A calculus for concurrent system with higher-order streaming communication

Masaki Murakami

Department of Information and Communication Systems, Graduate School of Natural Science and Technology, Okayama University, 3-1-1, Tsushima-Naka, Okayama 700-0082, Japan

Received 16 March 2004; received in revised form 12 August 2004; accepted 19 October 2004
Available online 13 January 2005

Abstract

This paper presents a formal model of concurrent system that is equipped with capabilities of sending and receiving higher-order terms. That is a modification of the asynchronous higher-order \( \pi \)-calculus. A new operation, input streaming, is introduced. An input process consists of an input stream and a process \( P \). It can receive a higher-order term \( t \) during the execution of \( P \). Input prefix and output process are also modified to represent non-atomic communication. The calculus models computations transferring mobile codes and links on a wide-area network in an asynchronous manner. A labeled transition system (lts) is presented for the operational semantics. Equivalence relations based on the lts are introduced. The equivalences are based on the idea of barbed bisimulation that is suitable for non-atomic/asynchronous communicating systems.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Higher-order \( \pi \)-calculus; Concurrency; Asynchronous communication; Streaming

1. Introduction

In recent years, it is common to send and receive programs using the internet. The capability of sending/receiving of program codes during the computation is represented using the notion of “mobility” of program codes. The mobility is one of the key
technologies for architectures of concurrent/distributed systems. Formal models of mobility are introduced as higher-order calculi of concurrency [6–9]. The term ‘higher-order communication’ means not only transfer of a program that is written in powerful language such as Java. A simple case of web access using a collection of links represented in HTML is also an instance of a higher collection of links (HTML code) as the contents of a message and then the browser executes the HTML code as a program.

In higher-order $\pi$-calculus, a term that represents a set of executable code is transmitted and received as an object of an output/input action. Actions are denoted with an input/output prefix (or an output process and an input prefix in the case of the asynchronous $\pi$-calculus [8,9]). As the execution of an input/output prefix is an atomic action, then any interruption during the sending or receiving of a term is not allowed. However, it is not suitable for the formalization of recent web technology. For example, a usual web browser can display a fragment of a page and make links in the page active by executing a part of HTML code while it is downloading the rest of the page. This situation cannot be represented using the input prefix operation without splitting a set of executable HTML code into fragments of a page. It is not convenient for the design, analysis and verification of code passing concurrent systems. One of the merits of higher-order terms is that an executable program can appear as a data object without any syntactic modification or encoding. We cannot enjoy the merit by splitting a page into fragments and represent them as a set of terms.

This paper presents a formal model of a concurrent system that is equipped with capabilities of sending and receiving higher-order terms. That is a modification of the asynchronous higher-order $\pi$-calculus. A new operation, input streaming [5], is introduced. An input process consists of an input stream and a process $P$. It can receive a higher-order term $t$ during the execution of $P$. The input prefix and the output process are also modified to represent non-atomic communication. The calculus models computations transferring mobile codes and links on a wide area network in an asynchronous manner. A labeled transition system (lts) is presented for the operational semantics. This paper presents equivalence relations using the lts which are based on the idea of the weak barbed bisimulation congruence.

2. Syntax

Let $\mathcal{N}$ be a set of names. A name indicates an object such as a channel, a constant data or a program (a process). A name sometimes indicates the other name. We denote a name as $a, b, \ldots$ or so, if it indicates a channel or a constant data. Names that are not constants are called variables. Variables are denoted using capital letters (or words starting with a capital letter) such as $X, Y, \ldots$. Names indicating constants are denoted as $a, b, c$ or words in lower case letters. A name denoted as $u, v, w, x, y, z, \ldots$ is a variable or a constant.

**Definition 2.1 (Terms and Processes).** A term is defined as follows.

1. An inaction process: $\textbf{0}$ is a term.
2. If $X \in \mathcal{N}$ is a variable, then $X$ is a term. If $a \in \mathcal{N}$ is a constant, then $a$ is a term.
(3) For processes $P_1, P_2, \ldots, P_n$, a parallel process:
\[
(\Pi P_1 P_2 \cdots P_n)
\]
is a term and a choice process:
\[
(\Sigma P_1 P_2 \cdots P_n)
\]
is a term.

