Termination Analysis of Active Rules with Priorities

Alain Couchot

Laboratoire Cedric-Isid,
Conservatoire National des Arts et Métiers, France
couchot-a@wanadoo.fr

Abstract. This paper presents an algorithm for termination static analysis of active rules with priorities. Active rules termination is an undecidable problem. Several recent works have suggested proving termination by using the concept of triggering graph. We propose here a refinement of these works, exploiting the priorities defined between rules. We introduce the notions of path set and destabilizing set. We show how to determine the priority of a path set. The triggering graph can then be reduced thanks to considerations about priorities of the path sets. Much more termination situations can be detected, since priorities are exploited.

Keywords. Active databases. Termination. Rules. Static analysis.

1 Introduction

Databases are now endowed with a reagent behavior. They can react to variations of the environment, and modify the stored data according to these variations. This results from two technological tendencies: on the one hand, the development of the rules languages (production rules, active rules, deductive rules), and, on the other hand, the development of object oriented technologies. Rules languages and object oriented technology play additional roles : the object oriented technology allows to integrate into the same entity the structural aspects and the behavioral aspects, and the rules languages allow to describe the reaction of an entity according to its environment.

However, although the vocation of rules is to facilitate design and programming, writing a rules set is still a delicate work, reserved for specialists. Indeed, a rules set is not a structured entity : the global behavior of a rules set is difficult to foresee and to control [AWH92]. Two important points were underlined by research works concerned with rules : the *termination* problem (the rules execution can sometimes be infinite), and the *confluence* problem (the same rules do not necessarily provide the same results, according to the order of rules execution).

We are here interested in the active rules termination problem. The active rules are structured according to paradigm Event-Condition-Action [DBC88]. Event is an

immediate fact, which can arise inside or outside of the system. Condition is a predicate or a request on the database. Action is generally composed of a sequence of database updates, or of a procedure containing database updates. Coupling modes allow to specify the evaluation moment of the Condition part, or the execution moment of the Action part. The most frequent modes are immediate mode and deferred mode. A rule can be triggered by a single occurrence of an event: *instance oriented rule*, or by a set of occurrences of an event: *set oriented rule*.

In the section 2, we present the previous work on the subject; in the section 3, we expose a motivating example; in the section 4, we introduce the path sets of a rule and the path sets of a path; in the section 5, we propose a reduction of the triggering graph based on the consideration of the prioritites defined for the active rules; the section 6 ends.

2 Previous works

The active rules termination is an undecidable problem, except when rules languages with very limited possibilities are used [BDR98]. The previous works on the active rules analysis generally propose criteria supplying sufficient conditions allowing to guarantee termination.

Some methods develop translation tools from active rules to rewriting terms systems [KU94], to Condition-Action rules [BW00], or to deductive rules [CT97, FG98]. However, the application of these methods is often complex, cannot be automated, and is reserved for relational databases.

The majority of works on active rules termination exploit the concept of *triggering graph*.

[CW90] introduced, for the first time, the notion of *triggering graph* ; this notion is clarified by [AWH92] : 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 oriented edge from *r1* to *r2* if the action of *r1* can provoke a triggering event of *r2*. The presence of cycles in a such graph means a risk of non-termination of the rules set. The absence of cycles in the triggering graph guarantees the termination of the rules set. However, the possible deactivation of the rule condition is not taken into account by this analysis. A finer analysis is led by [BCP95, BCP98] thanks to the *activation graphs*: in an *activation graph*, rules *r1 and r2* are connected by a oriented edge from *r1* to *r2* if the action of *r1* can make **TRUE** the condition of *r2*, and a rule *r* is connected with itself if the condition of *r* is still **TRUE** after execution of the action of *r*. Non-termination involves then the presence of cycles in the triggering graph and in the activation graph. An algorithm for building activation graphs is proposed by [BW94] in a relational context.

This technique is generalized by [LL97] : the concepts of *activator* and of *deactivator* are proposed. The rule *r1* is a *deactivator* of the rule *r2* if the action of *r1* makes always **FALSE** the truth value of the condition of *r2*. The rule *r1* is an *activator* of the rule *r2* if the action of *r1* can change the truth value of the condition of *r2* from **FALSE** in **TRUE**. A rule *r* is labeled *finite* if every activator of *r* is labeled *finite*, and if every cycle containing *r* also contains a deactivator of *r*.

