Electronic Notes in Theoretical Computer Science 138 (2005) 61–77 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Contextual Labelled Semantics for Higher-order Process Calculi

# Yongjian Li[1](#_bookmark0) ,[2](#_bookmark0)

*Institute for Software, Chinese Academy of Sciences, Beijing*

Abstract

In this paper, we study a contextual labelled transition semantics for Higher-Order process calculi. The labelled transition semantics are relatively clean and simple, and corresponding bisimulation equivalence can be easily formulated based on it. Besides we develop a novel approach to reason about behaviours of a higher-order substituted process *P* {*Q/X*}, based on which we can directly prove a very important result – factorisation theorem. To show the correspondence between our semantics and the well-established ones, we characterize our bisimulation in a version of barbed equivalence.

# Introduction

There are three major aims in this paper. The first is to study the principle to design LTS semantics in higher-order calculi, which gives not only the computation steps but also the congruence for higher-order processes. The second aim is to originate an approach to explicitly formulate higher-order substitution. We think higher-order substitution is an essential feature which makes higher order calculus different from first-order calculus. There should be a good formulation for behaviors of a higher-order substituted process *P* {*R/X*}*.* The third aim is to develop new techniques to solve the difficult questions in higher-order calculus such as congruence problems, factorization theorem, and organize the structure of theory with more intuition.

1 This work is supported by NSF project of China under Grant No.60173020.

2 Email: [lyj238@ios.ac.cn](mailto:lyj238@ios.ac.cn)

1571-0661 © 2005 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2005.05.005

Labelled semantics achieved great success in early studies of process cal- culi. Based on labelled semantics, the idea of bisimulation was introduced and formulated by Milner and Park [[1](#_bookmark20)][[2](#_bookmark21)][[3](#_bookmark22)] and has become one of the most important notions in process calculi. When moving to more powerful pro- cess calculi with mobility and higher order features, the traditional labelled semantics become hard to work with, and corresponding bisimulation formu- lations are often much more elaborated, resulting in less satisfactory theories. It seems that two problems made traditional labelled semantics difficult for richer process calculi. The first is passing of local names out of the scope where they are defined. The scope extrusion rule was introduced to cope with this [[4](#_bookmark23)], but it also introduced some delicate structure into the labels and compli- cated the definition of labelled transition relation. The second problem is the meaning of the label of higher order output action in traditional LTS. It only takes into account the object that the process emit, but not the context in which the emitted object is supposed to be used. For example, let us consider

∼−

an output action *P ν c a*⟨*K*⟩ *P* ' in *HOπ*'s LTS [[8](#_bookmark27)], where

−→

∼−

*νca* ⟨*K*⟩ represents the

output object and extruded names, and *P* ' represents the residual of *P.* Al- though the label contains the complex information, the whole transition rule does not contain any information about the context in which emitted object is supposed to be used.

This definition of traditional higher order output transition will cause the non-naturalness problems in the definition of bisimulation. In more detail, when defining some kind of bisimulation R of two example processes *P* , *Q*, we can not use the traditional clause as below to define the matching of the higher-order output actions of the two processes,

−→

*μ*

* Whenever *P* −→

*P* '*,* then *Q*' exists s.t. *Q μ*

*Q*' and *P* 'R*Q*'*,* where *μ* is a

higher-order output action. (1)

Unlike first-order output action, the comparison of labels of higher order output action, for example processes, can not be based on syntactic identity in formulation of bisimulation, that would be too strong for any reasonable semantic equivalence. Thomsen used bisimilarity instead of identity in com- paring labels[[5](#_bookmark24)], following earlier ideas by Astesiano and Boudol [[6](#_bookmark25)][[7](#_bookmark26)], but the resulting equivalence is still too strong.

Sangiorgi had some very illustrative examples of the problems in [[8](#_bookmark27)], [[9](#_bookmark28)]. He pointed out that the separation between the object part (i.e., the pro- cess emitted) and the context of a higher-order output prevents a satisfactory treatment of the channels private to the two, and then causes the problems of higher order bisimulation mentioned above. To avoid this separation between object part and context of an output action, he proposed a context bisimu-

lation as below which explicitly takes into account the context in which the emitted object is supposed to be used.

~ − ∼ −

* + Whenever *P* (*ν b* )*a*⟨*K*1⟩ *P* '*,* then *Q*' exists s.t. *Q* (*ν c* )*a*⟨*K*2⟩ *Q*' and for all *G*

−→

−→

∼ ∼ ∼ ∼

with *f n*(*G*) ∩ (*b* ∪ *c*)= ∅*,* (*νb*) (*P* '|*G*{*K*1*/U* }) R (*νc*) (*Q*'|*G*{*K*2*/U* }) *.* (2)

In [[11](#_bookmark30)], Sewell introduced the contextual point of view of labelled seman- tics and pinpointed the connection between labelled semantics and reduction semantics by making explicit the intuition that labelled transitions capture the possible interactions between a term and a surrounding context. Roughly his idea is that labels of transitions from a process *P* will be contexts that, when put together with *P,* create an occurrence of a reduction rule. As a test of the idea, he shows that the new definition of labelled transition introduces the same bisimulation for a fragment of CCS. Notice that the key difference between Sewell’s contextual transition label and traditional transition label is in that the former comes from the process, but the latter comes from the process’s environment.

In this paper we want to show that the same idea can be successfully applied to higher-order process calculi*.* In our labelled transition semantics,

there are five kinds of labels : *τ, a*(*U* )*.Q*, *m.Q*,

—

*a* ⟨*K*⟩.0,

—

*m*.0. As usual,

*P* −*τ*→ *P* ' represents internal communication as traditional LTS. However

*a*(*U* )*.Q*

'

*P*

*m.Q* '

—

*a*⟨*K*⟩*.*0 '

—

*m.*0 '

−→ *P*

, *P* −→ *P* , *P*

−→ *P* and *P* −→ *P*

represent that *P* can respond

− −

to the test *a*(*U* )*.Q*, *m.Q* , *a* ⟨*K*⟩.0, *m.*0*,* which are fragments of program from

context respectively, and then become *P* '. For example, the higher-order test label *a*(*U* )*.Q* means that the environment can accept an object emitted by *P* at port *a* and provide a continuation *Q* after the interaction, and the object emitted will be used to instantiate the higher-order variable *U* in *Q*.

