PAPER

Analysis of Database Production Rules by Process Algebra

Yoshinao ISOBE†, **Isao KOJIMA**†, *Nonmembers and* **Kazuhito OHMAKI**†, *Member*

SUMMARY

The purpose of this research is to analyze *production rules* with *coupling modes* 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. Coupling modes are useful to control execution order of transactions. For example, a transaction for consistency check should be executed after transactions for update.

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 a set of production rules will behave because of cascading rule triggers, concurrency, and so on.

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, CSP, ACP, and π-calculus, are well known. However there are some difficulties to apply existing process algebras to analysis of production rules in growing process trees by process creation.

In this paper we propose a process algebra named *CCSPR* (a Calculus of Communicating Systems with Production Rules), which is an extension of CCS. An advantage of CCSPR is to *syntactically* describe growing process trees. Therefore, production rules can be appropriately analyzed in CCSPR. After giving definitions and properties of CCSPR, we show an example of analysis of production rules in CCSPR.

key words: active database, production rule, process algebra, process creation, process tree, multi-way communication.

1. Introduction

Active databases[2][5] 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 *child-transactions* (also called nested transactions)[2]. Thus, cascading rule triggers produce a *transaction tree*.

Manuscript received June 8, 1994.
Manuscript revised January 9, 1995.

†The authors are with Computer Science Division, Electrotechnical Laboratory, Japan

An example of a system with four production rules is shown in Fig.1. In this example, when a variable `x` is updated, an event `updatex` occurs. An event `updatey` is similar to the case of `updatex`. Assume that the variable `x` is updated. It causes the event `updatex`, then `Rule1` and `Rule2` are triggered. If the variable `x` is less than 100, then the action in `Rule1` decreases a variable `y`. Thereafter, the update of `y` triggers `Rule3` and `Rule4`, and a transaction tree is produced as shown in Fig.1. Furthermore `Rule3` may trigger `Rule1` and `Rule2`.

We sometimes require transactions in a transaction tree to be executed in some order. For example, transactions for consistency check should be executed just prior to all transactions committing. In order to control their execution, *coupling modes* are very useful as introduced in a survey[1] of active databases. HiPAC[2] and SAMOS[5] are well known as active databases with coupling modes.

Coupling modes do not only specify relations between a parent-transaction and a child-transaction, but often express relations between a transaction and all the other transactions in its transaction tree (such as a coupling mode **deferred**). Coupling modes are simple but are powerful notions to control execution order. There are three possible coupling modes[2][5]:

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 commits of parent-transactions, the second coupling mode **separate** may be used. In other cases, such as consistency constraints, the third coupling mode **deferred** may be used.

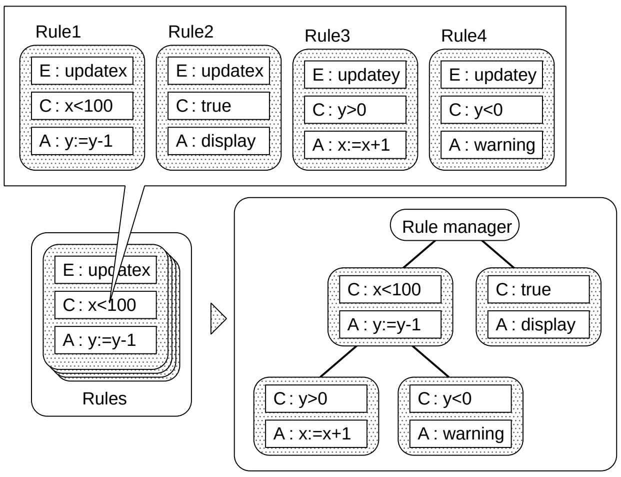

Fig. 1 An example of production rules

In above example, `Rule1` and `Rule3` may have an **immediate**, `Rule2` may have a **separate**, and `Rule4` may have a **deferred**, as a coupling mode.

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, because of cascading rule triggers, coupling modes, and so on. Hence, it is important to statically analyze behaviors of rules before they are executed and to aid rule programming.

Static analysis methods of active databases have already been proposed. For example, directed triggering graphs[3][4] are used for providing information about properties of rule behavior. On the other hand, Petri nets[6] are used for detecting composite events. However, existing static analysis does not consider coupling modes. We want to statically check execution order of production rules with coupling modes.

We are attempting to adopt a *process algebra* as a basic tool to analyze production rules with coupling modes. Process algebras such as CCS[7], CSP[8], ACP[9], and π-calculus[10], are well known as mathematical tools to describe and analyze concurrent and communicating systems. Process algebras have advantages as follows:

- A process algebra can be one kind of programming languages, and seems useful as base of an active database language with static analysis ability, like a parallel programming language $\mathcal{M}$[7] on CCS.
- We can use various equivalence relations, for example, trace equivalence, failures equivalence, and observation equivalence, which have different levels of equality.

We have three requirements at least in order to analyze production rules as follows:

1. A process can call out processes as its child-processes from a resource which holds the processes.
2. Relations between parents-processes and child-processes must be uniquely determined, for producing a process tree.
3. Multi-way local communications between a parent-process and child-processes are needed for implementing coupling modes.

We can point out some difficulties of existing process algebras for analyzing the above properties. For example, it is difficult to describe and analyze processes in growing process trees.

In this paper we propose a process algebra named *CCSPR* (a Calculus of Communicating Systems with Production Rules). The above three requirements are appropriately described and analyzed in CCSPR. For example, growing process trees can be *syntactically* described.

In Section 2 we introduce process algebras and point out some difficulties of existing process algebras for analyzing production rules. In Section 3 it is informally explained how to create child-processes in CCSPR. In Section 4 definitions of CCSPR are given. In Section 5 we show several properties for strong equivalence in CCSPR. In Section 6 an example of analysis of production rules in CCSPR is given.

2. Process algebra

Process algebras are well known as mathematical tools to describe and analyze concurrent and communicating systems. Behavior of processes is described as *(process) expressions* in a process algebra, then equality of behavior of two processes is proven by rewriting their expressions according to algebraic (rewriting) laws in the process algebra.

