On Congruence Property of Scope Equivalence for 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, Okayama, 700-0082, Japan.

murakami@momo.cs.okayama-u.ac.jp

Abstract. Representation of scopes of names is important for analysis and verification of concurrent systems. However, it is difficult to represent the scopes of channel names precisely with models based on process algebra. We introduced a model of concurrent systems with higher-order communication based on graph rewriting in our previous work. A bipartite directed acyclic graph represents a concurrent system that consists of a number of processes and messages in that model. The model can represent the scopes of local names precisely. We defined an equivalence relation such that two systems are equivalent not only in their behavior, but also in extrusion of scopes of names. This paper shows that our equivalence relation is a congruence relation w.r.t. τ-prefix, new-name, replication and composition, even when higher-order communication is allowed. We also show our equivalence relation is not congruent w.r.t. input-prefix though it is congruent w.r.t. input-prefix in the first-order case.

Keywords. theory of concurrency, π-calculus, bisimilarity, graph rewriting, higher-order communication

Introduction

There are a number of formal models of concurrent systems. In models such as π-calculus [11], "a name" represents, for example, an IP address, a URL, an e-mail address, a port number and so on. Thus, the scopes of names in formal models are important for the security of concurrent systems.

On the other hand, it is difficult to represent the scopes of channel names precisely with models based on process algebra. In many such models based on process algebra, the scope of a name is represented using a binary operation such as the ν-operation. Thus the scope of a name is a subterm of an expression that represents a system. For example, in a π-calculus term: $\nu a_2(\nu a_1(b_1|b_2)|b_3)$, the scope of the name a_2 is the subterm $(\nu a_1(b_1|b_2)|b_3)$ and the scope of the name a_1 is the subterm $(b_1|b_2)$. However, this method has several problems. For example, consider a system **S** consisting of a server and two clients. A client b_1 communicates with the server b_2 using a channel a_1 whose name is known only by b_1 and b_2. And a client b_3 communicates with b_2 using a channel a_2 that is known only by b_2 and b_3. In this system a_1 and a_2 are private names. As b_2 and b_1 knows the name a_1 but b_3 does not, then the scope of

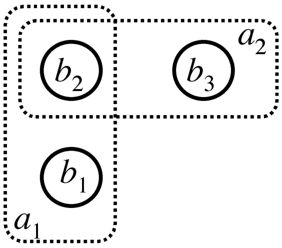


Figure 1. Scopes of names in **S**.

a_1 includes b_1 and b_2 and the scope of a_2 includes b_3 and b_2. Thus the scopes of a_1 and a_2 are not nested as shown in Figure 1.

The method denoting private names as bound names using ν-operator cannot represent the scopes of a_1 and a_2 precisely because scopes of names are subterms of a term and then they are nested (or disjoint) in any π-calculus term.

Furthermore, it is sometimes impossible to represent the scope even for one name precisely with ν-operator. Consider the example, $\nu a(\bar{v}a.P) \mid v(x).Q$ where x does not occur in Q. In this example, a is a private name and its scope is $\bar{v}a.P$. The scope of a is extruded by communication with prefixes $\bar{v}a$ and $v(x)$. 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 rules of structural congruence. We cannot see the fact that a is 'leaked' to Q from the resulting expression: $(\nu aP)|Q$. Thus we must keep the trace of communications for the analysis of scope extrusion. This makes it difficult to analyze extrusions of scopes of names.

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 representing the scopes of names [6]. We defined an equivalence relation on processes called *scope equivalence* such that it holds if two processes are equivalent not only on their behavior but also on the scopes of channel names. We showed the congruence results of weak bisimulation equivalence [7] and of scope equivalence [9] on the graph rewriting model.

On the other hand, a number of formal models with higher-order communication have been reported. LHO$_\pi$ (Local Higher Order π-calculus) [12] is the one of the most well studied model in that area. It is a subcalculus of higher-order π-calculus with asynchronous communication. However the problem of scopes of names also happens in LHO$_\pi$. We need a model with higher-order communication that can represent the scopes of names precisely. We extended the graph rewriting model of [6] for systems with higher-order communication [8]. We extended the congruence results of the behavioral equivalence to the model with higher-order communication [10].

This paper discusses the congruence property of scope equivalence for the graph rewriting model with higher-order communication introduced in [8]. We show that the scope equivalence relation is a congruence relation w.r.t. τ-prefix, new-name, replication and composition even if higher-order communication is allowed as presented in section *4.1*. These results are extensions of the results presented in [9]. On the other hand, in section *4.2*, we show that it is not congruent w.r.t. input-prefix though it is congruent w.r.t. input-prefix in first-order case [9].

Congruence results on bisimilarity based on graph rewriting models are reported in [2,13]. Those studies adopts graph transformation approach for proof techniques. In this paper, graph rewriting is introduced to extend the model for the representation of name scopes.

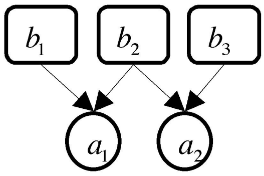


Figure 2. A bipartite directed acyclic graph.

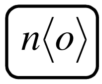


Figure 3. A message node.

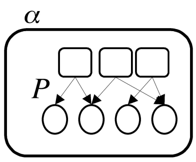
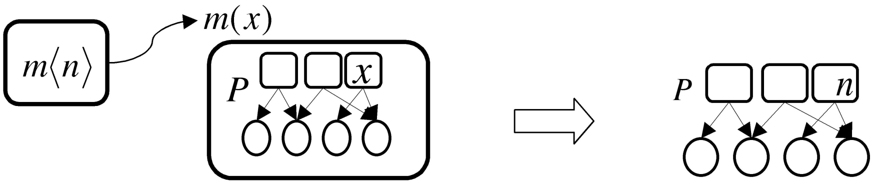


Figure 4. A behavior node $\alpha.P$.



a. A message and its receiver process.

b. After receiving the message.

Figure 5. Message receiving.

1. Basic Idea

Our model is based on graph rewriting system such as [2,3,5,4,13]. We represent a concurrent program that consists of a number of processes (and messages on the way) with 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 of Figure 1 that consists of three processes b_1, b_2 and b_3 and two names $a_i (i = 1, 2)$ shared by b_i and b_{i+1} is represented with a graph as Figure 2.

Processes and messages on the way are represented with source nodes. We call source nodes *behaviors*. In Figure 2, b_1, b_2 and b_3 are behaviors.

message: A behavior node that represents a message is a node labeled with a name of the recipient n (it is called the subject of the message) and the contents of the message o as Figure 3. The contents of the message is a name or a program code as we allow higher-order messages. As a program code to be sent is represented with a graph structure, then the content of a message may have bipartite graph structure also. Thus the message node has a nested structure that has a graph structure inside of the node.

message receiving: A message is received by a receiver process that executes an input action and then continues the execution. We denote a receiver process with a node that consists of its

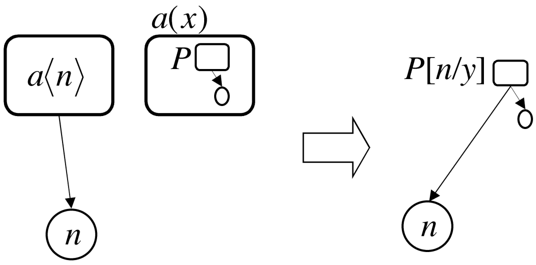


Figure 6. Extrusion of the scope of n.

