AN APPROACH TO MONITORING CHANGES IN DEDUCTIVE DATABASES

(preliminary version)

Toni Urpí

Universitat Politècnica de Catalunya
Facultat d'Informàtica
Pau Gargallo 5
08028 Barcelona
Catalunya
Tel. 34-3-4017004
Fax. 34-3-4017040
e-mail: urpi@lsi.upc.es

ABSTRACT

We propose a method to monitor changes in deductive databases. The method augments a database with a set of transition and events rules, which explicitly define the insertions, deletions and modifications induced by a database update. The main advantage of the method is that it allows a high degree of expressiveness in the representation of derived predicates. The method also uses the classical concept of key in order to obtain a set of rules which are semantically richer.

KEYWORDS

Monitoring, Deductive databases

June 1991

1. INTRODUCTION

In deductive databases, predicates are either base or derived. Facts of base predicates are explicitly stored. Facts of derived predicates can be deduced from base and/or derived predicates using deductive rules. When base facts are modified by update operations, derived facts may also change. The problem addressed in this paper is the efficient monitoring of these changes.

There are many database areas that require to monitor database changes: condition monitoring in active databases [RCB 89]; integrity constraints enforcement [Oli 91, SaK 88]; maintenance of materialized views [BLT 86] and production rules systems in general [Han 89, MD 89, Sto 90, WiF 90]. The efficient monitoring of these changes can be useful in all of these areas.

There is a simple but inefficient way to do this: if we deduce the derived facts before and after the update operations, we can easily obtain the differences and then the changes induced by the transition. In order to overcome this obvious inefficency the existent methods propose more efficient ways.

We propose to use an extension of the events method to improve monitor performance. Originally, the events method was introduced as an approach for the design of information systems from deductive conceptual models [Oli 89]. As a particular application, the method was also applied for integrity checking in deductive databases [Oli 91]. The method augments a database with a set of rules, called transition and events rules, which explicitly define the insertions and deletions induced by a database update.

We propose an extension to the events method that incorporates a new event: the modification. The introduction of this new event leads us to introduce the concept of key, used to relate facts that hold values of the same object before and after the modifications. We also use keys as knowledge that we incorporate into our transition and events rules, allowing us to obtain a set of rules which are semantically richer.

The paper is organized as follows. The second section defines basic concepts of deductive databases, and presents an example that will be used throughout the paper. Section 3 defines the concept of event, and presents a method for deriving transition and events rules. In section 4 we compare our method with [RCB 89]. Finally, in section 5 we present our conclusions.

2. DEDUCTIVE DATABASES

A deductive database D consists of three finite sets: a set F of facts, a set R of deductive rules, and a set I of integrity constraints. A fact is a ground atom. The set of facts is called the Extensional Database, and the set of deductive rules is called the Intensional Database.

We assume that database predicates are either base or derived. A base predicate appears only in the extensional database and (eventually) in the body of the deductive rules. A derived predicate appears only in the intensional database. Every database can be defined in this form [BaR 86].

We also assume that database predicates (base or derived) can have two types of arguments: those (k) that form a key for the predicate, and those (x) that do not. We use the classical concept of key, and so each database predicate must have key arguments. We have, then, two types of predicates: either P(k,x) or P(k). To enforce the concept of key we assume that associated to each P(k,x) there is a key integrity constraint:

$$\leftarrow P(k,x) \wedge P(k,x') \wedge x \neq x'$$

We show in figure 1 the database example that will be used throughout the paper. In this example, there are five base predicates, five derived, five deductive rules and nine key integrity constraints. We underline the key components in each predicate.

2.1 Deductive rules

A deductive rule is a formula of the form:

$$A \leftarrow L_1 \wedge \ldots \wedge L_n \quad \text{with } n \geq 1$$

where A is an atom denoting the conclusion, and the $L_1, \ldots, L_n$ are literals representing conditions. Each L_i is either an atom or a negated atom. Any variables in A, $L_1, \ldots, L_n$ are assumed to be universally quantified over the whole formula. The terms in the conclusion must be distinct variables, and the terms in the conditions must be variables or constants.

Condition predicates may be evaluable ("built-in") or ordinary (non-evaluable). The latter are base or derived predicates, while the former are predicates, such as the comparison or arithmetic predicates, that can be evaluated without accessing the database.

As usual, the database must be *allowed* before and after any update, that is, any variable that

occurs in a deductive rules has an occurence in a positive condition of the rule. This ensures that all negative conditions can be fully instantiated before they are evaluated by the "negation as failure" rule.

2.2 Integrity constraints

An integrity constraint is a closed first-order formula that the database is required to satisfy. We deal with constraints that have the form of a denial:

$$\leftarrow L_1 \wedge \ldots \wedge L_n \quad \text{with } n \geq 1$$

where the L_i are literals, and variables are asumed to be universally quantified over the whole formula.

In this paper we are specially interested in key integrity constraints that we define as:

$$\leftarrow P(k,x) \wedge P(k,x') \wedge x \neq x'$$

Normally, keys of derived predicates can be deduced from the deductive rules of these predicates. However, in figure 1, we declare their keys for clarity.

Note that x and x' are vectors of variables, and so the inequality $x \neq x'$ could be more precisely defined with the "different" predicate:

$$\text{different}(x_1,\ldots,x_n, x'_1,\ldots,x'_n) \leftarrow x_1 \neq x'_1$$

...

$$\text{different}(x_1,\ldots,x_n, x'_1,\ldots,x'_n) \leftarrow x_n \neq x'_n$$

For the same reason if we have an expression like x = x', we define it more precisely as:

$$(x_1 = x'_1 \wedge \ldots \wedge x_n = x'_n)$$

Base predicates

Emp(e,n,s) employee e has name n and salary s
User1(u,n) user u has name n and he is a user of system 1
User2(u,n) user u has name n and he is a user of system 2
Ed(e,d) employee e works in the department d
Dept(d,n) department d has name n

Deductive rules

DR.1 Emp32(e,n,s) ← Emp(e,n,s) ∧ s>32
DR.2 Pemp(e,s) ← Emp(e,n,s)
DR.3 Edn(e,d,n) ← Ed(e,d) ∧ Dept(d,n)
DR.4 User1n2(u,n) ← User1(u,n) ∧ ¬User2(u,n)
DR.5 User1emp(u,n) ← User1(u,n) ∧ Emp(e,n,s)

Integrity constraints

← Emp(e,n,s) ∧ Emp(e,n',s') ∧ different(n,s,n's')
← User1(u,n) ∧ User1(u,n') ∧ n ≠ n'
← User2(u,n) ∧ User2(u,n') ∧ n ≠ n'
← Dept(d,n) ∧ Dept(d,n') ∧ n ≠ n'
← Emp32(e,n,s) ∧ Emp32(e,n',s') ∧ different(n,s,n',s')
← Pemp(e,s) ∧ Pemp(e,s') ∧ s ≠ s'
← Edn(e,d,n) ∧ Edn(e,d,n') ∧ n ≠ n'
← User1n2(u,n) ∧ User1n2(u,n') ∧ n ≠ n'
← User1emp(u,n) ∧ User1emp(u,n') ∧ n ≠ n'

Figure 1. Example of deductive database

3. TRANSITION AND EVENTS RULES

In this section we define the events, a key concept in our method. We also explain how to derive the transition and events rules for a given database. These rules depend only on the deductive rules and integrity constraints. They are independent from the particular base facts stored in the database, and from any particular update.

3.1 Events

Let D be a database, U an update and D' the updated database. We say that U induces a transtion from D (the current state) to D' (the new, updated state). We assume for the moment that U consists of an unspecified set of base facts to be inserted, deleted or modified.