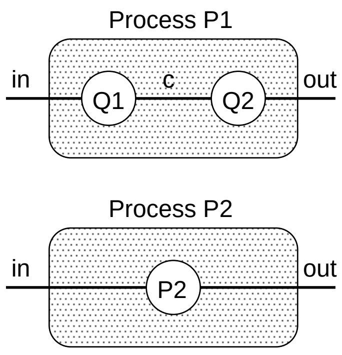

Fig. 2 An example of processes

At first we introduce process algebra by using an example. Next we show how to describe production rules in existing process algebras, and point out difficulties of the description.

2.1 Introduction of process algebras

We explain how to describe and analyze concurrent systems in CCS by using a simple example of two processes P_1 and P_2 in Fig.2. CCS is a fundamental process algebra proposed by R.Milner[7].

P_1 consists of two concurrent processes Q_1 and Q_2, and P_2 consists of one process. When Q_1 receives an event *in*, Q_1 sends an event c and becomes an inaction process $\mathbf{0}$. We can describe the behavior of Q_1 in CCS as follows:

$$Q_1 \overset{\text{def}}{=} in.\overline{c}.\mathbf{0}$$

where '.' is a sequential composition of events and an overline of $\overline{c}$ means sending. In general a transition by an event a is denoted by $\xrightarrow{a}$. In this case, $(in.\overline{c}.\mathbf{0})$ becomes $(\overline{c}.\mathbf{0})$ by receiving the envet *in* and $(\overline{c}.\mathbf{0})$ becomes $\mathbf{0}$ by sending the envet c. Hence the transitions are written as follows:

$$in.\overline{c}.\mathbf{0} \xrightarrow{in} \overline{c}.\mathbf{0} \xrightarrow{\overline{c}} \mathbf{0}$$

As similar to Q_1, Q_2 is described as follows:

$$Q_2 \overset{\text{def}}{=} c.\overline{out}.\mathbf{0}$$

It means that Q_2 receives the event c and thereafter sends an event *out*. P_1 is a composite process of the two processes Q_1 and Q_2 and is described as follows:

$$P_1 \overset{\text{def}}{=} (Q_1|Q_2)\backslash\{c\}$$

where '|' and '\' are a concurrent composition combinator and a restriction combinator, respectively. Therefore the event c is not observed from the outside of P_1. On the other hand, P_2 is simply describe as follows:

$$P_2 \overset{\text{def}}{=} in.\overline{out}.\mathbf{0}$$

The two processes are observationally equal because difference between behaviors of P_1 and P_2 is not observed. This equality is proven by rewriting the expression of P_1 into the expression of P_2 according to algebraic laws in CCS as follows:

$$\begin{array}{rcll}
P1 & = & (Q_1|Q_2)\backslash\{c\} & \text{Definition} \\
 & = & in.(\overline{c}.\mathbf{0}|Q_2)\backslash\{c\} & \text{Expansion law} \\
 & = & in.\tau.(\mathbf{0}|\overline{out}.\mathbf{0})\backslash\{c\} & \text{Expansion law} \\
 & = & in.\tau.\overline{out}.(\mathbf{0}|\mathbf{0})\backslash\{c\} & \text{Expansion law} \\
 & = & in.\overline{out}.(\mathbf{0}|\mathbf{0})\backslash\{c\} & \tau \text{ law} \\
 & = & in.\overline{out}.\mathbf{0} & \text{Static laws} \\
 & = & P_2 & \text{Definition}
\end{array}$$

We may also consider that P_1 is a concurrent implementation for a specification P_2. The specification show a explicit behavior of the implementation. As shown in this example we can analyze concurrent systems.

2.2 Problems of existing process algebras

An important problem of existing process algebras for analyzing production rules is to describe a *growing* process tree. A subordinate composition to relate parent-processes and child-processes in a *fixed* process tress is easily defined in existing process algebras. In fact it is defined in CSP[8] as follows:

$$Q /\!/ P$$

where Q is a subordinate process of P. This description syntactically shows the relation between processes. But it is not available for a growing process tree. The other way to relate parent-processes and child-processes is to use links such as *process-id's* or *private links*, as shown in practical implementations.

For example, a growing process tree is described by using process-id's in CCS as follows:

$$\begin{array}{l}(R \mid PR1 \mid PR2 \mid ID(I)) \xrightarrow{\tau}\xrightarrow{\tau} \\ \quad (R \mid NEWPR(I) \mid PR1'(I) \mid PR2 \mid ID(I+1))\end{array}$$

where each component is defined as follows:

$$\begin{array}{rcl}
R & \overset{\text{def}}{=} & a(i).(R \mid NEWPR(i)) \\
PR1 & \overset{\text{def}}{=} & getid(i).\overline{a}(i).PR1'(i) \\
PR2 & \overset{\text{def}}{=} & getid(i).\overline{a}(i).PR2'(i) \\
ID(i) & \overset{\text{def}}{=} & \overline{getid}(i).ID(i+1)
\end{array}$$

A process $PR1$ gets its own process-id from a process ID which gives a process-id through an event *getid*, and $PR1$ calls out a process $NEWPR$ from a process R through an event a. A process $PR2$ is similar to $PR1$. The first communication denoted by $\xrightarrow{\tau}$ is caused by *getid* between $PR1$ and ID, and the second communication denoted by $\xrightarrow{\tau}$ is caused by a between $PR1$ and R. A postfix (i) of an event is a message passed by the event. The process-id I determinate that $NEWPR$ is a

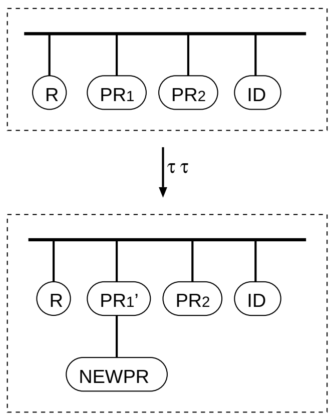

Fig. 3 An example of process creation

child-process of *PR*1, but not of *PR*2. Fig.3 shows the above transitions by process creation of *NEWPR*.

