Updating Knowledge Bases while Maintaining their Consistency

Ernest Teniente
Antoni Olivé

Report LSI–94–25–R

Updating Knowledge Bases while Maintaining their Consistency

Ernest Teniente
Antoni Olivé

Universitat Politècnica de Catalunya
Facultat d'Informàtica
Pau Gargallo 5
08028 Barcelona - Catalonia
e-mail: [teniente/olive]@lsi.upc.es

Abstract

In general, several problems may arise when updating a knowledge base. One of the most important ones is that of integrity constraints satisfaction. The classical approach to deal with 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 which *maintain* knowledge base consistency. Another major problem present in knowledge base updating is that of *view updating*, which is concerned with determining how a request to update a view should be translated into an update of the underlying base facts.

In this paper we propose a new method for updating knowledge bases while maintaining their consistency. Our method can be used for integrity constraints maintenance, view updating and their combination. 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.

KEYWORDS: knowledge base, 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 (derived 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 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.

In general, several problems may arise when updating a knowledge base [Abi88, Kow92]. 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 [Nic82] and deductive databases (see [BMM90, Oli91] and the references therein). The problem has also been studied for full, first-order logical databases [GMN84, Rei84].

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 the integrity constraints but now, 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. Eventually, 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. Now, 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 just 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 [DB82], usually restricted to particular integrity constraints (such as keys, functional dependencies or subset constraints). Recently some methods have been developed for deductive databases [CW90, ML91, CFPT92]. The problem has also been studied for logical databases [FUV83, FKUV86, Gär88, Win90].

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 appropriately translated 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 [BS81, CP84, Mas84, Kel85, Kel86, Dat86, Lan90, LS91] and in deductive databases [Tom88, Bry90, Dec90, KM90, GL90, GL91, TA91, TO92, AT92]. It has also been studied for logical databases [FUV83, FKUV86].

In principle, it may happen that some translations corresponding to a given view update request do 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. Eventually, no such translation may be found, and then the view update request would be rejected.

However, it is also possible to combine *view updating and integrity maintenance*. Now, 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 (eventually empty) of translations, that do not necessarily correspond to a subset of the translations obtained by view updating alone.

In this paper, we describe a method, that we call the Events Method, that can be used for integrity constraints maintenance, view updating and their combination [Ten92]. The method can also be combined with any integrity checking method for view updating and integrity checking. A simplified version of the method was presented in [TO92] for a particular case of view updating in deductive databases.

Our method is an application of an approach developed for the design of information systems from deductive conceptual models [Oli89]. 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 have also been used for developing a new integrity checking method [Oli91] and for condition monitoring in active databases [UO92, Urp93].

Our method takes into account not only classical updates of base facts and view updates, but also other kinds of updates like insertions and deletions of deductive rules and integrity constraints.

This paper is organized as follows. Next section reviews basic concepts of knowledge bases. Section 3, which is based on [Oli91], reviews the concept of event 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 prove correctness and completeness of our method. Section 7 extends the types of updates by considering also 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. Finally, in section 11 we present our conclusions. We assume the reader is familiar with logic programming [Llo87].

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 [GMN84, Llo87, Ull88] and present our notation.

Throughout the paper, we consider a first order language with a universe of constants, a set of variables, a set of predicate names and no function symbols. We will use names beginning with a capital letter for predicate symbols and constants (with the exception that constants are also permitted to be numbers) and names beginning with a lower case letter for variables.

A *term* is a variable symbol or a constant symbol (that is, 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 which 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:

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

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 [Dec89] and then using the procedure described in [LlT84].

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:

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

states that Persons can only live in known Cities. If Lives(John,A) is a fact and there does not exist a corresponding City (A) fact, then Ic1(John,A) will be true and, therefore, the integrity constraint Ic1 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 (eventually) in the body of deductive rules. A derived predicate appears only in the intensional part. Every knowledge base can be defined in this form [BR86].

As usual, we require that the knowledge base before and after any updates is *allowed* [Llo87], 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(**C**) holds in K, where **C** is a vector of constants, two cases are possible:

a.1. P'(**C**) also holds in K' (both P(**C**) and P'(**C**) are true).
a.2. P'(**C**) does not hold in K' (P(**C**) is true but P'(**C**) is false).

and assuming that P'(**C**) holds in K', two cases are also possible:

b.1 P(**C**) also holds in K (both P(**C**) and P'(**C**) are true).
b.2. P(**C**) does not hold in K (P'(**C**) is true but P(**C**) is false).

