*VLDB Journal,* **4**, 193-241 (1995), Georg Gottlob, Editor
©VLDB

# Updating Knowledge Bases While Maintaining Their Consistency

**Ernest Teniente and Antoni Olivé**

*Received March 10, 1993; revised version received, June 22, 1994; accepted September 14, 1994.*

**Abstract.** When updating a knowledge base, several problems may arise. One of the most important problems is that of integrity constraints satisfaction. The classic approach to this problem has been to develop methods for *checking* whether a given update violates an integrity constraint. An alternative approach consists of trying to repair integrity constraints violations by performing additional updates that *maintain* knowledge base consistency. Another major problem in knowledge base updating is that of *view updating,* which determines how an update request should be translated into an update of the underlying base facts. We propose a new method for updating knowledge bases while maintaining their consistency. Our method can be used for both integrity constraints maintenance and view updating. It can also be combined with any integrity checking method for view updating and integrity checking. The kind of updates handled by our method are: updates of base facts, view updates, updates of deductive rules, and updates of integrity constraints. Our method is based on events and transition rules, which explicitly define the insertions and deletions induced by a knowledge base update. Using these rules, an extension of the SLDNF procedure allows us to obtain all possible minimal ways of updating a knowledge base without violating any integrity constraint.

**Key Words.** View updating, integrity checking, integrity maintenance.

## 1. Introduction

Knowledge bases generalize relational databases by including not only base facts and integrity constraints, but also deductive rules. Using these rules, new facts may be derived from facts explicitly stored. Among other components, knowledge bases include an update processing system that provides the users with a uniform

Ernest Teniente, Ph.D., is Associate Professor, and Antoni Olivé, Ph.D., is Professor, Universitat Politècnica de Catalunya, Facultat d'Informàtica, Pau Gargallo 5, E-08028 Barcelona–Catalonia.

interface in which they can request different kinds of updates (i.e., updates of base facts, updates of derived facts, updates of deductive rules, and updates of integrity constraints).

Several problems may arise when updating a knowledge base (Abiteboul, 1988; Kowalski, 1992). Perhaps the best-known problem is that of *integrity constraints checking.* An integrity constraint is a condition that a knowledge base is required to satisfy at any time. Integrity checking is the process of verifying that a given base update (a set of insertions and/or deletions of base facts) satisfies the integrity constraints. If some constraint is violated, then the update is rejected; otherwise the update is accepted. Efficient integrity checking methods have been developed for relational (Nicolas, 1982) and deductive databases (Bry et al., 1990; Olivé, 1991). The problem has also been studied for full, first-order logical databases (Gallaire et al., 1984; Reiter, 1984).

An alternative way to deal with integrity constraints is *integrity constraints maintenance,* which is a process that also starts with a given base update and integrity constraints except that, if some integrity constraint is violated, an attempt is made to find a *repair,* that is, an additional set of insertions and/or deletions of base facts to be added to the base update, such that the resulting base update satisfies all integrity constraints. In general, there may be several repairs and the user must select one of them. In some cases, no such repair exists, and the base update must be rejected.

As a simple example, assume that in a relational database, a relation $R$ has been defined with a key attribute $A$. Suppose that a user requests the insertion of a new tuple t1 into $R$, having the same value for $A$ as that of an existing tuple t2. In integrity checking, the request would be rejected, because it violates the key constraint, while in integrity maintenance the request would be "repaired." In this case, a possible repair would be the deletion of t2.

Some integrity maintenance methods have been developed for relational databases (Dayal and Bernstein, 1982), usually restricted to particular integrity constraints (such as keys, functional dependencies, or subset constraints). Recently, some methods have been developed for deductive databases (Ceri and Widom, 1990; Moerkotte and Lockemann, 1991; Ceri et al., 1992). The problem has also been studied for logical databases (Fagin et al., 1983, 1986; Gärdenfors, 1988; Winslett, 1990).

Another well-known problem is that of *updating derived facts* (also known as *view updating*). View updating is concerned with determining how a request to update a view can be translated appropriately into correct updates of the underlying base (stored) facts. In general, several translations may exist, and the user must select one of them. This problem has attracted much research during the last years in relational databases (Bancilhon and Spyratos, 1981; Cosmadakis and Papadimitriou, 1984; Masunaga, 1984; Keller, 1985, 1986; Date, 1986; Langerak, 1990; Larson and Sheth, 1991) and in deductive databases (Tomasic, 1988; Bry, 1990; Decker, 1990; Kakas and Mancarella, 1990; Guessoum and Lloyd, 1990, 1991; Torlone and Atzeni, 1991; Teniente and Olivé, 1992; Atzeni and Torlone, 1992). It also has been studied for logical databases (Fagin et al., 1983, 1986).

In principle, some translations corresponding to a given view update request may not satisfy the integrity constraints. For this reason, view updating is usually followed by an integrity checking process. The result of the combined process of view updating and integrity checking is the subset of translations obtained by view updating that would leave the knowledge base consistent. In some cases, no such translation is found, and then the view update request is rejected.

However, it is also possible to combine view updating and integrity maintenance. If a given translation does not satisfy some integrity constraint it is "repaired," thus adding new insertions and/or deletions of base facts to the translation. The result of the combined process is a set (possibly empty) of translations, that do not necessarily correspond to a subset of the translations obtained by view updating alone.

In this article, we describe what we call the Events Method, which can be used for integrity constraints maintenance, view updating, or their combination (Teniente, 1992). This method also can be combined with any integrity checking method for view updating and integrity checking. We presented a simplified version of the method for a particular case of view updating in deductive databases (Teniente and Olivé, 1992).

Our method is an application of an approach developed for the design of information systems from deductive conceptual models (Olivé, 1989). The knowledge base is augmented with a set of rules, called transition and event rules, which explicitly define insertions and deletions induced by an update. These rules are then used for updating a knowledge base. The rules also have been used for developing a new integrity checking method (Olivé, 1991) and for condition monitoring in active databases (Urpí and Olivé, 1992; Urpí, 1993). Our method takes into account not only classical updates of base facts and view updates, but also other kinds of updates, such as insertions and deletions of deductive rules and integrity constraints.

This article is organized as follows: Section 2 reviews basic concepts of knowledge bases. Section 3, which is based on Olivé (1991), reviews the event concept and the procedure for automatically deriving transition and event rules. Section 4 discusses the application of these rules to view updating. In Section 5, we present our method for combining view updating with integrity maintenance and integrity checking. In Section 6, we show correctness and completeness of our method. Section 7 extends the types of updates by also considering insertion and deletion of deductive rules and integrity constraints. Section 8 presents additional features of the method. In Sections 9 and 10, we compare in detail our method with related literature in view updating and in integrity constraints maintenance, respectively. In Section 11, we present our conclusions. We assume that the reader is familiar with logic programming (Lloyd, 1987).

## 2. Basic Definitions and Notation

In this section, we briefly review some definitions of the basic concepts related to first-order theories and knowledge bases (Gallaire et al., 1984; Lloyd, 1987; Ullman, 1988), and present our notation.

Throughout this article, we consider a first-order language with a universe of constants, a set of variables, a set of predicate names, and without function symbols. We will use names beginning with a capital letter for predicate symbols and constants (with the exception that constants also are permitted to be numbers) and names beginning with a lower case letter for variables.

A *term* is a variable symbol or a constant symbol (i.e., we restrict ourselves to function-free terms). We assume that the possible values for the terms range over finite domains. If $P$ is an m-ary predicate symbol and $t_1, ..., t_m$ are terms, then $P(t_1, ..., t_m)$ is an *atom*. The atom is *ground* if every $t_i$ $(i = 1, ..., m)$ is a constant. A *literal* is defined as either an atom or a negated atom.

A *fact* is a formula of the form:

$$P(t_1, ..., t_m) \leftarrow$$

where $P(t_1, ..., t_m)$ is a ground atom.

A *deductive rule* is a formula of the form:

$$P(t_1, ..., t_m) \leftarrow L_1 \wedge ... \wedge L_n \quad \text{with } m \geq 0, n \geq 1$$

where $P(t_1, ..., t_m)$ is an atom denoting the *conclusion*, and $L_1, ..., L_n$ are literals representing *conditions*. $P(t_1, ..., t_m)$ is called the *head* and $L_1 \wedge ... \wedge L_n$ the *body* of the deductive rule. Variables in the conclusion or in the conditions are assumed to be universally quantified over the whole formula. If a condition is an atom, then it is a *positive condition* of the deductive rule. If a condition is a negated atom, then it is a *negative condition*. The definition of a predicate $P$ is the set of all rules in the knowledge base that have $P$ in their head. We assume that the terms in the head are distinct variables.

An *integrity constraint* is a formula that the knowledge base is required to satisfy. We deal with constraints in *denial* form:

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

where the $L_i$ are literals, and all variables are assumed to be universally quantified over the whole formula. More general constraints can be transformed into this form by first applying the range form transformation (Decker, 1989), and then using the procedure described by Lloyd and Topor (1984).

For the sake of uniformity, we associate to each integrity constraint an inconsistency predicate *Icn*, with or without terms and, thus, they have the same form as the deductive rules. We call them *integrity rules*. Then, we would rewrite the former denial as $Ic1 \leftarrow L_1 \wedge ... \wedge L_n$.

For example, the following integrity rule

$Ic1\ (p,c) \leftarrow \texttt{Lives}\ (p,c) \wedge \neg\ \texttt{City}\ (c)$

states that `Persons` can only live in known `Cities`. If `Lives(John,A)` is a fact and there is no corresponding `City (A)` fact, then *Ic*1`(John,A)` will be true, and the integrity constraint *Ic*1 will be violated.

A *knowledge base* $K$ is a triple $K = (F, DR, IC)$ where $F$ is a set of facts, $DR$ a set of deductive rules, and $IC$ a set of integrity constraints. The set $F$ of facts is called the *extensional* part of the knowledge base and the set $DR$ of deductive rules is called the *intensional* part.

We assume that knowledge base predicates are partitioned into base and derived (view) predicates. A base predicate appears only in the extensional part and (possibly) in the body of deductive rules. A derived predicate appears only in the intensional part. Every knowledge base can be defined in this form (Bancilhon and Ramakrishnan, 1986).

As usual, we require that the knowledge base before and after any updates is *allowed* (Lloyd, 1987), that is, any variable that occurs in a deductive or integrity rule has an occurrence 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.

## 3. Transition and Event Rules

In this section, we define the concept of *event* (a key concept in our method) and describe the procedure for automatically deriving the transition and event rules for a given knowledge base. These rules depend only on the deductive and integrity rules, being independent from the base facts stored in the knowledge base and from any particular update.

### 3.1 Events

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

Due to the deductive and integrity rules, $U$ may induce other updates on some derived or inconsistency predicates. Let $P$ be one of such predicates in $K$, and let $P'$ denote the same predicate evaluated in $K'$. Assuming that a fact $P(\mathbf{C})$ holds in $K$, where $\mathbf{C}$ is a vector of constants, two cases are possible:

1*a*. $P'(\mathbf{C})$ also holds in $K'$ (both $P(\mathbf{C})$ and $P'(\mathbf{C})$ are true).

1*b*. $P'(\mathbf{C})$ does not hold in $K'$ ($P(\mathbf{C})$ is true but $P'(\mathbf{C})$ is false).

and assuming that $P'(\mathbf{C})$ holds in $K'$, two cases are also possible:

2*a*. $P(\mathbf{C})$ also holds in $K$ (both $P(\mathbf{C})$ and $P'(\mathbf{C})$ are true).

2*b*. $P(\mathbf{C})$ does not hold in $K$($P'(\mathbf{C})$ is true but $P(\mathbf{C})$ is false).

In case 1*b*, we say that a deletion event occurs in the transition, and we denote it by $\delta P(\mathbf{C})$. In case 2*b*, we say that an insertion event occurs in the transition, and we denote it by $\iota P(\mathbf{C})$.

For example, if `Works(employee, unit)` is a derived predicate, $\iota$`Works(John, Sales)` denotes an insertion event corresponding to predicate `Works`. `Works(John, Sales)` is true after the update and it was false before.

Formally, we associate to each derived or inconsistency predicate $P$ an *insertion event predicate* $\iota P$ and a *deletion event predicate* $\delta P$, defined as:

(1) $\forall\ \mathbf{x}(\iota P(\mathbf{x}) \leftrightarrow P'(\mathbf{x}) \wedge \neg P(\mathbf{x}))$
(2) $\forall\ \mathbf{x}(\delta P(\mathbf{x}) \leftrightarrow P(\mathbf{x}) \wedge \neg P'(\mathbf{x}))$

where $\mathbf{x}$ is a vector of variables. We then have the equivalencies (Urpí, 1993):

(3) $\forall\ \mathbf{x}(P'(\mathbf{x}) \leftrightarrow (P(\mathbf{x}) \wedge \neg\delta P(\mathbf{x})) \vee \iota P(\mathbf{x}))$
(4) $\forall\ \mathbf{x}(\neg P'(\mathbf{x}) \leftrightarrow (\neg P(\mathbf{x}) \wedge \neg\iota P(\mathbf{x})) \wedge\ \delta P(\mathbf{x}))$

If $P$ is a derived predicate, then $\iota P$ and $\delta P$ facts represent induced insertions and induced deletions, respectively. If $P$ is an inconsistency predicate, then $\iota P$ facts represent violations of its integrity constraint. Notice that, for inconsistency predicates, $\delta P$ facts cannot happen in any transition, since we assume that the knowledge base is consistent before the update and, thus, $P(\mathbf{x})$ is always false.

We also use definitions (1) and (2) above for base predicates. In this case, $\iota P$ and $\delta P$ facts represent insertions and deletions of base facts, respectively. We say that an event ($\iota P$ or $\delta P$) is base (respectively derived) if $P$ is base (respectively derived).

## 3.2 Transition Rules

Consider a derived or inconsistency predicate $P$ of the knowledge base. Assume that $P$ consists of $m$ rules, $m \geq 1$. For our purposes, we rename predicate symbols in the heads of the $m$ rules by $P_1$, ..., $P_m$, and we add the set of clauses:

(5) $P(\mathbf{x}) \leftarrow P_i(\mathbf{x}) \quad i = 1 \ldots m$

*Example 1:* Assume a knowledge base with the following rules:

$P(x,y) \leftarrow R(x,y)$
$P(x,y) \leftarrow R(x,z) \wedge P(z,y)$
$T(x) \quad \leftarrow P(x,y) \wedge \neg S(x)$

They would be rewritten as:

$P_1(x,y) \leftarrow R(x,y)$
$P_2(x,y) \leftarrow R(x,z) \quad \wedge P(z,y)$
$P(x,y) \quad \leftarrow P_1(x,y)$
$P(x,y) \quad \leftarrow P_2(x,y)$
$T_1(x) \quad \leftarrow P(x,y) \quad \wedge \neg S(x)$
$T(x) \quad \leftarrow T_1(x)$

Consider one of the rules, $P_i(\mathbf{x}) \leftarrow L_{i,1} \wedge ... \wedge L_{i,n}$. When this rule is to be evaluated in the new state, its form is $P'_i(\mathbf{x}) \leftarrow L'_{i,1} \wedge ... \wedge L'_{i,n}$, where $L'_{i,r}$ ($r = 1 ... n$) is obtained by replacing the predicate $Q$ of $L_{i,r}$ by $Q'$. Then, if we replace each literal in the body by its equivalent expression given in (3) or (4), we get a new rule, called the *transition rule,* which defines the new state predicate $P'_i$ in terms of old state predicates and events.

More precisely, if $L'_{i,r}$ is a positive literal $Q'_{i,r}(\mathbf{x}_{i,r})$, we apply (3) and replace it with:

$$(Q_{i,r}(\mathbf{x}_{i,r}) \wedge \neg\delta Q_{i,r}(\mathbf{x}_{i,r})) \vee \iota Q_{i,r}(\mathbf{x}_{i,r})$$

and if $L'_{i,r}$ is a negative literal $\neg Q'_{i,r}(\mathbf{x}_{i,r})$, we apply (4) and replace it with:

$$(\neg Q_{i,r}(\mathbf{x}_{i,r}) \wedge \neg\iota Q_{i,r}(\mathbf{x}_{i,r})) \vee \delta Q_{i,r}(\mathbf{x}_{i,r})$$

After distributing $\wedge$ over $\vee$, we get the set of transition rules for $P'_i$.

