データベースシステム 99−20
（1994. 7. 21）

アクティブデータベースの動作解析のためのプロセス代数の開発

磯部 祥尚, 小島 功, 大蒔 和仁

電子技術総合研究所 情報アーキテクチャ部 情報ベース研究室

本研究の目的は、アクティブデータベースのプロダクションルールを解析し、ルールの設計支援システムを開発することにある。各プロダクションルールは、イベント、コンディション、アクションを含む仕様であり、これらは、イベントが起こり、コンディションが満たされたときに実行されるアクションを意味している。アクティブデータベースはプロダクションルールをもつデータベースであり、状況に応じて自動的にデータベースの状態を更新したり、状態の無矛盾検査などをすることができる。また、プロダクションルールは知識ベースのための有効な道具ともなりうる。しかし、プロダクションルールは並行性や協調性により、その動作を予測することは一般に困難である。

並行システムを記述し解析するために、CCS や π 計算のようなプロセス代数が知られている。我々はプロダクションルールを解析する道具としてプロセス代数を採用している。しかしながら、従来のプロセス代数をプロダクションルールの解析に適用したところ、いくつかの問題点を指摘できる。

我々は、CCS を拡張した **CCSPR** (a Calculus of Communicating Systems with Production Rules) を提案する。CCSPR は動的に変化するプロセス木を容易に記述することができるため、プロダクションルールの解析に適している。本稿では、先ず CCS をもとにプロセス代数によるプロダクションルールの解析方法を説明し、その問題点を指摘する。その後 CCSPR の特長について述べ、その有効性を示す。

A Study of Process Algebras for Active Databases

Yoshinao ISOBE, Isao KOJIMA, Kazuhito OHMAKI

Information Base Section, Computer Science Division, Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, Ibaraki 305, Japan

The purpose of this research is to analyze *production rules* in active databases and to exploit an assistant system for rule programming. Each production rule is a specification including an *event*, a *condition*, and an *action*. The action is automatically executed whenever the event occurs and the condition is satisfied. An *active database*, which is a database with production rules, can spontaneously update database states and check their consistency. Production rules provide a powerful mechanism for knowledge-bases. However, it is very difficult in general to predict how production rules will behave due to cascading rule triggers, concurrency, and so on.

In order to describe and analyze concurrent and communicating systems, *process algebras* such as CCS, CSP, ACP, and π-calculus, are well known. We are attempting to adopt a *process algebra* as a basic tool to analyze production rules. However, there are difficulties to apply existing process algebras to analysis of production rules, for example, multi-way local communications between a parent-process and created child-processes.

In this paper, we propose a process algebra named *CCSPR* (a Calculus of Communicating Systems with Production Rules), which is an extension of CCS. The advantage of CCSPR is to easily describe variable process trees. Therefore, production rules can be appropriately analyzed in CCSPR. First, we explain how to analyze production rules by CCS and point out problems of the analysis. Second, we introduce CCSPR and show its advantages for analysis of production rules.

isobe@etl.go.jp, kojima@etl.go.jp, ohmaki@etl.go.jp

1 Introduction

Active databases[1][4] can spontaneously react to specific situations, by means of *production rules*. Production rules specify relations between situations and (re)actions. Each rule is a specification including an *event*, a *condition*, and an *action*. When an event occurs, all rules including the event are *triggered* simultaneously, and they can be transacted concurrently. In each transaction, the action in the triggered rule is executed if the condition in the rule is satisfied. Furthermore, the action execution can trigger other rules which are transacted as its *child-transactions* (also called nested transactions). Thus, cascading rule triggers produce a *transaction tree*.

Relations between a parent-transaction and a child-transaction are specified by *coupling modes*. There are three possible coupling modes[1][4]:

1. **immediate** : A child-transaction is immediately executed after triggered, and its parent-transaction has to wait to commit until the child-transaction committing.
2. **separate** : A child-transaction is immediately executed after triggered, but its parent-transaction has *not* to wait to commit until the child-transaction committing.
3. **deferred** : A child-transaction execution is delayed just prior to the top-level-transaction committing of the transaction tree including the child-transaction.

In most cases, the first coupling mode **immediate** may be used. In some cases, such as display of messages without respect to parent-transactions committing, the second coupling mode **separate** may be used. In other cases, such as consistency constraints, the third coupling mode **deferred** may be used.

Users of an active database system want to freely design production rules, so that the system automatically react to specific situations. However, it is very difficult in general to design rules[2][3][5], because of cascading rule triggers, coupling modes, concurrency, communications, and so on. Hence, it is important to analyze behaviors of rules before they are executed and to aid rule programming.

We are attempting to adopt a *process algebra* as a basic tool to analyze production rules. In order to describe and analyze concurrent and communicating systems, *process algebras* such as CCS[6], CSP[7], ACP[8], and π-calculus[9], are well known. We tried to apply existing process algebras to analysis of production rules. We have at least three requirements in order to analyze production rules as follows:

1. A process can call out saved processes and to create executable processes for them.
2. Relations between parent-processes and child-processes must be uniquely determined in order to construct a process tree[1].
3. Multi-way local communications between a parent-process and child-processes are needed in order to implement coupling modes.

It may be possible to describe the above properties by existing process algebras. However, there are difficulties to describe by them. In order to construct a process tree, for example, a notion of *process-id* is available, but it explosively increase cost of calculus for analysis by process algebras.

In this paper, we propose a process algebra named *CCSPR*. CCSPR is 'a Calculus of Communicating Systems with Production Rules', and it is an extension of CCS. An important feature of CCSPR is to explicitly indicate relations between parent-processes and child-processes by *Subordinate compositions* ⟩ instead of *process-id's*.

In Section 2, we show an example of a scheduler with production rules and explain how to analyze it by process algebras. Then, problems of existing process algebras for production rules are pointed out. In Section 3, combinators of CCSPR are informally introduced, and it is explained how to create child-processes. In Section 4, definitions of CCSPR are given. In Section 5, the example given in Section 2 is analyzed using CCSPR.

2 An analysis of production rules by process algebras

In this section, we give an example of production rules and analyze the scheduler using CCS. Then, problems of existing process algebras for production rules are pointed out.

2.1 An example of production rules

In this Subsection, an example of a scheduler with seven production rules is given. For simplify, we assume that a condition in each rule is always true and can be omitted. Therefore, each rule includes an event, an action, and a coupling mode. The action consists of scheduled events $a1, \cdots, a7$ and triggering events $e1, \cdots, e7$. Assume that triggering events are hidden from its environment. In other words, rules can be triggered by only a rule managers and actions in rules. The production rules are specified in Figure 1.

