A Method for Change Computation in Deductive Databases

Toni Urpí Antoni Olivé

Universitat Politècnica de Catalunya
Facultat d'Informàtica
Pau Gargallo, 5
E 08028 Barcelona - Catalonia

Abstract

Change computation is an essential component in several capabilities of a deductive database, such as integrity constraints checking, materialized view maintenance and condition monitoring. In this paper, we present a general method for change computation, which is based on the use of transition and internal events rules. These rules explicitly define the insertions, deletions and modifications induced by a database update. Standard SLDNF resolution can be used to compute the induced changes, but other procedures could be used as well. Our method generalizes and extends previous work on change computation methods, and in some cases computes changes in a more efficient way.

1 Introduction

Deductive databases generalize relational databases by including not only base predicates (or relations), but also derived predicates (or views). A derived predicate is defined by means of one or more deductive rules.

In a deductive database, an update to base predicates may induce changes on one or more derived predicates. Change computation refers to the process of computing the changes induced by an update. The obvious way to compute changes would be to evaluate derived predicates in the states before and after the update, and to compute the differences between the two states. However, this can be very inefficient in most cases.

Efficient change computation is essential in several capabilities of a deductive database, such as integrity constraints checking [BMM90], view maintenance [CeW91] and condition or situation monitoring [RCB+89], and several methods have been proposed in the past years. Some methods are specific for a particular problem, but others are more general. Methods for change computation can be analyzed in terms of: (1) What kind of changes are defined?; (2) When are changes computed?; and (3) How are changes computed?. Some methods make a distinction between "potential" changes and "real" changes induced by an update [Küc91], but we are only interested here in the computation of real changes, representing the net effect of an update.

Most of the methods have been developed as part of methods for integrity constraints checking. We can only mention two of them here, and refer to [BMM90] for a state-of-the-art survey. The method described in [BDM88,BrD88] defines insertion and deletion changes. An insertion occurs when a fact is true in the updated state and false before, while a deletion occurs when a fact is false in the updated state and true before. Thus, only "real" or "net" changes are computed, and the computation is performed before the database is updated. Changes are computed using expressions derived from an analysis of deductive rules. A similar method is given in [Oli91], where in some cases the derived expressions are more simplified.

Incremental methods for view maintenance also compute changes induced by an update on some materialized view. A method where views are specified using a standard query language, and considering arbitrary database updates, is given in [CeW91]. The method defines insertion and deletion changes of a materialized view as before, but changes are computed once base relations have been updated. Changes are computed by production rules [WiF90,WCL91] derived from an analysis of the view definition. Key constraints of base relations are also taken into account. As an example of a more specialized work, we mention [BCL89] where a method is presented to determine irrelevant updates (cannot change a view) and autonomously computable updates (the view can be updated using the view itself and the update).

Proceedings of the 18th VLDB Conference Vancouver, Bristish Columbia, Canada 1992

Change computation has also been used for situation monitoring in active databases. [BuC79] is one of the earlier works in this field, describing a method for detecting that a change in base relations cannot induce a change on an alerter. [RCB+89] presents a method developed as part of the HiPAC DBMS [CBB+89]. They consider not only insertion and deletions changes, as before, but also modification changes. Each tuple of a relation (base or derived) has an attribute that provides a unique immutable identifier, so that a tuple is modified if some of its attributes change. The method derives expressions for computing induced changes, again from an analysis of the definition of the derived predicate.

We present here a general method for change computation that can be applied in all database capabilities discussed above. The method takes into account key constraints of base and derived predicates. This allows us to define insertion, deletion and modification changes of derived predicates, where modification is defined as a change in some non-key argument of a predicate. The method computes the changes once the database has been updated. The changes are computed using expressions that are more simplified than those obtained in the previous methods, thus providing more efficient ways of change computation. The expressions are derived at compilation time, and evaluated when the database is updated.

The paper is organized as follows. Next Section defines basic concepts of deductive databases. In Section 3 we present the concept of internal event, a key concept of our method. Internal events capture in a natural way the notion of change. We also present the transition and internal events rules. Transition rules relate the old database state with the new state and the events that have occurred in a transition. Internal events rules define the conditions upon which an internal event happens. These rules are a particular application of the rules that we developed for the design of information systems [Oli89]. In Section 4 we show how internal events rules can be simplified. We give a set of simplifications that allow us to obtain simplified expressions for change computation. Then, in Section 5 we present our method for change computation, which can be based on the use of standard SLDNF resolution. We also point out some optimization techniques that can be applied. Our method is compared with some of the previous work in Section 6. Finally, we give in Section 7 the conclusions and point out future research.

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 (EDB), and the set of deductive rules is called the Intensional Database (IDB).

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 each database predicate (base or derived) has a non-null vector of arguments, **k**, that form a key for the predicate. We have then two types of predicates: those, P(**k**,**x**), with key and non-key arguments and those, P(**k**), with only key arguments, where both **k** and **x** are vectors.

2.1 Deductive Rules

A deductive rule is a formula of the form:

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

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

Condition predicates may be ordinary or evaluable. The former are base or derived predicates, while the latter are predicates, such as the comparison or arithmetic predicates, that can be evaluated without accessing the database.

As usual, we require that the database before and after any update is *allowed* [Llo 87], that is any variable that occurs in a deductive rule has an occurrence in a positive condition of an ordinary predicate. This ensures that all negative conditions can be fully instantiated before they are evaluated by the "negation as failure" rule.

In this paper we deal with stratified databases [ABW 88]. A database is stratified if the set of its predicate symbols can be partitioned into a finite set of classes, say $S_0, \dots, S_n$ such that for every deductive rule P ← Conditions, with $P \in S_j$,

(i) if $Q \in S_i$ is the predicate symbol of a positive condition of P, then $i \leq j$, and

(ii) if $Q \in S_i$ is the predicate symbol of a negative condition of P, then $i < j$.

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:

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

where the L_i are literals, and variables are assumed to be universally quantified over the whole formula. More general constraints can be transformed into this form as described in [LIT 84]. For the sake of uniformity, we (as in [DaW 89, Kow 78]) associate to each integrity constraint an inconsistency predicate Icn and thus it has the same form as the deductive rule. We call them integrity rules.

To enforce the concept of key we assume that associated to each $P(\mathbf{k},\mathbf{x})$ there is a key integrity constraint that we define as:

$$Icn(\mathbf{k}) \leftarrow P(\mathbf{k},\mathbf{x}) \wedge P(\mathbf{k},\mathbf{x}') \wedge \mathbf{x} \neq \mathbf{x}'$$

For example, if the EDB has the predicate Employee(emp,dept), the key integrity rule stating that emp forms a key for the predicate would be:

$$Ic1(\underline{emp}) \leftarrow Employee(\underline{emp},dept) \wedge Employee(\underline{emp},dept') \wedge dept \neq dept'$$

Note that, for clarity, we underline the key arguments of each predicate.

Keys of derived predicates can be deduced from the deductive rules of these predicates, using a procedure similar to that presented in [Dat90, chapters 19,20]

3 Transition and Internal Events Rules

In this section we define the events, a key concept in our method. We also explain how to derive the transition and internal events rules for a given database. These rules depend only on the deductive rules. They are independent from the base facts stored in the database. In a later section we will discuss the use of these rules for change computation.

We extend here the work reported in [Oli 91] in three directions. First, we define not only insertions and deletions, but also modifications of base and derived predicates. Usually, modifications are handled as deletions followed by insertions, but handling them as a base concept allows to improve efficiency. Second, we change the definition of transition rules to deal with the case where induced changes must be computed once base predicates have been updated. And third, we take into account key information.

3.1 Events

Let $D°$ be a database, U an update and D the updated database. We say that U induces a transition from $D°$ (the old state) to D (the new state). We assume for the moment that U consists of an unspecified set of base facts that have been inserted, deleted and/or modified.

Due to the deductive rules, U may induce other updates on some derived predicates. Let P be a derived predicate, and let $P°$ and P denote the evaluation of P in $D°$ and D, respectively. Assuming that $P°(K,X)$ holds in $D°$, where $\mathbf{K}$ and $\mathbf{X}$ are vectors of constants, three cases are possible:

a.1 $P(\mathbf{K},\mathbf{X})$ also holds in D
a.2 $\neg\exists y$ such that $P(\mathbf{K},\mathbf{y})$ holds in D
a.3. $\exists \mathbf{x}'$, such as $\mathbf{X}'$, for which $P(\mathbf{K},\mathbf{X}')$ and $\mathbf{X}\neq\mathbf{X}'$ holds in D

and assuming that $P(\mathbf{K},\mathbf{X})$ holds in D, three cases are also possible:

b.1 $P°(\mathbf{K},\mathbf{X})$ also holds in $D°$
b.2 $\neg\exists \mathbf{y}$ such that $P°(\mathbf{K},\mathbf{y})$ holds in $D°$
b.3. $\exists \mathbf{x}'$, such as $\mathbf{X}'$, for which $P°(\mathbf{K},\mathbf{X}')$ and $\mathbf{X}\neq\mathbf{X}'$ holds in $D°$

In case a.2 we say that a deletion internal event occurs in the transition, and we denote it by $\delta P(\mathbf{K},\mathbf{X})$. In case b.2 we say that an insertion internal events occurs in the transition, and we denote it by $\iota P(\mathbf{K},\mathbf{X})$. In cases a.3 and b.3 we say that a modification internal event occurs in the transition, and we denote it by $\mu P(\mathbf{K},\mathbf{X},\mathbf{X}')$ and $\mu P(\mathbf{K},\mathbf{X}',\mathbf{X})$, respectively.

Formally, we associate to each derived predicate P an insertion and a deletion internal event predicate defined as:

(1) $\forall \mathbf{k},\mathbf{x}\ (\iota P(\mathbf{k},\mathbf{x}) \leftrightarrow P(\mathbf{k},\mathbf{x}) \wedge \neg\exists \mathbf{y} P°(\mathbf{k},\mathbf{y}))$
(2) $\forall \mathbf{k},\mathbf{x}\ (\delta P(\mathbf{k},\mathbf{x}) \leftrightarrow P°(\mathbf{k},\mathbf{x}) \wedge \neg\exists \mathbf{y} P(\mathbf{k},\mathbf{y}))$

where $\mathbf{k}$ and $\mathbf{x}$ are vectors of variables.

