Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

# Termination Analysis of Active Rules - A Petri Net Based Approach

Lorena Chavarría-Báez
Programming and System Development Department
Superior School of Computing,
National Polytechnic Institute
Mexico, D. F.
lchavarria@computacion.cs.cinvestav.mx

Xiaoou Li
Computer Science Department
CINVESTAV-IPN
Mexico, D. F.
lixo@cs.cinvestav.mx

***Abstract*— Active rules allow software systems behave automatically when relevant events take place. Due to unstructured rule processing, it is necessary to inspect behavior characteristics such as termination which guarantees that rule processing finishes.**

**In this paper we introduce potential termination concept which gives valuable information about those rules whose processing may not terminate during execution time. It is very useful to manage possible bad scenarios. We also describe our Petri net-based approach to effectively detect termination and potential termination problems.**

**Keywords: Active rule, termination, potential termination, CCPN.**

## I. Introduction

Active rules allow software systems to specify actions to be performed automatically when certain relevant events take place and some conditions are met. Several applications, such as smart homes [7], sensor [5] and active databases [4], [6], integrate active rules for the management of some of their important activities.

An active rule consists of three parts: an *Event*, a *Condition* and an *Action*, so it is also called an *ECA rule*. The occurrence of an event can generate many rules may fire and the execution of them may cause other rules to be firable. This rule execution process is performed non-deterministically and, as result, it is quite difficult (even for a small rule set) to predict the rule set behavior. Two desirable properties of active rule behavior are *termination* and *confluence*. A rule set is guaranteed to *terminate* if rule execution processing cannot continue indefinitely. A rule set is *confluent* if for any initial system state the final state is independent of the rule execution order.

Several works have been proposed in the literature to analyze static properties of active rules, e.g. termination and confluence. Some approaches tackled the problem based on triggering and execution graphs analysis [8], [9]. A triggering graph is a directed graph whose nodes represent the rules and whose edges indicate that a rule produces an event that may trigger another rule. If the graph is acyclic, rule execution terminates. In execution graphs, nodes represent the state and directed edges are labeled with the name of the rule whose execution makes the system switch from one state to another. If the graph is acyclic and every pair of rules commute, the rule execution process is confluent. In order to describe termination completely information of both graphs has to be used. In reference [6] the authors propagate the effect of the action part of a condition - action rule to the condition part of another rule to determine if the arcs of the triggering graph have to be included in the graph and to verify if two rules commute. In some cases, the proposed propagation algorithm may produce incorrect results. In reference [1] the authors translate a set of active rules into logical clauses and then apply results about termination and confluence available in the literature for deductive rules to conclude about fulfilment of those properties in active rules. In reference [2], confluence is investigated by using a rewriting technique. A user-defined transaction is translated by means of active rules into an induced one(s) and then they check their equivalence. This work doesn't address termination issue.

In this paper we present an approach for analyzing termination in an active rule base (ARB). First, we introduce traditional termination conception, additionally we introduce for the first time the concept *potential termination* which represents situations in which rule processing may terminate or not. Second, we develop a Petri net-based method for detecting termination and potential termination in an ARB. Our method is performed in three steps: "Rule Normalization" is done in order to simplify later analysis. "Rule Modeling" is executed to represent the ARB as a Conditional Colored Petri Net (CCPN) structure. "Termination Analysis" is carried out to identify structures that may have termination problems. Since rule type transitions in CCPN model stand for rules, we actually obtain the rules that may exhibit abnormal behavior and termination is investigated by analyzing their interaction. The main contributions of this paper are the introduction of potential termination concept and the development of CCPN-based method to detect both termination and potential termination.

## II. Active Rules

An active rule has both knowledge and execution models. The general form of an active rule is the following:

**ON** *event*
**IF** *condition*

**THEN** *action*

An *event* is something that occurs at a point in time. It can be of two types: *primitive* or *composite*. An event is primitive when it cannot be decomposed in smaller events, for example, a transaction in database operation. An event is composite when it is defined by some combination of primitive or composite events using a range of operators that constitute the event algebra. In this paper we will consider disjunction -OR($E_1$, $E_2$) and conjunction -AND($E_1$, $E_2$) operators. The *condition* examines the context in which the event has taken place. The *action* describes the task to be carried out by the rule if the condition is fulfilled once an event has taken place.