[1] It corresponds to a transaction tree in active databases.

Name	Event	Action	Coupling Mode
Rule 1	$e1$	$\overline{a1};\overline{e3}$	**immediate**
Rule 2	$e1$	$\overline{a2};\overline{e4}$	**immediate**
Rule 3	$e3$	$\overline{a3};\overline{e6}$	**deferred**
Rule 4	$e4$	$\overline{a4};\overline{e7}$	**separate**
Rule 5	$e4$	$\overline{a5}$	**immediate**
Rule 6	$e6$	$\overline{a6}$	**immediate**
Rule 7	$e7$	$\overline{a7}$	**deferred**

Figure 1: The specification of each production rule

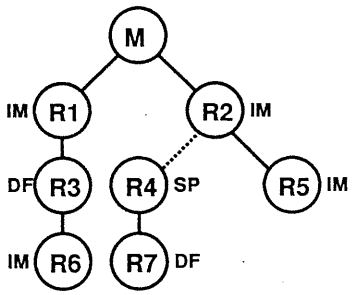

Figure 2: The process tree of this example

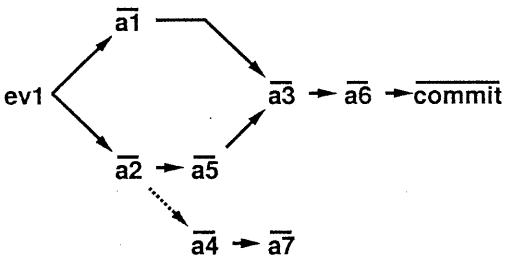

Figure 3: The expected order of scheduled events

For simplify, the rule manager M can perform only the event $\overline{e1}$ when an event $ev1$ is signaled from its environment, and it becomes an inaction process after committed. When the event $\overline{e1}$ occurs, **Rule 1** and **Rule 2** are triggered. And cascading rule triggers produce a process tree as shown in Figure 2. The order of scheduled events is expected as shown in Figure3, by means of coupling modes. For example, the action in **Rule 3** is delayed just prior to **M** committing. The commitment of **M** is delayed until **Rule 2** committing. The commitment of **Rule 2** is delayed until only **Rule 5** committing, because **Rule 4** has a coupling mode **separate**. Therefore, the action in **Rule 3** is executed after the actions in **Rule 1**, **Rule 2**, and **Rule 5**. On the other hand, the action in **Rule 7** waits for only **Rule 4**, because **Rule 4** and **Rule 7** are separated from other rules.

2.2 An application of CCS

It is important to check whether the scheduler with the seven production rules behave as shown in Figure 3, or not. We tried to apply existing process algebras to this checking. We explain how to describe the scheduler in CCS (a Calculus of Communicating Systems) which is a fundamental process algebra proposed by R.Milner[6].

Each rule is described as follows[2]:

$$Rule(X,Y) \stackrel{\text{def}}{=} (s.X[d/done]|Y)\backslash\{s,d\}$$

$$R1 \stackrel{\text{def}}{=} e1_1(i,j).(Rule(A1(i), IM1(i,j))|R1)$$
$$R2 \stackrel{\text{def}}{=} e1_2(i,j).(Rule(A2(i), IM2(i,j))|R2)$$
$$R3 \stackrel{\text{def}}{=} e3(i,j).(Rule(A3(i), DF1(i,j))|R3)$$
$$R4 \stackrel{\text{def}}{=} e4_1(i,j).(Rule(A4(i), SP1(i,j))|R4)$$
$$R5 \stackrel{\text{def}}{=} e4_2(i,j).(Rule(A5(i), IM0(i,j))|R5)$$
$$R6 \stackrel{\text{def}}{=} e6(i,j).(Rule(A6(i), IM0(i,j))|R6)$$
$$R7 \stackrel{\text{def}}{=} e7(i,j).(Rule(A7(i), DF0(i,j))|R7)$$

$$R \stackrel{\text{def}}{=} (R1|R2|R3|R4|R5|R6|R7)$$

$Rule(X,Y)$ is an operator upon processes. A process substituted into X is controlled a process substituted into Y. In this case, a process for executing an action is substituted into X, and a process for a coupling mode is substituted into Y. For example, $R1$ which is a process for **Rule 1** creates a process $A1(i)$ for its action and a process $IM(i,j)$ for its coupling mode, when an event $e1_1(i,j)$ occurs. The parameters i and j are a self-process-id and a parent-process-id, respectively. The process-id's are received from its parent-process thought the event $e1_1(i,j)$, when its parent-process calls out.

Each action is described as follows:[3]

$$A1(i) \stackrel{\text{def}}{=} \overline{a1}.getid(k).\overline{e3(k,i)}.\overline{done}.0$$
$$A2(i) \stackrel{\text{def}}{=} \overline{a2}.getid(k).\overline{e4_1(k,i)}.getid(k).\overline{e4_2(k,i)}.\overline{done}.0$$
$$A3(i) \stackrel{\text{def}}{=} \overline{a3}.getid(k).\overline{e6(k,i)}.\overline{done}.0$$
$$A4(i) \stackrel{\text{def}}{=} \overline{a4}.getid(k).\overline{e7(k,i)}.\overline{done}.0$$
$$A5(i) \stackrel{\text{def}}{=} \overline{a5}.\overline{done}.0$$
$$A6(i) \stackrel{\text{def}}{=} \overline{a6}.\overline{done}.0$$
$$A7(i) \stackrel{\text{def}}{=} \overline{a7}.\overline{done}.0$$

[2] $\omega.P$ denotes a process which performs an event ω thereafter becomes a process P. $|$ denotes a concurrency composition. $\backslash$ denotes a restriction of a communication. $[a/b]$ denotes a relabelling of a event from b to a.

[3] In CCS, overlined events such as $\overline{a}$ is used for output events, and unoverlined events such as a is used for input events. Complementary events such as a and $\overline{a}$ can spontaneously occur and they can communicate.