Furthermore, we associate to each derived predicate P with non-key arguments, a modification internal event predicate defined as:

(3) $\forall \mathbf{k},\mathbf{x},\mathbf{x}'\ (\mu P(\mathbf{k},\mathbf{x},\mathbf{x}') \leftrightarrow P°(\mathbf{k},\mathbf{x}) \wedge P(\mathbf{k},\mathbf{x}') \wedge \mathbf{x}\neq\mathbf{x}')$

We handle the modification of a key as a deletion $\delta P(\mathbf{k},\mathbf{x})$ and an insertion $\iota P(\mathbf{k}',\mathbf{x})$.

From the above, we then have the equivalences:

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

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

which relate the old state with the new state and the internal events induced in the transition.

We also use definition (1), (2) and (3) above for base predicates. In this case, ιP, δP and μP facts represent the external events (given by the update) corresponding to insertion, deletion and modifications of base facts, respectively. Therefore, we assume from now on that U consists of an unspecified set of insertion and/or deletion and/or modification external events. Notice that by (1), (2) and (3) we require:

(6) $\forall \mathbf{k},\mathbf{x}$ $(\iota P(\mathbf{k},\mathbf{x}) \rightarrow \neg \exists y P°(\mathbf{k},\mathbf{y}))$ and
(7) $\forall \mathbf{k},\mathbf{x}$ $(\delta P(\mathbf{k},\mathbf{x}) \rightarrow P°(\mathbf{k},\mathbf{x}))$ and
(8) $\forall \mathbf{k},\mathbf{x},\mathbf{x'}$ $(\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \rightarrow P°(\mathbf{k},\mathbf{x}) \wedge \mathbf{x} \neq \mathbf{x'})$

also to hold for base predicates. Again, the μP predicate is defined only if P has non-key arguments. Due to this similar definition, we use sometimes the term "event" to denote either an internal or external event.

Example 1

Consider the following database D:

Base Facts
Person(John,19), Person(Ann,15), Person(Tom,20), Works(Tom)
Deductive rules
(E.1) Young(p,a) ← Person(p,a) ∧ a<20
(E.2) Student(p,a) ← Young(p,a) ∧ ¬Works(p)

Let the update be the set of external events U = { ιPerson(Mary,15), μPerson(John,19,20), μPerson(Ann,15,16) }. The internal events induced by U on Young are: ιYoung(Mary,15), δYoung(John,19) and μYoung(Ann,15,16) and the internal events induced on Student are: ιStudent(Mary,15), δStudent(John,19) and μStudent(Ann,15,16).

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≥1) such rules. For our purposes, we require to rename the predicate symbol in the conclusions of the m rules by $P_1 \dots P_m$ and add the set of clauses:

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

Consider now one of the rules $P_i(\mathbf{k},\mathbf{x}) \leftarrow L_1 \wedge \dots \wedge L_n$. When the rule is to be evaluated in the old state its form is $P°_i(\mathbf{k},\mathbf{x}) \leftarrow L°_1 \wedge \dots \wedge L°_n$ where $L°_r$ (r = 1...n) is obtained by replacing the predicate Q of L_r by Q°. Now, if we replace each literal in the body by its equivalent definition given in (4) or (5), we get a new rule, called a *transition rule*, which defines predicate $P°_i$ (old state) in terms of new state predicates and events.

More precisely, if $L°_r$ is an ordinary positive literal $Q°_r(\mathbf{k}_r,\mathbf{x}_r)$ we apply (4) and replace it by:

$$\begin{aligned}&(Q_r(\mathbf{k}_r,\mathbf{x}_r) \wedge \neg \iota Q_r(\mathbf{k}_r,\mathbf{x}_r) \wedge \neg \mu Q_r(\mathbf{k}_r,\mathbf{x'}_r,\mathbf{x}_r))\\ &\vee\ \delta Q_r(\mathbf{k}_r,\mathbf{x}_r)\\ &\vee\ \mu Q_r(\mathbf{k}_r,\mathbf{x}_r,\mathbf{x'}_r)\end{aligned}$$

and if $L°_r$ is an ordinary negative literal $\neg Q°_r(\mathbf{k}_r,\mathbf{x}_r)$ we apply (5) and replace it by:

$$\begin{aligned}&(\neg Q_r(\mathbf{k}_r,\mathbf{x}_r) \wedge \neg \delta Q_r(\mathbf{k}_r,\mathbf{x}_r) \wedge \neg \mu Q_r(\mathbf{k}_r,\mathbf{x}_r,\mathbf{x'}_r))\\ &\vee\ \iota Q_r(\mathbf{k}_r,\mathbf{x}_r)\\ &\vee\ \mu Q_r(\mathbf{k}_r,\mathbf{x'}_r,\mathbf{x}_r)\end{aligned}$$

If $L°_r$ is an evaluable predicate, we just replace $L°_r$ (positive or negative) by its new state version L_r.

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

$$\begin{aligned}U(L°_r) &= Q_r(\mathbf{k}_r,\mathbf{x}_r) \wedge \neg \iota Q_r(\mathbf{k}_r,\mathbf{x}_r) \wedge \neg \mu Q_r(\mathbf{k}_r,\mathbf{x'}_r,\mathbf{x}_r) && \text{if } L°_r = Q°_r(\mathbf{k}_r,\mathbf{x}_r)\\ &= \neg Q_r(\mathbf{k}_r,\mathbf{x}_r) \wedge \neg \delta Q_r(\mathbf{k}_r,\mathbf{x}_r) \wedge \neg \mu Q_r(\mathbf{k}_r,\mathbf{x}_r,\mathbf{x'}_r) && \text{if } L°_r = \neg Q°_r(\mathbf{k}_r,\mathbf{x}_r)\\ &= L_r && \text{if } L°_r \text{ is evaluable}\end{aligned}$$

$$\begin{aligned}D(L°_r) &= \delta Q_r(\mathbf{k}_r,\mathbf{x}_r) && \text{if } L°_r = Q°_r(\mathbf{k}_r,\mathbf{x}_r)\\ &= \iota Q_r(\mathbf{k}_r,\mathbf{x}_r) && \text{if } L°_r = \neg Q°_r(\mathbf{k}_r,\mathbf{x}_r)\end{aligned}$$

$$\begin{aligned}M(L°_r) &= \mu Q_r(\mathbf{k}_r,\mathbf{x}_r,\mathbf{x'}_r) && \text{if } L°_r = Q°_r(\mathbf{k}_r,\mathbf{x}_r)\\ &= \mu Q_r(\mathbf{k}_r,\mathbf{x'}_r,\mathbf{x}_r) && \text{if } L°_r = \neg Q°_r(\mathbf{k}_r,\mathbf{x}_r)\end{aligned}$$

Notice that all variables $\mathbf{x'}_r$ are new, that is, not used before.

$U(L°_r)$, $D(L°_r)$ and $M(L°_r)$ express condition for which $L°_r$ is true. $U(L°_r)$ corresponds to the case in which $L°_r$ is Unchanged in the transition. $D(L°_r)$ corresponds to the case in which $L°_r$ is Deleted, while $M(L°_r)$ corresponds to the case when $L°_r$ is Modified.

With this notation we then have:

$$(9)\ P°_i(\mathbf{k},\mathbf{x}) \leftrightarrow \bigwedge_{r=1}^{r=n} [\, U(L°_r) \vee D(L°_r) \vee M(L°_r) \mid U(L°_r) \,]$$

where the first option is taken if $L°_r$ is an ordinary literal and the second one if $L°_r$ is evaluable. After distributing ∧ over ∨, we get an equivalent set of transition rules, each of them with the general form:

$$(10)\ P°_{i,j}(\mathbf{k},\mathbf{x}) \leftarrow \bigwedge_{r=1}^{r=n} [\, U(L°_r) \mid D(L°_r) \mid M(L°_r) \,] \qquad j = 1 \dots \alpha$$

$$(11)\ P°_i(\mathbf{k},\mathbf{x}) \leftarrow P°_{i,j}(\mathbf{k},\mathbf{x}) \qquad j = 1 \dots \alpha$$

with $\alpha = 3^{nk_i} * 2^{k_i}$, where nk_i is the number of ordinary literals with non-key arguments, and k_i is the number of ordinary literals with only key arguments.

In the above set of rules (10) it will be useful to assume that the rule corresponding to j = 1 is:

$$(12)\quad P°_{i,1}(\mathbf{k},\mathbf{x}) \leftarrow U(L°_1) \wedge \dots \wedge U(L°_n)$$

Example 2

The transition rules corresponding to derived predicates Young and Student defined in Example 1 are:

(E.3) $Young°_{1,1}(p,a) \leftarrow Person(p,a) \wedge \neg \iota Person(p,a) \wedge \neg \mu Person(p,a',a) \wedge a<20$
(E.4) $Young°_{1,2}(p,a) \leftarrow \delta Person(p,a) \wedge a<20$
(E.5) $Young°_{1,3}(p,a) \leftarrow \mu Person(p,a,a') \wedge a<20$
with:
(E.6...8) $Young°_1(p,a) \leftarrow Young°_{1,j}(p,a) \quad j = 1 \dots 3$
(E.9) $Young°(p,a) \leftarrow Young°_1(p,a)$

(E.10) $\text{Student}^\circ_{1,1}(\underline{p},a) \leftarrow \text{Young}(\underline{p},a) \wedge \neg\, \iota\text{Young}(\underline{p},a) \wedge \neg\, \mu\text{Young}(\underline{p},a',a) \wedge \neg\text{Works}(\underline{p}) \wedge \neg\, \delta\text{Works}(\underline{p})$

(E.11) $\text{Student}^\circ_{1,2}(\underline{p},a) \leftarrow \text{Young}(\underline{p},a) \wedge \neg\, \iota\text{Young}(\underline{p},a) \wedge \neg\, \mu\text{Young}(\underline{p},a',a) \wedge \iota\text{Works}(\underline{p})$

(E.12) $\text{Student}^\circ_{1,3}(\underline{p},a) \leftarrow \delta\text{Young}(\underline{p},a) \wedge \neg\text{Works}(\underline{p}) \wedge \neg\, \delta\text{Works}(\underline{p})$

(E.13) $\text{Student}^\circ_{1,4}(\underline{p},a) \leftarrow \delta\text{Young}(\underline{p},a) \wedge \iota\text{Works}(\underline{p})$

(E.14) $\text{Student}^\circ_{1,5}(\underline{p},a) \leftarrow \mu\text{Young}(\underline{p},a,a') \wedge \neg\text{Works}(\underline{p}) \wedge \neg\, \delta\text{Works}(\underline{p})$

(E.15) $\text{Student}^\circ_{1,6}(\underline{p},a) \leftarrow \mu\text{Young}(\underline{p},a,a') \wedge \iota\text{Works}(\underline{p})$

with:

(E.16...21) $\text{Student}^\circ_1(\underline{p},a) \leftarrow \text{Student}^\circ_{1,j}(\underline{p},a)$ $j = 1...6$

(E.22) $\text{Student}^\circ(\underline{p},a) \leftarrow \text{Student}^\circ_1(\underline{p},a)$

Some transition and internal events rules are not allowed, due to the presence of some negative literals in their bodies. This could be solved with a minor transformation. For example, in rule E.3 replace $\neg\mu\text{Person}(\underline{p},a',a)$ by $\neg\text{Aux}(\underline{p},a)$ and add the rule: $\text{Aux}(\underline{p},a) \leftarrow \mu\text{Person}(\underline{p},a',a)$.