[VGD97] studies the influence of composite events on the termination property: the triggering graphs are modified, in order to take into account the composite conjunction events. The modified triggering graphs hold two types of edges : the *total edges* (there is a total edge from *r1* to *r2* if the action of *r1* can trigger *r2*), and the *partial edges* (there is a partial edge from *r1* to *r2* if the action of *r1* contains one of the primitive events of *r2*).

The Refined Triggering Graph method [KU96, TUDK97] brings a complement to the analysis of the triggering graphs. A edge (*r1, r2*) can be removed from the triggering graph in the following circumstances : the post-condition of *r1* is combined with the condition of *r2* by means of the unification of the variables appearing in the action of *r1* and the variables appearing in the event of *r2* ; if the obtained formula (qualified as *triggering formula*) can be never satisfied, the edge (*r1 , r2*) can be removed. But variables appearing in the triggering formula must not be updated by rules actions. [LL98] widens the principle of the triggering formulae (*generalized triggering formulae*) as follows : if the rule *r1* can indirectly trigger the rule *r2* along a path *P*, a triggering formula is built along the path *P*. [LL98] proposes to remove a path instead of removing a node. For this, a new triggering graph is built, equivalent to the initial graph. [LL99] brings a new extension : the generalized triggering formulae are studied for a path, which corresponds to a cycle executed (*k+1*) times : if the triggering formula can never be satisfied, the cycle is called a *k-cycle*. A method is proposed to unroll a *k-cycle* : an equivalent triggering graph is built. [Cou02] generalizes the notion of triggering formula, allowing to include attributes which can be updated by rules actions.

[Cou01a] improves [VGD97] and [LL98, LL99], introducing the notion of *composite path*. A *composite path* is composed of paths, linked by the operators **AND** and **OR**. The notion of *composite path* allows to take into account for termination analysis at the same time both the deactivation of a overall condition of a rules path and the conjunction events.

[WH95] exploits the principle of monotonous bounded operations to remove some edges of the triggering graph : for example, if a rule action always deletes an object of a class, and if no rule action can create an object of this class, the delete operation is bounded, and the corresponding rule can only be executed a finite number of times. This principle is resumed and detailed by [DH00], that exposes a method to calculate the maximal number of iterations of a cycle, by using some limits of the database.

[BCP96] takes place in a context of active rules modular design, and introduces a *behavioral stratification* of the active rules : one *stratum* is a set of rules carrying out some task, whose progress can be measured by a metrics. [Cou01b] proposes the notion of public event and private event, in order to guarantee the termination of a rules set in a modular design context.

Priorities are considered by [BCP95]. A self-deactivating rule *R* can be removed from the triggering graph, if the priorities of the rules included in the paths from *R* to the rule which can reactivate the condition of *R* (the paths do not include *R*) is strictly lower than the priorities of the rules included in the paths from *R* to the rule which can trigger *R* (the paths include *R*).

Our work is based on the following observation : priorities between rules can be used to refine termination analysis. This observation has just partially been taken into account by [BCP95], within the framework of the self-deactivating rules. We show in this paper that much more termination cases can be detected using the priorities between rules.

The aim of our work is to reduce the triggering graph, thanks to the priorities of the rules. We first introduce the notions of *path set of a rule* and *path set of a path*. The path set includes the paths which are executed before a rule (or before a path). We determine then the priority of a path set. We also propose the notion of *destabilizing set of a path*. The destabilizing set of a path includes the rules which oppose the deactivation of the path. Thanks to the priorities of a path set, we can sometimes reduce the destabilizing sets of a path. This can lead to reduce the path set of a rule. When the path set of a rule is empty, the rule can be removed from the triggering graph.

3 Motivating example

We propose in this section an example which illustrates the motivation of our work. We take place within the framework of a banking application. An object oriented database is used. Four active rules are defined. We suppose that the coupling modes of the rules are immediate.

Rule R_1: When the loan capacity of an account is updated, if the loan rate of the account is equal to "low", the allowed overdraft of the account is set to "low". The priority of this rule is strong.

Rule R_2: When the allowed overdraft of an account is updated, if the loan rate of the account is equal to "high", the loan capacity of the account is set to "high". The priority of this rule is strong.

