ソフトウェア基礎論 26 − 6
（１９８８．１０．１４）

ＣＣＳにおける動的な通信名前づけについて

結縁祥治　　坂部俊樹　　稲垣康善

名古屋大学工学部

本稿では，並行システムの形式的記述の枠組みであるMilnerのＣＣＳを拡張し，通信リンクの動的な切り替えを直接記述できるようにする．MilnerのＣＣＳは非常に静的な枠組みであり，動作式を与える際に全ての通信の名前が陽に決定していなければならない．実際的なシステムをＣＣＳによって記述し，その形式的な枠組みで検証・変換等を行うという応用の観点からすると，この制約は厳しいものであると思われる．この制約を緩めるために，本稿における拡張では，通信の名前を変数を含む項で表現し，通信によって受け渡される値をこの変数に代入することで通信の名前を決定する．拡張された動作式の意味は，通信の名前を表す項のトップレベルの関数記号に，値領域と通信の意味領域とを対応づける関数を割り当てることによって行う．ここでは，通信の意味領域をMilnerのＣＣＳと同じ領域に取ることで，MilnerのＣＣＳにおける強合同関係に基づいて意味を定義する．最後に，通信の名前づけの粗さによって，拡張されたＣＣＳの意味を与える．

Dynamic Communication Naming in CCS

Shoji YUEN,　Toshiki SAKABE,　and Yasuyoshi INAGAKI

Faculty of Engineering, Nagoya University

Furo-cho, Chikusa-ku, Nagoya-shi, Aichi, 464 Japan

In this paper, Milner's CCS, a formalism for concurrent systems, is extended so that communication linkages between agents can be changed by the values passed by communications. Milner's CCS is a formalism with a static framework, where all the communications must be explicitly determined when a behaviour expression is given. When we try to apply CCS to describe a practical system, this restriction is very strict. In order to reduce this restriction we propose an extension of CCS, where a communication name is represented as a term allowed to have variables. By interpreting the term as a communication, the semantics of an expression in our extension is given. In this interpretation, a function from values to communications is assigned to each term at the top level symbol of the term representing a communication name. By taking the same communication domain as in Milner's CCS, the semantics of our extended CCS is given through the strong congruence of Milner's CCS. Finally, we give a direct semantics of our extended CCS with respect to the coarseness of the assignment of communication linkages to communication names.

1. Introduction

Milner's CCS[1] is a formalism for concurrent systems whose computing mechanism is modelled by communications. An agent synchronously communicates with some other agent. In CCS, an agent is represented as a term, called a *behaviour expression* which is built from the CCS operators and the terms over the value-domain. The semantics of the CCS operators are defined by the derivation relation, called the *action relation.*

An obvious application of CCS is to model and reason concurrent and/or distributed systems within the formal framework of CCS [1(Chapter 9)],[5],[6], [7],[8]. But in CCS the communication is restricted to be *static.* This restriction is very strict, when we try to apply CCS to reason practical systems. For this problem, in [5],[6], and [8], the framework of CCS is extended to allow communication linkages between agents to be changed dynamically to some extent, but the semantics is only informally outlined. They show that such an extension of the framework is needed in order to describe practical systems in a simple and natural way.

Here, we intend to formally give the semantics of extended CCS. The point of the extension is that a term possibly having variables is allowed as a communication name. The variables in the term are instantiated by importing values on communications. If all the variables are instantiated, the term is mapped to a communication. Thus, a communication can be changed by the value of the variables.

An extended behaviour expression is *interpreted* by assigning a *naming function* to each term occurring in it, according to the top level symbol of the term, where a naming function is a function from the value-domain to the communication-domain. A *name assignment* is a function from the top level symbols of those terms to naming functions. Given a set of extended behaviour expressions *interpreted* by *one* fixed name assignment, we define the action relation as in Milner's CCS, which is the basic semantics of our extended CCS. Under a name assignment we can define a *strong congruence* over the extended CCS, which is consistent with that of Milner's CCS. Thus, under one name assignment we can say that the semantics of our extended CCS is equivalent to the one given in Milner's CCS. We use Milner's strong congruence to relate the interpretations given by the different name assignments. Two extended behaviour expressions are said to behave equally if their interpretations behave equally in the sense of the strong congruence under the name assignments associated with them.

