Termination Analysis Approach and Implementation

L. Baba-hamed and H. Belbachir
Laboratory: Systems, Networks and Data bases, USTO, B.P. 1505 El Menouar, Oran, Algeria

Abstract: An active database system is a system which provides the same functionalities as a classic database system and is, equally, capable to react, automatically, to state changes by means of rules said active rules. The triggering of these rules can produce an infinite cycle and leads to the no termination problem. In this study, we propose a method of termination analysis of active rules based on Petri nets (PN) and give an object oriented representation to implement it. This approach is better than the previous ones because it takes into a count composite events and the rule priority on the one hand and both rule representation and rule analysis are performed in the same PN on the other hand.

Key words: ECA rule, termination, petri net, priority, path, class

INTRODUCTION

Active database systems (Widom and Ceri, 1996; Paton and Diaz, 1999) aim at the representation of more real-world semantics in the database by supporting event-condition-action rules (ECA-rules). ECA-rules can be interpreted as “when the specified event occurs and the condition holds, execute the action”. An event indicates the point in time when some sort of reaction is required from the DBMS. For primitive events, this point in time can be specified by an occurrence in the database, by an occurrence in the DBMS, or by an occurrence in the database environment. For composite events (Chakravarty and Mishra, 1994; Gatzii and Dittrich, 1994; Gehani et al., 1992), the point in time is defined on the basis of other points in time which represent other primitive and/or composite events (called component events). These components are combined by means of event constructors as: negation, conjunction, disjunction, sequence etc. The action describes treatments to achieve when a specific event happens and some condition holds.

The active rule behaviour is difficult to predict and require the development of techniques to analyse properties of a set of rules automatically. One of these important properties is the property of the rule termination. Many works have tried to solve the no termination problem; nevertheless, most of them present insufficiencies.

The aim of this study is to propose a rule termination analysis approach which considers the impact of both the composite events and the rule priority on the termination analysis problem. This method detects cyclic paths in the base of ECA rules, can analyse the relationships among ECA rule components and detects cases of termination which are not detected by the other approaches.

Among all methods which proposed to study rule termination, some are based on models of graphs and others use formal basis as the systems of rewrite or the Petri nets. We present some of them in the following.

Aiken et al. (1995) are the first to introduce the notion of Triggering Graph (TG). They show that a triggering graph without cycle determines and guarantees the termination of a set of active rules in the context of relational systems. Baralis et al. (1996) grouped the active rules into modules, termination of rule execution, within each module is assumed and inter-module termination is analysed. It is the only method that presents a modular conception of the active rules. Baralis et al. (1998) proposed a technique that exploits the information of the graph TG and other graph called Activation Graph (AG) to analyse the termination of a set of ECA rules. This analysis uses an algorithm called algorithm of reduction. This approach presents an inconvenience because it doesn't propose a method of building the AG graph which is not obvious. Baralis and Widom (2000) try to improve these last methods. Their approach is based on a “propagation algorithm”, which uses an extended relational algebra to accurately determine when the action of one rule can affect the condition of another. The termination analysis is made by building the graph AG. Their study is considered as a complementary method for the two last one. Lee and Ling (1998) propose a path technique for reducing the graph TG. The method considers together the conditions of long triggering sequences called activation formulas. It is necessary to
guarantee that the execution of rules outside the
triggering sequence cannot unpredictably change the
database state. Hence, only non-updateable predicates can
be included in the activation formula. This condition
severely limits the applicability of the technique.
Bailey et al. (1997) use abstract interpretation for the
termination analysis of active rules. The idea is to reason
about sequences of database states using "approximate
semantics" and to use the fix point computation (over a
lattice) to handle cycles. This approach is applicable to a
simple and restricted rule language. A different approach
is taken in (Karadimos and Urban, 1994), where ECA rules
are reduced to term rewriting systems and known analysis
techniques for termination of term rewriting systems are
applied. The analysis is based on an object-oriented data
model and instance-oriented rule execution model. This
approach is powerful, since it exploits the body of
work on Conditional Term Rewriting System, but its
implementation appears to be complex even for small rule
Nets (PPN) to analyse the active rule termination in the
relational model. A PPN is a Petri Net (PN) whose places
are parameterised. The firing of the transition corresponds
to the rule execution. If there is no cycle in the PPN model
then the rules execution in the ADBS must terminate.
The inconvenience of this approach is that, the use of PPN in
modelling complex systems results in complex graphic
representations which are very difficult to be
conceptualised and handled. Other PN based method is
presented in (Zimmer et al., 1996). To represent the
triggering and activation notions in the PN, the authors
give for each rule two subsets Ei (for the triggering)
and Ci (for the activation). The authors detect a
non-terminating behaviour of rules using a coverability
table based on the reachability graph. In this approach,
the conception of the PN is too complex. In addition, the
presence of a cycle in the PN does not imply that will
occur an infinite rule triggering. Li and Marin (2004)
present an approach based on coloured Petri nets named
CCPN (Conditional Coloured Petri Net) for modelling the
active database behaviour. Incidence matrix of PN theory
is used to find cyclic paths existing in the CCPN. Cycles
which satisfy some theorems given by the authors are
deleted. If there is no cycle in the CCPN, the termination
of the corresponding set of rules is guaranteed.
Nevertheless, this approach does not consider the priority
of rules.

