# The Search Ahead Conflict Resolution for Parallel Firing of Production Systems

Ching-Chi Hsu and Feng-Hsu Wang

Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, ROC

## Abstract

To explore the parallelism among rules is one of the ways to increase the speed of production systems. In this paper, an Object Pattern Matching model is proposed to interpret the dependency between production rules, and based on the rule dependency analysis, we develop a new conflict resolution principle, Search Ahead Conflict Resolution (SACR), which can select more than one rule to fire at each execution cycle. The so called Search Ahead Parallelism is thus achieveable by applying the SACR principle at run time. It is proved that the results of running programs in the new production system inference mechanism with SACR principle are the same as those running in the sequential inference mechanism with conflict resolution principle that is based on Last-In First-Out (LIFO) strategy. As a consequence, with the SACR principle, a program composed of a large set of production rules can run in a multiprocessor system and get the same result as those running in a uniprocessor system.

## 1. Introduction

Forward-chaining production systems are used extensively in artificial intelligence and expert system areas. Many tools for building production systems have been developed, such as OPS5 [Forg81] and ORBS lFick 85]. Especially, over the past several years, improvements in the OPS production systems from software approach have brought about substantial speed increase rForg79][Forg81][ScLe801[NcMa86]. The speed-up from OPS2 to OPS83 resulted from a number of factors, including changing important data structures, embedding special codes, compiling left-hand sides, etc. OPS83 is at least 120 to 180 times faster than OPS2, and further substantial improvements in software techniques have become difficult to achieve. So a hardware support for OPS5 interpreter is essential [FoWe84]. As OPS5 is widely used in constructing expert systems, it is taken as a tool to discuss the topics presented in this paper.

A parallel production system machine named DADO fStSh82][StSh83][StMi86] is another architectural approach trying to increase the speed of production systems. The processors in DADO are organized as a binary tree, and the full production rule level parallelism is explored by simultaneous matching of rulesn on the processors dedicated in the rule level. Many algorithms implemented on DADO to improve the speed of matching are proposed in [Stol84][Gupta84]. NON-VON [Hish86] is another example of production machine which is likely to DADO. In addition to the speed-up of rule matching, another speed-up we can obtain is through the parallel firing of production rules, which can reduce the total execution cycles of production systems, and the total speed-up may be the product of these two kinds of profit which will be quite significant.

Although nondetcrminism of production rule firing *is* the major characteristic in production system behaviour, we believe that by the static rule analysis of rule dependency, we can obtain some information that are useful for reducing search space, anticipating and composing rule firing sequence, and partitioning and mapping production rules. Many researchers have proposed various approaches to analyze the rule dependency. Tenorior and Modolvan [TeMo851 had adopted the graph grammar theory [Erhi78] to model the production system behaviour. There they considered the left hand side and right hand side of production rules as colored graphs, and the application of a production rule is nothing but a graph manipulation. Based on this theory, they had developed many useful interdependent relations between two rules. Graph grammar theory, however, needs to be extended to take into account the case of negative references in production rules.

Ishida and Stolfo [IsSt85] used data dependency graph to analyze the interdependency between rules. Many sufficient conditions are detected for synchronization problem - a problem of deciding which conflicting rules need to be synchronized. Conflicting rules without synchronization problem can be fired parallelly. They have considered the case of negative reference in rules and the production systems have the ability to delete WMEs in the RHS without binding them in the LHS. Both of the above two approaches are developed with the assumption that production system applications are written without considering any particular selection algorithms (ie, conflict resolution principles).

In this paper, an Object Pattern Matching model is proposed to interpret the dependency between production rules, and based on the rule dependency analysis, we develop a new conflict resolution principle , Search Ahead Conflict Resolution (SACR), which can select more than one rule to fire at each execution cycle. The so called Search Ahead Parallelism is thus achieveable by applying the SACR principle at run time. We also prove that the results of running programs in the new production system inference mechanism with SACR principle are the same as those running in the sequential inference mechanism with conflict resolution principle that is based on Last-In First-Out (LIFO) strategy. All the work needed to detect the parallelism (either at compile time or at run time) takes only polynomial time. And under the SACR resolution principle, the parallelism hidden in the production rules can be exploited as early as possible.

## 2. Rule Interference and Data Inconsistency

The major problem for the parallel execution of production systems is how to resolve the conflicting rules and choose the maximum number of rules to fire. It is the conflict resolution principle that determines the search scheme of the production systems. For example, the search scheme of OPS5 is LIFO search (LEX strategy), but it choose only one rule to fire at each cycle. Before we go into details of parallel firing of production rules, let us first examine the possible problems that might be caused by parallel firing of production rules.
For example, there arc two rules showed below :

```
(p ri                (p rj
   C1                   C1
   C2                   C3
   -->                  -->
  (modify 1 C4)        (remove 1)
  (make C5 ))          (make C6))
```

Assume that there are two rule instantiations $r_i$ and $r_j$ in the conflict set. In sequential inference scheme, no matter which rule is selected to fire, working memory C\ will always be removed, and thus causes the other rule instantiation to be removed from the conflict set. This is the effect of *rule interference.* In addition, rule interference may be indirect. For example, the firing of rule $r_j$ may trigger another rule $r_k$, and $r_k$ interferes rule $r_j$. This kind of rule interference is of level 2. Similarly, rule interference may be of level n. On the other way, if the two rule instantiations are selected to fire simultaneously, then when $r_i$ wants to modify $C_1$, $C_1$ may have been removed by $r_j$. This is the problem of data inconsistency.