Rule R_3: When the loan capacity of an account is updated, the loan rate of the account is set to "low". The priority of this rule is weak.

Rule R_4: When the allowed overdraft of an account is updated, the loan rate of the account is set to "high". The priority of this rule is weak.

Rule:	R_1
Priority:	strong (numeric value: 2)
Event:	*account_capacity_update_event*(A_1)
Condition:	A_1.*rate* = "low"
Action:	A_1.*overdraft* = "low"
Raised event:	*account_overdraft_update_event*(A_1)

Rule:	R_2
Priority:	strong (numeric value:2)
Event:	*account_overdraft_update_event*(A_2)
Condition:	A_2.*rate* = "high"
Action:	A_2.*capacity* = "high"
Raised event:	*account_capacity_update_event*(A_2)

Rule:	R_3
Priority:	weak (numeric value: 1)
Event:	*account_capacity_update_event*(A_3)
Condition:	-
Action:	A_3.*rate* = "low"
Raised event:	*account_rate_update_event*(A_3)

Rule:	R_4
Priority:	weak (numeric value: 1)
Event:	*account_overdraft_update_event*(A_4)
Condition:	-
Action:	A_4.*rate* = "high"
Raised event:	*account_rate_update_event*(A_4)

Let us try to guarantee termination of this rules set using the algorithms proposed in the literature. The triggering graph is represented by figure 1.

The Refined Triggering Graph method [KU96, TUDK97] is unable to remove any edge of the cycle ($R_1 \rightarrow R_2 \rightarrow R_1$). Indeed, the attribute *rate* can be updated by rule actions and cannot be included in a triggering formula.

If we use the improvement of the RTG method proposed by [Cou02] (which allows to include in a triggering formula attributes which can be updated by rule actions), we obtain the following triggering formula for the edge $R_1 \rightarrow R_2$:

$((A_1.rate = \text{"low"}) \vee ((A_1 = A_4) \wedge (A_4.rate = \text{"high"}))) \wedge (A_1 = A_2) \wedge (A_2.rate = \text{"high"})$

This formula can be satisfied, and termination cannot be guaranteed.

The algorithm [BCP95], which considers rules with priorities, just considers self-deactivating rules. It cannot guarantee termination in this case, since no rule is self-deactivating.

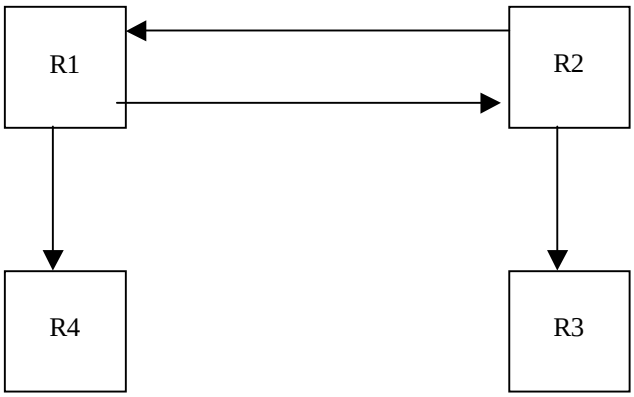

Figure 1. Triggering graph.

However, this rules set cannot exhibit an infinite processing. Let us suppose that an infinite process occurs. There is an infinite number of instances of R_1 and an infinite number of instances of R_2 (else, R_1 or R_2 would be removed from the triggering graph, and the process would be finite).

But no instance of R_3 can occur between two instances of R_1, and no instance of R_4 can occur between two instances of R_1. Indeed, let us take place at the moment where an instance of R_4 is evaluated and executed. At this moment, R_2 is not triggered (else R_2 would be evaluated and executed before R_4). If R_2 is not triggered, R_1 is not triggered. Since the action of R_4 triggers no rule, the rules process stops. We can do the same reasoning for R_3. As we have supposed that an infinite rules process occurred, we obtain a contradiction. Thus no instance of R_3 can occur between two instances of R_1, and no instance of R_4 can occur between two instances of R_1. That is the attribute *rate* cannot be updated between two instances of R_1.