*Example 2:* Consider the rules given in *Example 1.* In the new state, they have the form:

$$P'_1(x,y) \leftarrow R'(x,y)$$
$$P'_2(x,y) \leftarrow R'(x,z) \wedge P'(z,y)$$
$$T'_1(x) \leftarrow P'(x,y) \wedge \neg S'(x)$$

Then, replacing the literals in the body by their equivalent expressions given by (3) and (4) we get:

$$P'_1(x,y) \leftarrow [(R(x,y) \wedge \neg\delta R(x,y)) \vee \iota R(x,y)]$$
$$P'_2(x,y) \leftarrow [(R(x,z) \wedge \neg\delta R(x,z)) \vee \iota R(x,z)] \wedge [(P(z,y) \wedge \neg\delta P(z,y)) \vee \iota P(z,y)]$$
$$T'_1(x) \leftarrow [(P(x,y) \wedge \neg\delta P(x,y)) \vee \iota P(x,y)] \wedge [(\neg S(x) \wedge \neg\iota S(x)) \vee \delta S(x)]$$

and, after distributing $\wedge$ over $\vee$, we get the following set of transition rules for $P'_1$, $P'_2$, and $T'_1$:

$$P'_{1,1}(x,y) \leftarrow R(x,y) \wedge \neg\delta R(x,y)$$
$$P'_{1,2}(x,y) \leftarrow \iota R(x,y)$$
$$P'_{2,1}(x,y) \leftarrow R(x,z) \wedge \neg\delta R(x,z) \wedge P(z,y) \wedge \neg\delta P(z,y)$$
$$P'_{2,2}(x,y) \leftarrow R(x,z) \wedge \neg\delta R(x,z) \wedge \iota P(z,y)$$
$$P'_{2,3}(x,y) \leftarrow \iota R(x,z) \wedge P(z,y) \wedge \neg\delta P(z,y)$$
$$P'_{2,4}(x,y) \leftarrow \iota R(x,z) \wedge \iota P(z,y)$$

$$T'_{1,1}(x) \leftarrow P(x,y) \wedge \neg\delta P(x,y) \wedge \neg S(x) \wedge \neg\iota S(x)$$
$$T'_{1,2}(x) \leftarrow P(x,y) \wedge \neg\delta P(x,y) \wedge \delta S(x)$$
$$T'_{1,3}(x) \leftarrow \iota P(x,y) \wedge \neg S(x) \wedge \neg\iota S(x)$$
$$T'_{1,4}(x) \leftarrow \iota P(x,y) \wedge \delta S(x)$$

with:

$P'_1(x) \leftarrow P'_{1,j}(x) \quad j = 1,2$
$P'_2(x) \leftarrow P'_{2,j}(x) \quad j = 1, ..., 4$
$T'_1(x) \leftarrow T'_{1,j}(x) \quad j = 1, ..., 4$

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

$$\begin{aligned} O(L'_{i,r}) &= (Q_{i,r}(\mathbf{x}_{i,r}) \wedge \neg\delta Q_{i,r}(\mathbf{x}_{i,r})) && \text{if } L'_{i,r} = Q'_{i,r}(\mathbf{x}_{i,r}) \\ &= (\neg Q_{i,r}(\mathbf{x}_{i,r}) \wedge \neg\iota Q_{i,r}(\mathbf{x}_{i,r})) && \text{if } L'_{i,r} = \neg Q'_{i,r}(\mathbf{x}_{i,r}) \end{aligned}$$

$$\begin{aligned} N(L'_{i,r}) &= \iota Q_{i,r}(\mathbf{x}_{i,r}) && \text{if } L'_{i,r} = Q'_{i,r}(\mathbf{x}_{i,r}) \\ &= \delta Q_{i,r}(\mathbf{x}_{i,r}) && \text{if } L'_{i,r} = \neg Q_{i,r}(\mathbf{x}_{i,r}) \end{aligned}$$

Both $O(L'_{i,r})$ and $N(L'_{i,r})$ express conditions for which $L'_{i,r}$ is true. $O(L'_{i,r})$ corresponds to the case that $L'_{i,r}$ holds because $L_{i,r}$ was already true in the *O*ld state and has not been deleted, while $N(L'_{i,r})$ corresponds to the case that $N(L'_{i,r})$ holds because it is *N*ew, induced in the transition, and it was false before. Note that $O(L'_{i,r}) \rightarrow L_{i,r}$ and $N(L'_{i,r}) \rightarrow \neg L_{i,r}$.

With this notation, the equivalencies (3) and (4) become:

(6) $\forall \mathbf{x}(P'(\mathbf{x}) \leftrightarrow O(P'(\mathbf{x})) \vee N(P'(\mathbf{x})))$
(7) $\forall \mathbf{x}(\neg P'(\mathbf{x}) \leftrightarrow O(\neg P'(\mathbf{x})) \vee N(\neg P'(\mathbf{x})))$

and applying them to each of the $L'_{i,r}$ $(r = 1 \ldots n)$ literals we get:

(8) $P'_i(\mathbf{x}) \leftarrow \wedge_{r=1}^{r=n} [O(L'_{i,r}) \vee N(L'_{i,r})]$

After distributing $\wedge$ over $\vee$, we get an equivalent set of $2^{ki}$ transition rules (where $k_i$ is the number of literals in the $P'_i$ rule), each of them with the general form:

(9) $P'_{i,j}(\mathbf{x}) \leftarrow \wedge_{r=1}^{r=n} [O(L'_{i,j,r}) \mid N(L'_{i,j,r})]$ with $j = 1 \ldots 2^{ki}$

and

(10) $P'_i(\mathbf{x}) \leftarrow P'_{i,j}(\mathbf{x}) \quad j = 1 \ldots 2^{ki}$

In rules (9), it will be useful to assume that the rule corresponding to $j = 1$ is:

(11) $P'_{i,1}(\mathbf{x}) \leftarrow O(L'_{i,1,1}) \wedge \ldots \wedge O(L'_{i,1,n})$

## 3.3 Insertion Event Rules

Let $P$ be a derived or inconsistency predicate. Insertion predicates $\iota P$ were defined in (1) as:

$\forall \mathbf{x}(\iota P(\mathbf{x}) \leftrightarrow P'(\mathbf{x}) \wedge \neg P(\mathbf{x}))$

If there are $m$ rules for predicate $P$, then $P'(\mathbf{x}) \leftrightarrow P'_1(\mathbf{x}) \vee \ldots \vee P'_m(\mathbf{x})$. Replacing $P'(\mathbf{x})$ in (1) we obtain:

$\iota P(\mathbf{x}) \leftarrow P'_i(\mathbf{x}) \wedge \neg P(\mathbf{x}) \quad$ with $i = 1 \ldots m$

and again replacing $P'_i(\mathbf{x})$ with its equivalent definition given in (10) we get:

(12) $\iota P(\mathbf{x}) \leftarrow P'_{i,j}(\mathbf{x}) \wedge \neg P(\mathbf{x})$ for $i = 1 \ldots m$ and $j = 1 \ldots 2^{ki}$

and given that $P(\mathbf{x}) \leftrightarrow P_1(\mathbf{x}) \vee \ldots \vee P_m(\mathbf{x})$ we obtain:

(13) $\iota P(\mathbf{x}) \leftarrow P'_{i,j}(\mathbf{x}) \wedge \neg P_1(\mathbf{x}) \wedge \ldots \wedge \neg P_m(\mathbf{x})$ for $i = 1 \ldots m$ and $j = 1 \ldots 2^{ki}$

The rules in (13) are called the *insertion event rules* of predicate *P*. These rules allow us to deduce which $\iota P$ facts (induced insertions) happen in a transition in terms of old state predicates and events. If $P$ is an integrity constraint, $\iota P$ facts correspond to violations of the integrity constraint $P$.

We can remove the rules from (13) that correspond to $j = 1$, and which have the form shown in (11). These rules cannot produce $\iota P$ facts, since $P'_{i,1}(\mathbf{x}) \rightarrow P_i(\mathbf{x})$. Therefore, we reduce the set (13) to:

(14) $\iota P(\mathbf{x}) \leftarrow P'_{i,j}(\mathbf{x}) \wedge \neg P_1(\mathbf{x}) \wedge \ldots \wedge \neg P_m(\mathbf{x})$ for $i = 1 \ldots m$ and $j = 2 \ldots 2^{ki}$

On the other hand, there are three important simplifications that, when applicable, transform some of the rules in (14) into equivalent, but simplified rules, which can be evaluated more efficiently.

The first is the *consistency assumption simplification.* It can be applied when $P$ is an inconsistency predicate. In this case, we can remove literals $\neg P_k(\mathbf{x})$ in (14), since we will assume that the knowledge base is consistent before the update and, thus, $P_k(\mathbf{x})$ must be false, for all $k$ and $\mathbf{x}$.

To explain the other simplifications, we need to add some new terminology. Given a rule $P_i(\mathbf{x}) \leftarrow L_{i,1} \wedge \ldots \wedge L_{i,n}$, we distinguish two parts in its body: the universal and the existential part. The universal part, denoted $U(P_i)$, is the conjunction of the literals in the body whose variables are a subset of $\mathbf{x}$. The existential part, denoted $E(P_i)$, is the conjunction of the literals in the body having some variable which is not in $\mathbf{x}$. We have $P_i(\mathbf{x}) \leftrightarrow U(P_i) \wedge E(P_i)$.

For example, in the rule:

$T_1(x) \leftarrow P(x,y) \wedge \neg S(x)$

we have:

$U(T_1) = \neg S(x)$
$E(T_1) = P(x,y)$

If $E(P_i)$ is not empty, sometimes we will need an auxiliary predicate $E\_P_i$ defined by the rule: $E\_P_i(x) \leftarrow E(P_i)$.

In the above example, this predicate would be defined as:

$E\_T_1(x) \leftarrow P(\mathbf{x},\mathbf{y})$

The other two simplifications can be applied when $U(P_i)$ is not empty. We call them the *New and Old simplification.* The New simplification can be applied to rules in (14), for which the transition rule corresponding to $P'_{i,j}(\mathbf{x})$ has some literal $N(L'_{i,j,h})$ in $U(P'_{i,j})$. In this case, we can remove the literal $\neg P_i(\mathbf{x})$, and the rule becomes:

(15) $\iota P(\mathbf{x}) \leftarrow P'_{i,j}(\mathbf{x}) \wedge \neg P_1(\mathbf{x}) \wedge \ldots \wedge \neg P_{i-1}(\mathbf{x}) \quad \wedge \neg P_{i+1}(\mathbf{x}) \wedge \ldots \wedge \neg P_m(\mathbf{x})$

We give the proof in Appendix A. The basic idea is that, if the rule $P'_{i,j}(\mathbf{x})$ has a literal $N(L'_{i,j,h})$ in $U(P'_{i,j})$, then $P'_{i,j}(\mathbf{x})$ are $P'_i$ facts, true in the new state, but false in the old state and, therefore, $P'_{i,j}(\mathbf{x}) \rightarrow \neg P_i(\mathbf{x})$.

The Old simplification applies to rules in (14) for which the transition rule corresponding to $P'_{i,j}(\mathbf{x})$ has all literals in $U(P'_{i,j})$ of the form $O(L'_{i,j,h})$. In such a case, we can remove $U(P_i)$ from $P_i(\mathbf{x}) \leftrightarrow U(P_i) \wedge E(P_i)$ and the rule becomes:

(16) $\iota P(\mathbf{x}) \leftarrow P'_{i,j}(\mathbf{x}) \wedge \neg P_1(\mathbf{x}) \wedge \ldots \wedge \neg P_{i-1}(\mathbf{x}) \wedge \neg E\_P_i(\mathbf{x}) \wedge \neg P_{i+1}(\mathbf{x}) \wedge \ldots \wedge \neg P_m(\mathbf{x})$

We also give the proof in Appendix A. The basic idea here is that if all literals in $U(P'_{i,j})$ have the form $O(L'_{i,j,h})$, then $U(P'_{i,j}) \rightarrow U(P_i)$.

The following example illustrates the application of these simplifications.

*Example 3:* Consider predicate $T$ as defined in *Example 2.* Applying (14), we get the insertion event rules:

$\iota T(x) \leftarrow T'_{1,j}(x) \wedge \neg T_1(x)$ with $j = 2 \ldots 4$

Given that $U(T'_{1,2})$ has literal $N(\neg S'(\mathbf{x})) = \delta S(\mathbf{x})$, we can apply the New simplification to the first rule, and we obtain:

$\iota T(x) \leftarrow T'_{1,2}(x)$

where we have removed $\neg T_1(x)$ because $\delta S(x) \rightarrow \neg S(x) \rightarrow \neg T_1(x)$.

The same considerations apply to the rule corresponding to $j = 4$, and we obtain:

$\iota T(x) \leftarrow T'_{1,4}(x)$

The Old simplification can be used in the rule corresponding to $T'_{1,3}$, since $U(T'_{1,3}) = \neg S(x) \wedge \neg \iota S(x) = O(\neg S(x))$. We obtain:

$\iota T(x) \leftarrow T'_{1,3}(\mathbf{x}) \wedge \neg E\_T_1(x)$

with: $E\_T_1(x) \leftarrow P(x,y)$

## 3.4 Deletion Event Rules

Let $P$ be a derived predicate. Deletion predicates $\delta P$ were defined in (2) as:

$\forall \mathbf{x}(\delta P(\mathbf{x}) \leftrightarrow P(\mathbf{x}) \wedge \neg P'(\mathbf{x}))$

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

(17) $\delta P(\mathbf{x}) \leftarrow P_i(\mathbf{x}) \wedge \neg P'(\mathbf{x}) \quad$ for $i = 1 \ldots m$

and replacing $P'(\mathbf{x})$ by its equivalent definition $P'(\mathbf{x}) \leftrightarrow P'_1(\mathbf{x}) \vee \ldots \vee P'_m(\mathbf{x})$, we obtain:

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

These rules can be transformed into a set of equivalent but simplified rules, which can be evaluated more efficiently. Let $k^u_i$ be the number of literals in $U(P_i)$,

$k_i^e$ be the number of literals in $E(P_i)$, and let $A/B$ denote $A$ without $B$ (true if $A{=}B$), where $A$ is a conjunction of literals, and $B$ is a literal. Then, a rule in (18) is equivalent to the following $k_i^u + k_i^e$ rules:

(19) $\delta P(\mathrm{x}) \leftarrow U(P_i)/L_{i,j} \wedge [\delta Q_{i,j}(\mathrm{x}_{i,j}) \mid \iota Q_{i,j}(\mathrm{x}_{i,j})] \wedge E(P_i) \wedge \alpha$
for $i = 1 \ldots m,\ j = 1 \ldots k_i^u$

(20) $\delta P(\mathrm{x}) \leftarrow U(P_i) \wedge E(P_i)/L_{i,j} \wedge [\delta Q_{i,j}(\mathrm{x}_{i,j}) \mid \iota Q_{i,j}(\mathrm{x}_{i,j})] \wedge \neg E\_P'_i(\mathrm{x}) \wedge \alpha$
for $i = 1 \ldots m,\ j = 1 \ldots k_i^e$

where $\alpha \equiv \neg P'_1(\mathrm{x}) \wedge \ldots \wedge \neg P'_{i-1}(\mathrm{x}) \wedge \neg P'_{i+1}(\mathrm{x}) \wedge \ldots \wedge \neg P'_m(\mathrm{x})$, and where the first option in $[\delta Q_{i,j}(\mathrm{x}_{i,j}) \mid \iota Q_{i,j}(\mathrm{x}_{i,j})]$ is taken if $L_{i,j}$ is positive, and the second if $L_{i,j}$ is negative.

We prove the above transformation in Appendix A. The main idea is that deletions of $P$ are induced by deletions of positive literals (or insertions of negative literals) in $P_i$. This explains why we get $k_i^u + k_i^e$ rules.

This set of rules is called the *deletion event rules* for predicate $P$. They allow us to deduce which $\delta P$ facts (induced deletions) happen in a transition in terms of old state predicates and events.

*Example 4.* Again consider predicate $T$ as defined in *Example 2.* Applying (19) and (20), we get the deletion event rules:

$\delta T(x) \leftarrow \iota S(x) \wedge P(x,y)$
$\delta T(x) \leftarrow \neg S(x) \wedge \delta P(x,y) \wedge \neg E\_T'_1(x)$

with:

$E\_T'_{1,1}(x) \leftarrow P(x,y) \wedge \neg \delta P(x,y)$
$E\_T'_{1,2}(x) \leftarrow \iota P(x,y)$
$E\_T'_1(x) \leftarrow E\_T'_{1,j}(x) \quad j = 1, 2$

Observe that the literal $\neg T'_1(x)$ is not required in the body of the first rule because $\iota S(x) \rightarrow S'(x) \rightarrow \neg T'_1(x)$. In the second rule we can only remove $\neg U(T'_1(x))$.

## 3.5 Augmented Knowledge Base

Let $K$ be a knowledge base. The *augmented knowledge base,* denoted by $A(K)$, consists of $K$ and the transition, insertion, and deletion event rules for $K$. In next sections, we will discuss the important role of $A(K)$ in our method for updating knowledge bases while maintaining their consistency. It is easy to show that if $K$ is allowed, then $A(K)$ is also allowed.

## 4. View Updating

In this section, we present the Events Method for view updating in knowledge bases. To simplify the presentation, the problem of integrity constraints satisfaction will not be considered until Section 5. That is, we assume for the moment that the knowledge base does not contain any integrity constraint.

### 4.1 Events Method

The view update problem is concerned with determining how a request to update a view can be appropriately translated into updates of the underlying base facts. Two basic approaches have been proposed to solve this problem. The first one suggests treating views as *abstract data types* (Furtado and Casanova, 1985; Manchanda and Warren, 1988), so that the definition of a view includes all permissible view updates together with their translation.

The second approach is to define a general translation procedure (a *translator;* Tomasic, 1988; Bry, 1990; Decker, 1990; Kakas and Mancarella, 1990; Guessoum and Lloyd, 1990, 1991; Torlone and Atzeni, 1991; Teniente and Olivé, 1992; Atzeni and Torlone, 1992). Inputs to the translator are a view definition, a view update request, and the current knowledge base; the output is a knowledge base update that satisfies the request. The method that we propose in this article follows the translator approach.

Usually, there are several possible ways of satisfying a view update request. Our approach consists of generating all minimal translations for a given request. A translation is *minimal* when there does not exist a subset of it, which is also a translation. In general, several minimal translations may exist. The problem of how to choose among them will not be addressed in this article (some comments on this problem can be found in Kakas and Mancarella, 1990).

Moreover, we consider only translations that involve solely updates of the extensional part of the knowledge base (i.e., insertions and deletions of ground facts of base predicates). For this reason, translations that involve updates of the intensional part of the knowledge base (e.g., deletion, insertion, or modification of rules, insertion of ground facts of view predicates) are not considered here. Several authors have argued the suitability of translating views in this way (e.g., Decker, 1990, Kakas and Mancarella, 1990).

For simplicity, we assume here that view updates are restricted to insertions and deletions. Later on, in Section 8, we will explain how to deal with modifications. In our method, an insertion (respectively, deletion) corresponds to an event $\iota P(\mathbf{C})$ (respectively, $\delta P(\mathbf{C})$), where $P'(\mathbf{C})$ is the fact that must hold (respectively, must not hold) in the new state of the knowledge base.

A translation of an insertion $\iota P(\mathbf{C})$ (respectively, deletion $\delta P(\mathbf{C})$), denoted by $T$, defines a set of insertions and/or deletions of base facts such that $P'(\mathbf{C})$ is (respectively, is not) a logical consequence of the completion of the knowledge base updated according to $T$. Due to the definition of the concept of event, we require that if $\iota P(\mathrm{C})$ corresponds to an insertion (respectively, $\delta P(\mathrm{C})$ to a deletion), then $P(\mathrm{C})$ must not hold in the current knowledge base (respectively, $P(\mathrm{C})$ must hold).