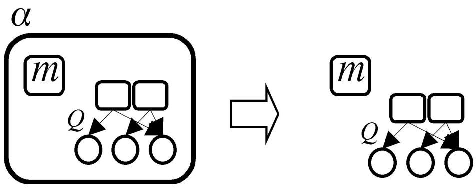


Figure 7. Message sending.

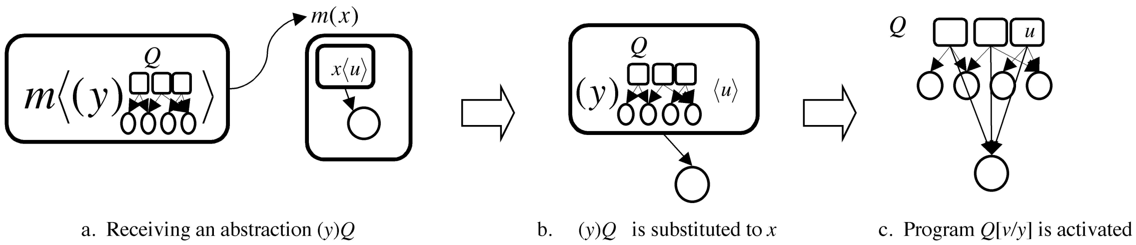


Figure 8. Receiving a program code.

epidermis that denotes the first input action and its *content* that denotes the continuation. For example, a receiver that executes an input action α and then become a program P (denoted as $\alpha.P$ in CCS term) is denoted with a node whose epidermis is labeled with α and the content is P (Figure 4). As the continuation P is a concurrent program, then it has a graph structure inside of the node. Thus the receiver process also has a nested structure.

Message receiving is represented as follows. Consider a message to send an object (a name or an abstraction) n and the receiver with a name m (Figure 5a). The execution of message input action is represented by "peeling the epidermis of the receiver process node". When the message is received then it vanishes, the epidermis of the receiver is removed and the content is exposed (Figure 5b). Now the continuation P is activated. The received object n is substituted to the name x in the content P.

The scope of a name is extruded by message passing. For example, π-calculus has a transition such that $(\nu n a\langle n\rangle)|a(y).P \xrightarrow{\tau} \nu n P[n/y]$. This extrusion is represented by a graph rewriting as Figure 6. A local name n occurs in the message node but there is no edge from the node of the receiver because n is new to the receiver. After receiving the message, as n is a newly imported local name, then a new sink node corresponding to n is added to the graph and new edges are created from each behavior of the continuation to n as the continuation of the receiver is in the scope of n.

message sending: In asynchronous π-calculus, message sending is represented in the same way as process activation. We adopt the similar idea. Consider an example that executes an action α and sends a message m (Figure 7 left). When the action α is executed, then the epidermis is peeled and the message m is exposed as Figure 7 right. Now the message m is transmitted and m can move to the receiver. And the execution of Q continues.

higher-order communications: Consider the case that the variable x occurs as the subject of a message like $x\langle u\rangle$ in the content of a receiver (Figure 8a). If the received object n is a program code, then $n\langle u\rangle$ becomes a program to be activated. As LHO$_\pi$, a program code to transfer is in the form of an abstraction in a message. An abstraction denoted as $(y)Q$ consists of a graph Q representing a program and its input argument y. When an abstraction $(y)Q$ is sent to the receiver and substituted to x in Figure 8a, the behavior node $(y)Q\langle u\rangle$ is exposed and ready to be activated(Figure 8b). To activate $(y)Q\langle u\rangle$, u is substituted to y in Q (Figure 8c). This action corresponds to the β-conversion in LHO$_\pi$. Then we have a program Q with input value u, and it is activated. Note that new edges from each behaviours Q to the sink node which had a edge from $x\langle u\rangle$ are created.

2. Formal Definitions

In this section, we present formal definitions of the model presented informally in the previous section.

2.1. Programs

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

Definition 2.1 (program, behavior) Programs and behaviors are defined recursively as follows.

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

- Each source node $b_i (1 \leq i \leq m)$ is a behavior. 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 b_i 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$(\mathbf{0})$ = snk$(\mathbf{0})$ = edge$(\mathbf{0}) = \emptyset$ 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 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.
2. 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 $n\langle o\rangle$.
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 (local program) A program P is *local* if for any input prefix $c(x).Q$ and any 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: fn$(\mathbf{0}) = \emptyset$, fn$(a) = \{a\}$ for a name a, fn$(a\langle o\rangle) =$ fn$(o) \cup \{a\}$, fn$((x)P) =$ fn$(P) \setminus \{x\}$, fn$(!P) =$ fn(P), fn$(\tau.P) =$ fn(P), fn$(a(x).P) = ($fn$(P) \setminus \{x\}) \cup \{a\}$ and fn$(o_1\langle o_2\rangle) =$ fn$(o_1) \cup$ fn(o_2).
2. For a program P where src$(P) = \{b_1, \ldots, b_m\}$, fn$(P) = \bigcup_i$ fn$(b_i) \setminus$ 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 $b \in$ src(P) and for any $n \in$ fn$(b) \cap$ snk(P), $(b, n) \in$ edge(P) and any program occurs in b is also normal.

It is quite natural to assume the normality for programs, because someone must know a name to use it. In the rest of this paper we consider normal programs only.

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

Intuitively, $P\|Q$ is the parallel composition of P and Q. Note that we do not assume snk$(P) \cap$ snk$(Q) = \emptyset$. 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 src$(P) \cup$ src(Q) and edge$(P) \cup$ edge(Q) denote the multiset unions while snk$(P) \cup$ 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 $N \cap$ bn$(P) = \emptyset$, *the N-closure* $\nu N(P)$ is the program such that src$(\nu N(P)) =$ src(P), snk$(\nu N(P)) =$ snk$(P) \cup N$ and edge$(\nu N(P)) =$ edge$(P) \cup \{(b, n) | b \in$ src$(P), n \in N\}$.

We denote $\nu N_1(\nu N_2(P)))$ as $\nu N_1 \nu N_2(P)$ for a program P and sets of names N_1 and N_2.

Definition 2.7 (deleting a behavior) For a normal program P and $b \in \text{src}(P)$, $P \setminus b$ is a program that is obtained by deleting a node b and edges that are connected with b from P. Namely, $\text{src}(P \setminus b) = \text{src}(P) \setminus \{b\}$, $\text{snk}(P \setminus b) = \text{snk}(P)$ and $\text{edge}(P \setminus b) = \text{edge}(P) \setminus \{(b,n)|(b,n) \in \text{edge}(P)\}$.