(4) Let $x$ be a name, $X_1, \ldots, X_m$ be variables and $t_1, \ldots, t_m$ be terms, then a sender process:
\[
!x(t_1/X_1)
\]
is a term, and an active sender process:
\[
!x[t_1/X_1, \ldots, t_m/X_m]
\]
is a term. An output process is a sender process or an active sender process. In an output process as above, we call each $t_i$ an object of the output process.

(5) Let $x$ be a name, $X_1, \ldots, X_m$ be variables and $P$
\[
(?x(X_1) P)
\]
is a term and an active receiver process:
\[
(?x[X_1, \ldots, X_m] P)
\]
is a term.

(6) Let $x$ be a name, $X_1, \ldots, X_m$ be variables and $P$ be a process, then an input prefix process:
\[
(?x(X_1) \triangleright P)
\]
is a term and an active input prefix process:
\[
(?x[X_1, \ldots, X_m] \triangleright P)
\]
is a term.

(7) a $\tau$-prefix: $(\tau \triangleright P)$ is a term if $P$ is a process.

(8) Let $X$ be a variable and $P$ be a process, then a recursive process:
\[
\mu X. P
\]
is a term.

(9) If $t$ is a term that is not a constant, then it is a process.

$?x[X_1, \ldots, X_n]$ appearing in an active receiver process or an active receiver prefix is called an input stream. We denote the set of all terms constructed from $\mathcal{N}$ as $\mathcal{T}_\mathcal{N}$ or just $\mathcal{T}$ if $\mathcal{N}$ is obvious. We denote the set of all processes on $\mathcal{N}$ as $\mathcal{P}_\mathcal{N}$ or just $\mathcal{P}$. 
Definition 2.2 (Bound Names and Free Names).

1. $X_1$ is bound in $\lambda x(t_1/X_1)$. A name $v$ that occurs in $t_1$ is bound in $\lambda x(t_1/X_1)$ if it is bound in $t_1$.
2. For $i (1 \leq i \leq m)$, each $X_i$ is bound in $\lambda x[t_1/X_1, \ldots, t_m/X_m]$ if and only if $v$ is bound in $t_i$.
3. $X_1$ is bound in $(?x(X_1)P)$ and names bound in $P$ are also bound in $(?x(X_1)P)$.
4. For $i (1 \leq i \leq m)$, $X_i$ is bound in $(?x[X_1, \ldots, X_m]P)$ and $v$ is bound in $P$. Then it is also bound in $(?x[X_1, \ldots, X_m]P)$.
5. $X_1$ is bound in $(?x(X_1)P)$ and names bound in $P$ are also bound in $(?x(X_1)P)$. Let names bound in $P$ and names bound in $P$.
6. For $i (1 \leq i \leq m)$, $X_i$ is bound in $(?x[X_1, \ldots, X_m]P)$ and $v$ is bound in $P$.
7. $X$ is bound in $(\Pi P_1 P_2 \cdots P_n)$ and $(\Sigma P_1 P_2 \cdots P_n)$ if it is bound in $P_i$ for some $i (1 \leq i \leq n)$.
8. $X$ is bound in $\mu X. P$ and names bound in $P$ are also bound in $\mu X. P$.
9. If $X$ is bound in $P$ then it is bound in $(\tau \triangleright P)$.

A name $x$ which is not bound in a term $t$ is free in $t$. We assume that the set of bound names and the set of free names are disjoint. We denote a sequence of terms $t_1, t_2, \ldots, t_n$ as $\bar{t}$. $\bar{t} \setminus s$ denotes the sequence that is obtained deleting $s$ from $\bar{t}$.

