Congruence Results of Behavioral Equivalence for A Graph Rewriting Model of Concurrent Programs with Higher-Order Communication

Masaki Murakami
Department of Computer Science
Graduate School of Natural Science and Technology
Okayama University 3-1-1
Tsushima-Naka, Kita-ku, Okayama
700-0082, Japan
murakami@momo.cs.okayama-u.ac.jp

ABSTRACT: *This paper presents congruence results of a behavioural equivalence on a graph rewriting model of concurrent processes with higher-order communication. A bipartite directed acyclic graph represents a concurrent system that consists of a number of processes and messages in our model. The model presented here makes it possible to represent local names that their scopes are not nested. We show that strong bisimulation equivalence relation is a congruence relation w.r.t. operations that correspond to τ-prefix, input prefix, new-name, replication, composition and application respectively.*

Keywords: Higher-order communication, Bipartite Graph, Graph rewriting model

Received: 11 May 2010, Revised 14 June 2010, Accepted 29 June 2010

1. Introduction

LHO$_\pi$ (Local Higher Order π-calculus) [12] is a formal model of concurrent systems with higher-order communication. It is a subcalculus of higher order π-calculus[11] with asynchronous communication capability. The calculus has the expressive power to represent many practically and/or theoretically interesting examples that include program code transfer.

On the other hand, as we reported in [5, 7, 6], it is difficult to represent the scopes of names of communication channels using models based on process algebra. As well known, it is important to describe "which process knows whose name" for the security of concurrent systems. In other words, it is essential to describe "which process is in the scope for each name in the system" for modeling of distributed systems. In many models based on process algebra, the scope of a name is represented using a binary operation as the ν-operation. However, this method has several problems. For example, the scope of a name is a subterm of an expression if it is represented with ν-operator. So the scopes of bound names are nested (or disjoint) in any π-calculus term. However the scopes of names are not always nested in practical systems.

Furthermore, it is impossible to represent the scope even for one name precisely with ν-operator. Consider the example, $\nu a(\bar{b}a.P\ ) \mid b(x).Q$ where x does not occur in Q. In this example, a is a local name and its scope is $\bar{b}a.P$. The scope of a is extruded by communication with prefixes $\bar{b}a$ and $b(x)$. Then the result of the action is $\nu a(P\ |Q)$ and Q is included in the scope of a. However, as a does not occur in Q, it is equivalent to $(\nu aP)|Q$ by the rules of structural congruence. We cannot see the fact that a is 'leaked' to Q from the resulting expression: $(\nu aP\ )|Q$. This makes difficult to analyze extrusions of scopes of names by executions.

This kind of problem also happens in LHO$_\pi$. In our previous work we presented a model that is based on graph rewriting instead of process algebra as a solution to the problem of representating the scopes of names for firstorder case [5].

The model that we presented is based on graph rewriting system such as [1, 2, 4, 10, 3, 13]. We represent a concurrent program consists of a number of processes (and messages on the way) using a bipartite directed acyclic graph. A bipartite graph is a graph whose nodes are decomposed into two disjoint sets: source nodes and sink nodes such that no two graph nodes within the same set are adjacent. Every edge is directed from a source node to a sink node. The system three processes b_1,b_2 and b_3

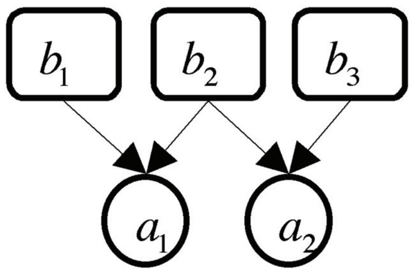

Figure 1. Bipartite Directed Acyclic Graph

and two names $a_i(i = 1, 2)$ shared by b_i and b_{i+1} is represented with a graph as Figure 1. Processes and messages on the way are represented with source nodes. We call source nodes as *behaviors*. In Figure 1, b_1,b_2 and b_3 are behaviors. A behavior has a recursive structure. Namely a behavior is a node consisting of its *epidermis* that denotes the first action and its *content* that denotes the continuation. The operational semantics is defined as a set of rules for graph rewriting. The execution of the an action is represented by "peeling" the epidermis of the behavior node.