The second aim is to originate an approach to explicitly formulate higher- order substitution. We think higher-order substitution is an essential feature which makes higher order calculus different from first-order calculus. There should be a good formulation for behaviors of a higher-order substituted pro- cess *P* {*R/X*}*.* In fact, we can gain a very intuitive classification on behaviors of process *P* {*R/X*}, which are either solely due to a part of *P* , or solely due toa copy of *R*, or due to interactions between a component of *P* and one copy of *R,* or due to interactions between two copy of *R*. Despite the classification having clear intuition, people have never formulated it explicitly in higher or- der calculus. The difficulty lies in that *P* is an open process with free variables, which has not ever been dealt with directly. To overcome this shortcoming, we extend our LTS semantics to associate symbolic transition rules for open

processes. For example, we will have transitions such as *P* −*α*→ *P* ' for open

processes, and these transitions describe behaviors which are independent in open variables in *P* . We will note that *P* {*R/X*} −*α*→ *P* '{*R/X*} if *P* −*α*→ *P* ' in our LTS. Besides, we also develop a new technique to explore contexts where open variables are located, which will analyze those behaviors of *P* {*R/X*} that are solely due to a copy of *R*. Note that if *P* = *C*[*X*], and *C* is a static context, and *R* −*α*→ *R*'*,* then *C*[*X*]{*R/X*} −*α*→ *C*[*R*']{*R/X*} in our framework.

The third aim is to develop new techniques to solve the difficult questions in higher-order calculus such as congruence problems, factorization theorem, and organize the structure of theory with more intuition. First of all, we place factorisation theorem as a more fundamental result than other authors in dif- ferent semantics in higher-order calculus. Intuitively, this result tells us that higher-order substitution can be simulated by using trigger substitution and private resource. We directly prove the result just by showing how *P* {*R/X*} and *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐= *R*⟩) simulate each other. The techniques we adopt are closely related with a fine-grain formulation on higher-order substitution. It does not need any use of congruence property anymore, which shows fac- torisation theorem is a rather independent result in higher-order calculus. Furthermore, the congruence property of ≈ w.r.t higher-order substitution is a simple corollary of the factorisation theorem.

The rest of the paper is organized as follows: Section [2](#_bookmark1) states the syntax of the calculi that will be considered in this paper. Section [3](#_bookmark3) states the reduction semantics of our language, and the motivation of our work on the following contextual labelled semantics in detail. Section [4](#_bookmark4) specifies the contextual la- belled transition semantics and introduces the corresponding bisimulation in the calculi*.* Section [5](#_bookmark8) proves the factorisation theorem and congruence proper- ties of the bisimulation. Section [6](#_bookmark15) will show the correspondence between our labelled transition semantics and reduction semantics. Section [7](#_bookmark19) is conclusion and related work.

# Syntax

The language we explored is similar to the second fragment of higher-order *π*−calculi proposed in [[8](#_bookmark27)] by Sangiorgi, which satisfies the following restric- tions: (1) all arities are unary, (2) each process is finitely describable, (3)only guarded choices are permitted, (4)only processes are allowed to transmit (or processes are the only communicated values). In this calculi replication may be avoided because recursion and replication can be simulated using restric- tion and parallel composition in our calculi*,* as shown by Thomsen [[5](#_bookmark24)]*.* But for convenience, we introduce the so-called resource program ⟨*k* ⇐ *P* ⟩ *,* which is a server process referenced by *k* forever and generate a copy of *P* after being trig-

gered once. Let FN (HN ) be the countable infinite set of first-order (higher- order) names. Then FN =*df* {*m*|*m* ∈ FN } *,* HN =*df* {*a*|*a* ∈ HN } *,* N = FN ∪ HN *,* N = FN ∪ HN *.* The special symbol *τ,* which does not occur in N , denotes a silent symbol. We use *K*, *L*, *P* , *Q*, *R* to range over processes, *a*, *b*, ..., to range over higher-order names, *m*, *n,* ... to range over first-order names, *x*, *y*,... to range over names, *U* , *V* , *X, Y* to range over variables, *l, k* to range over FN ∪FN ∪ {*τ* }*, α, μ* to range over N ∪N ∪ {*τ* }*.* By convention,

if *l* /= *τ,* then *l* = *l.*

Definition 2.1 The syntax of our language is as follows:

*P* ::= {*αi.Pi*|*i* ∈ *I*}|*P*1|*P*2|*νaP* |*X*| ⟨*k* ⇐ *P* ⟩

Σ

—

*α* ::= *a* ⟨*P* ⟩ |*a*(*X*)|*l*

Σ

As usual, ∅ is denoted by 0. We write *P* {*Q/X*} for the capture-avoiding higher-order substitution of *Q* for the free occurrences of *X* in *P, fn*(*P* )( *fv*(*P*

)) for the set of free names (variables) occurring in *P, n*(*P* )( *V ars*(*P* ) ) for the set of names (variables) occurring in *P* . We write *Pr* for the set of all the processes, *Prc* for the set of all closed processes. For convenience, we shall treat processes that are *α*−equivalent as syntactically identical processes, and *P* = *Q* means that *P* and *Q* are syntactically identical.

Definition 2.2 A context is obtained when the hole [] replaces an occurrences of 0 in a process-term given by the grammar in Definition [2.1](#_bookmark2). A static context *C* is defined by the BNF grammar: *C* := [] | *C*|*P* | *P* |*C* | *νbC*. And for a context *C*, we define *bn*(*C*) by induction:

* 1. *bn*([]) = ∅
  2. *bn*(*X*)= ∅
  3. *bn*(*C*|*P* )= *bn*(*C*)
  4. *bn*(*P* |*C*)= *bn*(*C*)
  5. *bn*(*νbC*)= *bn*(*C*) ∪ {*b*}

We use *C* to range over contexts, and use *C*[*P* ] to denote the process obtained by replacing [] with *P* in *C*.

# Reduction semantics

In this section, we first introduce reduction semantics of our calculi. The rea- son why we choose reduction semantics as a start lies in three points: (1) The interpretation of reduction semantics is uniformly given by a set of rewrite

rules, a set of reduction contexts in which they may be applied, and a struc- tural congruence. So it can be easily formulated in a style similar to that in contextual calculi. (2) As mentioned before, we adopt the contextual point view to work out a new labelled semantics, and essence of the so-called contex- tual point view is just making explicit the intuition that labelled transitions capture the possible interactions between a term and a surrounding context. So it is helpful to interpret our motivation of the new labeled semantics. (3) If the reduction system is available, the correctness of the new labelled semantics can be shown by proving the correctness between the two semantics.