So, we can build the following triggering formula for the edge (R_1, R_2) by using the RTG method, since we know now that the attribute *rate* cannot be updated between the evalutation of the condition of R_1 and the evaluation of the condition of R_2:

$(A_1.rate = \text{"low"}) \wedge (A_1 = A_2) \wedge (A_2.rate = \text{"high"})$

This formula cannot be satisfied. So, an infinite rules process is impossible for this rules set. An algorithm could draw this conclusion, if this algorithm would consider the priorities between rules.

4 Path sets

The utility of this section is to introduce the notions of *path set of a rule* and *path set of a path*. The path set serves for replacing the notion of cycle, used in the previous termination algorithms.

4.1 Considered active rules

The active rules that we consider in this paper are defined according to the paradigm Event-Condition-Action. The database model is a relational model or a model object. We suppose that the database is treated in a transactional context. Triggering events are either data operation events *(internal* events), or events reported to the system by the outside *(external* events), or a disjunction of internal and external events. The rule condition is a request expressed with some requests language. The rule action is a sequence of database updates, or, more generally, a procedure containing database updates.

We suppose that each rule is defined with a numeric priority. The rules process is the following:

1. Choose a rule instance with the strongest priority in the set of triggered rules instances.
2. Remove the chosen rule instance from the set of the triggered rule instances.
3. Evaluate the condition.
4. If the condition is false : go to step 1.
5. If the condition is true, execute the action.
6. Update the set of triggered rules instances.
7. Go to step 1.

4.2 Path set of a rule

We replace the classic notion of cycle (used by the previous termination algorithms) by the notion of *path set of a rule*. The path set of a rule captures the rules paths which are executed before a rule instance.

We first recall the notion of triggering graph. The nodes of the triggering graph are the active rules. There is an oriented edge from a rule R_1 to a rule R_2 if the rule R_1 can trigger the rule R_2.

Let G be a triggering graph. We first precise the notion of *path*. Let N_1 , N_2... N_i... N_n be *n node*s (not necessarily all different) of G, such as there is a oriented edge

since N_{i+1} towards N_i. The tuple (N_1 , N_2... N_n) constitutes a *path*. We adopt the following notation : $N_n \rightarrow N_{n-1} \rightarrow ... \rightarrow N_i \rightarrow ... \rightarrow N_1$. N_1 is called the *last node* of the path. N_n is called the *first node* of the path.

The *path set* of the rule R *Path_Set*(R ; G) is the set of the paths *Path* of G which satisfy the following properties:
1. The last rule of the path *Path* is R.
2. The path *Path* does not contain twice the same node.
3. The path *Path* is not included in a path which satisfies the properties 1 and 2 (except itself).

The path set of R is built performing a "depth search" in the opposite direction of the edges. The procedure is the following : (the first path provided to the procedure is the path (R))

Path_Set_Building_Procedure ((R))

Path_Set_Building_Procedure (**incoming variable :** $\boldsymbol{Path_{in}}$ **)**
Let N be the first node of $Path_{in}$
Let N_1, N_2, ... N_p be the nodes of G such as there is an edge from N_i to N
FOR each node N_i ($1 \leq i \leq p$)
IF N_i is not in $Path_{in}$
Path_Set_Building_Procedure ($N_i \rightarrow Path_{in}$)
ELSE
Add $Path_{in}$ to *Path_Set*(R ; G)
ENDIF
ENDFOR

Algorithm 1.

Example.
See the figure 1. *Path_Set*(R_4 ; G) = {($R_2 \rightarrow R_1 \rightarrow R_4$)}

4.3 Path set of a path

We generalize now the notion of path set, defining the *path set of a path*. The path set of a path corresponds to the set of the paths which are executed before an occurrence of the path. Let us consider a path :

$Path = N_n \rightarrow N_{n-1} \rightarrow ... \rightarrow N_i \rightarrow ... N_2 \rightarrow N_1$.