We defined an equivalence relation of processes called "scope equivalence" based on scopes of names. We showed the congruence results of a behaviour equivalence [6] and the scope equivalence [8] for first order case. We extended the model for systems with higher-order communication[7] and showed the congruence results of scope equivakece[9] for higher-order model. This paper presents the congruence results of behaviour equivalence for the extended model with higher-order communication.

Congruence results on bisimilarity based on graph rewriting models are reported in [1, 13]. Those studies adopts graph transformation approach for prooftechniques. In this paper, graph rewriting is introduced to extend the model for the representation of scopes of names.

2. The Model

This section presents formal definitions of the model. The intuitive ilustration of the model is presented in [7, 9].

2.1. Programs

First, a countably-infinite set of *names* is presupposed as other formal models based on process algebra.

Definition 2.1 (program, behaviour) Programs and behaviours are defined recursively as follows.

(i) Let $a_1,...,a_k$ are distinct names. *A program* is a a bipartite directed acyclic graph with source nodes $b_1,...,b_m$ and sink nodes $a_1,...,a_k$ such that

- Each source node $b_i(1 \leq i \leq m)$ is a behaviour. Duplicated occurrences of the same behavior are possible.
- Each sink node is a name a_j $(1 \leq j \leq k)$. All a_j's are distinct.
- Each edge is directed from a source node to a sink node. Namely, an edge is an ordered pair (b_i,a_j) of a source node and a name. For any source node bi and a name a_j there is at most one edge from b_i to a_i.

For a program P , we denote the multiset of all source nodes of P as src(P), the set of all sink nodes as snk(P) and the set of all edges as edge(P). Note that the empty graph: **0** such that src(**0**)= snk(**0**)= edge(**0**)= ϕ is a program.

(ii) *A behavior* is an application, a message or a node consists of *the epidermis* and *the content* defined as follows. In the following of this definition, we assume that any element of snk(P) nor x does not occur in anywhere else in the program.

1. A node labeled with a tuple of a name: n (called *the subject of the message*) and an object: o is *a message* and denoted as $n\langle o\rangle$.
2. A tuple of a variable x and a program P is *an abstraction* and denoted as $(x)P$. *An object* is a name or an abstraction.
3. A node labeled with a tuple of an abstraction and an object is *an application*. We denote an application as $A\langle o\rangle$ where A is an abstraction and o is an object.
4. A node whose epidermis is labeled with "!" and the content is a program P is *a replication*, and denoted as $!P$.

5. *An input prefix* is a node (denoted as $a(x).P$) that the epidermis is labeled with a tuple of a name a and a variable x and the content is a program P.
6. *A τ-prefix* is a node (denoted as $\tau.P$) that the epidermis is labeled with a silent action τ and the content is a program P .

Definition 2.2 (locality condition) A program P is *local* if for any input prefix $c(x)$. Q and abstraction $(x)Q$ occurring in P, x does not occur in the epidermis of any input prefix in Q. An abstraction $(x)P$ is local if P is local. A local object is a local abstraction or a name.

The locality condition says that "anyone cannot use a name given from other one to receive messages". Though this condition affects the expressive power of the model, we do not consider that the damage to the expressive power by this restriction is significant. Because as transfer of receiving capability is implemented with transfer of sending capability in many practical example, we consider local programs have enough expressive power for many important/interesting examples. So in this paper, we consider local programs only. Theoretical motivations of this restriction are discussed in [12].

Definition 2.3 (free/bound name)

1. For a behavior or an object p, *the set of free names of p* : fn(p) is defined as follows: $\mathrm{fn}(\mathbf{0})=\phi$, $\mathrm{fn}(a)=\{a\}$ for a name a, $\mathrm{fn}(a\langle o\rangle) = \mathrm{fn}(o) \cup \{a\}$, $\mathrm{fn}((x)P) = \mathrm{fn}(P) \setminus\{x\}$, $\mathrm{fn}(!P)= \mathrm{fn}(P)$, $\mathrm{fn}(\tau.P) = \mathrm{fn}(P)$, $\mathrm{fn}(a(x).P)=(\mathrm{fn}(P) \setminus\{x\}) \cup \{a\}$ and $\mathrm{fn}(o_1\langle o_2\rangle) = \mathrm{fn}(o_1) \cup \mathrm{fn}(o_2)$.
2. For a program P where $\mathrm{src}(P)= \{b_1,...,b_m\}$, $\mathrm{fn}(P) =\bigcup_i \mathrm{fn}(b_i) \setminus \mathrm{snk}(P)$.

