

# Improving Active Rules Termination Analysis by Graphs Splitting

Alain Couchot

Université Paris Val de Marne, Créteil, France  
alaincouchot@minitel.net

**Abstract.** This article expounds an algorithm for static analysis of termination of active database rules. As most previous termination algorithms, our algorithm uses the notion of triggering graph. But whereas previous termination algorithms only remove from a triggering graph rules which are triggered a finite number of times or rules whose condition is detected as deactivated, our algorithm also removes some rules which can be triggered infinitely and whose condition can be evaluated to *true* infinitely. To achieve this, our algorithm splits the initial triggering graph in several triggering graphs. New termination situations can so be discovered.

## 1 Introduction

This paper deals with the termination problem of active rules. Active rules (or Event-Condition-Action rules[7]) are intended to facilitate the work of design and programming databases. But writing a set of rules actually remains a tricky work, often devolved upon specialists. Indeed, a set of rules is not a structured entity : the global behavior of a set of rules can be hard to predict and to control [1]. In particular, research works have brought to the fore the termination problem of the rules (the execution of the rules can be infinite in some situations), or the confluence problem (the same rules do not necessarily give the same results, depending on the execution order of the rules).

In section 2, we expose the related work ; in section 3, we present a motivating example ; in section 4, we introduce the maximal order  $N$  paths preceding a rule ; in section 5, we propose a function for the evaluation of the truth value of the condition of a rule; in section 6, we introduce the notion of incompatible paths; in section 7, we expose our algorithm for termination analysis ; section 8 concludes.

## 2 Related Work

The termination of the active rules is an undecidable problem, except when languages of rules with very limited possibilities are used [2]. The previous works on the static analysis of the active rules propose criteria supplying sufficient conditions allowing to guarantee termination. The majority of works on the termination of the active rules exploit the concept of *triggering graph*.

[5] introduced, for the first time, the notion of *triggering graph*. This notion is clarified by [1] : a such graph is built by means of a syntactic analysis of rules ; the nodes of the graph are rules. Two rules  $r1$  and  $r2$  are connected by a directed edge from  $r1$  towards  $r2$  if the action of  $r1$  contains an triggering event of  $r2$ . The presence of cycles in a such graph means a risk of non-termination of the set of rules. The absence of cycles in the triggering graph guarantees the termination of the set of rules. However, the possible deactivation of the condition of the rule is not taken into account by this analysis. A finer analysis is led by [3, 4, 8, 9, 14], taking the possible deactivation of the condition of a rule into account or the possible deactivation of the overall condition of a path. Generalized connection formulas are introduced by [8], in order to test if the overall condition of a path can be satisfied. [10] proposes a technique to remove a path instead of removing a node, [11] proposes a technique to unroll a cycle. [13] refines the triggering graphs, using partial and total edges, to take into account the influence of composite events.

We present in this paper an improvement of the previous termination algorithms. The basic idea of our improvement is the following : let us suppose that we have two rules  $R_1$  and  $R_2$  of the triggering graph  $G$  such that if  $R_1$  occurs infinitely,  $R_2$  can not occur infinitely. We can then split the initial triggering graph in two triggering graphs  $G_1$  and  $G_2$  such that  $G_1$  contains  $R_1$  and not  $R_2$ , and  $G_2$  contains  $R_2$  and not  $R_1$ . Termination analysis is then performed with the two triggering graphs  $G_1$  and  $G_2$ , instead of the initial triggering graph  $G$ . New termination situations can so be discovered.

### 3 Motivating Example

We present here an example, which motivates our proposition. We consider a banking application. Six active rules are defined. The production mode of these rules is "after the triggering event". The coupling modes of the rules are "immediate".