$getid(k)$ is an event for binding a new process-id into k from a process $ID(i)$ defined as follows:

$$ID(i) \stackrel{\text{def}}{=} \overline{getid(i)}.ID(i+1)$$

Processes for coupling modes depend on the number of child-processes called out by action executions.

$$IM0(i,j) \stackrel{\text{def}}{=} \overline{s}.d.\overline{c_j}.sd_j.\overline{cd_j}.\mathbf{0}$$
$$IM1(i,j) \stackrel{\text{def}}{=} \overline{s}.d.c_i.\overline{c_j}.sd_j.\overline{sd_i}.cd_i.\overline{cd_j}.\mathbf{0}$$
$$IM2(i,j) \stackrel{\text{def}}{=} \overline{s}.d.c_i.c_i.\overline{c_j}.sd_j.\overline{sd_i}.\overline{sd_i}.cd_i.cd_i.\overline{cd_j}.\mathbf{0}$$

$$SP0(i,j) \stackrel{\text{def}}{=} ((\overline{s}.d.\mathbf{0})|(\overline{c_j}.sd_j.\overline{cd_j}.\mathbf{0}))$$
$$SP1(i,j) \stackrel{\text{def}}{=} ((\overline{s}.d.c_i.\overline{sd_i}.cd_i.\mathbf{0})|(\overline{c_j}.sd_j.\overline{cd_j}.\mathbf{0}))$$
$$SP2(i,j) \stackrel{\text{def}}{=} ((\overline{s}.d.c_i.c_i.\overline{sd_i}.\overline{sd_i}.cd_i.cd_i.\mathbf{0})|(\overline{c_j}.sd_j.\overline{cd_j}.\mathbf{0}))$$

$$DF0(i,j) \stackrel{\text{def}}{=} \overline{c_j}.sd_j.\overline{s}.d.\overline{cd_j}.\mathbf{0}$$
$$DF1(i,j) \stackrel{\text{def}}{=} \overline{c_j}.sd_j.\overline{s}.d.c_i.\overline{sd_i}.cd_i.\overline{cd_j}.\mathbf{0}$$
$$DF2(i,j) \stackrel{\text{def}}{=} \overline{c_j}.sd_j.\overline{s}.d.c_i.c_i.\overline{sd_i}.\overline{sd_i}.cd_i.cd_i.\overline{cd_j}.\mathbf{0}$$

$IM0$, $IM1$, and $IM2$ are processes to implement a coupling mode **immediate** to control actions which call out *zero*, *one*, and *two* child-process(es), respectively. SP and DF are processes for coupling modes **separate** and **deferred**, respectively. s and d are events to control a start point and an end point of an action execution. c_i is an event to transmit to a parent-process that its child-process with a coupling mode **immediate** is committed. sd_i is an event to start a deferred child-process from the parent-process. cd_i is an event to transmit to a parent-process that its child-process with a coupling mode **deferred** is committed. Finally, the system SYS of the scheduler is built as follows:

$$SYS \stackrel{\text{def}}{=} (R|M|ID(1))\backslash L$$

$$M \stackrel{\text{def}}{=} ev1.getid(k).\overline{e1_1(k,0)}.getid(k).\overline{e1_2(k,0)} .c_0.c_0.\overline{sd_0}.\overline{sd_0}.cd_0.cd_0.\mathbf{0}$$

$$L = \{s,d\} \cup \{e1_1, e1_2, e3, e4_1, e4_2, e6, e7\} \cup \{c_i, sd_i, cd_i : i \in \{0,1,2,3,\cdots\}\}$$

M is a process for the rule manager. L is a set for hiding events in it from environment.

Next, we describe expected oreder $ORDER$ in shown Figure 3. It is described in CCS as follows :

$$X;Y \stackrel{\text{def}}{=} (X[d/done]|d.Y)\backslash d$$
$$X \| Y \stackrel{\text{def}}{=} (X[d_1/done]|Y[d_2/done]|d_1.d_2.\overline{done})\backslash d_1, d_2$$
$$\langle\langle X \rangle\rangle \stackrel{\text{def}}{=} (X[d/done]|d.\mathbf{0}|\overline{done}.\mathbf{0})\backslash d$$

$$EA1 \stackrel{\text{def}}{=} \overline{a1}.\overline{done}.\mathbf{0} \qquad EA2 \stackrel{\text{def}}{=} \overline{a2}.\overline{done}.\mathbf{0}$$
$$EA3 \stackrel{\text{def}}{=} \overline{a3}.\overline{done}.\mathbf{0} \qquad EA4 \stackrel{\text{def}}{=} \overline{a4}.\overline{done}.\mathbf{0}$$
$$EA5 \stackrel{\text{def}}{=} \overline{a5}.\overline{done}.\mathbf{0} \qquad EA6 \stackrel{\text{def}}{=} \overline{a6}.\overline{done}.\mathbf{0}$$
$$EA7 \stackrel{\text{def}}{=} \overline{a7}.\overline{done}.\mathbf{0} \qquad COM \stackrel{\text{def}}{=} \overline{com}.\overline{done}.\mathbf{0}$$

$$ORDER \stackrel{\text{def}}{=} ev1.(EA1 \| (EA2; (EA5 \| \langle\langle EA4; EA7 \rangle\rangle)) ; EA3; EA6; COM)$$

where, ; and $\|$ are defined as a Sequential composition and a Parallel composition, respectively. $\langle\langle P \rangle\rangle$ means separation of the process P. $EA1, \cdots, EA7$ contain only scheduled events unlike $A1, \cdots, A7$, because triggering events can not be observed.

We can check whether the scheduling system SYS behave as the expected oreder $ORDER$ shown in Figure 3 or not, using algebraic laws in CCS. We proved $SYS = ORDER$[4] using CWB[11] (Concurrency workbench).

2.3 Problems to be resolved

As shown in Section 2.2, it is possible to analyze production rules in CCS. However, there are problems as follows:

1. Parameters such as process-id's explosively increase cost of calculus for analysis.
2. Processes for coupling modes depend on the number of child-processes.

For the first problem, a process with parameters must be translated to processes with no parameter for analysis. The number of processes with no parameter is equal to the number of state of parameters. In order to avoid this increase, it may be elegant to use *private links* in π-calculus[9] instead of process-id's. For the second problem, multi-way communications may overcome this problem. They are supported by CSP and SCCS[10].

However, it seems that there is not a process algebra comprehending the properties above mentioned. For describing production rules, a process algebra such as π-calculus may be extended to describe multi-way communications. It may be possible. However, passing events among processes in π-calculus makes a calculus be complex, although it is very useful in general. We want to analyze production rules as easily as possible. Thus, in our research, a *specific* process algebra for production rules is preferred to a *general* useful process algebra.