Now, consider the following three rules $r_1, r_2, r_3$ :

```
(p r1            (p r2            (p r3
   C1               C2               C3
 -->              -->              -->
  (make C4)        (make C5)        (make C6)
 )                )                )
```

With C1,C2 and C3 are available, Fig 1 shown below is the search tree of this example. There are totally six sequential firing sequences in this search tree, but all these sequences get the same result (final working memory is (C1, C2, C3, C4, C5, C5)) and take three execution cycles. On the other way, if the three rules are fired simultaneously, then the result is also (C1, C2, C3, C4, C5, C6), taking only one cycle.

So, suppose that a rule firing sequence in sequential inference mechanism is ( $r_1$, $r_2$, $r_3$, .. , $r_n$), and the last working memory is WM. We develop a new conflict resolution which considers the interefrence relation and parallelism among the rules, such that when it is applied in production systems, the result is also WM but with less execution cycles (< n).

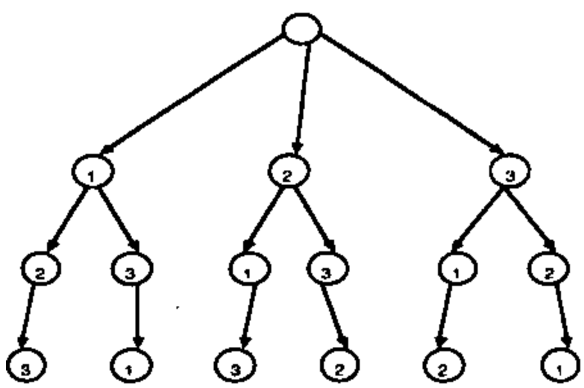



**Fig.1 search tree example**

## 3. Categories of rule dependency

At first, we define the following important terms and concepts which are essential for rule dependency analysis.

[definition II

An Object Pattern OP is an n-tuple with an object body. It has the following general format:

$OP(x_1, X2, .., x_n)$: object body

where OP is the calss-name of this object pattern, and $x_i$ is the value of the i'th attribute in OP, and $x_i$ must meet some constraint $C_i$.

**Object body** is an associated list composed of object calss and attribute-value pairs. It has the following format :

(object-class ^attr$_1$ *value$_1$* ^attr$_2$ *value$_2$* .. ^attr$_n$ *value$_n$*)

where *value$_i$* can be a constant or a variable with some constraint.

Let $P(x_1, x_2, .., x_n)$ be an object body, then we call $P(c_1, c_2, .., c_n)$ is an **object instance** of P, where $c_i$ is a constant and must meet the constraint of value$_i$. Its instance body is

(object-class ^attr$_1$ $c_1$ ^attr$_2$ $c_2$ .. ^attr$_n$ $c_n$)

**I(P)** denotes the set of all possible object instances of object pattern P.

Condition elements in the LHS or the actions in the RHS of a production rule can both be object patterns. The example below illustrates the terms listed in this definition.

**[Example]**

Here is an object pattern :

class1($x_1, x_2, x_3$) : (class1 ^atr1 3 ^atr2 { <x> <> 2 } ^atr3 {<> tag} )

where $x_1$ is a constant 3, $x_2$ is associated with a numeric constraint (numeric values not equal to 2 ), $x_3$ is associated with a symbol constraint (symbol values not equal to string "tag" ) .

class1(3,1,lin) is an object instance of class1($x_1, x_2, x_3$), whose instance body is :

(class1 ^atr1 3 ^atr2 1 ^atr3 lin )

[definition 2]

The precondition set $PRESET_i$ of a rule i is a set of marked object patterns, ie, the condition elements (positively/negatively referenced) in the left-hand side of rule i. If an object pattern Cj is positively referenced, then it will be marked as +Cj in PRESETi ; if negatively referenced, then marked as -Cj.

[definition 3]

The changed set $CHANGE_i$ of a rule i is a set of marked object patterns, ic, the added or deleted object patterns in the right-hand side of rule i. If an object pattern $W_i$ is added by a make action, then it will be marked as +Wj in CHANGEj ; if deleted by a remove or a modify action, then marked as -Wj.

**[definition 4]**

Rule $r_i$ has **interference dependency** on rule $r_j$ (abbreviated as **I-DEP**) iff the set formed by all the negated instances of the changed set of rule $r_i$ and the set formed by all the instances of the precondition set of rule $r_j$ are disjoint. That is,

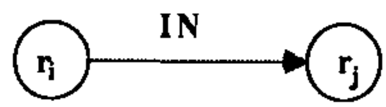
$$-I(CHANGE_i) \cap I(PRESET_j) \neq \phi ,$$

Let IN be the set formed as follows :

$$IN = \{ p \mid p \in PRESET_j, I(p) \cap -I(CHANGE_i) \neq \phi \}$$

where $-I(CHANGE_i)$ means to negate the elements in $I(CHANGE_i)$, ie, added(deleted) object pattern will become deleted(added).