Due to the deductive rules, U may induce other updates on some derived predicates. Our objective is to monitor these derived predicates updates efficiently. Let P be a derived predicate and P' the same predicate evaluated on D'. Assuming that P(K,X) holds on D, where K and X are vectors of constants, three cases are possible in D':

1. $P'(K,X)$ also holds	we say that	nothing has occurred
2. $\neg\exists Y$ such that $P'(K,Y)$ also holds	"	a $\delta P(K,X)$ has occurred
3. $\exists X'$ such that $P'(K,X')$ also holds and $X \neq X'$	"	a $\mu P(K,X,X')$ has occurred

and assuming that $P'(K,X)$ holds in D', three cases are possible in D:

1. P(K,X) also holds	we say that	nothing has occurred
2. $\neg\exists Y$ such that P(K,Y) also holds	"	a $\iota P(K,X)$ has occurred
3. $\exists X'$ such that $P(K,X')$ also holds and $X \neq X'$	"	a $\mu P(K,X',X)$ has occurred

Formally, we associate three meta predicates to each base or derived predicate P: an insertion events predicate ιP, a deletion events predicate δP and a modification events predicate μP, defined as:

(1) $\forall k,x\ (\ \iota P(k,x) \leftrightarrow P'(k,x) \wedge \neg\exists y P(k,y))$

(2) $\forall k,x\ (\ \delta P(k,x) \leftrightarrow P(k,x) \wedge \neg\exists y P'(k,y))$

(3) $\forall k,x,x'\ (\ \mu P(k,x,x') \leftrightarrow P(k,x) \wedge P'(k,x') \wedge x \neq x')$

Examples:

Emp32(e, n, s)	Emp32'(e, n, s)		
1 Joan 60	1 Joan 40	=>	μEmp32(1,Joan,60,Joan,40)
2 Enric 50		=>	δEmp32(2,Enric,50)
	4 Anna 70	=>	ιEmp32(4,Anna,70)

From (1), (2) and (3) we have:

(4) $\forall k,x\ (P'(k,x) \leftrightarrow (P(k,x) \wedge \neg\delta P(k,x) \wedge \neg\mu P(k,x,x')\ \vee$
$\iota P(k,x)\ \vee$
$\mu P(k,x',x)\ )$

(5) $\forall k,x\ (\neg P'(k,x) \leftrightarrow (\neg P(k,x) \wedge \neg\iota P(k,x) \wedge \neg\mu P(k,x',x)\ \vee$
$\delta P(k,x)\ \vee$
$\mu P(k,x,x')\ )$

If P is a base predicate, ιP facts, δP facts and μP facts represent insertions, deletions and modifications of base facts, respectively. Therefore, we assume from now on that U consists of an unspecified set of insertion, deletion and/or modification events of base predicates.

From (1), (2) and (3) we require that

(6) $\forall k,x\ (\ \iota P(k,x) \rightarrow \neg\exists y P(k,y))$ and

(7) $\forall k,x\ (\ \delta P(k,x) \rightarrow P(k,x))$ and

(8) $\forall k,x,x'\ (\ \mu P(k,x,x') \rightarrow P(k,x) \wedge x \neq x')$

also hold for base predicates.

Furthermore, from (1), (2) and (3) we deduce that ιP(k,x), δP(k,x) and μP(k,x,x') are mutually exclusive (we will require this also for base predicates):

(9) $\forall k,x\ (\ \iota P(k,x) \rightarrow \neg\exists y\ \delta P(k,y))$
$\forall k,x\ (\ \iota P(k,x) \rightarrow \neg\exists y,y'\ \mu P(k,y,y'))$

(10) $\forall k,x\ (\ \delta P(k,x) \rightarrow \neg\exists y\ \iota P(k,y))$
$\forall k,x\ (\ \delta P(k,x) \rightarrow \neg\exists y,y'\ \mu P(k,y,y')$

(11) $\forall k,x,x'\ (\ \mu P(k,x,x') \rightarrow \neg\exists y\ \delta P(k,y))$
$\forall k,x,x'\ (\ \mu P(k,x,x') \rightarrow \neg\ \exists y\ \iota P(k,y))$

In [Oli 91] we can find the analogue rules for the case of predicates with only key arguments:

(1') $\forall k\ (\ \iota P(k) \leftrightarrow P'(k) \wedge \neg P(k))$

(2') $\forall k\ (\ \delta P(k) \leftrightarrow P(k) \wedge \neg P'(k))$

(3') Not defined. We can only modify predicates with non-key arguments.

(4') $\forall k\ (P'(k) \leftrightarrow (P(k) \wedge \neg \delta P(k)) \ \vee \iota P(k))$

(5') $\forall k\ (\neg P'(k) \leftrightarrow (\neg P(k) \wedge \neg\ \iota P(k)) \ \vee \delta P(k))$

(6') $\forall k\ (\ \iota P(k) \rightarrow \neg P(k))$

(7') $\forall k\ (\ \delta P(k) \rightarrow P(k))$

(8') Not defined. We can only modify predicates with non-key arguments.

(9') $\forall k\ (\ \iota P(k) \rightarrow \neg \delta P(k))$

(10') $\forall k\ (\ \delta P(k) \rightarrow \neg \iota P(k))$

(11') Not defined.

Additionally, we note two more things. First, we assume that U and the induced facts by U do not violate any integrity constraint. In particular, key integrity constraints are not violated. So, we assume that integrity constraints are checked in a previous step. Second, as we have observed, some of the rules - (1), (2),.. - do not satisfy the allowedness property. In our particular case, this is not worrying because we can rewrite these rules in order to satisfy that property. For instance:

rule (1) $\forall k,x\ (\ \iota P(k,x) \leftrightarrow P'(k,x) \wedge \neg \exists y P(k,y))$ can be rewritten as:

(1") $\forall k,x\ (\ \iota P(k,x) \leftrightarrow P'(k,x) \wedge \neg someP(k))$
(2") $someP(k) \leftrightarrow P(k,y)$

Up to now, we have defined $\iota P(k,x)$, $\delta P(k,x)$, $\mu P(k,x,x')$, $\iota P(k)$ and $\delta P(k)$ as a particular first-order formula in which P and P' intervene. The evaluation of these events would need a complete extension of P and P' to be obtained. First, we would have to apply U over D (current state) to obtain D' (new state), and then, we would get P'. This is particulary inefficient. We will show a way in which this evaluation can be improved.

3.2 Transition rules

Let P be a derived predicate of the database. The definition of P consists of the rules in the database having P in the conclusion. Assume that there are m ($m \geq 1$) such rules. For our purposes, we require to rename the conclusions of the m rules by $P_1 \ldots P_m$, change the implication by an equivalence and add the set of clauses:

$$P \leftarrow P_i \qquad i = 1...m$$

Thus, refering to rule DR.3:

$$Edn_1(\underline{e,d},n) \leftrightarrow Ed(\underline{e,d}) \wedge Dept(\underline{d},n)$$
$$Edn(\underline{e,d},n) \leftarrow Edn_1(\underline{e,d},n)$$

Given a rule $P_i(k,x) \leftrightarrow L_1 \wedge ... \wedge L_n$ (where x can be empty), we will denote by $U(P_i)$ the conjunction of the literals in the body having a vector of variables k_j (k_j key) such that $k \supseteq k_j$, and we will denote by $E(P_i)$ the conjunction of the literals in the body such that their keys have some variable which is not in k. Thus, refering to rules DR.3 and DR.5, we have:

DR.3 $U(Edn_1) = \{ Ed(\underline{e},\underline{d}) \wedge Dept(\underline{d},n) \}$ $\quad E(Edn_1) = \varnothing$

DR.5 $U(User1nemp_1) = \{ User1(\underline{u},n) \}$ $\quad E(User1nemp_1) = \{ Emp(\underline{e},n,s) \}$

Consider now one of the rules $P_i(k,x) \leftrightarrow L_1 \wedge ... \wedge L_n$ (where x can be empty). When the rule is to be evaluated in the updated state its form is $P'_i(k,x) \leftrightarrow L'_1 \wedge ... \wedge L'_n$. Now if we replace each literal in the body by its equivalent definition in terms of the current state and the events, we get a new rule, called a *transition rule*, which defines predicate P'_i (new state) in terms of current state predicates and events.

More precisely:

if L'_j is positive and has non-key variables [$Q_j(k_j,x_j)$] we apply (4) and replace it by:

$$(Q_j(k_j,x_j) \wedge \neg \delta Q_j(k_j,x_j) \wedge \neg \mu Q_j(k_j,x_j,x'_j)) \ \vee \ \iota Q(k_j,x_j) \ \vee \ \mu Q(k_j,x'_j,x_j)$$

if L'_j is positive and does not have any non-key variables [$Q_j(k_j)$] we apply (4') and replace it by:

$$(Q_j(k_j) \wedge \neg \delta Q_j(k_j)) \vee \iota Q(k_j)$$

if L'_j is negative and has non-key variables [$\neg Q_j(k_j,x_j)$] we apply (5) and replace it by:

$$(\neg Q_j(k_j,x_j) \wedge \neg \iota Q_j(k_j,x_j) \wedge \neg \mu Q_j(k_j,x'_j,x_j)) \vee \delta Q(k_j,x_j) \ \vee \ \mu Q(k_j,x_j,x'_j)$$

if L'_j is negative and does not have any non-key variables [$\neg Q_j(k_j)$] we apply (5') and replace it by:

$$(\neg Q_j(k_j) \wedge \neg \iota Q_j(k_j)) \vee \delta Q(k_j)$$

if L_j is an evaluable predicate, we just replace L'_j (positive or negative) by its current state version L_j.

Note that when we introduce variables x'_j, we are refering to new variables, not used previously.