The structural congruence is as follows:

(i) (*P* |*Q*) |*R* ≡ *P* | (*Q*|*R*); *P* |*Q* ≡ *Q*|*P* ; *P* |0 ≡ *P*

(ii) *νx*0 ≡ 0; *νxνy P* ≡ *νyνx P* ; (*νx P* ) |*Q* ≡ *νx* (*P* |*Q*) *,* if *x* ∈*/ f n*(*Q*)*.*

The reduction relation are the set of the rules as following:

H-COM: Σ{*a*(*U* )*.P, ...*}| Σ{− ⟨*K*⟩ *.Q, ...*} −→ *P* {*K/U* }|*Q*

*a*

F-COM: Σ{*m.P, ...*}| Σ{ − } −→ *P* |*Q*

*m.Q, ...*

R-COM: ⟨*m* ⇐ *P* ⟩| Σ{*m.Q, ...*} −→ *P* | ⟨*m* ⇐ *P* ⟩ |*Q*

*P* −→ *P* ' *P* −→ *P* '

PAR: *P* |*Q* −→ *P* '|*Q* RES: *νx P* −→ *νx P* '

*Q* ≡ *P, P* −→ *P* '*,P* ' ≡ *Q*'

STRUCT:

*Q* −→ *Q*'

Example 3.1 Let *R*1 = *νxa < S > .P, R*2 = *νya*(*U* )*.Q, R* = *R*1|*R*2, and *x* /=

*y, x* ∩ *f n*(*Q*)= ∅*,* obviously, there is one reduction *R* −→ *νx* ((*νyQ*) {*S/U* }|*P* )

As shown above, the process *R*1 can output an object *S*, which possibly contains the local name *x*. If *R*1 is surrounded by a parallel context, i.e., we put *R*1 in parallel with some environment process *X*, then there is one reduction in the term *R*1|*X* involving the interaction between *R*1 and *X* if and only if *X* can perform an input action, i.e., *a*(*U* )*.Y* occurs unguarded in *X* with *a* unrestricted for some *Y.* After the reduction occurs, the effect of the object *S* to *X* will be represented by instantiating with *S* some variable *U* in the continuation part of *X*. In the above example, *R*2 surely can provide an input action at name *a*, and the continuation of it is *νyQ* after the action occurs. Therefore we can view the input action of *X* as a test provided by *X*, by which *X* can input the object emitted by *R*1 at name *a* and pass a continuation *X*' to *R*1. More formally, we can write the test as *a*(*U* )*.X*'*.* For the above example, we can write *R*2 can provide a test *a*(*U* )*.νyQ.* An output action of *R*1 is just its reaction to such a test. So it is natural for us to represent the meaning of the output action of *R*1 by a reaction to an input test provided by its environment. Note that the syntax of the test *a*(*U* )*.X*' is

extracted from the context representing the environment, and this is just the essence of the contextual point view.

On the other side, we can adopt the contextual view to work out the meaning of an input action of a process. But this procedure is much simpler. Say the process *R*2 in the above example, if *R*2 is put in parallel with some environment process *X*, then there is one reduction in the term *R*2|*X* involving the interaction between *R*2 and *X* if and only if *X* can perform an output action, i.e., *a* ⟨*K*⟩ *.Y* occurs unguarded in *X* for some *Y.* The effect of the interaction to *R*2 can be represented by instantiating the continuation of *R*2 with the object *K*. We can view the output action of *X* as a test provided by *X* , by which *X* can output the object *K* at name *a.* An input action of *R*2 is just its reaction to such a test. So we can represent the meaning of the input action of *R*1 by an input test provided by its environment. To uniformly represent tests as a fragment of context, we write such an input test as *a* ⟨*K*⟩ *.*0 instead of *a* ⟨*K*⟩ *.* In the above example, we can write *R*1 can provide a test *a* ⟨*S*⟩ *.*0*.*

Similarly, we can adopt the contextual view to work out the meaning of first-input and first-output actions.

Having introduced the motivation behind the contextual point of view on the meaning of actions of the processes, then we can work out a so-called contextual labelled transition semantics for them. The transition labels in our LTS are *contextual* in the sense that each labelled transtion represents a small program which induces an appropriate reduction.[[11](#_bookmark30)] There are five kinds

—

of labels: *τ, a*(*U* )*.Q*, *a* ⟨*R*⟩ *.*0*, m.Q, m.*0. As expected *P*

−→

−*τ*→ *P* ' represents

—

*a*(*U* )*.Q*

internal communication. However *P* −→

*P* '*, P m.Q*

*P* ', and *P*

*a*⟨*R*⟩*.*0

−→

*P* '*,*

*P* −*m*→*.*0

*P* 'represent that *P* can respond to the test *a*(*U* )*.Q*,

*m.Q,*

—

*a* ⟨*R*⟩ *.*0*,*

and *m.*0 respectively, and then becomes *P* '. In the following section, we will formally introduce our contextual labelled semantics.

# Contextual Labelled Semantics

For notation simplicity, we also extend restriction and parallel operation on labels *a*(*U* )*.Q* and *m.Q* in the following definition of transition semantics.

If *α* = *a*(*U* )*.Q,* if *U* ∈*/ fv*(*P* )*, α*|*P* denotes *a*(*U* )*.*(*Q*|*P* )*,* and *P* |*α* denotes

*a*(*U* )*.*(*P* |*Q*); if *x* ∈*/ f n*(*α*)*, νxα* denotes *a*(*U* )*.νxQ.* Let *α* = *m.Q, α*|*P* denotes

*m.*(*Q*|*P* )*,* and *P* |*α* denotes *m.*(*P* |*Q*);if *x* ∈*/ f n*(*α*)*, νxα* denotes *l.νxQ.* We

define *α*{*R/X*} =*df τ* if *α* = *τ, α*{*R/X*} =*df P* {*R/X*} if *α* = *P* for some *P* .