<i>Rule <math>R_1</math> :</i>	Event :	$A_1 \rightarrow \text{decrease\_overdraft}(X_1)$
	Condition :	$6000 < A_1.\text{capacity}$
	Action :	$A_1 \rightarrow \text{decrease\_capacity}(1000)$
<i>Rule <math>R_2</math> :</i>	Event :	$A_2 \rightarrow \text{decrease\_capacity}(X_2)$
	Condition :	$100 < A_2.\text{overdraft}$
	Action :	$A_2 \rightarrow \text{decrease\_overdraft}(20)$
<i>Rule <math>R_3</math> :</i>	Event :	$A_3 \rightarrow \text{decrease\_capacity}(X_3)$
	Condition :	$A_3.\text{type} = \text{standard\_account}$
	Action :	$A_3 \rightarrow \text{increase\_rate}(1)$
<i>Rule <math>R_4</math> :</i>	Event :	$A_4 \rightarrow \text{decrease\_overdraft}(X_4)$
	Condition :	$A_4.\text{type} = \text{stocks\_account}$
	Action :	$A_4 \rightarrow \text{decrease\_rate}(1)$
<i>Rule <math>R_5</math> :</i>	Event :	$A_5 \rightarrow \text{increase\_rate}(X_5)$
	Condition :	$A_5.\text{amount} > 10000$
	Action :	$A_5 \rightarrow \text{increase\_capacity}(1000)$



Hence, this rules set can not exhibit an infinite behavior. But no previous termination algorithm is able to prove this termination. This is due to the two following facts :

- (i) previous termination algorithms only try to remove rules which are not triggered infinitely or whose condition is deactivated, but do not consider "incompatible" rules ;
- (ii) previous termination algorithms only try to remove rules from cycles, but do not try to remove rules which do not belong to cycles, but which are triggered by cycles.

We propose a termination algorithm, which solves these two shortcomings.

## 4 Maximal Order $N$ Paths in Triggering Graphs

### 4.1 Triggering Graphs

We consider in this paper Event-Condition-Action rules. The underlying database model is a relational model or an object-oriented model. We assume that the coupling modes Event/Condition and Condition/Action are immediate or deferred. We do not make any assumption about the execution model of the rules.

Definition. Let  $R$  be an arbitrary active rule set. The *triggering graph* is a directed graph where each node corresponds to a rule  $R_i \in R$ . A directed arc  $(R_1, R_2)$  belongs to the triggering graph iff the action of rule  $R_1$  generates events which trigger rule  $R_2$ .

### 4.2 Maximal Order $N$ Path Preceding a Rule

**Simple Path.** We build a *simple path* of the triggering graph in this way :  $R_1, R_2, \dots, R_i, \dots, R_n$  are  $n$  rules (not necessarily all distinct) of a triggering graph such that there is an edge from  $R_{i+1}$  to  $R_i$ . The tuple  $(R_1, R_2, \dots, R_n)$  make up a *simple path*. We adopt the following notation :  $R_1 \leftarrow R_2 \leftarrow \dots \leftarrow R_i \leftarrow \dots \leftarrow R_n$ . (Note that we adopt a notation in the opposite direction of the edges, for convenience reasons).  $R_1$  is said the *last rule* of the simple path.  $R_n$  is said the *first rule* of the simple path.

**Maximal Order  $N$  Path.** We replace the classical notion of cycle by the notion of *maximal order  $N$  path preceding a rule*. The usefulness of this notion is to represent in one single entity all the simple paths preceding a rule.  $N$  is a number fixed by the designer, and corresponds to a limit on the length of the considered paths.

Let  $R_0$  be a rule. We perform a "depth search" in the opposite direction of the edges in order to calculate the maximal order  $N$  path preceding  $R_0$ :  $Max\_Path(R_0; N; Graph)$ . The procedure is the following:

$Path_0 = (R_0)$   
 $Max\_Path(R_0; N; Graph) = Path\_Building\_Function ( Path_0 )$   
 $Path\_Building\_Function$  (incoming variable :  $Path_{in}$ , outgoing variable :  $Path_{out}$ ) {  
    Let  $R$  be the first rule of  $Path_{in}$   
    Let  $R_1, R_2, \dots, R_p$  be the rules such that there is an edge from  $R_j$  to  $R$   
    FOR each rule  $R_j$  ( $1 \leq i \leq p$ ) {  
        IF  $R_j$  appears less than  $N$  times in  $Path_{in}$  {  
             $Path_i = Path\_Building\_Function (Path_{in} \leftarrow R_j)$  } }  
     $Path_{out} = ( Path_1 \text{ OR } Path_2 \dots \text{ OR } Path_p )$  }