It will be easier to refer to the resulting expresion if we denote it by:

$$
\begin{aligned}
M(L'_j) &= \mu Q(k_j,x'_j,x_j) && \text{if } L'_j = Q'_j(k_j,x_j)\\
&= \mu Q(k_j,x_j,x'_j) && \text{if } L'_j = \neg Q'_j(k_j,x_j)
\end{aligned}
$$

$$
\begin{aligned}
N(L'_j) &= \iota Q(k_j,x_j) && \text{if } L'_j = Q'_j(k_j,x_j)\\
&= \iota Q(k_j) && \text{if } L'_j = Q'_j(k_j)\\
&= \delta Q(k_j,x_j) && \text{if } L'_j = \neg Q'_j(k_j,x_j)\\
&= \delta Q(k_j) && \text{if } L'_j = \neg Q'_j(k_j)
\end{aligned}
$$

$$
\begin{aligned}
O(L'_j) &= (Q_j(k_j,x_j) \wedge \neg\delta Q_j(k_j,x_j) \wedge \neg\mu Q_j(k_j,x_j,x'_j)) && \text{if } L'_j = Q'_j(k_j,x_j)\\
&= (Q_j(k_j) \wedge \neg\delta Q_j(k_j)) && \text{if } L'_j = Q'_j(k_j)\\
&= (\neg Q_j(k_j,x_j) \wedge \neg\, \iota Q_j(k_j,x_j) \wedge \neg\mu Q_j(k_j,x'_j,x_j)) && \text{if } L'_j = \neg Q'_j(k_j,x_j)\\
&= (\neg Q_j(k_j) \wedge \neg\, \iota Q_j(k_j)) && \text{if } L'_j = \neg Q'_j(k_j)\\
&= L_j && \text{if } L'_j \text{ is evaluable}
\end{aligned}
$$

With this notation we then have:

$$
(12) \qquad P'_i(k,x) \leftrightarrow \bigwedge_{r=1}^{r=n} [\, O(L_r) \vee N(L_r) \vee M(L_r) \mid O(L_r) \vee N(L_r) \mid O(L_r) \,]
$$

where the first option is taken if L'_r is an ordinary literal with non-key variables, the second one if L'_r is an ordinary literal without non-key variables, and the third one if L'_r is evaluable. After distributing $\wedge$ over $\vee$, we get an equivalent set of transition rules, each of them with the general form:

$$
(13) \qquad P'_{i,j}(k,x) \leftrightarrow \bigwedge_{r=1}^{r=n} [\, O(L'_r) \mid N(L'_r) \mid M(L'_r) \,] \text{ for } j = 1 \ldots \alpha
$$

where $\alpha = 2^{lk} * 3^{lnk}$; **lk** is the number of ordinary literals in $P_i(k,x)$ with only key variables and **lnk** is the number of ordinary literals with non-key variables, and

$$
(14) \qquad P'_i(k,x) \leftarrow P'_{i,j}(k,x) \quad j = 1 \ldots \alpha
$$

Figure 2 shows the transition rules of the example.

In the above set of rules (13) it will be useful to assume that the rule corresponding to j = 1 is:

$$
P'_{i,1}(k,x) \leftarrow O(L'_1) \wedge \ldots \wedge O(L'_n)
$$

For each derived predicate, we have obtained a set of rules, the transition rules, which define P'_i (new state) in terms of current state predicates and events. Applying these results to the definition of events predicates, we get the insertion, deletion and modification events rules.

T.1 $Emp32'_{1,1}(\underline{e},n,s) \leftrightarrow Emp(\underline{e},n,s) \wedge \neg\delta Emp(\underline{e},n,s) \wedge \neg\mu Emp(\underline{e},n,s,n',s') \wedge s>32$

T.2 $Emp32'_{1,2}(\underline{e},n,s) \leftrightarrow \iota Emp(\underline{e},n,s) \wedge s>32$

T.3 $Emp32'_{1,3}(\underline{e},n,s) \leftrightarrow \mu emp(\underline{e},n',s',n,s) \wedge s>32$

T.4 $Pemp'_{1,1}(\underline{e},s) \leftrightarrow Emp(\underline{e},n,s) \wedge \neg\delta Emp(\underline{e},n,s) \wedge \neg\mu Emp(\underline{e},n,s,n',s')$

T.5 $Pemp'_{1,2}(\underline{e},s) \leftrightarrow \iota Emp(\underline{e},n,s)$

T.6 $Pemp'_{1,3}(\underline{e},s) \leftrightarrow \mu emp(\underline{e},n',s',n,s)$

T.7 $Edn'_{1,1}(\underline{e},\underline{d},n) \leftrightarrow Ed(\underline{e},\underline{d}) \wedge \neg\delta Ed(\underline{e},\underline{d}) \wedge Dept(\underline{d},n) \wedge \neg\delta Dept(\underline{d},n) \wedge \neg\mu Dept(\underline{d},n,n')$

T.8 $Edn'_{1,2}(\underline{e},\underline{d},n) \leftrightarrow Ed(\underline{e},\underline{d}) \wedge \neg\delta Ed(\underline{e},\underline{d}) \wedge \iota Dept(\underline{d},n)$

T.9 $Edn'_{1,3}(\underline{e},\underline{d},n) \leftrightarrow Ed(\underline{e},\underline{d}) \wedge \neg\delta Ed(\underline{e},\underline{d}) \wedge \mu Dept(\underline{d},n',n)$

T.10 $Edn'_{1,4}(\underline{e},\underline{d},n) \leftrightarrow \iota Ed(\underline{e},\underline{d}) \wedge Dept(\underline{d},n) \wedge \neg\delta Dept(\underline{d},n) \wedge \neg\mu Dept(\underline{d},n,n')$

T.11 $Edn'_{1,5}(\underline{e},\underline{d},n) \leftrightarrow \iota Ed(\underline{e},\underline{d}) \wedge \iota Dept(\underline{d},n)$

T.12 $Edn'_{1,6}(\underline{e},\underline{d},n) \leftrightarrow \iota Ed(\underline{e},\underline{d}) \wedge \mu Dept(\underline{d},n',n)$

T.13 $User1n2'_{1,1}(\underline{u},n) \leftrightarrow User1(\underline{u},n) \wedge \neg\delta User1(\underline{u},n) \wedge \neg\mu User1(\underline{u},n,n') \wedge \neg User2(\underline{u},n) \wedge \neg \iota User2(\underline{u},n) \wedge \neg\mu User2(\underline{u},n'',n)$

T.14 $User1n2'_{1,2}(\underline{u},n) \leftrightarrow User1(\underline{u},n) \wedge \neg\delta User1(\underline{u},n) \wedge \neg\mu User1(\underline{u},n,n') \wedge \delta User2(\underline{u},n)$

T.15 $User1n2'_{1,3}(\underline{u},n) \leftrightarrow User1(\underline{u},n) \wedge \neg\delta User1(\underline{u},n) \wedge \neg\mu User1(\underline{u},n,n') \wedge \mu User2(\underline{u},n,n'')$

T.16 $User1n2'_{1,4}(\underline{u},n) \leftrightarrow \iota User1(\underline{u},n) \wedge \neg User2(\underline{u},n) \wedge \neg \iota User2(\underline{u},n) \wedge \neg\mu User2(\underline{u},n',n)$

T.17 $User1n2'_{1,5}(\underline{u},n) \leftrightarrow \iota User1(\underline{u},n) \wedge \delta User2(\underline{u},n)$

T.18 $User1n2'_{1,6}(\underline{u},n) \leftrightarrow \iota User1(\underline{u},n) \wedge \mu User2(\underline{u},n,n')$

T.19 $User1n2'_{1,7}(\underline{u},n) \leftrightarrow \mu User1(\underline{u},n',n) \wedge \neg User2(\underline{u},n) \wedge \neg \iota User2(\underline{u},n) \wedge \neg\mu User2(\underline{u},n'',n)$

T.20 $User1n2'_{1,8}(\underline{u},n) \leftrightarrow \mu User1(\underline{u},n',n) \wedge \delta User2(\underline{u},n)$

T.21 $User1n2'_{1,9}(\underline{u},n) \leftrightarrow \mu User1(\underline{u},n',n) \wedge \mu User2(\underline{u},n,n'')$

T.22 $User1emp'_{1,1}(\underline{u},n) \leftrightarrow User1(\underline{u},n) \wedge \neg\delta User1(\underline{u},n) \wedge \neg\mu User1(\underline{u},n,n') \wedge Emp(\underline{e},n,s) \wedge \neg \delta Emp(\underline{e},n,s) \wedge \neg\mu Emp(\underline{e},n,s,n'',s'')$

T.23 $User1emp'_{1,2}(\underline{u},n) \leftrightarrow User1(\underline{u},n) \wedge \neg\delta User1(\underline{u},n) \wedge \neg\mu User1(\underline{u},n,n') \wedge \iota Emp(\underline{e},n,s)$

T.24 $User1emp'_{1,3}(\underline{u},n) \leftrightarrow User1(\underline{u},n) \wedge \neg\delta User1(\underline{u},n) \wedge \neg\mu User1(\underline{u},n,n') \wedge \mu Emp(\underline{e},n'',s'',n,s)$