3 Introduction of CCSPR

The first idea in CCSPR is to introduce processes named *resources*. Each resource holds processes and a running process can call out the processes from the resource as its child-processes. We will use resources for holding production rules. In this section, we informally introduce new combinators of CCSPR and explain how to create child-processes from resources in CCSPR.

[4] The = is observation congruence in CCS[6].

3.1 New combinators of CCSPR compared with CCS

We extend CCS to a process algebra *CCSPR* (CCS with Production Rules), adding seven basic combinators $\|$, $\rangle$, $/$, $\triangleright$, $\{\!\!\{\ \}\!\!\}$, $::$, and $[\![\ ]\!]$ for analysis of production rules. Furthermore, local-up-events $\lfloor a \rfloor$ and local-down-events $\lceil a \rceil$ are added for local communications between a parent-process and child-processes. The function of each new combinator is informally explained as follows:

- **Synchronous composition** $\|$. It is similar to a Composition | in CCS except for synchronization of only local-up-events $\lfloor a \rfloor$. For example:

  $$((\lfloor a \rfloor.P1) \,\|\, (\lfloor a \rfloor.P2)) \xrightarrow{\lfloor a \rfloor} (P1 \| P2)$$

  It is similar to a concurrency combinator $\|$ in CSP.

- **Subordinate composition** $\rangle$. It is similar to a Composition | in CCS except for multi-way local communications between a parent-process and child-processes. For example:

  $$((\lceil a \rceil.P0) \rangle (\lfloor a \rfloor.P1 \| \lfloor a \rfloor.P2)) \xrightarrow{\lceil a \rceil} (P0 \rangle (P1 \| P2))$$

  where both $P1$ and $P2$ are child-processes of $P0$. Notice that the event on the transition is $\lceil a \rceil$ instead of τ.[5] This implies that further communications with other child-processes are possible.

- **Hiding** $/$. It is exactly the same to a Hiding / in CSP. For example:

  $$(\lceil a \rceil.P3)/\lceil a \rceil \xrightarrow{\tau} P3/\lceil a \rceil$$

  In CCS, since a Hiding can be defined by a Composition and a Restriction, it is removed from basic combinators. But in CCSPR, it is needed for local-events.

- **Supplier** $\triangleright$. It supplys child-processes from left side of $\triangleright$ to right side. For example:

  $$\{\!\!\{a.Q1\}\!\!\} \triangleright (\overline{a}.P3) \xrightarrow{\tau} \{\!\!\{a.Q1\}\!\!\} \triangleright (P3 \rangle Q1)$$

  where $\{\!\!\{\ \}\!\!\}$ is a Resource combinator as explained next.

- **Resource** $\{\!\!\{\ \}\!\!\}$. It builds a resource from processes. Processes in a resource are called out by their initial events. For example:

  $$\{\!\!\{a.Q1\}\!\!\} \xrightarrow{a} \{\!\!\{a.Q1\}\!\!\} \triangleright Q1$$

- **Union** $::$ It unites two resources into one resource. If resources hold processes with same initial events, then such processes are simultaneously called out. For example:

  $$\{\!\!\{a.Q1\}\!\!\} :: \{\!\!\{a.Q2 + b.Q3\}\!\!\} :: \{\!\!\{c.Q4\}\!\!\} \xrightarrow{a} (\{\!\!\{a.Q1\}\!\!\} :: \{\!\!\{a.Q2 + b.Q3\}\!\!\} :: \{\!\!\{c.Q4\}\!\!\}) \triangleright (Q1 \| Q2)$$

- **Packing** $[\![\ ]\!]$. It is used for making two or more parent-processes. For example :

  $$\{\!\!\{a.Q1\}\!\!\} \triangleright [\![\overline{a}.P1 \mid P2]\!] \xrightarrow{\tau} \{\!\!\{a.Q1\}\!\!\} \triangleright ([\![P1 \mid P2]\!] \rangle Q1)$$

  $$\{\!\!\{a.Q1\}\!\!\} \triangleright [\![\overline{a}.P1 \mid P2]\!] \not\xrightarrow{\tau} \{\!\!\{a.Q1\}\!\!\} \triangleright [\![(P1 \rangle Q1) \mid P2]\!]$$

  In this case, parent-processes of $Q1$ are always both $P1$ and $P2$.

[5] In CCS, an internal event such as a communication is represented by τ.

3.2 Creation of child-processes

Penetration

A Supplier $\triangleright$ can *penetrate* seven combinators $+$, $|$, $\|$, $\rangle$, $[\ ]$, $\backslash$ and $/$, and it can supply child-processes from resources. A Packing $[\![\ ]\!]$ is used for forbidding the penetration. For example:

$$\{\!\!\{a.Q1\}\!\!\} \triangleright ([\![\overline{a}.P1 | P2]\!] \backslash \overline{a} | P3) \qquad (Ex.1)$$
$$\xrightarrow{\tau} \{\!\!\{a.Q1\}\!\!\} \triangleright (([\![P1 | P2]\!] \rangle Q1) \backslash \overline{a} | P3)$$

where parent-processes of $Q1$ are always both $P1$ and $P2$. If the Restriction $\backslash\overline{a}$ is removed, the following transition is *also* possible :

$$\{\!\!\{a.Q1\}\!\!\} \triangleright ([\![\overline{a}.P1 | P2]\!] | P3)$$
$$\xrightarrow{\tau} \{\!\!\{a.Q1\}\!\!\} \triangleright (([\![P1 | P2]\!] | P3) \rangle Q1)$$

On the other hand, if the Packing $[\![\ ]\!]$ is removed, the following transition is *also* possible :

$$\{\!\!\{a.Q1\}\!\!\} \triangleright ((\overline{a}.P1 | P2) \backslash \overline{a} | P3)$$
$$\xrightarrow{\tau} \{\!\!\{a.Q1\}\!\!\} \triangleright (((P1 \rangle Q1) | P2) \backslash \overline{a} | P3)$$

Therefore, the example *Ex.1* indicates that a pair of a Packing and a Restriction is useful in order to avoid nondetermination of parent-processes. It is important to notice that new child-processes are created inside of Restrictions and outside of Packings.