On the other hand, the above example can be also described by private links in π-calculus as follows:

$$\begin{aligned}&(R \mid PR1 \mid PR2) \xrightarrow{\tau}\\ &\quad ((pr)((R \mid NEWPR)\{pr/x\} \mid PR1') \mid PR2)\end{aligned}$$

where each component is defined as follows:

$$\begin{aligned}R &\overset{\text{def}}{=} a(x).(R \mid NEWPR)\\ PR1 &\overset{\text{def}}{=} \overline{a}(pr).PR1'\\ PR2 &\overset{\text{def}}{=} \overline{a}(pr).PR2'\end{aligned}$$

By passing a name *pr* from *PR*1 to *NEWPR*, a private link is made between them. They can locally communicate each other through *pr*.

As shown in the above example, we can describe growing process trees in existing process algebras. But such descriptions may be too cumbersome to analyze in general large systems.

3. Introduction of CCSPR

We propose a process algebra named *CCSPR* (a Calculus of Communicating Systems with Production Rules). An important property of CCSPR is to *syntactically* describe relations between processes in *growing* process tress by *Subordinate compositions* $\rangle$. The example used in Section 2 can be described in CCSPR as follows:

$$R \rhd (PR1 \mid PR2) \xrightarrow{\tau} R \rhd ((PR1' \rangle NEWPR) \mid PR2)$$

where each component is defined as follows[†]:

$$\begin{aligned}R &\equiv \{\!\!\{a.NEWPR\}\!\!\}\\ PR1 &\overset{\text{def}}{=} \overline{a}.PR1'\\ PR2 &\overset{\text{def}}{=} \overline{a}.PR2'\end{aligned}$$

[†] $\equiv$ means syntactic identity

where R is a *resource* which holds a process *NEWPR* and can supply the process through an event a. $\{\!\!\{\ \}\!\!\}$ is a resource combinator. A process P in $\{\!\!\{P\}\!\!\}$ is called out through *initial* events which P can perform. $\rhd$ is a combinator which supplies child-processes from resources. Notice the location of the created process *NEWPR*. $\rhd$ supplys *NEWPR* just under *PR*1 from R, *penetrating* $|$. The *penetration* is the most important property in CCSPR. On the other hand, the following transition is also possible:

$$\begin{aligned}&R \rhd (PR1 \mid PR2) \xrightarrow{\tau}\\ &\quad R \rhd ((PR1' \mid PR2) \rangle NEWPR)\end{aligned}$$

In this case, both *PR*1 and *PR*2 are parent-processes of *NEWPR*.

We can use Restrictions $\backslash$ and Packings $[\![\ ]\!]$ in order to prevent the nondeterminism of parents, thus construct an unique process tree. For example:

$$\begin{aligned}&\{\!\!\{a.Q1\}\!\!\} \rhd ([\![\overline{a}.P1|P2]\!]\backslash\overline{a}|P3) \xrightarrow{\tau}\\ &\quad \{\!\!\{a.Q1\}\!\!\} \rhd (([\![P1|P2]\!]\rangle Q1)\backslash\overline{a}|P3)\end{aligned}$$

where parent-processes of $Q1$ are always both $P1$ and $P2$. A Supplier $\rhd$ can penetrate $|$ and $\backslash$, and it can supply child-processes from resources. A Packing $[\![\ ]\!]$ is used for forbidding the penetration. If the Restriction $\backslash\overline{a}$ is removed, the following transition is *also* possible:

$$\begin{aligned}&\{\!\!\{a.Q1\}\!\!\} \rhd ([\![\overline{a}.P1|P2]\!]|P3) \xrightarrow{\tau}\\ &\quad \{\!\!\{a.Q1\}\!\!\} \rhd (([\![P1|P2]\!]|P3))\rangle Q1)\end{aligned}$$

In this case, all P_1, P_2, and P_3 are parents of Q_1. On the other hand, if the Packing $[\![\ ]\!]$ is removed, the following transition is *also* possible:

$$\begin{aligned}&\{\!\!\{a.Q1\}\!\!\} \rhd ((\overline{a}.P1|P2)\backslash\overline{a}|P3) \xrightarrow{\tau}\\ &\quad \{\!\!\{a.Q1\}\!\!\} \rhd (((P1\rangle Q1)|P2)\backslash\overline{a}|P3)\end{aligned}$$

In this case, only P_1 is a parent of Q_1. It is important to notice that new child-processes are created inside of Restrictions and outside of Packings.

Next we show example of cascading creation. Thus a child-process can call out the other process as its child-process from a resource as follows:

$$\begin{aligned}&(\{\!\!\{a.(\overline{b}.Q1)\backslash\overline{b}\}\!\!\} :: \{\!\!\{b.Q2\}\!\!\}) \rhd (\overline{a}.P1)\backslash\overline{a} \xrightarrow{\tau}\\ &(\{\!\!\{a.(\overline{b}.Q1)\backslash\overline{b}\}\!\!\} :: \{\!\!\{b.Q2\}\!\!\}) \rhd (P1\rangle(\overline{b}.Q1)\backslash\overline{b})\backslash\overline{a} \xrightarrow{\tau}\\ &(\{\!\!\{a.(\overline{b}.Q1)\backslash\overline{b}\}\!\!\} :: \{\!\!\{b.Q2\}\!\!\}) \rhd (P1\rangle(Q1\rangle Q2)\backslash\overline{b})\backslash\overline{a} \quad (*)\end{aligned}$$

where a combinator :: is used for uniting two resources into one resource. In this case, $Q1$ is a child-process of $P1$, and $Q2$ is a child-process of $Q1$. Thus $Q2$ is a grandchild-process of $P1$.