If rule $r_i$ has interference dependency on rule $r_j$ w.r.p to the set IN, then in the i_dependency augmented digraph, there exists one subdigraph shown below :



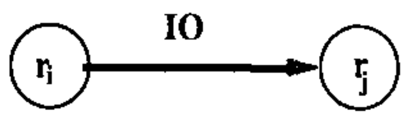

**[definition 5]**

Rule $r_i$ has **input-output dependency** on rule $r_j$ w.r.p to a set IO (abbreviated as IO-DEP) iff

$$I(CHANGE_i) \cap I(PRESET_j) \neq \phi ,$$

where $IO = \{ p \mid p \in PRESET_j, I(p) \cap I(CHANGE_i) \neq \phi \}$

If rule $r_i$ has input-output dependency on rule $r_j$ w.r.p IO, then in the io dependency augmented digraph, there exists one subdigraph shown below :




**[definition 6]**

The **Following Rule Set** of a rule r, **Follow(r)**, is the set of all rules that are reachable from rule r in the io dependency augmented digraph unioned with the rule r itself.

**[definition 7]**

If rule r ε Follow($r_i$), then we call rule r is a **descendent rule** of $r_i$.

**[Example]**

The contents of rule $r_i$ and $r_j$ are listed below :

```
(p   ri
     C1
     -C2
     -->
     (make  C3) )

(p   rj
     C1
     C3
     -->
     (make  C2)
     (remove  C1))
```

where $C_1$,$C_2$,$C_3$ are all object patterns listed below :

$C_1 (x_1)$ : (class1 ^atr1 3)
$C_2 (x_1)$ : (class2 ^atr1 <x>)
$C_3 (x_1)$ : (class3 ^atr3 <x>)

then

$PRESET_i = \{+C_1 , -C_2\}$
$PRESET_j = \{+C_1 , +C_3\}$
$CHANGE_i = \{+C_3\}$
$CHANGE_j = \{-C_1 , +C_2\}$

and because

$$I( CHANGE_i ) \cap I( PRESET_j ) = \{ I( C_3) \}$$

rule $r_i$ has IO-DEP on rule $r_j$ w.r.t { $C_3$ }. This implies that the firing of rule $r_i$ may trigger the match work of rule $r_j$ by making an object instance of $C_3$ (but not necessarily make the rule $r_j$ firable).

On the other way, because

$$\sim I(CHANGE_j) \cap I(PRESET_i) = \{ I(C_1) , -I(C_2) \}$$

rule $r_j$ has I-DEP on rule $r_i$ w.r.t { $C_1$ , $-C_2$ }. This implies that the firing of rule $r_j$ may cause the LHS of rule $r_i$ unsatisfiable, and thus interferes the firing of rule $r_i$.

Note that IO-DEP impiles an sequential order of execution, that is, if rule $r_1$ is IO-DEP on rule $r_2$, then the execution order may be ($r_1$,$r_2$).

For those rules that may be satisfied simultaneously, they need to be further analyzed to detect possible synchronization.

**[definition 8]**

Rule $r_i$ has **interference relation** on rule $r_j$ (abbreviated as **I-R**) if there exists at least one rule r, r ε Follow($r_i$), such that rule r has I-DEP on rule $r_j$ or rule $r_j$ has I-DEP on rule r.

If rule $r_i$ has I-R on rule $r_j$, this implies that the firing of rule $r_i$ may interfere the firing of rule $r_j$, or the firing of rule $r_j$ may interfere the firing of some descendent rule r of $r_i$.

**[Example]**

Suppose we have four rules listed below :

```
(p  r1
    :
    -->
    (make C2)
    (make C3))

(p  r2
    C2
    -->
    (make C4))

(p  r3
    C3
    -->
    (make C5))

(p  r4
    C3
    C4
    -->
    (remove 1))
```

Then by the previous definitions, we get a rule dependency graph shown below :

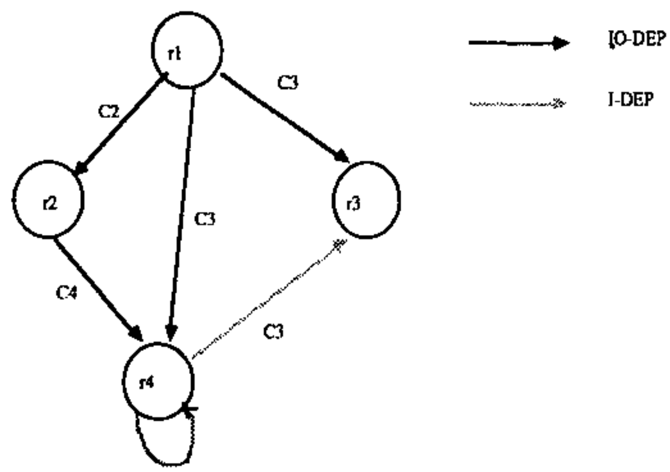



Fig 2. Example of rule dependency digraph

and the I-R matrix is shown below :

| | r1 | r2 | r3 | r4 |
|---|---|---|---|---|
| r1 | 0 | 0 | 1 | 1 |
| r2 | 0 | 0 | 1 | 0 |
| r3 | 0 | 0 | 0 | 1 |
| r4 | 0 | 0 | 1 | 1 |