EXTENDED COLORED PETRI NET (ECPN)

Our approach is inspired by Li and Marin (2004). We
improve this method by adding the notion of rules priority
and showing how the termination analysis can be affected
by this notion.

PN is a graphical and mathematical tool and may be
applied in various areas. Active database is a promising
application area of PN. Up to now, few researches have
used PN as ECA specification language. In SAMOS
(Gatziou and Dittrich, 1994; Geppert et al., 1995), PN is
partially used for composite event detection and
termination analysis.

In our model named ECPN, the rule event is
represented by a place p1, the condition c and the priority
pr of the rule are attached to a transition t and the rule
action a is represented by a place p2. Relationships
between the rules can be viewed in the same graph
(Fig. 1).

ECPN is a colored PN (David and Alla, 1989, Jensens,
1992) defined as follows:

ECPN = {E, P, T, A, N, C, Con, Action, D, τ, I} where:

E is a finite set of non-empty type, called colour sets.
It determines all the data value, the operations and
functions that can be used in the net expressions.

P is a finite set of places. It is divided into four
subsets: P_{pre}, P_{comp}, P_{act} and P_{copy}. P_{pre} represents the set of
primitive places and correspond graphically to a single
circle. P_{comp} represents the set of composite event. P_{comp}
includes the following events: negation, sequence,
closure, last, history and simultaneous. They correspond
graphically to a double circle. P_{act} represents the set of
composite events which includes the conjunction,
disjunction and any. They correspond graphically to a
single dashed circle. P_{copy} is the set of places which are
used when two or more rules are triggered by the same
event. They correspond graphically to a double circle
where the interior circle is a dashed one.

T is a finite set of transitions, it is divided into three
subsets: T_{pre}, T_{comp} and T_{copy}. T_{pre} corresponds to the set of
rules. Each transition of rule type is represented
graphically by a rectangle. T_{copy} is the set of transitions of
copy type. They are represented graphically by a single

---

Fig. 1: Relationships between two rules r1 and r2
bar. A copy transition is used when one event e can trigger two or more rules. A copy transition will produce n same events where n is the number of rules which are triggered by the same event e. \( T_{\text{comp}} \) is the set of transitions of composite type. They are represented graphically by a double bar. A composite transition is used for generating a composite event from a set of primitive or composite events.

A is a finite set of arcs. It is divided into two subsets: The input arcs which are defined from P to T and output arcs which are defined from T to P. Inhibitor arcs are used to represent the Negation composite event.