3.3 Insertion Internal Events Rules

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

$$\forall \mathbf{k},\mathbf{x}\ (\iota P(\mathbf{k},\mathbf{x}) \leftrightarrow P(\mathbf{k},\mathbf{x}) \wedge \neg\, \exists y P^\circ(\mathbf{k},\mathbf{y}))$$

If there are m rules for predicate P, we have:

(13) $\iota P(\mathbf{k},\mathbf{x}) \leftarrow P_i(\mathbf{k},\mathbf{x}) \wedge \neg\, \exists y P^\circ_1(\mathbf{k},\mathbf{y}) \wedge \ldots \wedge \neg\, \exists y P^\circ_i(\mathbf{k},\mathbf{y}) \wedge \ldots \wedge \neg\, \exists y P^\circ_m(\mathbf{k},\mathbf{y})$ $\quad i=1...m$

Notice that $P_i(\mathbf{k},\mathbf{x}) \wedge \neg\exists y P^\circ_i(\mathbf{k},\mathbf{y})$ represent insertion events of predicate P_i. Thus, we have:

(14) $\iota P(\mathbf{k},\mathbf{x}) \leftarrow \iota P_i(\mathbf{k},\mathbf{x}) \wedge \neg\, \exists y P^\circ_1(\mathbf{k},\mathbf{y}) \wedge \ldots \wedge \neg\, \exists y P^\circ_{i-1}(\mathbf{k},\mathbf{y}) \wedge \neg\, \exists y P^\circ_{i+1}(\mathbf{k},\mathbf{y}) \wedge \ldots \wedge \neg\, \exists y P^\circ_m(\mathbf{k},\mathbf{y})$ $\quad i=1...m$

(15) $\iota P_i(\mathbf{k},\mathbf{x}) \leftarrow P_i(\mathbf{k},\mathbf{x}) \wedge \neg\, \exists y P^\circ_i(\mathbf{k},\mathbf{y})$ $\quad i=1...m$

Rules (14) and (15) are called *insertion internal events rules* of predicate P and P_i, respectively. They allow us to deduce which ιP and ιP_i facts (induced insertions) happen in a transition.

In section 4.3 we show how rules (15) can be simplified.

3.4 Deletion Internal Events Rules

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

$$\forall \mathbf{k},\mathbf{x}\ (\delta P(\mathbf{k},\mathbf{x}) \leftrightarrow P^\circ(\mathbf{k},\mathbf{x}) \wedge \neg\, \exists y P(\mathbf{k},\mathbf{y}))$$

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

(16) $\delta P(\mathbf{k},\mathbf{x}) \leftarrow P^\circ_i(\mathbf{k},\mathbf{x}) \wedge \neg\, \exists y P_1(\mathbf{k},\mathbf{y}) \wedge \ldots \wedge \neg\, \exists y P_i(\mathbf{k},\mathbf{y}) \wedge \ldots \wedge \neg\, \exists y P_m(\mathbf{k},\mathbf{y})$ $\quad i=1...m$

Note that $P^\circ_i(\mathbf{k},\mathbf{x}) \wedge \neg\exists y P_i(\mathbf{k},\mathbf{y})$ represent deletion events of predicate P_i. Therefore, we have:

(17) $\delta P(\mathbf{k},\mathbf{x}) \leftarrow \delta P_i(\mathbf{k},\mathbf{x}) \wedge \neg\, \exists y P_1(\mathbf{k},\mathbf{y}) \wedge \ldots \wedge \neg\, \exists y P_{i-1}(\mathbf{k},\mathbf{y}) \wedge \neg\, \exists y P_{i+1}(\mathbf{k},\mathbf{y}) \wedge \ldots \wedge \neg\, \exists y P_m(\mathbf{k},\mathbf{y})$ $\quad i=1...m$

(18) $\delta P_i(\mathbf{k},\mathbf{x}) \leftarrow P^\circ_i(\mathbf{k},\mathbf{x}) \wedge \neg\, \exists y P_i(\mathbf{k},\mathbf{y})$ $\quad i=1...m$

Rules (17) and (18) are called *deletion internal events rules* of predicate P and P_i, respectively. They allow us to deduce which δP and δP_i facts (induced deletions) happen in a transition.

In section 4.1 we show how rules (18) can be simplified.

3.5 Modification Internal Events Rules

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

$$\forall \mathbf{k},\mathbf{x},\mathbf{x'}\ (\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftrightarrow P^\circ(\mathbf{k},\mathbf{x}) \wedge P(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'})$$

If there are m rules for predicate P we then have:

(20) $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow P^\circ_i(\mathbf{k},\mathbf{x}) \wedge P_h(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'}$ $\quad i,h=1...m$

Notice that $P^\circ_i(\mathbf{k},\mathbf{x}) \wedge P_i(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'}$ represent modifications events of predicate P_i. Therefore, we have:

(21) $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow P^\circ_i(\mathbf{k},\mathbf{x}) \wedge P_h(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'}$ $\quad i=1...m$, $h=1...m$ except i

(22) $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow \mu P_i(\mathbf{k},\mathbf{x},\mathbf{x'})$ $\quad i=1...m$

(23) $\mu P_i(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow P^\circ_i(\mathbf{k},\mathbf{x}) \wedge P_i(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'}$ $\quad i=1...m$

Rules (21)-(22) and (23) are called *modification internal events rules* of predicate P and P_i, respectively. They allow us to deduce which μP and μP_i facts (induced modifications) happen in a transition. Notice that rules (21) are defined only when m>1.

We show in section 4.2 how rules (21) and (23) can be simplified.

<u>Example 3</u>

The insertion, deletion and modification internal events rules corresponding to predicate Young defined in Example 1 are (the rules for Student are similar):

(E.23) $\iota\text{Young}(\underline{p},a) \leftarrow \text{Young}(\underline{p},a) \wedge \neg\, \exists y\text{Young}^\circ(\underline{p},y)$

(E.24) $\delta\text{Young}(\underline{p},a) \leftarrow \text{Young}^\circ(\underline{p},a) \wedge \neg\, \exists y\text{Young}(\underline{p},y)$

(E.25) $\mu\text{Young}(\underline{p},a,a') \leftarrow \text{Young}^\circ(\underline{p},a) \wedge \text{Young}(\underline{p},a') \wedge a \neq a'$

with the transition rules for $\text{Young}^\circ(\underline{p},a)$ given in Example 2.

4 Simplification of Internal Events Rules

In this section we introduce the simplifications that can be applied to the deletion, insertion and modification internal events rules. As mentioned in section 3, we can often simplify and even remove some of these rules. Applying these simplifications, we obtain a set of rules semantically equivalent to the former but with a smaller evaluation cost. In fact, we will see in section 6 that the application of our simplifications produces expressions that are more optimized that those obtained by other methods.

We need to introduce first some terminology. If P is a predicate defined with a single rule, we denote by B(P) the body of its defining rule. B(P) is a conjunction of one o more literals. We also denote by A\B A without B (true if A=B).

4.1 Simplification of Deletion Internal Events Rules

Deletion internal events rules for predicate P_i were defined in (18) as:

$$\delta P_i(\mathbf{k},\mathbf{x}) \leftarrow P°_i(\mathbf{k},\mathbf{x}) \wedge \neg \exists y P_i(\mathbf{k},\mathbf{y}) \quad i = 1 \ldots m$$

replacing $P°_i(\mathbf{k},\mathbf{x})$ by its equivalent definition given in (11) we get:

$$(24)\quad \delta P_i(\mathbf{k},\mathbf{x}) \leftarrow P°_{i,j}(\mathbf{k},\mathbf{x}) \wedge \neg \exists y P_i(\mathbf{k},\mathbf{y}) \quad i = 1 \ldots m,\ j = 1 \ldots \alpha$$

We can remove from (24) the rules corresponding to j = 1, which have the form given in (12), since $P°_{i,1}(\mathbf{k},\mathbf{x}) \rightarrow P_i(\mathbf{k},\mathbf{x})$. We can then reduce the set (24) to:

$$(25)\quad \delta P_i(\mathbf{k},\mathbf{x}) \leftarrow P°_{i,j}(\mathbf{k},\mathbf{x}) \wedge \neg \exists y P_i(\mathbf{k},\mathbf{y}) \quad i = 1 \ldots m,\ j = 2 \ldots \alpha$$

which can be rewritten as:

$$(26)\quad \delta P_i(\mathbf{k},\mathbf{x}) \leftarrow B(P°_{i,j}) \wedge \neg \exists y B(P_i)\sigma \quad i = 1 \ldots m,\ j = 2 \ldots \alpha$$

where σ is the substitution $\{\mathbf{y}/\mathbf{x}, \mathbf{z}'/\mathbf{z}\}$, $\mathbf{z}$ is the set of variables in $B(P_i)$ except $\mathbf{k}$ and $\mathbf{x}$; and $\mathbf{z}'$ are new variables.

