TERMINATION ANALYSIS IN ACTIVE DATABASES BY USING EVOLUTION GRAPHS

Hany Harb
Faculty of Engineering, El-Azhar University, Egypt
Hamdy Kelash
Ahmed Shehata am_shehata@yahoo.com
Faculty of Engineering, Menoufia University, Egypt

Abstract: We introduce a method for rule termination analysis within active databases by using evolution graphs which simulating rule processing statically and considering both rule activation and deactivation. The evolution graph provides a more detailed analysis than activation and deactivation graphs. We propose a modification for a previous algorithm that can detect some cases of non-termination. This modification can detect more cases by selecting next rule for execution when more than one rule is ready to run regardless to the priority which is the main factor in the previous algorithm. A new selection algorithm can detect more cases that can be terminated that can not be detected by the previous algorithm.

1. INTRODUCTION

An active database system is a DBMS endowed with active rules. An active rule consists of three parts: the event part, which specifies a list of events, the condition part, which is a query on the database, and the action part, which generally consists of one or more updates or queries on the database [1]. Rules in active database systems can be very difficult to program, due to the unstructured and unpredictable nature of rule processing. There are a lot of static analysis techniques for predicting useful properties for active rule sets. During rule processing, rules can interact in complex and sometimes unpredictable ways: they may trigger and "untrigger" each other, and the intermediate and final states of the database can depend upon which

rules are triggered and executed in which order. Active rule analysis is aimed at defining compile-time techniques that allow a rule designer to predict in advance important aspects of rule behavior such as termination of rule execution. These techniques are used to statically analyze a set of rules before installing them in the database. Thus, static rule analysis can provide a fundamental building block for both a design methodology and programming environment for the development of active rule application. There are static methods using only some of the active rules properties like starburst, which use only the triggering property, and chimera, in which triggering and activation are considered. In this algorithm we use all the properties of the active rules triggering, activation, and deactivation. The remainder of this paper is organized as follows. In section 2, some of the related work which are concerning to the non-termination problem. In section 3, basic concepts for understanding selection algorithm is explained. In section 4, we introduce selection algorithm and an example to explain the selection algorithm and applying it and compare with the algorithm in [2]. Finally, in section 5, we draw some conclusions.

2. RELATED WORK

Several methods have been proposed to perform compile–time termination analysis. Many techniques such as [3, 4, and 5] are based on building a triggering graph by considering the type of triggering events and of events generated by the execution of rule actions. Nontermination may occur if the graph contains cycles. Other techniques [6, 3, and 7] rely mostly on the semantic information contained in the rule condition and actions, using which an Activation Graph is built. Cycles are then searched for in this graph. A synthesis of the two techniques is proposed in [4] which allow the discarding of some "false" cycles that are detected by the former methods. Termination analysis based on triggering and activation graphs is discussed later. When a ordering is defined on a rule set, the analysis technique may be improved by [4, 8] as described later.

3. BASIC CONCEPTS

In this section we will define some of important basics to describe the evolution graphs and abstract states which are main parts in our algorithm.

3.1 Graphs

We consider three different relations among active rules. Let r_i and r_j be two rules then:

- r_i triggers r_j, if one of the r_i's actions has the corresponding event in r_j;

- r_i activates r_j , if the possible execution of the r_i 's actions makes r_j 's condition true;
- r_i deactivates r_j , if the possible execution of the r_i 's actions makes r_j 's condition false.

These relations can be described with three directed graphs denoted as Triggering Graph (TG), Activation Graph (AG) and Deactivation Graph (DG), respectively. Nodes represent rules and arcs stand for the specific relations among rules.