Let us suppose that:

Path_Set(N_n ; G) = {$Path_1$, $Path_2$,… , $Path_p$ }

The path set of the path *Path* is:

$Path\_Set(Path\ ;\ G) = \{(Path_1 \rightarrow N_{n-1} \rightarrow ... \rightarrow N_i \rightarrow ... \rightarrow N_1),$
$(Path_2 \rightarrow N_{n-1} \rightarrow ... \rightarrow N_i \rightarrow ... \rightarrow N_1), \ldots, (Path_p \rightarrow N_{n-1} \rightarrow ... \rightarrow N_i \rightarrow ... \rightarrow N_1)\}$

Example. See figure 1.
$Path\_Set((R_1 \rightarrow R_4)\ ;\ G) = \{R_2 \rightarrow R_1 \rightarrow R_4\}$

4.4 Priority of a path / path set

We introduce here the notion of *priority of a path*. The priority of a path is the weakest priority of the rules of the path. The priority of the path corresponds to a narrow gorge: this is the priority which will delay the execution of the path.

We use the notion of priority of a path to define now the *minimal priority* of a path set: this is the weakest priority of the paths of the path set. We adopt the following notation: $m\_p(Path\_Set(Entity\ ;\ G))$. (*Entity* is a path or a rule).

We also define the *maximal priority* of a path set: this is the stronger priority of the paths of the paths set. We adopt the following notation: $M\_p(Path\_Set(Entity\ ;\ G))$.

5 Reduction of the triggering graph

This section introduces our termination algorithm. The previous termination algorithms exploit the following principle: when every cycle containing the rule *R* contains a deactivated path, the rule *R* can be removed from the triggering graph. Our termination algorithm suggests improving this principle: the rule *R* will be removed by our algorithm when the path set of *R* will be empty. Considerations about priorities of the rules will allow to reduce the path set of a rule.

5. 1 Destabilizing set of a path

In order to refine the deactivation of a path, we introduce the notion of *destabilizing set*. The utility of this notion is to list the rules which can oppose the deactivation of the path. A *destabilizing set of a path* is a set of rules. Informally, the conjunction of the conditions of the rules of the path is **TRUE** only a finite number of times, if there is only a finite number of occurrences of the rules of the destabilizing set.

Definition. Let *Path* be a path of the triggering graph *G*. Let R_1 , R_2, …, R_s be *s* rules of *G*.

We say that the set $\{R_1, R_2, \ldots, R_s\}$ is a *destabilizing set* of *Path* iff the following property holds for each rules process *P*:
(A finite number of instances of the rules $R_1, R_2, \ldots, \ldots, R_s$ occur during *P*) $\Rightarrow$ (There is only a finite number of occurrences of *Path* during *P*).

For a destabilizing set, we will use the following notation: *Desta_Set*(*Path* ; *G*)
The destabilizing sets can be used in conjunction with all the deactivation cases of path listed by the previous algorithms [BCP95, BCP98, BW94, DH00, KU96, LL97, TUDK97, WH95].

For example, if it is possible to establish a generalized triggering formula along a path which cannot be satisfied [KU96, LL98, TUDK97], but if this formula contains attributes which can be updated by the database, a destabilizing set of the path contains all the rules R_i such as the action of the rule R_i can modify attributes contained in the generalized triggering formula.

Note that a path can have zero or several destabilizing sets.

Example. See the motivation example and figure 1. $\{R_3, R_4\}$ is a destabilizing set of $(R_2 \rightarrow R_1)$. Indeed, we can establish a triggering formula along the edge $(R_2 \rightarrow R_1)$ using the "Refined Triggering Graph" Method [KU96, TUDK97] :

$(A_1.rate = \text{"low"}) \wedge (A_1 = A_2) \wedge (A_2.rate = \text{"high"})$

This triggering formula can not be satisfied. But the attribute *rate* can be updated by the rules R_3 and R_4. If there is just a finite number of instances of R_3 and R_4 during a rules process, there is just a finite number of occurrences of the path $(R_2 \rightarrow R_1)$ during the rules process.

5.2 Reduction of the triggering graph

We can reduce the triggering graph thanks to the four following considerations:

(1) The rule *R* can be removed from the triggering graph if *R* has no incoming edge. Indeed, *R* will be just triggered a finite number of times.

(2) Let *R* be a rule. Let $Path_j$ be a path of *Path_Set*(*R* ; *G*). We can remove $Path_j$ from *Path_Set*(*R* ; *G*) if $\varnothing$ is a destabilizing set of $Path_j$. Indeed, in this case, it is impossible for the rules process to go through the path $Path_j$ an infinite number of times.