Cascading creation

Child-processes can also create their child-processes by penetration. For example:

$$(\{\!\!\{a.(\overline{b}.Q1) \backslash \overline{b}\}\!\!\} :: \{\!\!\{b.Q2\}\!\!\}) \triangleright (\overline{a}.P1) \backslash \overline{a} \qquad (Ex.2)$$
$$\xrightarrow{\tau} (\{\!\!\{a.(\overline{b}.Q1) \backslash \overline{b}\}\!\!\} :: \{\!\!\{b.Q2\}\!\!\}) \triangleright (P1 \rangle (\overline{b}.Q1) \backslash \overline{b}) \backslash \overline{a}$$
$$\xrightarrow{\tau} (\{\!\!\{a.(\overline{b}.Q1) \backslash \overline{b}\}\!\!\} :: \{\!\!\{b.Q2\}\!\!\}) \triangleright (P1 \rangle (Q1 \rangle Q2) \backslash \overline{b}) \backslash \overline{a}$$

where, $Q1$ is a child-process of $P1$, and $Q2$ is a child-process of $Q1$. $P1$ can not directly communicate with $Q2$ through local-events. Accordingly, $Q2$ is a grandchild-process of $P1$.

Sequential creation

One process can also sequentially create its child-processes. For example:

$$(\{\!\!\{a.Q1\}\!\!\} :: \{\!\!\{b.Q2\}\!\!\}) \triangleright (\overline{a}.\overline{b}.P1) \backslash \{\overline{a}, \overline{b}\} \qquad (Ex.3)$$
$$\xrightarrow{\tau} (\{\!\!\{a.Q1\}\!\!\} :: \{\!\!\{b.Q2\}\!\!\}) \triangleright (\overline{b}.P1 \rangle Q1) \backslash \{\overline{a}, \overline{b}\}$$
$$\xrightarrow{\tau} (\{\!\!\{a.Q1\}\!\!\} :: \{\!\!\{b.Q2\}\!\!\}) \triangleright ((P1 \rangle Q1) \rangle Q2) \backslash \{\overline{a}, \overline{b}\}$$

where, both $Q1$ and $Q2$ are child-processes of $P1$. Compare the locations of parentheses in *Ex.2* with in *Ex.3*.

4 The definitions of CCSPR

In this section, we formally define events, syntax, and semantics of CCSPR.

4.1 Events

We first assume that an infinite set $\mathcal{N} = \{a, b, c, \cdots\}$ of names is given. It is ranged over by a. Then, we define sets of events, where $\tau \notin \mathcal{N}$ and τ is a special event called an *internal event.*

Definition 4.1 *We define the following five sets of events:*

- $E_G = \{a, \overline{a} : a \in \mathcal{N}\}$ is a set of *global-events.*
- $E_{LU} = \{\lfloor a \rfloor : a \in \mathcal{N}\}$ is a set of *local-up-events.*
- $E_{LD} = \{\lceil a \rceil : a \in \mathcal{N}\}$ is a set of *local-down-events.*
- $E_L = E_{LU} \cup E_{LD}$ is a set of *local-events.*
- $Event = E_G \cup E_L \cup \{\tau\}$ is a set of *events.*

where $\overline{\overline{a}} = a$. ∎

In this paper, the sets E_G and $Event$ are ranged over by ρ and ω, respectively.

4.2 Syntax

We define the set $\mathcal{E}$ of CCSPR expressions as follows:

Definition 4.2 *The set of process expressions, $\mathcal{E}$ ranged over by $E, F, \cdots$, is the smallest set including the following expressions:*

$$\begin{array}{rl} X & : \text{a Variable } (X \in \mathcal{X}) \\ A & : \text{a Constant } (A \in \mathcal{K}) \\ R & : \text{a Resource } (R \in \mathcal{R}) \\ \mathbf{I} & : \text{a Synchronous identity} \\ \omega.E & : \text{a Prefix } (\omega \in Event) \\ \Sigma_{i\in I} E_i & : \text{a Summation } (I \text{ is an indexing set}) \\ E_1 \| E_2 & : \text{a Synchronous composition} \\ E_1 | E_2 & : \text{an Asynchronous composition} \\ E_1 \rangle E_2 & : \text{a Subordinate composition} \\ E[f] & : \text{a Relabelling } (f \in \mathcal{F}) \\ E \backslash L & : \text{a Restriction } (L \subseteq E_G \cup E_L) \\ E/L & : \text{a Hiding } (L \subseteq E_G \cup E_L) \\ [\![E]\!] & : \text{a Packing} \\ R \rhd E & : \text{a Supplier } (R \in \mathcal{R}) \end{array}$$

where E, E_i are already in $\mathcal{E}$. $\mathcal{X}$ and $\mathcal{K}$ are sets of process variables and process constants, respectively. $\mathcal{F}$ is a set of relabelling functions. The set of Resources, $\mathcal{R}$ ranged over by $R, R_1, R_2, \cdots$, is the smallest set including the following expressions:

$$\begin{array}{rl} \{\!|P|\!\} & : \text{a Resource } (P \in \mathcal{P}) \\ R_1 :: R_2 & : \text{an Union} \end{array}$$

where R_1, R_2 are already in $\mathcal{R}$. The set of process, $\mathcal{P}$ ranged over by $P, Q, \cdots$, is the smallest set including the following expressions:

$$A\,(\in \mathcal{K}),\ R\,(\in \mathcal{R}),\quad \mathbf{I},\quad \omega.P,\ \Sigma_{i\in I} P_i,\ P_1 \| P_2,\ P_1 | P_2,\ P_1 \rangle P_2,\quad P[f],\quad P\backslash L,\ P/L,\quad [\![P]\!],\quad R \rhd P$$

where P, P_i are already in $\mathcal{P}$. ∎

A *Constant* is an process whose meaning is given by defining equation. In fact, we assume that for every Constant A there is a defining equation of the following form:

$$A \stackrel{\text{def}}{=} P \qquad (P \in \mathcal{P})$$

where each occurrence of A in P is within some subexpression $\omega.P'$. In other words, A is *weakly guarded*[6] in P.

A special process *inaction* $\mathbf{0}$ is defined by using *Summation* as follows:

$$\mathbf{0} \stackrel{\text{def}}{=} \Sigma_{i\in\emptyset} E_i$$

4.3 Semantics

The semantics of CCSPR is defined by the following labelled transition system like one of CCS:

$$(\mathcal{E}, Event, \{\stackrel{\omega}{\longrightarrow} : \omega \in Event\})$$

For example, $E \stackrel{\omega}{\longrightarrow} E'$ $(E, E' \in \mathcal{E})$ indicates that the process expression E may perform the event ω and thereafter become the process expression E'. The semantics of process expressions consists of a definition of the transition relations $\stackrel{\omega}{\longrightarrow}$ over $\mathcal{E}$.