There are several simplifications that can be applied to rules (26) above. All of them are based on the analysis of the relationship between a literal $L_h(\mathbf{k_h},\mathbf{u_h})$ in $B(P°_{i,j})$ and the corresponding literal in $B(P_i)\sigma$. The result of this simplification can be either a reduced form of the expression $\neg\exists y B(P_i)\sigma$ or a removal of the rule.

Deletion of a positive literal
If $\delta Q_h(\mathbf{k_h},\mathbf{u_h})$ is a literal in $B(P°_{i,j})$ and $Q_h(\mathbf{k_h},\mathbf{v_h})$ is the corresponding literal in $B(P_i)\sigma$, then the expression $\neg \exists y B(P_i)\sigma$ can be removed from (26). Notice that $\mathbf{u_h},\mathbf{v_h}$ can be null.

Proof: By (2), $\delta Q_h(\mathbf{k_h},\mathbf{u_h}) \rightarrow \neg \exists y Q_h(\mathbf{k_h},\mathbf{y})$ and thus $Q_h(\mathbf{k_h},\mathbf{v_h})$ is false. ⊕

Insertion of a negative literal
If $\iota Q_h(\mathbf{k_h})$ is a literal in $B(P°_{i,j})$ and $\neg Q_h(\mathbf{k_h})$ is the corresponding literal in $B(P_i)\sigma$, then the expression $\neg\exists y B(P_i)\sigma$ can be removed from (26).

If $\iota Q_h(\mathbf{k_h},\mathbf{u_h})$ is a literal in $B(P°_{i,j})$ and $L = \neg Q_h(\mathbf{k_h},\mathbf{v_h})$ is the corresponding literal in $B(P_i)\sigma$, then the expression $\neg \exists y B(P_i)\sigma$ can be simplified to $\neg\exists y (B(P_i)\sigma\backslash L \wedge \mathbf{u_h} \neq \mathbf{v_h})$.

Proof: By (1), $\iota Q_h(\mathbf{k_h}) \rightarrow Q_h(\mathbf{k_h})$ and thus $\neg Q_h(\mathbf{k_h})$ is false. Also by (1), $\iota Q_h(\mathbf{k_h},\mathbf{u_h}) \rightarrow Q_h(\mathbf{k_h},\mathbf{u_h})$. Given that $\mathbf{k_h}$ is a key for Q_h, then $\neg Q_h(\mathbf{k_h},\mathbf{v_h})$ is true only when $\mathbf{u_h} \neq \mathbf{v_h}$. ⊕

Modification of a literal
If $\mu Q_h(\mathbf{k_h},\mathbf{u_h},\mathbf{u'_h})$ (resp., $\mu Q_h(\mathbf{k_h},\mathbf{u'_h},\mathbf{u_h})$) is a literal in $B(P°_{i,j})$ and $L = Q_h(\mathbf{k_h},\mathbf{v_h})$ (resp., $L = \neg Q_h(\mathbf{k_h},\mathbf{v_h})$) is the corresponding literal in $B(P_i)\sigma$, then the expression $\neg\exists y B(P_i)\sigma$ can be simplified to $\neg\exists y (B(P_i)\sigma\backslash L \wedge \mathbf{u'_h}=\mathbf{v_h})$ (resp., $\neg\exists y (B(P_i)\sigma\backslash L \wedge \mathbf{u_h}\neq\mathbf{v_h})$).

Proof: We give the proof for the modification of a positive literal. By (3), $\mu Q_h(\mathbf{k_h},\mathbf{u_h},\mathbf{u'_h}) \rightarrow Q_h(\mathbf{k_h},\mathbf{u'_h})$. Given that $\mathbf{k_h}$ is a key for Q_h, then $Q_h(\mathbf{k_h},\mathbf{v_h})$ is true only when $\mathbf{u'_h} = \mathbf{v_h}$. We prove similarly the modification of a negative literal. ⊕

Unchanged positive literal
If $Q_h(\mathbf{k_h},\mathbf{u_h})$ is a literal in $B(P°_{i,j})$ and $L = Q_h(\mathbf{k_h},\mathbf{v_h})$ is the corresponding literal in $B(P_i)\sigma$, then the expression $\neg\exists y B(P_i)\sigma$ can be simplified to $\neg \exists y (B(P_i)\sigma\backslash L \wedge \mathbf{u_h} = \mathbf{v_h})$. Notice that $\mathbf{u_h},\mathbf{v_h}$ can be null.

Proof: Similar to the previous one. ⊕

Unchanged negative literal
If $\neg Q_h(\mathbf{k_h})$ is a literal in $B(P°_{i,j})$ and $L = \neg Q_h(\mathbf{k_h})$ is the corresponding literal in $B(P_i)\sigma$, then the expression $\neg \exists y B(P_i)\sigma$ can be simplified to $\neg \exists y (B(P_i)\sigma\backslash L)$.

Proof: Straightforward.⊕

Auxiliary simplifications
Some of the simplifications insert comparison literals in the expression $\neg\exists y B(P_i)\sigma$. A simple analysis of such literals produce further simplifications. For example:

a) If a literal has the form $\mathbf{u} = \mathbf{u}$, then it can be removed.

b) If a literal has the form $\mathbf{u} \neq \mathbf{u}$, then $\neg \exists y B(P_i)\sigma$ becomes true.

c) If a literal L has the form $\mathbf{u_h} = \mathbf{v_h}$, where $\mathbf{u_h}$ (resp., $\mathbf{v_h}$) is free (resp., bound) in $\neg \exists y B(P_i)\sigma$, then we remove the literal and replace the expression by $(\neg \exists y B(P_i)\sigma\backslash L)\{\mathbf{u_h}/\mathbf{v_h}\}$.

Example 4

Applying the above simplifications to the rules corresponding to δYoung(p,a) and δStudent(p,a) given in Example 3, we get:

(E.26) δYoung(p,a) $\leftarrow$ δPerson(p,a) $\wedge$ a<20
(E.27) δYoung(p,a) $\leftarrow$ μPerson(p,a,a')$\wedge$a<20 $\wedge$ $\neg$a'<20
(E.28) δStudent(p,a) $\leftarrow$ Young(p,a) $\wedge$ $\neg$ ιYoung(p,a) $\wedge$ $\neg$ μYoung(p,a',a) $\wedge$ ιWorks(p)
(E.29) δStudent(p,a) $\leftarrow$ δYoung(p,a) $\wedge$ $\neg$Works(p) $\wedge$ $\neg$ δWorks(p)
(E.30) δStudent(p,a) $\leftarrow$ δYoung(p,a) $\wedge$ ιWorks(p)
(E.31) δStudent(p,a) $\leftarrow$ μYoung(p,a,a') $\wedge$ ιWorks(p)

4.2 Simplification of Modification Internal Events Rules

Modification internal events rules for predicate P_i were defined in section 3.5 as:

(21) $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow P^\circ_i(\mathbf{k},\mathbf{x}) \wedge P_h(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'}$ i=1...m, h=1...m except i
(22) $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow \mu P_i(\mathbf{k},\mathbf{x},\mathbf{x'})$ i=1...m
(23) $\mu P_i(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow P^\circ_i(\mathbf{k},\mathbf{x}) \wedge P_i(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'}$ i =1...m

We first show that (21) can simplified to:

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

We give the proof in the Appendix. The main idea is that, due to the key integrity constraint, $P_i(\mathbf{k},\mathbf{x})$ must not hold and that the only way to get $P^\circ_i(\mathbf{k},\mathbf{x})$ true, being $P_i(\mathbf{k},\mathbf{x})$ false, is that $\delta P_i(\mathbf{k},\mathbf{x})$ holds. Also due to the key integrity constraint, $P^\circ_h(\mathbf{k},\mathbf{x'})$ must not hold and the only way to get $P^\circ_h(\mathbf{k},\mathbf{x'})$ false, being $P_h(\mathbf{k},\mathbf{x'})$ true, is that $\iota P_h(\mathbf{k},\mathbf{x'})$ holds.

Now, we show how to simplify rules (23). Replacing in them $P^\circ_i(\mathbf{k},\mathbf{x})$ by its equivalent definition given in (11) we get:

(28) $\mu P_i(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow P^\circ_{i,j}(\mathbf{k},\mathbf{x}) \wedge P_i(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'}$ i = 1 ... m, j = 1 ... α

We can remove from (28) the rules corresponding to j=1, which have the form given in (12), since $P^\circ_{i,1}(\mathbf{k},\mathbf{x}) \rightarrow P_i(\mathbf{k},\mathbf{x})$ and, due the key integrity constraint, $P_i(\mathbf{k},\mathbf{x}) \rightarrow \neg\exists \mathbf{x'} P_i(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'}$. We can then reduce the set (28) to:

(29) $\mu P_i(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow P^\circ_{i,j}(\mathbf{k},\mathbf{x}) \wedge P_i(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'}$ i = 1 ... m, j = 2 ... α

which can be rewritten as:

(30) $\mu P_i(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow B(P^\circ_{i,j}) \wedge B(P_i)\sigma \wedge \mathbf{x} \neq \mathbf{x'}$ i = 1 ... m, j = 2 ... α

where σ is the substitution $\{\mathbf{x'}/\mathbf{x}, \mathbf{z'}/\mathbf{z}\}$, $\mathbf{z}$ is the set of variables in $B(P_i)$ except $\mathbf{k}$ and $\mathbf{x}$; and $\mathbf{z'}$ are new variables.

As in the deletion case, there are several simplifications that can be applied to rules (29) above. All of them are based on the analysis of the relationship between a literal $L_h(\mathbf{k}_h,\mathbf{u}_h)$ in $B(P^\circ_{i,j})$ and the corresponding literal in $B(P_i)\sigma$. The result of this simplification can be either a reduced form of the expression $B(P_i)\sigma$ or a removal of the rule.

Deletion of a positive literal
If $\delta Q_h(\mathbf{k}_h,\mathbf{u}_h)$ is a literal in $B(P^\circ_{i,j})$ and $Q_h(\mathbf{k}_h,\mathbf{v}_h)$ is the corresponding literal in $B(P_i)\sigma$, then this rule can be removed. Notice that $\mathbf{u}_h, \mathbf{v}_h$ can be null.