**Example.** See figure 1. The maximal order 1 path preceding  $R_5$  is :

$Max\_Path(R_5; 1; Graph) = (R_5 \leftarrow R_3 \leftarrow R_1 \leftarrow R_2)$

## 5 Evaluation of the Truth Value of the Condition of a Rule

We introduce a function  $TV$ , which allows us to store the truth value of the condition of a rule due to a simple path. Thanks to the properties of the function  $TV$ , we can then evaluate the truth value of the condition of a rule due to the maximal order  $N$  path preceding the rule.

Let  $Graph$  be the initial triggering graph, or a subgraph of the initial triggering graph ; let  $Path$  be a simple path of  $Graph$  and  $Rule$  the last rule of  $Path$ . We evaluate the truth value of the condition of the rule  $Rule$  due to the path  $Path$  for the graph  $Graph$  using a function  $TV$ , which associates a boolean value to the triple (  $Rule ; Path ; Graph$  ).

The meaning of the function  $TV$  is the following :

If  $TV ( Rule ; Path ; Graph )$  is *FALSE*, this means that, for each rules process  $P$ , composed of rules of  $Graph$ , there is only a finite number of occurrences of  $Path$ .

If  $TV ( Rule ; Path ; Graph )$  is *TRUE*, this means that we are not sure of the truth value of the condition of  $Rule$  due to  $Path$  for  $Graph$ .

The function  $TV$  is determined using the following formulas :

An unknown value  $TV ( Rule ; Path ; Graph )$  is supposed equal to *TRUE*. (1)

$TV( Rule ; Path ; Graph ) = FALSE$  (2)

if we are sure that the path  $Path$  can just occur a finite number of times for any rules process composed of rules of  $Graph$ . Previous termination algorithms can be used to determine this [3, 4, 8, 9, 12, 14]. For example, we can use a generalized connection formula [8] along  $Path$ .

We extend then the function  $TV$  to the maximal paths, using the following formula:

$TV( Rule ; Path \text{ OR } Path_1 ; Graph ) =$  (3)

$TV( Rule ; Path ; Graph ) \text{ OR } TV( Rule ; Path_1 ; Graph )$ .

(where  $Path_1$  is a path such that  $Rule$  is the last rule of  $Path_1$ )

## 6 Incompatible Paths

**Definition.** Let *Graph* be the initial triggering graph, or a subgraph of the initial triggering graph. Let *Path*<sub>1</sub>, *Path*<sub>2</sub> be two simple paths of *Graph*. We say that *Path*<sub>1</sub> and *Path*<sub>2</sub> are *incompatible for Graph* iff we have the following property :

For each rules process *P* composed of rules of *Graph*, if there is an infinite number of occurrences of *Path*<sub>1</sub> during *P*, there is just a finite number of occurrences of *Path*<sub>2</sub> during *P* (and vice versa).

Previous termination algorithms can be extended to determine incompatible paths. For example, we can study if the conjunction of the following generalized connection formulas [8, 12] can be satisfied for each pair (*n*, *p*): (i) a generalized connection formula binding the *n*<sup>th</sup> occurrence of *Path*<sub>1</sub> and the *p*<sup>th</sup> occurrence of *Path*<sub>2</sub>; (ii) a generalized connection formula concerning the *n*<sup>th</sup> occurrence of *Path*<sub>1</sub>; (iii) a generalized connection formula concerning the *p*<sup>th</sup> occurrence of *Path*<sub>2</sub>.

## 7 Termination Algorithm

Our termination algorithm removes rules from the current triggering graph, and splits the current triggering graph in several triggering graphs. The algorithm is composed of three main parts. The first part deals with the "forward propagation" of rules removal. The second part deals with the detection of the deactivation of the condition of a rule due to the maximal order *N* path preceding the rule. The thirist part deals with splitting the current triggering graph.