Our method is able to translate view update requests that consist of a set of insertions and/or deletions. That is, we deal with *multiple* (more than one view update request at the same time) and *mixed* (including insertions and deletions) *view update requests*. Thus, a multiple and mixed request of the form: insert($P_1$) and ... and insert($P_n$) and delete ($Q_1$) and ... and delete($Q_m$) will be denoted by the conjunction: $\iota P_1 \wedge \ldots \wedge \iota P_n \wedge \delta Q_1 \wedge \ldots \wedge \delta Q_m$. We note that the translations that satisfy a request do not depend on the order of the literals in the conjunction. Application of each translation leaves the knowledge base in a state that satisfies the multiple and mixed request. Therefore, we would obtain the same translations for the request $\{\iota P \wedge \delta Q\}$ as for $\{\delta Q \wedge \iota P\}$.

Let $K$ be a knowledge base, $A(K)$ its augmented knowledge base, $u$ a view update request that consists of a conjunction of derived events, and $T$ a translation consisting of a set of base events. In our method, a translation $T$ satisfies the request $u$ if, using SLDNF resolution (Lloyd, 1987), the goal $\{\leftarrow u\}$ succeeds from input set $A(K) \cup T$. The translation set $T$ is obtained by having some failed SLDNF derivation of $A(K) \cup \{\leftarrow u\}$ succeed. The possible ways in which a failed derivation may succeed correspond to the different translations $T_i$ that satisfy the request. If no translation $T$ is obtained, then the view update cannot be satisfied by changing only the extensional part of the knowledge base.

## 4.2 Simplified Case

In this section, we describe the Events Method for a particular kind of knowledge bases. More precisely, we assume that all variables appearing in the body of a deductive rule appear also in its conclusion. That is, following the terminology defined in Section 3, we consider that the knowledge base contains only rules that do not have an existential part ($E(P_i) = \emptyset$). For instance, the rule $P(\mathrm{x}) \leftarrow Q(\mathrm{x}) \wedge R(\mathrm{x,y}) \wedge \neg S(y)$ does not satisfy this condition since $E(P) = R(\mathrm{x,y}) \wedge \neg S(y)$.

*Example 5:* Let $K$ be a knowledge base with the following base and view predicates (adopted from Sadri and Kowalski, 1988):

| | |
|---|---|
| Alien (x) | x is registered as an alien. |
| Cr (x) | x has a criminal record. |
| Cit (x) | x is a citizen. |
| Rr (x) | x has right of residence. |

Assume that the current content of the knowledge base is the following:

| | |
|---|---|
| F.1 | Cit (John) |
| DR.1 | Rr(x) $\leftarrow$ Alien (x) $\wedge \neg$Cr (x) |
| DR.2 | Rr(x) $\leftarrow$ Cit (x) |

Transition, insertion, and deletion rules associated with this knowledge base are:

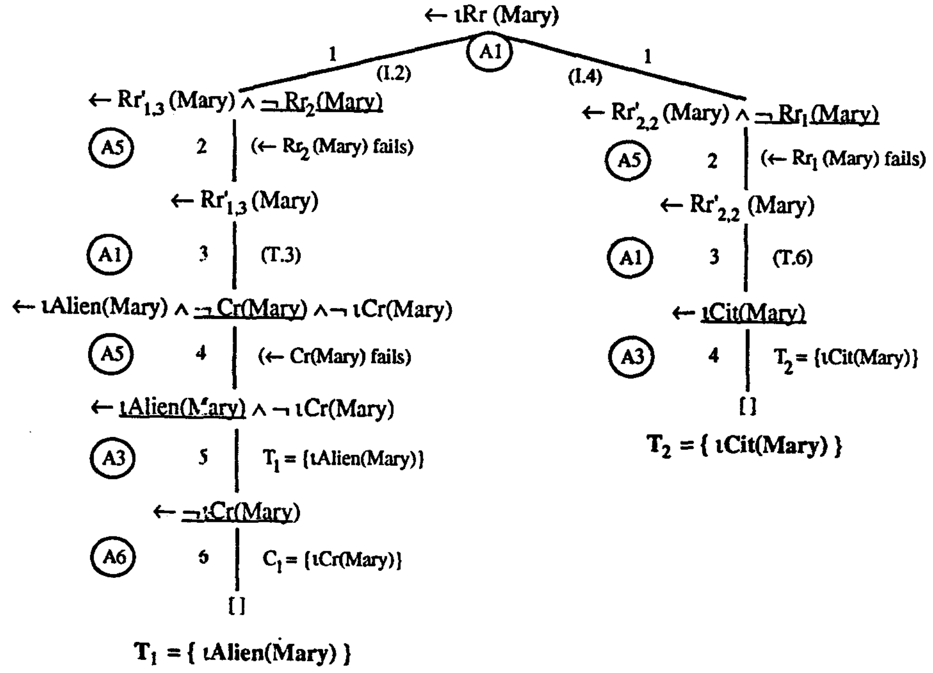
| | | |
|---|---|---|
| T.1 | $Rr'_{1,1}(x) \leftarrow \text{Alien}(x) \land \neg\delta\text{Alien}(x) \land \neg Cr(x) \land \neg\iota Cr(x)$ | |
| T.2 | $Rr'_{1,2}(x) \leftarrow \text{Alien}(x) \land \neg\delta\text{Alien}(x) \land \delta Cr(x)$ | |
| T.3 | $Rr'_{1,3}(x) \leftarrow \iota\text{Alien}(x) \land \neg Cr(x) \land \neg\iota Cr(x)$ | |
| T.4 | $Rr'_{1,4}(x) \leftarrow \iota\text{Alien}(x) \land \delta Cr(x)$ | |
| T.5 | $Rr'_{2,1}(x) \leftarrow Cit(x) \land \neg\delta Cit(x)$ | |
| T.6 | $Rr'_{2,2}(x) \leftarrow \iota Cit(x)$ | |
| T.7..10 | $Rr'_{1}(x) \leftarrow Rr'_{1,j}(x)$ | $j = 1 \ldots 4$ |
| T.11,12 | $Rr'_{2}(x) \leftarrow Rr'_{2,j}(x)$ | $j = 1, 2$ |
| I.1..3 | $\iota Rr(x) \leftarrow Rr'_{1,j}(x) \land \neg Rr_2(x)$ | $j = 2 \ldots 4$ |
| I.4 | $\iota Rr(x) \leftarrow Rr'_{2,2}(x) \land \neg Rr_1(x)$ | |
| D.1 | $\delta Rr(x) \leftarrow \delta\text{Alien}(x) \land \neg Cr(x) \land \neg Rr'_2(x)$ | |
| D.2 | $\delta Rr(x) \leftarrow \text{Alien}(x) \land \iota Cr(x) \land \neg Rr'_2(x)$ | |
| D.3 | $\delta Rr(x) \leftarrow \delta\text{Cit}(x) \land \neg Rr'_1(x)$ | |

Let the view update request be the insertion of the derived fact *Rr* (Mary). Translations that satisfy this request are obtained by having some failed SLDNF derivation of $A(K) \cup \{\leftarrow \iota Rr\ (\text{Mary})\}$ succeed. This is shown in Figure 1 (circled labels at the left of a derivation are references to the rules of the method, defined in Section 4.4). Steps 1 to 4 in the left derivation are SLDNF resolution steps. At step 5, the selected literal is $\iota$Alien (Mary), which is a positive base event. To get a successful derivation, we must include it in the input set, and use it as input clause. Therefore, it is added to the translation set *T*. At step 6, the selected literal is $\neg\iota Cr$ (Mary). To get a successful derivation for this branch, $\iota Cr$ (Mary) must fail, which implies that it must not belong to *T*. We use an auxiliary set *C*, which we call the *condition set,* to check that the event $\iota Cr$ (Mary) will not be included in *T* during the derivation process. Thus, in step 6, $\iota Cr$ (Mary) is included in *C*.

In general, a condition set *C* is a set of base events that cannot be included in the translation *T*. Due to the opposite meaning of *T* and *C*, before including a base event in *T* (respectively in *C*), we must check that it does not belong to the subset of *C* (respectively of *T*) already determined. If it does, we get a contradiction for the current branch and, then, no valid translation can be obtained from it. In the above example, no contradiction is found when including base events in *T* or *C*.

Once we get the empty clause, the process finishes, and *T* gives the base events that produce the desired effect. From this derivation we have $T_1 = \{\iota\text{Alien(Mary)}\}$. Now the update request will be satisfied by updating the extensional part of the knowledge base with these base events. In this case, the request is achieved by

**Figure 1. Successful derivations of request $\leftarrow \iota Rr$ (Mary)**



inserting the fact that Mary is registered as an alien.

In a similar way, the right derivation reaches the goal $\leftarrow \iota Cit$ (Mary), that can be succeeded by including $\iota Cit$ (Mary) in the input set (step 4). Another possible translation that satisfies the request is $T_2 = \{\iota Cit(\text{Mary})\}$.

Selecting clauses I.1 and I.3 in step 1 of Figure 1, we get failed derivations that cannot be succeeded (these derivations are not shown in the tree above). The first (I.1) would require Alien(Mary), which does not hold, while the second (I.3) would require deleting *Cr*(Mary), which is not possible. Thus, no other translation can be obtained from them.

*Example 6:* Consider the knowledge base defined in *Example 5,* and assume that we want to delete the view fact that John has right of residence. All possible translations that satisfy this request are obtained by having a failed derivation of $A(K) \cup \{\leftarrow \delta Rr \text{ (John)}\}$ succeed. One of these derivations is shown in Figure 2. Step 1 is an SLDNF resolution step. At step 2, the selected base event $\delta Cit$ (John) is included in $T$. After this step, we get the goal $\leftarrow \neg Rr'_1$ (John). This derivation will succeed if we ensure that the subsidiary tree rooted at $\leftarrow Rr'_1$ (John) fails finitely. Part of this subsidiary tree is shown in Figure 3.

In Figure 3, steps 1, 2 and 3 are SLDNF resolution steps. After these steps, we get the goal $\leftarrow \iota$Alien (John) $\wedge \neg \iota Cr$ (John). All possible ways in which this goal can fail must be taken into account (i.e., obtain all the possible translations

**Figure 2. Successful derivation of request $\leftarrow \delta Rr$ (John)**

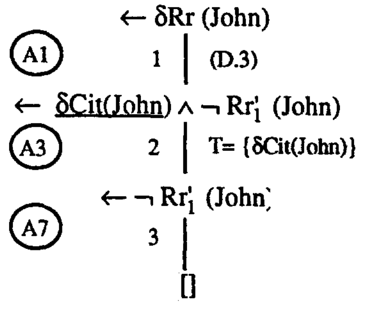
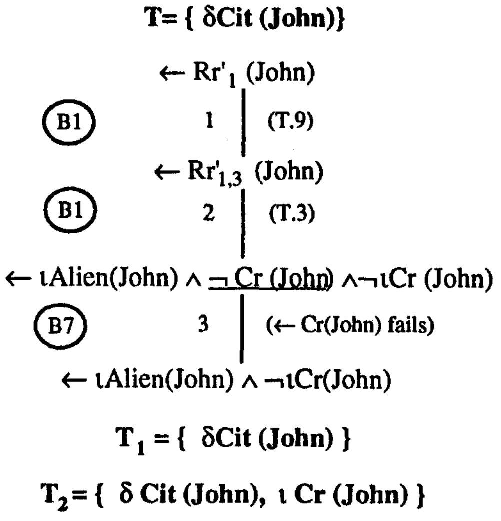


**Figure 3. Part of subsidiary tree of $\leftarrow Rr'_1$ (John)**


T= { δCit (John)}
← Rr'₁ (John)
B1
1
(T.9)
← Rr'₁,₃ (John)
B1
2
(T.3)
← ιAlien(John) ∧ ¬ Cr (John) ∧¬ιCr (John)
B7
3
(← Cr(John) fails)
← ιAlien(John) ∧ ¬ιCr(John)
T₁ = { δCit (John) }
T₂ = { δ Cit (John), ι Cr (John) }

that could make this goal fail). In this case, we have two options: to add $\iota$Alien (John) to the condition set $C$ or to include $\iota Cr$ (John) in the translation set $T$.

Selecting clauses T.7, T.8, and T.10 in the first step of Figure 3, we get failed derivations that do not lead to any other solution (these derivations are not shown in the example). Then, in the initial derivation of Figure 2, we get the empty clause, and the derivation is finished, obtaining two different translations: $T_1 = \{\delta Cit$ (John)$\}$ and $T_2 = \{\delta Cit$ (John), $\iota Cr$ (John)$\}$. Notice that $T_2$ is not a minimal translation, because it has a subset, $T_1$, which is itself a translation. Because we are interested in only minimal translations, $T_2$ would be refused.

## 4.3 General Case

In the previous section, we described the Events Method in the particular case where all variables appearing in the body of a deductive rule appear also in its conclusion. This simplification allows us to consider that the condition set $C$ contains only base events and, therefore, the procedure for verifying that all conditions are satisfied is restricted to checking whether a base event belongs to the condition set.

In the general case, when local variables appear in the definition of view predicates, the condition set $C$ will contain not only base events, but also some of the goals reached in the subsidiary derivations. For this reason, the process for verifying conditions must be modified. Now, before adding a base event to the translation set $T$, we have to guarantee that it is consistent with the content of the condition set $C$ already determined. This is done by ensuring, for each condition $C_i \in C$, that the tree rooted at $C_i$ fails finitely. Notice that this procedure may cause the inclusion of new elements in $T$ and/or in $C$.

Another important difference is that, in the general case, a non-ground base event may be selected during the derivations. Before including it in the translation set $T$, it must be fully instantiated. This can be done either by asking the user or by assigning default values. To obtain all translations, all possible instantiations must be taken into account. Note that, as we consider finite domains, the number of translations that satisfy an update request is always finite. We illustrate these extensions with an example.

*Example 7:* Let $K$ be a knowledge base containing the following predicates:

| | |
|---|---|
| Pract $(x,y)$ | $x$ practices $y$. |
| Sport $(x)$ | $x$ is a sport. |
| Athlete $(x)$ | $x$ is an athlete. |

The current content of the knowledge base is:

F.1 Pract (Sue,Chess)
F.2 Pract (Sue,Tennis)
F.3 Sport (Tennis)
DR.1 Athlete $(x) \leftarrow$ Pract $(x,y) \wedge$ Sport $(y)$

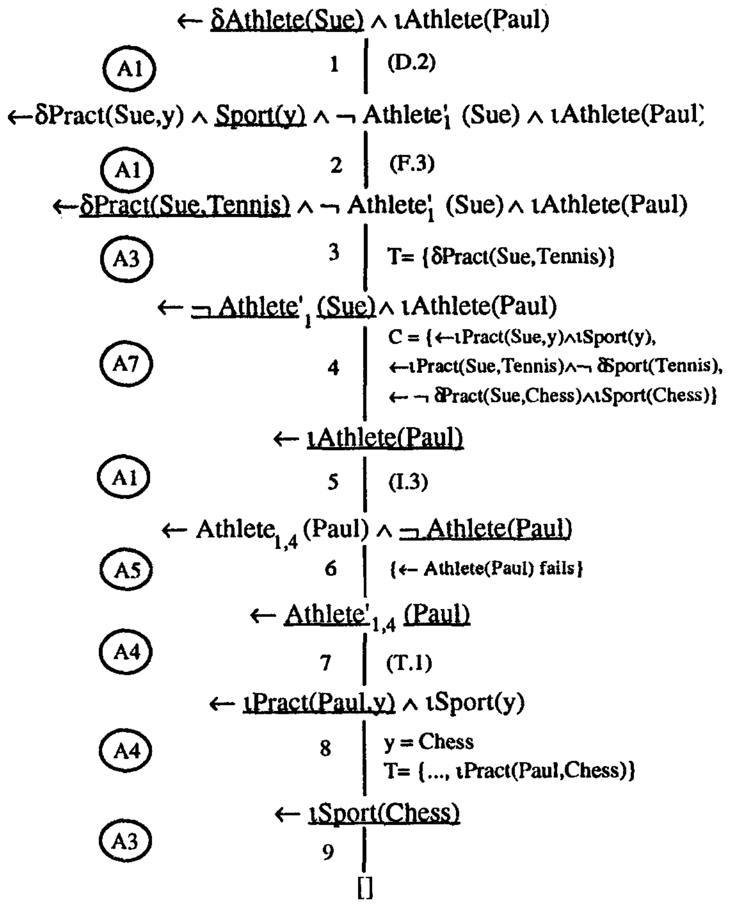
Transition and event rules associated with this knowledge base are:

T.1 $\text{Athlete}'_{1,1}(x) \leftarrow \text{Pract}(x,y) \wedge\neg\delta\text{Pract}(x,y) \wedge \text{Sport}(y) \wedge\neg\delta\text{Sport}(y)$
T.2 $\text{Athlete}'_{1,2}(x) \leftarrow \text{Pract}(x,y) \wedge\neg\delta\text{Pract}(x,y) \wedge\iota\text{Sport}(y)$
T.3 $\text{Athlete}'_{1,3}(x) \leftarrow \iota\text{Pract}(x,y) \wedge \text{Sport}(y) \wedge\neg\delta\text{Sport}(y)$
T.4 $\text{Athlete}'_{1,4}(x) \leftarrow \iota\text{Pract}(x,y) \wedge\iota\text{Sport}(y)$
T.5..8 $\text{Athlete}'_1\ (x) \leftarrow \text{Athlete}'_{1,j}(x)$ $\quad j = 1 \ldots 4$

I.1..3 $\iota\text{Athlete}\ (x) \leftarrow \text{Athlete}'_{1,j}(x) \wedge\neg\text{Athlete}\ (x)$ $\quad j = 2 \ldots 4$

D.1 $\delta\text{Athlete}\ (x) \leftarrow \text{Pract}(x,y) \wedge\delta\text{Sport}\ (y) \wedge\neg\text{Athlete}'_1\ (x)$
D.2 $\delta\text{Athlete}\ (x) \leftarrow \delta\text{Pract}(x,y) \wedge \text{Sport}\ (y) \wedge\neg\text{Athlete}'_1\ (x)$