Proof: By (2), $\delta Q_h(\mathbf{k}_h,\mathbf{u}_h) \rightarrow \neg\exists y Q_h(\mathbf{k}_h,y)$ and thus $Q_h(\mathbf{k}_h,\mathbf{v}_h)$ is false. ⊕

Insertion of a negative literal
If $\iota Q_h(\mathbf{k}_h)$ is a literal in $B(P^\circ_{i,j})$ and $\neg Q_h(\mathbf{k}_h)$ is the corresponding literal in $B(P_i)\sigma$, then this rule can be removed.

If $\iota Q_h(\mathbf{k}_h,\mathbf{u}_h)$ is a literal in $B(P^\circ_{i,j})$ and $L = \neg Q_h(\mathbf{k}_h,\mathbf{v}_h)$ is the corresponding literal in $B(P_i)\sigma$, then the expression $B(P_i)\sigma$ can be simplified to $(B(P_i)\sigma \backslash L \wedge \mathbf{u}_h \neq \mathbf{v}_h)$.

Proof: By (1), $\iota Q_h(\mathbf{k}_h) \rightarrow Q_h(\mathbf{k}_h)$ and thus $\neg Q_h(\mathbf{k}_h)$ is false. Also by (1), $\iota Q_h(\mathbf{k}_h,\mathbf{u}_h) \rightarrow Q_h(\mathbf{k}_h,\mathbf{u}_h)$. Given that $\mathbf{k}_h$ is a key for Q_h, then $\neg Q_h(\mathbf{k}_h,\mathbf{v}_h)$ is true only when $\mathbf{u}_h \neq \mathbf{v}_h$. ⊕

Modification of a literal
If $\mu Q_h(\mathbf{k}_h,\mathbf{u}_h,\mathbf{u'}_h)$ (resp., $\mu Q_h(\mathbf{k}_h,\mathbf{u'}_h,\mathbf{u}_h)$) is a literal in $B(P^\circ_{i,j})$ and $L = Q_h(\mathbf{k}_h,\mathbf{v}_h)$ (resp., $L = \neg Q_h(\mathbf{k}_h,\mathbf{v}_h)$) is the corresponding literal in $B(P_i)\sigma$, then the expression $B(P_i)\sigma$ can be simplified to $(B(P_i)\sigma \backslash L \wedge \mathbf{u'}_h = \mathbf{v}_h)$ (resp., $(B(P_i)\sigma \backslash L \wedge \mathbf{u}_h \neq \mathbf{v}_h)$).

Proof: We give the proof for the modification of a positive literal. By (3), $\mu Q_h(\mathbf{k}_h,\mathbf{u}_h,\mathbf{u'}_h) \rightarrow Q_h(\mathbf{k}_h,\mathbf{u'}_h)$. Given that $\mathbf{k}_h$ is a key for Q_h, then $Q_h(\mathbf{k}_h,\mathbf{v}_h)$ is true only when $\mathbf{u'}_h = \mathbf{v}_h$. We prove similarly the modification of a negative literal. ⊕

Unchanged positive literal
If $Q_h(\mathbf{k}_h,\mathbf{u}_h)$ is a literal in $B(P^\circ_{i,j})$ and $L = Q_h(\mathbf{k}_h,\mathbf{v}_h)$ is the corresponding literal in $B(P_i)\sigma$, then the expression $B(P_i)\sigma$ can be simplified to $(B(P_i)\sigma \backslash L \wedge \mathbf{u}_h = \mathbf{v}_h)$. Notice that $\mathbf{u}_h, \mathbf{v}_h$ can be null.

Proof: Similar to the previous one. ⊕

Unchanged negative literal
If $\neg Q_h(\mathbf{k}_h)$ is a literal in $B(P^\circ_{i,j})$ and $L = \neg Q_h(\mathbf{k}_h)$ is the corresponding literal in $B(P_i)\sigma$, then the expression $B(P_i)\sigma$ can be simplified to $(B(P_i)\sigma \backslash L)$.

Proof: Straightforward. ⊕

Example 5

Applying the above simplifications to the rules corresponding to μYoung(p,a,a') and μStudent(p,a,a') given in Example 3, we get:

(E.32) μYoung(p,a,a') $\leftarrow$ μPerson(p,a,a') $\wedge$ a<20 $\wedge$ a'<20
(E.33) μStudent(p,a,a') $\leftarrow$ μ Young(p,a,a') $\wedge$ $\neg$Works(p) $\wedge$ $\neg$ δWorks(p)

4.3 Simplification of Insertion Internal Events Rules

Insertion internal events rules of predicate P_i were defined in (15) as:

$$\iota P_i(\mathbf{k},\mathbf{x}) \leftarrow P_i(\mathbf{k},\mathbf{x}) \wedge \neg \exists y P^\circ_i(\mathbf{k},\mathbf{y}) \quad i = 1 \dots m$$

replacing $P^\circ_i(\mathbf{k},\mathbf{y})$ by its equivalent definition given in (11) we get:

$$(31)\ \iota P_i(\mathbf{k},\mathbf{x}) \leftarrow P_i(\mathbf{k},\mathbf{x}) \wedge \neg \exists y P^\circ_{i,1}(\mathbf{k},\mathbf{y}) \wedge \dots \wedge \neg \exists y P^\circ_{i,\alpha}(\mathbf{k},\mathbf{y}) \quad i = 1 \dots m$$

which can be rewritten as:

$$(32)\ \iota P_i(\mathbf{k},\mathbf{x}) \leftarrow B(P_i) \wedge \neg \exists y (B(P^\circ_{i,1})\sigma) \wedge \dots \wedge \neg \exists y (B(P^\circ_{i,\alpha})\sigma) \quad i = 1 \dots m$$

where σ is the substitution {y/x, z'/z}, z is the set of variables in $B(P^\circ_{i,\alpha})$ except **k** and x; and **z'** are new variables.

Assume $B(P_i)$ has n ordinary literals, and let $L_h = [Q_h(\mathbf{k_h},\mathbf{u_h}) \mid \neg Q_h(\mathbf{k_h},\mathbf{u_h})]$ be one of them. It is not difficult to see that in (32) an ιP_i fact can only be induced by an insertion, deletion or modification event of some L_h. In fact, we prove in [Urp91a] that each rule (32) is equivalent to the set of rules:

$$(33)\ \iota P_i(\mathbf{k},\mathbf{x}) \leftarrow B(P_i)\backslash L_h \wedge [\iota Q_h(\mathbf{k_h},\mathbf{u_h}) \mid \delta Q_h(\mathbf{k_h},\mathbf{u_h})] \wedge \neg \exists y (B(P^\circ_{i,1})\sigma) \wedge \dots \wedge \neg \exists y (B(P^\circ_{i,\alpha})\sigma) \quad i = 1 \dots m,\ h = 1 \dots n$$

$$(34)\ \iota P_i(\mathbf{k},\mathbf{x}) \leftarrow B(P_i)\backslash L_h \wedge [\mu Q_h(\mathbf{k_h},\mathbf{u'_h},\mathbf{u_h}) \mid \mu Q_h(\mathbf{k_h},\mathbf{u_h},\mathbf{u'_h})] \wedge \neg \exists y (B(P^\circ_{i,1})\sigma) \wedge \dots \wedge \neg \exists y (B(P^\circ_{i,\alpha})\sigma) \quad i = 1 \dots m,\ h = 1 \dots n$$

As in the previous case, there are several simplifications that can be applied to rules (33) and (34) above. Again, these simplifications are based on the analysis of the relationship between a literal in $B(P_i)\backslash L_h$ or ιQ_h, δQ_h, μQ_h literals and the corresponding literal in each $B(P^\circ_{i,j})\sigma$, where j = 1...α. The result of this simplification can be either a reduced form of the expressions $\neg\exists y B(P^\circ_{i,j})\sigma$ or the removal of whole rule.

Insertion of a positive literal
If $\iota Q_h(\mathbf{k_h},\mathbf{u_h})$ is a literal in (33) and the key of the corresponding literals of some $B(P^\circ_{i,j})\sigma$ is $\mathbf{k_h}$, then the expression $\neg \exists y (B(P^\circ_{i,j})\sigma)$ can be removed from (33). Notice that $\mathbf{u_h}$ can be null.

Proof: The corresponding literals of $\iota Q_h(\mathbf{k_h},\mathbf{u_h})$ in $B(P^\circ_{i,j})\sigma$ have one of the form $Q_h(\mathbf{k_h},\mathbf{v_h}) \wedge \neg \iota Q_h(\mathbf{k_h},\mathbf{v_h}) \wedge \neg \mu Q_h(\mathbf{k_h},\mathbf{v'_h},\mathbf{v_h})$ or $\delta Q_h(\mathbf{k_h},\mathbf{v_h})$ or $\mu Q_h(\mathbf{k_h},\mathbf{v_h},\mathbf{v'_h})$. Assume the first form. Then, by (1), $\iota Q_h(\mathbf{k_h},\mathbf{u_h}) \rightarrow Q_h(\mathbf{k_h},\mathbf{u_h})$. Given that $\mathbf{k_h}$ is a key for Q_h, then $Q_h(\mathbf{k_h},\mathbf{v_h})$ is true only when $\mathbf{u_h}=\mathbf{v_h}$. But in this case, $\neg\iota Q_h(\mathbf{k_h},\mathbf{v_h})$ is false.

For the orther two forms, we have $\iota Q_h(\mathbf{k_h},\mathbf{u_h}) \rightarrow \neg\exists y \delta Q_h(\mathbf{k_h},\mathbf{y})$ and $\iota Q_h(\mathbf{k_h},\mathbf{u_h}) \rightarrow \neg\exists \mathbf{y},\mathbf{y'} \mu Q_h(\mathbf{k_h},\mathbf{y},\mathbf{y'})$. ⊕

Deletion of a negative literal
If $\delta Q_h(\mathbf{k_h})$ is a literal in (33) and $\neg Q_h(\mathbf{k_h}) \wedge \neg \delta Q_h(\mathbf{k_h})$ or $\iota Q_h(\mathbf{k_h})$ is the corresponding literal in some $B(P^\circ_{i,j})\sigma$, then the expression $\neg \exists y (B(P^\circ_{i,j})\sigma)$ can be removed from (33).

If $\delta Q_h(\mathbf{k_h},\mathbf{u_h})$ is a literal in (33) and $\iota Q_h(\mathbf{k_h},\mathbf{v_h})$ or $\mu Q_h(\mathbf{k_h},\mathbf{v'_h},\mathbf{v_h})$ is the corresponding literal in some $B(P^\circ_{i,j})\sigma$, then the expression $\neg \exists y (B(P^\circ_{i,j})\sigma)$ can be removed from (33).

