Journal of Engineering Sciences, Assiut University, Vol. 35, No.1, pp.163-176, January 2007

STATIC TOOL ENVIRONMENT FOR RULE TERMINATION ANALYSIS BY REFINED EVOLUTION GRAPHS

***Hany Harb**[1], **Hamdy Kelash**[2] **and Ahmed Shehata**[3]*

[1] *Faculty of Engineering, El-Azhar University, Egypt*
[2] *Faculty of Electronic Engineering, Menouf, 32952, Egypt*
am_shehata@yahoo.com

(Received October 12, 2006 Accepted December 7, 2006)

A new algorithm for static rule termination analysis within active databases is introduced. This algorithm uses evolution graphs which simulating rule processing statically and considering both rule activation and deactivation. This algorithm has been refined for some cases that cannot be assured of termination this refinement by using Refined Evolution Graphs and analyzing transactions and triggers. A Static Tool Environment is proposed that can be used in termination analysis algorithm. The Static tool proposed is suitable for this algorithm where its component can execute the proposed algorithm. We show that several termination analysis algorithms are captured with our algorithm. The proposed algorithm tests rule termination is presented considering deferred and detached executions. The proposed algorithm turns out to be practical and general with respect to various rules languages and thus it may be applied to many databases.

KEY WORDS: *Active database, Static tools, transactions, evolution graphs, triggers, nontermination analysis*

1. INTRODUCTION

Active databases have been the focus of several researches aiming to extend the functionality of traditional (passive) databases [1, 2]. They generally use active rules to express their active behavior. Active rules have a tripartite structure, called event condition action (ECA). An important issue in designing a set of rules is to generate rules for which it is possible to guarantee the execution termination. In fact, rules can trigger each other infinitely. This behavior is not acceptable, and thus many commercial systems impose severe restrictions to prevent non-terminating rules execution.

There are three ways to approach termination. A common way (supported by commercial products) is to handle termination only during rule execution. That means the system assumes that an infinitely cascading triggering takes place if a specific upper bound of triggered rules has been reached and thus it stops rule execution. The drawbacks are that counters may either cause abrupt ending even if rule processing would have terminated on its own in a few more steps or loops are let to execute for too long time before the system can halt them. This approach handles the effects but cannot remove the causes of infinite triggering [3].

A better way to handle termination for a given set of rules is by means of *termination analysis*, which aims to detect rule subsets that may potentially lead to infinite triggering of rules. If such a rule subset exists, the rule developer has to modify the specification of some rules and investigate termination again. The iterative process is performed until termination analysis guarantees that the rule set will never manifest a nonterminating rule behavior [4].

Termination analysis may be *static* or *dynamic*. Static analysis investigates rule definitions at compile time in order to determine subsets consisting of rules that may trigger (directly or indirectly) each other. When no such rule subsets are found, termination of rule execution is guaranteed for the given rule set. Even if static analysis is sometimes too conservative in relation to the actual behavior, it never yields unreliable information. It guarantees termination in all possible situations. In contrast, dynamic methods examine rule behavior at runtime. Until now, they could not guarantee termination for all possible combinations of initial database states and event occurrences, no matter if applied before or during the actual use of the active DBMS. Thus, dynamic analysis cannot substitute; at most it can complement or refine static analysis methods [5].

In this paper, a new analysis method that builds over the analysis of three relations among rules has been presented that can be statically checked. These relations are triggering, activation and deactivation. There are several static and runtime approaches for active rules termination analysis. We consider only static methods that at compile time detect potential non-terminating active rules execution. Furthermore, no method uses information provided by updates used to form transactions. There are static methods using only some of the active rules properties like Starburst [6, 7], which use only the triggering property, and Chimera [8, 9], in which triggering and activation are considered but deactivation is used only in restricted cases (a rule cannot deactivate other rules, but only itself). Our aim is to propose a static analysis method that considers how user transaction information and structures can affect active rule execution termination. The proposed method improves a technique, introduced in [10]. This new method (refined method) considers the analysis of three relationships among rules (triggering, activation and deactivation) that can be statically checked. Furthermore the properties of updates forming transactions that trigger active rule processing are used to enrich analysis. Refined method is simple and easy to understand, but powerful and flexible enough to be used with current active database languages. This is possible because it investigates relationships among rules and transaction updates that are independent of the specific rule language.