Note that $\text{src}(P) \setminus \{b\}$ and $\text{edge}(P) \setminus \{(b,n)|(b,n) \in \text{edge}(P)\}$ mean the multiset subtractions.

Definition 2.8 (context) Let P be a program and $b \in \text{src}(P)$ 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 a graph $P[\,]$ such that the contents $\mathbf{0}$ of b is replaced with *a hole* "[ ]". We call a simple context *a* τ*-context* if the hole is the contents of a τ-prefix, *an input context* if is the contents of an input prefix and *a replication context* if it is the contents of a replication.

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

A context is a simple context or the graph $P[Q[\,]]$ that is obtained by replacing the hole of $P[\,]$ with $Q[\,]$ for a simple context $P[\,]$ and a context $Q[\,]$ (with some renaming of the names which 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 which 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 behavior 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 $a \in \text{fn}(p)$. The mapping $[o/a]$ defined as follows is *a substitution*.

- for a name c, $c[o/a] = \begin{cases} o & \text{if } c = a \\ c & \text{otherwise} \end{cases}$
- for behaviors, $((x)P)[o/a] = (x)(P[o/a])$, $(o_1\langle o_2\rangle)[o/a] = o_1[o/a]\langle o_2[o/a]\rangle$, $(!P)[o/a] = !(P[o/a])$, $(c(x).P)[o/a] = c(x).(P[o/a])$ and $(\tau.P)[o/a] = \tau.(P[o/a])$,
- and for a program P and $a \in \text{fn}(P)$, $P[o/a] = P'$ where P' is a program such that $\text{src}(P') = \{b[o/a]|b \in \text{src}(P)\}$, $\text{snk}(P') = \text{snk}(P)$ and $\text{edge}(P') = \{(b[o/a],n)|(b,n) \in \text{edge}(P)\}$.

For the cases of abstraction and input prefix, note that we can assume $x \neq a$ because $a \in \text{fn}((x)P)$ or $\in \text{fn}(c(x).P)$ 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 the rest of this paper, we consider acceptable substitutions only for a program or an abstraction. Because in any execution of a local programs if a substitution is applied by one of the rules of operational semantics then it is acceptable. Namely we assume that $[o/a]$ is applied only for the objects such that a does not occur as a subject of any input prefix.

It is easy to show that substitution and N-closure can be distributed over "$\|$" and "$\setminus$" from the definitions.

Definition 2.11 (action) For a name a and an object o, *an input action* is a tuple of a and o that is denoted as $a(o)$, and *an output action* is a tuple that is denoted as $a\langle o\rangle$. *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 $P \xrightarrow{a(o)} (P\backslash b)\|\nu\{n|(b,n) \in \mathrm{edge}(P)\}\nu M(Q[o/x])$ 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 elsewhere in P nor programs executed in parallel with P, 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 elsewhere in P nor programs executed in parallel with P, 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,

$$P \xrightarrow{\tau} ((P \setminus b_1) \setminus b_2)\|\nu\{n|(b_2,n) \in \mathrm{edge}(P)\}\nu(\mathrm{fn}(o) \cap \mathrm{snk}(P))(Q[o/x]).$$

In all rules above except **replication 1/2**, the behavior that triggers an action is removed from $\mathrm{src}(P)$. Then the edges from the removed behaviors no longer exist after the action.

The set of names M that occur in the **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 and 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.

Proposition 2.1 For any normal programs P, P' and Q, and any action α if $P \xrightarrow{\alpha} P'$ then $P\|Q \xrightarrow{\alpha} P'\|Q$.

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

Proposition 2.2 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 $Q \xrightarrow{\alpha} Q'$.

proof (outline): Straightforward from the definition.

Proposition 2.3 If $Q \xrightarrow{a\langle o\rangle} Q'$ and $R \xrightarrow{a(o)} R'$ then $Q\|R \xrightarrow{\tau} Q'\|R'$ (and $R\|Q \xrightarrow{\tau} R'\|Q'$) .

proof (outline): By the induction on the total number of **replication 1/2** rules to derive $Q \xrightarrow{\alpha} Q'$ and $R \xrightarrow{\alpha} R'$.

Strong bisimulation relation is defined as usual. It is easy to show $\sim$ defined as **Definition 2.13** is an equivalence relation.

Definition 2.13 (strong bisimulation equivalence) A binary relation $\mathcal{R}$ on normal programs is *a strong bisimulation* if: for any $(P, Q) \in \mathcal{R}$ (or $(Q, P) \in \mathcal{R}$), for any α and P' if $P \stackrel{\alpha}{\rightarrow} P'$ then there exists Q' such that $Q \stackrel{\alpha}{\rightarrow} Q'$ and $(P', Q') \in \mathcal{R}$ ($(Q', P') \in \mathcal{R}$) and for any $Q \stackrel{\alpha}{\rightarrow} Q'$ the similar condition holds.

Strong bisimulation equivalence $\sim$ is defined as

$$\bigcup\{\mathcal{R}|\mathcal{R} \text{ is a strong bisimulation}\}.$$

The following proposition is straightforward from the definitions.

Proposition 2.4 If $\mathrm{src}(P_1) = \mathrm{src}(P_2)$ then $P_1 \sim P_2$.

We can show the congruence results of strong bisimulation equivalence [10] as **Proposition 2.6 - 2.10** and **Theorem 2.1**. First we have the congruence result w.r.t. "$\|$".

Proposition 2.5 For any program R, if $P \sim Q$ then $P\|R \sim Q\|R$.

The following propositions **Proposition 2.6 - 2.9** say that $\sim$ is a congruence relation w.r.t. τ-prefix, replication, input prefix and application respectively.

Proposition 2.6 For any P and Q such that $P \sim Q$ and for any τ-context, $R[P] \sim R[Q]$.

Proposition 2.7 For any P and Q such that $P \sim Q$ and for any replication context, $R[P] \sim R[Q]$.

Proposition 2.8 For any P and Q such that $P \sim Q$ and for any input context, $R[P] \sim R[Q]$.

Proposition 2.9 For any P and Q such that $P \sim Q$ and for any application context $R[\ ]$, $R[P] \sim R[Q]$.

From **Proposition 2.6 - 9**, we have the following result by the induction on the definition of context.

Theorem 2.1 For any P and Q such that $P \sim Q$ and for any context $R[\ ]$, $R[P] \sim R[Q]$.

For asynchronous π-calculus, the congruence results w.r.t. name restriction: "$P \sim Q$ implies $\nu x P \sim \nu x Q$" is reported also. We can show the corresponding result with the similar argument to the first order case [7].

Proposition 2.10 For any P and Q and a set of names N such that $N \cap (\mathrm{bn}(P) \cup \mathrm{bn}(Q)) = \emptyset$, if $P \sim Q$ then $\nu N(P) \sim \nu N(Q)$.