If $\delta Q_h(\mathbf{k_h},\mathbf{u_h})$ is a literal in (33) and $L = \neg Q_h(\mathbf{k_h},\mathbf{v_h}) \wedge \neg \delta Q_h(\mathbf{k_h},\mathbf{v_h}) \wedge \neg \mu Q_h(\mathbf{k_h},\mathbf{v_h},\mathbf{v'_h})$ is the corresponding literal in some $B(P^\circ_{i,j})\sigma$, then the expression $\neg\exists y (B(P^\circ_{i,j})\sigma)$ can be simplified to $\neg \exists y (B(P^\circ_{i,j})\sigma\backslash L \wedge \mathbf{u_h} \neq \mathbf{v_h})$.

Proof: By (2), $\delta Q_h(\mathbf{k_h},\mathbf{u_h}) \rightarrow \neg \exists y Q_h(\mathbf{k_h},\mathbf{y})$. Given that $\mathbf{k_h}$ is a key for Q_h, then $\neg \delta Q_h(\mathbf{k_h},\mathbf{v_h})$ is true only when $\mathbf{u_h} \neq \mathbf{v_h}$. Finally, $\delta Q_h(\mathbf{k_h},\mathbf{u_h}) \rightarrow \neg \exists y \iota Q_h(\mathbf{k_h},\mathbf{y})$ and $\delta Q_h(\mathbf{k_h},\mathbf{u_h}) \rightarrow \neg \exists \mathbf{y},\mathbf{y'} \mu Q_h(\mathbf{k_h},\mathbf{y},\mathbf{y'})$. ⊕

Modification of a positive literal
If $\mu Q_h(\mathbf{k_h},\mathbf{u'_h},\mathbf{u_h})$ is a literal in (34) and $Q_h(\mathbf{k_h},\mathbf{v_h}) \wedge \neg\iota Q_h(\mathbf{k_h},\mathbf{v_h}) \wedge \neg \mu Q_h(\mathbf{k_h},\mathbf{v'_h},\mathbf{v_h})$ or $\delta Q_h(\mathbf{k_h},\mathbf{v_h})$ is the corresponding literal in some $B(P^\circ_{i,j})\sigma$, then the expression $\neg \exists y (B(P^\circ_{i,j})\sigma)$ can be removed from (34).

If $\mu Q_h(\mathbf{k_h},\mathbf{u'_h},\mathbf{u_h})$ is a literal in (34) and $L = \mu Q_h(\mathbf{k_h},\mathbf{v_h},\mathbf{v'_h})$ is the corresponding literal in some $B(P^\circ_{i,j})\sigma$, then the expression $\neg\exists y (B(P^\circ_{i,j})\sigma)$ can be simplified to $\neg \exists y (B(P^\circ_{i,j})\sigma\backslash L \wedge \mathbf{u'_h}=\mathbf{v_h} \wedge \mathbf{u_h}=\mathbf{v'_h})$.

Proof: By (3), $\mu Q_h(\mathbf{k_h},\mathbf{u'_h},\mathbf{u_h}) \rightarrow Q_h(\mathbf{k_h},\mathbf{u_h})$. Given that $\mathbf{k_h}$ is a key for Q_h, then $Q_h(\mathbf{k_h},\mathbf{v_h})$ is true only when $\mathbf{u_h}=\mathbf{v_h}$. But in this case, $\neg \mu Q_h(\mathbf{k_h},\mathbf{v'_h},\mathbf{v_h})$ is false. Furthermore, $\mu Q_h(\mathbf{k_h},\mathbf{u'_h},\mathbf{u_h}) \rightarrow \neg \exists y \delta Q_h(\mathbf{k_h},\mathbf{y})$.

On the other hand, given that $\mathbf{k_h}$ is a key for Q_h, then $\mu Q_h(\mathbf{k_h},\mathbf{v_h},\mathbf{v'_h})$ is true only when $\mathbf{u'_h}=\mathbf{v_h} \wedge \mathbf{u_h}=\mathbf{v'_h}$. ⊕

Modification of a negative literal
If $\mu Q_h(\mathbf{k_h},\mathbf{u_h},\mathbf{u'_h})$ is a literal in (34) and $\iota Q_h(\mathbf{k_h},\mathbf{v_h})$ is the corresponding literal in some $B(P^\circ_{i,j})\sigma$, then the expression $\neg \exists y (B(P^\circ_{i,j})\sigma)$ can be removed from (34).

If $\mu Q_h(k_h,u_h,u'_h)$ is a literal in (34) and $L = \neg Q_h(k_h,v_h) \wedge \neg\, \delta Q_h(k_h,v_h) \wedge \neg\, \mu Q_h(k_h,v_h,v'_h)$ (resp., $L = \mu Q_h(k_h,v'_h,v_h)$) is the corresponding literal in some $B(P^\circ_{i,j})\sigma$, then the expression $\neg\, \exists y(B(P^\circ_{i,j})\sigma)$ can be simplified to $\neg\, \exists y(B(P^\circ_{i,j})\sigma \backslash L \wedge u'_h \neq v_h \wedge u_h \neq v_h)$ (resp., $\neg\, \exists y(B(P^\circ_{i,j})\sigma \backslash L \wedge u_h = v'_h \wedge u'_h = v_h)$.

Proof: Similar to previous one. ⊕

Unchanged positive literal

If $Q_h(k_h,u_h)$ is a literal in (33) (or (34)) and $\delta Q_h(k_h,v_h)$ is the corresponding literal in some $B(P^\circ_{i,j})\sigma$, then the expression $\neg\, \exists y(B(P^\circ_{i,j})\sigma)$ can be removed from (33) (from (34)).

If $Q_h(k_h,u_h)$ is a literal in (33) (or (34)) and $L = Q_h(k_h,v_h) \wedge \neg\, \iota Q_h(k_h,v_h) \wedge \neg\, \mu Q_h(k_h,v'_h,v_h)$ (resp., $L = \mu Q_h(k_h,v_h,v'_h)$) is the corresponding literal in some $B(P^\circ_{i,j})\sigma$, then the expression $\neg\, \exists y(B(P^\circ_{i,j})\sigma)$ can be simplified to $\neg\, \exists y(B(P^\circ_{i,j})\sigma \backslash L \wedge u_h = v_h)$ (resp., $\neg \exists y(B(P^\circ_{i,j})\sigma \wedge u_h = v'_h)$. Notice that u_h, v_h can be null.

Proof: We prove the first case. By the sake of a contradiction, if $\delta Q_h(k_h,v_h)$ was true, then, by (2), $\neg \exists y\, Q_h(k_h,y)$. But, since $Q_h(k_h,u_h)$ is a literal in (33) (or (34)), we have a contradiction. ⊕

Unchanged negative literal

If $\neg Q_h(k_h)$ is a literal in (33) (or (34)) and $\iota Q_h(k_h)$ is the corresponding literal in some $B(P^\circ_{i,j})\sigma$, then the expression $\neg\, \exists y(B(P^\circ_{i,j})\sigma)$ can be removed from (33) (from (34)).

If $\neg Q_h(k_h)$ is a literal in (33) (or (34)) and $L = \neg Q_h(k_h) \wedge \neg\, \delta Q_h(k_h)$ is the corresponding literal in some $B(P^\circ_{i,j})\sigma$, then the expression $\neg \exists y(B(P^\circ_{i,j})\sigma)$ can be simplified to $\neg\, \exists y(B(P^\circ_{i,j})\sigma \backslash L)$.

Proof: Similar to the previous one. ⊕

Example 6

Applying the above simplifications to the rules corresponding to ιYoung(p,a) and ιStudent(p,a) given in Example 3, we obtain:

(E.34) ιYoung(p,a) ← ιPerson(p,a) ∧ a<20
(E.35) ιYoung(p,a) ← μPerson(p,a',a) ∧ a<20 ∧ ¬a'<20
(E.36) ιStudent(p,a) ← ιYoung(p,a) ∧ ¬Works(p)
(E.37) ιStudent(p,a) ← Young(p,a) ∧ δWorks(p)

5 Change Computation

We present in this Section a method for the definition and computation of changes in deductive databases. Efficient change computation is essential in a wide range of applications in deductive databases, including integrity constraints checking, view materialization and condition monitoring. The common pattern in all these applications consists of:

a) The definition of one or more changes to be monitored.
b) The computation of the changes induced by a database update.
c) The execution of some action when some of the defined changes has been induced.

We will show first that our internal event concept can be used to define the changes to be monitored. Assume that Ic is an inconsistency predicate, such as, for example:

Ic(e,c) ← Works(e,c) ∧ ¬Company(c)

meaning that employees must work in companies. Then, insertion internal events ιIc will represent violations of the corresponding integrity constraint. If an update to base predicates induces some ιIc fact then the update must be rejected. Deletion and modification internal events are not defined for inconsistency predicates, since we assume that the database is consistent before the update and, therefore, predicate Ic is false.

Now, assume that Em is a derived predicate corresponding to a materialized view, such as, for example:

Em(emp,manager)← Ed(emp,dept) ∧ Dm(dept,manager)

In this case, internal events ιEm, δEm and μEm correspond to the insertion, deletion or modification of facts in the extension of Em. Thus, for instance, if the update induces an ιEm(E,M) fact, then Em(E,M) will be inserted into the extension of the materialized view Em.

General conditions can also be represented as insertion, deletion or modification internal events of a derived predicate. Assume, for example, that we want to monitor changes of employees earning more than 1000. We may define a predicate C:

C(emp,salary) ← Sal(emp,salary) ∧ salary > 1000

and then ιC(e,s) can be used to define a change meaning that e is an employee earning more than 1000 after the update, but not before; δC(e,s) for a change meaning that employee e ceases to earn more than 1000; and μC(e,s,s') for a change meaning that the salary of employee e has been modified from s to s', both greater than 1000. Appropriate actions could be associated to each, or some, of the above changes.

Thus, we see that the single concept of internal event may serve for defining relevant changes in a variety of applications in deductive databases.