The remainder of this paper is organized as follows. In section 2 ECA Rules in SAMOS are presented. An overview of our approach and some basic concepts that includes graphs and abstract states for understanding our approach are explained in Section 3. Rule termination analysis using transactions and our algorithm are introduced in section 4. The proposed Static Tool Environment and its component and how it is suitable for SRTA algorithm are presented in section 5. An example for explaining our algorithm is introduced in section 6. Finally conclusions are introduced in section 7.

2. ECA RULES IN SAMOS

A rule definition language is used for the specification of the Event/Condition/Action (ECA) rules. Each rule consists of an event, a condition and an action. An event is an occurrence the active database system must react to. When an event occurs, all respective rules are triggered. If the condition of a triggered rule is fulfilled then its action is executed. A condition may be a predicate on the database state or a database query. If the result of evaluation is true or non-empty, respectively, the condition is satisfied. The action specifies the reactive behavior of the rule. It may contain data modification and retrieval (in relational DBMS), transaction operations like commit or abort, etc.

3. TERMINATION OF RULE EXECUTION

This section includes an overview of our approach and the basic concepts of the proposed algorithm.

3.1. Our Approach

In our approach we consider three relations among rules: triggering, activation, and deactivation. Informally, there is a triggering relation between two rules if one of the rule's actions has the corresponding event in the other rule. There is an activation relation, if the possible execution of the rule's actions makes the other rule's condition true. Finally, there is a deactivation relation, if the possible execution of the rule's actions makes the other rule's condition false. A triggered and activated rule is considered potentially executable (or eligible for execution). We represent the fact that a rule may deactivate other rules by considering it in conjunction with rules triggering and activation through a new graph: the evolution graph (EG). We simulate rule execution using EGs that store information about the simulated execution in abstract states. Abstract state models the fact that for termination analysis we are interested to observe states where a rule is triggered, activated, deactivated both triggered and activated, or neither triggered nor activated. A similar structure, called refined evolution graph (REG), is defined when we consider in the analysis also transactions. Providing termination for a specific transaction, although allowing capturing more termination cases, is weaker than providing termination in the general case. Indeed, a rule set could be terminating under a transaction currently defined for the system, but be not terminating for another transaction.

3.2. Basic Concepts

The basic concepts needed for explanation of our algorithm.

3.2.1. Active Rules and Programs

The execution semantics describes the behavior of an active program. We say that when a rule is triggered (if several rules are triggered, a selection policy must be established), the system evaluates its condition. If the condition is satisfied, the rule is eliminated from the triggered rules set and its action is performed. If the condition is not satisfied, then there are two different ways to proceed: in the first case an *event-preserving* the system leave the rule in the group of the triggered rules set, and then have more than one triggered rule. The other the system eliminates the rule from the triggered rules set; and that is called an *event-consuming* [10]. In this paper we use event-preserving execution model.

Finally, a transaction $T = U_1, U_2, \dots, U_n$ is an atomic sequence of operations, that means a sequence of insertions, deletions and updates, such that all of them are executed or none has to be performed [11].

3.2.2. 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. Triggering relation was first introduced in [12]. It describes the rules mutual ability to “wake up” each other. The notion of cycle is defined starting from a TG. We define a cycle C in P as a subset C of P, such that the rules of C form a cycle in TG. We say that a rule is involved in the execution by a cycle C if the rule is activated, deactivated or triggered by the rules of C. The notion of activation relation was introduced in [13].

3.2.3. 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 abstract states as(r_i)= s_i , s^t_i , s^a_i or s^n_i

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 [6]:

if as(r_i)= s^a_i and r_i is deactivated then as(r_i) will be = s^n_i .

if as(r_i)= s^a_i and r_i is triggered then as(r_i) will be = s_i .

if as(r_i)= s^n_i and r_i is triggered then as(r_i) will be = s^t_i .

if as(r_i)= s^n_i and r_i is activated then as(r_i) will be = s^a_i .

4. RULE TERMINATION ANALYSIS USING TRANSACTIONS

In order to understand our algorithm, we must define the evolution graph and refined evolution graph that are explained in the following sections.

4. 1. Evolution Graph

Let P be an active program consists of a group of rules constraints the database operations. S be an abstract states to explain the state of each rule at this time; R be a subset of rules that 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 in Fig. 1.