The labelled transition relation is given by the rules in Table [1](#_bookmark5). together with the rules for the auxiliary relation ↓ used in the side condition for com-

H-Input:Σ{*a*(*U* )*.P, ...*}

—

*a*⟨*K*⟩*.*0

−→ *P* {*K/U* }

H-Output:Σ − *K*⟩ *.P, ...*

{ ⟨

*a*

}

*a*(*U* )*.Q*

*P* |*Q*{*K/U* }

−→

F-Input:Σ{*m.P, ...*} −*m*→*.*0 *P*

—

F-Output:Σ{*m.P, ... m.Q P* |*Q*

*a*(*U* )*.Q*

} −→

*P* −→ *R*

H-Coml: (*P* ' ↓ *a*(*U* )*.Q*)

*P* |*P* ' −*τ*→ *R*

*m.Q*

*P* −→ *R*

F-Coml: *τ*

*P* |*P* ' −→ *R*

*P* −*α*→ *P* '

(*P* ' ↓ *m.Q*)

Parl:

Res:

*P* |*Q* −*α*→ *P* '|*Q P* −*α*→ *P* '

*α*

(*c* ∈*/ f n*(*α*))

*νc.P* −→ *νc.P* '

—

Resource: ⟨*m* ⇐ *P* ⟩ −*m*→*.*0

*P* | ⟨*m* ⇐ *P* ⟩

*τ* -Prefix: Σ{*τ.P, ...*} −*τ*→ *P*

F-Input-Pred:Σ{*m.P, ...*}↓ *m.P*

Σ

H-Input-Pred: {*a*(*U* )*.P, ...*}↓ *a*(*U* )*.P*

*P* ↓ *α*

Res-Pred:

*νc P* ↓ *νc α*

*P* ↓ *α*

(*c* ∈*/ f n*(*α*))

Left-Pred:

*P* |*Q* ↓ *α*|*Q*

(*bv*(*α*) ∈*/ fv*(*Q*))

Resource: ⟨*m* ⇐ *P* ⟩↓ *m.*(*P* | ⟨*m* ⇐ *P* ⟩)

Table 1

Rules for labelled transitions

munication rule.

The auxiliary predicate *P* ↓ *a*(*U* )*.P* ' says that *P* can provide the test

*a*(*U* )*.P* '. In this case it is the same as the commitment predicate in [[15](#_bookmark33)]. The

*a*(*U* )*.Q*

*P*

communication rule

−→ *R*

(*P* ' ↓ *a*(*U* )*.Q*) says that if *P* can react to

*P* |*P* ' −*τ*→ *R*

the test *a*(*U* )*.Q* and become *R*, and *P* ' can provide the test, then put them in parallel the system can do an internal communication and become *R*. In other words, an internal communication of *R* is due to a reaction of *R*1 to a test provided by *R*2*,* where *R*1*, R*2 are some subprocesses of *R*. We can easily prove the following three lemmas.

As usual, let =⇒ be the transitive and reflexive closure of −*τ*→*, μ* be

=⇒

*μ μ*b *μ*

=⇒−→=⇒*,* and =⇒ be =⇒ if *μ* = *τ,* and =⇒ otherwise. *P* ⇓ *α* denotes

*P* =⇒↓ *α.*

With this labelled transition relation the notion of bisimulation can be given in the usual way.

Definition 4.1 A binary relation R⊆ *Prc*× *Prc* is a strong/weak simulation

*μ*

if *P* R *Q* implies that whenever *P* −→

*P* '*,* then there exists *Q*' such that

*Q μ* '

*μ*b ' ' '

−→ *Q /Q* =⇒ *Q* and *P* R *Q ,* where *fv*(*μ*)= ∅.

A relation R is a strong/weak bisimulation if both R and R−1 are strong/weak simulations. We say that *P* , *Q* are bisimilar, written *P* ∼ / ≈ *Q* if *P* R *Q* for some strong/weak bisimulation R.

Definition 4.2 [Bisimulation for open processes] For processes *P*1, *P*2 with

free variables *fv*(*P*1) ∪ *fv*(*P*2) ⊆ *X*˜, we set *P*1 ∼ / ≈ *P*2 if for all closed

˜ ˜ ˜ ˜ ˜

processes *Q*, it holds that *P*1{*Q/X*} ∼/≈ *P*2{*Q/X*}*.*

Lemma 4.3 *Suppose P* ≈ *Q, and P* ↓ *a*(*X*)*.S, then for any K, there exists*

*S*' *s.t. Q* ⇓ *a*(*X*)*.S*' *and S*{*K/X*} ≈ *S*'{*K/X*}*.*

The next lemma says that ≈ is preserved under non-object context.

Lemma 4.4 *Let P, Q, R, S be closed processes expressions, if P* ≈ *Q, then*

1. *νxP* ≈ *νxQ*
2. *α.P* + *R* ≈ *α.Q* + *R*
3. *P* |*R* ≈ *Q*|*R, R*|*P* ≈ *R*|*Q*
4. ⟨*k* ⇐ *P* ⟩ ≈ ⟨*k* ⇐ *Q*⟩

Remark 4.5 In lemma [4.4](#_bookmark6), proof of (3) desreves special care. Rather than proving that {*< P* |*R* ≈ *Q*|*R >* | *P* ≈ *Q, P, Q, R* ∈ *Prc*} is a bisimulation, here we should prove that {(*C*[*P* ]*, C*[*Q*]) | *P* ≈ *Q, P, Q* ∈ *Prc*} is a bisimulation, where *C* is a static context.

Now we have proved the weak bisimulation is preserved under all operators

except object constructor (i.e., *P* ≈ *Q* implies

—

*a* ⟨*P* ⟩ *.S* + *R* ≈

—

*a* ⟨*Q*⟩ *.S* +

*R*)*,* which is the most difficult due to the higher-order setting to which our language belongs*.* To derive this, we must prove congruence property of ≈ over higher-order substitution (i.e. *P* ≈ *Q* implies *P* {*R/X*} ≈ *Q*{*R/X*}). We can follow Sangiorgi’s approach to prove this by induction on the structure of *P* [[8](#_bookmark27)][[9](#_bookmark28)]. But in this paper, we first prove factorisation theorem independently, then show congruence property of ≈ over higher-order substitution is just a corollary of the former.