Table 1. Example of I-R matrix

The entry in this matrix is 1 (eg. $I[r_i, r_j] = 1$) if rule $r_i$ has I-R on rule $r_j$ and is 0 if rule $r_i$ doesn't has I-R on rule $r_j$. For example, rule $r_2$ has I-R on rule $r_3$ (because in the following set of rule $r_2$, $r_4$ has I-DEP on $r_3$), so $I\_R\ [r_2, r_3] = 1$.

In this example, suppose we are running the above rules in a sequential inference engine. The firing of rule $r_1$ causes the rules $r_2$ and $r_3$ to be in the conflict set. At this time, if the inference engine selects the rule $r_2$ to fire, it will then triggers the firing of rule $r_4$, which will interferes the firing of $r_3$. In this case, the firing sequence is $r_1$->$r_2$->$r_4$. In another case, after firing $r_1$, the rule selected to fire may be $r_3$, then $r_2$ and finally $r_4$. The firing sequence is $r_1$->$r_3$->$r_2$->$r_4$. At run time, if the first case occurs, then $r_2$ and $r_3$ can not be fired simultaneously. If the second case occurs, r2 and r3 can be fired simultaneously. This is fully dependent on the selection principle of the sequential inference engine at run time. So, in the next session, we will propose a new conflict resolution principle that considers the characteristics of the sequential inference engine to predict the possible parallel firing at run time.

## 4. The SACR conflict resoltion principle

In this session, we will propose a new conflict resolution principle that can choose more than one rule to fire. The adventage of this new selection scheme is that for any sequential conflict resolution principle that is based on LIFO (Last-In First-Out) strategy, we can easily expand it to our new one to achieve the goal of parallel firing and get the same result as in the sequential case. For example, one of the conflict resolurion principles of OPS5 is LEX, which is based on the LIFO starategy. As a result, the programs written in original sequential language(eg. OPS5) can now be portable to the new system with SACR conflict resolution principle without further modification.

If the conflict resolution adopted by OPS5 is LEX, then the time recency (represented by time_tag) of the rule instantiations is the major factor when considering the selection of the rules in the conflict set. To fit into the parallel processing model, we first extend the concept of time_tag as follows :

time_tag : **major_time_tag . minor_time_tag**

where **major_time_tag:** an ordered number of a rule instantiation in the conflict set, which represents the time recency order of the rule instantiation.

**minor_time_tag:** an ordered number of a working memory element created in the RHS of aproduction rule.

For example, the time_tag of the first working memory element created by a rule whose major_time_tag is 3 will be 3.1.

We assume that initially there is only one rule in the conflict set, which starts the execution of the production system. Let the rule be denoted as INIT_RULE, then we assign 0 to be the major_time_tag assocated with the rule INIT_RULE.

Let the tim_tag, $x_i.y_i$ , be the time tag associated with the i'th positive condition element (CE) of rule instantiation r, then we define the **parent_time_tag** of rule instantiation r as P(r) = x.y, where

$x.y = \mathrm{Min}_{x_i} \{ x_i.y_i \}$ for all the i'th positive CE in the LHS of rule r.

That is, x.y is the time_tag with smallest major_time_tag component among all the time_tags of the positive CEs in the LHS of rule r.

We call rule instantiation $r_i$ has **data dependency** on $r_j$ if the parent_time_tag associated with $r_j$ is just the major_time_tag associated with rule $r_i$ . The data dependency means that the firing of rule $r_i$ makes $r_j$ firable immediately. Let $P(r_i)$ be the parent_time_tag associated with $r_i$ , $M(r_i)$ be the major_time_tag with $r_i$ . If $P(r_i) > P(r_j)$, then $M(r_i) > M(r_j)$. This implies that if the time recency of $r_i$ is more recent than that of $r_j$ , then the time recency of the descendent rules of $r_i$ is more recent than that of the descendent rules of $r_j$ . This, in terms of sequential search strategy, is the LIFO (or depth first) search strategy, in which the events with most recent time tag is searched first.

The basic concepts of our algorithm for conflict resolution to select parallel firable rules to fire such that the results of parallel firing is the same as the sequential case is : at each conflict resolution cycle, if $M(r_i) > M(r_j)$ , this means that in sequential firing, the firing of $r_i$ must be ahead of $r_j$ . So, if $r_i$ has no I_R relation on $r_j$ , they are parallel firable. The marking phase in the step 3 of our algorithm below is just for this purpose. Following is the outline algorithm of our new conflict resolution principle :

### Algorithm *SACR conflict resolution*

Step 1 : **current_major_time_tag** : the ordered number of rule instantiations up to now. CS is the current conflict set, suppose there are n rule instantiations in the conflict set, which are sorted in descending order by the value of the parent_time_tag if the rule instantiation is a newly created one, or by the value of the major time tag of the rule instantiation at the last cycle if the rule instantiation is a propagated one from last cycle.
/* all the rule instantiations in CS are not marked, but are divided into $CS_{l-1}$,.., $CS_2$, $CS_1$, $CS_0$ as descrived above, while the rule instantiations in $CS_i$ are sorted in descending order by the minor_time_tag . */
selected_rule_list = $\phi$ .