3. Scope Equivalence

This section presents an equivalence relation on programs which ensures that two systems are equivalent in their behavior and for the scopes of names.

Definition 3.1 For a process graph P and a name n such that n, P/n is the program defined as follows:$\mathrm{src}(P/n) = \{b | b \in \mathrm{src}(P), (b, n) \in \mathrm{edge}(P)\}, \mathrm{snk}(P/n) = \mathrm{snk}(P) \setminus \{n\}$ and $\mathrm{edge}(P/n) = \{(b, a) | b \in \mathrm{src}(P/n), a \in \mathrm{snk}(P/n), (b, a) \in \mathrm{edge}(P)\}$.

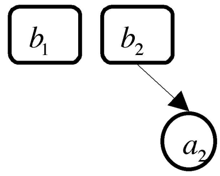


Figure 9. The graph P/a_1.

Intuitively P/n is the subsystem of P that consists of behaviors which are in the scope of n. Let P be an example of Figure 2, P/a_1 is a subgraph of Figure 2 obtained by removing the node of b_3 (and the edge from b_3 to a_2) and a_1 (and the edges to a_1) as shown in Figure 9. It consists of process nodes b_1 and b_2 and a name node a_2.

The following propositions are straightforward from the definitions. We will refer to these propositions in the proof of congruence results w.r.t. to scope equivalence that will be defined below.

Proposition 3.1 For any P, Q and $n \in \text{snk}(P) \cup \text{snk}(Q)$, $(P\|Q)/n = P/n\|Q/n$.

Proposition 3.2 For a program P, a set of names N such that $N \cap \text{bn}(P) = \emptyset$ and $n \in \text{snk}(P)$, $(\nu N(P))/n = \nu N(P/n)$.

Proposition 3.3 Let $R[\ ]$ be a context and P be a program. For any name $m \in \text{snk}(R)$, $(R[P])/m = R/m[P]$.

Definition 3.2 (scope bisimulation) A binary relation $\mathcal{R}$ on programs is *scope bisimulation* if for any $(P, Q) \in \mathcal{R}$,

1. $P = \mathbf{0}$ iff $Q = \mathbf{0}$,
2. $\text{src}(P/n) = \emptyset$ iff $\text{src}(Q/n) = \emptyset$ for any $n \in \text{snk}(P) \cap \text{snk}(Q)$,
3. $P/n \sim Q/n$ for any $n \in \text{snk}(P) \cap \text{snk}(Q)$ and
4. $\mathcal{R}$ is a strong bisimulation.

It is easy to show that the union of all scope bisimulations is a scope bisimulation and it is the unique largest scope bisimulation.

Definition 3.3 (scope equivalence) The largest scope bisimulation is *scope equivalence* and denoted as $\perp$.

It is obvious from the definition that $\perp$ is an equivalence relation. The motivation and the background of the definition of $\perp$ is reported in [6,8]. As $\perp$ is a strong bisimulation from **Definition 3.2**, 4, we have the following proposition.

Proposition 3.4 $P \perp Q$ implies $P \sim Q$.

Definition 3.4 (scope bisimulation up to $\perp$) A binary relation $\mathcal{R}$ on programs is *a scope bisimulation up to* $\perp$ if for any $(P, Q) \in \mathcal{R}$,

1. $P = \mathbf{0}$ iff $Q = \mathbf{0}$,
2. $\text{src}(P/n) = \emptyset$ iff $\text{src}(Q/n) = \emptyset$ for any $n \in \text{snk}(P) \cap \text{snk}(Q)$,
3. $P/n \sim Q/n$ for any $n \in \text{snk}(P) \cap \text{snk}(Q)$ and
4. $\mathcal{R}$ is a strong bisimulation up to $\perp$, namely for any P and Q such that $(P, Q) \in \mathcal{R}$ (or $(Q, P) \in \mathcal{R}$), for any P' such that $P \stackrel{\alpha}{\rightarrow} P'$, there exists Q' such that $Q \stackrel{\alpha}{\rightarrow} Q'$ and $P' \perp\mathcal{R}\perp Q'$ $(Q' \perp\mathcal{R}\perp P'.)$.

The following proposition is straightforward from the definition and the transitivity of "$\perp$".

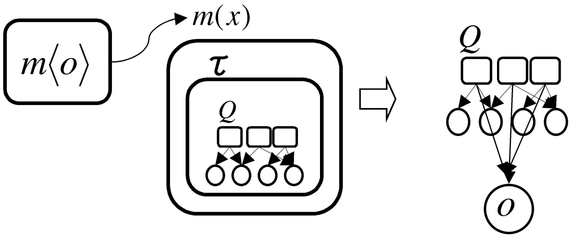


Figure 10. Graph representation of P_1 and $\overline{m}o$.

Proposition 3.5 If $\mathcal{R}$ is a strong bisimulation up to $\perp$, then $\perp\mathcal{R}\perp$ is a scope bisimulation.

Proposition 3.6 If $b \in \mathrm{src}(P)$ and $!Q = b$ then, $P \| \nu\{n|(b,n) \in \mathrm{edge}(P)\}Q' \perp P$ 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 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 $\mathrm{bn}(Q).\} \cup \perp$ is a scope bisimulation up to $\perp$ and **Proposition 3.5**.

Example 3.1 Consider the following (asynchronous) π-calculus processes: $P_1 = m(x).\tau.Q$ and $P_2 = \nu n(m(u).\overline{n}a \mid n(x).Q)$. Assume that neither x nor n occurs in Q. P_1 and P_2 are strongly bisimilar. Consider the case that a message $\overline{m}o$ is received by $P_i (i = 1, 2)$. In P_1, the object o reaches $\tau.Q$ by the execution of $m(o)$. On the other hand, o does not reach to Q in the case of P_2. Assume that o is so confidential that it must not be received by any unauthorized process and Q and $\tau.Q$ are not authorized. (Here, we consider that just receiving is not allowed even if the data is not stored.) Then P_1 is an illegal process but P_2 is not. Thus P_1 and P_2 should be distinguished but they cannot be distinguished with usual behavioral equivalences in π-calculus. Furthermore we cannot see if o reached to unauthorised process or not just from the resulting processes Q and νnQ.

This means that for a system which is implemented with a programming language based on a model such as π-calculus, if someone optimize the system into behavioural equivalent one without taking care of the scopes, the security of the original system may be damaged.

One may say that stronger equivalence relations such as syntactic equivalence or structural congruence work. Of course, syntactic equivalence can distinguish these two cases, but it is not convenient. How about structural congruence? Unfortunately it is not successful. It is easy to give an example such that $P_2 \not\equiv P_3$ but both of them are legal (and behavioural equivalent), for example $P_3 = \nu n(m(u).(\overline{n}_1 a_1|\overline{n}_1 a_2) \mid n_1(x_1).n_2(x_2).Q)$. (Furthermore, we can also give an example of two processes which are structural congruent and one of them is legal but the other is not.)