The set of *bound names* of P (denoted as bn(P)) is the set of all names that occur in P but not in fn(P) (including elements of snk(P) even if they do not occur in any element of src(P)).

The role of free names is a little bit different from that of π-calculus in our model. For example, a free name x occurs in Q is used as a variable in $(x)Q$ or $a(x).Q$.A channel name that is used for communication with the environments is an element of snk, so it is not a free name.

Definition 2.4 (normal program) A program P is *normal* if for any $b \in \mathrm{src}(P)$ and for any $n \in \mathrm{fn}(b) \cap \mathrm{snk}(P)$, $(b, n) \in \mathrm{edge}(P)$ and any programs occur in b is also normal.

It is quite natural to assume the normality for programs. So in this paper, we consider normal programs only.

Definition 2.5 (composition) Let P and Q be programs such that $\mathrm{src}(P) \cap \mathrm{src}(Q) = \phi$ and $\mathrm{fn}(P) \cap \mathrm{snk}(Q) = \mathrm{fn}(Q) \cap \mathrm{snk}(P) = \phi$. *The composition $P \| Q$* of P and Q is the program such that $\mathrm{src}(P \| Q) = \mathrm{src}(P) \cup \mathrm{src}(Q)$, $\mathrm{snk}(P \| Q) = \mathrm{snk}(P) \cup \mathrm{snk}(Q)$ and $\mathrm{edge}(P \| Q) = \mathrm{edge}(P) \cup \mathrm{edge}(Q)$.

Intuitively, $P \| Q$ is the parallel composition of P and Q. Note that we do not assume $\mathrm{snk}(P) \cap \mathrm{snk}(Q)= \phi$. Obviously $P \| Q = Q \| P$ and $((P \| Q) \| R)=(P \| (Q \| R))$ for any P, Q and R from the definition. The empty graph $\mathbf{0}$ is the unit of "$\|$". Note that $\mathrm{src}(P) \cup \mathrm{src}(Q)$ and $\mathrm{edge}(P) \cup \mathrm{edge}(Q)$ denote the multiset unions while $\mathrm{snk}(P) \cup \mathrm{snk}(Q)$ denotes the set union.

It is easy to show that for normal and local programs P and Q, $P \| Q$ is normal and local.

Definition 2.6 (N-closure) For a normal program P and a set of names N such that $N \cap \mathrm{bn}(P)= \phi$, *the N-closure* $\nu N(P)$ is the program such that $\mathrm{src}(\nu N(P)) = \mathrm{src}(P)$, $\mathrm{snk}(\nu N(P)) = \mathrm{snk}(P) \cup N$ and $\mathrm{edge}(\nu N(P)) = \mathrm{edge}(P) \cup \{(b, n) | b \in \mathrm{src}(P), n \in N\}$.

We sometimes denote $\nu N_1(\cdots (\nu N_i(P)) \cdots)$ as $\nu N_1 \cdots \nu N_i P$ for a program P and sets of names $N_1,...N_i$.

Definition 2.7 (deleting a behaviour) For a normal program P and $b \in \mathrm{src}(P)$, $P \setminus b$ is a program such that $\mathrm{src}(P \setminus b) = \mathrm{src}(P) \setminus\{b\}$, $\mathrm{snk}(P \setminus b) = \mathrm{snk}(P)$ and $\mathrm{edge}(P \setminus b) = \mathrm{edge}(P) \setminus\{(b, n)|(b, n) \in \mathrm{edge}(P)\}$.

Note that $\mathrm{src}(P) \setminus\{b\}$ and $\mathrm{edge}(P) \setminus\{(b, n)|(b, n) \in \mathrm{edge}(P)\}$ mean the multiset subtractions. We can show the following propositions from the definitions.