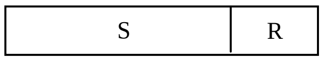


Fig. 1. Evolution Graph node

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 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 cannot 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. 2. Refined Evolution Graph

The EG considers only the rules forming a cycle over the TG. But the REG is used with transactions we do not consider all active rules, because we know which rules will be involved by transaction execution. Each node in the REG will be as in the EG.
An arc from node N_i to node N_{i+1} of REG specifies the abstract state changes resulting from the execution of the triggered and activated rule of N_i or from the execution of one of the updates in T. Each arc is labeled with the rule number or with the transaction update for which we simulate execution. The initial node of all possible REG (not depending on user transactions) is empty because no update has executed yet [6].
The creation of nodes in EG or REG stops when:

- The new node is obtained before (it finds a cycle while creating it). Or
- When a node has not any abstract state ready for running (it is acyclic state).

4. 3. Testing Termination

If an abstract state set presents itself again during a computation we cannot guarantee the termination. Using EG and REG this termination test can be implemented by checking the presence of a cycle. If the graph is acyclic we can assure the termination; otherwise, we leave the decision for the rule developer to improve the rule set.

4. 4. SRTA Algorithm

In this section we explain our Static Rule Termination Analysis (SRTA). The algorithm depicted in Fig. 2. This algorithm calls two functions; Build_EG that build the evolution graphs for each rule in a cycle Fig. 3. The other is Build_REG that builds the refined evolution graphs for each transaction Fig. 4. The two calling functions use a selection method to select the next rule for execution when more than one rule is eligible for execution according to their properties triggering, activation and deactivation.

```
ALGORITHM SRTA
BEGIN
    Draw the TG.
    IF there are not any cycles in TG,
    THEN termination guaranteed and EXIT.
    Draw the AG and DG.
    FOR EACH cycle:
        IF there is a rule that deactivated and not activated again
        THEN termination guaranteed and EXIT.
    FOR EACH rule $r_i$ in the cycle
        Build_EG( $r_i$).
        IF all EGs are acyclic
        THEN termination guaranteed and EXIT.
    FOR EACH update $U_i$ in each transaction
        Build_REG($U_i$).
        IF the REG is acyclic
        THEN termination guaranteed.
        ELSE termination is not guaranteed and designer improves the rule set.
END.
```

Fig. 2. SRTA Algorithm

```
FUNCTION Build_EG($r_i$)
BEGIN
    1. The abstract state for $r_i$ is $S_i$ other rules in the cycle are activated only $S^a$.
    2. Change the abstract states for each rule according to the simulation
       execution of the rule i.
    3. f there are more than one rule is eligible for execution (activated and
       triggered) then select one of these rules according to:
       a. The rule that will trigger a smaller number of rules according to TG.
       b. If there is a number of rules will trigger equal number of rules in TG,
          select one that activates small number of rules according to AG.
       c. If there is a number of rules will activate equal number of rules, select
          one that deactivates large number of rules according to DG.
    4. The creation of new nodes stops when the new node is obtained before
       (cyclic) or there are not any rules eligible for execution (acyclic).
END.
```

Fig. 3. Function Build_EG

```
FUNCTION Build_REG($U_i$)
BEGIN
    1. Draw an empty node.
    2. Update abstract state of the rule that affected by the update in the
       transaction, the arc from the empty node to new node is labeled with $U_i$ .
    3. Change the abstract states for each rule according to the simulation
       execution of the rule i . Assuming the abstract states for all rules at first
       appending in REG are $S^a{}_i$ and need triggering only to be eligible for
       execution, label the arc with the rule that executed.
    4. The creation of new nodes stops when the new node is obtained before
       (cyclic) or there are not any rules eligible for execution (acyclic). .
END
```

Fig. 4. Function Build_REG

5. STATIC TOOL ENVIRONMENT

Static Tool Environment (STE) is a proposed architecture can be built which is suitable for applying out SRTA algorithm. This architecture is coupled to the active object-oriented DBMS SAMOS [2] but the concepts are applicable to other active DBMS with similar functionality as well. The idea is to derive from the particular case of SAMOS features and implementation issues that should be considered in the construction of active rule development environments in general [14].