In the following there are some active rules which were taken from [6]. They are part of an active database system (it consists of a traditional database and an ARB). The relation scheme account(*num, name, balance, rate*) is part of the database, it contains tuples about bank's accounts. Rules in the ARB established on relation scheme account automatically enforce some of the bank's policies for managing customers' accounts. Below there is an informal description of the rules.

*Example 1:* Bank's policies for managing customers' accounts.

**Policy 1**: When an new account is registered, if that account has a balance less than 500 and an interest rate greater than 0%, then that account's interest rate is lowered to 0%.

**Policy 2**: When an account's interest rate is modified, if an account has a balance less than 500 and an interest rate greater than 0%, then that account's interest rate is lowered to 0%.

**Policy 3**: When an account's balance is modified, if that account has an interest rate greater than 1% but less than 3%, then that account's interest rate is raised to 2%.

**Policy 4**: When an account's balance is modified, if that account has an interest rate greater than 1% but less than 3%, then that account is deleted.

Above policies are represented as active rules as follows:

**Rule 1**
ON insert account
IF insert.*balance* $<$ 500 and insert.*rate* $>$0
THEN update account set *rate* = 0
    where *balance* $<$ 500 and *rate* $>$ 0

**Rule 2**
ON update account.*rate*
IF update.*balance* $<$ 500 and update.*rate* $>$0
THEN update account set *rate* = 0
    where *balance* $<$ 500 and *rate* $>$ 0

**Rule 3**
ON update account.*balance*
IF update.*rate* $>$1 and update.*rate* $<$ 3
THEN update account set *rate* = 2
    where *rate* $>$ 1 and *rate* $<$ 3

**Rule 4**
ON update account.*balance*
IF update.*rate* $>$1 and update.*rate* $<$ 3
THEN delete from account
    where *rate* $>$ 1 and *rate* $<$ 3

The execution model of an active rule specifies how a rule base is treated at run-time. There are several phases during rule execution that require special attention:

1) *Signaling* phase refers to the appearance of an event occurrence caused by an event source.
2) *Triggering* phase takes the events produced thus far, and triggers the corresponding rules.
3) *Evaluation* phase evaluates the condition of the triggered rules. The rule conflict set is formed from all rules whose condition is satisfied.
4) *Scheduling* phase indicates how the rule conflict set is processed.
5) *Execution* phase carries out the actions of the chosen rules. During the action execution other events can in turn be signaled that may produce a cascade rule firing.

Let's show ECA rule execution process with rule base of Example 1. Suppose the event "update account.*balance*" has been detected (*signaling phase*), then **Rule 3** and **Rule 4** are triggered (*triggering phase*). After the event has been detected, conditions of **Rule 3** and **Rule 4** must be evaluated in order to determine if their corresponding actions can be executed (*evaluation phase*). Let's assume the condition update.*rate* $>$ 1 and update.*rate* $<$ 3 is true, then it is necessary to schedule the rule execution (*scheduling phase*). For simplicity, rule execution will be done by following the order of rules in the list. Therefore, **Rule 3**'s action will be executed first, only then **Rule 4**'s action will be executed (*execution phase*). After **Rule 3**'s action execution, the event "update account.*rate*" is signaled, so **Rule 2** is triggered and execution process is repeated until there is no rule eligible to trigger, this is a rule execution chain. On the other hand, when **Rule 4**'s action is executed, rule processing finishes since there is no rule such that is triggered by the event "delete from account".

During rule execution rules interact in different ways, they can *trigger*, *activate/deactivate* and *commute* each other. A rule *triggers* another rule if the action of the former generates the event that the latter is overseeing. For example, in above rule execution process, after execution phase finishes, **Rule 3** triggers **Rule 2** because **Rule 3**' action coincides with **Rule 2**'s event. A rule *activates* (*deactivates*) another rule when the condition of the last one is true (false) after the action execution of the former rule. **Rule 3** activates **Rule 2** because when **Rule 3** executes its action it makes **Rule 2**'s condition true. On the other hand, **Rule 4** deactivates **Rule 3** since the former takes away data which satisfy **Rule 3**'s condition. Two rules *commute* each other if they don't activate/deactivate each other and their actions commute. Since **Rule 4** deactivates **Rule 3**, those rules cannot commute because of different final results are obtained when **Rule 4** is executed before **Rule 3** and vice versa.