3.2 Abstract State

Now we introduce the notion of rule abstract state describing rule characteristics, such as whether the rule is triggered, activated or deactivated or relevant combinations of these relations. We only want to determinate a way to characterize a rule in a certain possible execution point. In this way we can establish the relevant abstract computational states with respect to termination. Let as be a function that for each rule describes its abstract state. Every rule r_i of an active program can be in one of the four following abstract states as(r_i):

- as(r_i) = s_i denotes that r_i is triggered and activated;
- as(r_i) = s'_i denotes that r_i is triggered and deactivated;
- as(r_i) = s^a_i denotes that r_i i is activated and not triggered;
- as(r_i) = s^n_i denotes that r_i is deactivated and not triggered.

An abstract state can be changed by update execution (i.e. rule action or user update operation).We can calculate the new configuration of r_i according to the following rules [9, 13]:

- if as(r_i)= s^a_i and r_j deactivates r_i then as(r_i) will be = s^n_i .
- if as(r_i)= s^a_i and r_j triggers r_i then as(r_i) will be = s_i .
- if as(r_i)= s^n_i and r_j triggers r_i then as(r_i) will be = s'_i .
- if as(r_i)= s^n_i and r_j activates r_i then as(r_i) will be = s^a_i .
- if r_j activates or triggers a rule $r_i \in AR_k$ such that s(r_i)$\notin S_1$, then we assume that as(r_i) = s^a_i. as(r_i) is updated as above described and inserted in S_1.

3.3 Evolution Graph

In order to introduce the EG it is necessary to define two subsets of the active rules set forming P(P is an active program). Let C_i be a cycle in P. Let $AR_i \subseteq (P \backslash C_i)$ and $DR_i \subseteq (P \backslash C_i)$ are two subsets of P's rules that are not part of the cycle C_i. The rules in AR_i (Activation Rules) have no action A such that the execution of A deactivates a rule of C_i. That is, there is no arc in DG between any rule of AR_i and any rule of C_i. On the contrary rules in DR_i (Deactivation Rules) have an action A such that the execution of A deactivates at least a rule of C_i. That is to say there is at least an arc in DG between each rule of DR_i and at least a rule of C_i. Now we define the EG

which allows us to graphically visualize the cycle execution. We start with the definition of two sets. Let P be an active program and S be an abstract states set such that $S \subseteq as(C_i \cup AR_i \cup DR_i)$. Let R be a subset of P: R is the rule set for which the simulated execution is postponed when more than one rule is eligible for execution that is when more than one rule can be executed. R is empty whenever there is only one rule at a time that is eligible for execution (i.e. is triggered and activated). Together S and R will form a node of the Evolution Graph as shown below.

Definition of Evolution Graph:

Let P be an active program and let $C_i \subseteq P$ be a cycle. The Evolution Graph (EG) is a direct labeled graph <N, D, Φ > where N is the set of nodes, D the set of arcs and Φ the labeling function. Each node $N_i \in N$ is a pair: N_i = <S, R> where S and R are defined above.

An arc from node N_i to node N_{i+1} of an EG, specifies the abstract state changes operated by the execution of the triggered and activated rule of N_i. Each arc is labeled through Φ with the rule for which we simulate the execution statically.

For each cycle C_i, a number n_i of EG has to be generated where n_i is the number of rules for cycle C_i. This is due by the fact that we do not know from which rule of the cycle the execution may start. So we must consider in our simulation all possible cases. Therefore the starting node of an EG is just composed by the abstract states of C_i's rules, where one rule alone is triggered and activated while the others are only activated. This is the worst case. Subsequent triggering may cause non-terminating execution of the cycle. A rule can not be activated and deactivated at the same time from the same rule, because we examine only the final effect of the execution about the rule action part.

4. OUR APPROACH

In this section we will explain the Selection Algorithm that used in the termination analysis of the rules set.