5. 1. Architecture

The STE(static tools environment) is strongly connected to the active DBMS prototype SAMOS. The architecture is a layered architecture see Fig. 5 consisting of ObjectStore and an active layer on top of it. These two layers are main components in SAMOS. The active layer includes rule execution component and a rule manager. The rule manager is responsible for the persistent storage and retrieval of information about event and rule definitions. Events, conditions, actions and rules are represented as objects. The Static Tool Environment may be considered as another layer that wraps the kernel of SAMOS. However, the developer makes use of the STE during application development. Next sections will explain each component of STE layer.

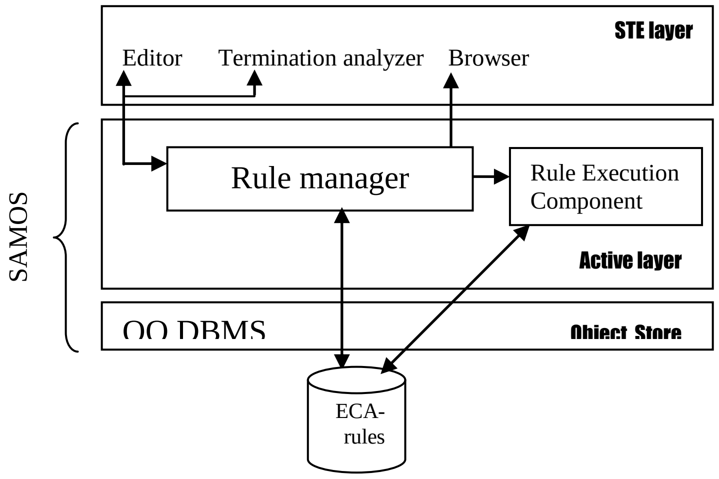


Fig. 5. The integration of STE with the active DBMS SAMOS

5. 2. STE layer

We have STE layer that include an editor; a browser and a termination analyzer, which are the main parts of the STE. They are used before the execution of applications, during the specification and design of Active Database Systems. The STE is strongly connected to the active DBMS prototype SAMOS. The browser and the editor are graphical interfaces for easily inserting and retrieving rule definitions from the rulebase. The termination analyzer is responsible for detecting rules that may generate nontermination as a consequence of their interactions. We will explain each part in more detail.

Browser

The task of the browser is to support the navigation through the rulebase. The required information is provided by the rule manager, which makes use of the retrieval facilities of the underlying DBMS ObjectStore for querying the rulebase. The returned information may include:

- Lists of event, condition, action and rule definitions.
- Lists of rules that an event will be triggered by that action.
- Lists of rules that their conditions truth values turned to be true and will be activated.
- Lists of rules that their conditions truth values turned to be false and will be deactivated.

Therefore, the browser allows the inspection of the rulebase and may easily return various information about rules themselves and dependencies between rules and their components.
The browser consists of four blocks: event, condition, action and rule browser. These blocks are strongly coupled so that the control easily flows between them when the browser is in use. The editor and the browser have been implemented by making a user-friendly interface.

Editor

The editor offers a graphical interface for the definition, updating and deletion of rules and their constituent parts. The editor translates the input in the rule language of SAMOS. Then, the rule compiler is invoked. This, in turn, makes use of the rule manager that finally stores the new data in the rulebase. The rule compiler is responsible for the syntactic and partly semantic analysis of rule definitions. Its inputs are event and rule definitions with the syntax of the rule definition language available from the editor. The syntax and semantic analysis performed by the rule compiler include the parsing of actions and conditions. If the syntactic and semantic analyses have been successful, the compiler uses the interface offered by the rule manager to create corresponding objects that are persistently stored in the rulebase.

Termination analyzer

The architecture of the termination analyzer is more complex. Fig. 6 explains the main components in that part of the STE. It contains components that are responsible for the steps to be performed as described by our approach for termination analysis.