We can use bipartite directed acyclic graph model presented here to distinguish P_1 and P_2. The example that corresponds to the system consists of P_1 and the message $m\langle o\rangle$ is given by the graph in Figure 10 left [1]. This graph evolves to the graph in Figure 10 right (in the case that o is a name) that corresponds to the π-calculus process Q. This graph explicitly denotes that Q is in the scope of the newly imported name o.

On the other hand the example of P_2 with $m\langle o\rangle$ is Figure 11 left. After receiving the message carrying o, the graph evolves into Figure 11 right. This explicitly shows that Q is not in the scope of o. We can see this difference by showing $P_1 \not\perp P_2$.

One may consider that an equivalence relation similar to $\perp$ can be defined on a model based on process algebra, for example, by encoding a graph into an algebraic term. However it is

[1] The sink nodes corresponding n are not depicted in the following examples.

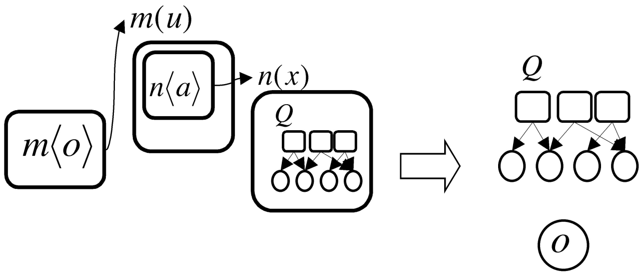


Figure 11. Graph representation of P_2 and $\overline{m}o$.

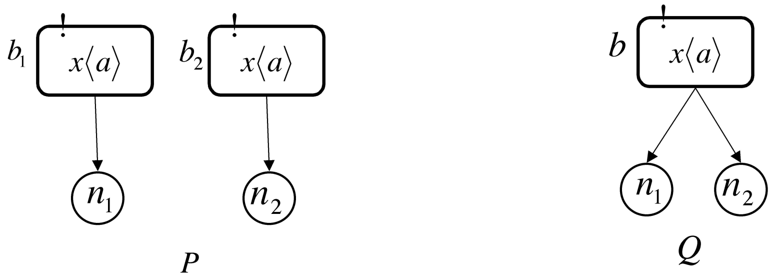


Figure 12. Graph P and Q.

not easy to define an operational semantics on which we can enjoy the merit of algebraic model by naive encoding of the graph model. Especially, it seems difficult to give an orthodox structural operational semantics or reduction semantics that consists of a set of rules to rewrite subterms locally. We consider that we need some tricky idea for encoding.

4. Congruence Results of Scope Equivalence

This section discusses on congruence property of scope equivalence.

4.1. Congruence Results w.r.t. Composition, τ-prefix and Replication

The next proposition says that $\perp$ is a congruence relation w.r.t. $\|$.

Proposition 4.1 If $P \perp Q$ then $P\|R \perp Q\|R$.

proof: See **Appendix I**.

The following proposition is also straightforward from the definitions.

Proposition 4.2 For any program P and Q, let P' and Q' be programs obtained from P and Q respectively by renaming $n \in \text{snk}(P) \cap \text{snk}(Q)$ to a fresh name n'. If $P \perp Q$ then $P' \perp Q'$.

The following proposition is the congruence result of $\perp$ w.r.t. new name.

Proposition 4.3 For any P and Q and a set of names N such that $N \cap (\text{bn}(P) \cup \text{bn}(Q)) = \emptyset$, if $P \perp Q$ then $\nu N(P) \perp \nu N(Q)$.

proof (outline): We show that the following relation is a scope bisimulation:

$$\{(\nu N(P), \nu N(Q)) | P \perp Q\}.$$

It is straightforward from the definition to show **Definition 3.2** 1 and 2. 3. is from **Proposition 3.2** and **Proposition 2.10**. 4. is by the induction on the number of **replication 1/2**.

Proposition 4.4 For any P and Q such that $P \perp Q$ and for any τ-context $R[\,]$, $R[P] \perp R[Q]$.

proof: See **Appendix II**.

Proposition 4.5 For any P and Q such that $P \perp Q$ and for any replication context $R[\,]$, $R[P] \perp R[Q]$.

proof: See **Appendix III**.

4.2. Input and Application Context

We can show that the strong bisimulation equivalence is congruent w.r.t. input prefix context and application context [10]. Unfortunately, this is not the case for the scope equivalence of higher-order programs. Our results show that $\perp$ is not congruent w.r.t. the input context nor the application context. The essential problem is that $\perp$ is not congruent w.r.t. substitutions of abstractions as the following counter example shows.

Example 4.1 (i) Let P be a graph such that src$(P) = \{b_1, b_2\}$, edge$(P) = \{(b_1, n_1), (b_2, n_2)\}$ and snk$(P) = \{n_1, n_2\}$ and Q be a graph such that src$(Q) = \{b\}$, edge$(Q) = \{(b, n_1), (b, n_2)\}$ and snk$(Q) = \{n_1, n_2\}$ where both of b and $b_i(i = 1, 2)$ are $!x\langle a\rangle$ as Figure 12. Note that $n_j(j = 1, 2)$ does not occur in b nor $b_i(i = 1, 2)$.

Lemma 4.1 Let P and Q be as **Example 4.1** (i). Then we have $P \perp Q$.

proof (outline): We show that the relation $\{(P, Q)\}$ is a scope bisimulation. **Definition 3.2**, 1 is obvious as neither P nor Q is an empty graph. For $n_j(j = 1, 2)$, both of P/n_j and Q/n_j are not **0**, so **Definition 3.2**, 2. holds. For 3. P/n_j is the graph such that src$(P/n_j) = \{b_j\}$ and Q/n_j is the graph such that src$(Q/n_j) = \{b\}$. As $b_i = b = !x\langle a\rangle$, src$(P/n_j) =$ src(Q/n_j). From **Proposition 2.4**, $P/n_j \sim Q/n_j$. For 4., it is easy to show that the relation $\{(P, Q)\}$ is a bisimulation because $P \stackrel{x\langle a\rangle}{\rightarrow} P$ and $Q \stackrel{x\langle a\rangle}{\rightarrow} Q$ are the only transition for P and Q respectively.

Example 4.1 (ii) Let P and Q be as **Example 4.1**(i). Now, let o be an abstraction : $(y)c(u).d(v).R$ where R is a program. $P[o/x]$ is the graph such that src$(P) = \{b_1[o/x], b_2[o/x]\}$, snk$(P) = \{n_1, n_2\}$ and edge$(P) = \{(b_1[o/x], n_1), (b_2[o/x], n_2)\}$ as Figure 13a, top. And $Q[o/x]$ is a graph such that src$(Q) = \{b[o/x]\}$, snk$(Q) = \{n_1, n_2\}$ and edge$(Q) = \{(b[o/x], n_1), (b[o/x], n_2)\}$ where $b[o/x]$ and $b_i[o/x](i = 1, 2)$ are $!(y)c(u).d(v).R\langle a\rangle$ as Figure 13b, top.

