid out\ trash \\ \quad \mid trash[out\ a] \mid P_4]\ ).\end{array}$$

Then, the *trash* ambient in the lower a ambient gets out of the a ambient as follows:

$$\begin{array}{l}\xrightarrow{out\ a} \\ !open\ a \mid !open\ b \\ \mid (\nu\ trash\ sync)(\\ \quad trash[\ ] \\ \quad a[in\ trash\ \ go(in\ n.out\ n) \\ \quad\quad\quad\quad .sync\,[out\ trash|trash[out\ a]|P_3] \\ \quad\quad\quad \mid open\ sync] \\ \quad a[n[!in\ a] \mid in\ trash \mid out\ trash \mid P_4]\ ).\end{array}$$

Then, both the upper and lower a ambient enter the *trash* ambient and only the lower a ambient that was chosen by $n[]$ ambient gets out of the *trash* ambient. So, only the upper a ambient that was not chosen by $n[]$ ambient remains in the *trash* ambient as follows:

$$\begin{array}{l}\xrightarrow{go(in\ trash)} \xrightarrow{go(in\ trash)} \xrightarrow{go(out\ trash)} \\ !open\ a \mid !open\ b \\ \mid (\nu\ trash\ sync)(\\ \quad trash[a[go(in\ n.out\ n) \\ \quad\quad\quad .sync\,[out\ trash|trash[out\ a]|P_3] \\ \quad\quad \mid open\ sync]] \\ \quad a[n[!in\ a] \mid P_4]\ ).\end{array}$$

Here, we *open* the lower a ambient as follows:

$$\begin{array}{l}\xrightarrow{open\ a} \\ !open\ a \mid !open\ b \\ \mid (\nu\ trash\ sync)(\\ \quad trash[a[go(in\ n.out\ n) \\ \quad\quad\quad .sync\,[out\ trash|trash[out\ a]|P_3] \\ \quad\quad \mid open\ sync]] \\ \quad n[!in\ a] \mid P_4\ ) \\ \equiv n[!in\ a] \mid !open\ a \mid !open\ b \\ \mid (\nu\ trash\ sync)((a[P_3] + c[c2[]]) \\ \quad \mid trash[a[go(in\ n.out\ n) \\ \quad\quad\quad .sync\,[out\ trash|trash[out\ a]|P_3] \\ \quad\quad \mid open\ sync]]\ ).\end{array}$$

(Received October 6, 2004)
(Accepted September 2, 2005)
(Online version of this article can be found in the IPSJ Digital Courier, Vol.1, pp.590–603.)

Toru Kato was born in 1965. He received his Ph.D. degree from Okayama University in 1997. Since 1998 he had been a research fellow of the Japan society for the promotion of science. Since 2000 he has been in Kinki University as a lecturer. His current research interests are theoretical study of concurrent processes, implementation of process algebras, and formal semantics of concurrent logic programming languages. He is a member of IPSJ, IEICE and JSSST.