# Factorisation theorem and Congruence property of bisimulation w.r.t. Higher-order substitution

For convenience, we use ↑ *k* to denote a trigger program, which is a process of the form *k.*0*.* Intuitively, in the process *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐ *Q*⟩)*,* the process *Q* represents a “local resource” for *P* , and *k* is a “pointer” to the resource, and the only function of the trigger ↑ *k* is to activate a copy of *Q*, and each copy of *Q* is activated by a trigger ↑ *k* when needed. It is natural for us to explore whether *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐ *Q*⟩) and *P* {*Q/X*} are equivalent in the sense of weak bisimulation. The Factorisation theorem, which we are to prove, tells us it is so. In order to prove that *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐ *Q*⟩) is bisimilar with *P* {*Q/X*}*,* we need four lemmas, Lemma [5.2](#_bookmark9) and [5.3](#_bookmark10) analyze transitions which occurs in the process *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐ *Q*⟩)*,* and Lemma [5.2](#_bookmark9) analyzes those derived by triggering the private resource*,* Lemma [5.3](#_bookmark10) analyzes those which have nothing to do with triggering the resource; Lemma [5.4](#_bookmark11), [5.5](#_bookmark12) analyze non- communication-derived and communication-derived transitions which occur in *P* {*Q/X*}*.* The central technique we use is based on a formulation of the non-communication-derived transition or communication-derived transitions performed by these terms. By analysis on the fine-grained formulation of the these transitions, we can see how the transitions of theirs can be simulated by the terms’ counterparts. The detail of the formulation of a higher-order substituted process can be found in [[17](#_bookmark34)].

Σ

In our semantics, transitions fall into two groups: non-communication de- rived (or prefix) transitions; communication-derived transitions. For example,

—

*νb*(*a* ⟨0⟩ *.*0)

*a*(*X*)*.X*

−→ *νb*(0|0) is non-communication derived because it is due to a

prefix transition − ⟨0⟩ *.*0) *a*(*X*)*.X* 0|0; but the transition *νb*(− ⟨0⟩ *.*0|*a*(*X*)*.X*) −*τ*→

*a*

−→

*a*

*νb*(0|0) is communication derived because it is due to a communication tran-

sition − ⟨0⟩ *.*0|*a*(*X*)*.X* −*τ*→ 0|0*.*

*a*

Definition 5.1 Let *P*

−*α*→ *P* ' be a transition, if its deriving tree is not in-

volved in a H-Coml, H-Comr, F-Coml, F-Comr rule, then we call it a non- commucation-derived transition; otherwise, we call it a commucation-derived transition.

The following lemma analyze the form of the result process, which is de- rived by a copy of the resource has been triggered.

Lemma 5.2 *Let P* ∈ *P r, fv*(*P* ) ⊆ {*X*}*, Q* ∈ *Prc, and k does not occur in P, Q, if P* {↑ *k/X*}| *< k* ⇐ *Q >*−*τ*→ *P* '*, and the last rule applied to derive this transition is F-Comm , then there exists a context C such that P* = *C*[*X*]*, P* ' ≡ *C*[*Q*]{↑ *k/X*}| *< k* ⇐ *Q > .*

The following lemma says that if *P* {↑ *k/X*} perform an action *α* which has nothing to do with *k,* then *P* can perform *α.*

Lemma 5.3 *If P* {↑ *k/X*} −*α*→ *P* '*, fv*(*P* ) ⊆ {*X*}*,fv*(*α*)= ∅*, and k does not occur in P, Q, α, then there exists a program P*0 *such that P* ' = *P*0{↑ *k/X*}*,*

*and P* −*α*→ *P , moreover, it holds P* {*R/X*} −*α*→ *P* {*R/X*} *for any closed*

0

0

*process R.*

The following lemma analyzes how a non-communication-derived transi- tion *P* {*R/X*} −*α*→ *A* can be simulated by *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐ *R*⟩)*.*

Lemma 5.4 *Let P* ∈ *P r, fv*(*P* ) ⊆ {*X*}*, if P* {*R/X*} −*α*→ *P* '*, and this transi- tion is non-communication-derived, then*

* 1. *either P* {*R/X*} = *P* ' *for some P , and νk*(*P* {↑ *k/X*}| *< k* ⇐ *R >*) −*α*→ *νk*(*P*0{↑ *k/X*}| *< k* ⇐ *R >*);

0 0

* 1. *or P* = *C*[*X*] *with R* −*α*→ *R*'*, C*[*R*']{*R/X*} = *P* ' *for some C, R*'*, and*

*νk*(*P* {↑ *k/X*}| *< k* ⇐ *R >*) −*τ*→−*α*→ *νk*(*C*[0|*R*'| *< k* ⇐ *R >*]{↑ *k/X*})*.*

The following Lemma analyzes how communication-derived transition of

*P* {*R/X*} can be simulated by *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐ *R*⟩)*.*

Lemma 5.5 *Let P* ∈ *P r, fv*(*P* ) ⊆ {*X*}*, if P* {*R/X*} −*τ*→ *P* '*, and this transi- tion is communication-derived, then there exists a process P*0 *such that P* ' = *P* {*R/X*}*, and νk*(*P* {↑ *k/X*}| *< k* ⇐ *R >*) −*τ*→ ∪ −*τ*→−*τ*→ ∪ −*τ*→−*τ*→−*τ*→ ≡

0

*νk*(*P*0{↑ *k/X*}| *< k* ⇐ *R >*)*.*

Before we proceed in our proof the Factorization theorem, we first define a relation *TrigForm,*

Definition 5.6 A binary relation *TrigForm* ⊂ *Prc* × *Prc* is defined as fol- lowing:

*TrigForm* =

⎧⎨⟨*P, Q*⟩ *P* = *P*0{*R/X*}*,* and *Q* ≡ *νk*(*P*0{↑ *k/X*}| ⟨*k* ⇐ *R*⟩) ⎫⎬

*df*

⎩

⎭

for some *k, P ,R* with *fv*(*P* ) ⊆ {*X*} and *R* ∈ *Pr*

0

0

*c*

Lemma 5.7 *Let P, Q* ∈ *P r, TrigForm*(*P, Q*) *implies TrigForm*(*C*[*P* ]*, C*[*Q*])

*for any static context C.*

Combining Lemma [5.2](#_bookmark9), [5.3](#_bookmark10), [5.5](#_bookmark12), and [5.4](#_bookmark11), we can analyze how *P* {*R/X*}

and *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐ *R*⟩) can simulate each other.