Step 2 : Let $r_0$ , $r_1$ , .... , $r_{n-1}$ denote the n rule instantiations sorted in CS, then assign (current_major_time_tag+n-i ) to be the major_time_tag of rule instantiation $r_i$ .

Step 3 :
```
FOR   i = 0 TO n-1 DO
  If ( ri is marked) then continue ;
  else
    FOR   j = i+1 TO n-1 DO
      If ( ri has I-R on rj ) them mark rj .
    ENDofFOR
  ENDofFOR
```

Step 4 : Select those rule instantiations not marked into the selected_rule_list.

Step 5 : current_major_time_tag = current_major_time_tag + n

To demonstrate our algorithm, let $r_1+r_2+r_3+r_4$ denotes the parallel firing of rule $r_1$,$r_2$, $r_3$ and $r_4$. In the example in section 3, after $r_1$ is fired, the contents of the conflict set will be { $r_3$ , $r_2$ }. By step 4 in this algorithm, the original selection principle of OPS5 will select $r_3$ to fire ( because the time_tag of $C_3$ is the most recent). At this time, by step 3, we know that $r_3$ doesn't has I-R on $r_2$, so $r_2$ is the next rule instantiation selected to fire. when $r_3$ and $r_2$ are fired parallelly, $r_4$ will be triggered to fire. So the total firing sequence is $r_1$ ->$r_3$+$r_2$ ->$r_4$ , taking only three cycles; while the original firing sequence is $r_1$ ->$r_3$ ->$r_2$ ->$r_4$, taking 4 cycles.

**[Definition 9]**

Let G0 be a set of object instantiations. We call $G0 \xrightarrow{r} G1$ is a **derivation step if**

for all p $\in$ $PRESET_r$ ,
if p is marked + , and there exist one element x of I(p) , such that x $\in$ $G_0$ , or
if p is marked - , and for all x in I(p) , x is not in $G_0$.

And, $G_1 = G_0 + A(r) - D(r)$

where

A(r) is the set of all object instances added by rule r, and
D(r) is the set of all object instances deletedby rule r.

And let ILHS(r) be the set of positive marked object instances, then D(r) is a subset of ILHS(r) .

And if

$$G_0 \xrightarrow{r_1} G_1 \xrightarrow{r_2} G_2 \xrightarrow{r_3} \dots \xrightarrow{r_n} G_n ,$$

We call $r_1 \to r_2 \to \dots \to r_n$ is a **derivation sequence**, which can be denoted by $r^*$, then

$$G_0 \overset{r^*}{\to} G_n.$$

**[Definition 10]**

We call two derivation sequences, $r_1{}^*$ and $r_2{}^*$ are **equivalent**, if

$$G_0 \overset{r_1^*}{\to} G_n,\ G_0 \overset{r_2^*}{\to} G_m \text{ and } G_n = G_m.$$

The relation is denoted as $r_1{}^* = r_2{}^*$.

**Theorem 1:**

**If rule b $\in$ Follow(a), then Follow(b) $\subseteq$ Follow(a).**

proof:

Because b $\in$ Follow(a), so there eixst in the io dependency augmented digraph a path from rule a to rule b, say path (a,b). And let $r_b \in$ Follow(b), then there eixst in the io dependency augmented digraph a path from rule b to rule $r_b$, say path (b, $r_b$).

So, there eixst in the io dependency augmented digraph a path from rule a to rule $r_b$, which is composed of path (a,b) and path (b, $r_b$). And we get $r_b \in$ Follow(a), **QED**.

**Theorem 2:**

**If rule $r_i$ has no I-R on $r_j$, then $\forall$ r $\in$ Follow($r_i$), r has no I-R on $r_j$ either.**

proof:

If $\exists$ r $\in$ Follow($r_i$), and r has I-R on $r_j$, then by definition 6, $\exists$ r' $\in$ Follow(r) such that r' has I-DEP on $r_j$ or $r_j$ has I-DEP on r'. But by theorem 1, Follow(r) $\subseteq$ Follow($r_i$), and r' $\in$ Follow($r_i$), then by definition 6, $r_i$ has I-R on $r_j$. This is a contradiction. QED.