(3) The rule R can be removed from the triggering graph if $Path\_Set(R\ ;\ G) = \emptyset$. Indeed, in this case, it is impossible for the rules process to reach the rule R an infinite number of times, since all the paths which lead to R are deactivated after a finite time.

(4) Let us consider now a destabilizing set $\{R_1, R_2, \ldots, R_s\}$ of the path *Path*.

If we observe one of the following properties :
(i) the maximal priority of $Path\_Set(Path\ ;\ G)$ is strictly smaller than the minimal priority of $Path\_Set(R_i\ ;\ G)$ or:
(ii) the maximal priority of $Path\_Set(R_i;\ G)$ is strictly smaller than the minimal priority of $Path\_Set(Path\ ;\ G)$,
then the rule R_i can be removed from the destabilizing set $\{R_1, R_2, \ldots, R_s\}$.

Proof of (4). Let us consider the case (i). We show that there is no occurrence of *Path* between two instances of R_i. Let us suppose that there is an occurrence of *Path* between two instances of R_i. This means that there is a rule R' which belongs to the path set of *Path* and which is evaluated and executed before a rule R'', which is triggered and which belongs to the path set of R_i. But the priority of R' is strictly smaller than the priority of R'' (since each rule of the path set of R_i has a priority stronger than each rule of the path set of *Path*). This leads to a contradiction. So, since there is no occurrence of *Path* between two instances of R_i, there is no instance of R_i between two occurrences of *Path*. Thus, R_i can be removed from the destabilizing set. We can do the same type of reasoning for the case (ii).

5.3 Termination algorithm

We can now sketch the termination algorithm. The termination algorithm captures the four properties which we have shown above :
(1) We can remove a rule from the current graph if the rule has no incoming edge.
(2) We can remove a path *Path* from a path set if $\emptyset$ is a destabilizing set of *Path*.
(3) We can remove a rule from the current graph if the path set of the rule is empty.
(4) We can remove a rule from a destabilizing set depending on the priorities.

REPEAT
  REPEAT
    Remove the nodes without incoming edge
    Remove the edges without origin node
  UNTIL (no node is removed)

  FOR EACH rule *R* of *G*
    Determine *Path_Set*(*R* ; *G*)
    FOR EACH path *Path* of *Path_Set*(*R* ; *G*)
      Determine *Path_Set*(*Path* ; *G*)
      Determine the sets *Desta_Set*(*Path* ; *G*)
      FOR EACH set *Desta_Set*(*Path* ; *G*)
        FOR EACH rule R_i of *Desta_Set*(*Path* ; *G*)

IF ($M\_p(Path\_Set(Path\ ;\ G)) < m\_p(Path\_Set(R_i\ ;\ G))$)
OR ($M\_p(Path\_Set(R_i\ ;\ G)) < m\_p(Path\_Set(Path\ ;\ G))$)
Remove R_i from $Desta\_Set(Path\ ;\ G)$
ENDIF
ENFOR
ENDFOR
IF there is a set $Desta\_Set(Path\ ;\ G)$ such as $Desta\_Set(Path\ ;\ G) = \emptyset$
Remove *Path* from $Path\_Set(R\ ;\ G)$
ENDIF
ENDFOR
IF $Path\_Set(R\ ,\ G) = \emptyset$
Remove *R* from *G*
ENDIF
ENDFOR
UNTIL (no entity is removed)

Algorithm 2.

If the final triggering graph is empty, termination is guaranteed. Else, the remaining rules can possibly be triggered an infinite number of times. Note that our termination algorithm can discover the termination situations detected due to the self-deactivating rules [BCP95, BCP98, BW94], due to the monotonous bounded operations [DH00, WH95], due the RTG method [KU96, TUDK97], due to the deactivation of a condition because of the action of a rule [LL97], and the termination situations detected by the "Path Removing Technique " [LL98].

Much more termination situations can in fact be discovered by our algorithm, since our termination algorithm exploits the priorities of the active rules.

Example.
Let us study the termination of the set of rules of our motivating example.
$\{R_3, R_4\}$) is a destabilizing set of the path ($R_2 \rightarrow R_1$) (see section 5.1).