Definition 2.3 (Structural Congruence). Let $t_1$ and $t_2$ be terms. If $t_1$ is obtained by renaming bound names in $t_2$ and vice versa, then they are $\alpha$-convertible and denoted $t_1 \equiv_{\alpha} t_2$.

Structural congruence $\equiv$ is the smallest congruence relation that satisfies the following.

1. If $t_1 \equiv_{\alpha} t_2$, then $t_1 \equiv t_2$.
2. $(\Sigma P_1 \cdots P_n) \equiv (\Sigma P_1 \cdots P_n)$.
3. $(\Pi P_1 \cdots P_n) \equiv (\Pi P_1 \cdots P_n)$.
4. $(\Sigma P_1 \cdots P_j \cdots P_n) \equiv (\Sigma P_1 \cdots P_j \cdots P_n)$.
5. $(\Pi P_1 \cdots P_j \cdots P_n) \equiv (\Pi P_1 \cdots P_j \cdots P_n)$.
6. $(\Pi P_1 \cdots (\Pi P_{i_1} \cdots P_{i_m}) \cdots P_n) \equiv (\Pi P_1 \cdots P_{i_1} \cdots P_{i_m} \cdots P_n)$.
7. $(\Sigma P_1 \cdots (\Sigma P_{i_1} \cdots P_{i_m}) \cdots P_n) \equiv (\Sigma P_1 \cdots P_{i_1} \cdots P_{i_m} \cdots P_n)$.
8. $(\lambda x(X)(\lambda y(Y))P) \equiv (\lambda y(Y)(\lambda x(X))P)$.
9. $(\lambda x[X](\lambda y[Y])P) \equiv (\lambda y[Y](\lambda x[X])P)$.
10. $(\lambda x(X)(\lambda y[Y])P) \equiv (\lambda y[Y](\lambda x(X))P)$.
11. If any name in $\bar{X}$ does not occur in $P$, then
   a. $(\lambda x(X)(\Pi P Q)) \equiv (\Pi P (\lambda x(X)Q))$
   b. $(\lambda x(X)(\Pi P Q)) \equiv (\Pi P (\lambda x(X)Q))$.
12. If $x$ does not occur in $P$, then
    $$(?a[\bar{X}]P) \equiv (?a[\bar{X} \setminus x]P)$$
    and
    $$(?a(x)P) \equiv P.$$
3. Operational semantics

Definition 3.1 (Labels, Actions, Bound Names). Let $\mathcal{N}$ be a set of names. A label is $!a[t/X], ?a[t/X] !a'/a'[X'/X]$ or $?a'/a[X'/X]$ where $X, X', a, a' \in \mathcal{N}, t \in T$. A label $l$ is internal if it is $!a[t/X]$ or $?a[t/X]$ and $l$ is external if it is $!a'/a[X'/X]$ or $?a'/a[X'/X]$. $a$ and $X$ are bound in $!a[t/X]$ and $?a[t/X]$. If a name $v$ is bound in $t$ then it is bound in $!a[t/X]$ and $?a[t/X]$. All names occurring in $!a'/a[X'/X]$ or $?a'/a[X'/X]$ except $a$ are bound in the label.

We denote the set of all labels as $L$. Let $\text{Act} = L \cup \{\tau\}$. Note that we do not have the notion of bound outputs with the reason mentioned later.

Definition 3.2 (Labeled Transition System). A labeled transitions system is a tuple $(P, \rightarrow)$ where $\rightarrow \subseteq P \times \text{Act} \times P$ is the smallest relation that satisfies the following axioms.