### 7.1 Algorithm

The sketch of the termination algorithm is the following :

```
G = { Initial_Triggering_Graph }
FOR each graph Gi of G
  set down Current_Graph = Gi
  WHILE a new split of Current_Graph is detected {
    WHILE new rules of Current_Graph are removed {
      WHILE new rules of Current_Graph are removed {
        Part One: Forward propagation of the rules removal
        remove the rules of Current_Graph without incoming edge }
      WHILE (a deactivation of condition is not detected)
      AND (all the rules of Current_Graph are not tried (as rule R0)) {
        Part Two: Detection of the deactivation of the condition of a rule }
      WHILE (a split of Current_Graph is not detected)
      AND (all the rules of Current_Graph are not tried (as rule R1)) {
        Part Three : "Splitting the graph" } } }
```

**Part Two: Detection of the Deactivation of the Condition of a Rule.** The goal of this part is to find a rule  $R_0$  whose condition is deactivated by the maximal order  $N$  path preceding  $R_0$ . A such rule can be removed from the current triggering graph.

Part Two:  
 Choose a rule  $R_0$   
 Calculate  $Max\_Path(R_0; N; Current\_Graph)$   
 IF  $TV(R_0; Max\_Path(R_0; N; Current\_Graph); Current\_Graph) = FALSE$  {  
     remove  $R_0$  from  $Current\_Graph$  }

**Part Three: Splitting the Graph.** The goal of this part is to find a pair of simple paths ( $SP_1, SP_2$ ) such that  $SP_1$  and  $SP_2$  are incompatible for the current triggering graph. In this case, we split the current triggering graph in two triggering graphs : in the first triggering graph, we consider that  $SP_1$  can not occur ; in the second triggering graph, we consider that  $SP_2$  can not occur.

Part Three :  
 Choose a rule  $R_1$   
 Calculate  $Max\_Path(R_1; N; Current\_Graph)$   
 ( $Max\_Path(R_1; N; Current\_Graph) = Simple\_Path_{11} OR \dots OR Simple\_Path_{1n}$ )  
 WHILE (a split of  $Current\_Graph$  is not detected)  
 AND (all the rules of  $Current\_Graph$  are not tried (as rule  $R_2$ )) {  
     Choose a rule  $R_2$   
     Calculate  $Max\_Path(R_2; N; Current\_Graph)$   
     ( $Max\_Path(R_2; N; Current\_Graph) = Simple\_Path_{21} OR \dots OR Simple\_Path_{2p}$ )  
     Determine two integers  $k$  and  $m$  ( $k \in \{1, 2, \dots, n\}, m \in \{1, 2, \dots, p\}$ ) such that :  
     (i)  $Simple\_Path_{1k}$  and  $Simple\_Path_{2m}$  are incompatible for  $Current\_Graph$   
     (ii)  $TV(R_1; Simple\_Path_{1k}; Current\_Graph) = TRUE$   
     (iii)  $TV(R_2; Simple\_Path_{2m}; Current\_Graph) = TRUE$   
     IF  $k$  and  $m$  exist {  
         Split  $Current\_Graph$  in  $G_1$  and  $G_2$   
         Initialize  $G_1$  and  $G_2$  with  $Current\_Graph$   
         Set down :  
          $TV(R_1; Simple\_Path_{1k}; G_1) = FALSE$   
          $TV(R_2; Simple\_Path_{2m}; G_2) = FALSE$   
          $G = G \cup \{G_1, G_2\} \setminus \{Current\_Graph\}$   
          $Current\_Graph = G_1$  }

**Termination of the Initial Triggering Graph.** If all the rules of all the graphs of the set  $G$  have been removed, the termination of the initial triggering graph is guaranteed. If, after application of the termination algorithm, there is at least one graph in the set  $G$  with remaining rules, these rules risk being triggered infinitely. The designer has to examine, and possibly, modify the remaining active rules. Several rules subsets are then provided to the designer : if two rules  $R_1$  and  $R_2$  are in the same remaining graph, they may both occur infinitely during the same rules process ; if there is no

remaining graph such that  $R_1$  and  $R_2$  are in the same remaining graph,  $R_1$  and  $R_2$  can not both occur infinitely during the same rules process.