- **Selection Algorithm:**
  - Create the TG of P: if there are not any cycles in TG, then termination is assured based on the results of [11].
  - If there are cycles in TG, then it is necessary to create both the AG and DG to exploit further information contained in the active rules. Assume that $\{C_1, \ldots, C_n\}$ are the TG's cycles in P. Let n_i denotes the number of rules involved in cycle C_i. At this point for each C_i we create n_i Evolution Graphs (EG), one for each rule in the cycle C_i.
  - If there are more than one rule eligible for execution (activated and triggered) then select one of these rules according to:

- The rule that will trigger a smaller number of rules according to TG.
- If there is a number of rules that will trigger equal number of rules in TG, select one that activates small number of rules according to AG.
- If there is a number of rules that will activate equal number of rules, select one that deactivates large number of rules according to DG.

– The creation of EGs ended when the new node is obtained before (cyclic) or there are not any rules eligible for execution.
– If all EGs are acyclic, then the execution of P will terminate; otherwise rules set may be non-terminated.

Now we try to explain our Selection Algorithm through a number of examples. First we consider a simple rule language taken from Starburst [10]. The syntax to define a rule is:

- **create** rule *name* on *table*
- **when** *triggering operation*
- [**if** *condition*]
- **then** *action*
- **[precedes** *rules-list*]
- [**follows** *rules-list*]

To simplify the example we consider as actions numerical values assignments (just 0 and 1) to the database table attributes, implemented with the *set* command.

Example 1:

Consider a simple database schema with three tables: R(A,B,C,D), S(E,F,G) T(H, I, L, M). The rules are defined in rules set 1. The relations among rules are three: Triggering, Activation and Deactivation. Detect the termination property for that rules set.

By applying first step in Selection Algorithm we obtain figure 1 which has a cycle $\{r_1, r_5\}$; it means the termination can not be assured. So it is necessary to draw both AG and DG as in figures 2, 3.

create rule r_1 on R	create rule r_2 on S	create rule r_3 on R
when *update* (D)	when *update* (E)	when *update* (D)
if (T.I=1)	if (S.F=1)	if (T.H=1)
then *update* T	then *update* T	then *update* T
set G=1	*set* I=1	*set* L=1
set M=1	*update* S	*set* I=0
update S	*set* G=0	*update* S
set E=1	**precedes** r_1, r_3, r_4, r_5	*set* E=1

follows r_2 ,r_3 , r_4 , r_5	**follows** r_1 , r_5 **precedes** r_2 ,r_4
create rule r4 on R when update (C) if (T.L=1) then update R set L=0 update S set G=0 **precedes** r_1 , r_3 , r_5 **follows** r_2	create rule r_5 on T when *update* (M) if (S.G=1) then *update* S *set* F=1 *update* R *set* D=1 *set* C=1 **precedes** r_1 **follows** r_2 , r_3 ,r_4

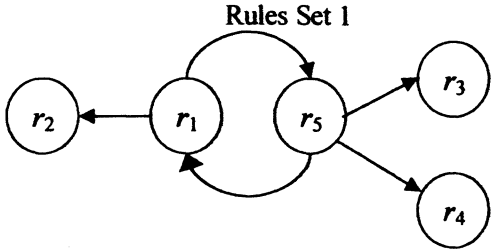

Figure 1. TG of the Rules set 1

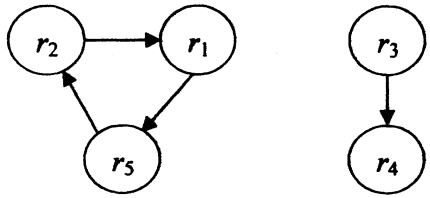

Figure 2. AG of the Rules Set 1

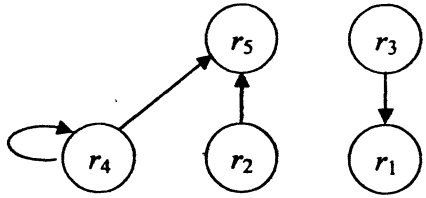

Figure 3. DG of the Rules Set 1

Now we will draw the EG for each rule in the cycle, in this example we have two EGs. Figure 4a simulates the execution of the rule r_1.