Sequential creation is also possible. Thus one process can sequentially call out processes as its child-processes from a resource as shown in the following example:

$$\begin{aligned}&(\{\!\!\{a.Q1\}\!\!\} :: \{\!\!\{b.Q2\}\!\!\}) \rhd (\overline{a}.\overline{b}.P1)\backslash\{\overline{a},\overline{b}\} \xrightarrow{\tau}\\ &(\{\!\!\{a.Q1\}\!\!\} :: \{\!\!\{b.Q2\}\!\!\}) \rhd (\overline{b}.P1\rangle Q1)\backslash\{\overline{a},\overline{b}\} \xrightarrow{\tau}\\ &(\{\!\!\{a.Q1\}\!\!\} :: \{\!\!\{b.Q2\}\!\!\}) \rhd ((P1\rangle Q1)\rangle Q2)\backslash\{\overline{a},\overline{b}\} \quad (**)\end{aligned}$$

In this case, both $Q1$ and $Q2$ are child-processes of $P1$. Compare the locations of parentheses in $(*)$ with $(**)$.

4. The definition 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 $\overline{\overline{a}} = a$. ▮

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. If a local-down-event $\lceil a \rceil$ of a parent-process has the same event name a as local-up-events $\lfloor a \rfloor$ of child-processes, then they can communicate though $\lceil a \rceil$ and $\lfloor a \rfloor$. 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.

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

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

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 $R, R', R_i, \cdots$, is the smallest set including the following expressions:

$\{\!\{P\}\!\}$	: a Resource $(P \in \mathcal{P})$
$R_1 :: R_2$	: an Union

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:

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

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

Synchronous composition $\Vert$ is similar to a Composition $\vert$ in CCS except for synchronization of only local-up-events $\lfloor a \rfloor$ (see $\mathbf{sync_4}$ of Semantics). *Hiding* / is exactly the same to a Hiding / in CSP, thus it changes an event into an internal event.

A relabelling function $f : Event \rightarrow Event$ must satisfy the following conditions:

$\bullet f(\omega) \in E_G$ **iff** $\omega \in E_G$	$\bullet f(\omega) \in E_L$ **iff** $\omega \in E_L$
$\bullet f(\rho) = f(\overline{\rho})$	$\bullet \lfloor f(a) \rfloor = f(\lfloor a \rfloor)$
$\bullet \lceil f(a) \rceil = f(\lceil a \rceil)$	$\bullet f(\tau) = \tau$

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:

$$A \overset{\mathrm{def}}{=} P \qquad (P \in \mathcal{P})$$

where each occurrence of A in P is within some subexpression $\omega.P'$. In other words, A is *weakly guarded*[7] in P. Constants which are not weakly guarded make a calculus be more complex, and the behavior is indefinite[12]. Practically, we are interested in only weakly guarded Constants.

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

$$\mathbf{0} \overset{\mathrm{def}}{=} \Sigma_{i\in\emptyset} E_i$$

Another special process *Synchronous identity* $\mathbf{I}$ is a process which can always perform arbitrary local-up-events. Although it can be defined by an infinite Summation, it is given as a basic process because we often treat only finite Summations.

4.3 Semantics

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

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

For example, $E \xrightarrow{\omega} E'$ $(E, E' \in \mathcal{E})$ 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 $\xrightarrow{\omega}$ 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:

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

∎

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

Definition 4.4: The transition relation $\xrightarrow{\omega}$ over process expressions is the smallest relation satisfying the inference rules in Table 1. Each rule means that if transition relations above a line exist and side conditions are satisfied, then a transition relation below the line also exists.

∎

Penetration of a Supplier is implemented by means of $\mathbf{S.Sum_j}$, $\mathbf{S.Sync_{1,2}}$, $\mathbf{S.Com_{1,2}}$, $\mathbf{S.Subo_{1,2}}$, **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.

As shown in $\mathbf{Subo_4}$, the result of communication though local-events is a local-down-event instead of an internal event. It makes multi-way communications be possible, and we should use a Hiding / instead of a Restriction \ in order to prevent interaction of local events.

Under the semantics, a set $ev(P)$ of *syntactic initial global-events* of a process P is actually equivalent to a set of global-events which P can initially perform.

5. Equivalence relations in CCSPR

We define equivalence relations in CCSPR like in CCS, for example, strong equivalence and observation equivalence. In this section, we show some properties of CCSPR for strong equivalence.

Strong equivalence is defined by strong bisimulations as follows[7].

Definition 5.1: Strong bisimulations

A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ over processes is a *strong bisimulation* if $(P, Q) \in \mathcal{S}$ implies, for all $\omega \in$ *Event*, that

(*i*) whenever $P \xrightarrow{\omega} P'$ then, for some Q', $Q \xrightarrow{\omega} Q'$ and $(P', Q') \in \mathcal{S}$,

(*ii*) whenever $Q \xrightarrow{\omega} Q'$ then, for some P', $P \xrightarrow{\omega} P'$ and $(P', Q') \in \mathcal{S}$.

∎

Definition 5.2: Strong equivalence

P and Q are strongly equivalent, written $P \sim Q$, if $(P, Q) \in \mathcal{S}$ for some strong bisimulation $\mathcal{S}$. ∎

In CCSPR, the most used expression has a form $R \rhd P$. First, we show expansion laws which expand an expression such as $R \rhd P$ into an expression with Prefixes and Summations. The expansion laws are inductively given for outmost structure of P. We show a part of all the laws by lack of spaces.

Proposition 5.1: The expansion laws (part)

- Prefix with Resource

  If $(R \sim \sum_{(i)} \rho_i.(R \rhd Q_i))$, then

$$\begin{aligned}
&R \rhd (\omega.P) \sim \\
&\omega.(R \rhd P) + \textstyle\sum_{(i)}\{\tau.(R \rhd (P\rangle Q_j)) : \omega = \overline{\rho_j}\}
\end{aligned}$$