Rule interaction is an important issue in inspecting relevant characteristics of rule behavior such as termination which verifies if rule processing is guaranteed to finish.

For simplicity, from now onwards we denote an active rule as $r(e, c, a)$ where $e, c$ and $a$ are the event, condition and action of rule $r$, respectively. An ARB $R$ is formed by a set of active rules $r_1, r_2, ..., r_n$, i. e., $R = \{r_1, r_2, ..., r_n\}$ where $r_i$, $i = 1, ..., n$ has the described form $r_i(e_i, c_i, a_i)$.

## III. Termination Analysis

Active rules' behavior can be modeled by many graphical representations such as triggering [10], activation [11] and evolution [12] graphs. However, modeling characteristics of Petri nets, particularly Conditional Colored Petri Nets (CCPN) [3], make them a suitable tool for representing both rule behavior and rule elements cooperation.

### A. Definitions

Termination problems arise when in an ARB there exists *circular rules*.

*Definition 1:* **Circular rules.** If in R, there exists a triggering rule sequence $r_i$, $r_j$, ..., $r_p$, $r_q$ where $r_i$ triggers and activates $r_j$ and so on, and $r_q$ triggers and activates $r_i$, then rules $r_i, r_j, ..., r_p, r_q$ are circular rules. A special case is when rule $r_i$ triggers and activates itself indefinitely.

In ARB of Example 1, although **Rule 2** triggers itself it doesn't activate itself, since its condition becomes false after its action is performed. So, it is not a circular rule and its processing is guaranteed to terminate.

Sometimes it is not possible to accurately determine if circular rules processing will finish. Then we propose the concept *potential termination*. We identified two circumstances under which circular rule processing may not terminate.

*Definition 2:* **Potential termination.** Given a triggering rule sequence $S$, its processing may not terminate in the following cases:

1) When it is not possible to determine if $r_i$ activates/deactivates $r_{i+1}$.
2) When $r_i$ "pseudo-triggers" $r_{i+1}$.

In the first case, rule processing will not finish until we have a system state in which $c_i$ becomes false. Let's modify **Rule 2**'s condition of above Example 1 as follows:

**Rule 2'**
ON update account.*rate*
IF update.*balance* $<$ 500
THEN update account set *rate* = 0
    where *balance* $<$ 500

It is clear that action execution of **Rule 2'** neither activates nor deactivates its condition. So, once its processing has started, rule processing is not going to finish until there is a new database state in which attribute *balance* takes a value greater than 500.

In the second case, we need to define *pseudo-triggering* relation.

*Definition 3:* A rule $r_i$ pseudo-triggers $r_{i+1}$ if $a_i \subset e_{i+1}$, i.e., $r_i$ partially produces the needed events to trigger $r_{i+1}$.

Let's consider the following rule **Rule 2"**:

**Rule 2"**
ON AND(update account.*rate,* update account.*balance*)
IF update.*balance* $<$ 500 and update.*rate* $>$0
THEN update account set *rate* = 2
    where *balance* $<$ 500 and *rate* $>$ 0

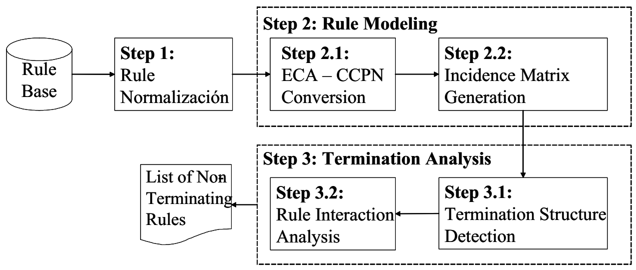

Rule Base
Step 1: Rule Normalización
Step 2: Rule Modeling
Step 2.1: ECA – CCPN Conversion
Step 2.2: Incidence Matrix Generation
Step 3: Termination Analysis
Step 3.1: Termination Structure Detection
Step 3.2: Rule Interaction Analysis
List of Non Terminating Rules

Fig. 1. CCPN based termination analysis approach

In this case, **Rule 2"** pseudo-triggers itself since its action partially produces the needed events to trigger itself.