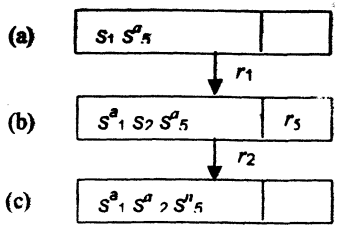

Figure 4. EG Starting from r_1 by Applying Selection Algorithm

Figure 4b has two rules that are ready for running (r_2,r_5). By applying step 3a of the selection algorithm, r_2 is selected for execution and rule r_5 is postponed. Figure 4c is a final state because there are not any rules eligible for execution. So, we can say that EG are acyclic and the termination is guaranteed.

Let's try to apply the algorithm in [2] on the same example. In [2] execution of rules depends on the priority of the execution so, rule r_5 run before r_2 where r_5 has priority higher than r_2 as in figure 5b.

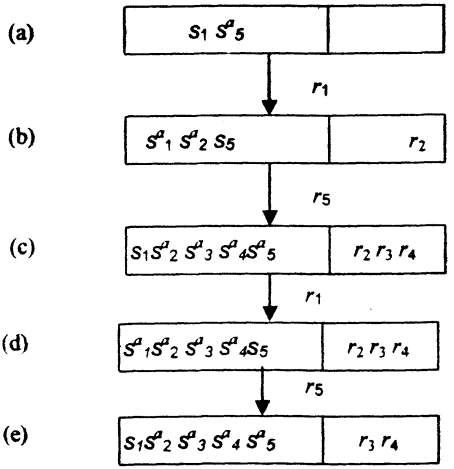

Figure 5. EG Starting from r_1 Applying Algorithm in [2]

Similarly rules r_1, r_2, r_3 and r_4 are ready for run as in figure 5c but according to priority r_1 run first. EG in figure 5e is obtained previously so this lead to cyclic situation and the termination property can not be guaranteed.

Now we will apply our selection algorithm starting from rule r_5. Termination is guaranteed in our algorithm as depicted in figure 6d where it is a final state but termination in the other algorithm [2] is not guaranteed as shown in figure 7d where this state is obtained before.

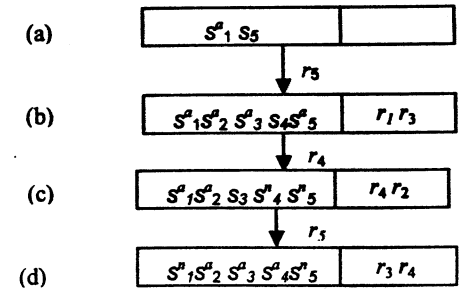

Figure 6. EG Starting from r_5 by Applying Selection Algorithm

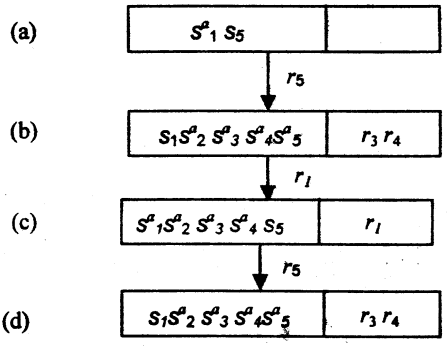

Figure 7. EG Starting from r_5 by Applying Algorithm in [2]

For more explaining of our approach, let's take another example.

Example 2:

Let the Triggering Graph, Activation Graph and Deactivation Graph of a rule set explained by the figures 8, 9 and 10.

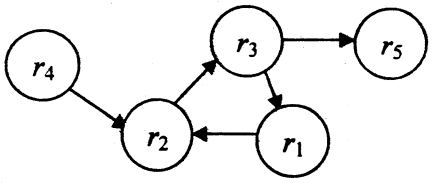

Figure 8. Triggering Graph of Example 2

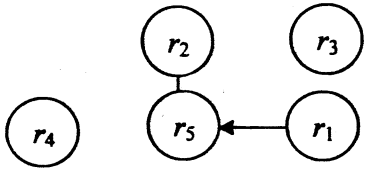

Figure 9. Activation Graph of Example 2

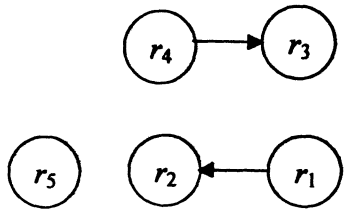

Figure 10. Deactivation Graph of Example 2

It is clear that there is a cycle c1= $\{r_1, r_2, r_3\}$ there may be a nontermination problem. We will apply the algorithm in [2] (apply rules by their priorities) the result is shown in figure 11.