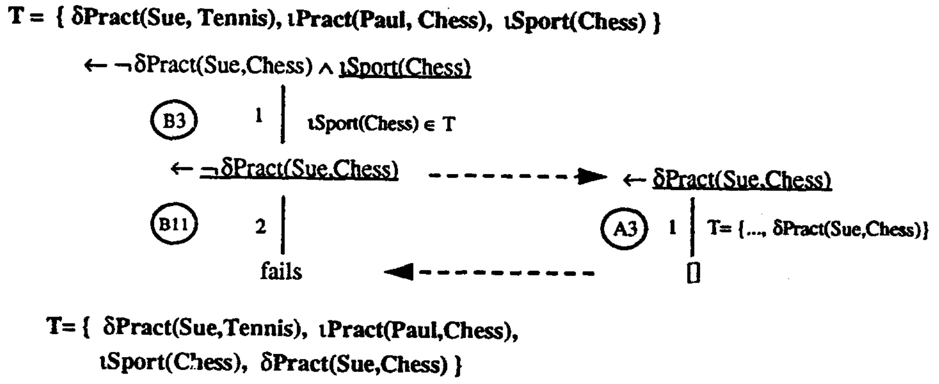
**Figure 4. Successful derivation of request $\leftarrow \delta$ Athlete(Sue) $\wedge$ $\iota$ Athlete(Paul)**


← δAthlete(Sue) ∧ ιAthlete(Paul)
A1
1
(D.2)
←δPract(Sue,y) ∧ Sport(y) ∧ ¬ Athlete'₁ (Sue) ∧ ιAthlete(Paul)
A1
2
(F.3)
←δPract(Sue,Tennis) ∧ ¬ Athlete'₁ (Sue) ∧ ιAthlete(Paul)
A3
3
T= {δPract(Sue,Tennis)}
← ¬Athlete'₁ (Sue) ∧ ιAthlete(Paul)
A7
4
C = {←ιPract(Sue,y)∧ιSport(y),
←ιPract(Sue,Tennis)∧¬ δSport(Tennis),
← ¬ δPract(Sue,Chess)∧ιSport(Chess)}
← ιAthlete(Paul)
A1
5
(I.3)
← Athlete₁,₄ (Paul) ∧ ¬Athlete(Paul)
A5
6
{← Athlete(Paul) fails}
← Athlete'₁,₄ (Paul)
A4
7
(T.1)
← ιPract(Paul,y) ∧ ιSport(y)
A4
8
y = Chess
T= {..., ιPract(Paul,Chess)}
← ιSport(Chess)
A3
9
[]

**T= { δPract(Sue,Tennis), ιPract(Paul,Chess), ιSport(Chess), δPract(Sue,Chess) }**

Now, let the request be the deletion of Athlete(Sue) and the insertion of Athlete(Paul). Translations that satisfy this request are obtained by having a failed SLDNF derivation of $A(K) \cup \{\leftarrow \delta\text{Athlete(Sue)} \wedge \iota\text{Athlete(Paul)}\}$ succeed. One of these derivations is shown in Figure 4. Steps 1 to 7 correspond to steps already described in previous examples. Subsidiary derivation associated to the literal $\neg\text{Athlete}'_1\text{(Sue)}$ (step 4) is not shown in Figure 4, but it implies the inclusion of three conditions: $\leftarrow \iota\text{Pract(Sue,}y) \wedge \iota\text{Sport(}y)$, $\leftarrow \iota\text{Pract(Sue,Tennis)} \wedge \neg\delta\text{Sport(Tennis)}$ and $\leftarrow \neg\delta\text{Pract(Sue,Chess)} \wedge \iota\text{Sport(Chess)}$ into $C$.

**Figure 5. Ensuring satisfaction of condition set**


T = { δPract(Sue, Tennis), ιPract(Paul, Chess), ιSport(Chess) }
← ¬δPract(Sue,Chess) ∧ ιSport(Chess)
B3
1
ιSport(Chess) ∈ T
← ¬δPract(Sue,Chess)
← δPract(Sue,Chess)
B11
2
fails
A3
1
T= {..., δPract(Sue,Chess)}
□
T= { δPract(Sue,Tennis), ιPract(Paul,Chess), ιSport(Chess), δPract(Sue,Chess) }

After step 7 we get the goal $\leftarrow$ $\iota$Pract(Paul,$y$) $\wedge\iota$Sport($y$). This goal can be succeeded if we instantiate the variable $y$ to some value $Y$, for which $\iota$Pract(Paul,$Y$) and $\iota$Sport($Y$) are true. This can be done by asking the user or by assigning default values. In this case, we assume that the value given is "Chess." Thus, in step 8 the event $\iota$Pract(Paul,Chess) is included in the translation set $T$.

At step 9, the selected literal $\iota$Sport(Chess) must be added to $T$. However, we have to verify that this inclusion satisfies $C$ = $\{\leftarrow$ $\iota$Pract(Sue,$y$) $\wedge\iota$Sport($y$), $\leftarrow$ $\iota$Pract(Sue,Tennis) $\wedge\neg\delta$Sport(Tennis), $\leftarrow$ $\neg\delta$Pract(Sue,Chess) $\wedge\iota$Sport(Chess)$\}$. This verification is performed by ensuring that, for each condition $C_i$ of $C$, the SLDNF search space of $A$ $(K)$ $\cup$ $T_i$ fails finitely, where $T_i$ is the subset of $T$ already determined. In some cases, this may cause the addition of new elements to $T$ and/or to $C$.

In Figure 5, we show the necessity of adding new base events to $T$ to maintain satisfaction of the condition $\leftarrow$ $\neg\delta$Pract(Sue,Chess) $\wedge\iota$Sport(Chess). Consistency of the other conditions is not affected by the inclusion of $\iota$Sport(Chess) into $T$. Step 1 is an SLDNF resolution step. At step 2, the selected literal is $\neg\delta$Pract(Sue,Chess). This derivation will fail if it is possible to succeed the subsidiary tree associated to the negation of this literal. This derivation is shown in the right part of Figure 5, and it causes the inclusion of $\delta$Pract(Sue,Chess) in the translation set $T$.

We get the empty clause in the initial derivation of Figure 4 and, hence, the translation process finishes, obtaining the translation $T$=$\{\delta$Pract(Sue,Tennis),$\iota$Pract(Paul, Chess),$\iota$Sport(Chess),$\delta$Pract(Sue,Chess)$\}$. Notice that the addition of $\iota$Sport(Chess) to $T$ in step 9 of Figure 4 is contradictory with the deletion of the view fact Athlete(Sue). The existence of the condition set $C$ allows us to detect this contradiction (through the violation of condition $\leftarrow$ $\neg\delta$Pract(Sue,Chess) $\wedge\iota$Sport(Chess)), and to perform necessary repair actions (which motivates the inclusion of $\delta$Pract(Sue,Chess) in $T$).

Selecting rule D.2 in step 1 of Figure 4, we would obtain another translation $T_2$ = {$\delta$Sport(Tennis),$\iota$Pract(Paul,Chess),$\iota$Sport(Chess), $\delta$Pract(Sue,Chess)}. This derivation is not shown in Figure 4.

## 4.4 Formalization of Events Method

In this section, we give a formal definition of our method for obtaining all minimal translations that satisfy a view update request. We will formalize it for the general case of knowledge bases, where the only condition that deductive rules must satisfy is allowedness.

As shown in previous examples, the Events Method is an interleaving of two activities: (1) satisfying an update request by including base events in the translation set, and (2) checking if the view updates induced by these base events are contradictory with the requested update. These two activities are performed during constructive and consistency derivations, respectively, as defined below.

Let $u$ be an update request. In our method, $T$ will be a translation of $u$ if there is a constructive derivation from ($\leftarrow u\ \emptyset\ \emptyset$) to ([] $T\ C$). Base events contained in the translation set $T$ correspond to the updates (insertions and/or deletions) of base facts that must be performed on the extensional part of the knowledge base to satisfy the view update request. To obtain all possible translations that satisfy an update request $u$, we have to consider all possible constructive derivations. We will see in Section 6 that, when no such derivation exists, the requested update cannot be satisfied by changing only the extensional part.

Predicates appearing in $A$ ($K$) may be (we include examples from the previous section):

- old base predicates (from the old state of the knowledge base): Pract,Sport
- old derived predicates (from the old state of the knowledge base): Athlete
- base events: $\delta$Pract, $\iota$Pract, $\delta$Sport, $\iota$Sport
- derived events: $\delta$Athlete, $\iota$Athlete
- new predicates (from the new state of the knowledge base): Athlete$'_1$, Athlete$'_{1,3}$

The rules applied in constructive and consistency derivations depend on the type of the selected literal. In Figure 6, we summarize which rule is applied in each case.

*Constructive Derivation.* A constructive derivation from ($G_1\ T_1\ C_1$) to ($G_n\ T_n\ C_n$) via a safe computation rule $R$ (Lloyd, 1987) is a sequence:

($G_1\ T_1\ C_1$), ($G_2\ T_2\ C_2$) ... ($G_n\ T_n\ C_n$)

such that for each $i \geq 1$, $G_i$ has the form $\leftarrow L_1 \wedge \ldots \wedge L_k$, $R(G_i) = L_j$ and ($G_{i+1}$ $T_{i+1}\ C_{i+1}$) is obtained according to one of the following rules:

**Figure 6. Rules applied in derivation**

| *Constructive* | Positive | Negative |
|---|---|---|
| Old base predicate | | |
| Old derived predicate | A1 | A5 |
| Base event | A2,A3,A4 | A6 |
| Derived event | | |
| New predicate | A1 | A7 |

| *Consistency* | Positive | Negative |
|---|---|---|
| Old base predicate | | |
| Old derived predicate | B1,B2 | B7,B8 |
| Base event | B3,B4,B5,B6 | B9,B10,B11 |
| Derived event | | |
| New predicate | B1,B2 | B12,B13 |

A1. If $L_j$ is positive, it is not a base event and $S$ is the resolvent of some clause in $A(K)$ with $G_i$ on the selected literal $L_j$, then $G_{i+1}=S$, $T_{i+1}=T_i$, and $C_{i+1}=C_i$.

A2. If $L_j$ is a ground positive base event, and $L_j \in T_i$, then $G_{i+1}=G_i/L_j$, $T_{i+1}=T_i$, and $C_{i+1}=C_i$.

A3. If $L_j$ is a ground positive base event $\iota P$ (respectively, $\delta P$), $L_j \notin T_i$, $P$ does not hold (respectively, $P$ holds) in the current knowledge base, and there is a consistency derivation from $(C_i\ T_i \cup \{L_j\}\ C_i)$ to $(\{\}\ T'\ C')$, then $G_{i+1}=G_i/L_j$, $T_{i+1}=T'$, and $C_{i+1}=C'$.

If $C_i=\emptyset$, then $G_{i+1}=G_i/L_j$, $T_{i+1}=T_i \cup \{L_j\}$, and $C_{i+1}=C_i$.

A4. If $L_j$ is a non-ground positive base event, $\sigma$ is a substitution such that:

(a) If $L_j$ is a non-ground positive base event $\iota P$, then $\sigma$ is a substitution of variables of $P$ such that $P\sigma$ does not hold in the current knowledge base.

(b) If $L_j$ is a non-ground positive base event $\delta P$, then $\sigma$ is a substitution of variables of $P$ such that $P\sigma$ holds in the current knowledge base.

and there is a consistency derivation from $(C_i\ T_i \cup \{L_j\sigma\}\ C_i)$ to $(\{\}\ T'\ C')$, then $G_{i+1}=G_i\sigma/L_j\sigma$, $T_{i+1}=T'$, and $C_{i+1}=C'$.

If $C_i=\emptyset$, then $G_{i+1}=G_i/L_j$, $T_{i+1}=T_i \cup \{L_j\sigma\}$, and $C_{i+1}=C_i$.

A5. If $L_j$ is a base or derived predicate, negative and old and, using SLDNF resolution, the goal $\leftarrow \neg L_j$ fails finitely, then $G_{i+1}=G_i/L_j$, $T_{i+1}=T_i$, and $C_{i+1}=C_i$.

A6. If $L_j$ is a ground-negative base event, and $\neg L_j \notin T_i$, then $G_{i+1}=G_i/L_j$, $T_{i+1}=T_i$, and $C_{i+1} = C_i \cup \{\leftarrow \neg L_j\}$.

A7. If $L_j$ is a negative, new, or derived event predicate, and there is a consistency derivation from $(\{\leftarrow \neg L_j\}\ T_i\ C_i)$ to $(\{\}\ T'\ C')$, then $G_{i+1}=G_i/L_j$, $T_{i+1}=T'$, and $C_{i+1}=C'$.

Rules A1, A2, and A5 are SLDNF resolution steps where $A\,(K)$ or $T$ act as input set.

In rule A3, the selected base event is included in the translation set $T_i$, to get a successful derivation for the current branch, provided that we can ensure that this will not violate the consistency of any condition in $C_i$. Note that if $C_i=\emptyset$, consistency derivations must not be performed because there is no condition to satisfy.

In rule A4, a non-fully-ground positive base event is selected, and we have to instantiate it (substeps (a) and (b)). In (a), this is done either by assigning default values or by asking the user. In (b), there will be as many alternatives as facts of the knowledge base can be unified with $P$. Once the base event is fully instantiated, we proceed in the same way as in rule A3.

In rule A6, selected base events are added to the condition set to ensure that they will not be included in the translation set afterwards. In rule A7, we achieve the next goal if we can ensure consistency for the selected literal.

*Consistency Derivation.* A consistency derivation from $(F_1\ T_1\ C_1)$ to $(F_n\ T_n\ C_n)$ via a safe computation rule $R$ is a sequence:

$(F_1\ T_1\ C_1)$, $(F_2\ T_2\ C_2)$, ..., $(F_n\ T_n\ C_n)$

such that for each $i \geq 1$, $F_i$ has the form $H_i \cup F'_i$, where $H_i = \leftarrow L_1 \wedge \ldots \wedge L_k$ and, for some $j=1\ldots k$, $(F_{i+1}\ T_{i+1}\ C_{i+1})$ is obtained according to one of the following rules:

B1. If $L_j$ is positive, it is not a base event; $S'$ is the set of all resolvents of clauses in $A\,(K)$ with $H_i$ on the literal $L_j$ and $[] \notin S'$, then $F_{i+1}=S' \cup F'_i$, $T_{i+1}=T_i$, and $C_{i+1}=C_i$.

B2. If $L_j$ is positive, it is not a base event, and there is no input clause in $A\,(K)$ that can be unified with $L_j$, then $F_{i+1}=F'_i$, $T_{i+1}=T_i$, and $C_{i+1}=C_i$.

B3. If $L_j$ is a ground positive base event, $L_j \in T_i$ and $k>1$, then $F_{i+1}=H_i/L_j \cup F'_i$, $T_{i+1}=T_i$, and $C_{i+1}=C_i$.

B4. If $L_j$ is a ground positive base event and $L_j \notin T_i$, then $F_{i+1}=F'_i$, $T_{i+1}=T_i$, and $C_{i+1} = C_i \cup \{H_i\}$.

B5. If $L_j$ is a non-ground positive base event, $S'$ is the set of all resolvents of clauses in $T_i$ with $H_i$ on the literal $L_j$ and $[] \notin S'$, then $F_{i+1}=S' \cup F'_i$, $T_{i+1}=T_i$, and $C_{i+1}=C_i \cup \{H_i\}$.

B6. If $L_j$ is a non-ground positive base event, and no input clause in $T_i$ can be unified with $L_j$, then $F_{i+1}=F'_i$, $T_{i+1}=T_i$, and $C_{i+1}=C_i \cup \{H_i\}$.

B7. If $L_j$ is a base or derived predicate, negative and old, $k$>1 and using SLDNF resolution as the goal $\leftarrow \neg L_j$ fails finitely, then $F_{i+1} = H_i/L_j \cup F'_i$, $T_{i+1} = T_i$, and $C_{i+1} = C_i$.

B8. If $L_j$ is a base or derived predicate, negative and old, and there is an SLDNF refutation of $A(K) \cup \{\leftarrow \neg L_j\}$, then $F_{i+1} = F'_i$, $T_{i+1} = T_i$, and $C_{i+1} = C_i$.

B9. If $L_j$ is a ground negative base event and $\neg L_j \in T_i$, then $F_{i+1} = F'_i$, $T_{i+1} = T_i$, and $C_{i+1} = C_i$.

B10. If $L_j$ is a ground negative base event, $L_j \notin T_i$, and $k$>1, then $F_{i+1} = H_i/L_j \cup F'_i$, $T_{i+1} = T_i$, and $C_{i+1} = C_i$.

B11. If $L_j$ is a ground negative base event, $\neg L_j \notin T_i$, and there is a constructive derivation from $(\{\leftarrow \neg L_j\}\ T_i\ C_i)$ to $([]\ T'\ C')$, then $F_{i+1} = F'_i$, $T_{i+1} = T'$, and $C_{i+1} = C'$.

B12. If $L_j$ is a negative, new, or derived event, predicate, $k$>1, and there is a consistency derivation from $(\{\leftarrow \neg L_j\}\ T_i\ C_i)$ to $(\{\}\ T'\ C')$, then $F_{i+1} = H_i/L_j \cup F'_i$, $T_{i+1} = T'$, and $C_{i+1} = C'$.

B13. If $L_j$ is a negative, new, or derived event, predicate, and there is a constructive derivation from $(\{\leftarrow \neg L_j\}\ T_i\ C_i)$ to $([]\ T'\ C')$, then $F_{i+1} = F'_i$, $T_{i+1} = T'$, and $C_{i+1} = C'$.

Rules B1, B2, B3, B7, B8, B9, and B10 are SLDNF resolution steps where $A(K)$ or $T$ act as input set.

In rules B4 and B6, the current branch is dropped from the consistency derivation because the subset of $T$ already determined ensures failure for it. Moreover, the current goal $H_i$ must be included in the condition set $C_i$ to guarantee that later additions to $T_i$ will not make this branch succeed.

In rule B5, the selected literal can be unified with one or more base events in $T$. We must then verify that each resulting branch fails.

In rule B11, the current branch is dropped if there is a constructive derivation for the negation of the selected literal. In rules B13 and B12, the current branch may be dropped, depending on whether there is a constructive or consistency derivation for the negation of the selected literal.