Termination in a pseudo-triggering relation depends on the events flow. For example, in **Rule 2"** if event "update account.*balance*" raises only once then rule processing finishes.

Potential termination needs to be supervised to avoid infinite rule processing during execution time.

### B. CCPN-Based Analysis Approach

Our analysis is a CCPN-based approach which consists of three steps: *Rule Normalization*, *Rule Modeling*, and *Termination Analysis*. Figure 1 shows the order in which they are performed.

Rule Normalization step takes the original rule base and transforms each rule into an atomic one. An atomic rule is that whose event and condition are a conjunction of one or more primitive events and conditional clauses, and its action is only one instruction. This stage is done in order to simplify the later analysis. In the second step, the atomic rule base is represented by a CCPN structure and its corresponding incidence matrix. This representation enables us, in the third step, to detect termination structures and to extract the rules which may not fulfill that property. Finally, interaction among extracted rules is analyzed in order to draw a final conclusion.

*1) Step 1: Rule Normalization:* Active rules can be classified into the following four types according to event - condition combination pattern.

**Rule type 1.** *Conjunction operator in premise and action.*
ON AND $(e_1, e_2, ..., e_n)$
IF AND$(c_1, c_2, \cdots, c_m)$
THEN AND$(a_1, a_2, ..., a_k)$

**Rule type 2.** *Conjunction operator in both event and action, and disjunction operator in condition.*
ON AND$(e_1, e_2, ..., e_n)$
IF OR$(c_1, c_2, \cdots, c_m)$
THEN AND$(a_1, a_2, ..., a_k)$

**Rule type 3.** *Disjunction operator in event and conjunction operator in both condition and action.*
ON OR $(e_1, e_2, ..., e_n)$
IF AND$(c_1, c_2, \cdots, c_m)$
THEN AND$(a_1, a_2, ..., a_k)$

**Rule type 4.** *Disjunction operator in premise and conjunction operator in action*

ON OR$(e_1, e_2, ..., e_n)$
IF OR$(c_1, c_2, \cdots, c_m)$
THEN AND$(a_1, a_2, ..., a_k)$

A rule $r_i(e_i, c_i, a_i)$ can always be divided into several atomic rules by the following steps:

**Step 1**. If $r_i(e_i, c_i, a_i)$ is atomic, it´s OK. If not, go to Step 2.

**Step 2** Transform each element of $r_i(e_i, c_i, a_i)$ into disjunction form, by using rules from the boolean algebra, so that each element consists of one or more disjuncts each one of them is a conjunction of one or more instructions. If the transformed rule has no disjunctions in its elements, then it is an atomic rule according to the definition. Otherwise, go to the next step.

**Step 3** Divide $r_i(e_i, c_i, a_i)$ into a set of atomic rules whose premises and actions are the obtained disjuncts in Step 2.

The normalizations of above four rule types are:

**Normalization 1.** Since rules type 1 has no disjuncts in its premises, its normalization can be done directly by executing Step 3.

| **Rule 1.1** | | **Rule 1.n** |
|---|---|---|
| ON AND$(e_1, e_2, ..., e_n)$ | $\cdots$ | ON AND$(e_1, e_2, ..., e_n)$ |
| IF AND$(c_1, c_2, ..., c_m)$ | | IF AND$(c_1, c_2, ..., c_m)$ |
| THEN $a_1$ | | THEN $a_k$ |

For rule types 2, 3 and 4, we consider a simple case in which rules performs only one action. For conjunctive actions, we only need one step more to divide the conjunctive action like Normalization 1.