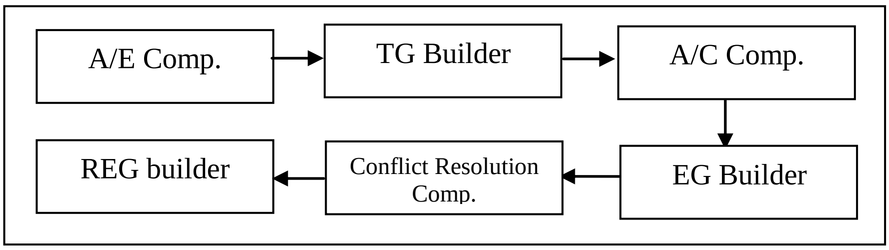


Fig. 6. The Termination Analyzer

The A/E relationship component: performs the syntactic analysis for each action and the related events will be signaled by it. The information derived from the syntactic analysis is persistently stored. Thus, it may be accessed and displayed using the browser. The output of the A/E component is passed to the triggering graph builder where by it we draw the triggering graph and determines the cycles that may be not terminated. These cycles are passed to the A/C relationship component. That detects each action of each rule in the cycle to know its effect on the conditions of other rules. If the execution of the action of a rule modifies the truth value of a condition of another rule to true then the first rule activate the other, if the truth value is modified to false then this rule is deactivated by the first rule. Cycles are false if there is a rule in the cycle deactivated and not activated again. The correct cycles are passed to Evolution Graph Builder (EG Builder). The EG Builder builds Evolution Graphs, one for each rule in the cycle. When more than one rule is eligible for execution, the conflict resolution part will select next rule for trying to guarantee termination of cycle. If the termination is not guaranteed the REG builder will be used to build the REGs according to the group of transactions requested by the database application and finally gives a report of the transactions that will be terminated.

6. USING THE STE

For illustrating the functionality of the STE, how our SRTA algorithm can be applied on it. We consider a banking application in the credit department that concerns with some of operations can be done for customers and the rules restrict these operations. Five active rules are defined and the coupling mode of the rules are "deferred".

Example:

Rule R_1: ON decrease _overdraft
IF account_type =Normal
DO decrease_capacity(2000)
Rule R_2: ON decrease_capacity
IF amount <1000
DO decrease rate(1)
Rule R_3: ON decrease_rate
IF account_type =Normal
DO decrease _overdraft(20)
Rule R_4: ON increase_amount
IF account_type= important
DO decrease_capacity(1000)
AND increase_amount(200)
Rule R_5: ON decrease _overdraft
IF capacity < 5000
DO decrease_amount(500).

According to our algorithm, first we start by drawing the triggering graph (TG) of these rules as in Fig. 7.
There is a cycle $C_1 = \{r_1, r_2, r_3\}$ so the termination of rules can not be guaranteed as in the algorithm. The next step is to draw the activation (AG) and deactivation graphs (DG) as in Figures 8, 9.
There is a rule r_2 in the cycle C_1 deactivated by rule r_4 but it is activated again by r_5 so and then the termination of rules cannot be guaranteed as in the algorithm.

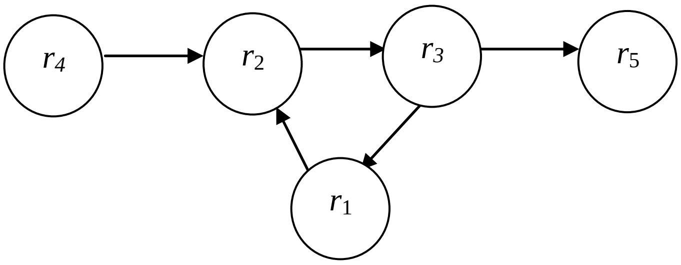
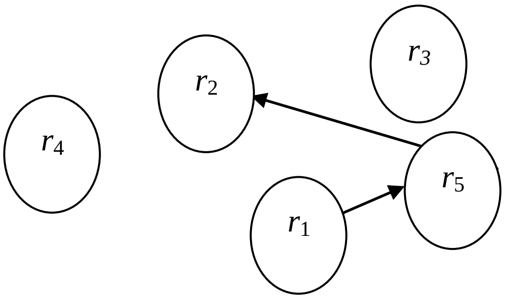