Proposition 2.1 For normal programs P and Q and a set of names N, $\nu N(P \| Q) = \nu NP \| \nu NQ$ and $\nu N\nu\{m|(b, m) \in \mathrm{edge}(P)\} Q = \nu\{m|(b, m) \in \mathrm{edge}(\nu NP)\}Q$ and $\nu N(P \setminus b) = (\nu NP) \setminus b$ for $b \in \mathrm{src}(P)$.

Definition 2.8 (context) Let P be a program and $b \in \mathrm{src}(P)$ where b is an input prefix, a τ-prefix or a replication and the content of b is $\mathbf{0}$. *A simple first-order context* is the graph $P[\,]$ such that the content $\mathbf{0}$ of b is replaced with *a hole* "[ ]". We call a simple context as *a τ-context*, *an input context* or *a replication context* if the hole is the content of a τ-prefix, of an input prefix or of a replication respectively.

Let P be a program such that $b \in \mathrm{src}(P)$ and b is an application $(x)\boldsymbol{0}\langle Q\rangle$. *A simple application context* P [] is the graph obtained by replacing the behaviour b with $(x)[\ ]\langle Q\rangle$. *A simple context* is a simple first-order context or a simple application context.

A context is a simple context or the graph $P\,[Q[\ ]]$ that is obtained by replacing the hole of P [] replacing with Q[] for a simple context P [] and a context Q[] (with some renaming of the names occur in Q if necessary).

For a context P [] and a program Q, $P\,[Q]$ is the program obtained by replacing the hole in P [] by Q (with some renaming of the names occur in Q if necessary).

2.2 Operational Semantics

We define the operational semantics with a labeled transition system. The substitution of an object to a program, to a behaviour or to an object is defined recursively as follows.

Definition 2.9 (substitution) Let p be a behavior, an object or a program and o be an object. For a name a, we assume that $a \in \mathrm{fn}(p)$. The mapping $[o/a]$ is *a substitution* if $p[o/a]$ is defined as follows respectively.

- For a name c, $c[o/a] = o$ if $c = a$ or $c[o/a] = c$ otherwise.
- For behaviours, $((x)P\,)[o/a]{=}(x)(P\,[o/a])$, $(o_1\langle o_2\rangle)[o/a]{=}\ o_1[o/a]\langle o_2[o/a]\rangle$, $(!P\,)[o/a] = !(P\,[o/a])$, $(c(x).P\,)[o/a]{=}\ c(x).(P\,[o/a])$ and $(\tau.P\,)[o/a]{=}\ \tau.(P\,[o/a])$.
- For a program P and $a \in \mathrm{fn}(P\,)$, $P\,[o/a] = P'$ where P' is a program such that $\mathrm{src}(P') = \{b[o/a]|b \in \mathrm{src}(P)\}$, $\mathrm{snk}(P') = \mathrm{snk}(P)$ and $\mathrm{edge}(P') = \{(b[o/a], n)|(b, n) \in \mathrm{edge}(P)\}$.

For the cases of abstraction and input prefix, note that we can assume $x \neq a$ because $a \in \mathrm{fn}((x)P)$ or $a \in \mathrm{fn}(c(x).P\,)$ without losing generality. (We can rename x if necessary.)

Definition 2.10 Let p be a local program or a local object. A substitution $[a/x]$ is *acceptable* for p if for any input prefix $c(y).Q$ occurring in p, $x \neq c$.

In any execution of local programs, if a substitution is applied by one of the rules of operational semantics then it is acceptable. Thus in the rest of this paper, we consider acceptable substitution only for a program, a context or an abstraction. Namely we assume that $[o/a]$ is applied only for the objects such that a does not occur as a subject of input prefix. This is the reason why $(c(x).P\,)[o/a] = c(x).(P[o/a])$ but not $(c(x).P\,)[o/a] = c[o/a](x).(P[o/a])$ in **Definition 2.9**.

The following proposition is straightforward from the definitions.

Proposition 2.2 Let P and Q are normal programs and o be an object. If M is a set of names such that $M \subset \mathrm{fn}(o)$ and $x \notin (\mathrm{bn}(P) \cup \mathrm{bn}(Q))$, then $\nu M(P \,\|\, Q)[o/x] = (\nu MP[o/x]) \,\|\, (\nu MQ[o/x])$ and $(P \setminus b)[o/x] = P[o/x] \setminus b[o/x]$.