**Normalization 2.** Rules type 2 can be normalized into a set of rules with the following form:

| **Rule 1.1** | | **Rule 1.m** |
|---|---|---|
| ON AND$(e_1, e_2, ..., e_n)$ | $\cdots$ | ON AND$(e_1, e_2, ..., e_n)$ |
| IF $c_1$ | | IF $c_m$ |
| THEN $a_1$ | | THEN $a_1$ |

**Normalization 3.** Rules type 3 are normalized into rules with the following form:

| **Rule 1.1** | | **Rule 1.n** |
|---|---|---|
| ON $e_1$ | $\cdots$ | ON $e_n$ |
| IF AND$(c_1, c_2, ..., c_m)$ | | IF AND$(c_1, c_2, ..., c_m)$ |
| THEN $a_1$ | | THEN $a_1$ |

**Normalization 4.** Rules type 4 can be normalized as follows:

| **Rule 1.1** | **Rule 1.2** | | **Rule 1.n** |
|---|---|---|---|
| ON $e_1$ | ON $e_1$ | $\cdots$ | ON $e_1$ |
| IF $c_1$ | IF $c_2$ | | IF $c_m$ |
| THEN $a_1$ | THEN $a_1$ | | THEN $a_1$ |
| **Rule 2.1** | **Rule 2.2** | | **Rule 2.n** |
| ON $e_2$ | ON $e_2$ | $\cdots$ | ON $e_2$ |
| IF $c_1$ | IF $c_2$ | | IF $c_m$ |
| THEN $a_1$ | THEN $a_1$ | | THEN $a_1$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| **Rule n.1** | **Rule n.2** | | **Rule n.m** |
| ON $e_n$ | ON $e_n$ | $\cdots$ | ON $e_n$ |
| IF $c_1$ | IF $c_2$ | | IF $c_m$ |
| THEN $a_1$ | THEN $a_1$ | | THEN $a_1$ |

After rule normalization the number of rules may increase but they don't affect the behavior of the rule base.

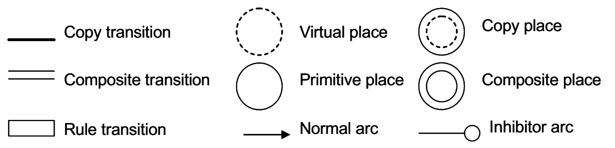


Fig. 2. List of CCPN elements

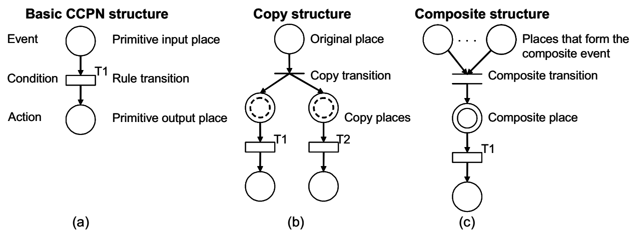


Fig. 3. Basic CCPN structures of an active rule

*2) Step 2: Rule Modeling:* An ARB as well as its interaction can be represented by a CCPN [3]. Figure 2 shows the basic graphical elements of CCPN.

*a) Step 2.1: ECA - CCPN Conversion:* In CCPN, an active rule is mapped to a transition where its condition is attached, event and action parts are mapped to input and output places of the transition, respectively (see Figure 3(a)). Matching between events and input places has the following characteristics:

1) *Primitive* places, represent primitive events.
2) *Composite* places, represent composite events such as AND.
3) *Copy* places, are used when one event triggers two or more rules. A copy place takes the same information as its original one.
4) *Virtual* places are used to represent the composite event OR. Since after Rule Normalization there are only event conjunctions we don't elaborate on this.

Rules and transitions are related in the following form:

1) *Rule* transitions, represent rules.
2) *Composite* transitions, represent composite event generation.
3) *Copy* transitions, duplicate one event for each triggered rule.

Whenever an event triggers two or more rules it has to be duplicated by means the copy structure depicted in Figure 3(b). Composite events formation is considered in CCPN using the composite structure drawn in Figure 3(c). Composite transition's input places represent all the events needed to form a composite event while its output place correspond to the whole composite event. The CCPN model of a set of ECA rules is formed by connecting those places that represent both the action of one rule and the event of another rule.