**Theorem 3:**

**If rule $r_i$ has no I-R on $r_j$, and $r_i, r_j \in$ CS, then $r_i \to r_j = r_i + r_j = r_j \to r_i$.**

proof:

Because $r_i$ has no I-R on $r_j$, so the firing of $r_i$ will not interfere the firing of $r_j$ and the firing of $r_j$ will not interfere the firing of $r_i$ either.

let $G_0 \overset{r_i}{\to} G \overset{r_j}{\to} G_1$, because $r_i$, $r_j \in$ CS, so $r_i$ and $r_j$ is both firable in the context of $G_0$. Then,

$G = G_0 + A(r_i) - D(r_i)$,

and $D(r_i)$ and ILHS($r_j$) are disjoint, so $r_j$ is also firable in G.

By firing rule $r_j$, we get

$G_1 = G + A(r_j) - D(r_j)$,

$= G_0 + A(r_i) - D(r_i) + A(r_j) - D(r_j)$

$= G_0 + (A(r_i) + A(r_j)) - (D(r_i) + D(r_j))$

$= G_0 + A(r_i+r_j) - D(r_i+r_j)$

so $G_0 \overset{r_i+r_j}{\to} G_1$, that is ri ->rj = ri + rj.

By the same way, we can prove rj ->ri = ri + rj.

**QED.**

**Theorem 4:**

**The results of running programs in the new production system inference mechanism with *SACR* principle are the same as those running in the original sequential inference mechanism with LIFO strategy.**

proof:

We denote a parallel derivation step by ==>. Suppose the parallel derivation sequence is k steps, as shown below:

$P_1{}^* \Longrightarrow P_2{}^* \Longrightarrow P_3{}^* \Longrightarrow \dots \Longrightarrow P_k{}^*$

and the transition state of conflict set is shown as below:

$CS_0 \to CS_1 \to \dots \to CS_{k-1} \to \phi$

(we assume that halting of production system is due to the empty of conflict set).

If we can prove that the parallel derivation from $CS_0$ is equivalent to the sequential derivation from $CS_0$, then the theorem is proved. At first, let ${}^ir_k$ denotes the i'th rule instantiation selected at k'th cycle in the parallel derivation sequence ( ie, the rule instantiation whose major_time_tag is the i'th largest among the rule in the conflict set); ${}^ir_k{}^*$ denotes the possible sequential derivation sequence after the firing of rule ${}^ir_k$ by applying sequential conflict resolution principle, then

1). ${}^jr_k$ has no I-R on ${}^ir_k$, $\forall$ j=1 ,.., i-1. Clearly, we can see it from step 3 and 4 in the ***SACR conflict resolution*** algorithm.

2). ${}^jr_k{}^*$ has no I-R on ${}^ir_k$, $\forall$ j=1 ,...,i-1 . $\forall$ j=1 ,.., i-1, because ${}^jr_k$ has no I-R on ${}^ir_k$, by theorem 2, all the rule instantiations in ${}^jr_k{}^*$ have no I-R on ${}^ir_k$, that is, the firing of ${}^jr_k{}^*$ will not interefre the firing of ${}^ir_k$.

3). If the major_time_tag of ${}^ir_k$ is larger than that of ${}^jr_k$, then the parent_time_tag of ${}^ir_k$ is larger than or equal to that of ${}^jr_k$. In this algorithm the rule instantiation with higher parent_time_tag will be preferred (step 4 in the algorithm) and assigned higher major_time_tag (step 2 in the algorithm), so the parent_time_tag of ${}^ir_k$ is larger than or equal to that of ${}^jr_k$ if the major_time_tag of ${}^ir_k$ is larger than that of ${}^jr_k$.

4). If ${}^ir_k > {}^jr_k$ ( in relation with major_time_tag), then ${}^jr_k$ has no data dependency (by definition of data dependency) on every rule instance r in ${}^ir_k{}^*$. In other words, rule instantiation r is not added to the conflict set by the firing of ${}^jr_k$.

5). k1 < k2, and ${}^ir_{k2}$ has no I-R on ${}^jr_{k1}$ and ${}^jr_{k1}$ has no data dependency on ${}^ir_{k2}$, then ${}^jr_{k1} \to {}^ir_{k2} = {}^ir_{k2} \to {}^jr_{k1}$. This is because ${}^jr_{k1}$ has no data dependency on ${}^ir_{k2}$. By defnition, ${}^jr_{k1}$ doesn't trigger the firing of ${}^ir_{k2}$, so if (${}^jr_{k1}$, ${}^ir_{k2}$) is in CS, and ${}^ir_{k2}$ has no I-R on ${}^jr_{k1}$, by theorem 2, ${}^jr_{k1} \to {}^ir_{k2} = {}^ir_{k2} \to {}^jr_{k1}$.

6). Because the sequential inference mechanism is LIFO strategy, the rule instantiation with higher major_time_tag will be executed before those with lower major_time_tag, then,

$${}^1r_k + {}^2r_k + \dots + {}^nr_k = {}^1r_k \to {}^2r_k \to \dots \to {}^nr_k.$$

Now we prove the theorem by induction on cycle numbers. First, without loss of generality, we can assume that the total execution cycles of a program running in the parallel version of production system with SACR conflict resolution principle is k.

a). for cycle number = k, by 6).