Definition 2.11 (action) For a name a and an object o, *an input action* is a tuple $a(o)$ and *an output action* is a tuple *ahoi*. *An action* is *a silent action* τ, an output action or an input action.

Definition 2.12 (labeled transition) For an action α, $\xrightarrow{\alpha}$ is the least binary relation on normal programs that satisfies the following rules.

input : If $b \in \mathrm{src}(P)$ and $b = a(x).Q$, then $P \xrightarrow{a(o)} (P \setminus b)\|\nu\{n|(b, n) \in \mathrm{edge}(P)\}\ \nu MQ[o/x]$ for an object o and a set of names M such that $\mathrm{fn}(o) \cap \mathrm{snk}(P) \subset M \subset \mathrm{fn}(o) \setminus \mathrm{fn}(P)$.

β-conversion : If $b \in \mathrm{src}(P)$ and $b = (y)Q\langle o\rangle$, then $P \xrightarrow{\tau} (P \setminus b)\|\nu\{n|(b, n) \in \mathrm{edge}(P)\}Q[o/y]$.

τ -action : If $b \in \mathrm{src}(P)$ and $b = \tau.Q$, then $P \xrightarrow{\tau} (P \setminus b)\|\nu\{n|(b, n) \in \mathrm{edge}(P)\}Q$.

replication 1 : $P \xrightarrow{\alpha} P'$ if $!Q = b \in \mathrm{src}(P)$, and $P\|\nu\{n|(b, n) \in \mathrm{edge}(P)\}Q' \xrightarrow{\alpha} P'$, where Q' is a program obtained from Q by renaming all names in $\mathrm{snk}(R)$ to distinct fresh names that do not occur in anywhere else, for all R's where each R is a program that occur in Q (including Q itself).

replication 2 : $P \xrightarrow{\tau} P'$ if $!Q = b \in \mathrm{src}(P)$ and $P\|\nu\{n|(b, n) \in \mathrm{edge}(P)\}(Q_1'\|Q_2') \xrightarrow{\tau} P'$, where each $Q_i'(i = 1, 2)$ is a program obtained from Q by renaming all names in $\mathrm{snk}(R)$ to distinct fresh names that do not occur in anywhere else, for all R's where each R is a program that occur in Q (including Q itself).

output : If $b \in \mathrm{src}(P)$, $b = a\langle v\rangle$ then, $P \xrightarrow{a\langle v\rangle} P \setminus b$.

communication : If $b_1, b_2 \in \mathrm{src}(P)$, $b_1 = a\langle o\rangle$, $b_2 = a(x).Q$ then,
$P \xrightarrow{\tau} ((P \setminus b_1)\setminus b_2)\|\nu\{n\setminus (b_2, n) \in \mathrm{edge}(P)\}\ \nu(\mathrm{fn}(o) \cap \mathrm{snk}(P))Q[o/x]$.

The set of names M that occur freely in o of **input** rule is the set of local names imported by the input action. Some name in M may be new to P, and other may be already known to P but b is not in the scope.

We can show that for any program P, P' and any action α such that $P \xrightarrow{\alpha} P'$, if P is local then P' is local and if P is normal then P' is normal. The following propositions are straightforward from the definitions.

Proposition 2.3 For any normal programs P, P' and Q, and any action α if $P \xrightarrow{\alpha} P'$ then $P \| Q \xrightarrow{\alpha} P' \| Q$.

Proposition 2.4 For any program P,Q and R and any action α, if $P \| Q \xrightarrow{\alpha} R$ is derived by one of **input**, β-**conversion**, τ -**action** or **output** immediately, then $R = P' \| Q$ for some $P \xrightarrow{\alpha} P'$ or $R = P \| Q'$ for some $Q \xrightarrow{\alpha} Q'$.

Definition 2.13 (substitution for action) For an action α and a substitution $[o/x]$, $\alpha[o/x]$ is τ if $\alpha = \tau$, is $a(n[o/x])$ if $\alpha = a(n)$ and is $a[o/x]\langle n\rangle[o/x]$ if $\alpha = a\langle n\rangle$.