## 7.2 Example

Let us study the example of section 3. Let  $G$  be the initial triggering graph (figure 1). The parts 1 and 2 of our algorithm give no result. Let us apply the part 3. Let us choose the rule  $R_4$ . We build the maximal order 1 path preceding  $R_4$  :

$$Max\_Path(R_4 ; 1 ; G) = (R_4 \leftarrow R_2 \leftarrow R_1 \leftarrow R_2)$$

Let us then choose the rule  $R_3$ . We build the maximal order 1 path preceding  $R_3$  :

$$Max\_Path(R_3 ; 1 ; G) = (R_3 \leftarrow R_1 \leftarrow R_2 \leftarrow R_1)$$

Let  $A_{i3}(n)$  be the variable referring to the account raising the event of the rule  $R_i$  during the  $n^{th}$  occurrence of  $Max\_Path(R_3 ; 1 ; G)$  and let  $A_{i4}(p)$  be the variable referring to the account raising the event of the rule  $R_i$  during the  $p^{th}$  occurrence of  $Max\_Path(R_4 ; 1 ; G)$ . We have the following property:  $A_{23}(n) = A_{24}(p)$ , for each  $n$  and each  $p$ . We can then establish the following generalized connection formula :

```
\*  $n^{th}$  occurrence of  $Max\_Path(R_3 ; 1 ; G)$  :*\  
( $A_{33}(n).type = standard\_account$  AND  $A_{33}(n) = A_{13}(n)$  AND  $A_{13}(n) = A_{23}(n)$ ) AND  
\*  $p^{th}$  occurrence of  $Max\_Path(R_4 ; 1 ; G)$  :*\  
( $A_{44}(p).type = stocks\_account$  AND  $A_{44}(p) = A_{24}(p)$ ) AND  
\* Formula binding the  $n^{th}$  occurrence of  $Max\_Path(R_3 ; 1 ; G)$  and the  $p^{th}$   
occurrence of  $Max\_Path(R_4 ; 1 ; G)$  *\  
 $A_{23}(n) = A_{24}(p)$ 
```

This generalized connection formula can not be satisfied. The attribute type can not be modified by a rule action. Thus,  $Max\_Path(R_3 ; 1 ; G)$  and  $Max\_Path(R_4 ; 1 ; G)$  are incompatible paths for  $G$ .

We split  $G$  in two triggering graphs  $G_1$  and  $G_2$ . In  $G_1$ , we set down :  $TV(R_4 ; (R_4 \leftarrow R_2 \leftarrow R_1 \leftarrow R_2) ; G_1) = FALSE$ . Hence,  $R_4$  can be removed from  $G_1$  (figure 2).

We have then:

$$TV(R_2 ; Max\_Path(R_2 ; 1 ; G_1) ; G_1) = FALSE \text{ (using [8,12])}.$$

$R_2$  can be removed from  $G_1$ . By forward propagation, all the rules of  $G_1$  are removed.

In  $G_2$ , we set down :  $TV(R_3 ; (R_3 \leftarrow R_1 \leftarrow R_2 \leftarrow R_1) ; G_2) = FALSE$ . Hence,  $R_3$  can be removed from  $G_2$  (figure 3). We have then:

$$TV(R_1 ; Max\_Path(R_1 ; 1 ; G_2) ; G_2) = FALSE \text{ (using [8, 12])}.$$

$R_1$  can be removed from  $G_2$ . By forward propagation, all the rules of  $G_2$  are removed.

So, termination of this active rules set is guaranteed by our algorithm.

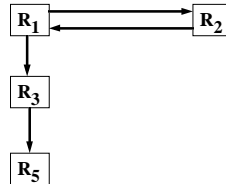


Fig. 2. Triggering Graph  $G_1$ .