\( N \) is a node function. It maps each arc into a pair where the first element is the source node and the second is the destination node (N: A \( \rightarrow P \times T \times T \times P \)).

\( C \) is a color function. It maps each place p to a type C(p) (i.e., C: P \( \rightarrow \mathbb{E} \)).

\( \text{Con} \) is a condition function. It is defined from either \( T_{\text{rule}} \) or \( T_{\text{comp}} \) into expression such that:

\[
\forall t \in T_{\text{rule}} \, \text{Type} \left( \text{con} \left( t \right) \right) = B,
\]

where Con function evaluates the rule condition.

\[
\forall t \in T_{\text{comp}} : \text{Type} \left( \text{con} \left( t \right) \right) = B,
\]

where Con function evaluates the time interval of t against tokens timestamp. B is used to denote the Boolean type containing the values false and true.

Action is an action function. It maps each rule type transition \( t \in T_{\text{rule}} \) into a type C(p) which will be deposited into its output place.

\( D \) is a time interval function. It is defined from \( T_{\text{comp}} \) to a time interval \([d_i(t), d_f(t)]\), where \( t \in T_{\text{comp}} \) and \( d_i(t), d_f(t) \) are the initial and the final interval time, respectively. The interval is used by the Con function to evaluate transitions \( t \in T_{\text{comp}} \).

\( \tau \) is a timestamp function. It assigns each token in place p a timestamp.

\( \text{I} \) is an initialization function. It maps each place p into a closed expression which must be of type C(p).

**Example of an ECPN:** Let's consider the set of rules R1, R2, R3 and R4 expressed according to the following formalism:

Define rule rule-name On event If condition Then action

Define rule R1
On reduce-salary () (e0)
If employee.salary < 1500 (T0)
Then raise-salary () (e1)

Define rule R2
On raise-salary () (e3 a copy of e1)
If employee.children-nbr > 5 (T3)
Then send-bonus () (e4)

Define rule R3
On raise-salary () (e2 a copy of e1)
If employee.age > 60 (T2)
Then be-retired () (e5)

Define rule R4
On send-bonus () (e4)
If employee.salary < 10000 (T4)
Then raise-salary () (e1)

These rules necessitate the class Employee which has the attributes: id-emp, salary, bonus, children-nbr, age. The ECPN which corresponds to the set rules given above is shown in Fig. 2. Furthermore, we consider that R1 > R3 > R2 > R4 where R1 > Rj means that Ri has priority than Rj.

Fig. 2: Example of an ECPN

<table>
<thead>
<tr>
<th></th>
<th>e0</th>
<th>e1</th>
<th>e2</th>
<th>e3</th>
<th>e4</th>
<th>e5</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 3: Incidence matrix from ECPN of Fig. 2
Termination analysis: Let $S = \{r_1, r_2, r_3, \ldots, r_m\}$ be a set of rules, when the action of rule $r_1$ triggers the rule $r_2$, action of rule $r_2$ triggers the rule $r_3$ and so on and finally, action of rule $r_m$ triggers the rule $r_1$, this process performs a cyclic rule triggering and it can produce an inconsistent state of the database when it executes these rules infinitely. To study the problem of rules termination and the impact of rules priority on it, we give an algorithm which uses the incidence matrix, the notions of paths, cyclic paths and acyclic paths of PN theory. In incidence matrix, places are represented by its columns and transitions are represented by its rows, so it is possible to identify both initial and final nodes of an ECPN.

ECPN is a directed graph constituted by a sequence of nodes forming paths, where each node is either a place or a transition in an alternate way. If a cyclic path is found, then it may produce an infinite rule triggering.

A path $R$ is a sequence of pairs $(r, c)$ which are obtained from the incidence matrix of the ECPN, where $r$ and $c$ are incidence matrix indexes.