Note that $a(n)[o/x] \neq a[o/x](n[o/x])$ because we consider local programs only.

Strong bisimulation relation is defined as usual. It is easy to show ~ defined as **Definition 2.14** is an equivalence relation.

Definition 2.14 (strong bisimulation equivalence) A binary relation R on normal programs is *a strong bisimulation* if for any P and Q such that $(P,Q) \in R$ or $(Q, P) \in R$, for any α and P' if $P \xrightarrow{\alpha} P'$ then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P',Q') \in R$ and for any $Q \xrightarrow{\alpha} Q'$ the similar condition holds. Strong bisimulation equivalence ~ is defined as follows: $P \sim Q$ iff $(P,Q) \in R$ for some strong bisimulation R.

3. Congruence Results

Proposition 3.1 If $\mathrm{src}(P_1) = \mathrm{src}(P_2)$ then $P_1 \sim P_2$.

proof: (outline) We can show that $\{(P_1, P_2)|\mathrm{src}(P_1) = \mathrm{src}(P_2)\}\})$ is a strong bisimulation from the definitions.

Proposition 3.2 Let P be a normal program, α be an action, $[o/x]$ be a substitution that is acceptable to P, $(\mathrm{fn}(o) \cup \mathrm{bn}(o)) \cap \mathrm{bn}(P) = \phi$ and $x \in \mathrm{fn}(P)$ that does not occur elsewhere and M be a set of names such that $M \subset \mathrm{fn}(o)$. If $\nu MP[o/x] \xrightarrow{\alpha} P'$, then one of the followings holds.

- There exists P'' such that $P \xrightarrow{\alpha'} P''$, $\alpha'[o/x] = \alpha'$ and $P''[o/x] \sim P'$,
- there exists P'' such that $P \xrightarrow{x\langle b\rangle} P''$, α is a silent action, $o = (y)R$ and $P' \sim (P''[(y)R/x] \| R[b/y])$ or
- α is a silent action and there exists P'' such that $P \xrightarrow{a\langle n\rangle} \xrightarrow{b\langle m\rangle} P''$, $a[o/x] = b$, $n[o/x] = m[o/x]$ and $P''[o/x] \sim P'$.

proof: (outline) By the induction on the number of **replication 1/2** rules to derive $\nu MP[o/x] \xrightarrow{\alpha} P'$ and **Proposition 2.1, 2.2** and **3.1**.

Note that $a[o/x] = b$ because we assume $b \neq x$ as $[o/x]$ is acceptable for P.

Proposition 3.3 Let P be a normal program, α be an action, o be an object such that $(\mathrm{fn}(o) \cup \mathrm{bn}(o)) \cap \mathrm{bn}(P) = \phi$ and M be a set of names such that $M \subset \mathrm{fn}(o)$. If $P \xrightarrow{\alpha} P'$ and x is a free name of P that does not occur in anywhere else then $P[o/x] \xrightarrow{\alpha[o/x]} \sim P'[o/x]$ or α is an output action $x\langle n\rangle$, o is an abstraction $(y)R$ and $\nu MP[o/x] \xrightarrow{\tau} \sim \nu M(P'[o/x] \| R[n/y])$.

proof: (outline) By the induction on the number of **replication 1/2** rules to derive $P \xrightarrow{\alpha} P'$ and **Proposition 2.1, 2.2** and **3.1**.

Proposition 3.4 Let P be a normal program, α be an output action, β be an input action, $[o/x]$ be a substitution such that $\alpha[o/x] = a\langle n\rangle$, $\beta[o/x] = a(n)$ and $(\mathrm{fn}(o) \cup \mathrm{bn}(o)) \cap \mathrm{bn}(P) = \phi$ and M be a set of names such that $M \subset \mathrm{fn}(o)$. If $P \xrightarrow{\alpha} \xrightarrow{\beta} P'$, then $\nu MP[o/x] \xrightarrow{\tau} \sim P'[o/x]$.

proof: (outline) By the induction on the number of **replication 1/2** rules to derive $P \xrightarrow{\alpha} \xrightarrow{\beta} P'$ and **Proposition 2.1** ~ **2.3** and **3.1**.