- Summation with Resource

$$R \rhd (\textstyle\sum_{(i\in I)} P_i) \sim \sum_{(i\in I)}(R \rhd P_i)$$

- Synchronous composition with Resource

  If $(R \sim \sum_{(i)} \rho_i.(R \rhd Q_i))$, $(R \rhd P_i \sim \sum_{(j)} \omega_{ij}.(R \rhd P_{ij}))$, for $i \in \{1,2\}$), then

$$\begin{aligned}
&R \rhd (P_1 \| P_2) \sim \\
&\textstyle\sum_{(i)}\{\omega_{1i}.(R \rhd (P_{1i}\|P_2)) : \omega_{1i} \notin E_{LU}\} \\
&+\textstyle\sum_{(i)}\{\omega_{2i}.(R \rhd (P_1\|P_{2i})) : \omega_{2i} \notin E_{LU}\} \\
&+\textstyle\sum_{(i)}\sum_{(j)}\{\tau.(R \rhd ((P_{1i}\|P_2)\rangle Q_j)) : \omega_{1i} = \overline{\rho_j}\} \\
&+\textstyle\sum_{(i)}\sum_{(j)}\{\tau.(R \rhd ((P_1\|P_{2i})\rangle Q_j)) : \omega_{2i} = \overline{\rho_j}\} \\
&+\textstyle\sum_{(i)}\sum_{(j)}\{\tau.(R \rhd (P_{1i}\|P_{2j})) : \omega_{1i} = \rho, \omega_{2j} = \overline{\rho}\} \\
&+\textstyle\sum_{(i)}\sum_{(j)}\{\lfloor a\rfloor.(R \rhd (P_{1i}\|P_{2j})) : \omega_{1i} = \omega_{2j} = \lfloor a\rfloor\}
\end{aligned}$$

- Relabelling with Resource

  If $(R \sim \sum_{(i)} \rho_i.(R \rhd Q_i))$, $(R \rhd P \sim \sum_{(i)} \omega_i.(R \rhd P_i))$, then

$$\begin{aligned}
&R \rhd (P[f]) \sim \\
&\textstyle\sum_{(i)} f(\omega_i).(R \rhd (P_i[f])) \\
&+\textstyle\sum_{(i)}\sum_{(j)}\{\tau.(R \rhd ((P_i[f])\rangle Q_j)) : f(\omega_i) = \overline{\rho_j}\}
\end{aligned}$$

- Packing with Resource

  If $(R \sim \sum_{(i)} \rho_i.(R \rhd Q_i))$, $(P \sim \sum_{(i)} \omega_i.P_i)$, then

$$\begin{aligned}
&R \rhd [\![P]\!] \sim \\
&\textstyle\sum_{(i)} \omega_i.(R \rhd [\![P_i]\!]) \\
&+\textstyle\sum_{(i)}\sum_{(j)}\{\tau.(R \rhd ([\![P_i]\!]\rangle Q_j)) : \omega_i = \overline{\rho_j}\}
\end{aligned}$$

Table 1 The inference rules of CCSPR

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

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

$$\textbf{Sync}_{\textbf{1}}\ \frac{E \xrightarrow{\omega} E'}{E\|F \xrightarrow{\omega} E'\|F}\quad (\omega\notin E_{LU})$$

$$\textbf{Sync}_{\textbf{2}}\ \frac{F \xrightarrow{\omega} F'}{E\|F \xrightarrow{\omega} E\|F'}\quad (\omega\notin E_{LU})$$

$$\textbf{Sync}_{\textbf{3}}\ \frac{E \xrightarrow{\rho} E' \qquad F \xrightarrow{\overline{\rho}} F'}{E\|F \xrightarrow{\tau} E'\|F'}$$

$$\textbf{Sync}_{\textbf{4}}\ \frac{E \xrightarrow{\lfloor a\rfloor} E' \qquad F \xrightarrow{\lfloor a\rfloor} F'}{E\|F \xrightarrow{\lfloor a\rfloor} E'\|F'}$$

$$\textbf{Com}_{\textbf{1}}\ \frac{E \xrightarrow{\omega} E'}{E|F \xrightarrow{\omega} E'|F}$$

$$\textbf{Com}_{\textbf{2}}\ \frac{F \xrightarrow{\omega} F'}{E|F \xrightarrow{\omega} E|F'}$$

$$\textbf{Com}_{\textbf{3}}\ \frac{E \xrightarrow{\rho} E' \qquad F \xrightarrow{\overline{\rho}} F'}{E|F \xrightarrow{\tau} E'|F'}$$

$$\textbf{Subo}_{\textbf{1}}\ \frac{E \xrightarrow{\omega} E'}{E\rangle F \xrightarrow{\omega} E'\rangle F}\quad (\omega\notin E_{LD})$$

$$\textbf{Subo}_{\textbf{2}}\ \frac{F \xrightarrow{\omega} F'}{E\rangle F \xrightarrow{\omega} E\rangle F'}\quad (\omega\notin E_{LU})$$

$$\textbf{Subo}_{\textbf{3}}\ \frac{E \xrightarrow{\rho} E' \qquad F \xrightarrow{\overline{\rho}} F'}{E\rangle F \xrightarrow{\tau} E'\rangle F'}$$

$$\textbf{Subo}_{\textbf{4}}\ \frac{E \xrightarrow{\lceil a\rceil} E' \qquad F \xrightarrow{\lfloor a\rfloor} F'}{E\rangle F \xrightarrow{\lceil a\rceil} E'\rangle F'}$$

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

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

$$\textbf{Hide}_{\textbf{1}}\ \frac{E \xrightarrow{\omega} E'}{E/L \xrightarrow{\tau} E'/L}\quad (\omega\in L)$$

$$\textbf{Hide}_{\textbf{2}}\ \frac{E \xrightarrow{\omega} E'}{E/L \xrightarrow{\omega} E'/L}\quad (\omega\notin L)$$

$$\textbf{Pack}\ \frac{E \xrightarrow{\omega} E'}{\llbracket E\rrbracket \xrightarrow{\omega} \llbracket E'\rrbracket}$$

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

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

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

$$\textbf{Uni}_{\textbf{1}}\ \frac{R_1 \xrightarrow{\rho} R_1\triangleright P}{(R_1::R_2) \xrightarrow{\rho} (R_1::R_2)\triangleright P}\quad (\rho\notin ev(R_2))$$

$$\textbf{Uni}_{\textbf{2}}\ \frac{R_2 \xrightarrow{\rho} R_2\triangleright P}{(R_1::R_2) \xrightarrow{\rho} (R_1::R_2)\triangleright P}\quad (\rho\notin ev(R_1))$$

$$\textbf{Uni}_{\textbf{3}}\ \frac{R_1 \xrightarrow{\rho} R_1\triangleright P \qquad R_2 \xrightarrow{\rho} R_2\triangleright Q}{(R_1::R_2) \xrightarrow{\rho} (R_1::R_2)\triangleright(P\|Q)}$$