Theorem 5.8 *Suppose P* ∈ *P r, R* ∈ *Prc, fv*(*P* ) ⊆ {*X*}*, and k does not occur free in P and R, then νk*(*P* {↑ *k/X*}| *< k* ⇐ *R >*) ≈ *P* {*R/X*}*. Furthermore, it also holds that:*

1. *whenever νk*(*P* {↑ *k/X*}| *< k* ⇐ *R >*)

⇒

−*α*→ *P* '*, there exists Q*' *s.t.*

*P* {*R/X*}

=*α*b

*Q*' *and TrigForm*(*Q*'*,P* ');

1. *and vice versa, whenever P* {*R/X*} −*α*→ *P* '*, there exists Q*' *s.t. νk*(*P* {↑

) =⇒

*k/X*}| *< k* ⇐ *R >*

*α*b *Q*' *and TrigForm*(*P* '*, Q*');

Proof. Let

঩ = {⟨*νk*(*P* {*R/X*}*,P* {↑ *k/X*}| ⟨*k* ⇐ *R*⟩)⟩ |if *k* does not occur free in *P, R* } *.* We show ঩ is a bisimulation up to ≈ *.* Note that *TrigForm*(*P* '*, Q*') implies *P* '঩*Q*'' ≡ *Q*' for some *Q*''*.*

1. We show *P* {*R/X*} can simulate any action performed by *νk*(*P* {↑ *k/X*} |

⟨*k* ⇐ *R*⟩)*.*

Suppose *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐ *R*⟩) −*α*→ *νkP* '*,* then *P* {↑ *k/X*}| ⟨*k* ⇐ *R*⟩ −*α*→ *P* '*,* then by case analysis on the last rule to derive *P* {↑ *k/X*}| ⟨*k* ⇐ *R*⟩ −*α*→ *P* '*.*

Case 1 The last rule is F-Coml, *α* = *τ,* then by Lemma [5.2](#_bookmark9), there exists a context *C* such that *P* = *C*[*X*]*, P* ' ≡ *C*[*R*]{↑ *k/X*}| ⟨*k* ⇐ *R*⟩ *,* by *P* = *C*[*X*]*,* then *P* {*R/X*} = *C*[*R*]{*R/X*}*.* So *P* {*R/X*} =⇒ *P* {*R/X*} = *C*[*R*]{*R/X*}*,*

and *TrigForm*(*C*[*R*]{*R/X*}*, νkP* ')*.*

Case 2 The last rule is Parl, then there exists a program *P* ' such that *P* {↑

*k/X*} −*α*→ *P* '*, k* does not occur in *P* , *Q, α,* so by Lemma [5.3](#_bookmark10)*,* there exists a

program *P* such that *P* −*α*→ *P , P* ' = *P* {↑ *k/X*}*, P* {*R/X*} −*α*→ *P* {*R/X*},

0 0 0 0

and *TrigForm*(*P*0{*R/X*}*, νk*(*P* '| ⟨*k* ⇐ *R*⟩))*.*

1. We show *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐ *R*⟩) can simulate any action performed by

*P* {*R/X*} *.*

Suppose *P* {*R/X*} −*α*→ *P* '*,* then

Case 1 This transition is communication-derived, by Lemma [5.5](#_bookmark12), there exists

*τ*

a process *P*0 such that *P* ' = *P* {*R/X*}*,* and *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐ *R*⟩)(−→

0

∪ −*τ*→−*τ*→ ∪ −*τ*→−*τ*→−*τ*→) = *Q*' ≡ *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐ *R*⟩) for some *Q*'*,*

0

and *TrigForm*(*P* '*, Q*');

Case 2 This transition is non-communication-derived, by Lemma [5.4](#_bookmark11), then there are 2 cases:

(2-1) either *P*0{*R/X*} = *P* ' for some *P*0*,* and *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐ *R*⟩) −*α*→

*νk*(*P*0{↑ *k/X*}| ⟨*k* ⇐ *R*⟩)*,* and *TrigForm*(*P*0{*R/X*}*, νk*(*P*0{↑ *k/X*}| ⟨*k* ⇐ *R*⟩));

(2-2) or *P* = *C*[*X*] with *R* −*α*→ *R*'*, C*[*R*']{*R/X*} = *P* ' for some *C, R*'*,* and

*νk*(*P* {↑ *k/X*}|*k* ⇐ *R*) −*τ*→−*α*→ *νk*(*C*[0]{↑ *k/X*}| ⟨*k* ⇐ *R*⟩ |*R*') ≡ *νk*(*C*[*R*']{↑ *k/X*}| ⟨*k* ⇐ *R*⟩)*.*

obviously, *TrigForm*(*C*[*R*']{*R/X*}*, νk*(*C*[0]{↑ *k/X*}| ⟨*k* ⇐ *R*⟩ |*R*'))*.*

Lemma 5.9 (abs-congruence) *Let P*1*, P*2 *, P* ∈ *P r, fv*{*P* } ⊆ {*X*}*, P*1 ≈

*P*2 *implies P* {*P*1*/X*} ≈ *P* {*P*2*/X*}*.*

Proof. Combine Lemma [4.4](#_bookmark6), and [5.8](#_bookmark13), we have

*P* {*P*1*/X*} ≈ (Lemma [5.8](#_bookmark13))

*νk* (*P* {↑ *k/X*}| ⟨*k* ⇒ *P*1⟩) ≈ (Lemma [4.4](#_bookmark6))

*νk* (*P* {↑ *k/X*}| ⟨*k* ⇒ *P*2⟩) ≈ (Lemma [5.8](#_bookmark13))

*P* {*P*2*/X*}

Theorem 5.10 (Congruence of ≈ under object operator) *Let P, Q, R,*

− −

*S be processes expressions, if P* ≈ *Q, then a* ⟨*P* ⟩ *.S* + *R* ≈*a* ⟨*Q*⟩ *.S* + *R*

Proof. Simply by Lemma [5.9](#_bookmark14).