Consistency derivations do not rely on the particular order in which selection rule $R$ selects literals since, in general, all possible ways in which a conjunction $\leftarrow L_1 \wedge \ldots \wedge L_k$ can fail should be explored. Each one may lead to a different translation.

Our method obtains all minimal translations that satisfy a given update request (Section 6). However, as we have seen in *Example 6,* in some cases we may also obtain translations that are a superset of the minimal ones. These translations are then refused because they do not satisfy our criterion of minimality.

## 5. Integrity Constraints Satisfaction

There is a close relationship between updating a knowledge base and integrity constraints satisfaction because, in general, consistency of a knowledge base can only be violated when performing an update. Translations of a view update request correspond to base updates. Then, some translations could be invalidated because they violate an integrity constraint. On the other hand, any mechanism that restores consistency needs to solve the view update problem when derived predicates may appear in some integrity constraint. For these reasons, it becomes necessary to combine view updating and integrity constraints satisfaction.

As we mentioned in the introduction, this combination can be performed in two different ways. The first possibility consists of combining *view updating* and *integrity constraints maintenance*. In this case, new insertions and/or deletions of base facts are added to the view update translations that violate some integrity constraints to restore knowledge base consistency. Then, we obtain translations that satisfy the view update request and all the integrity constraints of the knowledge base. The result of the combined process is a set (possibly empty) of translations that do not necessarily correspond to a subset of the translations obtained by view updating alone (Section 5.1).

The second possibility consists of combining view updating and integrity checking. In this case, we would first obtain all possible translations that satisfy a view update, and then check whether they satisfy the integrity constraints. The result of the combined process is the subset of translations obtained by view updating that would leave the knowledge base consistent (Section 5.2).

### 5.1 Integrity Constraints Maintenance

In the Events Method, a translation $T$ corresponding to an update request $u$ violates an integrity constraint $Icn$ if $T$ includes some base events such that some fact $\iota Icn$ becomes true in the transition from the old state to the new state of the knowledge base. Therefore, if we are only interested in those translations that do not violate $Icn$, we only have to use $\{\leftarrow u \wedge \neg \iota Icn\}$ as root goal.

In general, if there are $n$ integrity constraints, we define the auxiliary predicate $\iota Ic$ as: $\iota Ic \leftarrow \iota Ic1(\mathrm{x}_1)$, ..., $\iota Ic \leftarrow \iota Icn(\mathrm{x}_n)$, (where $\mathrm{x}_i, i{=}1 \ldots n$, is a vector of terms), and use the goal $\{\leftarrow u \wedge \neg \iota Ic\}$ to obtain all the translations that satisfy $u$ while maintaining knowledge base consistency.

In our method, $T$ will be one of these translations if there is a constructive derivation (as formally defined in Section 4.4) from $(\{\leftarrow u \wedge \neg \iota Ic\}\ \emptyset\ \emptyset)$ to $([]\ T C)$. During this derivation, the literal $\neg \iota Ic$ will be selected. Because it corresponds to a negative derived event predicate (Constructive Derivation Rule 7), the next step of the constructive derivation will be reached if there is a consistency derivation from $(\{\leftarrow \iota Ic\}\ T'\ C')$ to $(\{\}\ T C)$, where $T'$ and $C'$ are the subsets of $T$ and $C$, which are already determined when the literal $\neg \iota Ic$ is selected. The existence of this consistency derivation guarantees that no integrity constraint will be violated.

In this way, integrity maintenance is included as a part of the existing consistency derivation.

*Example 8:* Consider again the knowledge base defined in *Example 7,* and assume that it also contains the following facts and integrity constraint (which states that Ron must practice Swimming or Climbing):

F.4 Pract (Ron,Swimming)
F.5 Sport (Swimming)
F.6 Sport(Climbing)
IC.1 $Ic1 \leftarrow \neg$Pract(Ron,Swimming) $\wedge\neg$Pract(Ron,Climbing)

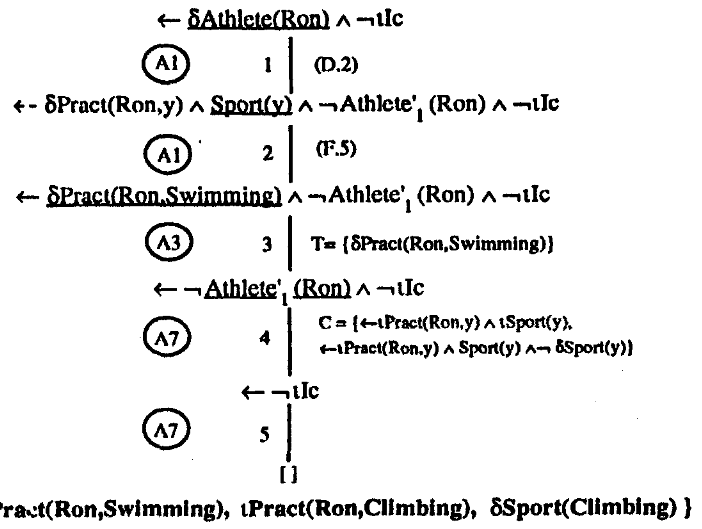
Transition and event rules associated with the inconsistency predicate are:

| | | | | |
|---|---|---|---|---|
| T.9 | $Ic1'_{1,1}$ | $\leftarrow$ | $\neg$Pract(Ron,Swimming) $\wedge\neg\iota$Pract(Ron,Swimming) $\wedge\neg$Pract(Ron,Climbing) $\wedge\neg\iota$Pract(Ron,Climbing) | |
| T.10 | $Ic1'_{1,2}$ | $\leftarrow$ | $\neg$Pract(Ron,Swimming) $\wedge\neg\iota$Pract(Ron,Swimming) $\wedge\delta$Pract(Ron,Climbing) | |
| T.11 | $Ic1'_{1,3}$ | $\leftarrow$ | $\delta$Pract(Ron,Swimming) $\wedge\neg$Pract(Ron,Climbing) $\wedge\neg\iota$Pract(Ron,Climbing) | |
| T.12 | $Ic1'_{1,4}$ | $\leftarrow$ | $\delta$Pract(Ron,Swimming) $\wedge\delta$Pract(Ron,Climbing) | |
| I.4..6 | $\iota Ic1$ | $\leftarrow$ | $Ic1'_{1,j}$ | $j = 2...4$ |

Now, assume that we want to delete the derived fact that Ron is an athlete following the integrity maintenance approach. A translation $T$ will satisfy this request if there is a constructive derivation from ($\{\leftarrow \delta$Athlete(Ron) $\wedge\neg\iota Ic\}$ $\emptyset$ $\emptyset$) to ([] $T$ $C$). Part of this derivation is shown in Figure 7.

Steps 1 and 2 are SLDNF resolution steps. At step 3, the selected literal $\delta$Pract(Ron,Swimming) is included in the translation set $T$. At step 4, there is a consistency derivation associated with the goal $\leftarrow$Athlete$'_1$(Ron). This derivation is not shown in Figure 7, but it implies the inclusion of two conditions: $\leftarrow$ $\iota$Pract(Ron,$y$) $\wedge\iota$Sport($y$), $\leftarrow$ $\iota$Pract(Ron,$y$) $\wedge$Sport($y$) $\wedge\neg\delta$Sport($y$) into the condition set $C$. At step 5, the next goal is reached because there is a consistency derivation associated to $\leftarrow$ $\iota Ic$. Part of this derivation is shown in Figure 8. Steps 1 to 5 are SLDNF resolution steps. At step 6, this branch can be removed because there is a constructive derivation for the negation of the selected literal. This derivation is shown in the right part of Figure 8. At step 1 of this constructive derivation, $\iota$Pract(Ron,Climbing) is included in $T$. Note that this inclusion is necessary to not violate $Ic1$. Moreover, we have to verify that this inclusion satisfies $C = \{\leftarrow \iota$Pract(Ron,$y$) $\wedge\iota$Sport($y$), $\leftarrow$ $\iota$Pract(Ron,$y$) $\wedge$ Sport($y$) $\wedge\neg\delta$Sport($y$)$\}$. In this case, the event $\delta$Sport(Climbing) must be included in $T$ to maintain consistency of the second condition (this is done in a way similar to Figure 5).

**Figure 7. Constructive derivation of $\leftarrow \delta$ Athlete(Ron) $\wedge \neg\iota$ Ic**


← δAthlete(Ron) ∧ ¬ιIc
A1
1
(D.2)
← δPract(Ron,y) ∧ Sport(y) ∧ ¬Athlete'₁ (Ron) ∧ ¬ιIc
A1'
2
(F.5)
← δPract(Ron,Swimming) ∧ ¬Athlete'₁ (Ron) ∧ ¬ιIc
A3
3
T= {δPract(Ron,Swimming)}
← ¬Athlete'₁ (Ron) ∧ ¬ιIc
A7
4
C = {←ιPract(Ron,y) ∧ ιSport(y), ←ιPract(Ron,y) ∧ Sport(y) ∧ ¬ δSport(y)}
← ¬ιIc
A7
5
[]
T= { δPract(Ron,Swimming), ιPract(Ron,Climbing), δSport(Climbing) }

Selecting clauses I.4 and I.6 at step 2 of the previous consistency derivation, we get failed derivations that do not modify the translation nor the condition sets (these derivations are not shown in Figure 8).

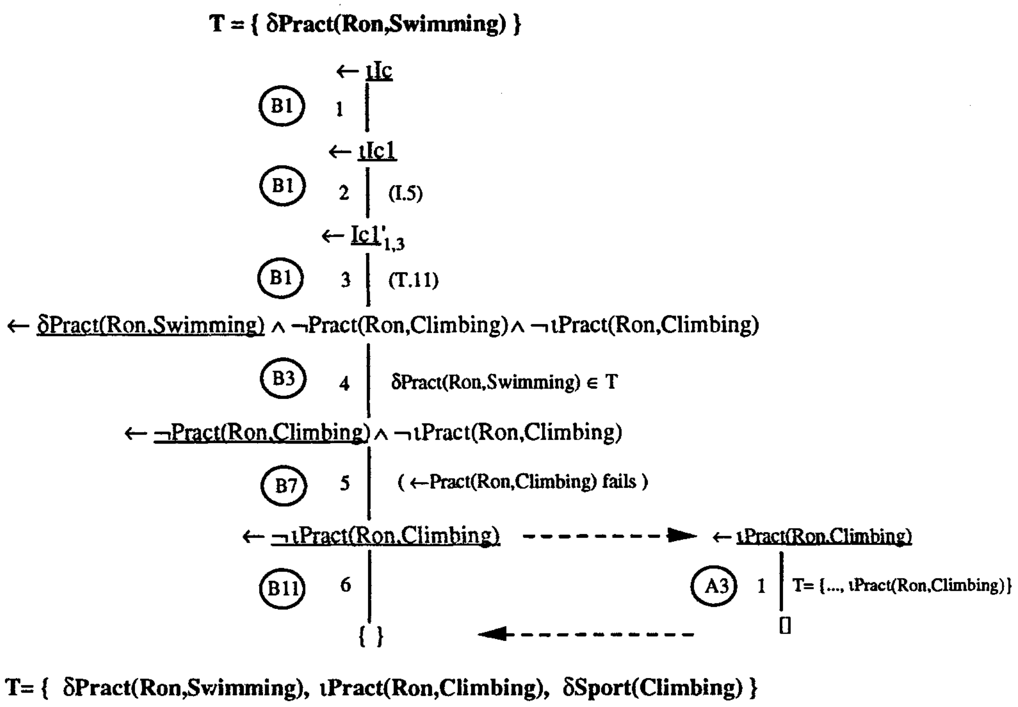
Then, in the initial derivation of Figure 7, we get the empty clause and the derivation is finished obtaining $T = \{\delta$Pract(Ron,Swimming), $\iota$Pract(Ron,Climbing), $\delta$Sport(Climbing)$\}$, which satisfies the view update request, and maintains knowledge base consistency. Selecting the clause D.1 at step 1 of Figure 7, we obtain another solution $T_2 = \{\delta$Sport(Swimming)$\}$.

As we have shown in the previous example, integrity constraints maintenance is incorporated into the translation process in our method. Thus, in a single step, we obtain translations that satisfy the view update request and all the integrity constraints.

It might seem that view updating and integrity maintenance also could be performed in two separate steps. In the first step, we would obtain the translations that satisfy the view update request and, in the second step, integrity constraints would be maintained. This approach, however, does not work since the result of view updating is not only a set of translations, but also a set of conditions that must be false during the transition. If one does not take into account this latter set, the resulting translations might be incorrect.

As an example, consider again the same knowledge base and update request as in *Example 8.* In the first step, we would translate the view update request into base updates as explained in Section 4, obtaining two translations: $T_1$ =

**Figure 8. Consistency derivation of ιIc**


T = { δPract(Ron,Swimming) }
← ιIc
B1 1
← ιIc1
B1 2 (I.5)
← Ic1'1,3
B1 3 (T.11)
← δPract(Ron,Swimming) ∧ ¬Pract(Ron,Climbing) ∧ ¬ιPract(Ron,Climbing)
B3 4 δPract(Ron,Swimming) ∈ T
← ¬Pract(Ron,Climbing) ∧ ¬ιPract(Ron,Climbing)
B7 5 ( ←Pract(Ron,Climbing) fails )
← ¬ιPract(Ron,Climbing)
← ιPract(Ron,Climbing)
B11 6
A3 1 T= {..., ιPract(Ron,Climbing)}
{ }
□
T= { δPract(Ron,Swimming), ιPract(Ron,Climbing), δSport(Climbing) }

{δPract(Ron,Swimming)} and $T_2$ = {δSport(Swimming)}. In the second step, we would repair the translations that do not satisfy some integrity constraint by adding new insertions and/or deletions of base facts to them. In this example, we would finally obtain $T_1$ = {δPract(Ron,Swimming), ιPract(Ron,Climbing)}, and $T_2$ = {δSport(Swimming)}. Notice that $T_1$ does not satisfy the original view update request and, then, it is not a valid solution. It lacks δSport(Climbing), which can only be discovered if one takes into account that the condition ← ιPract(Ron,$y$) ∧ Sport($y$) ∧¬δSport($y$) must fail.

Integrity maintenance may also be applied when considering base updates, that is, insertions and/or deletions of base facts. In this case, additional updates for restoring knowledge base consistency might be necessary. These additional updates can be obtained in the Events Method in the same way as defined above.

## 5.2 Integrity Constraints Checking

Our method also can be used for combining view updating and integrity checking. In this case, we should first obtain all minimal translations that satisfy the update request (by applying the procedure defined in Section 4.4) and, afterwards, we would reject the solutions that violate some integrity constraint. This second step could be performed by applying the method presented by Olivé (1991) (which is also based on the concept of event) or any other method for integrity constraints checking.

As it has been pointed out recently (Ceri and Widom, 1990; Moerkotte and Lockemann, 1991), this approach may not be satisfactory because rejecting the translations that violate some integrity constraint may leave the user at a loss to the potential causes of the integrity violation and, hence, with few clues as to the needed changes to the transaction. Moreover, we see another important advantage of the integrity maintenance approach over integrity checking because, in some cases, translations that would not be obtained in the latter approach are found in integrity maintenance.

As an example, consider again the same knowledge base and view update request as before. In the first step, we would translate the view update request into base updates, obtaining two translations: $T_1 = \{\delta\text{Pract(Ron,Swimming)}\}$ and $T_2 = \{\delta\text{Sport(Swimming)}\}$. In the second step, we must check if the obtained translations satisfy the integrity constraints. It is not difficult to see that translation $T_1$ violates $Ic1$ because, if it were applied, Ron would not practice Swimming or Climbing. Thus, this translation would be rejected and the only solution $T_2 = \{\delta\text{Sport(Swimming)}\}$ would be obtained following the integrity constraints checking approach.

To finish this section, we would like to point out that *transition integrity constraints* can also be maintained in our method. These constraints involve two consecutive knowledge base states. That is, they are constraints that knowledge base transitions must satisfy. A typical example could be a constraint stating that salaries can not decrease (see Olivé, 1991 for the details of transition and event rules in this case).

## 6. Soundness and Completeness of Events Method

We have proved that the Events Method is sound and complete for all cases when SLDNF resolution is sound and complete. In Appendix B we see that, up to now, it has been proved that SLDNF resolution is complete when $A(K)$ is allowed, $A(K) \cup T$ is call-consistent, and $A(K) \cup T \cup \{\leftarrow u\}$ is even. The corresponding properties on $K$ are that $K$ is allowed and stratified (or call-consistent).

In this section, we present the main theorems for soundness and completeness of the Events Method. The technical proofs are omitted from this article due to lack of space, and can be found in Teniente and Olivé (1994).

### 6.1 Soundness

The Events Method is sound in the sense that, given a knowledge base $K$ and its associated augmented knowledge base $A(K)$, if the method obtains a translation $T$ for an update request $u$, then the application of $T$ to $K$ (i.e., inserting and/or deleting the base facts included in $T$) leaves the knowledge base in a state $K'$ such that $u$ holds in $K'$. Soundness of the Events Method is based on the following Lemma:

*Lemma 1:* Let $K$ be a knowledge base, $A(K)$ the augmented knowledge base, $u$ an update request, and $T$ a translation obtained by the Events Method. Then, there is an SLDNF refutation of $A(K) \cup T \cup \{\leftarrow u\}$.

As can be seen, the lemma relates the constructive derivation $(\{\leftarrow u\}\ \emptyset\ \emptyset)$ to $([]\ T\ C)$ of our method to an SLDNF refutation of $A(K) \cup T \cup \{\leftarrow u\}$. Given that SLDNF resolution has been proven sound (Clark, 1978), then the following theorem follows:

*Theorem 1* (Soundness of the Events Method): Let $K$ be a knowledge base, $A(K)$ the augmented knowledge base, and $u$ an update request such that $u$ is not a logical consequence of comp($A(K)$). Let $T$ be a translation obtained by the Events Method. Then, $u$ is a logical consequence of comp($A(K) \cup T$).

## 6.2 Completeness

The above relationship between a constructive derivation of our method and an SLDNF refutation also exists in the reverse direction. This is stated in the following theorem:

*Theorem 2:* Let $K$ be a knowledge base, $A(K)$ the augmented knowledge base, $u$ an update request, and $T$ a minimal set of base events such that, using SLDNF resolution, a refutation of $A(K) \cup T \cup \{\leftarrow u\}$ is given. Then, there is a constructive derivation from $(\{\leftarrow u\}\ \emptyset\ \emptyset)$ to $([]\ T\ C)$.