Another notion to be extended is the *restriction* operation. In CCS formalism, the module structure can be described by the restriction operation, which have the semantics of restricting communications within a certain part of a behaviour expression. In Milner's CCS, a communication is restricted directly by its name. But in our extended framework, a name of a communication is not completely known even after interpretation. Thus, here we propose to restrict by the top level symbol of a communication name. All the communications in the range of the naming function assigned to this top level symbol become restricted.

Further, we give a direct semantics of our extended CCS. It is shown that an extended behaviour expression is given its meaning up to relabelling if the *coarseness* of a name assignment is specified. As the direct semantics of an extended behaviour expression, We take the interpretation given by the *finest* name assignment as its semantics.

This paper is organised as follows. In Section 2, we have a brief overview of Milner's CCS. In Section 3, we define the extended CCS and relate its semantics with that of Milner's in the sense of the strong congruence. And in Section 4, the structure of name assignments is studied to introduce a direct semantics of our extended CCS.

2. An overview of Milner's CCS

In this section, we take a brief view over *Milner's CCS*[1]. Here the syntactic elements of CCS are listed and their informal semantics are outlined.

- Value

A *value* is a term conventionally built from function symbols standing for known total functions over values. □

- Value expression

A *value expression* is a term same as a value, but a value expression may contain variables to set values. For variables we shall use $x, y, \ldots$.

For convenience, we allow tuples $(E_1, \ldots, E_n)$ of value expressions to be treated as a value expression. An empty tuple (the case of $n = 0$) is denoted as $<>$. Usually, an empty value tuple is omitted when a communication is handled. □

These two elements above are used to describe objects to be passed when communications occur between agents.

- Name, co-name

A fixed set Δ of *names* is assumed. Names are ranged over by a lower case greek letter as $\alpha, \beta, \gamma, \ldots$. Another fixed set of *co-names* $\bar{\Delta}$ is associated with Δ by a bijection called the *complementary function*, $(\bar{\ })$:

$\alpha\ (\in \Delta) \mapsto \bar{\alpha}\ (\in \bar{\Delta})$

$\bar{\Delta}$ is disjoint from Δ. We say $\bar{\alpha}$ is the *co-name* of α. We also use $(\bar{\ })$ as the inverse of the complementary function. Thus we have $\bar{\bar{\alpha}} = \alpha$. □

A name represents a *port* on an agent to communicate with its environment. A port is specified to communicate with the port complementarily named. A name is also called *positive label* and a co-name *negative label* where the complementary function is treated as the *sign* of a name. Conventionally, a positive label is used for an input port and a negative label for an output port.

- Relabelling

A bijection S from a set of labels L to a set of labels M is a *relabelling* if it respects complements. (i.e. $\overline{S(\alpha)} = \bar{\alpha}$). □

- Behaviour expression

A behaviour expression B is recursively built from the following operators. ($B_1, B_2, \ldots$ are also used to denote behaviour expressions.)

Inaction:	NIL represents the agent which has no possibility to communicate with the environment.
Summation:	'+' is the operation of the nondeterministic choice. $B_1 + B_2$ is the agent which behaves like either B_1 or B_2.
Input action:	$\alpha x.B$ is the agent which behaves like B after importing a value to x by the communication with name α. The free occurrences of x in B is set to the imported values by the communication. Thus the variable x is bound to α, and they are not free in $\alpha x.B$ any more.
Output action:	$\bar{\alpha}v.B$ is the agent which behaves like B after exporting a value v by the communication with name α.
Internal action:	$\tau.B$ is the agent which behaves like B after an *internal communication*, denoted as τ, which is a special communication representing a communication between a pair of complementary labels. τ does not belong to $\Delta \cup \bar{\Delta}$.
Composition:	'|' is the operation of the parallel composition. $B_1 \mid B_2$ behaves as B_1 and B_2 are running concurrently communicating with each other. If a communication between a pair of the complementary labels in B_1 and B_2, an internal communication can occur.

Restriction:	$B\backslash A$, where A is a set of names, represents the agent in which the communications of B with names in A are restricted. Note that an internal action is never restricted because τ will never be in A. This operation gives the scope of a communication.
Relabelling:	$B/[S]$ is the application of a relabelling S to B.
Identifier:	$b(E_1, \ldots, E_{n(b)})$ is the procedure call by the symbol b, where $E_1, \ldots, E_{n(b)}$ are actual parameters to be passed. b's body is determined by an identifier declaration explained below.
Conditional:	**if** E **then** B_1 **else** B_2 is the conditional branch, where E has a boolean value. $\square$

• Identifier declaration

An *identifier declaration* has the form:
$b(x_1, \ldots, x_{n(b)}) \Leftarrow B_b$,
where $x_1, \ldots, x_{n(b)}$ are the formal parameters. The free occurrences of $x_1, \ldots, x_{n(b)}$ in B_b will be substituted when a procedure call occurs. $\square$

The formal semantics of CCS is given as the derivation relation between behaviour expressions, denoted as $B \xrightarrow{com} B'$, where *com* is either a pair of a name and a value or τ. This relation means that an agent B will be B' after the communication *com*. $\xrightarrow{com}$ is defined formally in [1(Chapter 5)].