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

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

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

$$\textbf{S.Sync}_{\textbf{1}}\ \frac{R\triangleright E \xrightarrow{\tau} R\triangleright E'}{R\triangleright(E\|F) \xrightarrow{\tau} R\triangleright(E'\|F)}$$

$$\textbf{S.Sync}_{\textbf{2}}\ \frac{R\triangleright F \xrightarrow{\tau} R\triangleright F'}{R\triangleright(E\|F) \xrightarrow{\tau} R\triangleright(E\|F')}$$

$$\textbf{S.Com}_{\textbf{1}}\ \frac{R\triangleright E \xrightarrow{\tau} R\triangleright E'}{R\triangleright(E|F) \xrightarrow{\tau} R\triangleright(E'|F)}$$

$$\textbf{S.Com}_{\textbf{2}}\ \frac{R\triangleright F \xrightarrow{\tau} R\triangleright F'}{R\triangleright(E|F) \xrightarrow{\tau} R\triangleright(E|F')}$$

$$\textbf{S.Subo}_{\textbf{1}}\ \frac{R\triangleright E \xrightarrow{\tau} R\triangleright E'}{R\triangleright(E\rangle F) \xrightarrow{\tau} R\triangleright(E'\rangle F)}$$

$$\textbf{S.Subo}_{\textbf{2}}\ \frac{R\triangleright F \xrightarrow{\tau} R\triangleright F'}{R\triangleright(E\rangle F) \xrightarrow{\tau} R\triangleright(E\rangle F')}$$

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

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

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

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

Proof Use strong bisimulations and Proposition 5.3. ∎

Second, we give substitution laws for $R\triangleright P$. For strong equivalence, the part R of $R\triangleright P$ is substitutive as shown in the following proposition.

Proposition 5.2: The substitution law of Resources

For any $R_1, R_2 \in \mathcal{R}$, $P \in \mathcal{P}$,
If $R_1 \sim R_2$, then $R_1 \triangleright P \sim R_2 \triangleright P$.

Proof The proof is accomplished by showing that $\mathcal{S}$ defined below is a strong bisimulation, using induction over n and structures of processes.

$$\begin{aligned}\mathcal{S}^{(1)} = \{ & (R_1 \triangleright P, R_2 \triangleright P), (R_1 \triangleright Q_1, R_2 \triangleright Q_2) \\ & : (P, Q_1, Q_2 \in \mathcal{P}), (R_1, R_2 \in \mathcal{R}), \\ & R_1 \sim R_2, R_1 \triangleright Q_1 \sim R_2 \triangleright Q_2 \ \}\end{aligned}$$

$(n \geqq 2)$

$$\begin{aligned}\mathcal{S}^{(n)} = \{ & (R_1 \triangleright (P_1 \rangle Q_1), R_2 \triangleright (P_2 \rangle Q_2)), \\ & (R_1 \triangleright (P_1 \| Q_1), R_2 \triangleright (P_2 \| Q_2)), \\ & (R_1 \triangleright (P_1 | Q_1), R_2 \triangleright (P_2 | Q_2)), \\ & (R_1 \triangleright (P_1[f]), R_2 \triangleright (P_2[f])), \\ & (R_1 \triangleright (P_1 \backslash L), R_2 \triangleright (P_2 \backslash L)), \\ & (R_1 \triangleright (P_1 / L), R_2 \triangleright (P_2 / L)) \\ & \\ & : (R_1 \triangleright P_1, R_2 \triangleright P_2) \in \mathcal{S}^{(n-1)}, \\ & (R_1 \triangleright Q_1, R_2 \triangleright Q_2) \in \mathcal{S}^{(n-1)}, \\ & f \in \mathcal{F}, L \subseteqq E_G \cup E_L \ \}\end{aligned}$$

$$\mathcal{S} = \bigcup_i \mathcal{S}^{(i)}$$

∎

Unfortunately, the part P of $R \triangleright P$ is not substitutive even for the strong equivalence. For example,

$$\begin{aligned}(\overline{a}.\mathbf{0}) \backslash \overline{a} &\sim \mathbf{0} \\ \{\!\{a.\mathbf{0}\}\!\} \triangleright (\overline{a}.\mathbf{0}) \backslash \overline{a} &\not\sim \{\!\{a.\mathbf{0}\}\!\} \triangleright \mathbf{0}\end{aligned}$$

since $\{\!\{a.\mathbf{0}\}\!\} \triangleright (\overline{a}.\mathbf{0}) \backslash \overline{a}$ has a τ-derivation. Thus, $\sim$ is not a congruence relation in CCSPR.

In order to make up for the above defect, we give *construction laws* which are used for constructing larger expressions from smaller expressions, preserving strong equivalence. The construction laws are given for every structure of P in $R \triangleright P$, however the part are shown by lack of spaces in this paper.

Proposition 5.3: The construction laws (part)

- If $R \triangleright P \sim R \triangleright Q$, then $R \triangleright (\omega.P) \sim R \triangleright (\omega.Q)$ for any $\omega \in \mathit{Event}$.
- If $R \triangleright P_i \sim R \triangleright Q_i$ $(i \in I)$, then $R \triangleright (\Sigma_{i \in I} P_i) \sim R \triangleright (\Sigma_{i \in I} Q_i)$.
- If $R \triangleright P_i \sim R \triangleright Q_i$ $(i \in \{1, 2\})$, then $R \triangleright (P_1 \| P_2) \sim R \triangleright (Q_1 \| Q_2)$.
- If $R \triangleright P \sim R \triangleright Q$, then $R \triangleright (P[f]) \sim R \triangleright (Q[f])$ for any $f \in \mathcal{F}$.
- If $P \sim Q$, then $R \triangleright [\![P]\!] \sim R \triangleright [\![Q]\!]$.