5.1 Our Method

We now describe our method for change computation. The method can be entirely based on the use of standard SLDNF resolution. Let D be a deductive database and let us to denote by A(D) the augmented database consisting of the database D and its transition and events rules. Let T be a transaction consisting of a set of external events. If T induces a change in a derived predicate $P(\mathbf{k},\mathbf{x})$, then some of the $\iota P(\mathbf{k},\mathbf{x})$, $\delta P(\mathbf{k},\mathbf{x})$ or $\mu P(\mathbf{k},\mathbf{x},\mathbf{x}')$ facts will be true in the transition. Using the SLDNF proof procedure, T induces an internal event ιP (or δP or μP) if the goal $\leftarrow \iota P(\mathbf{k},\mathbf{x})$ succeeds from input set $A(D) \cup T$. If every branch of the SLDNF search space for $A(D) \cup T \cup \{\leftarrow \iota P(\mathbf{k},\mathbf{x})\}$ is a failure branch, then T does not induce an ιP fact.

Example 7

Assume the database given in Example 1, and let the transaction be T = {μPerson(Ann,15,16)}, that is, we change Ann's age from 15 to 16. The following refutation shows that T induces μStudent(Ann,15,16):

$\leftarrow \mu Student(p,a,a')$

(E.33)

$\leftarrow \mu Young(p,a,a') \wedge \neg Works(p) \wedge \neg \delta Works(p)$

(E.32)

$\leftarrow \mu Person(p,a,a') \wedge a<20 \wedge a'<20 \wedge \neg Works(p) \wedge \neg \delta Works(p)$

(T, p/Ann, a/15, a'/16)

$\leftarrow 15<20 \wedge 16<20 \wedge \neg Works(Ann) \wedge \neg \delta Works(Ann)$

(15<20 ∧ 16<20)

$\leftarrow \neg Works(Ann) \wedge \neg \delta Works(Ann)$

← Works(Ann) fails

$\leftarrow \neg \delta Works(Ann)$

← δWorks(Ann) fails

[ ]

It can be shown that all derivations with root goals {← ιStudent(p,a)} and {← δStudent(p,a)} fail finitely and, thus, T does not induce any ιStudent or δStudent fact.

A number of optimization techniques can be naturally incorporated into our method. The most important is the partial evaluation [LlS91] of the transition rules, internal events rules and a given transaction with respect to the relevant internal events. Partial evaluation produces, at compilation time, a set of equivalent rules which which can be evaluated more efficiently at execution time.

Example 8

In our example, partial evaluation of the transition rules and internal events rules and transaction T = {μPerson(P,A,A')}, where P, A and A' are parameters, with respect to literals ←ιStudent(p,a), ←δStudent(p,a) and ←μStudent(p,a,a'), produces the program:

(E.38) $\mu Student(P,A,A') \leftarrow A<20 \wedge A'<20 \wedge \neg Works(P)$
(E.39) $\iota Student(P,A) \leftarrow A'<20 \wedge \neg A<20 \wedge \neg Works(P)$
(E.40) $\delta Student(P,A) \leftarrow A<20 \wedge \neg A'<20 \wedge \neg Works(P)$

which can be evaluated efficiently at execution time, with a single access to the database (Works(P)).

We can also take into account some details of a given application of change computation. Thus, in view materialization we have available the old state of the view, or in integrity constraints checking we know that the old state is consistent. In such cases, we can easily adapt our rules to take advantage of this knowledge.

6 Comparison with other Methods

In this section we compare our method for change computation in deductive databases with some of the methods mentioned in the introduction. We discuss the method proposed by Rosenthal, Chakravarthy, Blaustein and Blakeley [RCB+89] for condition monitoring, and the method proposed by Ceri and Widom [CeW91] for incremental view maintenance. See [Oli91] for a comparison of a variant of our method with integrity checking methods.

6.1 Rosenthal et al.'s Method

One of the problems addressed in the HiPAC project [CBB+89] is condition monitoring in active database systems [RCB+89]. Rosenthal, Chakravarthy, Blaustein and Blakeley 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. The condition part of a situation is defined using a relational expression.

They consider not only insertion and deletions changes of a monitored condition, but also modification changes. Each tuple of a relation has a special attribute that provides a unique immutable identifier, so that a tuple is modified if some of its attributes changes. The method derives an algebraic expression for computing induced changes.

In general, when the expression that defines the view is a select, project, join or an arbitrary expression with a unary operator as a root of the expression, we obtain similar results. However, the main advantage of our method is that it allows more expressiveness in the definition of derived predicates that can be handled in incremental form: we can apply our method to more general derived predicates. As an example, we can have derived predicates defined with the negation operator and with more than one rule (with the binary union operator as a root of the expression).

Furthermore, our rules incorporate the knowledge of keys of predicates. This allow us to obtain a more simplified set of rules that fit to each particular situation. As an example, consider the derived predicate Young($\underline{p}$,a), defined in the example 1 as: Young($\underline{p}$,a) $\leftarrow$ Person($\underline{p}$,a) $\wedge$ a<20. We have shown that rules E.26, E.27, E.32, E.34 and E.36 compute changes to that predicate in incremental form. Assuming now that our knowledge of keys changes: Young($\underline{p,a}$) $\leftarrow$ Person($\underline{p,a}$) $\wedge$ a<20, applying our method we obtain the following simple events rules:

$$\iota\text{Young}(\underline{p,a}) \leftarrow \iota\text{Person}(\underline{p,a}) \wedge a<20$$
$$\delta\text{Young}(\underline{p,a}) \leftarrow \delta\text{Person}(\underline{p,a}) \wedge a<20$$

For a more detailed comparison see [Urp91b].

6.2 Ceri and Widom's Method

This method derives automatically production rules for incremental maintenance of materialized views. The rules are executable using the rule language of the Starburst database system [WCL91]. Views are specified using a standard query language, and arbitrary database updates (insertions, deletions and/or modifications) are considered. The method defines insertion and deletion changes of a materialized view. These changes are computed once base relations have been updated. Using the definition of a view and information about keys of the view's base tables, the method determines whether efficient view maintenance production rules [WiF90, WCL91] for updates on each base table can be generated.

If a base table reference in a view definition is *safe* [CeW91], incremental view maintenance rules can be generated. However, if a base table reference is *unsafe*, some of the updates on this table cannot be handled in incremental form and a rule that rematerializes the view in such cases is defined.

Once the rules have been generated, they must be ordered using a *precede* clause so that all rules performing deletions precede all rules performing insertions. The rematerialization rule of a view (if exists) has precedence over all rules of that view.

In general, using our method, we obtain similar results when a base table reference is *safe*. But there are two main differences. The first is that we can induce not only insertion and deletion changes of a materialized view, but also modification changes. To see the importance of this extension consider the derived predicate Young($\underline{p}$,a) given in the Example 1 and assume that Young is a materialized view. Using our method, we get the following simplified events rules that handle modifications events of base predicate Person:

(E.27) $\delta\text{Young}(\underline{p},a) \leftarrow \mu\text{Person}(\underline{p},a,a') \wedge a<20 \wedge \neg a'<20$
(E.32) $\mu\text{Young}(\underline{p},a,a') \leftarrow \mu\text{Person}(\underline{p},a,a') \wedge a<20 \wedge a'<20$
(E.35) $\iota\text{Young}(\underline{p},a) \leftarrow \mu\text{Person}(\underline{p},a',a) \wedge a<20 \wedge \neg a'<20$

The first rule can induce deletion changes of predicate young; the second one, modification changes and the third one, insertion changes. Instead, rules generated using Ceri and Widom's method handle case E.32 in a more inefficient way: as deletions of tuples Young($\underline{p}$,a) followed by insertions of tuples Young($\underline{p}$,a').

A second difference is that rules generated using Ceri and Widom's method do unnecessary work when the base table attributes that are irrelevant to the view definition are modified. As an example, consider the derived predicate Employee($\underline{e}$), defined as Employee($\underline{e}$) $\leftarrow$ Works($\underline{e}$,c). Applying our method we get the simplified events rules:

$$\iota\text{Employee}(\underline{e}) \leftarrow \iota\text{Works}(\underline{e},c)$$
$$\delta\text{Employee}(\underline{e}) \leftarrow \delta\text{Works}(\underline{e},c)$$

Notice that, since a modification of the company in which an employee works cannot induce insertion or deletion changes on Employee, our rules do not take those modifications into account. However, rules generated using Ceri and Widom's method do unnecessary work since such modifications will be handled as a deletion of tuple Employee($\underline{e}$) followed by an insertion of the same tuple.

As we mentioned previously, using Ceri and Widom's method, when a base table reference in a view definition is *unsafe*, some of the updates on that table cannot be supported efficiently. In this case, the main advantage of our method relies on the fact that we can handle those updates in incremental form. As an example, consider the derived predicate Tenant($\underline{p}$,h), defined as Tenant($\underline{p}$,h) $\leftarrow$ Lives($\underline{p}$,h) $\wedge$ Owns($\underline{c}$,h). Since Owns($\underline{c}$,h) is *unsafe*, if a tuple is modified or deleted from this table, the view Tenant($\underline{p}$,h) will be rematerialized. Instead, applying our method we get the following set of simplified rules that handle deletions from Owns($\underline{c}$,h) in incremental form:

$$\begin{aligned}
\delta\text{Tenant}(\underline{p},h) &\leftarrow \text{Lives}(\underline{p},h) \wedge \neg\,\iota\text{Lives}(\underline{p},h)\\
&\quad \wedge \neg\,\mu\text{Lives}(\underline{p},h',h) \wedge \delta\text{Owns}(\underline{c},h)\\
&\quad \wedge \neg\text{Owns}(\underline{c'},h)\\
\delta\text{Tenant}(\underline{p},h) &\leftarrow \delta\text{Lives}(\underline{p},h) \wedge \delta\text{Owns}(\underline{c},h)\\
\delta\text{Tenant}(\underline{p},h) &\leftarrow \mu\text{Lives}(\underline{p},h,h') \wedge \delta\text{Owns}(\underline{c},h)\\
&\quad \wedge \neg\text{Owns}(\underline{c'},h')\\
\mu\text{Tenant}(\underline{p},h,h') &\leftarrow \mu\text{Lives}(\underline{p},h,h') \wedge \delta\text{Owns}(\underline{c},h)\\
&\quad \wedge \text{Owns}(\underline{c'},h')
\end{aligned}$$

Similar rules are obtained for modifications of Owns($\underline{c}$,h).