Therefore, for all cases when SLDNF-resolution is complete, we have the following completeness result for our method:

*Theorem 3* (Completeness of the Events Method): Let $K$ be a knowledge base, $A(K)$ the augmented knowledge base, $u$ an update request, and T a minimal translation. Suppose that SLDNF resolution is complete for $A(K) \cup T$ and for goal $\{\leftarrow u\}$. Then, there is a constructive derivation from $(\leftarrow u\ \emptyset\ \emptyset)$ to $([]\ T\ C)$ for any translation $T$ that satisfies that $u$ is a logical consequence of comp($A(K) \cup T$).

From theorems of soundness and completeness, we can deduce two important conclusions. Let $u$ be an update request. Soundness of the Events Method ensures that if there is a constructive derivation from $(\leftarrow u\ \emptyset\ \emptyset)$ to $([]\ T\ C)$, then the knowledge base updated according to $T$ satisfies the request. Furthermore, completeness of the Events Method ensures that if the constructive derivation rooted at the view update fails finitely, then the request cannot be satisfied by changing only the extensional part of the knowledge base. In Appendix B we review the cases for which completeness results of SLDNF-resolution have been proven.

## 7. Other Kinds of Updates

In the previous sections, we faced the problem of view updating, and discussed how it could be combined with integrity checking and integrity maintenance. We also showed how to apply integrity maintenance when considering updates of base facts. However, knowledge bases allow other kinds of updates (i.e., updates of deductive or integrity rules), which also may violate knowledge base consistency.

The main goal of this section is to present an extension of the Events Method for maintaining knowledge base consistency when updating deductive or integrity rules. In general, there are several ways for maintaining integrity constraints. Our approach consists of generating all possible minimal solutions.

## 7.1 Insertions or Deletions of Deductive Rules

We consider first the case of inserting a new deductive rule:

$$P(x) \leftarrow L_1(x) \wedge ... \wedge L_n(x)$$

Due to this deductive rule, the updated knowledge base is likely to contain some new (implicit) $P$ facts which might violate some integrity constraint. Such violations can also be detected and repaired in our method. What needs to be done is determining which $\iota P$ facts are produced in the transition from the old state of the knowledge base to the new, updated state. Once known, integrity constraints can be maintained as usual.

If $P$ is a new predicate in the knowledge base, the insertion events produced by the update are given by $\iota P(\mathrm{x}) \leftarrow P'(\mathrm{x})$. As indicated in Section 3, transforming $P'(\mathrm{x})$ into its equivalent set of rules, we get the insertion event rules that give the $\iota P$ facts produced during the transition. Note that, in this particular case, we would not need to maintain consistency since, $P$ being new, no integrity constraint can be violated. However, when this kind of update occurs as part of a larger update request, it may be necessary to maintain it.

If $P$ is an existing predicate, we first rename $P$ in the conclusion of the deductive rule by $P_k$, with:

$k = 1$ if $P$ is a base predicate
$k = m+1$ if $P$ is a derived predicate with $m$ rules

Then, $\iota P$ facts produced by the update are given by : $\iota P(\mathrm{x}) \leftarrow P'_k(\mathrm{x}) \wedge \neg P(\mathrm{x})$, and transforming $P'_k(\mathrm{x})$ into its equivalent set of transition rules we get the insertion event rules that give the $\iota P$ facts produced during the transition.

Once this transformation has been done, we can apply the Events Method in the usual way. Let $u$ be an insertion of a deductive rule. $T$ will be a solution if there is a constructive derivation from $(\{\leftarrow \neg\iota Ic\}\ \emptyset\ \emptyset)$ to $([]\ T C)$. If $T = \emptyset$, then the deductive rule can be inserted into the knowledge base without performing any additional update. In any case, if some translation is obtained, the new rules are accepted, the knowledge base updated and the transition and event rules modified accordingly. If no solution $T$ is obtained, then it is not possible to insert the deductive rule without violating any integrity constraint by considering only repairs on the extensional part of the knowledge base.

In the case of deleting an existing deductive rule we would proceed in a similar way, but now deriving the deletion event rules that give the input deletion events produced by the update. The following example illustrates our approach.

*Example 9:* Let $K$ be a knowledge base with the following predicates and integrity constraints:

| | |
|---|---|
| Grant ($x$) | $x$ has a grant |
| Cont ($x$) | $x$ has a contract with some company |
| Assig ($x$,$y$) | employee $x$ is assigned to department $y$ |
| Dept ($y$) | $y$ is a department |
| Unemp ($x$) | $x$ is unemployed |
| Works ($x$) | $x$ works |

$Ic1$ ($x$) It is not possible for any $x$ to work and be unemployed at the same time.

The current content of the knowledge base and relevant transition and event rules are:

F.1 Grant(John)

F.2 Unemp(John)

F.3 Assig(Mary, Sales)

DR.1 Works($x$) $\leftarrow$ Cont($x$)

IC.1 $Ic1(x) \leftarrow$ Works($x$) $\wedge$ Unemp($x$)

T.1 $Ic'_{1,3}(x) \leftarrow \iota$Works($x$) $\wedge$ Unemp($x$) $\wedge \neg\delta$Unemp($x$)

I.1 $\iota Ic1(x) \leftarrow Ic1'_{1,3}(x)$

I.2 $\iota Ic1 \leftarrow \iota Ic1(x)$

Let the update be the insertion of the deductive rule:

Works($x$) $\leftarrow$ Grant($x$)

where Grant is a base predicate. Works is an existing derived predicate with one rule ($m = 1$). Then, we must first rename the predicate in the conclusion of the new rule by $\text{Works}_2$. The transition rules associated with this predicate are:

T.2 $\text{Works}'_{2,1}(x) \leftarrow$ Grant($x$) $\wedge \neg\delta$Grant($x$)

T.3 $\text{Works}'_{2,2}(x) \leftarrow \iota$Grant($x$)

The new insertion event rules are:

I.3 $\iota$Works($x$) $\leftarrow \text{Works}'_{2,1}(x) \wedge \neg$Works($x$)

I.4 $\iota$Works($x$) $\leftarrow \text{Works}'_{2,2}(x) \wedge \neg$Works($x$)

Note that the insertion of this deductive rule violates integrity constraint $Ic1$. Some updates of the base facts must be performed to maintain knowledge base consistency. $T$ will be a solution if, with these new rules, there is a constructive derivation from $(\{\leftarrow \neg\iota Ic\}\ \emptyset\ \emptyset)$ to $([]\ T\,C)$. This derivation is shown in Figure 9.

**Figure 9. Constructive derivation of $\leftarrow \neg\iota$ Ic**

$$\begin{tikzcd}
\leftarrow \neg \iota Ic \arrow[d, dash, "1"'] \\
{[\,]}
\end{tikzcd}$$

**Figure 10. Consistency derivation of $\leftarrow \iota$ Ic**

$$\begin{tikzcd}
\leftarrow \underline{\iota Ic} \arrow[d, dash, "{1, 2, 3}"', "{(I.2, I.1, T.1)}"] \\
\leftarrow \underline{\iota Works(x)} \land Unemp(x) \land \neg\, \delta Unemp(x) \arrow[d, dash, "{4, 5}"', "{(I.3, T.2)}"] \\
\leftarrow \underline{Grant(x)} \land \neg\delta\, Grant(x) \land \neg\, Works(x) \land Unemp(x) \land \neg\delta Unemp(x) \arrow[d, dash, "6"', "{(F.1)}"] \\
\leftarrow \neg\,\delta Grant(John) \land \underline{\neg Works(John)} \land \underline{Unemp(John)} \land \neg\, \delta Unemp(John) \arrow[d, dash, "{7, 8}"', "{(\leftarrow Works(John)\ fails, F.2)}"] \\
\leftarrow \neg\delta Grant(John) \land \neg\delta Unemp(John)
\end{tikzcd}$$

$$\mathbf{T_1 = \{\ \delta Grant(John)\ \} \qquad T_2 = \{\ \delta Unemp(John)\ \}}$$

The empty clause is reached because there is a consistency derivation associated to $\leftarrow \iota Ic$. Part of this derivation is shown in Figure 10. To save space, we sometimes group two or more steps into a single one, as in I.2, I.1 and T.1. Steps 1 to 8 are SLDNF resolution steps. After this step, all possible ways in which the goal $\leftarrow \neg\delta$Grant(John) $\land\neg\delta$Unemp(John) can fail must be studied. In this case, we will obtain two solutions: $T_1 = \{\delta\text{Grant(John)}\}$ and $T_2 = \{\delta\text{Unemp(John)}\}$. Notice that both solutions maintain knowledge base consistency when inserting the rule Works($x$) $\leftarrow$ Grant($x$).

## 7.2 Insertions or Deletions of Integrity Constraints

We deal with insertions of integrity constraints as with insertions of deductive rules for new predicates (see previous section). The deletion of an integrity constraint cannot violate knowledge base consistency and, therefore, there is no need to maintain it in this case. As an example, consider the same knowledge base of *Example 9,* and assume that its current content is exactly the same. Let the update request be the insertion of the integrity constraint: $Ic_2(\mathbf{x},\mathbf{y}) \leftarrow$ Assig(x,y) $\wedge\neg$Dept($y$). In our method, $T$ will be a solution if, considering the transition and event rules associated with $Ic_2$, there is a constructive derivation from $(\{\leftarrow \neg\iota Ic\}\ \emptyset\,\emptyset)$ to ([] $T\,C$). In this case, our method obtains two different solutions: $T_1 = \{\delta$Assig(Mary,Sales)$\}$ and $T_2 = \{\iota$Dept(Sales)$\}$.

## 7.3 Transactions With Multiple Updates

When a transaction consists of several kinds of updates (i.e., view updates, updates of base facts, deductive and/or integrity rules), we first determine the input events produced by each update, and then apply the integrity maintenance approach (Sections 5.1, 7.1, and 7.2). If there is some translation, the knowledge base is updated and the transition and event rules are modified.

## 8. Additional Features

### 8.1 Modification Requests

To simplify the presentation, we considered that our method could only deal with insertions and deletions of base and view facts. However, we can also handle *modification requests,* which are requests for replacing the value of some attribute in a base or view fact by a new, different value.

Two different approaches exist when dealing with modification requests. The first consists of regarding them as a deletion of a base or view fact followed by an insertion of another fact of the same predicate, where the values have been changed for the desired attributes. The natural meaning of the events allows us to define a goal where all these actions are considered.

As an example, consider the following knowledge base:

Ed (John, D1)

Dm (D1, Mary)

Edm (e,d,m) $\leftarrow$ Ed (e,d) $\wedge$ Dm (d,m)

Assume that the modification request consists of the replacement of the view fact Edm (John, D1, Mary) by Edm (John, D1, Sue). This update can be understood as a request for deleting the first fact and inserting Edm (John, D1, Sue). In our method, $T$ is a solution if there is a constructive derivation from $(\{\leftarrow \delta$Edm (John,D1,Mary) $\wedge\iota$Edm (John,D1,Sue)$\}\ \emptyset\,\emptyset)$ to ([] $T\,C$). In this case, we obtain the solution $T = \{\delta$Dm(D1,Mary), $\iota$Dm(D1,Sue)$\}$.

The second approach for dealing with modification requests consists of adapting the framework described by Urpí and Olivé (1992) and Urpí (1993). In this proposal it is assumed that each predicate has a non-null vector of arguments that form the key for that predicate. The kind of events considered by Olivé (1991) is extended by considering not only insertions and deletions, but also modifications of base and derived predicates, and deriving the modification event rules accordingly. Incorporating the concept of key (not considered in this article), we could adapt our method to use the modification event rules to handle modifications of view and base predicates.

## 8.2 Evaluable Predicates

To simplify the presentation, we considered that all predicates appearing in the body of deductive rules were ordinary (i.e., base or derived). However, we also can deal with evaluable (built-in) predicates like arithmetic operators. In this case, the allowedness condition that the knowledge base should satisfy is that any variable that occurs in a deductive rule has an occurrence in a positive condition of an ordinary predicate (Ullmann, 1988). This ensures that evaluable predicates can be fully instantiated before being evaluated. This evaluation can be performed without accessing the knowledge base. For this reason, the only thing that needs to be done when such a literal is selected, once ground, is to evaluate it. If the result of the evaluation succeeds, we can continue with the translation process. Otherwise, we must consider a different alternative.

## 8.3 Handling Recursion

The Events Method is sound and complete (Section 6). However, in the presence of recursive rules, the method may not terminate because it may enter into an infinite loop. For this reason, in a practical implementation of the Events Method, we should adapt some of the existing loop checking techniques for logic programs to avoid this problem.

In general, loop checking techniques (Bol et al., 1991; Bol, 1993) are based on excluding some kind of repetition in the derivations because such a repetition makes a method enter an infinite loop. We could adopt this technique by forcing our method to stop its search through a certain part of the derivation when it gets a goal (or a variant of it) that had been previously reached in the derivation, and the contents of the translation and condition sets are the same as before. Pruning a tree in this way prevents our method from entering into an infinite loop and, thus, it always terminates.

## 8.4 Preventing Side Effects

Due to the deductive rules, non-requested updates may be induced on some derived predicates. We say that a *side effect* occurs when this happens. The Events Method is able to prevent side effects by giving a set of facts for which we want insertions and/or deletions not to occur. The resulting solutions will satisfy the update request and will satisfy that no insertion and/or deletion is induced for the given set of facts.

A new literal $\neg\iota P_i$ (respectively, $\neg\delta Q_j$) is added to the root goal for each fact $P_i$ (respectively, $Q_j$) for which we want induced insertions (respectively, deletions) not to occur. The root goal has the form $\{\leftarrow u \wedge\neg\iota P_1\wedge\ldots\wedge\neg\iota P_n\wedge\neg\delta Q_1\wedge\ldots\wedge\neg\delta Q_m\}$, where $u$ corresponds to the update request. The Events Method must be applied in the usual way. Note that the approach followed for preventing side effects is similar to the way we deal with integrity constraint maintenance. This is not surprising because, in fact, a violation of an integrity constraint is a particular side effect involving inconsistency predicates.

Preventing side effects is particularly important because it allows us to follow the *constant complement approach* in view updating (Bancilhon and Spyratos, 1981). This elegant approach consists of defining for each view a complement that describes the information not visible within the view, in such a way that the knowledge base may be computed from the view and its complement. In this context, a view update is translated by changing only the view, while the complement remains invariant (that is, the information not visible within the view does not change). We could apply the constant complement approach in the Events Method. First, we should define the complement of each view. Then, when updating a view, we should prevent all possible side effects over its complement.

## 8.5 Repairing Inconsistent Knowledge Bases

Sometimes, it may be interesting to allow for intermediate inconsistent knowledge base states (e.g., to avoid excessive integrity checking). In this case, a method for repairing inconsistent knowledge bases becomes necessary. In this section, we outline how the Events Method can be used to deal with this problem.

As we have seen in Section 2, we associate with each integrity constraint an inconsistency predicate *Icn*, with or without terms. For this reason, an inconsistency predicate can be seen as a special kind of view. When the knowledge base is inconsistent, some of the *Icn* facts will be true. A request for repairing this inconsistency corresponds to a view delete request involving inconsistency predicates. That is, in our framework, the problem of repairing an inconsistent knowledge base is understood as a particular case of view updating.

However, we should adapt the procedure for deriving the augmented knowledge base (Section 3) to derive deletion event rules for inconsistency predicates, and not to apply the consistency simplification which assumes that the knowledge base is consistent before the update. Therefore, both insertion and deletion event rules for view and inconsistency predicates would be obtained in the same way.

To repair an inconsistent knowledge base, the root goal must be defined as $\leftarrow \delta Ic$, and our method can be applied in the usual way. In this case, a translation $T$ contains base facts updates needed to repair all integrity constraints violations.

As an example, consider the following inconsistent knowledge base, where $Q$, $R$, and $S$ are base predicates:

$Q(A)$
$R(A)$
$P(x) \leftarrow Q(x) \land R(x)$
$Ic1(x) \leftarrow P(x) \land \neg S(x)$

We can use our method to restore consistency of this knowledge base. In this case, predicate $Ic$ is defined as $Ic \leftarrow Ic1(x)$, because there is only one integrity constraint. We have to derive its transition and event rules:

$$
\begin{aligned}
&Ic'_{1,1} &&\leftarrow Ic1(\mathrm{x}) \wedge \neg \delta Ic1(\mathrm{x})\\
&Ic'_{1,2} &&\leftarrow \iota Ic1(\mathrm{x})\\
&Ic'_{1} &&\leftarrow Ic'_{1,j} \quad j = 1,2\\
&\iota Ic &&\leftarrow Ic'_{1,2} \wedge \neg Ic\\
&\delta Ic &&\leftarrow \delta Ic1(\mathrm{x}) \wedge \neg Ic'_{1}
\end{aligned}
$$

$T$ will be a solution if there is a constructive derivation from $(\{\leftarrow \delta Ic\}\ \emptyset\ \emptyset)$ to $([]\ T\ C)$. In this example, we obtain the solutions $T_1 = \{\delta Q(A)\}$, $T_2 = \{\delta R(A)\}$, and $T_3 = \{\iota S(A)\}$. Note that all of them restore knowledge base consistency.

An interesting conclusion can be drawn from the above explanation. In this article, we assume that the knowledge base is consistent before the update to derive more efficient event rules for inconsistency predicates and, thus, simplify the process of integrity constraint maintenance. However, we could also apply integrity constraints maintenance when the user requests an update over a known inconsistent knowledge base. In this case, we should first adapt the procedure for deriving the augmented knowledge base as explained above. The Events Method is applied in the same way as defined in Sections 5.1 and 7, with the only difference that the literal $\neg \iota Ic$ is replaced by $\delta Ic$ in the root goal.

## 8.6 Rule Annotation

In some cases, it is not necessary to obtain all possible translations for a given request. In this sense, Tomasic (1988) suggests an interesting technique for reducing the number of obtained solutions. This technique, called *rule annotation,* allows the designer to express additional information as simple markings of rules and predicates. These annotations are used to guide the translation process. The general idea is to explore only the branches which are permitted by the annotations.