**Sum**

If any bound name in $\alpha$ does not occur in $P_1, \ldots, P_n$, then

\[
\begin{align*}
\frac{}{(\Sigma P P_1 \cdots P_n) \xrightarrow{\alpha} P'}
\end{align*}
\]

**Parallel**

If any bound name in $\alpha$ does not occur in $P_1, \ldots, P_n$ as a free name, then

\[
\begin{align*}
\frac{}{(\Pi P P_1 \cdots P_n) \xrightarrow{\alpha} (\Pi P' P_1 \cdots P_n)}
\end{align*}
\]

**Restriction**

If $W$ does not occur in $\alpha$,

\[
\begin{align*}
\frac{}{(!a(W)P) \xrightarrow{\alpha} (!a(W)P')}
\end{align*}
\]

**Background**

If $a$ and $W$ do not occur in $\alpha$,

\[
\begin{align*}
\frac{}{(?a[W]P) \xrightarrow{\alpha} (?a[W]P')}
\end{align*}
\]

**Connection:**

**Open:Sender** If $X$ and $a'$ do not occur in $!a(t/W)$, then

\[
\begin{align*}
!a(t/W) \xrightarrow{\alpha'(a[X/W])} !a'[t/X].
\end{align*}
\]

**Open:Receiver** If $X$ and $a'$ do not occur in $?a(W)P$,

\[
\begin{align*}
(?a(W)P) \xrightarrow{\alpha'(a[X/W])} (?a'[X]P[X/W]).
\end{align*}
\]
\textbf{Open : Prefix} If $X$ and $a'$ do not occur in $(\alpha a(W) P)$.

$$(\alpha a(W) \triangleright P) \xrightarrow{\alpha a[X/W]} (\alpha a'[X] \triangleright P[X/W]).$$

\textbf{Establish}

$$
\begin{array}{c}
P \xrightarrow{\alpha a[X/W_1]} P' \xrightarrow{\alpha a[X/W_2]} Q' \\
(\Pi P Q) \xrightarrow{\tau} (\Pi P' Q')
\end{array}
$$

\textbf{Close : Sender} $\alpha [ ] \xrightarrow{\tau} 0$

\textbf{Close : Receiver} $(\alpha [ ] P) \xrightarrow{\tau} P$

\textbf{Close : Prefix} $(\alpha [ ] \triangleright P) \xrightarrow{\tau} P$.

\textbf{Output} Let $t_i = t'_i \theta$ for some substitution $\theta = \{ \tilde{s}/\tilde{Z} \}$. Assume that each of $\tilde{Y} \in \mathcal{N}$ does not occur in $\alpha[\tilde{i}/\tilde{X}]$ and $t_i$ is not marked with $\nu$.

$$
\begin{array}{c}
\alpha[\tilde{i}/\tilde{X}] \xrightarrow{\alpha[t'/\tilde{Y}/X_i]} \alpha[\tilde{i}/\tilde{X} \setminus \tilde{i}, \tilde{V}^\nu/\tilde{Y}, \theta]
\end{array}
$$

where $\tilde{i}/\tilde{X} \setminus \tilde{i}$ denotes the sequence

$$t_1/X_1 t_2/X_2 \cdots t_{i-1}/X_{i-1} t_i t_{i+1}/X_{i+1} \cdots t_n/X_n$$

and $\tilde{V}^\nu$ denotes the sequence

$V_1^\nu, V_2^\nu, \ldots, V_k^\nu$ if $\tilde{V}$ is $V_1, V_2, \ldots, V_k$.

\textbf{Input}

$$
\begin{array}{c}
(\alpha[\tilde{X}] Q) \xrightarrow{\alpha[t'/X_i]} (\alpha[\tilde{X} \setminus X_i, \tilde{Y}] Q[t/X_i])
\end{array}
$$

and

$$
\begin{array}{c}
(\alpha[\tilde{X}] Q) \xrightarrow{\alpha[t'/X_i]} (\alpha[\tilde{X} \setminus X_i, \tilde{Y}] \triangleright Q[t/X_i])
\end{array}
$$

where $\tilde{Y}$ are free variables in $t$, and if $X_i$ occurs as an object of an output process in $Q$ and is marked with $\nu$, then the mark is ignored and removed in $Q[t/X_i]$.