$P_k{}^* = {}^1r_k + {}^2r_k + \dots + {}^{nk}r_k = {}^1r_k \to {}^2r_k \to \dots \to {}^{nk}r_k$

clearly, the theorem is right.

b). for cycle number n = k-1, we want to prove the parallel deriviation step $P_{k-1}{}^* \Longrightarrow P_k{}^*$ is equivilant to the sequential deriviation steps from cycle k-1 to cycle k:

$P_{k-1}{}^* \Longrightarrow P_k{}^*$ (1)

$= {}^1r_{k-1} + {}^2r_{k-1} + \dots + {}^{nk-1}r_{k-1} \Longrightarrow P_k{}^*$

$= {}^1r_{k-1} + {}^2r_{k-1} + \dots + {}^{nk-1}r_{k-1} \to {}^1r_k \to {}^2r_k \to \dots \to {}^{nk}r_k$

By 6), we get (1) is equal to:

${}^1r_{k-1} \to {}^2r_{k-1} \to \dots \to {}^{nk-1}r_{k-1} \to {}^1r_k \to {}^2r_k \to \dots \to {}^{nk}r_k$

Let ${}^ir_{k-1}$ be the rule that has data dependency with ${}^1r_k$,

if ${}^ir_{k-1}$ exists, then by 4). , 5), we get (1) is equal to:

${}^1r_{k-1} \to \dots \to {}^ir_{k-1} \to {}^1r_k \to \dots \to {}^{nk-1}r_{k-1} \to {}^2r_k \to \dots \to {}^{nk}r_k$

if ${}^ir_{k-1}$ not exist, then ${}^1r_k$ must be a rule instance that is passed from $CS_{k-1}$, so there exist ${}^ir_{k-1}$ and ${}^jr_{k-1}$ such that ${}^ir_{k-1} > {}^1r_k > {}^jr_{k-1}$ (in relation of major_time_tag), so (1) will become:

${}^1r_{k-1} \to \dots \to {}^ir_{k-1} \to {}^1r_k \to {}^jr_{k-1} \to \dots \to {}^{nk-1}r_{k-1} \to {}^2r_k \to \dots \to {}^{nk}r_k$

And repeating the same process on ${}^{2}r_k \ldots {}^{nk}r_k$, (1) will be ${}^{1}r_{k-1} \to {}^{1}r_{k-1}^{*} \to {}^{i1}r_k \to {}^{2}r_{k-1} \to {}^{2}r_{k-1}^{*} \to {}^{i2}r_k \ldots \to {}^{nk-1}r_{k-1} \to {}^{nk-1}r_{k-1}^{*}$
where ${}^{i1}r_k$, ${}^{i2}r_k$ ... ,denote the rule instantiations passed from $CS_{k-1}$ to $CS_k$ and were selected to be fired by the SACR conflict resolution principle.

c). By the same way as b), we can show that for n=k-2,k-3, ... ,1, the parallel deriviation step $P_{n^*} \Longrightarrow P_k^*$ is equivilant to the sequential deriviation steps from cycle n to cycle k. **QED .**

Fig 4.a and fig 4.b are the firing sequences of a test program running in the sequential OPS5 and in the SACR conflict resolution algorithm respectively. The program contains 23 rules and in nature has some degrees of parallelism, which can be shown in the sequential firing graph in fig 4.a. In that figure, there are two cycles at which parallel firing might occur, one is at cycle 3 and the other is at cycle 7. The sequential firing sequence takes 14 steps, but as shown in fig 4.b, the parallel firing sequence takes only 10 steps and the fully parallelism is exploited in this example. From this example, we can make some comments on our conflict resolution algorithm :

1. Detect parallelism at run time. If the programs have high degree of parallelism in nature, the parallelism can be detected as early as possible. In worse case, except that the programs are sequential in nature, the parallelism will sooner or later be detected with a little cycle delay, which depends strongly on the nature of the program structure.

2. The information needed for parallelism (ie, the I_R table) can be obtained in polynomial time computation at compile time.

3. The selection phase at run time takes only polynomial time order in size of the conflict set.

## 5. Conclusions

In this paper, we form the concept of a new kind of parallelism, named search ahead parallelism which is different from AND and OR parallelism, and can be exploited to reduce the execution cycles of production systems. To exploit this kind of parallelism, we need a data dependency analysis of the production rules at compile time to detect the synchronization condition among rules, and we develop a new conflict resolution principle that takes the result of data dependency analysis into consideration to anticipate the rules that can fire parallely at run time. All the work needed (either at compile time or at run time) takes only polynomial time, so the efficiency is acceptable. And the test program shows that a full exploitation of the run time parallelism might be achieved.