*b) Step 2.2: Incidence matrix generation:* Incidence matrix shows flow relations between places and transitions in CCPN. Rows of the matrix represent transitions and columns

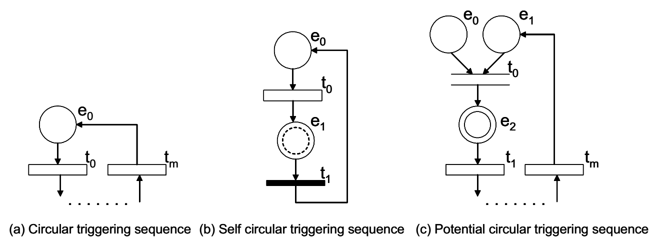

e0
t0
tm
e1
t1
e2
(a) Circular triggering sequence
(b) Self circular triggering sequence
(c) Potential circular triggering sequence

Fig. 4. CCPN structures for (potential) circular triggering sequences

represent places. The absolute value of an element $A_{i,j}$ represents the weight of the arc that connects the transition $i$ with the place $j$. If the value of $A_{i,j}$ is zero it means that there is not connection between the transition $i$ and the place $j$. An entry $A_{i,j}$ with negative value means that place $j$ is an input place for the transition $i$. An entry $A_{i,j}$ with positive value means that place $j$ is an output place for the transition $i$.

*3) Step 3: Termination Analysis:* It is performed in two steps: first, we find all the loops in CCPN. Second, we inspect rule interaction among rules involved in loops.

*a) Step 3.1: Termination Structure Detection:* In CCPN termination is depicted by (potential) circular triggering sequences. A triggering sequence is defined below:

*Definition 4:* **Triggering sequence.** A triggering sequence, S, is a finite path of the form: $(p_1, t_1) \rightarrow (t_1, p_2) \rightarrow (p_2, t_3) \rightarrow \ldots \rightarrow (t_{m-1}, p_m)$ where:

$p_i$ is a primitive place, $\forall i = 1 \ldots m$

$t_j$ is a rule typed transition, $\forall j = 1 \ldots m-1$

$p_{k-1} = t^{\bullet}_{k-1} = {}^{\bullet}t_k$

In above definition, a primitive place $p_{k-1}$ is both an output place of a rule transition $t_{k-1}$ and an input place of transition $t_k$ which means that action of rule $k-1$ is the event that triggers the rule $k$. When in a triggering sequence $p^{\bullet}_m = \emptyset$ or some pair is already considered in the path, the sequence is complete. If $p_m = p_1$, then we have a circular triggering sequence, whose CCPN structure is showed in Figure 4(a). Figure 4(b) depicts the CCPN structure for rules whose event and action are the same. Copy typed transition was added to make a pure CCPN, i.e., to avoid self-loops. Finally, Figure 4(c) stands for potential circular triggering sequences.

As a traditional Petri net, cycles can be detected by computing T-invariants over incidence matrix of CCPN. However, potential termination caused by pseudo-triggering relations cannot be identified by this method. In this case, we compute circular triggering sequences by following negative/positive entries in columns/rows of incidence matrix. Then, potential circular triggering sequence information is completed with data about events which are not part of the cycle to have a complete scenario of all the loop causes. For example, in Figure 4(c) information of place $e_0$ is added to circular triggering sequence $S = \{(e_1, t_0), (t_0, e_2), ..., (t_m, e_1)\}$ since $e_0$ is necessary to form the cycle.

*b) Step 3.2 Rule Interaction Analysis:* Rule interaction analysis checks if the rules computed in previous step are circular rules. According to Definition 1 circular rules triggers and activates each other. **Rule 2** triggers itself, so, we need to check if it activates itself also.

Our definition of rule activation is based on our condition's domain concept.

*Definition 5:* **Domain of a condition.** The domain of a condition $c$, denoted by $\text{Dom}(c)$, is the set of information that satisfies condition $c$.

Below definition gives the notion of rule activation.