Proposition 3.5 For any program P and Q, if $P \xrightarrow{a\langle n\rangle} P'$ and $Q \xrightarrow{a(n)} Q'$ then $P \| Q \xrightarrow{\tau} \sim P' \| Q'$.

proof: (outline) By the induction on the number of **replication 1/2** rules to derive $P \xrightarrow{a\langle n\rangle} P'$ and $Q \xrightarrow{a(n)} Q'$ and **Proposition 3.1**.

Proposition 3.6 If $b \in \mathrm{src}(P)$ and $!Q = b$ then, $P \| \nu\{n|(b, n) \in \mathrm{edge}(P)\}Q' \sim P$ where Q' is a program obtained from Q by renaming names in $\mathrm{bn}(Q)$ to fresh names.

proof:(outline) We have the result by showing the union of ~ and the following relation:

$\{(P \| \nu\{n|(b, n) \in \mathrm{edge}(P)\}Q', P)$ $!Q \in \mathrm{src}(P)$, Q' is obtained from Q by fresh renaming of $\mathrm{bn}(Q)$.$\}$

is a strong bisimulation up to ~.

Now we have the congruence result w.r.t. "||".

Proposition 3.7 For any program R, if $P \sim Q$ then $P \parallel R \sim Q \parallel R$.

proof:(outline) We can show that $\{(P \| R, Q \| R) | P \sim Q\}$ is a bisimulation up to ~. It is shown by induction on the number of **replication 1/2** rules to derive transitions on $P \| R$ and **Proposition 2.1 ~ 2.4, 3.5** and **3.6**.

The following propositions **Proposition 3.8** and **3.9** say that ~ is a congruence relation w.r.t. τ -prefix and replication respectively.

Proposition 3.8 For any P and Q such that $P \sim Q$ and for any τ-context $R[\ ]$, $R[P] \sim R[Q]$.

proof:(outline) By showing $\{(R[P], R[Q]) | P \sim Q, R[\ ]$ is a τ-context.$\} \cup \sim$ is a strong bisimulation using **Proposition 2.3** and **3.7**.

Proposition 3.9 For any P and Q such that $P \sim Q$ and for any replication context $R[\ ]$, $R[P] \sim R[Q]$.

proof:(outline) From **Proposition 2.3** and **3.6**, we can show that:

$\{(R[P], R[Q]) | P \sim Q, R[\ ]$ is a replication context.$\} \cup \sim$ is a strong bisimulation.

Proposition 3.10 For any P and Q such that $P \sim Q$, an object o such that $\mathrm{fn}(o) \cap (\mathrm{snk}(P) \cup \mathrm{snk}(Q)) = \phi$ and a set of names $M \subset \mathrm{fn}(o)$, $\nu MP[o/x] \sim \nu MQ[o/x]$.

proof:(outline) From **Proposition 3.2 ~ 3.4**, **3.7** and **2.3** , we can show that: $\{(\nu MP[o/x], \nu MQ[o/x]) | P \sim Q, \mathrm{fn}(o) \cap (\mathrm{snk}(P) \cup \mathrm{snk}(Q)) = \phi\}$

is a strong bisimulation up to ~.

From this proposition, we have that ~ is a congruence relation w.r.t. input prefix and application.

Proposition 3.11 For any P and Q such that $P \sim Q$ and for any input context $R[\ ]$, $R[P] \sim R[Q]$.

proof: (outline) From **Proposition 3.10**, the union of ~ and $\{(R[P], R[Q]) | P \sim Q, R[\ ]$ is an input context.$\}$ is a strong bisimulation up to ~ using **Proposition 3.7**.

Proposition 3.12 For any P and Q such that $P \sim Q$ and for any simple application context $R[\ ]$, $R[P] \sim R[Q]$.

proof: (outline) We can show that $\{(R[P], R[Q]) | \sim Q\} \cup \sim$ is a strong bisimulation from **Proposition 2.3, 3.7** and **3.10**.

From **Proposition 3.8, 3.9, 3.11** and **3.12**, we have the following result by the induction on the definition of context.

Theorem 3.1 For any P and Q such that $P \sim Q$ and for any context $R[\ ]$, $R[P] \sim R[Q]$.