∎

We can prove an equivalence relation $R \triangleright (P \rangle \mathbf{I}) \sim R \triangleright P$, for any $P \in \mathcal{P}$ and $R \in \mathcal{R}$, using a strong bisimulation. Then, we can prove an equivalence relation $R \triangleright ((P \rangle \mathbf{I}) \| Q) \sim R \triangleright (P \| Q)$, using the above construction laws.

6. An analysis of production rules in CCSPR

At first we explain how to describe production rules in CCSPR. Next we show a scheduler as an example of production rules and analyze the scheduler using CCSPR.

6.1 Process creation in CCSPR

In this subsection we give key technique to use CCSPR. It is summarized as follows:

1. **Resource**: A resource which holds a process P called out through an event a is described as $\{\!\{a.P\}\!\}$. It is a typical form of a resource. Events for call out must be global events.

   If P is called through either an event a or b, then we can describe as $\{\!\{a.P + b.P\}\!\}$.

   Two resources is united into one resource by a Uniting. For example, an united resource of $\{\!\{a.P\}\!\}$ and $\{\!\{b.Q\}\!\}$ is described as $\{\!\{a.P\}\!\} :: \{\!\{b.Q\}\!\}$. If resources hold processes with same events for call out, then such processes are simultaneously called out.

2. **Supplier**: Processes in a resource are supplied by a Supplier $\triangleright$. A running process calls out a process in a resource through a complementary event as follows:

   $$\{\!\{a.P\}\!\} \triangleright (\overline{a}.Q) \xrightarrow{\tau} \{\!\{a.P\}\!\} \triangleright (Q \rangle P)$$

   If several processes concurrently run, then Restrictions $\backslash$ and Packings $[\![\ ]\!]$ should be used in order to avoid nondeterminism of parents as shown in Section 3.

   In general, local-down-events of processes called out should be hidden in order to prevent communication with child-processes of the other processes. It does not implies that the processes do not communicate their child-processes, because their child-processes can intrude inside Hidings / by penetration. $\mathit{Rule}(X, Y)$ defined in Subsection 6.3 is such a typical example.

3. **Local communication**: 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:

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

   After a process $(\overline{a}.\lceil b \rceil.P1)$ calls out a process

Table 2 The specification of each production rule

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**

$(\lfloor b \rfloor.P0)$ as its child-process from a resource $\{\!\{a.\lfloor b \rfloor.P0\}\!\}$, a local-down-event $\lceil b \rceil$ in $(\lceil b \rceil.P1)$ is automatically connected to a local-up-event $\lfloor b \rfloor$ in $(\lfloor b \rfloor.P0)$. Hence, $(\lceil b \rceil.P1)$ can locally communicate with $(\lfloor b \rfloor.P0)$ through $\lceil b \rceil$ and $\lfloor b \rfloor$, but can not communicate with $(\lceil b \rceil.P2)$.

The automatic connections of local-events are used instead of connections by links such as process-id's. Links make analysis to be more complex. Therefore we wanted to remove such links.

6.2 An example of production rules

In this Subsection, an example of a scheduler with seven production rules is given. For simplicity, 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 only by a rule managers or actions in rules. The production rules are specified in Table 2.

For simplicity, 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 Fig.4. Circles in the figure mean concurrent processes, and lines between circles mean relations of a parent-process and child-processes. The order of scheduled events is expected as shown in Fig.5, 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.

6.3 An application of CCSPR

It is important to check whether the scheduler with the seven production rules behave as shown in Fig.5, or

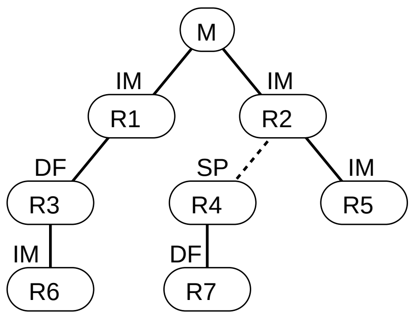

Fig. 4 The process concurrent tree of the example

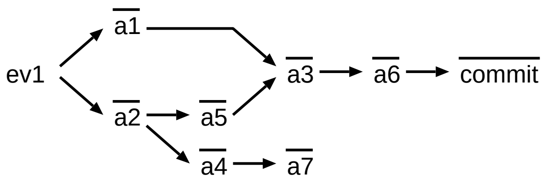

Fig. 5 The expected order of scheduled events

not. We apply CCSPR to this checking. We explain how to describe and analyze the scheduler in CCSPR. Each rule is described as follows:

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

$$\begin{aligned} &R1 \overset{\text{def}}{=} e1.Rule(A1, IM), && R5 \overset{\text{def}}{=} e4.Rule(A5, IM), \\ &R2 \overset{\text{def}}{=} e1.Rule(A2, IM), && R6 \overset{\text{def}}{=} e6.Rule(A6, IM), \\ &R3 \overset{\text{def}}{=} e3.Rule(A3, DF), && R7 \overset{\text{def}}{=} e7.Rule(A7, DF), \\ &R4 \overset{\text{def}}{=} e4.Rule(A4, SP) \end{aligned}$$

$Rule(X,Y)$ is a process expression with two process variables X and Y. A process substituted into X is controlled by a process substituted into Y through events s and d. An event $done$ means a termination of a process. In this case, a process for executing an action is substituted into X, and a process for a coupling mode is substituted into Y. X and Y are packed because both of them must be parent-processes of child-processes called by X. $A1$-$A7$ are processes for actions, and they are defined as follows:

$$\begin{aligned} &A1 \overset{\text{def}}{=} \overline{a1}.\overline{e3}.\overline{done}.\mathbf{0}, && A5 \overset{\text{def}}{=} \overline{a5}.\overline{done}.\mathbf{0}, \\ &A2 \overset{\text{def}}{=} \overline{a2}.\overline{e4}.\overline{done}.\mathbf{0}, && A6 \overset{\text{def}}{=} \overline{a6}.\overline{done}.\mathbf{0}, \\ &A3 \overset{\text{def}}{=} \overline{a3}.\overline{e6}.\overline{done}.\mathbf{0}, && A7 \overset{\text{def}}{=} \overline{a7}.\overline{done}.\mathbf{0} \\ &A4 \overset{\text{def}}{=} \overline{a4}.\overline{e7}.\overline{done}.\mathbf{0}, \end{aligned}$$