Lemma 4.2 Let $P[o/x]$ and $Q[o/x]$ be as **Example 4.1** (ii). Then, $P[o/x] \not\perp Q[o/x]$.

proof: See **Appendix IV**.

Note that the object o in the counter example is an abstraction. This incongruence happens only in the case of higher-order substitution. In fact, scope equivalence is congruent w.r.t. substitution of any first-order term by the similar argument as presented in [9].

From **Lemma 4.1** and **4.2**,we have the following results.

Proposition 4.1 There exist P and Q such that $P \perp Q$ but $P[o/x] \not\perp Q[o/x]$ for some object o.

Proposition 4.2 There exist P and Q such that $P \perp Q$ but $I[P] \not\perp I[Q]$ for some input context $I[\,]$.

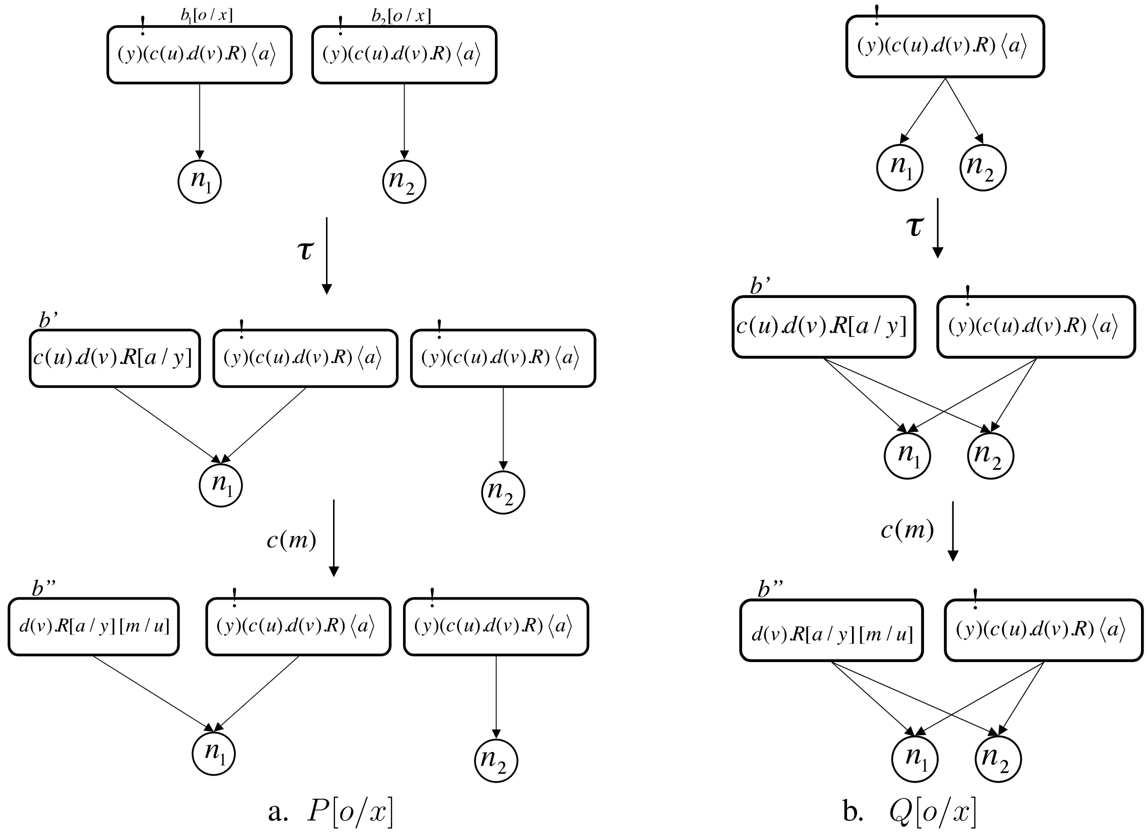


Figure 13. Transitions of $P[o/x]$ and $Q[o/x]$.

proof (outline): Let P and Q be as **Example 4.1** (i) and $I[\,]$ be a input context with a behavior $m(x).[\,]$. Consider the transitions: $I[P] \overset{m(o)}{\rightarrow} P[o/x]$ and $I[Q] \overset{m(o)}{\rightarrow} Q[o/x]$ for o of **Example 4.1** (ii).

Proposition 4.3 There exist P and Q such that $P \perp Q$ but $A[P] \not\perp A[Q]$ for some application context $A[\,]$.

proof: (outline) Let P, Q and o be as **Example 4.1** (ii) and $A[\,]$ be an application context with a behavior $(x)[\,]\langle o\rangle$.

5. Conclusion and Future Work

This paper presented congruence results of scope equivalence w.r.t. new name, composition, τ-prefix and replication for a model with higher-order communication. We also showed that scope equivalence is not congruent w.r.t. input context and application context. As we presented in [9], scope equivalence is congruent w.r.t. input context for first order case. Thus, the non-congruent problem arise from higher-order substitutions. The lack of substitutivity of the equivalence relation makes analysis or verification of systems difficult. We will study this problem by the following approaches as future work.

The first approach is revision of the definition of scope equivalence. The definition of $\perp$ is based on the idea that two process are equivalent if the components that know the name are equivalent for each name. This idea is implemented as the **Definition 3.2**, 3. Our alternative idea for the third condition is $P/N \sim Q/N$ for each subset N of common private names instead of $P/n \sim Q/n$. P and Q in **lemma 4.1** are not equivalent based on this definition. We should study if this alternative definition is suitable or not for the equivalence of processes.

The second approach involves the counter example. As the counter example presented in Section *4.2* is an artificial one, we should study whether there are any practical examples.

Finally, we must reconsider our model of higher-order communication. In our model an output message has the same form as a tuple of a process variable that receives a higher-order term and an argument term. This idea is from LHO_π [12]. One of the main reasons why LHO_π adopts this approach is type theoretical convenience. As we saw in **lemma 4.2**, this identification of output messages and process variables causes the problem with congruence. Thus we should reconsider the model of higher-order communication used.

References

[1] Martin Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: Spi Calculus. *Information and Computation* 148: pp. 1-70, 1999.

[2] Hartmut Ehrig and Barbara König. Deriving Bisimulation Congruences in the DPO Approach to Graph Rewriting with Borrowed Contexts. Mathematical Structures in Computer Science, vol.16, no.6, pp. 1133-1163, 2006.

[3] Fabio Gadducci. Term Graph rewriting for the π-calculus. Proc. of APLAS '03 (Programming Languages and Systems), LNCS 2895, pp. 37-54, 2003.

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

[5] Robin Milner. Bigraphical Reactive Systems, *Proc. of CONCUR'01.* LNCS 2154, Springer, pp. 16-35, 2001.

[6] Masaki Murakami. A Formal Model of Concurrent Systems Based on Bipartite Directed Acyclic Graph. *Science of Computer Programming*, Elsevier, 61, pp. 38-47, 2006.