*Definition 6:* **Rule activation.** A rule $r_i$ activates $r_{i+1}$ if after $a_i$ execution $c_{i+1}$ changes from the state in which $\text{Dom}(c_{i+1}) = \emptyset$ to $\text{Dom}(c_{i+1}) \neq \emptyset$.

$\text{Dom}(c_{i+1}) \neq \emptyset$ means that there is data satisfying $c_{i+1}$, so, $c_{i+1}$ is true after $a_i$ execution and $r_i$ activates $r_{i+1}$.

In this paper we consider simple conditions of the form $at\theta const$ where $at$ is an attribute belonging of a relation, $const$ is a constant value and $\theta = \{=, \neq, >, \geqslant, <, \leqslant\}$. However, we can analyze complex conditions even using SQL statements.

We say that $a_i$ activates $c_{i+1}$ if the following conditions are met:

1) $a_i$ performs "insert" or "update" operations, i.e., $a_i$ produces new data.
2) $a_i$ modifies the same relation and attribute as the condition $c_{i+1}$ supervises, i.e., it checks if $a_i$ changes the same attributes in the same relation than $c_i$
3) $a_i$ assigns a value $x$ to the attribute such that $x \in Dom(c_{i+1})$, i.e., it verifies if the new data satisfies $c_{i+1}$.

Potential termination is investigated taking into account both rule activation and information about events not considered in the cycle. For example, if in sequence $S = \{(e_1, t_0), (t_0, e_2), ..., (t_m, e_1)\}$ of Figure 4(c) all the transitions activate each other and $e_0$ is always ready then then cycle will occur.

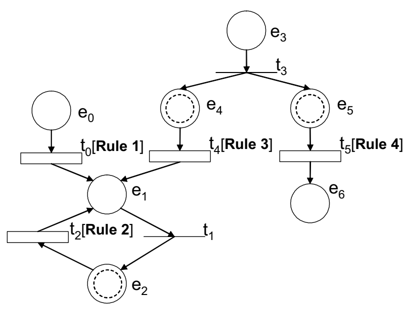
## IV. Example

Let's consider the rule base of Example 1 to demonstrate our analysis approach. Rules in the rule base can be easily implemented in a real database system in the form of triggers, so it is very important to assure its proper performance before they are delivered.

Rule Normalization step is not necessary since all rules are atomic.

During Rule Modeling stage, first, the rule base is modeled as a CCPN structure. Figure 5 shows the complete CCPN model of rules as well as a brief description of places. Each rule typed transition is attached with the rule's condition that it represents. For example, transition $t_0$ evaluates **Rule 1**'s conditions and also represents **Rule 1**. Primitive input and output places of each rule typed transition represent events and actions of the rule, respectively. CCPN graphical structure makes clear relationships among rules. For example, it is evident that **Rule 3** and **Rule 4** are triggered by the same event.

Then, we compute the incidence matrix of CCPN in Figure 5. It is showed in Figure 6 and it contains the same information as the original CCPN but in a summarized form.



| Place [Type] | Event/Action |
|---|---|
| $e_0$ [Primitive] | insert account |
| $e_1$ [Primitive] | update account.*rate* |
| $e_2$ [Copy] | Copy of $e_1$ |
| $e_3$ [Primitive] | update account.*balance* |
| $e_4$ [Copy] | Copy of $e_1$ |
| $e_5$ [Copy] | Copy of $e_1$ |
| $e_6$ [Primitive] | delete from account |

Fig. 5. CCPN of Example 1

$$\begin{array}{c|ccccccc|}
 & e_0 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\
t_0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\
t_1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\
t_2 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\
t_3 & 0 & 0 & 0 & -1 & 1 & 1 & 0 \\
t_4 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\
t_5 & 0 & 0 & 0 & 0 & 0 & -1 & 1
\end{array}$$

Fig. 6. Incidence matrix of CCPN in Figure 5

Finally, in Termination Analysis step, we compute all the circular triggering sequences in CCPN using its incidence matrix representation. We found that $S1 = (e_1, t_2) \rightarrow (t_2, e_1)$ is a circular triggering sequence.