We can adapt this framework to reduce the number of solutions obtained by the Events Method, by annotating the event rules to be used. When translating an update request, we only explore the branches permitted by the annotations.

## 8.7 Optimization of the Events Method

In this article, we present a version of the Events Method where the translation process is completely performed at execution time, and when the update request parameters and the content of the extensional part of the knowledge base are known. Nevertheless, we could do some preparatory work at compile time by using *partial evaluation* (Lloyd and Shepherdson, 1991), thus increasing efficiency of our method.

In the Events Method, we could partially evaluate the intensional part of the augmented knowledge base (i.e., deductive, transition, and event rules) with respect to the update request, thus obtaining at compile time a set of equivalent rules that can be evaluated more efficiently at execution time.

## 9. Comparison With Previous Work in View Updating

Related work can be divided into two groups: Methods that have been proposed to solve the view update problem, and methods that are concerned with integrity constraints maintenance. In this section, we compare in detail our method for view updating with the approaches taken by some of the methods in the first group. In Section 10, we compare our approach to integrity constraints maintenance with related methods in this field.

As mentioned in the introduction, the view update problem has attracted a lot of research during past years in relational as well as in deductive databases. To clarify the contribution of our Events Method, we select a representative set of existing methods and provide a comparison with them (see also, Teniente, 1992). All these methods deal with the same class of knowledge bases and view update requests as ours. However, none of them is able to handle transition integrity constraints or prevention of side effects on other views.

- **Drawing Updates From Derivations (Decker).** Let $D$ be a deductive database, and $u$ an update request. Decker (1990) obtained view updates for delete requests by having each non-failed branch in an SLDNF-tree of $D \cup \{\leftarrow u\}$ fail. This is effected by deleting, for each non-failed derivation in the tree, an input clause used in that derivation, or by requesting the insertion of an atom of a negative literal used as an input clause (negative literals are treated as subsidiary insert requests). In insert requests, the concept of *view update trees* is defined to obtain the translations impeded by the selection function employed in the SLDNF resolution. Informally, the basic idea of these trees consists of selecting and resolving each literal of every goal against each candidate input clause but, in general, not every literal has to be selected. Translations are obtained by having a failed goal of this tree succeed. This can be effected by inserting a ground instance of each positive literal appearing in some goal $G$ of the derivation, and by requesting the deletion of the atom of each ground negative literal in $G$ (negative literals are treated as subsidiary delete requests). We see three noteworthy problems with this method. First, the main problem is that it is possible to draw solutions that are invalidated due to negation. That is, solutions that do not satisfy the view update may be obtained. As an example, consider a database containing the following facts and deductive rules:

  $Q(A)$
  $R(A)$
  $P(x) \leftarrow Q(x) \wedge R(x) \wedge \neg S(x)$
  $P(x) \leftarrow T(x)$
  $S(A) \leftarrow Q(A)$

  and let the view update request be the insertion of $P(A)$. Decker's method obtains two different solutions: delete($Q(A)$) and insert($T(A)$). However, the deletion of $Q(A)$ does not satisfy the insert request $P(A)$. Thus, what can be drawn from derivations in the presence of negation are *possible* updates. A possible update is a *valid* one, if the updated database satisfies the update request. This must be validated by running the request in the updated database. In our method, only one translation $T=\{\iota T(A)\}$ would be obtained. Notice that this is the only valid solution. A second difference is that, in some cases, the method proposed by Decker must be iterated a number of times to obtain all valid solutions. The problem is that, in general, it is not known how many iterations should be performed. Because of this, Decker proposed to settle with single pass runs of the method, with which it is possible that not all valid translations are obtained. Our method always gets all valid solutions. Finally, this method only allows integrity checking. On the contrary, our method can also deal with integrity constraints maintenance.

- **Updating Knowledge Bases (Guessoum and Lloyd).** The method of Guessoum and Lloyd (1990, 1991) also uses SLDNF resolution to obtain the translations that satisfy a view update request. In a first version of the method (Guessoum and Lloyd, 1990), procedures for deleting an atom from a normal program and inserting an atom into a normal program were presented. In a later version (Guessoum and Lloyd, 1991), these procedures were generalized such that the deletion procedure calls the insertion procedure and vice versa. Let $K$ be a knowledge base. Translations satisfying a delete request $u$ are obtained by cutting each non-failed branch of the SLDNF tree of $K \cup \{\leftarrow u\}$. For insert requests, the translations are obtained by having a failed derivation of $K \cup \{\leftarrow u\}$ succeed. A translation will be valid if, once updated in the knowledge base, there is an SLDNF refutation associated with the view update request. This work closely parallels the work by Decker (1990). Nevertheless, there are some differences. First, Guessoum and Lloyd check the updated knowledge base to determine whether a solution is valid. Because of this, this method does not present the problem of solutions invalidated due to negation. However, the cost of this verification can be very high because, in general, the knowledge base must be accessed as many times as accesses have been performed during the translation process. The second difference is the most important. To deal with insert requests, Decker defined "view update trees" to obtain all translations impeded by the selection function used in the SLDNF resolution. However, Guessoum and Lloyd did not propose any alternative to this problem and, for this reason, they could not obtain all translations that satisfy an insert request. Our method always gets all valid solutions. Finally, in some cases, this method should be iterated a number of times to obtain a solution without knowing how many iterations should be performed. On the contrary, the Events Method does not need to be iterated.

- **Updates Through Abduction (Kakas and Mancarella).** These authors explored the view update problem within an elegant abductive approach (Kakas and Mancarella, 1990). They clearly distinguished two steps to obtaining the translations. In the first step, the update request was translated into several sets $\Delta_i$. Each $\Delta_i$ is a specification of a set of sufficient requirements that the extensional part of the database, *EDB*, should satisfy for the original request to be effected. Thus, every set $\Delta_i$ corresponds to a valid solution. The second step of the update procedure involves solving the update problem on the EDB generated in the previous step. A first problem of this method is that, in the first step, it may do some unnecessary work due to the fact that it does not take into account the contents of the current database to obtain the $\Delta_i$. Then, some $\Delta_i$ expressing requirements that the current database already satisfies may be obtained. Because of this, the number of $\Delta_i$ obtained may become very large and, thus, the amount of unnecessary work can increase. As an example, consider a knowledge base containing the following facts and rules:

  $Q$
  $R$
  $P \leftarrow Q \wedge R$
  $P \leftarrow S \wedge T$
  $S \leftarrow A \wedge B$
  $T \leftarrow C \wedge D$

  and let the view update request be the deletion of $P$. In this method, eight different sets $\Delta_i$ of sufficient requirements that the *EDB* should satisfy are obtained: $\Delta_1=\{Q^*, A^*\}$, $\Delta_2=\{Q^*, B^*\}$, $\Delta_3=\{Q^*, C^*\}$, $\Delta_4=\{Q^*, D^*\}$, $\Delta_5=\{R^*, A^*\}$, $\Delta_6=\{R^*, B^*\}$, $\Delta_7=\{R^*, C^*\}$, $\Delta_8=\{R^*, D^*\}$, where $P^*$ denotes the fact that $P$ must not hold in the new database state. However, there are only two solutions that satisfy the update request: {delete($Q$)} or {delete($R$)}. In our method, two translations are obtained: $T_1=\{\delta Q\}$ and $T_2=\{\delta R\}$. Notice that they correspond to the valid solutions. Another important difference is the way integrity constraints maintenance is handled. With the method of Kakas and Mancarella, integrity constraints can be maintained dynamically as part of the derivation performed in the first step. The problem is that perhaps not all violations of integrity constraints may be detected during this derivation. As an example, consider the database containing the following rules and integrity constraints:

  $S(x) \leftarrow Q(x)$
  $P(x) \leftarrow Q(x)$
  $Ic1(x) \leftarrow P(x) \wedge \neg R(x)$

  and let the view update request be the insertion of $S(A)$. In this method, a set $\Delta=\{Q(A)\}$ would be obtained. Thus, in step 2, $Q(A)$ would be inserted in the database. Note that this solution does not satisfy the integrity constraint. In our method, we obtain the solution $T=\{\iota Q(A),\ \iota R(A)\}$, which satisfies both the update request and the integrity constraint.

- **Updating Intensional Predicates (Atzeni and Torlone).** This method (Torlone and Atzeni, 1991; Atzeni and Torlone, 1992) deals with view updating in definite deductive databases (i.e., databases where view predicates can only be defined by means of function free Horn rules, and without negation). Despite this expressive limitation, we remark on some interesting features of this proposal. The main contribution of this work is to provide a formalization of a declarative semantics of updates of facts. Among the possible translations of an insertion, they considered as important the *minimal potential results,* which include only the information that is strictly needed to satisfy the insert request. They distinguished between deterministic and non-deterministic updates, depending on the existence of a minimal potential result. Semantics for deletions were defined in a similar way. They also proposed a method for translating view updates into updates of the underlying base facts. This method is based on SLD-resolution and its soundness and completeness are proved. In a later version (Torlone and Atzeni, 1991), this method was extended by considering some integrity constraints (specifically functional dependencies) in the management of view updates, thus being able to resolve potential ambiguities in several cases. We see two noteworthy differences between the method of Atzeni and Torlone and the Events Method. First, they considered a particular case of deductive databases where view predicates are defined by means of function free Horn rules, and integrity constraints are restricted to functional dependencies, while the Events Method deals with knowledge bases in general, and the only condition that clauses must satisfy is allowedness. In this context, the view update problem becomes more complex because the operations are not monotone in general. Second, they applied the integrity checking approach when dealing with functional dependencies, while the Events Method follows the integrity constraints maintenance approach. The next example (Torlone and Atzeni, 1991) illustrates the main advantages of the latter. Consider the following database, where the functional dependency $p \rightarrow c$ (every professor teaches at most one course) is defined:

  Teaches (Smith, CS101)
  CPS (c,p,s) $\leftarrow$ Teaches(p,c) $\wedge$ Attends(s,c)

  Let the view update request be the insertion of the fact CPS(CS202,Smith,Tom). In this case, Atzeni and Torlone's method would not obtain any solution since they take the constraints checking approach, and Smith cannot teach both CS101 and CS202. However, in our method the translation $T = \{\delta$Teaches(Smith,CS101), $\iota$Teaches(Smith,CS202), $\iota$Attends(Tom,CS202)$\}$ would be obtained.

## 10. Comparison With Previous Work in Integrity Maintenance

In this section, we compare in detail our approach to integrity constraints maintenance with related methods in this field. None of the other methods dealt with insertions and deletions of deductive rules, nor with insertions of integrity constraints.

- **Reactive Consistency Control (Moerkotte and Lockemann).** Moerkotte and Lockemann (1991) explored the problem of reactive consistency control in the context of definite deductive databases. When a transaction executed by the user violates one or more integrity constraints, this method automatically generates repairs (i.e., transactions that must be appended to the original transaction to regain consistency). This method clearly distinguishes three steps to obtain the repairs. In the first step, a set of *symptoms* is obtained from the violated integrity constraints. These symptoms correspond to (possibly derived) facts that violate the existing constraints. In the second step, a set of *causes* is generated from the symptoms. Causes correspond to base (stored) facts that give rise to a symptom. In step three, the causes are transformed into *repairs* by syntactic modification. The method of Moerkotte and Lockemann inspired our approach to integrity constraints maintenance, although it is restricted to definite deductive databases, and considers only flat transactions (i.e., transactions that consist of updates of base facts). On the contrary, the Events Method allows negation to appear in the body of clauses, and deals with updates of base facts, view updates, and insertions and deletions of deductive and integrity rules.

- **Integrity Maintenance Systems (Ceri et al.)** The approach taken by Ceri et al. (1992) is to provide automatic repair of inconsistent databases by using *production rules.* For each constraint, production rules are used to detect constraint violation, and to initiate database operations that restore consistency. This work extends, in some sense, the method presented by Ceri and Widom (1990), in which constraints (expressed in an SQL-like language) were used to generate production rules. The problem was that the translation from integrity constraints to constraint-maintaining production rules was not completely automatic, and it required designer intervention. Ceri et al. (1992) avoided this problem because, in their proposal, production rules can be automatically generated. This method is mainly concerned with obtaining a set of efficient, optimized production rules. This step can be fully performed at compile time, thus providing an efficient way for generating repairs of integrity constraints violations. In our Events Method, the analysis is completely performed at execution time, although we could do some preparatory work at compile time by means of partial evaluation (Section 8.6). A second important difference is that the method of Ceri et al. is restricted to definite databases and view predicates cannot appear in the body of integrity constraints, and that it considers only transactions which consist of updates of base facts. The Events Method deals with more general knowledge bases and kinds of updates.

## 11. Conclusions

In this article, we present a new method that can be used for integrity constraints maintenance, view updating, and their combination. Moreover, the method can also be combined with any integrity checking method for view updating and integrity checking.

Our method is based on events and transition rules, which explicitly define the insertions and deletions induced by a knowledge base update. Using these rules, an extension of the SLDNF proof procedure allows us to obtain all possible ways of updating a knowledge base without violating any integrity constraint. The kind of updates handled by our method are: updates of base facts, updates of deductive rules, updates of integrity constraints, and view updates. In the latter case, our method also translates a view update request into appropriate updates of the underlying base facts. We have proven soundness and completeness in this case.

Our method uniformly handles both insert and delete requests. Complex updates, such as mixed multiple updates and modifications, also can be requested. We have presented several important additional features of our method, such as preventing side effects on other views, repairing inconsistent knowledge bases, and maintaining transition integrity constraints.

We have also compared our method with previous ones and shown that we extend their functionalities, either by dealing with more general knowledge bases or by considering more general kinds of updates. Our method is able to solve the problems of view updating, integrity constraints maintenance, and their combination in this more general setting.

Our purpose in this article has been to develop a sound and complete method for dealing with view updating and integrity maintenance in knowledge bases. Efficiency issues have not been considered. We made a preliminary implementation of the method in Teniente (1992), but we plan to continue working to develop an efficient implementation.

On the other hand, we have not considered the computational cost required, in the general case, to obtain all minimal translations. Again, the aim has been to provide a method that guarantees completeness and extends the functionalities of previous methods. However, it is obvious that the obtention of all minimal translations may not be practical in some applications. In such cases, the users might be interested in finding only one solution (or a few), at the expense of losing the guarantee of minimality. We also plan to continue working in this direction.

## Acknowledgements

We would like to thank Dolors Costal, Hendrik Decker, Robert Demolombe, Carme Martín, Enric Mayol, Joan Antoni Pastor, Carme Quer, Maria Ribera Sancho, Jaume Sistac, and Toni Urpí for many useful comments and discussions. We are also indebted to the anonymous reviewers who provided us with excellent comments during the course of reviewing this work. This work has been partially supported by the CICYT PRONTIC program project TIC 680.

## References

Abiteboul, S. Updates, a new frontier. *Proceedings of the International Conference on Database Theory,* Bruges, Belgium, 1988.

Atzeni, P. and Torlone, R. Updating intensional predicates in datalog. *Data and Knowledge Engineering,* 8:1-17, 1992.

Bancilhon, F. and Ramakrishnan, R. An amateur's introduction to recursive query processing. *Proceedings of the International ACM SIGMOD Conference on the Management of Data,* Washington D.C., 1986.

Bancilhon, F and Spyratos, N. Update semantics of relational views. *ACM Transactions on Database Systems,* 6(4):557-575, 1981.

Bol, R.N. Loop checking and negation. *Journal of Logic Programming,* 15:147-175, 1993.

Bol, R.N., Apt, K.R., and Klop, J.W. An analysis of loop checking mechanisms for logic programs. *Theoretical Computer Science,* 86:35-79, 1991.

Bry, F. Intensional updates: Abduction via deduction. *Proceedings of the Seventh ICLP,* Jerusalem, 1990.

Bry, F., Manthey, R., and Martens, B. Integrity verification in knowledge bases. *ECRC Report D.2.1.a,* Munich, 1990.

Cavedon, L. and Lloyd, J. A completeness theorem for SLDNF resolution. *Journal of Logic Programming,* 7:177-191, 1989.

Ceri, S., Fraternali, P., Paraboschi, S., and Tanca, L. Integrity maintenance systems: An architecture. *Third International Workshop on the Deductive Approach to Information Systems and Databases,* Roses, Catalonia, 1992.

Ceri, S. and Widom, J. Deriving production rules for constraint maintenance. *Proceedings of the Sixteenth VLDB Conference,* Brisbane, Australia, 1990.

Clark, K.L. Negation as failure. In: Gallaire, H. and Minker, J., eds., *Logic and Databases.* New York: Plenum Press, 1978, pp. 293-322.

Cosmadakis, S. and Papadimitriou, C. Updates of relational views. *Journal of the Association for Computing Machinery,* 31(4):742-760, 1984.

Date, C.J. Updating views. In: *Relational Databases: Selected Writings,* Reading, MA: Addison-Wesley, 1986, pp.367-395.

Dayal, U. and Bernstein, P.A. On the correct translation of update operations on relational views. *ACM Transactions on Database Systems,* 8(3):381-416, 1982.

Decker, H. The range form of databases or: How to avoid floundering. *Proceedings of the Fifth ÖGAI,* Innsbruck, Austria, 1989.

Decker, H. Drawing updates from derivations. *Proceedings of the Third International Conference on Database Theory,* Paris, 1990.

Decker, H. and Cavedon, L. Generalizing allowedness while retaining completeness of SLDNF resolution. *Proceedings of the Third Workshop on Computer Science Logic,* Kaiserslautern, 1990.

Fagin, R., Kuper, G.M., Ullman, J.D., and Vardi, M.Y. Updating logical databases. *Advances in Computing Research,* 3:1-18, 1986.

Fagin, R., Ullman, J.D., and Vardi, M.Y. On the semantics of updates in databases. *Proceedings of the ACM PODS,* 1983.

Furtado, A.L. and Casanova, M.A. Updating relational views. In: Kim, W., Reiner, D.S., and Batory, D.S., eds., *Query Processing in Database Systems,* Berlin: Springer-Verlag, 1985, pp. 127-142.