[7] Masaki Murakami. Congruence Results of Behavioral Equivalence for A Graph Rewriting Model of Concurrent Programs. Proc of ICITA 2008, pp. 636-641, 2008 (to appear in IJTMS, Indersience).

[8] Masaki Murakami. A Graph Rewriting Model of Concurrent Programs with Higher-Order Communication. Proc. of TMFCS 2008, pp. 80-87, 2008.

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

[10] Masaki Murakami. Congruence Results of Behavioral Equivalence for A Graph Rewriting Model of Concurrent Programs with Higher-Order Communication. *Submitted to FST-TCS 2009.*

[11] Davide Sangiorgi and David Walker. *The π-calculus, A Theory of Mobile Processes*. Cambridge University Press, 2001.

[12] Davide Sangiorgi. Asynchronous Process Calculi: The First- and Higher-order Paradigms. *Theoretical Computer Science*, 253, pp. 311-350, 2001.

[13] Vladimiro Sassone and Paweł Sobociński. Reactive Systems over Cospans. Proc. of LICS '05 IEEE, pp. 311-320, 2005.

Appendix I: Proof of Proposition 4.1 (outline)

We can show that the following relation $\mathcal{R}$ is a scope bisimulation.

$$\mathcal{R} = \{(P \| R, Q \| R) | P \perp Q\}$$

Definition 3.2, 1. is straightforward from the definition of "$\|$". The second condition is also straightforward from **Proposition 3.1** and the definition of "$\|$". 3. is from **Proposition 2.5** and **Proposition 3.1**.

4. is by the induction on the number of **replication 1/2** used to derive $P\|R \xrightarrow{\alpha} P'$. If it is derived by one of **input**, β**-conversion**, τ**-action** or **output** immediately, there exists Q' such that $Q\|R \xrightarrow{\alpha} Q'$ from **Proposition 2.2** and **2.1** and $(P', Q') \in \mathcal{R}$ as $\perp$ is a bisimulation.

For the case that it is derived from **communication** rule immediately, we consider two cases. First, if both of b_1 and b_2 are in one of src(P) or src(R), we can show the existence of Q' such that $Q\|R \xrightarrow{\alpha} Q'$ and $P' \perp Q'$ by the similar argument as the cases of **input** etc. mentioned above.

The second case is that one of b_1 and b_2 is in src(P) and the other is in src(R). If b_1 is in src(P), then $P \xrightarrow{a\langle o \rangle} P_1'$ from **output** and $R \xrightarrow{a(o)} R_1'$ from **input**. From $P \perp Q$, $Q \xrightarrow{a\langle o \rangle} Q_1'$ and $P_1' \perp Q_1'$. From **Proposition 2.3**, $Q\|R \xrightarrow{\tau} Q_1'\|R_1'$ and $(P_1'\|R_1', Q_1\|R_1') \in \mathcal{R}$ from the definition. The case b_2 is in src(P) is similar.

Consider the case that $P\|R \xrightarrow{\alpha} P'$ is derived by applying $k+1$ **replication 1/2** rules. If the $k+1$th rule is **replication 1**, $b = !S \in$ src$(P\|R)$.

First we consider $b \in$ src(P). From the premises of **replication 1**, $P\|R\|\nu\{n|(b,n) \in$ edge$(P\|R)\}S' \xrightarrow{\alpha} P'$. As $b \in$ src(P), $\nu\{n|(b,n) \in$ edge$(P)\}S' = \nu\{n|(b,n) \in$ edge$(P\|R)\}S'$. From **Proposition 3.6** and the transitivity of $\perp$,

$$P\|\nu\{n|(b,n) \in \text{edge}(P)\}S' \perp Q.$$

Thus, $(P\|\nu\{n|(b,n) \in$ edge$(P)\}S'\|R, Q\|R) \in \mathcal{R}$. And $P\|R\|\nu\{n|(b,n) \in$ edge$(P)\}S' \xrightarrow{\alpha} P'$ is derived by applying k **replication 1/2** rules. From the inductive hypothesis, there exists Q' such that $Q\|R \xrightarrow{\alpha} Q'$ and $P' \perp Q'$.

If $b \in$ src(R), $\nu\{n|(b,n) \in$ edge$(P\|R)\}S' = \nu\{n|(b,n) \in$ edge$(R)\}S'$. From the premises of **replication 1**, $P\|R\|\nu\{n|(b,n) \in$ edge$(R)\}S' \xrightarrow{\alpha} P'$ with k applications of **replication 1/2**. As $(P\|R\|\nu\{n|(b,n) \in$ edge$(R)\}S', Q\|R\|\nu\{n|(b,n) \in$ edge$(R)\}S') \in \mathcal{R}$, there exists Q' such that $Q\|R\|\nu\{n|(b,n) \in$ edge$(R)\}S' \xrightarrow{\alpha} Q'$ and $P' \perp Q'$ from the inductive hypothesis. As $b = !S \in$ src$(Q\|R)$, $Q\|R \xrightarrow{\alpha} Q'$ by **replication 1**.

The case of **replication 2** is similar.

Appendix II: Proof of Proposition 4.4 (outline)

We have the result by showing that the following relation $\mathcal{R}$ is a scope bisimulation.

$$\mathcal{R} = \{(R[P_1], R[P_2])|P_1 \perp P_2, R[\,] \text{ is a } \tau\text{-context.}\} \cup \perp.$$

To show **Definition 3.2**, 1 is straightforward from the definitions. 2. is from **Proposition 3.3**. 3. is from **Proposition 3.3**, **3.4** and **2.6**.

For 4., we can assume that $R[\,]$ has the form of $\tau.[\,]\|R_1$ where $\tau.[\,]$ is a context that consists of just one behavior node that is a τ-prefix with a hole. Then any transition: $R[P_1] \xrightarrow{\alpha} P_1'$ of $R[P_1]$ is derived by application of τ-rule to $\tau.[P_1]$ or is caused by a transition of R_1. For the first case, P_1' has the form of $\nu N(P_1)\|R_1$. Similarly, there exists a transition for $R[P_2]$ such that $R[P_2] \xrightarrow{\alpha} \nu N(P_2)\|R_1$. As $P_1 \perp P_2$, we have $\nu N(P_1)\|R_1 \perp \nu N(P_2)\|R_1$ from **Proposition 4.1** and **4.3**.

If the transition is derived by applying some rule to R_1, P' has the form of $\tau.[P_1]\|R_1'$ where $R_1 \xrightarrow{\alpha} R_1'$. Then we have $\tau.[P_2]\|R_1 \xrightarrow{\alpha} \tau.[P_2]\|R_1'$ from **Proposition 2.1** and $(\tau.[P_1]\|R_1', \tau.[P_2]\|R_1') \in \mathcal{R}$.