We show an example of derivation to illustrate a behaviour of an agent:

$$
\begin{aligned}
&(p \mid q)\backslash\{\beta\} \qquad \text{where } p \Leftarrow \alpha x.\bar{\beta}x.p, \text{ and } q \Leftarrow \gamma.\beta y.\bar{\delta}y.q \\
\xrightarrow{(\alpha, v_1)}\ & (\bar{\beta}v_1.\bar{\delta}v_1.p \mid \gamma.\beta y.\bar{\delta}y.q)\backslash\{\beta\} \\
\xrightarrow{(\gamma, <>)}\ & (\bar{\beta}v_1.p \mid \beta y.\bar{\delta}y.q)\backslash\{\beta\} \\
\xrightarrow{\tau}\ & (p \mid \bar{\delta}v_1.q)\backslash\{\beta\} \\
\xrightarrow{(\bar{\delta}, v_1)}\ & (p \mid q)\backslash\{\beta\}
\end{aligned}
$$

The first derivation can be interchanged with the second because they can do communications concurrently. In the third derivation, only the internal action is possible because the communications with the names of β are restricted. And by this internal action, a value is passed to y. The complete derivation of this agent is infinite because it is defined recursively.

3. Extended CCS

In this section, we extend Milner's CCS to allow communication names to be changed dynamically by the values by communications. For this purpose, we put terms, possibly with variables, in place of names of CCS behaviour expressions. The values passed by communications are applied to a naming function, and the names are determined by evaluating the terms by the function assigned to them.

3.1 Syntax of Extended CCS

The extension made here is to introduce a term expression to be interpreted as communications. First we give a form of a term.

Definition (name expression)

A *name expression* is a term whose form is:

$f(E_1,\dots,E_m)$

where $E_1,\dots,E_m$ are value expression. We call f a *naming function symbol*, to which an arity $n(f)$ is preassigned. (Namely, $m = n(f)$ here.) We denote the set of name expressions H, ranged over by η. The variables occurring in η are written as $Var(\eta)$. And the set of name expressions without variables is denoted Ξ, ranged over by ξ.

For a name expression $\eta = f(E_1,\dots,E_m)$, we shall use the notation: $fname(\eta) = f$ and for a set of name expressions M:$fnames(M) = \{fname(m)\ ;\ m \in M\}$

Definition (extended behaviour expression)

An *extended behaviour expression* B_{ex} are given by the following BNF.

B_{ex}::=NIL	(inaction)
# $B_{ex} + B_{ex}$	(summation)
# $\eta\tilde{x}.B_{ex}$	(input action)
# $\bar{\eta}\tilde{E}.B_{ex}$	(output action)
# T.B_{ex}	(internal action)
# $B_{ex} \mid B_{ex}$	(composition)
# $B_{ex} \backslash F$	(restriction)
# $b(E_1,\dots,E_{n(b)})$	(identifier)

where T does not belong to the set of naming function symbols, and F is a set of naming function symbols.

In the extended behaviour expressions, a behaviour identifier is also defined in the same manner, but the body is an extended behaviour expression. An *identifier declaration* has the following form:

$b(x_1,\dots,x_{n(b)}) \Leftarrow B_{ex}$, where B_{ex} is an extended behaviour expression.

3.2 Interpretation of Extended behaviour expressions

The interpretation of an extended behaviour expression is given by a *name assignment*, by which the meaning of a name expression is determined. We call a function to be assigned to a name expression, a *naming function*.

Definition (naming function)

An *n-ary naming function* ψ is the total function over n-tuples of values to return names (an elements of Δ). Especially, a nullary naming function returns just a name.

i.e. $\psi\ :\ V^n \mapsto\ \Delta$

The arity of the function is expressed as $n(\psi)$.

Definition (name assignment)

A *name assignment* g is a total function from a naming function symbol to a naming function, preserving its arity.