# Barbed equivalence

Here we will present a notion of barbed equivalence in the style of [[13](#_bookmark32)], which is a variation of barbed bisimulation proposed in [[8](#_bookmark27)], which is also called ”reduction-closed” barbed bisimulation by Sangiorgi in [[10](#_bookmark29)]. The interested readers can see [[14](#_bookmark35)], [[10](#_bookmark29)] for a discussion regarding the two approaches. As

—

usual, *P* ↓*a*represents that *a*(*U* )*.Q*, *a* ⟨*K*⟩ *.Q* occurs unguarded in *P* with *a*

unrestricted, i.e., the occurrence is not a subexpression of *c*(*U* ')*.R*, *c* ⟨*K*'⟩ *.R,* and not within the scope of *νa.* Similarly, we can define ↓*m* where *m* is a first-order name*.* Besides, we define *P* ⇓ *a*(*U* )*.S* to denote *P* (−→)∗ ↓ *a*(*U* )*.S.*

Definition 6.1 A binary relation R ⊆ Pr × Pr is a barbed bisimulation if

*P*1R*P*2 implies

1. for any process *Q* ∈ Pr*, P*1|*Q* R *P*2|*Q.*
2. for each name *a*, *P*1 ⇓*x* if and only if *P*2 ⇓*x*;
3. whenever *P*1 −→ *P* '*,* then there exists *P* ' such that *P*2 (−→)∗ *P* ' and *P* '

1 2 2 1

R *P* '*.*

2

1. whenever *P*2 −→ *P* '*,* then there exists *P* ' such that *P*1 (−→)∗ *P* ' and *P* '

2 1 1 1

R *P* '*.*

2

Two processes *P* and *Q* are barbed equivalent, written *P* ≈*b Q* if *P* R*Q*

for some bisimulation R.

Now we show that ≈*b*and ≈ coincide for the higher-order calculi. We need the following lemmas which are easy to prove.

The following Lemma relates the correspondence between −→ and −*τ*→ *.*

Lemma 6.2 *P* −→ *P* ' *if and only if there exist P*0*, P* ' *such that P* ≡ *P*0*,*

0

*τ* ' ' '

*P*0 −→ *P*0*, P*0 ≡ *P .*

Lemma 6.3 *For any process P and name x, it holds that*

1. *if x is a higher-order name, it holds that P* ↓*xif and only if P*

−*α*→ *P* '

— '

*where α* = *x*(*U* )*.Q, x* ⟨*K*⟩ *.*0 *for some U, K, Q, P .*

1. *if x is a ﬁrst-order name, then P* ↓*xif and only if P*

−*α*→ *P* ' *where α* =

— '

1. *Q, x .*0 *for some Q, P .*

Lemma 6.4 *Let z be a name not occur free in P, a*(*U* )*.Q, m .*0 *then*

—

*a* ⟨*K*⟩ *.*0*, m.Q,*

*a*(*U* )*.Q*

'

* 1. *if P*
  2. '  '

=⇒ *P* /*P* =⇒ *P*

*m.z* ⟨0⟩ *.Q* =⇒≡ *P* ';

*then P* |*z* (*X*) *.*0|*a*(*U* )*.z* ⟨0⟩ *.Q* =⇒≡ *P* /*P* |*z* (*X*) *.*0|

—

*a*⟨*K*⟩*.*0 '

—

*m.*0 '

—  *τ* ' −

* 1. *if P* =⇒ *P* /*P* =⇒ *P then P* |*z* (*X*) *.*0| *a* ⟨*K*⟩ *.z* ⟨0⟩ *.*0 =⇒≡ *P* /*P* | *m*

*.z*(*U* )*.*0

=*τ*⇒≡ *P* ';

* 1. *if P* |*z* (*X*) *.*0|*a*(*U* )*.z* ⟨0⟩ *.Q* (−→)∗ *P* '' /*P* |*z* (*X*) *.*0|*m.z* ⟨0⟩ *.Q* (−→)∗ *P* ''*, and*

*P* '' / ⇓

*then there exists P* ' *such that P* '' ≡ *P* '*, and P a*(*U* )*.Q P* '/*P m.Q P* ';

*z*

—  ∗ ''

=⇒ =⇒

∗ ''

* 1. *if P* |*z* (*X*) *.*0| *a* ⟨*K*⟩ *.z* ⟨0⟩ *.*0 (−→) *P*

/*P* |*z* (*X*) *.*0|*m.z* ⟨0⟩ *.*0 (−→) *P*

*and*

*P* '' / ⇓*z*

*then there exists P* ' *such that P* '' ≡ *P* '*, and P*

—

*a*⟨*K*⟩*.*0

'

=⇒ *P* /*P*

—

*m.*0

=⇒

*P* ';

Theorem 6.5 *For two processes P and Q, P* ≈*b Q if and only if P* ≈ *Q.*

Proof. If: We show that ≈ is a barbed bisimulation. Suppose *P* ≈ *Q.* By Lemma [4.4](#_bookmark6), ≈ is preserved in parallel operator. If *P* −→ *P* '*,* then by Lemma

[6.2](#_bookmark16), there exists *P* ' such that *P* ≡ *P , P* −*τ*→ *P* '*, P* ' ≡ *P* '*.* Since ≡⊆ ≈*,* then

0 0 0 0 0

*P*0 ≈ *P* ≈ *Q,* so *Q* =⇒ *Q*' for some *Q*' with *Q*' ≈ *P* ' ≡ *P* '*.* Thus by Lemma

0

0

0

0

[6.2](#_bookmark16) again, we can find *Q*' such that *Q* (−→)∗ *Q*' and *P* ' ≈ *Q*'

0

≡ *Q*'*.* By Lemma

[6.3](#_bookmark17) and *P* ≈ *Q*, we have that *P* ⇓*x*if and only if *Q* ⇓*x .*

Only if: we prove that ≈*b* is a bisimulation up to ≈ *.* Consider *P* ≈*b Q.* Sup- pose that *P a*(*U* )*.R P* '*.* Choose some *z* ∈*/ f n*(*P, a*(*U* )*.R*)*,* by definition we have *P* |*z* (*X*) *.*0|*a*(*U* )*.z* ⟨0⟩ *.R* ≈*b Q*|*z* (*X*) *.*0|*a*(*U* )*.z* ⟨0⟩ *.R,* and by Lemma [6.4](#_bookmark18)(1), *P* |*z* (*X*) *.*0|*a*(*U* )*.z* ⟨0⟩ *.R* =⇒ *P*1 ≡ *P* ' for some *P*1 with *P*1 / ⇓*z,* then by Lemma [6.2](#_bookmark16), *P* |*z* (*X*) *.*0|*a*(*U* )*.z* ⟨0⟩ *.R* (−→)∗ *P* '*,* thus *Q*|*z* (*X*) *.*0|*a*(*U* )*.z* ⟨0⟩ *.R* (−→)∗ *Q*1

−→

for some *Q*1 with *Q*1 / ⇓*z* and *Q*1 ≈*b P*1*.* By Lemma [6.4](#_bookmark18)(3), there exists *Q*' such

—

that *Q a*(*U* )*.R Q*' ≡ *Q ,* then ' *b* '

*a*⟨*b*⟩*.*0 '

* 1. '

=⇒ 1

*P* ≡≈ ≡ *Q* . The case for *P*

−→ *P , P* −→ *P ,*

*P* −*m*→*.*0

*P* ' can be proved in the same way. The case for *P* −*τ*→ *P* ' is simpler.