Finally, we point out that our rules are generated using a simple procedure, while Ceri and Widom's rules are generated using a complex procedure, that needs to take into account the potentially complex syntactic structure of the view definition.

7 Conclusions

In this paper, we have presented a formal method to derive a set of transition and internal events rules for a deductive database. Given an update, the transition rules relate the old state to the new state and the events induced by the update. The internal events rules define explicitly the changes (insertions, deletions and modifications) induced by the update on the derived predicates.

We have then presented a method that use the above rules for computing the changes induced by an update in a deductive database. The method deals with allowed, stratified databases. Updates considered are sets of insertions, deletions and/or modifications of base facts. Our method is based on the use of the standard SLDNF procedure and, in this way, it can be implemented directly in Prolog. However, other proof procedures could be used as well. Some optimization techniques, including partial evaluation, can be easily incorporated into our method.

We have also compared our method with some other well-known methods, and we have shown how we improve their efficiency.

We plan to further simplify our transition and internal events rules by taking into account the complete set of integrity constraints of the database, including alternate keys.

Acknowledgements

We would like to thank 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.

Appendix

Proof of (27): We prove this simplification in two steps. Firstly, we prove that, due to the key integrity constraint, $P_i(\mathbf{k},\mathbf{x})$ must not hold and that the only way to get $P^\circ_i(\mathbf{k},\mathbf{x})$ true, being $P_i(\mathbf{k},\mathbf{x})$ false, is that $\delta P_i(\mathbf{k},\mathbf{x})$ holds. Secondly, we prove that, due to the key integrity constraint, $P^\circ_h(\mathbf{k},\mathbf{x'})$ must not hold and the only way to get $P^\circ_h(\mathbf{k},\mathbf{x'})$ false, being $P_h(\mathbf{k},\mathbf{x'})$ true, is that $\iota P_h(\mathbf{k},\mathbf{x'})$ holds.

First step.

Replacing $P^\circ_i(\mathbf{k},\mathbf{x})$ by its definition given in (4) we rewrite (21) into the set of rules:

(P.1) $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow P_i(\mathbf{k},\mathbf{x}) \wedge \neg\, \iota P_i(\mathbf{k},\mathbf{x}) \wedge \neg\, \mu P_i(\mathbf{k},\mathbf{x''},\mathbf{x}) \wedge P_h(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'}$

(P.2) $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow \delta P_i(\mathbf{k},\mathbf{x}) \wedge P_h(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'}$

(P.3) $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow \mu P_i(\mathbf{k},\mathbf{x},\mathbf{x''}) \wedge P_h(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'}$

where $i = 1 \ldots m$, $h = 1 \ldots m$ except i.

Rules (P.1) above can be removed since that, by the key integrity constraint, $P_i(\mathbf{k},\mathbf{x}) \rightarrow \neg\exists \mathbf{x'} P_h(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'}$.

Since, by (3), $\mu P_i(\mathbf{k},\mathbf{x},\mathbf{x''}) \rightarrow P_i(\mathbf{k},\mathbf{x''})$ and given that $\mathbf{k}$ is a key for P, then $\mathbf{x''} = \mathbf{x'}$ and thus rules (P.3) above can be removed since they are subsumed by (22).

Second step.

Let us consider rules (P.2). Since we assume that key integrity constraints hold before and after the update, we can rewrite rules (P.2) as:

(P.4) $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow \delta P_i(\mathbf{k},\mathbf{x}) \wedge P_h(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'} \wedge \neg(P^\circ_i(\mathbf{k},\mathbf{x}) \wedge P^\circ_h(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'})$

and given that, by (2), $\delta P_i(\mathbf{k},\mathbf{x}) \rightarrow P^\circ_i(\mathbf{k},\mathbf{x})$, we can rewrite (P.4) as:

(P.5) $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow \delta P_i(\mathbf{k},\mathbf{x}) \wedge P_h(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'} \wedge \neg P^\circ_h(\mathbf{k},\mathbf{x'})$.

Replacing $\neg P^\circ_h(\mathbf{k},\mathbf{x'})$ by its definition given in (5) we transform (P.5) into the set of rules:

(P.6) $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow \delta P_i(\mathbf{k},\mathbf{x}) \wedge P_h(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'} \wedge \neg P_h(\mathbf{k},\mathbf{x'}) \wedge \neg\, \delta P_h(\mathbf{k},\mathbf{x'}) \wedge \neg\, \mu P_h(\mathbf{k},\mathbf{x'},\mathbf{x''})$

(P.7) $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow \delta P_i(\mathbf{k},\mathbf{x}) \wedge P_h(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'} \wedge \iota P_h(\mathbf{k},\mathbf{x'})$

(P.8) $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \leftarrow \delta P_i(\mathbf{k},\mathbf{x}) \wedge P_h(\mathbf{k},\mathbf{x'}) \wedge \mathbf{x} \neq \mathbf{x'} \wedge \mu P_h(\mathbf{k},\mathbf{x''},\mathbf{x'})$

where $i = 1 \ldots m$, $h = 1 \ldots m$ except i.

Rules (P.6) above can be removed since $P_h(\mathbf{k},\mathbf{x'}) \wedge \neg P_h(\mathbf{k},\mathbf{x'})$ can not hold.

Given that, by (1), $\iota P_h(\mathbf{k},\mathbf{x'}) \rightarrow P_h(\mathbf{k},\mathbf{x'})$ we transform rules (P.7) in (30).

Since, by (2), $\delta P_i(\mathbf{k},\mathbf{x}) \rightarrow P^\circ_i(\mathbf{k},\mathbf{x})$, by (3), $\mu P_h(\mathbf{k},\mathbf{x''},\mathbf{x'}) \rightarrow P^\circ_h(\mathbf{k},\mathbf{x''})$ and given that $\mathbf{k}$ is key for P, then $\mathbf{x}=\mathbf{x''}$ and thus rules (P.8) can also be removed since they are subsumed by (22). ⊕

References

[ABW88] Apt,K.R.;Blair,H.A.;Walker,A. "Towards a theory of declarative knowledge". In Minker, J (Ed.) "Foundations of deductive databases and logic programming", Morgan Kaufmann Pub., 1988, pp 89-148.

[BaR86] Bancilhom,F; Ramakrishan,R. "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.

[BCL89] Blakeley,J.A.; Coburn,N.; Larson,P. "Updating derived relations: Detecting irrelevant and autonomously computable updates", ACM TODS, Vol. 14, No.3, September 1989, pp. 369-400.

[BrD88] Bry,F; Decker,H. "Préserver l'integrité d'une base de données deductive: une methode et son implementation". In compte-rendu des 4èmes Journées Base de Donnés Avancées, Mai 1988, Bénodet, France (in french).

[BDM88] Bry.F; Decker,H; Manthey,R. "A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases", Proc of Extending Database Technology, Venice, 1988, pp. 488-505.

[BMM90] Bry,F.; Manthey,R.; Martens,B. "Integrity verification in knowledge bases", ECRC report D.2.1.a, April 1990, Munich, Germany.

[BuC79] Buneman,O.P.; Clemons,E.K. "Efficiently monitoring relational databases", ACM TODS, Vol.4,No.3,September 1979, pp. 368-382.

[CBB+89] Chakravarthy, S.; Blaustein,B.; Buchmann,A. et al. "Hipac: A research project in Active, time-constrained database management", Final Project Report, XAIT, 1989.

[CeW91] Ceri,S.;Widom,J. "Deriving production rules for incremental view maintenance", Proc. of the 17th. VLDB Conf, Barcelona, 1991, pp 577-589.

[Dat90] Date,C.J. "Relational database. Writings 1985-1989", Addison-Wesley Publishing company, Inc.

[DaW89] Das,S.;Williams,H. "A path finding method for constraint checking in deductive databases", Data & Knowledge Engineering, 1989, No 4, pp. 223-224.

[Kow78] Kowalski, R. "Logic for data description", In Gallaire,H.; Minker,J.(Eds.) "Logic and Data Bases", Plenum Press, New York, 1978, pp. 77-103.

[Küc91] Küchenhoff,V. "On the efficient computation of the difference between consecutive database states", Proc. DOOD'91, Springer-Verlag, Munich, December 1991, pp. 478-502.

[Llo87] Lloyd, J.W. "Foundations of logic programming", 2nd. Ed. Springer-Verlag, 1987.

[LlT84] Lloyd, S.W.;Topor,R.W. "Making prolog more expresive", Journal of Logic programming, 1984, nº 3, pp 225-240.

[LlS91] Lloyd,J.W.;Shepherdson,J.C."Partial evaluation in logic programmming", Journal of Logic programming, 1991, nº 11, pp 217-247.

[Oli89] Olivé,A. "On the design and implementation of information systems from deductive conceptual models", Proc. of the 15th VLDB Conf., Amsterdam, 1989, pp. 3-11.

[Oli91] Olivé,A. " Integrity constraints checking in deductives databases", Proc. of the 17th. VLDB Conf., Barcelona, 1991, pp. 513-523.

[RCB+89] Rosenthal,A.; Chakravarthy,S.; Blaustein,B; Blakeley,J, "Situation monitoring for active databases", Proc. of the 15th VLDB Conf., Amsterdam, 1989, pp. 455-464.

[SaK88] Sadri,F; Kowalski,R. "Atheorem-proving approach to database integrity". In Minker, J (Ed.) " Foundations of deductive databases and logic programming", Morgan Kaufmann Pub.,1988, pp. 313-362.

[Urp91a] Urpí, T. "An approach to monitoring changes in deductive databases". Technical Report LSI-91-23, Universitat Politècnica de Catalunya, 1991.

[Urp91b] Urpí, T. "An approach to monitoring changes in deductive databases", Proc. of the Second International Workshop on the Deductive Approach to Information Systems and Databases, Aiguablava (Catalonia), september 1991, pp. 87-113.

[WiF90] Widom,F.; Finkelstein,S.J. "Set-oriented production rules in relational database systems", ACM SIGMOD Conf., May 1990, pp. 259-270.

[WCL91] Widom,J.; Cochrane,R.J.; Lindsay,B.G "Implementing set-oriented production rules as an extension to Starburst", Proc. of the 17th VLDB, Barcelona, 1991, pp. 275-285.