T.25 $User1emp'_{1,4}(\underline{u},n) \leftrightarrow \iota User1(\underline{u},n) \wedge Emp(\underline{e},n,s) \wedge \neg \delta Emp(\underline{e},n,s) \wedge \neg\mu Emp(\underline{e},n,s,n'',s'')$

T.26 $User1emp'_{1,5}(\underline{u},n) \leftrightarrow \iota User1(\underline{u},n) \wedge \iota Emp(\underline{e},n,s)$

T.27 $User1emp'_{1,6}(\underline{u},n) \leftrightarrow \iota User1(\underline{u},n) \wedge \mu Emp(\underline{e},n'',s'',n,s)$

T.28 $User1emp'_{1,7}(\underline{u},n) \leftrightarrow \mu User1(\underline{u},n',n) \wedge Emp(\underline{e},n,s) \wedge \neg \delta Emp(\underline{e},n,s) \wedge \neg\mu Emp(\underline{e},n,s,n'',s'')$

T.29 $User1emp'_{1,8}(\underline{u},n) \leftrightarrow \mu User1(\underline{u},n',n) \wedge \iota Emp(\underline{e},n,s)$

T.30 $User1emp'_{1,9}(\underline{u},n) \leftrightarrow \mu User1(\underline{u},n',n) \wedge \mu Emp(\underline{e},n'',s'',n,s)$

Figure 2. Transition rules of the example

3.3 Insertion events rules

Let P be a derived predicate. Insertion events for P were defined in (1) as:

$$\forall k,x\ (\ \iota P(k,x) \leftrightarrow P'(k,x) \wedge \neg\exists y P(k,y))$$

If there are m rules for predicate P, then $P'(k,x) \leftrightarrow P'_1(k,x) \vee \ldots \vee P'_m(k,x)$, and replacing $P'(k,x)$ we obtain the set of rules:

$$\iota P(k,x) \leftarrow P'_i(k,x) \wedge \neg\exists y P(k,y) \quad \text{with } i = 1 \ldots m$$

and replacing again $P'_i(k,x)$ by its equivalent definition given in (14) we get:

(15) $$\iota P(k,x) \leftarrow P'_{i,j}(k,x) \wedge \neg\exists y P(k,y) \quad \text{for } i = 1 \ldots m \text{ and } j = 1 \ldots \alpha$$

or the analogous rule:

(15') $$\iota P(k) \leftarrow P'_{i,j}(k) \wedge \neg P(k) \quad \text{for } i = 1 \ldots m \text{ and } j = 1 \ldots \alpha$$

Rules (15, 15') are called *insertion events rules* of predicate P. They allow us to deduce which ιP facts (induced insertions) happen in a transition.

We can remove some of these rules and, in some cases, simplify them (see [Urp 91]):

RI.1 For any i, the rule corresponding to j =1 cannot produce ιP facts, since in this case $P'_{i,1}(k,x) \rightarrow P(k,x)$ or $P'_{i,1}(k) \rightarrow P(k)$. We can then reduce the set (15) to:

(15'') $$\iota P(k,x) \leftarrow P'_{i,j}(k,x) \wedge \neg\exists y P(k,y) \quad \text{for } i = 1 \ldots m \text{ and } j = 2 \ldots \alpha$$

and similarly for the case of ιP(k).

SI.1 Rules in (15'') for which the transition rules corresponding to $P'_{i,j}(k,x)$ have some literal $N(L'_h)$ in $U(P'_{i,j})$ of type ιQ(k,x), ιQ(k) or δQ(k) can be simplified by removing P_i from P and becoming the rule (assuming that y has an existencial quantifier) :

(16) $$\iota P(k,x) \leftarrow P'_{i,j}(k,x) \wedge \neg P_1(k,y) \wedge \ldots \wedge \neg P_{i-1}(k,y) \wedge \neg P_{i+1}(k,y) \wedge \ldots \wedge \neg P_m(k,y) \quad \text{for } i = 1 \ldots m \text{ and } j = 2 \ldots \alpha$$

and similarly for the case of ιP(k). Note that we can always substitute $P'_{i,j}(k,x)$ by its equivalent definition given in (13).

SI.2 Rules in (15'') for which the transition rules corresponding to $P'_{i,j}(k,x)$ do not have a literal $N(L'_h)$ in $U(P'_{i,j})$ of type $\iota Q(k,x)$, $\iota Q(k)$ or $\delta Q(k)$, with $U(P'_{i,j}) \neq \varnothing$, can be rewritten as:

(17) $$\iota P(k,x) \leftarrow P'_{i,j}(k,x) \wedge \neg P_1(k,y) \wedge ... \wedge \neg P_{i-1}(k,y) \wedge simpl(\neg P_i(k,y)) \wedge \neg P_{i+1}(k,y) \wedge ... \wedge \neg P_m(k,y)$$

where $simpl(\neg P_i(k,y))$ is obtained as follows:

For each literal L'_h in $U(P'_{i,j})$ of type:

- $[\mu Q_j(k_j,x'_j,x_j) \mid \mu Q_j(k_j,x_j,x'_j)]$, we can substitute the corresponding literal from $\neg P_i(k,y)$, which is of the form $[Q_j(k_j,y_j) \mid \neg Q_j(k_j,y_j)]$, by $[\, y_j = x'_j \mid y_j \neq x_j]$

- $Q_j(k_j,x_j)$, we can substitute the corresponding literal from $\neg P_i(k,y)$, $Q_j(k_j,y_j)$, by $y_j = x_j$

- $[Q_j(k_j) \mid \neg Q_j(k_j)]$ we can remove the corresponding literal from $\neg P_i(k,y)$, $[Q_j(k_j) \mid \neg Q_j(k_j)]$

- $\delta Q_j(k_j,x_j)$ we can substitute the corresponding literal from $\neg P_i(k,y)$, $\neg Q_j(k_j,y_j)$, by $y_j \neq x_j$

and $P'_{i,j}(k,x)$ is subtituted by its equivalent definition given in (13).

FSP. Final simplification process

After applying SI.2, $\neg P_i(k,y)$ will contain the boolean expressions that we have incorporated. From these boolean expressions, their satisfactibility or insatisfactibility can sometimes be deduced for any value. For instance:

$P_i(k,y) \leftrightarrow (n \neq n)$ is insatisfiable for any value

$P_i(k,y) \leftrightarrow (n_y = n)$ where n_y is not used anywhere, is satisfiable for any value

Furthermore, $P'_{i,j}(k,x)$ can contain literals of type $[\mu Q_j(k_j,x'_j,x_j)]$. From these kind of literals we know that x'_j is different of x_j. It will be very useful to take this information into account. For instance:

$P_i(k,y) \leftrightarrow (n_y=n \wedge n_y = n')$ is insatisfiable for any value if we know that different(n,n')

$P_i(k,y) \leftrightarrow (n' \neq n)$ is satisfiable for any value if we know that different(n,n').

Additionally, in some cases, we can also reduce the rule. For instance:

$P_i(k,y) \leftrightarrow (s_y = s' \wedge s_y > 32)$ where s_y is not used anywhere, can be reduced to

$P_i(k,y) \leftrightarrow (s' > 32)$

We give in figure 3 the insertion events rules corresponding to the example of figure 1. As an example, we detail the steps to obtain Rule I.2:

First we apply the definition of ιEmp32 and we get:

$$\iota Emp32(\underline{e},n,s) \leftarrow Emp32'_{1,3}(\underline{e},n,s) \wedge \neg Emp32(\underline{e},n_y,s_y)$$

then we apply SI.2 getting:

$$\iota Emp32(\underline{e},n,s) \leftarrow \mu Emp(\underline{e},n',s',n,s) \wedge s{>}32 \wedge \neg(n_y{=}n' \wedge s_y{=}s' \wedge s_y{>}32)$$

and, finally, FSP is applied obtaining I.2.