We considered the context consists of composition, prefix, replication and application for congruence results. In asynchronous π-calculus, the congruence results w.r.t. name restriction "$P \sim Q$ implies $\nu xP \sim \nu xQ$" is also reported. We can show the corresponding result with the similar argument as the first order case [6].

Proposition 3.13 For any P and Q such that $P \sim Q$ and for any set of names M such that $M \cap (\mathrm{bn}(P) \cup \mathrm{bn}(Q)) = \phi$, $\nu M(P) \sim \nu M(Q)$.

4. Conclusions and discussions

This paper presented congruence results of strong bisimulation equivalence w.r.t. composition, τ -context, input context, replication context and application context for the graph rewriting model of concurrent systems.

We did not mention the cases that a hole occurs in the object part of a message or an application because strong bisimulation equivalence is not congruent for these cases. For example, for a context $R_1[\ ]$ that has just one behaviour node which is a message node with a hole $m\langle[\ ]\rangle$, it is obvious that $R_1[P] \not\sim R_2[Q]$ for P and Q such that $P \sim Q$ but $P \neq Q$ because $R_1[P] \xrightarrow{m\langle P\rangle}$ but $R_2[Q] \overset{m\langle P\rangle}{\not\longrightarrow}$. For the case of application, consider the context $R_2[\ ]$ that has just one behaviour $(x)(m\langle x\rangle)\langle[\ ]\rangle$. Then $R_2[P] \not\sim R_2[Q]$ again for P and Q such that $P \sim Q$ but $P \neq Q$ because $R_2[P] \xrightarrow{\tau} R_1[P]$ but $R_2[Q] \xrightarrow{\tau} R_1[Q]$ is the unique transition for $R_2[Q]$. We should adopt other equivalence relation e.g. barbed bisimilarity[11] etc. for the results on such cases.

References

[1] Ehrig, Hartmut., König, Barbara (2006). Deriving Bisimulation Congruences in the DPO Approach to Graph Rewriting with Borrowed Contexts, Mathematical Structures in Computer Science, 16 (6) 1133-1163.

[2] Gadducci, F. (2003). Term Graph rewriting for the π-calculus, Proc. of APLAS ’03 (Programming Languages and Systems), LNCS 2895, p. 37-54.

[3] König, B. (2000). A Graph Rewriting Semantics for the Polyadic π- Calculus, Proc. of GT-VMT ’00 (Workshop on Graph Transformation and Visual Modeling Techniques), p. 451-458.

[4] Milner, R. (2009). *The Space and Mortion of Communicating Systems*, Cambridge University Press.

[5] Murakami, M. (2006). A Formal Model of Concurrent Systems Based on Bipartite Directed Acyclic Graph, *Science of Computer Programming*, Elsevier, 61 p. 38-47.

[6] Murakami, M. (2010). Congruence Results of Behavioral Equivalence for A Graph Rewriting Model of Concurrent Programs, World Review of Science, Technology and Sustainable Development, 7 (1/2) 169 – 180.

[7] Murakami, M.(2008). A Graph Rewriting Model of Concurrent Programs with Higher-Order Communication, In: Proc. of TMFCS. p.80-87.

[8] Murakami, M., Congruence Results of Scope Equivalence for a Graph Rewriting Model of Concurrent Programs, Proc. of ICTAC2008, LNCS 5160, pp. 243-257 (2008)

[9] Murakami, M.(2009). On Congruence Property of Scope Equivalence for Concurrent Programs with Higher-Order Communication, In: Proc of CPA2009, IOS Press, p. 49-66.

[10] Odersky, M.(2000). Functional Nets, European Symposium on Programming 2000, In: Lecture Notes in Computer Science 1782, Springer Verlag.

[11] Sangiorgi, D., Walker, D (2001). *The π-calculus, A Theory of Mobile Processes*, Cambridge University Press.

[12] Sangiorgi, D.(2001). Asynchronous Process Calculi: The Firstand Higher-order Paradigms, *Theoretical Computer Science*, 253, p. 311-350.

[13] Sassone, V., Soboci´nski, P (2005). Reactive systems over cospans, In: Proc. of LICS ’05 IEEE, p. 311-320.