Gallaire, H., Minker, J., and Nicolas, J.M. Logic and databases: A deductive approach. *ACM Computing Surveys,* 16(2):153-185, 1984.

Gärdenfors, P. *Knowledge in Flux: Modeling the Dynamics of Epistemic States.* Cambridge, MA: MIT Press, 1988.

Guessoum, A. and Lloyd, J.W. Updating knowledge bases. *New Generation Computing,* 8(1):71-89, 1990.

Guessoum, A. and Lloyd, J.W. Updating knowledge bases II. *New Generation Computing,* 10:73-100, 1991.

Kakas, A. and Mancarella, P. Database updates through abduction. *Proceedings of the Sixteenth VLDB Conference,* Brisbane, Australia, 1990.

Keller, A.M. Algorithms for translating view updates to database updates for views involving selection, projections, and joins. *Proceedings of the Fourth ACM SIGACT-SIGMOD Symposium on the Principles of Database Systems,* Portland, OR, 1985.

Keller, A.M. Choosing translator at view definition time. *Proceedings of the Twelfth VLDB Conference,* Kyoto, Japan, 1986.

Kowalski, R. Database updates in the event calculus. *Journal of Logic Programming,* 12:121-146, 1992.

Kunen, K. Signed data dependencies in logic programs. *Journal of Logic Programming,* 7:231-245, 1989.

Langerak, R. View updates in relational databases with an independent schema interface. *ACM Transactions on Database Systems,* 15(1):40-66, 1990.

Larson, J. and Sheth, A. Updating relational views using knowledge at view definition and view update time. *Information Systems,* 16(2):145-168, 1991.

Lloyd, J.W. *Foundations on Logic Programming,* 2nd edition, New York: Springer, 1987.

Lloyd, J.W. and Shepherdson, J.C. Partial evaluation in logic programming. *Journal of Logic Programming,* 8(11):217-247, 1991.

Lloyd, J.W. and Topor, R.W. Making Prolog more expressive. *Journal of Logic Programming,* 1(3):225-240, 1984.

Manchanda, S. and Warren, D.S. A logic-based language for database updates. In: Minker, J., ed., *Foundations of Deductive Databases and Logic Programming.* Los Altos, CA: Morgan-Kaufmann, 1988, pp. 363-394.

Masunaga, Y. A relational database view update translation mechanism. *Proceedings of the Tenth VLDB Conference,* Singapore, 1984.

Moerkotte, G. and Lockemann, P.C. Reactive consistency control in deductive databases. *ACM Transactions on Database Systems,* 16(4):670-702, 1991.

Nicolas, J.M. Logic for improving integrity checking in relational data bases. Technical report, ONERA-CERT, 1979. Also in: *Acta Informatica,* 18(3):227-253, 1982.

Olivé, A. On the design and implementation of information systems from deductive conceptual models. *Proceedings of the Fifteenth VLDB Conference,* Amsterdam, 1989.

Olivé, A. Integrity checking in deductive databases. *Proceedings of the Seventeenth VLDB Conference,* Barcelona, 1991.

Reiter, R. Towards a logical reconstruction of relational database theory. In: Brodie, M.L., Mylopoulos, J., and Schmidt, J.W., eds., *On Conceptual Modeling,* New York: Springer-Verlag, 1984, pp. 191-233.

Sadri, F. and Kowalski R. A theorem-prover approach to database integrity. In: Minker, J., ed., *Foundations of Deductive Databases and Logic Programming,* Los Altos, CA: Morgan-Kaufman, 1988, pp. 313-362.

Teniente, E. El Mètode dels Esdeveniments per a l'actualització de vistes en bases de dades deductives. PhD Thesis, Universitat Politècnica de Catalunya, Barcelona, 1992 (in Catalan).

Teniente, E. and Olivé, A. The events method for view updating in deductive databases. *International Conference on Extending Database Technology,* Vienna, 1992.

Teniente, E. and Olivé, A. Updating knowledge bases while maintaining their consistency. Research Report LSI-94-25-R, UPC, Barcelona, 1994.

Tomasic, A. View update annotation in definite deductive databases. *Proceedings of the International Conference on Database Theory,* Bruges, Belgium, 1988.

Torlone, R. and Atzeni, P. Updating deductive databases with functional dependencies. *Second International Conference on Deductive and Object Oriented Databases,* Munich, 1991.

Ullman, J.D. *Principles of Database and Knowledge-Base Systems.* New York: Computer Science Press, 1988.

Urpí, T. El Mètode dels Esdeveniments per al càlcul de canvis en bases de dades deductives. PhD Thesis, Universitat Politècnica de Catalunya, Barcelona, 1993 (in Catalan).

Urpí, T. and Olivé, A. A method for change computation in deductive databases. *Proceedings of the Eighteenth VLDB Conference,* Vancouver, Canada, 1992.

Winslett, M. Updating logical databases. *Cambridge Tracts in Theoretical Computer Science,* 9, 1990.

## Appendix A: Simplifications of the Event Rules

**Proof:** *New simplification*

In: (14) $\iota P(\mathbf{x}) \leftarrow P'_{i,j}(\mathbf{x}) \wedge \neg P_1(\mathbf{x}) \wedge \ldots \wedge \neg P_m(\mathbf{x})$
for $i = 1 \ldots m$ and $j = 2 \ldots 2^{ki}$

if the transition rule (9) corresponding to $P'_{i,j}$ has a literal $N(L'_{i,j,h}) = [\iota Q_h(\mathbf{x}_h) \mid \delta Q_h(\mathbf{x}_h)]$ in $U(P'_{i,j})$, then (14) can be rewritten as:

(A.1) $\iota P(\mathbf{x}) \leftarrow [\iota Q_h(\mathbf{x}_h) \mid \delta Q_h(\mathbf{x}_h)] \wedge \alpha \wedge \neg P_1(\mathbf{x}) \wedge \ldots \wedge \neg P_i(\mathbf{x}) \wedge \ldots \wedge \neg P_m(\mathbf{x})$

where $\alpha$ comprises all other literals in $P'_{i,j}$. On the other hand, we have:

(A.2) $P_i(\mathbf{x}) \leftrightarrow [Q_h(\mathbf{x}_h) \mid \neg Q_h(\mathbf{x}_h)] \wedge \beta$

where $\beta$ comprises all other literals in the body of $P_i(\mathbf{x})$. Replacing (A.2) in (A.1) we get:

(A.3) $\iota P(\mathbf{x}) \leftarrow [\iota Q_h(\mathbf{x}_h) \mid \delta Q_h(\mathbf{x}_h)] \wedge \alpha \wedge \neg P_1(\mathbf{x}) \wedge \ldots \wedge \neg P_{i-1}(\mathbf{x}) \wedge$
$\neg([Q_h(\mathbf{x}_h) \mid \neg Q_h(\mathbf{x}_h)] \wedge \beta) \wedge$
$\neg P_{i+1}(\mathbf{x}) \wedge \ldots \wedge \neg P_m(\mathbf{x})$

and given that, by (1), $\iota Q_h(\mathbf{x}_h) \rightarrow \neg Q_h(\mathbf{x}_h)$ and, by (2), $\delta Q_h(\mathbf{x}_h) \rightarrow Q_h(\mathbf{x}_h)$, we can remove $\neg([Q_h(\mathbf{x}_h) \mid \neg Q_h(\mathbf{x}_h)] \wedge \beta)$ from (A.3), obtaining (15).

**Proof:** *Old simplification*

In: (14) $\iota P(\mathbf{x}) \leftarrow P'_{i,j}(\mathbf{x}) \wedge \neg P_1(\mathbf{x}) \wedge \ldots \wedge \neg P_m(\mathbf{x})$
for $i = 1 \ldots m$ and $j = 2 \ldots 2^{ki}$

if all literals in the transition rule (9) corresponding to $P'_{i,j}$ have the form $O(L'_{i,j,h})$ in $U(P'_{i,j})$, and there are $q$ such literals, then $P'_{i,j}$ is:

(A.4) $P'_{i,j}(\mathbf{x}) \leftrightarrow [Q_1(\mathbf{x}_1) \wedge \neg \delta Q_1(\mathbf{x}_1) \mid \neg Q_1(\mathbf{x}_1) \wedge \neg \iota Q_1(\mathbf{x}_1)] \wedge \ldots \wedge$
$[Q_1(\mathbf{x}_q) \wedge \neg \delta Q_q(\mathbf{x}_q) \mid \neg Q_q(\mathbf{x}_q) \wedge \neg \iota Q_q(\mathbf{x}_q)] \wedge E(P'_{i,j})$

On the other hand, we have:

$P_i(x) \leftrightarrow U(P_i) \wedge E(P_i)$
$P_i(x) \leftrightarrow U(P_i) \wedge E\_P_i(x)$
(A.5) $P_i(\mathbf{x}) \leftrightarrow [Q_1(\mathbf{x}_1) \mid \neg Q_1(\mathbf{x}_1)] \wedge \ldots \wedge [Q_q(\mathbf{x}_q) \mid \neg Q_q(\mathbf{x}_q)] \wedge E\_P_i(\mathbf{x})$

Replacing in (14) $P'_{i,j}(\mathbf{x})$ with (A.4), and $\neg P_i(\mathbf{x})$ with (A.5) we get:

(A.6) $\iota P(\mathbf{x}) \leftarrow [Q_1(\mathbf{x}_1) \wedge \neg \delta Q_1(\mathbf{x}_1) \mid \neg Q_1(\mathbf{x}_1) \wedge \neg \iota Q_1(\mathbf{x}_1)] \wedge \ldots \wedge$
$[Q_q(\mathbf{x}_q) \wedge \neg \delta Q_q(\mathbf{x}_q) \mid \neg Q_q(\mathbf{x}_q) \wedge \neg \iota Q_q(\mathbf{x}_q)] \wedge E(P'_{i,j})$
$\neg P_1(\mathbf{x}) \wedge \ldots \wedge \neg P_{i-1}(\mathbf{x}) \wedge$
$\neg([Q_1(\mathbf{x}_1) \mid \neg Q_1(\mathbf{x}_1)] \wedge \ldots \wedge [Q_q(\mathbf{x}_q) \mid \neg Q_q(\mathbf{x}_q)] \wedge E\_(P'_i(\mathbf{x})) \wedge$
$\neg P_{i+1}(\mathbf{x}) \wedge \ldots \wedge \neg P_m(\mathbf{x})$

from where we can remove literals $[Q_1(\mathbf{x}_1) \mid \neg Q_1(\mathbf{x}_1)] \wedge ... \wedge [Q_q(\mathbf{x}_q) \mid \neg Q_q(\mathbf{x}_q)]$, thus obtaining (16).

**Proof:** *Simplification of the deletion event rules*

If we denote: $\alpha = \neg P'_1(\mathbf{x}) \wedge ... \wedge \neg P'_{i-1}(\mathbf{x}) \wedge \neg P'_{i+1}(\mathbf{x}) \wedge ... \wedge \neg P'_m(\mathbf{x})$, then rules (18) become:

(A.9) $\delta P(\mathbf{x}) \leftarrow P_i(\mathbf{x}) \wedge \neg P'_i(\mathbf{x}) \wedge \alpha$ for $i = 1 ... m$

By (10) $P'_i(\mathbf{x}) \leftrightarrow P'_{i,1}(\mathbf{x}) \vee ... \vee P'_{i,2^{ki}}(\mathbf{x})$ and replacing the above:

(A.10) $\delta P(\mathbf{x}) \leftarrow P_i(\mathbf{x}) \wedge \neg P'_{i,1}(\mathbf{x}) \wedge ... \wedge \neg P'_{i,2^k}(\mathbf{x}) \wedge \alpha$

which, if we make $\beta = \neg P'_{i,2}(\mathbf{x}) \wedge ... \wedge \neg P'_{i,2^{ki}}(\mathbf{x})$, becomes:

(A.11) $\delta P(\mathbf{x}) \leftarrow P_i(\mathbf{x}) \wedge \neg P'_{i,1}(\mathbf{x}) \wedge \beta \wedge \alpha$ for $i = 1 ... m$

Assume $P_i(\mathbf{x}) \leftrightarrow L_1 \wedge ... \wedge L_n$, with $U(P_i) = L_1 \wedge ... \wedge L_q$, and $E(P_i) = L_{q+1} \wedge ... \wedge L_n$. Then, by (11):

(A.12) $P'_{i,1}(\mathbf{x}) \leftrightarrow O(L'_1) \wedge ... \wedge O(L'_n)$

and

(A.13) $P'_{i,1}(\mathbf{x}) \leftrightarrow U(P_i) \wedge [\neg\delta Q_1(\mathbf{x}_1) \mid \neg\iota Q_1(\mathbf{x}_1)] \wedge ... \wedge [\neg\delta Q_q(\mathbf{x}_q) \mid \neg\iota Q_q(\mathbf{x}_q)] \wedge E(P'_{i,1})$

Replacing (A.13) in (A.11) we get:

(A.14) $\delta P(\mathbf{x}) \leftarrow P_i(\mathbf{x}) \wedge \neg U(P_i) \wedge \beta \wedge \alpha$ for $i = 1...m$
(A.15) $\delta P(\mathbf{x}) \leftarrow P_i(\mathbf{x}) \wedge [\delta Q_j(\mathbf{x}_j) \mid \iota Q_j(\mathbf{x}_j)] \wedge \beta \wedge \alpha$ for $i = 1...m$ and $j = 1...q$
(A.16) $\delta P(\mathbf{x}) \leftarrow P_i(\mathbf{x}) \wedge \neg E(P_{i,1}) \wedge \beta \wedge \alpha$ for $i = 1...m$

where rules (A.14) can be removed, since $P_i(x) \rightarrow U(P_i)$. In (A.15), literals $L_j$ of $P_i(\mathbf{x})$ and $\beta$ can also be removed, since $[\delta Q_j(\mathbf{x}_j) \mid \iota Q_j(\mathbf{x}_j)] \rightarrow L_j$, $\delta Q_j(\mathbf{x}_j) \rightarrow \neg\iota Q_j(\mathbf{x}_j)$ and $\iota Q_j(\mathbf{x}_j) \rightarrow \neg\delta Q_j(\mathbf{x}_j)$ and we get rules (19).

In rules (A.16), if we replace $\beta$ and distribute $\wedge$ over $\vee$, we get:

(A.17) $\delta P(\mathbf{x}) \leftarrow P_i(\mathbf{x}) \wedge \neg E(P_{i,j}) \wedge ... \wedge \neg E(P_{i,2^k}) \wedge \alpha$ for $i = 1...m$

and replacing $E(P_{i,1})$ we obtain, after a simple transformation, rules (20).

## Appendix B: Syntactic Properties of A(K)

In this appendix, we review the definition of the syntactic properties of logic programs, related to the SLDNF completeness. We then show that if a knowledge base $K$ is stratified, then the augmented knowledge base $A(K)$ is call-consistent. This is an interesting result, since known completeness results for SLDNF require these properties.

Stratification and call-consistency are defined in terms of the dependency graph. We follow the terminology and definitions of Decker and Cavedon (1990), where more details can be found. The nodes of the dependency graph are the facts and rules of the knowledge base, and for each pair $F, F'$ of nodes there is an edge from $F'$ to $F$ if there is an atom $A$ in the body of $F$ such that the predicates in $A$ and the head of $F'$ are the same. The edge is marked positive (respectively, negative) if $A$ is positive (respectively, negative) in $F$.

If $F$ and $F'$ are two nodes in the dependency graph of $K$, we say that:

a) $F$ *depends* on $F'$ if there is a path from $F'$ to $F$.
b) $F$ *depends positively* (respectively, negatively) on $F'$ if there is a path from $F'$ to $F$ containing no negative edge (respectively, at least one negative edge).
c) $F$ *depends evenly* (respectively, oddly) on $F'$ if there is a path from $F'$ to $F$ containing an even (respectively, odd) number of negative edges.
d) $F$ *depends recursively on itself* if there is a path from $F$ to $F$ of length greater than 0.
e) The set of nodes in $K$ on which $F$ depends is denoted by $K_F$.

These definitions are used to characterize the following properties of a knowledge base $K$ and a goal $G$:

a) $K$ is *hierarchical* if no node in the dependency graph of $K$ depends recursively on itself.
b) $K$ is *stratified* if no node in the dependency graph of $K$ depends negatively on itself.
c) $K$ is *call-consistent* if no node in the dependency graph of $K$ depends oddly on itself.
d) $K \cup \{G\}$ is *strict* if there is no pair $F, F'$ of nodes in the dependency graph of $K_G \cup \{G\}$ such that $F$ depends evenly and oddly on $F'$.
e) $K \cup \{G\}$ is *even* if there is no pair $F, F'$ of nodes in the dependency graph of $K_G \cup \{G\}$ such that $F$ depends evenly and oddly on $F'$, and $F'$ depends recursively on itself.

It is easy to show the following relationships between the properties of $K$ and those of $A(K)$:

a) If $K$ is hierarchical then $A(K)$ is also hierarchical.
b) If $K$ is stratified then $A(K)$ is also call-consistent.
c) If $K$ is call-consistent then $A(K)$ is also call-consistent.

We illustrate relationship (b) by means of an example. Assume a stratified knowledge base $K$ and a recursive rule in $K$ such as $P \leftarrow P$ (we only show names). In $A(K)$, we would then have:

(A.1) $P' \leftarrow P \wedge \neg \delta P$
(A.2) $P' \leftarrow \iota P$
(A.3) $\iota P \leftarrow P' \wedge \neg P$
(A.4) $\delta P \leftarrow P \wedge \neg P'$

All these rules depend recursively on themselves, but it is always an even dependency. Thus $A(K)$ becomes call-consistent.

SLDNF resolution is incomplete, in general, but there are large classes of knowledge bases and goals for which it is complete. Clark (1978) proved completeness for hierarchical and allowed knowledge bases. Cavedon and Lloyd (1989) showed completeness for knowledge bases and goals which are allowed, strict, and stratified. Kunen (1989) showed completeness for knowledge bases and goals which are allowed, strict, and call-consistent. More recently, Decker and Cavedon (1990) generalized the above results by proving completeness for recursively covered (a generalization of allowed) knowledge bases $K$ and goals $G$ such that $K$ is call-consistent and $K \cup \{G\}$ is even.