I.1 $\iota Emp32(\underline{e},n,s) \leftarrow \iota Emp(\underline{e},n,s) \wedge s{>}32$

I.2 $\iota Emp32(\underline{e},n,s) \leftarrow \mu Emp(\underline{e},n',s',n,s) \wedge s{>}32 \wedge \neg(s' > 32)$

I.3 $\iota Pemp(\underline{e},s) \leftarrow \iota Emp(\underline{e},n,s)$

I.4 $\iota Edn(\underline{e},\underline{d},n) \leftarrow Edn'_{1,j}(\underline{e},\underline{d},n)$ where j = 2,4,5,6

I.5 $\iota User1n2(\underline{u},n) \leftarrow User1(\underline{u},n) \wedge \neg\delta User1(\underline{u},n) \wedge \neg\mu User1(\underline{u},n,n') \wedge \delta User2(\underline{u},n)$

I.6 $\iota User1n2(\underline{u},n) \leftarrow User1(\underline{u},n) \wedge \neg\delta User1(\underline{u},n) \wedge \neg\mu User1(\underline{u},n,n') \wedge \mu User2(\underline{u},n,n'')$

I.7 $\iota User1n2(\underline{u},n) \leftarrow User1n2'_{1,j}(\underline{u},n)$ where j = 4,5,6

I.8 $\iota User1n2(\underline{u},n) \leftarrow \mu User1(\underline{u},n',n) \wedge \neg User2(\underline{u},n) \wedge \neg\iota User2(\underline{u},n) \wedge \neg\mu User2(\underline{u},n'',n) \wedge User2(\underline{u},n')$

I.9 $\iota User1emp(\underline{u},n) \leftarrow User1(\underline{u},n) \wedge \neg\delta User1(\underline{u},n) \wedge \neg\mu User1(\underline{u},n,n') \wedge \iota Emp(\underline{e},n,s) \wedge \neg Emp(\underline{e_y},n,s_y)$

I.10 $\iota User1emp(\underline{u},n) \leftarrow User1(\underline{u},n) \wedge \neg\delta User1(\underline{u},n) \wedge \neg\mu User1(\underline{u},n,n') \wedge \mu Emp(\underline{e},n'',s'',n,s) \wedge \neg Emp(\underline{e_y},n,s_y)$

I.11 $\iota User1emp(\underline{u},n) \leftarrow User1emp'_{1,j}(\underline{u},n)$ where j = 4,5,6

I.12 $\iota User1emp(\underline{u},n) \leftarrow \mu User1(\underline{u},n',n) \wedge Emp(\underline{e},n,s) \wedge \neg\delta Emp(\underline{e},n,s) \wedge \neg\mu Emp(\underline{e},n,s,n'',s'') \wedge \neg Emp(\underline{e_y},n',s_y)$

I.13 $\iota User1emp(\underline{u},n) \leftarrow \mu User1(\underline{u},n',n) \wedge \iota Emp(\underline{e},n,s) \wedge \neg Emp(\underline{e_y},n',s_y)$

I.14 $\iota User1emp(\underline{u},n) \leftarrow \mu User1(\underline{u},n',n) \wedge \mu Emp(\underline{e},n'',s'',n,s) \wedge \neg Emp(\underline{e_y},n',s_y)$

Figure 3. Insertion events rules of the example

3.4 Deletion events rules

Let P be a derived predicate. Deletion events were defined in (2) as:

$$\forall k,x\ (\ \delta P(k,x) \leftrightarrow P(k,x) \wedge \neg\exists y P'(k,y))$$

if there are m rules for predicate P, we then have:

$$\delta P(k,x) \leftarrow P_i(k,x) \wedge \neg\exists y P'(k,y) \qquad \text{for } i = 1 \ldots m$$

and replacing $P'(k,y)$ by its equivalent definition $P'(k,y) \leftrightarrow P'_1(k,y) \vee \ldots \vee P'_m(k,y)$ we obtain

(18) $$\delta P(k,x) \leftarrow P_i(k,x) \wedge \neg\exists y P'_1(k,y) \wedge \ldots \wedge \neg\exists y P'_i(k,y) \wedge \ldots \wedge \neg\exists y P'_m(k,y)$$

or the analogous

(18') $$\delta P(k) \leftarrow P_i(k) \wedge \neg P'_1(k) \wedge \ldots \wedge \neg P'_i(k) \wedge \ldots \wedge \neg P'_m(k) \quad \text{for } i = 1 \ldots m$$

Rules (18,18') are called *deletion events rules* of predicate P. They allow us to deduce which δP facts (induced deletions) happen in a transition.

If $E(P_i) = \emptyset$, then after a number of tranformations described in [Urp 91], we can rewrite these rules, obtaining a simplified form of them (assume that there is an existencial quantifier for y):

For i =1 ... m
if $U(P_i) = L1 \wedge \ldots. \wedge Lp \wedge \ldots. \wedge Lq$ where 1..p literals with non-key variables
p+1..q literals without non-key variables
For j=1 ... p

(19) $$\delta P(k,x) \leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge \delta Q_j(k_j,x_j) \wedge L_{j+1} \wedge \ldots \wedge L_q \wedge \neg P'_1(k,y) \wedge \ldots \wedge \neg P'_{i-1}(k,y) \wedge \neg P'_{i+1}(k,y) \wedge \ldots \wedge \neg P'_m(k,y)$$

if L_j is a positive literal

(20) $$\delta P(k,x) \leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge \iota Q_j(k_j,x_j) \wedge L_{j+1} \wedge \ldots \wedge L_q \wedge \neg P'_1(k,y) \wedge \ldots \wedge \neg P'_{i-1}(k,y) \wedge \neg P'_{i+1}(k,y) \wedge \ldots \wedge \neg P'_m(k,y) \wedge simplif(\neg P'_{i,h}(k,y)) \quad \text{where } h = 2 \ldots \alpha$$

if L_j is a negative literal

(21) $\delta P(k,x) \leftarrow L_1 \wedge ... \wedge L_{j-1} \wedge [\mu Q_j(k_j,x_j,x'_j) \mid \mu Q_j(k_j,x'_j,x_j)] \wedge L_{j+1} \wedge ... \wedge L_q \wedge$
$\neg P'_1(k,y) \wedge ... \wedge \neg P'_{i-1}(k,y) \wedge \neg P'_{i+1}(k,y) \wedge ... \wedge \neg P'_m(k,y) \wedge$
$simplif(\neg P'_{i,h}(k,y))$ where $h = 2 ... \alpha$

where the first option is taken if L_j is positive and the second one if L_j is not

For j=p+1 ... q

(22) $\delta P(k,x) \leftarrow L_1 \wedge ... \wedge L_{j-1} \wedge [\delta Q_j(k_j) \mid \iota Q_j(k_j)] \wedge L_{j+1} \wedge ... \wedge L_q \wedge$
$\neg P'_1(k,y) \wedge ... \wedge \neg P'_{i-1}(k,y) \wedge \neg P'_{i+1}(k,y) \wedge ... \wedge \neg P'_m(k,y)$

where the first option is taken if L_j is positive and the second one if L_j is not

And the same for the analogous rules (18').

Now we are going to see how to construct the simplification of $\neg P'_{i,h}(k,y)$.

$\neg P'_{i,h}(k,y) \leftrightarrow \neg P'_{i,2}(k,y) \wedge ... \wedge \neg P'_{i,\alpha}(k,y)$

step 1. if we have substituted L_j by:

1. $\iota Q_j(k_j,x_j)$:
   we can remove $\neg P'_{i,j}(k,y)$ containing $\mu Q_j(k_j,y_j,y'_j)$ or $\delta Q_j(k_j,y_j)$
   we can simplify $\neg P'_{i,j}(k,y)$ containing $\neg Q_j(k_j,y_j) \wedge \neg \iota Q_j(k_j,y_j) \wedge \neg \mu Q_j(k_j,y'_j,y_j)$
   replacing this literal by $y_j \neq x_j$
2. $\mu Q_j(k_j,x_j,x'_j)$:
   we can remove $\neg P'_{i,j}(k,y)$ containing $Q_j(k_j,y_j) \wedge \neg \delta Q_j(k_j,y_j) \wedge \neg \mu Q_j(k_j,y_j,y'_j)$
   or $\iota Q_j(k_j,y_j)$
   we can simplify $\neg P'_{i,j}(k,y)$ containing $\mu Q_j(k_j,y'_j,y_j)$ replacing this literal by
   $y_j = x'_j \wedge y'_j = x_j$
3. $\mu Q_j(k_j,x'_j,x_j)$:
   we can remove $\neg P'_{i,j}(k,y)$ containing $\delta Q_j(k_j,y_j)$
   we can simplify $\neg P'_{i,j}(k,y)$ containing $\mu Q_j(k_j,y_j,y'_j)$ replacing this literal by
   $y_j = x'_j \wedge y'_j = x_j$
   we can simplify $\neg P'_{i,j}(k,y)$ containing $\neg Q_j(k_j,y_j) \wedge \neg \iota Q_j(k_j,y_j) \wedge \neg \mu Q_j(k_j,y'_j,y_j)$
   replacing this literal by $y_j \neq x'_j \wedge y_j \neq x_j$

step 2. For each literal L_l that we have not substituted if this literal is:

1. $Q_l(k_l,x_l)$ we can substitute from $\neg P'_{i,j}(k,y)$ the corresponding literal
   - 1.1. $Q_l(k_l,y_l) \wedge \neg\delta Q_l(k_l,y_l) \wedge \neg\mu Q_l(k_l,y_l,y'_l)$ by $y_l = x_l$
   - 1.2. $\iota Q_l(k_l,y_l)$ by false and so remove this $\neg P'_{i,j}(k,y)$
   - 1.3. $\mu Q_l(k_l,y'_l,y_l)$ by $\mu Q_l(k_l,x_l,y_l)$

2. $\neg Q_l(k_l,x_l)$ we can substitute from $\neg P'_{i,j}(k,y)$ the corresponding literal
   - 2.1. $\delta Q_l(k_l,y_l)$ by $y_l \neq x_l \wedge \delta Q_l(k_l,y_l)$
   - 2.2. $\mu Q_l(k_l,y_l,y'_l)$ by $y_l \neq x_l \wedge \mu Q_l(k_l,y_l,y'_l)$

3. $Q_l(k_l)$ we can substitute from $\neg P'_{i,j}(k,y)$ the corresponding literal
   - 3.1 $Q_l(k_l) \wedge \neg\delta Q_l(k_l)$ by nothing
   - 3.2 $\iota Q_l(k_l)$ by false and so remove this $\neg P'_{i,j}(k,y)$

4. $\neg Q_l(k_l)$ we can substitute from $\neg P'_{i,j}(k,y)$ the corresponding literal
   - 4.1 $\neg Q_l(k_l) \wedge \neg\, \iota Q_l(k_l)$ by nothing
   - 4.2 $\delta Q_l(k_l)$ by false and so remove this $\neg P'_{i,j}(k,y)$

step 3. Apply the simplification final process.