Fig. 7. TG of banking example



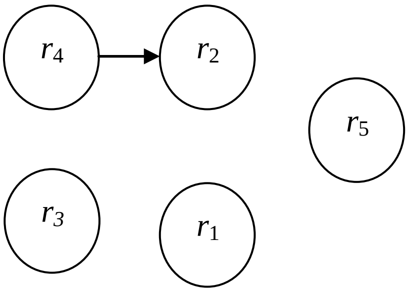
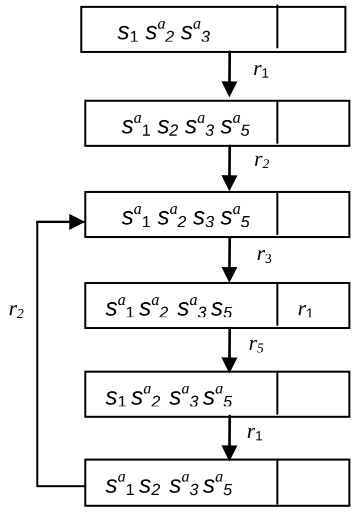


Fig. 8. AG of banking example **Fig. 9. DG of banking example**

Now we draw the evolution graphs (EGs) for each rule in the cycle as in the Build_EG function. Fig. 10 explains the EG of the simulated execution starting by r_1. The last node in Fig. 10 is obtained before so the EG is cyclic and the termination property is not guaranteed. Now the next step of the algorithm that uses the transaction will be used. We draw the refined evolution graphs for each transaction. If we have two transactions: T_1: decrease_capacity and T_2: increase_amount.

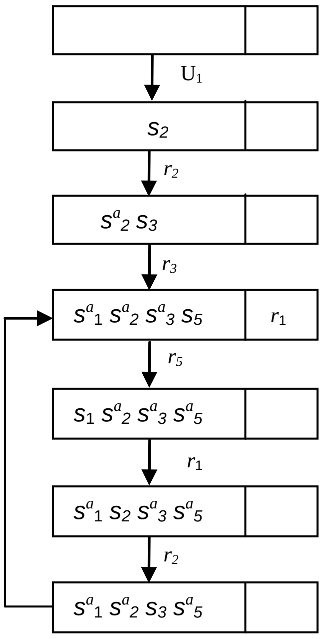
T_1 has one update U_1 that triggers r_2 and T_2 has one update U_2 that triggers r_4. According to Build_REG we draw REG for each update in each trigger.

Fig. 11 shows REG of applying U_1 the creation of new REG nodes is stopped where the last node in Fig. 11 is obtained and it gives a cyclic situation that cannot guarantee termination. Whereas in Fig. 12 U_2 is applied the REG obtained is acyclic and the creation of new nodes is stopped where there is not any rules are eligible for execution. So the termination is guaranteed for transaction T_2 only.



$s_1\ s^a_2\ s^a_3$
r_1
$s^a_1\ s_2\ s^a_3\ s^a_5$
r_2
$s^a_1\ s^a_2\ s_3\ s^a_5$
r_3
$s^a_1\ s^a_2\ s^a_3\ s_5$
r_1
r_5
$s_1\ s^a_2\ s^a_3\ s^a_5$
r_1
$s^a_1\ s_2\ s^a_3\ s^a_5$
r_2

Fig. 10. EG of banking example starting by r_1



U_1
s_2
r_2
$s^a_2\ s_3$
r_3
$s^a_1\ s^a_2\ s^a_3\ s_5$
r_1
r_5
$s_1\ s^a_2\ s^a_3\ s^a_5$
r_1
$s^a_1\ s_2\ s^a_3\ s^a_5$
r_2
$s^a_1\ s^a_2\ s_3\ s^a_5$

Fig. 11. REG for U_1 in transaction T_1

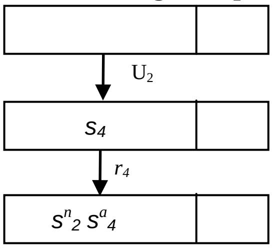


Fig. 12. REG for U_2 in transaction T_2

7. CONCLUSIONS

We have presented a SRTA (Static Rule Termination Analysis) that used for capturing termination in active database systems. This algorithm can be applied to existing active rule languages without particular restrictions. It is based on the notion of abstract state, evolution and refined evolution graphs which encode the simulated execution of triggering, activation and deactivation relations among rules. A new System (STE) that based on SAMOS is proposed that is suitable for our algorithm. The termination analyzer is the main part that execute our approach SRTA. We have shown that we can capture a large class of terminating rules by evolution graphs when next rule for execution is selected as in [15]. The refined evolution graphs and transactions are used when the termination cannot be guaranteed. This algorithm can be considered as a specific algorithm where it can guarantee termination only for those transactions. SRTA algorithm uses the deferred execution semantic where a rule action can be postponed. By that more termination cases can be detected than that detected in the immediate execution as in [5, 13].