Since rule typed transition $t_2$ stands for **Rule 2**, we need to confirm if it activates itself. Checking activation conditions for **Rule 2**, we have:

1) $a_2$ performs “update” operation.
2) $a_2$ modifies relation `account` and attribute *rate*. $c_2$ supervises that *rate* $> 0$.
3) $a_2$ performs *rate* = 0, it doesn't belong to $(0, +\infty)$ which is part of Dom$(c_2)$.

Since condition 3 is not satisfied by **Rule 2**'s action, that rule is not a circular one. Then, although **Rule 2** depicts a self-loop its processing always finishes. Therefore, termination is assured for the complete rule base.

## V. Conclusion

In this paper we have addressed the analysis of termination property in an active rule base. First, we describe the traditional termination property and novelty introduce *potential termination* concept. This concept is very useful to foresee and manage those rules whose processing may not terminate at runtime. Second, we describe a CCPN-based method for accurately detect both termination and potential termination. Our work gives a complete framework for analyzing termination, an important characteristic of rule behavior.

## References

[1] S. Comani, L. Tanca. Termination and Confluence by Rule Prioritization, *IEEE Trans. on Knowl. and Data Eng.*, Vol. 15, No. 2, pp. 257-270, 2003.

[2] D. Montesi, and R. Torlone, Analysis and Optimization of Active Databases, *Data and Knowledge Engineering*, Vol. 40, pp. 241 - 271, 2002.

[3] X. Li, J. Medina-Marín, and Chapa S., “Applying Petri Nets on Active Database Systems”, *IEEE Trans. on System, Man, and Cybernetics, Part C: Applications and Reviews,* Vol. 37, No. 4, pp. 482 - 493, 2007.

[4] N. Paton, O. Díaz, Active Database Systems, *ACM Computing Surveys*, Vol. 31, No. 1, pp. 62-103, 1999.

[5] M. Zoumboulakis, G. Roussos and A. Poulovassilis, Active Rules for Sensor Databases, in *Proc. of the 1st Intl. Workshop on Data Management for Sensor Networks: in conjunction with VLDB 2004*, Toronto, Canadá, pp. 98 - 103, 2004

[6] E.Baralis, J. Widom, An Algebraic Approach to Static Analysis of Active Database Rules, *ACM Trans. on Database Systems*, Vol. 25 , Issue 3, pp. 269 - 332, 2000.

[7] J. C. Augusto, and C. Nugent, A New Architecture for Smart Homes Based on ADB and Temporal Reasoning, in *Toward a Human Friendly Assistive Environment (Proc. of 2nd Intl. Conf. on Smart homes and health Telematic, ICOST2004), Assistive Technology Research Series*, Vol. 14, pp. 106-113, IOS Press, Singapore, September 15-17, 2004.

[8] A. Aiken, J. Widom, and J-M. Hellerstein, Behavior of Database Production Rules: Termination, Confluence, and Observable Determinism, *in Proc. of ACM-SIGMOD, Intl. Conf.,* pp. 59-68, 1992.

[9] E. Baralis, S. Ceri, and S. Paraboschi, Improved Rule Analysis by Means of Triggering and Activation Graphs, in *Proc. First Intl. Workshop Rules in Database Systems,* Aug. 1993.

[10] M. K. Tschudi, S. D. Urban, S. W. Dietrich, and A. P. Karadimce, An Implementation and Evaluation of the Refined Triggering Graph Method for Active Rule Termination Analysis, in *Proc. of the Third Intl. Workshop on Rules in Database Systems*, *Lecture Notes In Computer Science*, Vol. 1312, pp: 133 - 148, 1997.

[11] E. Baralis, S. Ceri, S. Paraboschi, Improved Rule Analysis by Means of Triggering and Activation Graphs, in *Proc. of 2nd Intl. Workshop on Rules in Database Systems, RIDS '95*, LNCS vol. 985, pp. 165 - 181, Athens, Greece, 1995.

[12] H. Harb, H. Kelash and A. Shehata, Termination Analysis in Active Database by using Evolution Graphs, in Proc. of *3th Intl. Conf. on Information & Communications Technology*, 5-6 Dec., 2005, Cairo, Egypt, pp. 781 - 791, 2005.