We give in figure 4 the deletion events rules corresponding to the example of figure 1. Note that the simplification rules can not be applied to DR.5, since E(User1emp) $\neq \varnothing$. As an example, we detail the steps to obtain Rule D.2:

First we apply (21) and we obtain:

$$\delta Emp32(\underline{e},n,s) \leftarrow \mu Emp(\underline{e},n,s,n',s') \wedge s{>}32 \wedge simplif(\neg Emp32'_{1,h}(\underline{e},n_y,s_y))$$

then we apply step1.2 part one and we get:

$$\delta Emp32(\underline{e},n,s) \leftarrow \mu Emp(\underline{e},n,s,n',s') \wedge s{>}32 \wedge simplif(\neg Emp32'_{1,3}(\underline{e},n_y,s_y))$$

applying step 1.2 part two we get:

$$\delta Emp32(\underline{e},n,s) \leftarrow \mu Emp(\underline{e},n,s,n',s') \wedge s{>}32 \wedge \neg(n'_y{=}n \wedge s'_y{=}s \wedge n_y{=}n' \wedge s_y{=}s' \wedge s_y{>}32\ )$$

finally, we apply FSP getting rule D.2

D.1 $\delta Emp32(\underline{e},n,s) \leftarrow \delta Emp(\underline{e},n,s) \wedge s>32$

D.2 $\delta Emp32(\underline{e},n,s) \leftarrow \mu Emp(\underline{e},n,s,n',s') \wedge s>32 \wedge \neg(s'>32)$

D.3 $\delta Pemp(\underline{e},s) \leftarrow \delta Emp(\underline{e},n,s)$

D.4 $\delta Edn(\underline{e},\underline{d},n) \leftarrow \delta Ed(\underline{e},\underline{d}) \wedge Dept(\underline{d},n)$

D.5 $\delta Edn(\underline{e},\underline{d},n) \leftarrow Ed(\underline{e},\underline{d}) \wedge \delta Dept(\underline{d},n)$

D.6 $\delta User1n2(\underline{u},n) \leftarrow \delta User1(\underline{u},n) \wedge \neg User2(\underline{u},n)$

D.7 $\delta User1n2(\underline{u},n) \leftarrow User1(\underline{u},n) \wedge \iota User2(\underline{u},n) \wedge \neg\mu User1(\underline{u},n,n_y)$

D.8 $\delta User1n2(\underline{u},n) \leftarrow \mu User1(\underline{u},n,n') \wedge \neg User2(\underline{u},n) \wedge$
$\neg(\neg User2(\underline{u},n') \wedge \neg\iota User2(\underline{u},n') \wedge \neg\mu User2(\underline{u},n_y'',n')) \wedge$
$\neg \delta User2(\underline{u},n') \wedge$
$\neg\mu User2(\underline{u},n',n_y''')$

D.9 $\delta User1n2(\underline{u},n) \leftarrow User1(\underline{u},n) \wedge \mu User2(\underline{u},n',n) \wedge \neg\mu User1(\underline{u},n,n') \wedge$
$\neg(\mu User1(\underline{u},n,n_y'') \wedge n_y'' \neq n')$

D.10 $\delta User1emp(\underline{u},n) \leftarrow User1emp(\underline{u},n) \wedge \neg User1emp'_{1,h}(\underline{u},n_y)$ for h = 2 ... 9

Figure 4. Deletion events rules of the example

3.5 Modification events rules

Let P be a derived predicate. Modification events for P were defined in (3) as:

$$\forall k,x,x' \; (\mu P(k,x,x') \leftrightarrow P(k,x) \wedge P'(k,x') \wedge x \neq x')$$

If there are m rules for predicate P, then $P'(k,x') \leftrightarrow P'_1(k,x') \vee ... \vee P'_m(k,x')$ and $P(k,x) \leftrightarrow P_1(k,x) \vee ... \vee P_m(k,x)$. Replacing $P'(k,x')$ and $P(k,x)$, and distributing $\wedge$ over $\vee$, we obtain the set of rules:

$$\mu P(k,x,x') \leftarrow P'_i(k,x') \wedge P_h(k,x) \wedge x \neq x' \qquad \text{with } i,h = 1 ... m$$

Considering $P'_i(k,x')$ as a positive literal, replacing it by its equivalent definition given in (4) and distributing $\wedge$ over $\vee$, we get:

(23) $\mu P(k,x,x') \leftarrow P_i(k,x') \wedge \neg\delta P_i(k,x') \wedge \neg\mu P_i(k,x',x'') \wedge P_h(k,x) \wedge x \neq x'$ with i,h = 1 ...m

(24) $\mu P(k,x,x') \leftarrow \mu P_i(k,x'',x') \wedge P_h(k,x) \wedge x \neq x'$ with i,h = 1 ... m

(25) $\mu P(k,x,x') \leftarrow \iota P_i(k,x') \wedge P_h(k,x) \wedge x \neq x'$ with i,h = 1 ... m

Rules (23) can be removed since, due the key integrity constraint, $P_i(k,x')$ is contradictory to $P_h(k,x) \wedge x \neq x'$.

Rules (24), (25) can be simplified as (see [Urp 91]):

(26) $\mu P(k,x,x') \leftarrow \mu P_i(k,x,x')$ with i = 1 ... m

(27) $\mu P(k,x,x') \leftarrow \iota P_i(k,x') \wedge \delta P_h(k,x) \wedge x \neq x'$ with i = 1 ... m; h = 1 ... m, except i

Rules (26), (27) are called *modification events rules* of predicate P. They allow us to deduce which μP facts (induced modifications) happen in a transition.

Replacing $\mu P_i(k,x,x')$ in (26) by its equivalent definition given in (3), we get:

(28) $\mu P(k,x,x') \leftarrow P'_i(k,x') \wedge P_i(k,x) \wedge x \neq x'$ for i = 1 ... m

and replacing again $P'_i(k,x)$ by its equivalent definition given in (14) we get:

(29) $\mu P(k,x,x') \leftarrow P'_{i,j}(k,x') \wedge P_i(k,x) \wedge x \neq x'$ for i = 1 ... m and j = 1 ... α

We can remove some of rules (29) and, in some cases, simplify them:

RM.1 For any i, the rule corresponding to j =1 cannot produce μP facts, since in this case we have $P'_{i,1}(k,x') \rightarrow P_i(k,x')$ which, due the key integrity constraint, is contradictory to the rest of the body $P_i(k,x) \wedge x \neq x'$. We can then reduce the set (29) to:

(30) $\mu P(k,x,x') \leftarrow P'_{i,j}(k,x') \wedge P_i(k,x) \wedge x \neq x'$ for i = 1 ... m and j = 2 ... α

RM.2 Rules in (30) for which the transition rules corresponding to $P'_{i,j}(k,x)$ have some literal $N(L'_h)$ in $U(P'_{i,j})$ of type $\iota Q(k,x)$, $\iota Q(k)$ or $\delta Q(k)$ can be removed, since SI.1, $P'_{i,j}(k,x') \rightarrow \neg\exists y P_i(k,y)$.

SM.2 Rules in (30) for which the transition rules corresponding to $P'_{i,j}(k,x)$ do not have a literal $N(L'_h)$ in $U(P'_{i,j})$ of type $\iota Q(k,x)$, $\iota Q(k)$ or $\delta Q(k)$, with $U(P'_{i,j}) \neq \varnothing$, can be rewritten as:

(31) $\mu P(k,x,x') \leftarrow P'_{i,j}(k,x') \wedge \mathit{simpl}(P_i(k,x)) \wedge x \neq x'$ for i = 1 ... m and j =2 ... α

where the process for obtaining $\mathit{simpl}(P_i(k,x))$ is quite similar to that used in insertion events rules and $P'_{i,j}(k,x)$ is subtituted by its equivalent definition given in (13). Finally, we also have to apply the final simplification process.