$g\ :\ f\ \mapsto\ \psi$, where $n(f) = n(\psi)$

The application of a name assignment g to an extended behaviour expression B_{ex} is denoted as $[\![B_{ex}]\!]_g$ where all the naming function symbols appearing in B_{ex} are evaluated by g. We call $[\![B_{ex}]\!]_g$ the *interpretation* of B_{ex} by g.

3.3 Semantics by derivation of Extended CCS

Under one name assignment g, the action relation $\xrightarrow{com}_{ex}$ between extended behaviour expressions is given as the one given in Milner's CCS. Only the action axioms and the restriction rule are changed due to the extension of the syntax.

Here *label* and *value* are the projection functions that:

$$label(com) = \begin{cases} \alpha, & \text{if } com = (\alpha, v) \\ \tau, & \text{if } com = \tau \end{cases}, \quad value(com) = \begin{cases} v, & \text{if } com = (\alpha, v) \\ <>, & \text{if } com = \tau \end{cases}$$

(Action)

(a1) $[\![\xi x.B_{ex}]\!]_g \xrightarrow{com}_{ex} [\![B_{ex}\{value(com)/x\}]\!]_g \qquad (g(\xi) = label(com), value(com) = v)$

(a2) $[\![\bar{\xi} v.B_{ex}]\!]_g \xrightarrow{com}_{ex} [\![B_{ex}]\!]_g \qquad (g(\xi) = \overline{label(com)}, value(com) = v)$

(a3) $[\![\mathrm{T}.B_{ex}]\!]_g \xrightarrow{\tau}_{ex} [\![B_{ex}]\!]_g$

(Summation)

(s1) $$\frac{[\![B_{ex1}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex1}]\!]_g}{[\![B_{ex1} + B_{ex2}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex1}]\!]_g}$$ (s2) $$\frac{[\![B_{ex2}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex2}]\!]_g}{[\![B_{ex1} + B_{ex2}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex2}]\!]_g}$$

(Composition)

(c1) $$\frac{[\![B_{ex1}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex1}]\!]_g}{[\![B_{ex1}|B_{ex2}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex1}|B_{ex2}]\!]_g}$$ (c2) $$\frac{[\![B_{ex2}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex2}]\!]_g}{[\![B_{ex1}|B_{ex2}]\!]_g \xrightarrow{com}_{ex} [\![B_{ex1}|B'_{ex2}]\!]_g}$$

(c3) $$\frac{[\![B_{ex1}]\!]_g \xrightarrow{com_1}_{ex} [\![B'_{ex1}]\!]_g \ , \ [\![B_{ex2}]\!]_g \xrightarrow{com_2}_{ex} [\![B'_{ex2}]\!]_g}{[\![B_{ex1}|B_{ex2}]\!]_g \xrightarrow{\tau}_{ex} [\![B'_{ex1}|B'_{ex2}]\!]_g} \quad \begin{array}{l} \overline{label(com_1)} = label(com_2), \\ value(com_1) = value(com_2) \end{array}$$

(restriction)

$$\frac{[\![B_{ex}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex}]\!]_g}{[\![B_{ex}\backslash F]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex}\backslash F]\!]_g}, \quad label(com) \notin range(g(F)) \cup \overline{range(g(F))}$$

In the following argument, we handle only *closed* expressions, namely those expressions, B and B_{ex} with $FV(B) = \emptyset$ and $FV(B_{ex}) = \emptyset$. In CCS[1], closed expressions are called *programs*. A program in CCS is closed under $\xrightarrow{com}$. A program in the extended CCS is closed under $\xrightarrow{com}_{ex}$.

3.4 Strong congruence over Extended CCS

We give a congruence relation $\sim_{ex}$ over the extended behaviour expressions, when they are interpreted by one fixed name assignment. This congruence is given in the same way as the strong congruence $\sim$ over the original behaviour expressions. Here we use a bisimulation as a proof technique, following Milner[2].

Definition (strong bisimulation over $\mathcal{B}_{ex}{}^2$)

A binary relation $\mathcal{R} \subseteq \mathcal{B}_{ex}{}^2$ is a *strong bisimulation under* g if, whenever $B_{ex1}\mathcal{R}B_{ex2}$ and for any communication *com*,

(i) if $[\![B_{ex1}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex1}]\!]_g$, then $\exists B'_{ex2}$:$[\![B_{ex2}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex2}]\!]_g$ and $B'_{ex1}\mathcal{R}B'_{ex2}$

(ii) if $[\![B_{ex2}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex2}]\!]_g$, then $\exists B'_{ex2}$,$[\![B_{ex2}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex2}]\!]_g$ and $B'_{ex1}\mathcal{R}B'_{ex2}$

We are dealing with $\xrightarrow{com}_{ex}$, which is isomorphic to $\xrightarrow{com}$ under one fixed name assignment. Due to Milner[2], the following results are straightforward.