Before defining the semantics, we define a set of *syntactic initial global-events* for each process P.

Definition 4.3 *We define a set $ev(P)$ of* syntactic initial global-events *of a process P as follows:*

$$\begin{array}{rcll} ev(\omega.P) & = & \begin{cases} \{\omega\} & (\omega \in E_G) \\ \emptyset & (\omega \notin E_G) \end{cases} & \\ ev(\Sigma_{i\in I} P_i) & = & \bigcup_{i\in I} ev(P_i) & \\ ev(P \| Q) & = & ev(P) \cup ev(Q) & \\ ev(P | Q) & = & ev(P) \cup ev(Q) & \\ ev(P \rangle Q) & = & ev(P) \cup ev(Q) & \\ ev(P[f]) & = & \{f(\rho) : \rho \in ev(P)\} & \\ ev(P \backslash L) & = & ev(P) - L & \\ ev(P/L) & = & ev(P) - L & \\ ev([\![P]\!]) & = & ev(P) & \\ ev(\{\!|P|\!\}) & = & ev(P) & \\ ev(R_1 :: R_2) & = & ev(R_1) \cup ev(R_2) & \\ ev(R \rhd P) & = & ev(P) & \\ ev(\mathbf{I}) & = & \emptyset & \\ ev(A) & = & ev(P) & (A \stackrel{\text{def}}{=} P) \end{array}$$ ∎

Since a Constant A must be weakly guarded, $ev(P)$ can be always evaluated. Then, the semantics of CCSPR is defined as follows:

Definition 4.4 *The transition relation $\stackrel{\omega}{\longrightarrow}$ over process expressions is the smallest relation satisfying the following inference rules. The each rule means that if the transition relations above the line exist and the side conditions are satisfied, then the transition relation below the line also exists.*

$$\textbf{Event}\ \frac{}{\omega.E \xrightarrow{\omega} E}$$

$$\textbf{Sum}_{\mathbf{j}}\ \frac{E_j \xrightarrow{\omega} E'_j}{\Sigma_{i\in I} E_i \xrightarrow{\omega} E'_j}\quad (j \in I)$$

$$\textbf{Sync}_1\ \frac{E \xrightarrow{\omega} E'}{E\|F \xrightarrow{\omega} E'\|F}\quad (\omega \notin E_{LU})$$

$$\textbf{Sync}_2\ \frac{F \xrightarrow{\omega} F'}{E\|F \xrightarrow{\omega} E\|F'}\quad (\omega \notin E_{LU})$$

$$\textbf{Sync}_3\ \frac{E \xrightarrow{\rho} E' \qquad F \xrightarrow{\overline{\rho}} F'}{E\|F \xrightarrow{\tau} E'\|F'}$$

$$\textbf{Sync}_4\ \frac{E \xrightarrow{\lfloor a \rfloor} E' \qquad F \xrightarrow{\lfloor a \rfloor} F'}{E\|F \xrightarrow{\lfloor a \rfloor} E'\|F'}$$

$$\textbf{Com}_1\ \frac{E \xrightarrow{\omega} E'}{E|F \xrightarrow{\omega} E'|F}$$

$$\textbf{Com}_2\ \frac{F \xrightarrow{\omega} F'}{E|F \xrightarrow{\omega} E|F'}$$

$$\textbf{Com}_3\ \frac{E \xrightarrow{\rho} E' \qquad F \xrightarrow{\overline{\rho}} F'}{E|F \xrightarrow{\tau} E'|F'}$$

$$\textbf{Subo}_1\ \frac{E \xrightarrow{\omega} E'}{E\rangle F \xrightarrow{\omega} E'\rangle F}\quad (\omega \notin E_{LD})$$

$$\textbf{Subo}_2\ \frac{F \xrightarrow{\omega} F'}{E\rangle F \xrightarrow{\omega} E\rangle F'}\quad (\omega \notin E_{LU})$$

$$\textbf{Subo}_3\ \frac{E \xrightarrow{\rho} E' \qquad F \xrightarrow{\overline{\rho}} F'}{E\rangle F \xrightarrow{\tau} E'\rangle F'}$$

$$\textbf{Subo}_4\ \frac{E \xrightarrow{\lceil a \rceil} E' \qquad F \xrightarrow{\lfloor a \rfloor} F'}{E\rangle F \xrightarrow{\lceil a \rceil} E'\rangle F'}$$

$$\textbf{Rel}\ \frac{E \xrightarrow{\omega} E'}{E[f] \xrightarrow{f(\omega)} E'[f]}$$

$$\textbf{Res}\ \frac{E \xrightarrow{\omega} E'}{E\backslash L \xrightarrow{\omega} E'\backslash L}\quad (\omega \notin L)$$

$$\textbf{Hide}_1\ \frac{E \xrightarrow{\omega} E'}{E/L \xrightarrow{\tau} E'/L}\quad (\omega \in L)$$

$$\textbf{Hide}_2\ \frac{E \xrightarrow{\omega} E'}{E/L \xrightarrow{\omega} E'/L}\quad (\omega \notin L)$$

$$\textbf{Pack}\ \frac{E \xrightarrow{\omega} E'}{[\![E]\!] \xrightarrow{\omega} [\![E']\!]}$$

$$\textbf{Id}\ \frac{}{\mathbf{I} \xrightarrow{\lfloor a \rfloor} \mathbf{I}}$$

$$\textbf{Con}\ \frac{P \xrightarrow{\omega} P'}{A \xrightarrow{\omega} P'}\quad (A \stackrel{\text{def}}{=} P)$$

$$\textbf{Reso}\ \frac{P \xrightarrow{\rho} P'}{\{\!|P|\!\} \xrightarrow{\rho} \{\!|P|\!\} \rhd P'}$$

$$\textbf{Uni}_1\ \frac{R_1 \xrightarrow{\rho} R_1 \rhd P}{(R_1 :: R_2) \xrightarrow{\rho} (R_1 :: R_2) \rhd P}\quad (\rho \notin ev(R_2))$$