Note that rules (27) can only be applied when there is more than one rule that defines P(k,x). Also note that applying (1) to $\iota P_i(k,x')$ we get:

$$\iota P_i(k,x') \leftrightarrow P'_i(k,x') \wedge \neg\exists y P_i(k,y)$$

and applying (2) to $\delta P_i(k,x)$:

$$\delta P_i(k,x) \leftrightarrow P_i(k,x) \wedge \neg\exists y P'_i(k,y)),$$

and therefore, we can apply to $\iota P_i(k,x')$ and $\delta P_i(k,x)$ the simplification process described in 3.3 and 3.4, respectively. Thus, we can replace them by their corresponding simplified form.

In figure 5 we show the modification events rules corresponding to the example of figure 1. As an example, we detail the steps to obtain Rule M.4:

First we apply (26) and we get:

$$\mu User1n2(\underline{u},n,n1) \leftarrow \mu User1n2_1(\underline{u},n1)$$

applying (30) we obtain:

$$\mu User1n2(\underline{u},n,n1) \leftarrow User1n2'_{1,7}(\underline{u},n1) \wedge User1n2_1(\underline{u},n) \wedge n \neq n1$$

then we apply SM.2 getting:

$$\mu User1n2(\underline{u},n,n1) \leftarrow \mu User1(\underline{u},n',n1) \wedge \neg User2(\underline{u},n1) \wedge \neg \iota User2(\underline{u},n1) \wedge \neg \mu User2(\underline{u},n'',n1) \wedge n=n' \wedge \neg User2(\underline{u},n) \wedge n \neq n1$$

and, finally, FSP is applied obtaining M.4.

M.1 $\mu Emp32(\underline{e},n,s,n1,s1) \leftarrow \mu Emp(\underline{e},n,s,n1,s1) \wedge s1 \rangle 32 \wedge s \rangle 32$

M.2 $\mu Pemp(\underline{e},s,s1) \leftarrow \mu Emp(\underline{e},n,s,n1,s1) \wedge s \neq s1$

M.3 $\mu Edn(\underline{e},\underline{d},n,n1) \leftarrow Ed(\underline{e},\underline{d}) \wedge \neg \delta Ed(\underline{e},\underline{d}) \wedge \mu Dept(\underline{d},n,n1)$

M.4 $\mu User1n2(\underline{u},n,n1) \leftarrow \mu User1(\underline{u},n,n1) \wedge \neg User2(\underline{u},n1) \wedge \neg \iota User2(\underline{u},n1) \wedge \neg \mu User2(\underline{u},n'',n1) \wedge \neg User2(\underline{u},n)$

M.5 $\mu User1n2(\underline{u},n,n1) \leftarrow \mu User1(\underline{u},n,n1) \wedge \delta User2(\underline{u},n1)$

M.6 $\mu User1n2(\underline{u},n,n1) \leftarrow \mu User1(\underline{u},n,n1) \wedge \mu User2(\underline{u},n1,n'')$

M.7 $\mu User1emp(\underline{u},n,n1) \leftarrow \mu User1(\underline{u},n,n1) \wedge Emp(\underline{e},n1,s) \wedge \neg \delta Emp(\underline{e},n1,s) \wedge \neg \mu Emp(\underline{e},n,s1,n'',s'') \wedge Emp(\underline{e_y},n,s_y)$

M.8 $\mu User1emp(\underline{u},n,n1) \leftarrow \mu User1(\underline{u},n,n1) \wedge \iota Emp(\underline{e},n1,s) \wedge Emp(\underline{e_y},n,s_y)$

M.8 $\mu User1emp(\underline{u},n,n1) \leftarrow \mu User1(\underline{u},n,n1) \wedge \mu Emp(\underline{e},n'',s'',n1,s1) \wedge Emp(\underline{e_y},n,s_y)$

Figure 5. The modifications events rules of the example

4. COMPARISON WITH [RCB 89]

One of the problems addressed in the HiPAC project [MD 89] is condition monitoring in active database systems [RCB 89]. Rosenthal, Chakravarthy and Blaustein study the expression and evaluation of a single situation. A situation describes a logical condition to be evaluated when one or more set of pre-defined events occur (for instance: database changes or temporal events). The condition part of a situation is a relational expression whose inputs are information from events (signal relations) and zero or more database relations. Each situation is associated with a set of actions to be fired when the condition succeeds.

Each relation has a special attribute (denoted tid) that provides a unique immutable identifier. This tid attribute is used to connect tuples that hold values of the same object before and after changes. In our case, instead of tid attribute, we provide key attributes to obtain the same effect.

To express situations involving database changes, they introduce the concept of Δrelation and operators to manipulate it. The main use of ΔR is to represent the net effect of a collection of updates to a relation. In such cases, if the "before value" is the relation R, the updates are represented as ΔR, and the updated relation is denoted by R'.

For each relation schema $\mathbf{R} \equiv (\text{tid}, A_1, ..., A_n)$ the schema of the ΔR associated relation is $\Delta\mathbf{R} \equiv (\breve{}\text{tid}, \breve{}A_1, ..., \breve{}A_n, \text{tid}\breve{}, A_1\breve{}, ..., A_n\breve{})$. Tuples to be inserted into R are tuples from ΔR with all ˘-prefix attributes null; tuples to be deleted from R are tuples from ΔR with all ˘-sufix attributes null; tuples to be modified from R are tuples from ΔR with both ˘-prefix attributes and ˘-sufix attributes non-null.

They introduce the changes operator to express how a derived relation changes when at least one of its input relations changes. The output of the changes operator is a ΔRelation that expresses all net changes produced in that derived relation.

As an example, assume the relation Emp and the view Emp32 defined as in DR.1. To monitor changes to view Emp32, they define the following situation:

Event:	Update to Emp
Condition:	changes(Emp32; [Emp, ΔEmp])

The intended meaning is that when an update to relation Emp occurs, we have to test whether that update affects the view Emp32.

In our method, changes produced in a derived or base predicates are captured by the insertion, deletion and modification events predicates. So we can express the same situation as:

Event:	Update to Emp
Condition:	$\iota Emp32(\underline{e},n,s)$ or $\delta Emp32(\underline{e},n,s)$ or $\mu Eemp32(\underline{e},n,s,n1,s1)$

of course, we can write any combination of sub-conditions. So, if we were only interested in monitoring insertions to Emp32, we could write a condition in which only $\iota Emp32(\underline{e},n,s)$ would appear.

To improve the efficiency of the changes operator, they introduce an incremental implementation of it when the expression that defines the derived relation contains a select, project or join operator. They also present the concept of chain rule: a transformation that is used to create an incremental form of the algebraic expression that defines a view. However, they only present the transformation for the case where the root of the expression is an unary operator.

With the incremental project (where tid attribute must belong to the projected attributes) and incremental select operators, they achieve an implementation of changes operator that does not reference the base relation. In our method we achieve the same goal with key attributes belonging to the projected attributes. As an example, consider the "Emp32" and "Pemp" derived predicates defined as a select and project expressions, respectively. Note that the corresponding events rules do not reference base predicates:

Rules corresponding to incremental select:

I.1 $\iota Emp32(\underline{e},n,s) \leftarrow \iota Emp(\underline{e},n,s) \wedge s>32$

I.2 $\iota Emp32(\underline{e},n,s) \leftarrow \mu Emp(\underline{e},n',s',n,s) \wedge s>32 \wedge \neg(s'>32)$

M.2 $\mu Emp32(\underline{e},n,s,n1,s1) \leftarrow \mu Emp(\underline{e},n,s,n1,s1) \wedge s>32 \wedge s1>32$

D.1 $\delta Emp32(\underline{e},n,s) \leftarrow \delta Emp(\underline{e},n,s) \wedge s>32$

D.2 $\delta Emp32(\underline{e},n,s) \leftarrow \mu Emp(\underline{e},n,s,n',s') \wedge s>32 \wedge \neg(s'>32)$

Rules corresponding to incremental project:

I.3 $\iota Pemp(\underline{e},s) \leftarrow \iota Emp(\underline{e},n,s)$

M.3 $\mu Pemp(\underline{e},s,s1) \leftarrow \mu Emp(\underline{e},n,s,n1,s1) \wedge s \neq s1$

D.3 $\delta Pemp(\underline{e},s) \leftarrow \delta Emp(\underline{e},n,s)$

The main advantage of the Events Method is that it allows more expressiveness in the representation of derived predicates: we can apply our method to more general derived predicates. As an example, we can have derived predicates defined:

- With the negation operator: DR.4 $User1n2(\underline{u},n) \leftarrow User1(\underline{u},n) \wedge \neg User2(\underline{u},n)$ and apply our method obtaining rules I.5, I.6, I.7, I.8, M.4, M.5, M.6, D.6, D.7,D.8 and D.9.