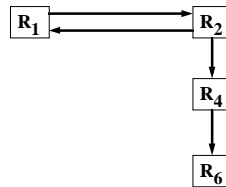


Fig. 3. Triggering Graph  $G_2$ .

## 8 Conclusion

We have presented an algorithm to improve the previous termination algorithms in the context of active databases. Our termination algorithm can remove rules which are triggered by cycles, but which are not contained in cycles (previous termination algorithms only remove rules from cycles). New termination situations can so be discovered. Thanks to the graph splitting, our algorithm removes from the triggering graph some rules which can in principle be triggered infinitely, and whose condition can in principle be evaluated to *TRUE* infinitely. This is a substantial improvement of the previous termination algorithms.

In the future, we will improve the previous termination algorithms, to determine if two simple paths are incompatible.

## References

1. A. Aiken, J. Widom, J.M. Hellerstein. Behavior of Database Production Rules : Termination, Confluence and Observable Determinism. In *Proc. Int'l Conf. on Management of Data (SIGMOD)*, San Diego, California, 1992.

2. J. Bailey, G. Dong, K. Ramamohanarao. Decidability and Undecidability Results for the Termination Problem of Active Database Rules. In *Proc. ACM Symposium on Principles of Database Systems (PODS)*, Seattle, Washington, 1998.
3. E. Baralis, S. Ceri, S. Paraboschi. Improved Rule Analysis by Means of Triggering and Activation Graphs. In *Proc. Int'l Workshop Rules in Database Systems (RIDS)*, Athens, Greece, 1995.
4. E. Baralis, J. Widom. An Algebraic Approach to Rule Analysis in Expert Database Systems. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, Santiago, Chile, 1994.
5. S. Ceri, J. Widom. Deriving Production Rules for Constraint Maintenance. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, Brisbane, Queensland, Australia, 1990.
6. S. Chakravarthy, D. Mishra. Snoop : An Expressive Event Specification Language for Active Databases. In *Data and Knowledge Engineering*, 14 , 1994.
7. U. Dayal, A.P. Buchmann, D.R. Mc Carthy. Rules are Objects too : a Knowledge Model for an Active Object Oriented Database System. In *Proc. Int'l Workshop on Object-Oriented Database Systems*, Bad Münster am Stein-Ebernburg, FRG, 1988.
8. A.P. Karadimce, S.D. Urban. Refined Triggering Graphs : a Logic-Based Approach to Termination Analysis in an Active Object-Oriented Database. In *Proc. Int'l Conf. on Data Engineering (ICDE)*, New-Orleans, Louisiana, 1996.
9. S.Y. Lee, T.W. Ling. Refined Termination Decision in Active Databases. In *Proc. Int'l Conf. on Database and Expert Systems Applications (DEXA)*, Toulouse, France, 1997
10. S.Y. Lee, T.W. Ling. A Path Removing Technique for Detecting Trigger Termination. In *Proc. Int'l Conf. on Extending Database Technology (EDBT)*, Valencia, Spain, 1998.
11. S.Y. Lee, T.W. Ling. Unrolling Cycle to Decide Trigger Termination. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, Edinburgh, Scotland, 1999.
12. M.K. Tschudi, S.D. Urban, S.W. Dietrich, A.P. Karadimce. An Implementation and Evaluation of the Refined Triggering Graph Method for Active Rule Termination Analysis. In *Proc. Int'l Workshop on Rules in Database Systems*, Skoevde, Sweden, 1997.
13. A. Vaduva, S. Gatzju, K.R. Dittrich. Investigating Termination in Active Database Systems with Expressive Rule Languages. In *Proc. Int'l Workshop on Rules in Database Systems*, Skoevde, Sweden, 1997.
14. T. Weik, A. Heuer. An Algorithm for the Analysis of Termination of Large Trigger Sets in an OODBMS. In *Proc. Int'l Workshop on Active and Real-Time Databases* Skoevde, Sweden, 1995.