\textbf{Communication}

$$
\begin{array}{c}
P \xrightarrow{\alpha[t'/X]} P' \xrightarrow{\alpha[t'/X]} Q' \\
(\Pi P Q) \xrightarrow{\tau} (\Pi P' Q')
\end{array}
$$

\textbf{Recursive} If $X$ does not occur in $\alpha$

$$
\begin{array}{c}
P(X) \alpha \xrightarrow{\mu X. P(X)} P'(\mu X. P(X))
\end{array}
$$

\textbf{\textcircled{\tt T}} - prefix

$$(\tau \triangleright P) \xrightarrow{\tau} P.$$
Structure  If $\alpha'$ is a renaming of $\alpha$ that is obtained by applying substitution to get $P, Q$ from $P', Q'$,

$$
P \equiv P' \quad \nu \quad \frac{P' \to Q'}{P \alpha' \to Q'}
$$

A name (variable) marked with "\(\nu\)" introduced in the Output rule is to avoid useless communication which just renames bound variables. A "\(\nu\)" marked variable is a newly introduced object and not instantiate by any communication yet. Once the Input rule is applied and the marked variable is instantiated, then the mark is removed and it is allowed to send it as an object.

As we mentioned before, we do not have the notion of bound outputs in our calculus. Then we do not have rules that correspond to Open and Close of $\pi$-calculus. Using a bound output action, it is possible to extend the scope of a bound name in $\pi$-calculus.

In our calculus, a variable occurring as an object of an output process and bound in the context of the output process is renamed to a new bound name and exported as a free variable of the output process. This means that the scope of the bound name is not extended syntactically. However, if an output process $\nu[a[v/X]]$ process performs an action $\nu[a[v'/X]]$ with a new bound name $\nu'$ for example, the continuation of the output process is $\nu[a[v'/v']]$. The newly introduced "$\nu$/\nu'" will work to forward the contents of the bound variable $\nu$. Thus the scope of the bound variable is semantically extended.

**Definition 3.3.** $\Rightarrow \subset P \times \text{Act} \times P$ is defined as:

$$
\hat{\alpha} \Rightarrow \tau \Rightarrow \hat{\alpha} \Rightarrow \tau' \Rightarrow .
$$

For $\alpha \in \text{Act}$, $\hat{\alpha}$ is an empty string if $\alpha = \tau$ and is $\alpha$ otherwise.

In the rules for connection establish (Establish), a channel name is replaced with a fresh name when the connection is established. This means that stream communication should be "one-to-one" in normal operation. However, multiple writer/reader processes such as $(\nu \nu[a[t_1/x] \nu[a[t_2/x] (?a[x]P)])$ are not prohibited syntactically. Most of such examples can be regarded as ill configurations. Consider the normal case that a process starts with all input/output stream inactive (in the form of $\nu \nu(t/X)$ or $\nu \nu(x/W)$) and the execution goes on. It is easy to see that multiple writers/readers do not occur in the process, and if there is a pair of active sender process and active receiver process (or active input prefix process) with the common channel name in the process, then the set of variables in the streams is identical. Otherwise, there is something causing error in the process. Then we can define the normal configuration of processes syntactically.

**Definition 3.4 (Normal Process).** A process $P$ is normal if

- for every active sender process $\nu[a[t_1/X_1, \ldots, t_m/X_m]]$ in $P$,
  - channel name $a$ does not occur elsewhere in $P$, or
  - there is just one other process with channel name $a$ in $P$ and it is $\nu[a[X_1, \ldots, X_m] Q]$ or $\nu[a[X_1, \ldots, X_m] Q] Q$.
- for every active receiver process $\nu[a[X_1, \ldots, X_m] Q]$ in $P$,
  - channel name $a$ does not occur elsewhere in $P$, or
there is just one other process with channel name $a$ in $P$ and it is

$$(\lambda t_1/X_1, \ldots, t_m/X_m)$$

and

- for every active input prefix process $?(a[X_1, \ldots, X_m] \triangleright Q)$ in $P$,
  - channel name $a$ does not occur elsewhere in $P$, or
  - there is just one other process with channel name $a$ and it is
    $$(\lambda t_1/X_1, \ldots, t_m/X_m)).$$