<u>**Proposition 3.1**</u>

For a name assignment g, there exists a *maximum bisimulation* $[\![\sim_{ex}]\!]_g$ under the set inclusion ordering. $[\![\sim_{ex}]\!]_g = \cup\{\mathcal{R}$; $\mathcal{R}$ is a bisimulation under g.$\}$

Proposition 3.2

$[\![\sim_{ex}]\!]_g$ is an equivalence relation for any g. $\square$

Proposition 3.3

For a given g, $[\![\sim_{ex}]\!]_g$ is a congruence for the operations of the extended CCS. $\square$

3.5 Relation between Extended CCS and Milner's CCS

We study how the extended CCS is related to the original one, with respect to the labelled derivation $\xrightarrow{com}_{ex}$. The extended CCS is different from the original CCS, when there are variables in the value expression tuples to be applied to a naming function. Although they are different relations, they are labelled by the communications. Thus, they can be related with each other in respect of their behaviours.

Another bisimulation between $\mathcal{B}_{ex}$ and $\mathcal{B}$ relates a CCS behaviour expression and an extended behaviour expression.

Definition (strong bisimulation over $\mathcal{B}_{ex} \times \mathcal{B}$)

A binary relation $\mathcal{R} \subseteq \mathcal{B}_{ex} \times \mathcal{B}$ is a *strong bisimulation under g*, if whenever $B_{ex}\mathcal{R}B$ and for any communication *com*,

(i) if $[\![B_{ex}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex}]\!]_g \exists B' : B \xrightarrow{com} B'$ and $B'_{ex}\mathcal{R}B'$.

(ii) if $B \xrightarrow{com} B'$, $\exists B'_{ex} : [\![B_{ex}]\!]_g \xrightarrow{com}_{ex} [\![B'_{ex}]\!]_g$ and $B'_{ex}\mathcal{R}B'$.

As for the extended behaviour expressions, there is a maximum bisimulation over $\mathcal{B}_{ex} \times \mathcal{B}$.

Proposition 3.4

There exists a maximum bisimulation:

$[\![\simeq]\!]_g = \cup\{\ \mathcal{R}\ ;\ \mathcal{R}$ is a strong bisimulation under $g\}$. $\square$

Proposition 3.5

$[\![\sim_{ex}]\!]_g \circ [\![\simeq]\!]_g \;=\; [\![\simeq]\!]_g \circ \sim \;=\; [\![\simeq]\!]_g$

(Proof)

It is clear from the definition that both $[\![\sim_{ex}]\!]_g \circ [\![\simeq]\!]_g$ and $[\![\simeq]\!]_g \circ \sim$ are strong bisimulations over $\mathcal{B}_{ex} \times \mathcal{B}$. Thus $[\![\sim_{ex}]\!]_g \circ [\![\simeq]\!]_g \subseteq [\![\simeq]\!]_g$ and $[\![\simeq]\!]_g \circ \sim \subseteq [\![\simeq]\!]_g$. Now suppose $< B_{ex}, B > \notin [\![\sim_{ex}]\!]_g \circ [\![\simeq]\!]_g$ and $< B_{ex}, B > \in [\![\simeq]\!]_g$. But this contradicts that the identity relation over ${\mathcal{B}_{ex}}^2$ is a strong bisimulation. And suppose $< B_{ex}, B > \notin [\![\simeq]\!]_g \circ \sim$ and $< B_{ex}, B > \in [\![\simeq]\!]_g$ Similarly, this contradicts that the identity relation over $\mathcal{B}^2$ is a strong bisimulation. $\square$

Based on this theorem, we can establish the semantics of the extended CCS for different name assignments. A pair of extended behaviour expressions B_{ex} and B'_{ex} behave equally under the name assignment g and g' respectively if there exist CCS behaviour expressions B and B' such that $B_{ex}[\![\simeq]\!]_g B$, $B'_{ex}[\![\simeq]\!]_{g'} B'$ and $B \sim B'$.