The sequence of pairs $(r, c)$ describes the connection between places and transitions as follows: $(t_1, p_1), (t_1, p_2), (t_2, p_2), \ldots, (t_{n-1}, p_{n-1}), (t_n, p_n)$. The first pair is formed by the transition $t_1$ and its input place $p_1$, following the path, the next pair is formed by the same transition $t_1$ and its output place $p_2$, the third pair is formed by the same place $p_2$ that now is the input place to transition $t_2$ and so on.

Paths search starts from the coordinate of $(t_1, p_1)$ and then a positive integer is looked for in the row corresponding to $t_1$, finding the coordinate to $(t_1, p_2)$. After, a negative integer is looked for in the column corresponding to $p_2$, finding the coordinate to $(t_2, p_2)$. Then another positive integer is looked for in the row corresponding to $t_2$ and so on, until either a terminal node or an existing node in the path is found. A cyclic path $CP$ is a path $R$ where the last pair $(r, c)$ has been already found. An acyclic path $AP$ is a path where the last pair $(r, c)$ is different from each other in AP. If all the paths, in the ECPN, are acyclic then the termination is guaranteed. If there is at least one cyclic path $CP$ in the ECPN, the rule triggering may not finish. But it does not mean that whenever exist a cyclic path $CP$ the rule triggering does not terminate; there are other facts that should be taken into account. For example, in ECPN model, the priority, the condition and the composite events of ECA rules are considered, so they can have an impact on the termination analysis problem. The composite events that affect the termination analysis are: Negation, conjunction, any and sequence. The conditions 1, 2 and 3, given below, correspond to them. The condition 4 concerns the rule condition belonging to a cyclic path.

**Condition 1:** If a cyclic path $CP$ contains an inhibitor arc i.e., a composite event negation is included in $CP$, then $CP$ finishes its rule triggering.

**Condition 2:** For composite events conjunction, sequence and simultaneous, if any of its constituent events is not generated by the action of a rule belonging to $CP$ then the rule triggering finishes.

**Condition 3:** If composite event any $(m, e_1, e_2, \ldots, e_n)$ is a part of a cyclic path $CP$ and if $k$ constituent events of the composite event any are not generated by the action of a rule belonging to $CP$ and $n-k < m$ then the rule triggering finishes.

**Condition 4:** If the condition of a transition $t_i \in \{ t \mid (t, p) \in CP \}$ is always false according to the event produced by the action of previous rule, then the rule triggering finishes.

**Algorithm of the priority:** The following algorithm shows the impact of the rules priority on the termination analysis for a set of rules modelled by an ECPN.

Step 1: Convert a base of ECA rules into an ECPN
Step 2: Create the incidence matrix from the ECPN
Step 3: Search all the paths of ECPN
Step 4: Create a set $Rset$ of paths $R$
Step 5: If $Rset$ contains no cycle.
Then returns “the termination is guaranteed”
Else explore the paths in $Rset$ one by one according to the priority of rules:

```plaintext
terminate = false
Repeat for each path $R$
    If $R$ is acyclic
        Then terminate = true
    Else /*$R$ is cyclic*/
        If $R$ satisfies the conditions given above
            Then terminate = true
        Else consider the next path
    End
End
Until the end of the paths exploration or terminate=true
If terminate = true
    Then returns “the termination is guaranteed”
Else returns “the termination is not guaranteed”
End
```

**Illustrative example:** To show the impact of the rules priority on the termination analysis in an ECPN, we consider the example given above. The incidence matrix
generated from the ECPN of Fig. 2 (which corresponds to the example) is presented in Fig. 3. It can be observed that there are two paths constituted by the elements: (0,0), (0,1), (1,1), (1,2), (2,2) and(2,5) for the first one and the elements: (0,0), (0,1), (1,1), (1,3), (3,3), (3,4), (4,4), (4,1) and (1,1) for the second one which includes the cycle. Nevertheless, as R3 has priority than R2, this cycle is not considered and the termination of the set {R1, R2, R3, R4} is guaranteed according to the algorithm. The path to which belongs R3 (represented by the transition T2) is acyclic.