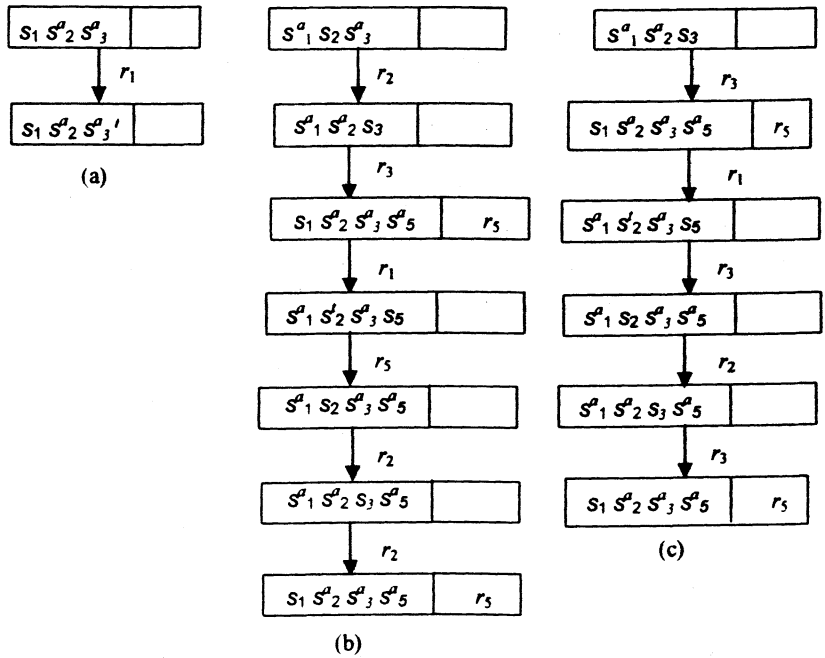

Figure 11. EG for Rules in Example 2 Applying Algorithm in [2]

Although the termination is guaranteed in figure 11a, it is not guaranteed in figures 11b, 11c. The last node in figures 11b, 11c is obtained before and this will lead to a cyclic state. So we can not assure that the rule set will terminate.

Figure 12 shows the states of the EG when the selection algorithm is applied. Figure 12a gives similar state as in figure 11a. But figures 12b and 12c give EGs that assure termination.

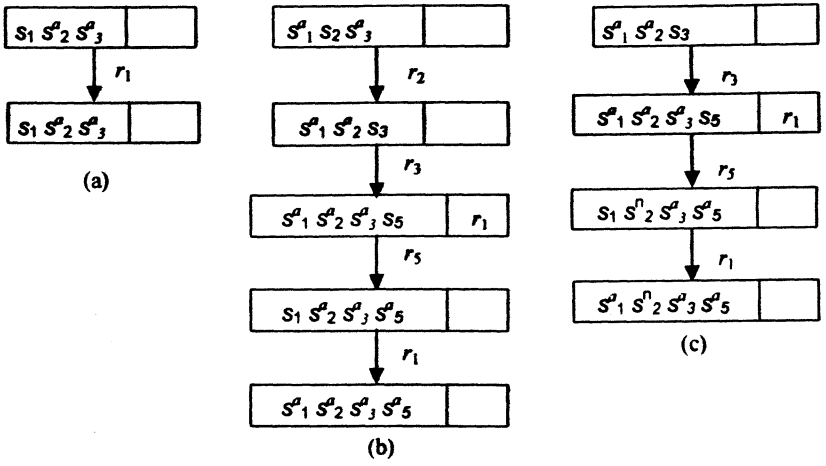

Figure 12. EG for Rules in Example 2 by Applying Selection Algorithm

From figure 12 the rules in the EG for rules are acyclic it means that the rules set of example 2 are terminated. This result can not be reached by applying algorithm in [2]. We can point out from the results obtained from applying selection algorithm to examples 1 and 2 that our selection algorithm can detect states more than of the algorithm in [2].