We have:
$Path\_Set((R_2 \rightarrow R_1); G) = \{(R_2 \rightarrow R_1)\}$
$Path\_Set(R_3\ ;\ G) = \{(\ R_1 \rightarrow R_2 \rightarrow R_3)\}$
$Path\_Set(R_4\ ;\ G) = \{(\ R_2 \rightarrow R_1 \rightarrow R_4)\}$

We determine the priorities of the path sets:
$m\_p(Path\_Set((R_2 \rightarrow R_1)\ ;\ G)) = 2$
$M\_p(Path\_Set(R_3\ ;\ G)) = 1$
$M\_p(Path\_Set(R_4\ ;\ G)) = 1$

We observe that:
$M\_p(Path\_Set((R_3)\ ;\ G)) < m\_p(Path\_Set((R_2 \rightarrow R_1)\ ;\ G))$
and that:
$M\_p(Path\_Set((R_4)\ ;\ G)) < m\_p(Path\_Set((R_2 \rightarrow R_1)\ ;\ G))$

So, we can remove R_3 and R_4 from the destabilizing set of the path $(R_2 \rightarrow R_1)$.

Thus, $\varnothing$ is a destabilizing set of $(R_2 \rightarrow R_1)$. The path set of R_1 is $\{(R_2 \rightarrow R_1)\}$. Since $\varnothing$ is a destabilizing set of $(R_2 \rightarrow R_1)$, we can remove the path $(R_2 \rightarrow R_1)$ from the path set of R_1. The path set of R_1 is then $\varnothing$. This leads to remove the rule R_1 from the triggering graph. Once the rule R_1 is removed, we can remove the other rules of the triggering graph.

The termination of this rules set can be guaranteed by our algorithm. Note that no previous algorithm is able to detect this termination situation.

5.4 Implementation

By means of a syntactic analysis of the set of rules, an initial triggering graph *G* is built (cf. figure 2). The graph *G* is analyzed by the main module. The main module uses the Deactivation Module to detect the destabilizing sets of a path. The Deactivation Module uses several Evaluation Modules. The Evaluation Modules allow to determine destabilizing sets of a path using the techniques developed by the previous termination analysis techniques. At the moment, we foresaw two Evaluation Modules: the RTG module (which corresponds to the method "Refined Triggering Graph" [KU96, TUDK97]), the Module "Monotonous Bounded Operations" (which corresponds to the "Monotonous Bounded Operations" method [DH00, WH95]). We plan to add other Evaluation Modules in the future.

Note that the main module is independent from the rules language. Only the graph building module and the evaluation modules depend on the used rules language.

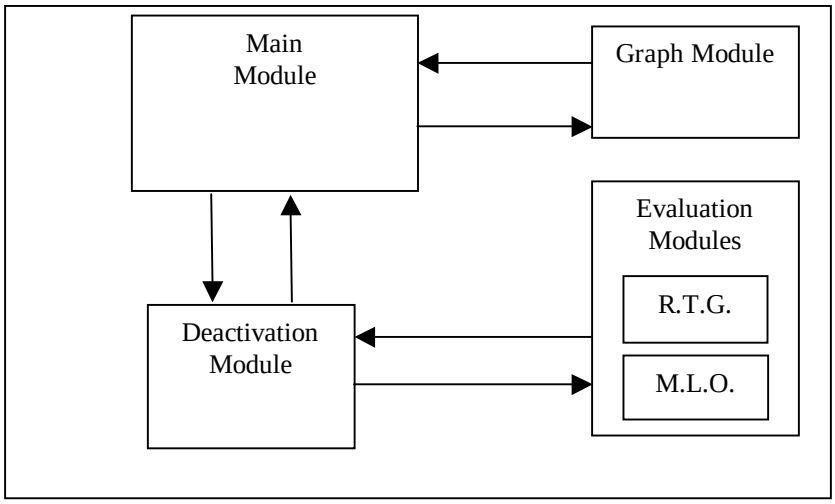

Fig. 2. Implementation of the termination algorithm.

For the optimization of the algorithm, we observe the following property: if a path *Path'* is included in a path *Path*, and if *Path'* is deactivated, then *Path* is deactivated. Thanks to this observation, some calculations can be avoided.