4. The structure of name assignments

In this section, we shall give a preorder over name assignments. It is shown that the coarseness of an interpretation by a name assignment g specifies an extended behaviour expression up to relabelling. Then, given a program (i.e. a closed extended behaviour expression) the finest name assignment can be derived. We will take it to be a direct semantics for our extended CCS.

4.1 Preorder over name assignments

Here we use these two notations.

By a name assignment g, a derived function $H(g)$ is defined:
$H(g)\ :\ \xi\ \mapsto\ \alpha$, where $\xi = f(v_1, \ldots, v_{n(f)})$ and $\alpha = g(f)(v_1, \ldots, v_{n(f)})$

and the *kernel* of a name assignment g, $Ker(g)$, is defined:
$Ker(g) = \Xi/(\mathcal{R}_g \circ \mathcal{R}_g{}^{-1})$ where $\mathcal{R}_g = \{< \xi, \alpha >\ ;\ H(g)(\xi) = \alpha\}$

Definition (preorder $\lhd$)

For a pair of name assignments g_1 and g_2, $g_1 \lhd g_2$ iff
$Ker(H(g_2))$ is a refinement of $Ker(H(g_1))$
i.e. $\forall X \in Ker(H(g_2)).\forall Y \in Ker(H(g_1)) : X \cap Y \neq \emptyset$ implies $X \subseteq Y$

Informally , $g_1 \lhd g_2$ means that the information of g_1 about communications is less than g_2.

Lemma 4.1

If $g_1\ \lhd\ g_2$, the following (a) and (b) hold.

(a) if $\llbracket B_{ex}\rrbracket_{g_1} \xrightarrow{com}_{ex} \llbracket B'_{ex}\rrbracket_{g_1}$, then

$\exists com' : \llbracket B_{ex}\rrbracket_{g_2} \xrightarrow{com'}_{ex} \llbracket B_{ex}\rrbracket_{g_2}$,
$label(com') \in \{g_2(\xi)\ ;\ \xi \in H(g_1)^{-1}(label(com))\}$, and $value(com) = value(com')$.

(b) if $\llbracket B_{ex}\rrbracket_{g_2} \xrightarrow{com}_{ex} \llbracket B''_{ex}\rrbracket_{g_2}$ then

$\exists com'' : \llbracket B_{ex}\rrbracket_{g_1} \xrightarrow{com''}_{ex} \llbracket B''_{ex}\rrbracket_{g_1}, \forall\xi \in H(g_2)^{-1}(label(com)) : H(g_1)(\xi) = label(com'')$,
and $value(com) = value(com'')$.

Here H is defined in the definition of $\lhd$.

(proof)

Induction on the number of applications of the inference rules. The relations between com and com' are derived from the induction base, which is the application of the action axioms. □

Theorem 4.2 (monotone)

$g_1\ \lhd\ g_2$ implies $\llbracket\sim_{ex}\rrbracket_{g_1} \supseteq \llbracket\sim_{ex}\rrbracket_{g_2}$

(proof)

What is to be shown is that $\llbracket\sim_{ex}\rrbracket_{g_2}$ is a strong bisimulation under g_1. Let $< B_{ex1}, B_{ex2} >$ be an arbitrary pair in $\llbracket\sim_{ex2}\rrbracket_{g_2}$. Suppose $\llbracket B_{ex1}\rrbracket_{g_1} \xrightarrow{com}_{ex} \llbracket B'_{ex1}\rrbracket_{g_1}$. Then $label(com) \in domain(g_1)$.

Thus, from Lemma 4.1(a),
$\exists com'\ :\ \llbracket B_{ex1}\rrbracket_{g_2} \xrightarrow{com'}_{ex} \llbracket B'_{ex1}\rrbracket_{g_2}$ and $label(com') \in \{g_2(\xi); \xi \in H(g_1)^{-1}(label(com))\}$,
and $value(com) = value(com')$.

From the definition of $\llbracket\sim_{ex}\rrbracket_{g_2}$, $\exists B'_{ex2}.\llbracket B_{ex2}\rrbracket_{g_2} \xrightarrow{com'}_{ex} \llbracket B'_{ex2}\rrbracket_{g_2}$ and $B'_{ex1}\llbracket\sim_{ex}\rrbracket_{g_2}B'_{ex2}$.