**Proposition 3.5.** If $P$ is normal and $P \xrightarrow{\alpha} P'$ then $P'$ is normal.

### 4. Example

This section presents a representation of a WWW system on the internet. We consider the WWW as a system consisting of a browser and a server. This WWW system is denoted as follows using the calculus.

\[
\text{index-C} = (\Sigma_j (\text{?link}_j \langle \text{click}/\text{CLICK} \rangle \triangleright (\text{?url-of-link}_j \langle \text{Address}/\text{Return} \rangle \\
(\text{?Address}(\text{Page}_j) \text{Page}_j))))
\]

\[
\text{server} = \mu X (\text{?www.foo.net} \langle \text{Address} \rangle \triangleright (\text{?my-address} \langle \text{IndexPage} \rangle \text{IndexPage}))
\]

\[
\text{browser} = (\Pi (\text{?input} \langle \text{URL} \rangle !\text{URL}(\text{my-address}/\text{Address})) \\
(\text{?my-address} \langle \text{IndexPage} \rangle \text{IndexPage}))
\]

\[
\text{WWW} = (\Pi \text{browser server})
\]

The browser gets the URL from a user and sends a request with its address my-address to the URL. The server whose URL is www.foo.net sends the contents of index page index-C to the browser address. The browser displays (executes) the received HTML code of the index page. The index page consists of links to the Page$_j$ ($j = 1, 2$) that receives a click from the user and then downloads and displays the contents of the Page$_j$.

From the **Open : Receiver** rule:

\[
(\text{?input}(\text{URL}) \cdots) \xrightarrow{\text{?input'[u]/URL}} (\text{?input'[u]} \cdots)
\]

then using **Parallel** rules,

\[
\text{browser} \xrightarrow{\text{?input'/input[u]/URL}} (\Pi (\text{?input'[u]}(\text{u(my-address/Address)})) \\
(\text{?my-address}(\text{IndexPage} \text{IndexPage}))
\]

\[
\text{?input'[www.foo.net/u]} \xrightarrow{\Pi \text{www.foo.net}(\text{my-address/Address})} (\text{?my-address} \langle \text{IndexPage} \rangle \text{IndexPage}))
\]

Then using the **Parallel** rule,

\[
\text{WWW} \xrightarrow{\text{?input'(input[u]/URL)} \text{?input'[www.foo.net/u]}}
\]
\[ (\Pi (\Pi !\text{www.foo.net/my-address}/\text{Address}) \\
\quad (\exists \text{my-address}(!\text{IndexPage} \mid \text{IndexPage})) \\
\quad \text{server}) \]

From the **Open**: **Sender** rule, **Output** rule, **Close**: **Sender**, **Parallel** rule, **Recursive** rule and **Communication** rule, we obtain

\[ \xrightarrow{\alpha} (\Pi (\Pi \exists \text{my-address}(!\text{IndexPage} \mid \text{IndexPage})) \\
\quad (\Pi !\text{my-address}(\text{index-C}/\text{IndexP} \mu X \ldots))) \]

Using the **Open**: **Sender** rule, **Parallel** rule, **Close**: **Sender** rule and **Communication** rules again:

\[ \xrightarrow{\alpha} (\Pi \text{index-C server}) \]

From the definition of **index-C** and the **Parallel** rule,

\[ (\Pi \text{index-C server}) \xrightarrow{\ell} (\Pi !\text{link} \mid \text{CLICK} \mid \text{click} \mid \text{C}) \]

Then the result accepts the access to the **Page j**.

5. Equivalence relations

We can define the bisimulation equivalence on \( \mathcal{P} \) in the standard manner [3,4] using the lts as follows.