An other observation can avoid calculations: if *G'* is a subgraph of *G*, we can obtain *Path_Set*(*Entity* ; *G'*) from *Path_Set*(*Entity* ; *G*), removing from *Path_Set*(*Entity* ; *G*) the paths containing a rule which is no more in *G'*. This observation is also available for *Desta_Set*(*Entity* ; *G'*) and *Desta_Set*(*Entity* ; *G*).

6 Conclusion

We have presented a significant improvement of the termination analysis of the active rules defined with priorities. We have developed the notions of path set of rule and path set of a path. The notion of destabilizing set has allowed us to represent the deactivation cases of a condition listed by the previous algorithms [BCP95, BCP98, DH00, KU96, LL97, TUDK97, WH95]. We can then reduce the destabilizing set of a path thanks to the priorities of the path sets. When the destabilizing set of a path is empty, the path can be removed from the path set of a rule. When the path set of a rule is empty, the rule can be removed from the triggering graph. So, the triggering graph can be reduced thanks to considerations about the priorities of the rules.

In the future, we plan to conceive an algorithm which proposes priorities between rules, when the termination can not be guaranteed.

References

AWH92 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.

BDR98 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.

BCP95 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.

BCP96 E. Baralis, S. Ceri, S. Paraboschi. Modularization Techniques for Active Rules Design. In *ACM Transactions on Database Systems, (TODS)*, 21(1), 1996.

BCP98 E. Baralis, S. Ceri, S. Paraboschi. Compile-Time and Run-Time Analysis of Active Behaviors. In *IEEE Transactions on Knowledge and Data Engineering*, 10 (3), 1998.

BW94 E. Baralis, J. Widom. An Algebraic Approach to Rule Analysis in Expert Database Systems. In *Proc. Int'l Conf. on Very Large Databases (VLDB)*, Santiago, Chile, 1994.

BW00 E. Baralis, J. Widom. Better Static Rule Analysis for Active Database Systems. In *ACM Transactions on Database Systems (TODS)*, 2000.

CW90 S. Ceri, J. Widom. Deriving Production Rules for Constraint Maintenance. In *Proc. Int'l Conf. on Very Large Databases (VLDB)*, Brisbane, Queensland, Australia, 1990.

CT97 S. Comai, L. Tanca. Using the Properties of Datalog to prove Termination and Confluence in Active Databases. In *Proc. Int'l Workshop on Rules in Database Systems (RIDS)*, Skoevde, Sweden, 1997.

Cou01a A. Couchot. Improving Termination Analysis of Active Rules with Composite Events. In *Proc. Int' l Conf. on Database and Expert Systems Applications (DEXA)*, Munich, Germany, 2001.

Cou01b A. Couchot. Termination Analysis of Active Rules Modular Sets. In *Proc. Int' l Conf. on Information and Knowledge Management (CIKM)*, Atlanta, Georgia, USA, 2001.

Cou02 A. Couchot. Improving the Refined Triggering Graph Method for Active Rules Termination Analysis. In *Proc. British National Conf. on Databases* (*BNCOD*), Sheffield, United Kingdom, 2002.

DBC88 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, Germany, 1988.

DH00 S. Debray, T. Hickey. Constraint-Based Termination Analysis for Cyclic Active Database Rules. In *Proc. Int' l Conf. on Deductive Object Oriented Databases (DOOD).* London, United Kingdom, 2000.

FG98 S. Flesca, S. Greco. Declarative Semantics for Active Rules. In *Proc. Int'l Conf. on Database and Expert Systems Applications (DEXA)*, Vienna, Austria, 1998.

KU94 A.P. Karadimce, S.D. Urban. Conditional Term Rewriting as a Formal Basis for Analysis of Active Database Rules. In *Proc. Int'l Workshop on Research Issues in Data Engineering (RIDE-ADS)*, Houston, Texas, 1994.

KU96 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.

LL97 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

LL98 S.Y. Lee, T.W. Ling. A Path Removing Technique for Detecting Trigger Termination. In *Proc. Int'l Conf. on Extended Database Technology (EDBT)*, Valencia, Spain, 1998.

LL99 S.Y. Lee, T.W. Ling. Unrolling Cycle to Decide Trigger Termination. In *Proc. Int'l Conf. on Very Large Databases (VLDB)*, Edinburgh, Scotland, 1999.

TUDK97 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.

VGD97 A. Vaduva, S. Gatziu, 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.

WH95 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.