Then from Lemma 4.1(b),
$\exists com'' : \llbracket B_{ex2}\rrbracket_{g_1} \xrightarrow{com''}_{ex} \llbracket B'_{ex2}\rrbracket_{g_1}, \forall\xi \in H(g_2)^{-1}(label(com')) : H(g_1)(\xi) = label(com'')$,
and $value(com') = value(com'')$. Since com' is an image by g_2 of the terms whose image by g_1 is com, $label(com) = label(com'')$. And $value(com) = value(com') = value(com'')$. So $com = com''$.

The case that $\llbracket B_{ex1}\rrbracket_{g_1} \xrightarrow{com}_{ex} \llbracket B'_{ex1}\rrbracket_{g_1}$ is proved symmetrically.

Thus, $\llbracket\sim_{ex}\rrbracket_{g_2}$ is a strong bisimulation under g_1. Then $\llbracket\sim_{ex}\rrbracket_{g_2} \subseteq \llbracket\sim_{ex}\rrbracket_{g_1}$ from the definition of $\llbracket\sim_{ex}\rrbracket_{g_1}$. □

Based on this theorem, we can establish a class of extended behaviour expressions, which, informally speaking, behave in the same pattern along with coarseness of a name assignment.

Definition (name assignment variant $\bowtie$)

$g_1 \bowtie g_2$ iff $g_1 \lhd g_2$ and $g_2 \lhd g_1$

Corollary 4.3

If $g_1 \bowtie g_2$, then $[\![\sim_{ex}]\!]_{g_1} = [\![\sim_{ex}]\!]_{g_2}$ $\square$

With Proposition 3.5, it can be seen that based on the original CCS the semantics of the extended CCS is given enough if it is interpreted by g within $\bowtie$.

Two name assignments under the relation $\bowtie$ make the same behaviour within a relabelling.

Proposition 4.4

If $g_1 \bowtie g_2$, for any (original) CCS behaviour expressions B and B' such that $B_{ex}[\![\simeq]\!]_{g_1}B$ and $B_{ex}[\![\simeq]\!]_{g_2}B'$, there exists a relabelling S and $B' = B[S]$ $\square$

4.2 The finest name assignment of the extended CCS

In this subsection, given an extended behaviour expression we take its interpretation by the finest name assignment as its meaning. Note that we only deal with a fixed set of values in the following argument.

The restriction of name assignment g by F is denoted as $g\,|_{res}\,F$. Namely, the domain of g is restricted to F. We also use the same notation for a set of name assignments:

$G\,|_{res}\,F = \{\, g' \;;\; g' = g\,|_{res}\,F,\; g \in G\}$

Here $G\,|_{res}$ writes G_F for short.

The quotient set of G_F by the equivalence $\bowtie$ is denoted as $G_F/\bowtie$. A partial order $\preceq$ over $G_F/\bowtie$ is introduced by $\lhd$ as follows:

For $G_1 \in G_F/\bowtie$, $G_2 \in G_F/\bowtie$, $G_1 \preceq G_2$ iff $\forall g_1 \in G_1\ \forall g_2 \in G_2 : g_1 \lhd g_2$

Proposition 4.5

G_F has the maximum element $max(G_F)$ under the order of $\preceq$. A name assignment in $max(G_F)$ has the following properties.

$$\forall g_{max} \in max(G_F) \;:\; \begin{cases} domain(g_{max}) = F \\ \forall \psi \in range(g_{max}).\psi \text{ is an injection.} \\ \forall f_1, f_2 \in F.f_1 \neq f_2 \text{ implies } range(g_{max}(f_1)) \cap range(g_{max}(f_2)) = \emptyset \end{cases}$$

$\square$

Theorem 4.6

$\forall g_{max} \in max(G_F) \;:\; \bigcap_{g \in G_F} [\![\sim_{ex}]\!]_g = [\![\sim_{ex}]\!]_{g_{max}}$

(proof)

For any name assignment $g \in G_F$, $g \lhd g_{max}$. Then from Theorem 4.2, $[\![\sim_{ex}]\!]_g \supseteq [\![\sim_{ex}]\!]_{g_{max}}$. Thus $\bigcap_{g \in G_F} [\![\sim_{ex}]\!]_g \supseteq [\![\sim_{ex}]\!]_{g_{max}}$ Since $g_{max} \in G_F$, then $[\![\sim_{ex}]\!]_{g_{max}} \supseteq \bigcap_{g \in G_F} [\![\sim_{ex}]\!]_g$ $\square$

Together with Corollary 4.3 and Theorem 4.6, it is seen that the unique name assignment g_{max} up to $[\![\sim_{ex}]\!]_{g_{max}}$ is given for a name assignment class G_F. Thus, we write $[\![\sim_{ex}]\!]_F$ instead of $[\![\sim_{ex}]\!]_{g_{max}}$ for some $g_{max} \in max(G_F)$.