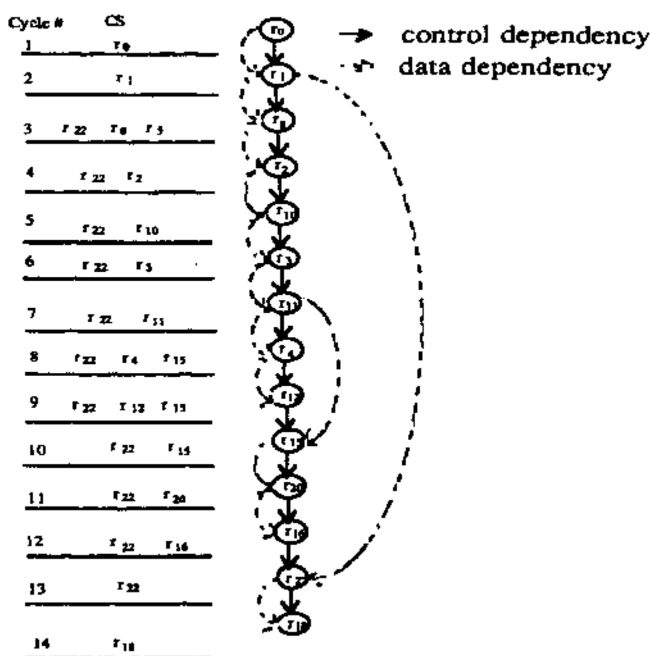


Fig 4.a sequential firing sequence

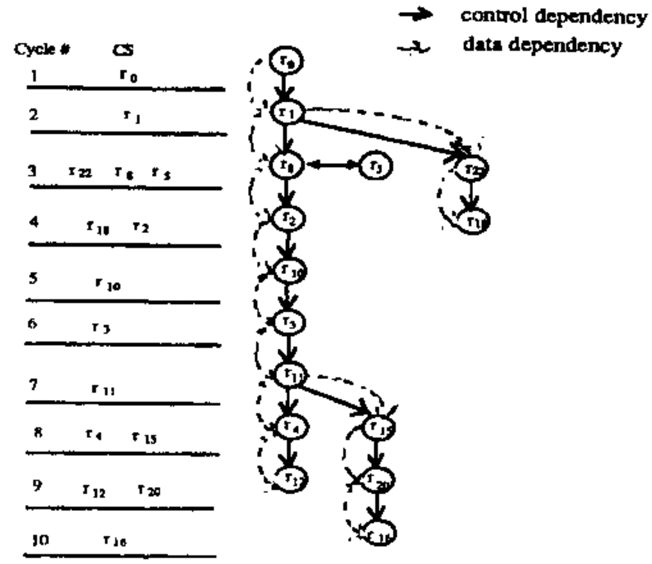


Fig 4 .b parallel firing sequence with parallel

## References


[Erhi78] Erhig, H. " Introduction of Algebraic Theory of Graph Grammars ", proc. Int. Workshop on Graph Grammars and their Application to Computer Science, and Biol., Bad rlonnef, 1978.

[Fick85] S. Fickas. " Design Issues in a Rule-Based System ", ACM SIGPLAN, vol. 20, pp.208-215, 1985.

[Forg79] C.L. Forgy," On the Efficient Implementation of Production Systems ", Ph.D Thesis, Carneige-Mellon University, 1979.

(Forg81J C.L. Forgy, " OPS5 user's manual ", Dept. of CS, CMU Technical Report CMU-CS-81-135,1981.

(Porg82] C.L. Forgy, " RETE, A Fast Algorithm For the Many Pattern/Many Object Pattern Problem ", Artificial Intelligence, vol. 19, ppl7-37, 1982.

[FoWe841 C. Forgy, A.Gupta, A. Newell & R.Weding. " Initial Assessment of Architectures for Production Systems ", proc. of the National Conference on Artificial Intelligence, Aug, 1984, pp. 116-120.

[Gupt84] Gupta, A. " Implementing OPS5 production systems on DADO ", proc. 1984 International Conference on Parallel Processing, Aug. 21-24, 1984, pp. 83-91.

[HiSh86] Bruce K.Hillyer and David Elliot Shaw," Execution of OPS5 Production Systems on a massively Parallel Machine ", Journal of Parallel and Distributed Computing 3, pp. 236-268, 1986.

[NeMa86J Neiman, D. and Martin, J., " Rule-Based Programming in OPS83 " AI Expert, Permier, 1986, pp.54-65.

[ScLe86] Schor, M.I, Daly, T.P. , Lee, H.S. " Advances in Rete Pattern Matching ", proc. of AAAI, pp.226-234, Philadelphia, 1986.

(StMi86] Salvators J.Stoflo & Daniel Miranker " The DADO Production System Machine ", Journal of Parallel and Distributed Computing, 3, pp. 269-296, 1986.

[Stol84] Salvators J.Stoflo," Five Parallel Algorithms for Production System Execution on the DADO Machine ", AAAI-84.

[StSh82] Salvators J.Stoflo & David Elliot Shaw, " DADO: A Tree-Structured Machine Architecture for Production systems ", AAAI 82.

[StSh83] Salvators J.Stoflo & David Elliot Shaw, " Architecture and Applications of DADO : a Large-Scale Parallel Computer for Artificial Intelligence ", IJCAI 83.

[TeMo85] M. F.M Tenorior and D.I. Modolvan " Mapping Production Systems into Multiprocessors ", International Conference on Parallel Processing, 1985, IEEE.

IlsSt85] Toru Ishida & Salvatore J.Stolfo, " Towards the Parallel Executition of Rules in Production System Programs ", International Conference on Parallel Processing, IEEE, 1985.