REFERENCES

[1] J. Widom, S. Ceri, Active Database Systems: Triggers and Rules for Advanced Database Processing; Morgan-Kaufmann, San Francisco, California 96.

[2] Norman W. Paton, *Active Rules in Database Systems*, Springer-Verlag New York 1999.

[3] A. Vaduva, S. Gatziu, and K. R. Dittrich, "Investigating Termination in Active Database Systems with Expressive Rule Languages," Proc. of 3rd Intl. Workshop on Rules in Database Systems, RIDS 97, Skovde, Sweden, June 97.

[4] S. Gatziu, and K. R. Dittrich, "Detecting Composite Events in Active Database Systems Using Petri Nets," Proc. of the 4th Intl. Workshop on Research Issues in Data Engineering: Active Database Systems, RIDE-ADS Houston, 94.

[5] D. Montesi, E. Bertino, M. Bagnato, "Refined rules termination analysis through transactions," In proceeding of the information systems 28, 2003, pp. 435-456.

[6] A. Aiken, J.M. Hellerstein, J. Widom, "Static analysis techniques for predicting the behavior of database production rules," ACM Transactions on Database Systems, 1995, pp. 3–41.

[7] J. Widom, "The starburst rule system: language design, implementation, and applications," IEEE Data Engineering Bulletin (1992), pp. 15–18.

[8] E. Baralis, S. Ceri, S. Paraboschi, "Improved rule analysis by means of triggering and activation graphs," RIDS'95, Lecture Notes in Computer Science, Springer, Berlin, 1995,pp. 165–181.

[9] 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, Nov. 1996.

[10] D. Montesi, M. Bagnato, C. Dallera, "A Termination Analysis in Active databases," In Proceedings of the IDEAS'99, IEEE Computer Society Press, Montreal, 1999.

[11] J. Gray, A. Reuter, Transaction Processing: Concepts and Techniques, Morgan-Kaufmann, 1993.

[12] A. Aiken, J. M. Hellerstein, J. Widom, "Behavior of Database Production Rules: Termination, Confluence and Observable Determinism," ACM-SIGMOD, 1992.

[13] D. Montesi, E. Bertino, M. Bagnato, "Rules termination analysis investigating the interaction between transactions and triggers," in: Proceedings of the IDEAS'02, IEEE Computer Society Press, Silver Spring, MD, 2002.

[14] Weinand A., Gamma E., Marty R. "Design and Implementation of ET++, a Seamless Object-Oriented Application Framework,:" Structured Programming, Vol. 10, 1989.

[15] H. Harb, H. Kelash, A. Shehata. "Termination Analysis in Active Databases By Using Evolution Graphs," ITI 3rd International Conference on Information & Communication Technology 2005.

استحداث أداة لتحليل خاصية الانتهاء بواسطة تهذيب الرسم المطور للقواعد

في هذا البحث قمنا بعمل خوارزم جديد لتحليل مشكلة اللانهاية للقواعد في نظم قواعد البيانات وذلك في حالة السكون. هذا الخوارزم يستخدم طريقة رسم معروفة باسم Evolution Graphs هذه الطريقة تحاكي تشغيل القواعد ولكن في حالة السكون وتستخدم خصائص القواعد التنشيط والتهبيط والطريقة المستخدمة في اختيار القاعدة التالية في التنفيذ ساعد في اكتشاف عدد من الحالات التي لم تكن تكتشف من قبل. وهذا الخوارزم تم تعديله بإضافة جزء في حالة عدم اكتشاف خاصية الانتهاء بتعديل بسيط ورسم Refined Evolution Graph وتحليل القواعد للإجراءات فقط. بهذا التعديل تم اكتشاف حالات أكثر لم تكن تكتشف من قبل. تم اقتراح تركيب بنائي يعتمد علي تركيب بنائي معروف هو active DBMS SAMOS مع إضافة طبقة مقترحة لكي يناسب الخوارزم حيث أن أجزاء هذا التركيب تقوم بتنفيذ خطوات الخوارزم وتم عمل Implementation لأجزاء من هذا الجزء المقترح .