$$\textbf{Uni}_2\ \frac{R_2 \xrightarrow{\rho} R_2 \rhd P}{(R_1 :: R_2) \xrightarrow{\rho} (R_1 :: R_2) \rhd P}\quad (\rho \notin ev(R_1))$$

$$\textbf{Uni}_3\ \frac{R_1 \xrightarrow{\rho} R_1 \rhd P \qquad R_2 \xrightarrow{\rho} R_2 \rhd Q}{(R_1 :: R_2) \xrightarrow{\rho} (R_1 :: R_2) \rhd (P\|Q)}$$

$$\textbf{Supp}\ \frac{R \xrightarrow{\rho} R \rhd P \qquad E \xrightarrow{\overline{\rho}} E'}{R \rhd E \xrightarrow{\tau} R \rhd (E'\rangle P)}$$

$$\textbf{Nosupp}\ \frac{E \xrightarrow{\omega} E'}{R \rhd E \xrightarrow{\omega} R \rhd E'}$$

$$\textbf{S.Sum}_{\mathbf{j}}\ \frac{R \rhd E_j \xrightarrow{\tau} R \rhd E'_j}{R \rhd (\Sigma_{i\in I} E_i) \xrightarrow{\tau} R \rhd E'_j}\quad (j \in I)$$

$$\textbf{S.Sync}_1\ \frac{R \rhd E \xrightarrow{\tau} R \rhd E'}{R \rhd (E\|F) \xrightarrow{\tau} R \rhd (E'\|F)}$$

$$\textbf{S.Sync}_2\ \frac{R \rhd F \xrightarrow{\tau} R \rhd F'}{R \rhd (E\|F) \xrightarrow{\tau} R \rhd (E\|F')}$$

$$\textbf{S.Com}_1\ \frac{R \rhd E \xrightarrow{\tau} R \rhd E'}{R \rhd (E|F) \xrightarrow{\tau} R \rhd (E'|F)}$$

$$\textbf{S.Com}_2\ \frac{R \rhd F \xrightarrow{\tau} R \rhd F'}{R \rhd (E|F) \xrightarrow{\tau} R \rhd (E|F')}$$

$$\textbf{S.Subo}_1\ \frac{R \rhd E \xrightarrow{\tau} R \rhd E'}{R \rhd (E\rangle F) \xrightarrow{\tau} R \rhd (E'\rangle F)}$$

$$\textbf{S.Subo}_2\ \frac{R \rhd F \xrightarrow{\tau} R \rhd F'}{R \rhd (E\rangle F) \xrightarrow{\tau} R \rhd (E\rangle F')}$$

$$\textbf{S.Rel}\ \frac{R \rhd E \xrightarrow{\tau} R \rhd E'}{R \rhd (E[f]) \xrightarrow{\tau} R \rhd (E'[f])}$$

$$\textbf{S.Res}\ \frac{R \rhd E \xrightarrow{\tau} R \rhd E'}{R \rhd (E\backslash L) \xrightarrow{\tau} R \rhd (E'\backslash L)}$$

$$\textbf{S.Hide}\ \frac{R \rhd E \xrightarrow{\tau} R \rhd E'}{R \rhd (E/L) \xrightarrow{\tau} R \rhd (E'/L)}$$

$$\textbf{S.Con}\ \frac{R \rhd P \xrightarrow{\tau} R \rhd P'}{R \rhd A \xrightarrow{\tau} R \rhd P'}\quad (A \stackrel{\text{def}}{=} P)$$

∎

Penetration of a Supplier is implemented by means of $\textbf{S.Sum}_{\mathbf{j}}$, $\textbf{S.Sync}_{1,2}$, $\textbf{S.Com}_{1,2}$, $\textbf{S.Subo}_{1,2}$, **S.Rel**, **S.Res**, **S.Hide**, and **S.Con**. It may seem that the inference rule **Pack** is useless, but notice that there is not an inference rule such as **S.Pack**. Thus, a Packing can forbid the penetration.

5 An application of CCSPR

In this section, the example of the scheduler used in Section 2 is analyzed in CCSPR. The production rules in the system is described in CCSPR as follows:

$$\begin{aligned} Rule(X,Y) &\stackrel{\text{def}}{=} ([\![s.X[d/done]\|Y]\!]\backslash L)/H \\ L &= \{s,d,e1,e2,e3,e4,e5,e6,e7\} \\ &\quad \cup\{\overline{s},\overline{d},\overline{e1},\overline{e2},\overline{e3},\overline{e4},\overline{e5},\overline{e6},\overline{e7}\} \\ H &= \{\lceil c\rceil,\lceil sd\rceil,\lceil cd\rceil\} \end{aligned}$$

$$\begin{aligned} IM &\stackrel{\text{def}}{=} \overline{s}.d.\lceil c\rceil.\lfloor c\rfloor.\lfloor sd\rfloor.\lceil sd\rceil.\lceil cd\rceil.\lfloor cd\rfloor.\mathbf{I} \\ SP &\stackrel{\text{def}}{=} ((\overline{s}.d.\lceil c\rceil.\lceil sd\rceil.\lceil cd\rceil.\mathbf{0})|\mathbf{I}) \\ DF &\stackrel{\text{def}}{=} \lfloor c\rfloor.\lfloor sd\rfloor.\overline{s}.d.\lceil c\rceil.\lceil sd\rceil.\lceil cd\rceil.\lfloor cd\rfloor.\mathbf{I} \end{aligned}$$

$$\begin{aligned} A1 &\stackrel{\text{def}}{=} \overline{a1}.\overline{e3}.\overline{done}.\mathbf{0} & R1 &\stackrel{\text{def}}{=} e1.Rule(A1,IM) \\ A2 &\stackrel{\text{def}}{=} \overline{a2}.\overline{e4}.\overline{done}.\mathbf{0} & R2 &\stackrel{\text{def}}{=} e1.Rule(A2,IM) \\ A3 &\stackrel{\text{def}}{=} \overline{a3}.\overline{e6}.\overline{done}.\mathbf{0} & R3 &\stackrel{\text{def}}{=} e3.Rule(A3,DF) \\ A4 &\stackrel{\text{def}}{=} \overline{a4}.\overline{e7}.\overline{done}.\mathbf{0} & R4 &\stackrel{\text{def}}{=} e4.Rule(A4,SP) \\ A5 &\stackrel{\text{def}}{=} \overline{a5}.\overline{done}.\mathbf{0} & R5 &\stackrel{\text{def}}{=} e4.Rule(A5,IM) \\ A6 &\stackrel{\text{def}}{=} \overline{a6}.\overline{done}.\mathbf{0} & R6 &\stackrel{\text{def}}{=} e6.Rule(A6,IM) \\ A7 &\stackrel{\text{def}}{=} \overline{a7}.\overline{done}.\mathbf{0} & R7 &\stackrel{\text{def}}{=} e7.Rule(A7,DF) \end{aligned}$$