The finest name assignment makes a standard interpretation for an extended behaviour expression. For example, a.NIL and b.NIL are equal because there are the finest name assignments $f_{1\{a\}} = [a \rightarrow \alpha]$ and $g_{2\{b\}} = [b \rightarrow \alpha]$ respectively, and $[\![a.\mathrm{NIL}]\!]_{g_1} \sim [\![b.\mathrm{NIL}]\!]_{g_2}$.

Proposition 4.7

$\forall F' \subseteq F \;:\; max(G_{F'}) = max(G_F)\,|_{res}\,F$ $\square$

Proposition 4.8

Given two extended behaviour expressions B_{ex} and B'_{ex},
if $fnames(B_{ex}) \subseteq F_1$ and $fnames(B'_{ex}) \subseteq F_1$, and $B_{ex}[\sim_{ex}]_{F_1} B'_{ex}$, then $\forall F_2 \supseteq F_1 . B_{ex}[\sim_{ex}]_{F_2} B'_{ex}$
where $fnames(B_{ex})$ is the set of the naming function symbols appearing in B_{ex} (generally not equal to $L(B_{ex})$). $\square$

The two propositions above can be used for a program to be embedded in a larger context by the finest name assignment of the context. However, the renaming operation of naming function symbols is necessary to avoid conflict of naming function symbols with its context. This renaming is always possible because g_{max} just specifies whether two name expressions mean equal or not. Thus, an appropriate name assignment can be taken out of $max(G_F)$ and rename its domain not to conflict with the context in which the program is embedded.

5. Concluding remarks

We have proposed an extension of Milner's CCS, in which the values of the communication names can be dynamically determined. And we have investigated the relation between the original CCS framework and ours. This extension is motivated by modelling and reasoning practical concurrent systems by CCS. In such cases, the descriptions become simpler and more natural by introducing the means to identify communications by the value passed by the communications. In the syntax of the extended CCS, a term with variables is allowed as a communication name. The semantics is given by interpreting a communication name as a communication. This interpreting procedure is called a *name assignment* which assigns a naming function to each term at the top level symbol. By a naming function the meanings of a term is determined. It is shown that if the coarseness of name assignment is given, the meanings of an extended behaviour expressions is specified up to relabelling. Based on this fact, we proposed a direct semantics of the extended CCS, namely the interpretation given by the *finest* name assignment.

Acknowledgement

The authors wish to express their gratitude to Dr. Namio Honda, President of Toyohashi University of Technology and Dr. Teruo Fukumura, Professor of Chukyo University for their encouragements to conduct this work. They also thank Assistant Professor Tomio Hirata, Dr. Tohru Naoi of Nagoya University and Mr. Olav Aanderaa of Nagoya University for helpful discussions.

References:

[1] R.Milner:"A Calculus of Communicating Systems", LNCS 92
[2] R.Milner:"Calculi for Synchrony and Asynchrony", Theoretical Computer Science 25 (1983) pp.267-310 (North-Holland)
[3] C.A.R.Hoare:"Communicating Sequential Processes", Comm. of ACM 21 (8) (1978) pp.666-677
[4] C.A.R.Hoare:"Communicating Sequential Processes", Prentice-Hall (1985)
[5] S.A.Smolka and R.E.Strom:"A CCS Semantics for NIL", IBM Journal of Research and Development 31 (1987) pp.556-570
[6] T.W.Doeppner Jr. and A.Glacalone:"A Formal Description of the UNIX Operating System", Proc. of the 2nd Annual ACM Sym. on Principles of Distributed Computing (1983) pp.241-253
[7] M.C.B.Hennessy and W.Li:"Translating a Subset of Ada into CCS", Formal Description of Programming Concepts II (Bjørner ed.) (1983) pp.227-249
[8] S.Yuen, T.Sakabe and Y.Inagaki:"A Formal Descriptin of Monitors by CCS", IEICE Tech. Rep. COMP87-84 (1988) (In Japanese)
[9] N.Francez:"Extended Naming Conventions For Communicating Processes", Science of Computer Programming 3 (1983) pp.101-114
[10] I.Castellani:"Bisimulations and Abstraction Homomorphisms", LNCS 185 (1985) pp.223-238