IM, SP, and DF are processes for coupling modes, **immediate**, **separate**, and **deferred**, respectively. Local communication between parent-processes and child-porcesses is important for coupling modes. Information of all processes in a process tree is gathered to a top-level-process and is distributed to them again, for example, in order to decide a staring point of an action with a **deferred** mode. IM, SP, and DF are defined as follows:

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

$\overline{s}$ and d are events to control a start point and an end point of an action execution. $\lfloor c\rfloor$ and $\lceil c\rceil$ are local-events to transmit to a parent-process that its child-process with an **immediate** mode is committed. $\lfloor sd\rfloor$ and $\lceil sd\rceil$ are used to start processes for actions with a **deferred** mode. $\lfloor cd\rfloor$ and $\lceil cd\rceil$ transmit to a parent-process that its child-process with a **deferred** mode is committed.

It is important to notice that IM, SP, and DF can be used for every rules without respect to the number of their child-processes. For example, if rules have no child-processes, then local-down-events of IM, SP, and DF are changed into internal events by Hidings. It is very difficult for existing process algebras to describe such independent coupling modes from the number.

Finally the system SYS of the scheduler is built as follows:

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

$$\begin{aligned} R &\equiv \{\!\{R1\}\!\} :: \{\!\{R2\}\!\} :: \{\!\{R3\}\!\} :: \{\!\{R4\}\!\} \\ &\quad :: \{\!\{R5\}\!\} :: \{\!\{R6\}\!\} :: \{\!\{R7\}\!\} \\ M &\stackrel{\text{def}}{=} ev1.\overline{e1}.\lceil c\rceil.\lceil sd\rceil.\lceil cd\rceil.\overline{com}.\overline{done}.\mathbf{0} \end{aligned}$$

M is a process for the rule manager.

Next, we describe expected oreder $ORDER$ in shown Fig.5. It is described in CCSPR (possibly in CCS) as follows:

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

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

$$\begin{aligned} ORDER \stackrel{\text{def}}{=} & \; ev1.(EA1[\!](EA2;(EA5[\!]\langle\!\langle EA4;EA7\rangle\!\rangle)) \\ & ;EA3;EA6;COM) \end{aligned}$$

where, ; and $[\!]$ are 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 Fig.5 or not, using CCSPR. We proved $SYS = ORDER$[†], using CWB[13] (Concurrency workbench) after we had expanded SYS into an expression including only Prefixes and Summations.

7. Related work

In the field of active databases, database production rules are analyzed in a *directed triggering graph*[3][4] and *petri nets*[6]. However these approaches do not consider coupling modes. We are interested in nested transactions with coupling modes, then we adopt 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.

In the field of process algebras, we have never met other research with the same purpose which is to exploit a specific process algebra for database production rules. Fixed process trees can be syntactically described in existing process algebras, but it may be impossible to syntactically describe growing process trees. We can describe growing process trees by using links such as process-id's or private links as shown in Section 2, but such links makes analysis to be hard.

8. 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 CCSPR, processes can be held in *resources*, and be created as child-processes when called out by a running process. For supplying child-processes, a new combinator $\triangleright$ called a Supplier has been introduced in this paper. It is the most important property of CCSPR that a Supplier can supply child-processes from resources to running processes, *penetrating* other combinators. This penetration makes syntactic expression of growing process trees be possible. Though the definition of the penetration is complex, users of CCSPR need not know the complex definition of the penetration in detail, and they should analyze growing process trees only by properties of the penetration, such as Proposition 5.1, 5.2, and 5.3. Consequently, CCSPR is appropriate to analysis of concurrent systems with the following features:

- Processes can call out new processes from resources,

[†]$=$ is observation congruence in CCS[7].

- Process trees can grow by process creation, and
- Multi-way local communications between a parent-process and child-processes are possible.

Active database systems with nested transactions have the above features.

Some propositions for proving strong equivalence have been given in Section 5, and have been used for an example in Section 6. 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] H.Ishikawa, "Active Databases", Journal of Information Processing Society of Japan, Vol.35, No.2, pp.120 - 129, 1994.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

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

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

[10] 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.

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

[12] D.J.Walker, "Bisimulations and Divergence", Proc. of third annual IEEE symposium on Logic in Computer Science, pp.186 - 192, 1988.

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

[14] Y.Isobe, I.Kojima, and K.Ohmaki, "A Study of Process Algebras for Active Databases", Information Processing Society of Japan SIG Notes, 94-DBS-99, pp.147 - 154, 1994.

[15] 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.

Yoshinao Isobe received his B.E. degree and the M.S. degree in Electrical Engineering from Shibaura Institute of Technology, Japan in 1990 and 1992 respectively. In 1992, Mr. Isobe joined Electrotechnical Laboratory (ETL), AIST, MITI. His research interests theoretical aspects for concurrency systems, especially process algebras. He is a member of JSSST and IPSJ.

Isao Kojima received the B.E. degree and the M.E. degree in information science from Kyoto University in 1982 and 1984,respectively. He is currently a Senior Reseacher at the Information Base Section of Electrotechnical Laboratory, MITI. His current research includes active databases, transaction systems and formal bases for database systems. He is a member of the ACM,the IEEE Computer Society and the IPSJ.

Kazuhito Ohmaki received his B.E. degree of Electrical Engineering from Iwate University, Iwate, Japan in 1974. He received the M.S. and Ph.D. degree of Information Science from Tohoku University, Sendai, Japan in 1976 and 1979, respectively. In 1979, Dr.Ohmaki joined Electrotechnical Laboratory (ETL) of the Agency of Industrial Science and Technology, the Ministry of International Trade and Industry. He was a visiting researcher of ETH (Swiss Federal Institute of Technology) for one year in 1985. Since 1992, he is a chief of Information Base Section, Computer Science Division, ETL. His research interests include theoretical aspects for software constructions. He is a member of ACM, IEEE, JSSST, IEICE, and IPSJ.