- With more than one rule (with the binary union operator as a root of the expression). For instance, if $User(\underline{u},n)$ is defined with two rules:

$$User(\underline{u},n) \leftarrow User1(\underline{u},n)$$
$$User(\underline{u},n) \leftarrow User2(\underline{u},n)$$

applying our method by:

first, renaming the conclusion of the rules, changing the implication and adding the corresponding set of rules:

$$User_1(\underline{u},n) \leftrightarrow User1(\underline{u},n)$$
$$User_2(\underline{u},n) \leftrightarrow User2(\underline{u},n)$$
$$User(\underline{u},n) \leftarrow User_1(\underline{u},n)$$
$$User(\underline{u},n) \leftarrow User_2(\underline{u},n)$$

and then getting the transition rules:

$$User'_{1,1}(\underline{u},n) \leftrightarrow User1(\underline{u},n) \wedge \neg\delta User1(\underline{u},n) \wedge \neg\mu User1(\underline{u},n,n')$$
$$User'_{1,2}(\underline{u},n) \leftrightarrow \iota User1(\underline{u},n)$$
$$User'_{1,3}(\underline{u},n) \leftrightarrow \mu User1(\underline{u},n',n)$$
$$User'_{2,1}(\underline{u},n) \leftrightarrow User2(\underline{u},n) \wedge \neg\delta User2(\underline{u},n) \wedge \neg\mu User2(\underline{u},n,n')$$
$$User'_{2,2}(\underline{u},n) \leftrightarrow \iota User2(\underline{u},n)$$
$$User'_{2,3}(\underline{u},n) \leftrightarrow \mu User2(\underline{u},n',n)$$

we obtain the events rules:

$$\iota User(\underline{u},n) \leftarrow \iota User1(\underline{u},n) \wedge \neg User_2(\underline{u},n_y)$$
$$\iota User(\underline{u},n) \leftarrow \iota User2(\underline{u},n) \wedge \neg User_1(\underline{u},n_y)$$
$$\mu User(\underline{u},n,n1) \leftarrow \iota User1(\underline{u},n1) \wedge \delta User2(\underline{u},n) \wedge n \neq n1$$
$$\mu User(\underline{u},n,n1) \leftarrow \mu User1(\underline{u},n,n1)$$
$$\mu User(\underline{u},n,n1) \leftarrow \iota User2(\underline{u},n1) \wedge \delta User1(\underline{u},n) \wedge n \neq n1$$
$$\mu User(\underline{u},n,n1) \leftarrow \mu User2(\underline{u},n,n1)$$
$$\delta User(\underline{u},n) \leftarrow \delta User1(\underline{u},n) \wedge \neg User'_{2j}(\underline{u},n_y) \qquad j = 1 \ldots 3$$
$$\delta User(\underline{u},n) \leftarrow \delta User2(\underline{u},n) \wedge \neg User'_{1j}(\underline{u},n_y) \qquad j = 1 \ldots 3$$

Furthermore, our rules incorporate the knowledge of keys of predicates. This allow us to obtain a set of rules, which are semantically richer, that fit to each particular situation. We use the "Edn" derived predicate to see this advantage:

When we have:

DR.3 $Edn(\underline{e,d},n) \leftarrow Ed(\underline{e,d}) \wedge Dept(\underline{d},n)$

we have seen that the corresponding events rules are:

I.4 $\iota Edn(\underline{e,d},n) \leftarrow Edn'_{1,j}(\underline{e,d},n)$ $\quad j = 2,4,5,6$

M.3 $\mu Edn(\underline{e,d},n,n1) \leftarrow Ed(\underline{e,d}) \wedge \neg\delta Ed(\underline{e,d}) \wedge \mu Dept(\underline{d},n,n1)$

D.4 $\delta Edn(\underline{e,d},n) \leftarrow \delta Ed(\underline{e,d}) \wedge Dept(\underline{d},n)$

D.5 $\delta Edn(\underline{e,d},n) \leftarrow Ed(\underline{e,d}) \wedge \delta Dept(\underline{d},n)$

now, assuming that our knowledge of keys changes:

DR.3' $Edn(\underline{e,d,n}) \leftarrow Ed(\underline{e,d}) \wedge Dept(\underline{d,n})$

the new events rules are:

R'.1 $\iota Edn(\underline{e,d,n}) \leftarrow Ed(\underline{e,d}) \wedge \neg\delta Ed(\underline{e,d}) \wedge \iota Dept(\underline{d,n})$

R'.2 $\iota Edn(\underline{e,d,n}) \leftarrow \iota Ed(\underline{e,d}) \wedge Dept(\underline{d,n}) \wedge \neg\delta Dept(\underline{d,n})$

R'.3 $\iota Edn(\underline{e,d,n}) \leftarrow \iota Ed(\underline{e,d}) \wedge \iota Dept(\underline{d,n})$

R'.4 $\delta Edn(\underline{e,d,n}) \leftarrow \delta Ed(\underline{e,d}) \wedge Dept(\underline{d,n})$

R'.5 $\delta Edn(\underline{e,d,n}) \leftarrow Ed(\underline{e,d}) \wedge \delta Dept(\underline{d,n})$

As we can observe, not only there are less R'.i rules but these rules are either equal or simpler than before.

In [RCB 89] the concept of key is not used and so the incremental forms generated for rule DR.3 and DR.3' are exactly the same. The implication of this fact is a loss of expressiveness, and also a loss of efficiency, since the incremental form for DR.3' should be simpler than that generated for DR.3.

Finally, as in [RCB 89], we can exploit the expected small number of facts of insertion, deletion and modification events predicates in order to improve the efficiency. If we evaluated the rules in Prolog, we could rewrite them in order to evaluate the events predicates first.

5. CONCLUSIONS

We have presented a method to monitor changes in deductive databases. The method is based on the events and transition rules, which explicitly define insertions, deletions and modifications induced by a database update.

Comparing our method with other works, we have shown that the use of the events method allows us to have a high degree of expresiveness in the representation of derived predicates: allowing more general derived predicates and incorporating the concept of key in their definition.

Futhermore, as we have shown, the expresiveness improvements can also result in improvements of efficiency.

On the other hand, in order to obtain an even more reduced set of rules, we believe that new simplifications can be found. Therefore, this will be a hopeful area for futur work.

Acknowledgements

I would like to thank A. Olivé who encouraged pursuance of this work and also D. Costal, E. Mayol, J.A. Pastor, C. Quer, M.R. Sancho, J.Sistac and E. Teniente for many useful comments and discussions.

This work has been partially supported by the CICYT PRONTIC program project TIC 680.

REFERENCES

[BaR 86] F. Bancilhom and R. Ramakrishan. "An amateur's introduction to recursive query processing strategies". Proc. ACM SIGMOD conf. on Management of data. Washington D.C., May 1986, pp. 16-52.

[BLT 86] J.A. Blakeley, P. Larson and F.W. Tompa. "Efficiently updating materialized views". Proc. ACM SIGMOD conf. on Management of data, Washington D.C., May 1986, pp. 61-71.

[Han 89] E.N. Hanson, "An initial report on the design of Ariel: a DBMS with an integrated production rules system". SIGMOD RECORD, Special Issue on Rule Managament and Processing in Expert Database Systems, Vol. 18, No. 3, September 1989, pp. 12-19.

[MD 89] D.R. McCarthy and U. Dayal, "The architecture of an active database management system," ACM SIGMOD Conf., San Francisco, California, May 1989, pp. 220-226.

[Oli 89] A. Olivé, "On the design and implementation of intormation systems from deductive conceptual models," Proc. of the 15th VLDB Conf., Amsterdam, 1989, pp. 3-11.

[Oli 91] A. Olivé, " Integrity constraints checking in deductives databases," to appear in the proc. of the 17th. VLDB , Barcelona, 1991.

[RCB 89] A. Rosenthal, S. Chakravarthy and B. Blaustein, "Situation monitoring for active databases", VLDB Conf., Amsterdam, 1989, pp. 455-464.

[SaK 88] F. Sadri and R. Kowalski. " A theorem-proving approach to database integrity". In Minker, J (Ed.) " Foundations of deductive databases and logic programing", Morgan Kaufmann Pub., 1988, pp 313-362.

[Sto 90] M. Stonebraker et al., "On rules, procedures, caching and views in data base systems", ACM SIGMOD Conf., May 1990, pp. 281-290.

[Urp 91] T. Urpí, "An approach to monitoring changes in deductive databases". Technical Report LSI-91-23.

[WiF 90] J. Widom and S.J. Finkelstein, "Set-oriented production rules in relational database systems", ACM SIGMOD Conf., May 1990, pp. 259-270.