Appendix III: Proof of Proposition 4.5 (outline)

We can show the result by showing the following relation $\mathcal{R}$ is a scope bisimulation up to $\perp$ and **Proposition 3.5**.

$$\{(R[P_1], R[P_2])|P_1 \perp P_2, R[\;] \text{ is a replication context.}\} \cup \perp .$$

To show **Definition 3.4**, 1. is straightforward from the definitions. 2 is from **Proposition 3.3**. 3 is from **Proposition 3.3**, **3.4** and **2.7**.

4. is by the induction on the number of the replication rules to derive $R[P_1] \xrightarrow{\alpha} R_1'$. We can assume that $R[\;]$ has the form of $![\;]\|R_1$ Where $![\;]$ is a context that consists of just one behavior node that is a replication of a hole.

If $R[P_1] = ![P_1]\|R_1 \xrightarrow{\alpha} R_1'$ is derived without any application of **replication 1/2**, that is a transition of R_1. For this case, we can show that there exists R_2' such that $R[P_2] \xrightarrow{\alpha} R_2'$ and $(R_1', R_2') \in \mathcal{R}$ with the similar argument to the proof of **Proposition 4.4**.

Now we go into the induction step. If **replication 1/2** is applied to R_1, then we can show the result by the similar way to the base case again.

We consider the case that $R[P_1] \xrightarrow{\alpha} R_1'$ is derived by **replication 1** for $![P_1]$. Then
$![P_1]\|\nu\{n|(![P_1], n) \in \text{edge}(R[P_1])\}(P_1')\|R_1 \xrightarrow{\alpha} R_1'$
where P_1' is a renaming of P_1. By the induction hypothesis, there exists R_2' such that
$![P_2]\|\nu\{n|(![P_1], n) \in \text{edge}(R[P_1])\}(P_1')\|R_1 \xrightarrow{\alpha} R_2"$
and $(R_1', R_2") \in \mathcal{R}$. From **Proposition 4.2**, $P_1 \perp P_2$ implies $P_1' \perp P_2'$, and we have
$![P_2]\|\nu\{n|(![P_1], n) \in \text{edge}(R[P_1])\}(P_1')\|R_1 \perp ![P_2]\|\nu\{n|(![P_1], n) \in \text{edge}(R[P_1])\}(P_2')\|R_1$
from **Proposition 4.3** and **Proposition 4.1**. From this, we have
$![P_2]\|\nu\{n|(![P_1], n) \in \text{edge}(R[P_1])\}(P_2')\|R_1 \perp ![P_2]\|\nu\{n|(![P_2], n) \in \text{edge}(R[P_2])\}(P_2')\|R_1$
as $\{n|(![P_1], n) \in \text{edge}(R[P_1])\} = \{n|(![P_2], n) \in \text{edge}(R[P_2])\}$.
Then there exists R_2' such that $![P_2]\|\nu\{n|(![P_2], n) \in \text{edge}(R[P_2])\}(P_2')\|R_1 \stackrel{\hat{\alpha}}{\Rightarrow} R_2'$ and $R_2" \perp R_2'$. Thus $(R_1', R_2') \in \mathcal{R} \perp$.

The case of **Replication 2** is similar and then $\mathcal{R}$ is a scope bisimulation up to $\perp$.

Appendix IV: Proof of **lemma 4.2** (outline)

We show that for any relation $\mathcal{R}$, if $(P[o/x], Q[o/x]) \in \mathcal{R}$, then it is not a scope bisimulation. If $\mathcal{R}$ is a scope bisimulation, $\mathcal{R}$ is a strong bisimulation from **Definition 3.2**. Then for any $P[o/x]'$ such that $P[o/x] \xrightarrow{\alpha} P[o/x]'$, there exists $Q[o/x]'$ such that $Q[o/x] \xrightarrow{\alpha} Q[o/x]'$ and $(P[o/x]', Q[o/x]') \in \mathcal{R}$.

From **replication 1** and **β-conversion**, we have $P[o/x]'$ such that: $\text{src}(P[o/x]') = \{b'\} \cup \text{src}(P[o/x])$ where $b' = c(u).d(v).R$, $\text{snk}(P[o/x]') = \text{snk}(P[o/x])$ and $\text{edge}(P[o/x]') = \text{edge}(P[o/x] \cup \{(b', n_1)\}$ for $\alpha = \tau$(Figure 13a, middle). On the other hand, the only transition for $Q[o/x]$ is $Q[o/x] \xrightarrow{\tau} Q[o/x]'$ where $\text{src}(Q[o/x]') = \{b'\} \cup \text{src}(Q[o/x]')$, $b' = c(u).d(v).R$, $\text{snk}(Q[o/x]') = \text{snk}(Q[o/x])$ and $\text{edge}(Q[o/x]') = \text{edge}(Q[o/x] \cup \{(b', n_1), (b', n_2)\}$ (Figure 13b, middle) by **replication 1** and **β-conversion**.

If $\mathcal{R}$ is a scope bisimulation, there exists $Q[o/x]"$ such that $Q[o/x]' \xrightarrow{c(m)} Q[o/x]"$ and $(P[o/x]", Q[o/x]") \in \mathcal{R}$ for any $P[o/x]' \xrightarrow{c(m)} P[o/x]"$. Let $P[o/x]"$ be a graph such that: $\text{src}(P[o/x]') = \{b"\} \cup \text{src}(P[o/x])$ where $b" = d(v).R[m/u]$, $\text{snk}(P[o/x]') = \text{snk}(P[o/x])$ and $\text{edge}(P[o/x]") = \text{edge}(P[o/x]) \cup \{(b", n_1)\}$ obtained by applying **input** rule (Fig-

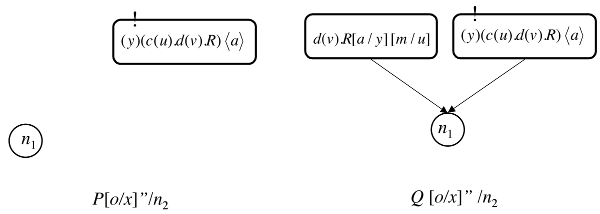


Figure 14. $P[o/x]"/n_2$ and $Q[o/x]"/n_2$.

ure 13a, bottom). The only transition of $Q[o/x]'$ by $c(m)$ makes src$(Q[o/x]') = \{b"\} \cup$ src$(Q[o/x]')$ where $b" = d(v).R[m/u]$, snk$(Q[o/x]') =$ snk$(Q[o/x])$ and edge$(Q[o/x]") =$ edge$(Q[o/x] \cup \{(b", n_1), (b", n_2)\}$ (Figure 13b, bottom).

Then $(P[o/x]", Q[o/x]")$ is in $\mathcal{R}$ if $\mathcal{R}$ is a bisimulation. However, $(P[o/x]", Q[o/x]")$ does not satisfy the condition 3. of **Definition 3.2** because $P[o/x]"/n_2$ and $Q[o/x]"/n_2$ (Figure 14) are not strong bisimilar. Thus $\mathcal{R}$ cannot be a scope bisimulation.