# Conclusion and Related Works

In fact, the work in this paper is the continuation and extension of that in [[16](#_bookmark36)]. In [[16](#_bookmark36)], we have studied the contextual labelled semantics in the first-order *π*−calculus and a simple fragment of higher-order *π*-calculus separately. In fact, the syntax of that fragment of higher-order *π*−calculus is very restricted, and much simpler than that in this paper. In this paper, when we extend the semantic framework to full second-order calculus, the routine proof method for parallel operator fails, and it must be revised as Remark [4.5](#_bookmark7) said. Besides, factorisation theorem has been derived here, which is not dealt with in [[16](#_bookmark36)]. This is a great progress which has been made because factorisation theorem reveals the essence of higher-order calculus.

Sangiorgi did the most pioneering study on the similar topic on higher process calculi [[8](#_bookmark27)][[9](#_bookmark28)]. In [[9](#_bookmark28)] he proposed a traditional late labelled semantics and context bisimulation for higher process calculi. To prove the congruence properties of context bisimulation, he first proved it to be preserved under parallel and replication operator, then he used structural induction on the structure of *P* to prove congruence property w.r.t higher-order substitution. By using the congruence property and some smart techniques, he prove the distributivity of the so-called implication substitution, then obtain the result of the factorisation theorem. Our approach to achieve the central results is substantially different from his. The key difference lies in the techniques to prove congruence property w.r.t higher-order substitution and factorisation theorem. In his proof, the first question is primary and proved independently, and the proof of the second question heavily uses the result of the first. We emphasize on the fact that the proof of factorisation theorem is not easy even with the result of congruence [[9](#_bookmark28)]. But we think of factorisation theorem as a more fundamental result, and it can be proved independently, the congruence property of ≈ w.r.t higher-order substitution is a simple corollary of the fac- torisation theorem. Our main technique to prove the factorisation theorem is on the intuition that how *P* {*R/X*} and *νk*(*P* {↑ *k/X*}| ⟨*k* ⇐ *X*⟩) can simulate each other, which is more intuitive and easier to understand. Factorisation theorem is a key result in higher-order calculi, which reveals that higher-order substitution can be simulated by using trigger and private resource. Basing on it, congruence property can be derived easily. Furthermore, it plays a key role when we derive normal bisimulation– which is a simple characterisation of context bisimulation.

Jeffrey and Rathke adopted the same idea and gave a labelled semantics for a fragment of CML with local names [[12](#_bookmark31)], and proved that the bisimula- tion equivalence thus obtained is the same as a version of barbed equivalence. The proof of the first characterization theorem in our paper is similar to their

work. But due to the functional nature of CML, they adopted a triggered semantics to prove the congruence properties of the bisimulation equivalence in their work, roughly speaking, they first directly prove the congruence prop- erty on trigger semantics, and then they proved the correspondence between the original semantics and the trigger semantics, which yields the congruence property of the original, so their proof is much more difficult than ours. In [[18](#_bookmark37)], they directly introduced a trigger semantics as a canonical semantics for a variety of higher-order *π* − *calculus,* and proved the correspondence between bisimulation equivalence in the trigger semantics and barbed equivalence in reduction semantics. The most important result is that they prove the congru- ence properties of the bisimulation equivalence in this semantics for a language with recursive types. His proof technique is still a result like factorisation the- orem, rather than induction techniques on the types, which generally fails for a higher-order language with recursive types. It would be very interesting to understand whether our semantics and formulation for higher-order calculi can be applied to these calculi.

# References

1. R. Milner. *Calculus of Communicating Systems*. LNCS 92, Springer Verlag, 1980.
2. D. Park. *Concurrency and automata on inﬁnite sequences*. LNCS 104, 1981.
3. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
4. R. Milner, J. Parrow, and D. Walker. *A calculus of mobile processes*, (parts I and II). Information and Computation, 100, 1992.
5. B. Thomsen. *Calculus for Higher Order Communicating Systems*. PHD thesis, Department of Computing, Imperial College, 1990.
6. E. Astesiano and A. Giovini. *Generalized bisimulation in relational speciﬁcations*. LNCS 294, pages 207-266.
7. G. Boudol. *Towards a lambda calculus for concurrent and Communicating systems*. LNCS 351, pages 149-161.
8. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order* *Paradigms.* Ph.D. thesis, University of Edinburgh, Mayfield Road, Edinburgh, Scotland, 1992.
9. D. Sangiorgi. *Bisimulation for higher-order process calculi.* Information and Computation,131(2):141-178,1996.
10. D. Sangiorgi and Davide Walker. *The π* − *calculus: a* theory of mobile processes. Pages 42-43. Cambridge university press, 2001.
11. Peter Sewell. *From rewrite rules to bisimulation congruences.* In proceedings of Concur’98, LNCS 1466, pages 269-284, 1998.
12. Alan Jeffrey and Julian Rathke. *A theory of bisimulation for a fragment of concurrent CML with local names.* In proceedings on Logic in Computer Science, 2000.
13. Kohei Honda, and Nobuko Yoshida. *On reduction-based process semantics*. Theoretical Computer Science, 152(2):437-486, 1995.
14. C. Fournet and G. Gonthier. *A hierarchy of equivalence for asynchronous calculi*. LNCS 1443, 1998. In proceedings of International Colloquium on Algorithms, Language and Programming 1998.
15. R. Milner. *The polyadic π*−*Calculus: a tutorial*. The Proceedings of the International Summer School on Logic and Algebra of Specification, 1991.
16. Liu Xinxin, and Li yongjian. *Bisimulation for Higher-order π*−*Calculus*. The Proceedings of the third Asian workshop on Programming Language and systems (Aplas’03). 2002.
17. Li yongjian, Contextual labelled semantics for Higher-Order process calculi (full version), [http://lcs.ios.ac.cn/˜ly](http://lcs.ios.ac.cn/)j238/fgc04.ps.
18. Alan Jeffrey and Julian Rathke. Contextual equivalence for higher-order pi-calculus revisited. Proc. Mathematical Foundations of Programming Semantics. Elsevier. 2003.