**Definition 5.1** (**Strong Bisimulation**). \( \mathcal{R} \subset \mathcal{P} \times \mathcal{P} \) is a (strong) bisimulation if for any \( (P, Q) \in \mathcal{R} \), for any \( \alpha \in \text{Act} \) and \( P' \) such that \( P xrightarrow{\alpha} P' \), there exists \( Q' \) that is \( Q xrightarrow{\alpha} Q' \) and \( (P', Q') \in \mathcal{R} \) and vice versa.

**Definition 5.2** (**Strong Bisimulation Equivalence**).

\[ \sim = \bigcup \{ \mathcal{R} | \mathcal{R} \subset \mathcal{P} \times \mathcal{P}, \mathcal{R} \text{ is a (strong) bisimulation} \}. \]

However, the standard bisimulation equivalence is not convenient for discussion of equivalence of processes with code streaming. The first reason is that the (standard) bisimulation equivalence is not congruent (as in the case of \( \pi \)-calculus). Note that \( (?a[X] P) \sim (?a[X] Q) \) and \( (?a[X] \triangleright P) \sim (?a[X] \triangleright Q) \) do not hold in general even if \( P \sim Q \). A typical counterexample is \( P \) and \( (\Sigma X P) \) where \( X \) does not occur in \( P \). It is easy to show that \( P \sim (\Sigma X P) \) but

\[ (?a(X) \triangleright P) \not\sim (?a(X) \triangleright (\Sigma X P)). \]

Let \( Q xrightarrow{\alpha} \) and \( P \not\xrightarrow{\gamma} \). Then

\[ (?a(X) \triangleright P) \xrightarrow{\gamma'} (?a'[X'/X]Q'/X') \not\xrightarrow{\gamma} \]

but

\[ (?a(X) \triangleright (\Sigma X P)) \xrightarrow{\gamma'} (?a'[X'/X]Q'/X') \xrightarrow{\gamma} (\Sigma Q P) xrightarrow{\alpha}. \]
It is similar to the case of the receiver process. Namely,

\[ (?a(X) P) \not\sim (?a(X) (\Sigma X P)). \]

It is easy to show that there exist \( P \) and \( Q \) such that \( P \sim Q \) but \( !a(P/X) \not\sim !a(Q/X) \).

This comes from the fact that a variable (without instantiation) is equal to 0 in the sense of bisimulation equivalence. Then we learn that the notion of bisimulation congruence should be introduced.

**Definition 5.3 (Strong Bisimulation Congruence).** For \( P, Q \in \mathcal{P} \), \( P \sim^c Q \) iff \( C[P] \sim C[Q] \) for any context \( C[\_] \).

It is easy to see that this equivalence relation is too discriminating because it is sensitive to \( \tau \) actions. Then we should introduce a weak version. We should notice that the weak equivalence should abstract not only \( \tau \) actions but also internal labels that are less effective for control/synchronization of processes.

**Definition 5.4 (Observability Predicate).** Let \( P \) be a process and \( a \) be a name.

1. \( P \downarrow !a \) if there exists a process \( P' \) such that
   \[ P \xrightarrow{!a/X} P'. \]
2. \( P \downarrow ?a \) if there exists a process \( P' \) such that
   \[ P \xrightarrow{?a/X} P'. \]

The weak observability for an external label \( l: \downarrow l \) is defined as the composition of \( \Rightarrow \) and \( \downarrow l \).

**Definition 5.5.** For a name \( a \) a relation \( \Rightarrow \subset \mathcal{P} \times \mathcal{P} \) is defined as follows:

\[ (\Rightarrow \cup \Rightarrow) \Rightarrow \]

for some internal label \( !a[t/X] \). The relation \( \Rightarrow \) is defined as

\[ (\Rightarrow \cup \Rightarrow) \Rightarrow . \]

**Definition 5.6 (Streaming Barbed Bisimulation).** \( \mathcal{R} \subset \mathcal{P} \times \mathcal{P} \) is a streaming barbed bisimulation if for any \( (P, Q) \in \mathcal{R} \),