**IMPLEMENTATION ISSUES**

In order to implement an ADDS, the architecture of a (passive) DBMS has to be augmented by new components like an ECA rule editor, an analyzer of rules, a rule manager, an event detector for primitive and composite events and a rule execution component.

ECA rules are defined by ECA rule editor. After ECA rules are converted into an ECPN, ECPN is saved into ECPN base as places, transitions and arcs. ECPN rule base is used by ECPN rule manager which calls the event detector for detecting events, the rule execution component for evaluating the condition and executing the action of rules. It calls also the termination analyzer component to check no-termination problem in ECPN rule base. This last component includes an incidence matrix generator, a paths generator, a cyclic paths detector and an analyzer of paths which take into a count the priority of rules. As the implementation of our passive DBMS is an object-oriented implementation and in accordance to the idea “to stay in the same world and exploit its advantages”, we built the rule structure by means of object-oriented features (Atkinson et al., 1989). In our model, the ECA rules are represented by transitions of rule type, the conditions and the priorities of rules are attached to these transitions. The events and the actions are modelled by places which are inputs and outputs of a transition respectively. The rule firing corresponds to transition firing. So we need four main classes named PLACE, TRANSITION ARC and TOKEN.

All places are instances of a class PLACE with attributes index, event-name, event-body, list-rules,
list-tokens and methods insert-place, delete-place, modify-list-rules and modify-list-tokens. The index is an internal number assigned by the system. The event-name is the event name. The event-body is the content of the rule event part. The list-rules is the list of references to objects of class T-RULE which are triggered by this event. The list-tokens is a list of references to objects of class TOKEN; these tokens represent the actual marking of the place. Class PLACE is the super-class of the subclasses P-COMPOSITE (which includes the attributes composed-by and interval), P-PRIMITIVE and P-COPY (which includes the attribute copy-of). composed-by holds the events (single or composite) that comprise the event interval refers to objects of class INTERVALS (i.e., refers to time intervals). copy-of holds the event for which we have made copies.

Each transition is an object of class TRANSITION with attribute index, name, arc-in, arc-out and methods insert-transition and delete-transition. The name is the transition name. The arc-in is the list of references to objects of class ARC (the set of the input arcs). The arc-out is the list of references to objects of class ARC (the set of the output arcs). Class TRANSITION is the super-class of the subclasses T-COMP, T-COPY and T-RULE. The last one regroups all transitions of rule type, it includes attributes rule-name, events, cond, act, priority and methods insert-rule, delete-rule, activate-rule, modify-priority, evaluate-condition. rule-name represents the rule name. events represents the rule event, it refers to an object of class PLACE. cond represents the rule condition; it refers to an object of class CONDITION. act represents the rule action; it refers to an object of class PLACE. priority is a number reflecting the importance of the rule.

Also we define a class ARC with three subclasses: ARC-IN (input arcs), ARC-OUT (output arcs) and ARC-INHIB (inhibitor arcs). Class ARC has as attributes index-p and index-t, the value of which are the appropriate place and transition indexes (in relation to the attribute index of classes PLACE and TRANSITION).

The attribute time in the class TOKEN (the class which regroups all the tokens of an ECPN) indicates the time when we put the token on a place. The Fig. 4 presents the class hierarchy of our metabase with more details.

CONCLUSIONS

The approach presented in this study, is based on the Petri net model to analyse the termination of a set of rules. In this PN named Extended Coloured Petri Net (ECPN), the different components of the rules are presented such as their events, conditions, actors and priorities. ECPN can model both primitive and composite events. Furthermore, not only composite events can affect the active rules termination, but also the rules priority. Our approach is better than those presented in related work section because the ECPN is a good model for modelling, analysing and simulation of active database systems. It does not perform a simple analysis of cyclic paths but analyzes each element of the graph to determine if the rule triggering in a cyclic path finishes or not. This approach is general and can be applied not only in the database area but also in others applications which need event detection.

REFERENCES