It is important to point out that assuming known the relation among rules our method can be applied in other context with different rule languages considering object manipulation, external events, methods invocation etc. So even for rules defined in [12], it is possible to use this algorithm to determine the execution termination of a rules set. A closer comparison, however, is not possible due to the difference of rule languages involved in such different database systems.

5. CONCLUSION

We have presented a Selection Algorithm for static rule termination analysis. This method can be applied to existing active rule languages without particular restrictions. It is based both on the notion of abstract state and evolution graph which encodes the simulated execution of triggering, activation and deactivation relations among rules. We have shown that we can capture a large class of terminating rules because of the use of deactivation relations. We have also presented an algorithm to check terminations. This algorithm has a good modification in algorithm presented in [2] and that leads to know a large number of cases that can be terminated which can not be captured by algorithm in [2].

6. REFERENCES

[1] S. Comai, L. Tanca. "Termination and Confluence by rule prioritization", In IEEE Transactions on Knowledge and Data Engineering, vol. 15, no. 2, 2003.

[2] D. Montesi, M. Bagnato and C. Dallera. "Termination Analysis in Active Databases", In Proc. IDEAS'99, IEEE Computer Society Press, Montreal, 1999.

[3] A. Aiken, J.M. Hellerstein, J. Widom. "Static Analysis Techniques for Predicting the Behavior of Database Production Rules", ACM TODS, pp. 3-41, 1995.

[4] E. Baralis, S. Ceri, S. Paraboschi. "Improved Rule Analysis by Means of Triggering and Activation Graphs", RIDS'95, Lecture Notes in Computer Science, pp. 165-181, 1995.

[5] A. P. Karadimce and S. D. Urban, "Refined Trigger Graphs: A Logic-Based Approach to Termination Analysis in an Active Object-Oriented Database", ICDE'96, 384-391.

[6] S. Ceri, J. Widom. "Deriving Production Rules for Constraint Maintenance", In Proc. Int'l Conf. on Very Large Databases (VLDB), Brisbane, Queensland, Australia, 1990.

[7] Elena Baralis and J. Widom. "An Algebraic Approach to Rule Analysis in Expert Database systems", In Proc. Twentieth intl. Conf. on VLDB, pages 475-486, Santiago, Chile, September 1994.

[8] Detlef Zimmer, Axel Meckenstock, and Rainer Unland. "Using Petri Nets for rule Termination Analysis", In proc. of Workshop on Databases: Active and Real-Time, Rockville, Maryland, November 1996.

[9] D. Montesi, E. Bertino, M. Bagnato, "Rules Termination Analysis Investigating the Interaction Between Transactions and Triggers", IDEAS'02, IEEE Computer Society Press, Silver Spring, MD, 2002.

[10] J. Widom. The Starburst Rule System: "Language Design, Implementation, and Applications" IEEE Data Engineering Bulletin, pp. 15-18, 1992.

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

[12] W. Dietrich, A. P. Karadimce, M. K. Tschudi, S. D. Urban. "An Implementation and Evaluation of the Refined Triggering Graph Method for Active Rule Termination Analysis", RIDS '97, Lecture Notes in Computer Science, pp. 133-148, 1997.

[13] D. Montesi, E. Bertino, M. Bagnato, "Refined rules termination analysis through transactions". IDEAS'02, IEEE Computer Society Press, Silver Spring, MD, 2002.