1. If \( P \downarrow !a \) then \( Q \downarrow !a \) and if \( Q \downarrow !a \) then \( P \downarrow !a \).
2. If \( P \downarrow ?a \) then \( Q \downarrow ?a \) and if \( Q \downarrow ?a \) then \( P \downarrow ?a \).
3. If \( P \xrightarrow{!a[t/X]} P' \), then \( Q \xrightarrow{!a} Q' \) and \( (P', Q') \in \mathcal{R} \), and vice versa.
4. If \( P \xrightarrow{?a[t/X]} P' \), then \( Q \xrightarrow{?a} Q' \) and \( (P', Q') \in \mathcal{R} \) and vice versa.
5. If \( P \xrightarrow{\tau} P' \) then \( Q \xrightarrow{\tau} Q' \) and \( (P', Q') \in \mathcal{R} \) and vice versa.
It is easy to show the following properties.

**Proposition 5.7.** Let $I$ be a set of indexes. If $\mathcal{R}_i$ is streaming barbed bisimulation for each $i \in I$, then

1. $\bigcup_i \mathcal{R}_i$ is a streaming barbed bisimulation.
2. $\mathcal{R}_1 \mathcal{R}_2$ is a streaming barbed bisimulation.

**Definition 5.8 (Streaming Barbed Bisimilarity).** Streaming barbed bisimilarity: $\simeq_s$ is the union of all streaming barbed bisimulations.

As the identity relation is a streaming barbed bisimulation and any streaming bisimulation is symmetric, we have the following proposition from Proposition 5.7. (2) and the definition of streaming barbed bisimilarity.

**Proposition 5.9.** $\simeq_s$ is an equivalence relation.

**Proposition 5.10.** $\sim \subset \simeq_s$.

It is easy to see that this inclusion is proper. For example we have $C[P] \simeq_s C[(\Sigma P P)]$ where

$$C[\_] = (\Pi !a(\_) X) (?a(W) (\Pi W ?b(U) c(\_)/0))$$

and $P = !b(0/V) > 0$. In this example, the output process $!a(\_)$ sends $P$ or $(\Sigma P P)$ that are equivalent (in the sense of strong bisimilarity). So the output processes in this example should be regarded as equivalent though they emit syntactically different terms. We can identify these processes using the streaming barbed bisimilarity.

The definition of streaming barbed bisimulation (Definition 5.6) says that if $P \xrightarrow{\alpha} P'$ for an internal label $\alpha$ the $P$ and $P'$ are related. It means that streaming barbed bisimilarity identifies a process $P$ and the continuation $P'$ after transitions with internal labels. It is from the idea that each step of transfer of terms through active streams is regarded as a kind of internal action like $\tau$ and is discriminated only with the channel name. It makes the equivalence relation “$\simeq_s$” too coarse. Consider the following example.

$$P = (\Pi (?a[X] X) (?b(\_) c(\_)/V))$$

$P$ is streaming barbed bisimilar to

$$(\Pi !b(\_) (?b(\_) c(\_)/V))$$

that is the continuation of $P$ after receiving of $!b(\_)$ to $X$ and it is streaming barbed bisimilar to its $\tau$-derivative: $c(\_)/V$. But obviously $(\Pi P R) \simeq_s (\Pi c(\_)/V) R$ does not hold for a process $R$ where $R$ forwards $!b(\_)$ from the channel $c$ to $a$.

The we need to define a congruence relation.

**Definition 5.11 (Streaming Barbed Congruence).** For $P, Q \in \mathcal{P}$, $P \simeq_c^\mathcal{C} Q$ iff for any context $C[\_], C[P] \simeq_c C[Q]$ if both of $C[P]$ and $C[Q]$ are normal.

This congruence relation is useful for discriminating $P$ and $(\Sigma X P)$.
6. Conclusion

This paper presented a new calculus for higher-order concurrent computation. The calculus is an extension of asynchronous higher-order $\pi$-calculus. A new operation, streaming of a higher-order term, is introduced. We presented equivalence relations based on the barbed bisimulation.

References