where, IM, SP, and DF are processes for implementing coupling modes **immediate**, **separate**, and

deferred, respectively. Notice that these processes can be used for every rules, without respect to actions in the rules. Then, the system is described as follows:

$$SYS \stackrel{\text{def}}{=} R \triangleright ((M \backslash L)/H)$$

$$R \equiv \{\!|R1|\!\} :: \{\!|R2|\!\} :: \{\!|R3|\!\} :: \{\!|R4|\!\} :: \{\!|R5|\!\} :: \{\!|R6|\!\} :: \{\!|R7|\!\}$$

$$M \stackrel{\text{def}}{=} ev1.\overline{el}.\lceil c \rceil.\lceil sd \rceil.\lceil cd \rceil.\overline{com}.\overline{done}.\mathbf{0}$$

We can check whether the scheduling system SYS behave as the expected oreder $ORDER$ defined in Section 2.2 or not, using CCSPR. We proved $SYS = ORDER$ using CWB[11] after we expanded SYS into an expression including only Prefixes and Summations by *expansion laws* which we have already obtained.

Notice that the expression in CCSPR has no process-id's. In CCSPR, local-up-events of child-processes and local-down-events of its parent-processes are *automatically* connected when the child-processes are called out. The following transition is shown as a simple example of automatic connections:

$$\{\!|a.\lfloor b \rfloor.P0|\!\} \triangleright ((\overline{a}.\lceil b \rceil.P1) \backslash a \mid \lceil b \rceil.P2)$$
$$\xrightarrow{\tau} \{\!|a.\lfloor b \rfloor.P0|\!\} \triangleright ((\lceil b \rceil.P1 \rangle \lfloor b \rfloor.P0) \backslash a \mid \lceil b \rceil.P2)$$

After $(\overline{a}.\lceil b \rceil.P1)$ calls out $(\lfloor b \rfloor.P0)$ as its child-process, $\lceil b \rceil$ in $(\lceil b \rceil.P1)$ is automatically connected to $\lfloor b \rfloor$ in $(\lfloor b \rfloor.P0)$. Hence, $(\lfloor b \rfloor.P0)$ can locally communicate with $(\lceil b \rceil.P1)$ through $\lfloor b \rfloor$ and $\lceil b \rceil$, but can not communicate with $(\lceil b \rceil.P2)$.

6 Related work

Production rules are analyzed in *directed triggering graphs*[2][3] and *Petri nets*[5]. Our approach is adopting a process algebra as an analysis tool. Since a process algebra can be one kind of programming languages, it seems useful as base of an active database language with static analysis ability.

We already proposed a process algebra $CCSGP$[12] for analysis of active database. CCSPR is more useful than CCSGP. For example, two or more parent-processes can be described by Packings in CCSPR, but can not be described in CCSGP. But the definition of CCSPR is simpler than one of CCSGP. For example, CCSPR has 37 inference rules for the semantics, while CCSGP has 51 rules.

7 Conclusion

We have stated difficulties of design of database production rules, and therefore necessity of an assistant system of rule programmers. We adopted a process algebra as a basic tool to analyze production rules, and have proposed a specific process algebra CCSPR for production rules.

In Section 2, we have pointed out problems for an application of existing process algebras to production rules. And in Section 5, we have shown that production rules can be easily analyzed in CCSPR. CCSPR is appropriate to analysis of concurrent systems with the following features:

- Processes can call out new processes from resources,
- Process trees must be constructed, and
- Multi-way local communications between a parent-process and child-processes are possible.

Active database systems have the above features. There are still remaining the researches for equivalence relations. We hope to have more powerful algebraic laws enough to prove equivalence relations between processes.

Acknowledgement

The authors wish to express our gratitude to Dr. Kimihiro Ohta, Director of Computer Science Division, Electrotechnical Laboratory. They also thank all colleagues in Information Base Section for their helpful discussions.

References

[1] D.R.McCarthy and U.Dayal, "The Architecture Of An Active Data Base Management System", Proc. of the 1989 ACM SIGMOD Conference, pp.215 - 224, 1989.

[2] A.Aiken, J.Widom, and J.M.Hellerstein, "Behavior of Database Production Rules: Termination, Confluence, and Observable Determinism", Proc. of the 1992 ACM SIGMOD Conference, pp.59 - 68, 1992.

[3] S.Ceri, P.Fraternali, S.Paraboschi, and L.Tanca, "An Architecture for Integrity Constraint Maintenance in Active Databases", Proc. Second International Computer Science Conference '92, pp.238 - 244, 1992.

[4] S.Gatziu, A.Geppert, and K.R.Dittrich, "Integrating Active Concepts into an Object-Oriented Database System", Proc. Database Programming Language, The third International Workshop, pp.399 - 415, 1991.

[5] S.Gatziu and K.R.Dittrich, "Detecting Composite Events in Active Database Systems Using Petri Nets", Proc. Fourth International Workshop on Research Issues in Data Engineering Active Database Systems, pp.2 - 9, 1994.

[6] R.Milner, "Communication and Concurrency", Prentice-Hall, 1989.

[7] C.A.R.Hoare, "Communicating Sequential Processes", Prentice-Hall, 1985.

[8] J.C.M.Baeten and W.P.Weijland, "Process Algebra", Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, 1990.

[9] R.Milner, J.Parrow and D.Walker, "A Calculus of Mobile Processes, I and II", Information and Computation, 100, pp.1 - 40 and pp.41 - 77, 1992.

[10] R.Milner, "Calculi for Synchrony and Asynchrony", Journal of Theoretical Computer Science, Vol.25, pp.267 - 310, 1983.

[11] F.Moller, "The Edinburgh Concurrency Workbench (Version 6)", 1991.

[12] Y.Isobe, I.Kojima, and K.Ohmaki, "Analysis of Communicating Systems with Generating Processes by Process Algebra", Information Processing Society of Japan SIG Notes, 94-PRG-15, pp.51 - 58, 1994.