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

Thus, for example, if Works(employee, unit) is a derived predicate, ιWorks(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* ιP and a *deletion event predicate* δP, 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 **x** is a vector of variables. From the above, we then have the equivalencies [Urp93]:

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

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

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

3.2 Transition Rules

Let us consider a derived or inconsistency predicate P of the knowledge base. Assume that the definition of P consists of m rules, m≥ 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) $P(\mathbf{x}) \leftarrow P_i(\mathbf{x}) \qquad i = 1...m$

Example 3.1: Assume a knowledge base with the following rules:

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

they would be rewritten as:

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

Consider now 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 *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 by:

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

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

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

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

Example 3.2: Consider the rules given in Example 3.1. 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: $P'_1(x) \leftarrow P'_{1,j}(x)$ $\quad j = 1, 2$
$P'_2(x) \leftarrow P'_{2,j}(x)$ $\quad j = 1, \ldots, 4$
$T'_1(x) \leftarrow T'_{1,j}(x)$ $\quad j = 1, \ldots, 4$

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

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

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

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 $L'_{i,r}$ $(r = 1...n)$ literals we get:

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

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:

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

and

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

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

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

3.3 Insertion Event Rules

Let P be a derived or inconsistency predicate. Insertion predicates ιP 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:

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

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

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

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

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

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

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

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

In order 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 **x**. The existential part, denoted $E(P_i)$, is the conjunction of the literals in the body having some variable which is not in **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, we will need sometimes an auxiliary predicate E_P_i defined by the rule: $E\_P_i(\mathbf{x}) \leftarrow E(P_i)$

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

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

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

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

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 3.3: Consider predicate T as defined in example 3.2. Applying (14) we get the insertion event rules:

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

Given that $U(T'_{1,2})$ has literal $N(\neg S'(x)) = \delta S(x)$, 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:

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

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

3.4 Deletion Event Rules

Let P be a derived predicate. Deletion predicates δP 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) $\quad \delta P(\mathbf{x}) \leftarrow P_i(\mathbf{x}) \wedge \neg P'(\mathbf{x}) \qquad\qquad$ for i = 1 ... 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) $\delta P(\mathbf{x}) \leftarrow P_i(\mathbf{x}) \wedge \neg P'_1(\mathbf{x}) \wedge ... \wedge \neg P'_m(\mathbf{x})$ for $i = 1 ... m$

These rules can be transformed into a set of equivalent, but simplified rules, which can be evaluated more efficiently. Let k_i^u 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) $\delta P(\mathbf{x}) \leftarrow U(P_i)\backslash L_{i,j} \wedge [\delta Q_{i,j}(\mathbf{x_{i,j}}) | \iota Q_{i,j}(\mathbf{x_{i,j}})] \wedge E(P_i) \wedge \alpha$
for $i = 1...m$, $j = 1... k_i^u$

(20) $\delta P(\mathbf{x}) \leftarrow U(P_i) \wedge E(P_i)\backslash L_{i,j} \wedge [\delta Q_{i,j}(\mathbf{x_{i,j}}) | \iota Q_{i,j}(\mathbf{x_{i,j}})] \wedge \neg E\_P'_i(\mathbf{x}) \wedge \alpha$
for $i = 1...m$, $j = 1... k_i^e$

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

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 δP facts (induced deletions) happen in a transition in terms of old state predicates and events.

Example 3.4: Consider again predicate T as defined in example 3.2. 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: $E\_T'_{1,1}(x) \leftarrow P(x,y) \wedge \neg \delta P(x,y)$
$E\_T'_{1,2}(x) \leftarrow \iota P(x,y)$
$E\_T'_1(x) \leftarrow E\_T'_{1,j}(x)$ $j = 1, 2$

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 The Augmented Knowledge Base

Let K be a knowledge base. We call *augmented knowledge base*, and we denote it by A(K), to the knowledge base consisting of K, its transition rules and its insertion and deletion event rules. 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. In order 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 The 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* [FC85, MW88], 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*) [Tom88, Bry90, Dec90, KM90, GL90, GL91, TA91, TO92, AT92]. Inputs to the translator are a view definition, a view update request and the current knowledge base; and the output is a knowledge base update that satisfies the request. The method that we propose in this paper follows the translator approach.

Usually, there exist 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 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 between them will not be addressed in this paper (some comments on this problem can be found in [KM90]).

Moreover, we consider only translations that involve solely updates of the extensional part of the knowledge base, that is, insertions and deletions of ground facts of base predicates. For this reason, translations that involve updates of the intensional part of the knowledge base (deletion, insertion or modification of rules, insertion of ground facts of view predicates, etc.) are not considered here. Several authors have argued the suitability of translating views in this way (see for example [Dec90, KM90]).

For simplicity of presentation, we assume for the moment 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 (resp. deletion) corresponds to an event $\iota P(\mathbf{C})$ (resp. $\delta P(\mathbf{C})$), where $P'(\mathbf{C})$ is the fact that must hold (resp. must not hold) in the new state of the knowledge base.

A translation of an insertion $\iota P(\mathbf{C})$ (resp. 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 (resp. 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 $\iota P(\mathbf{C})$ corresponds to an insertion (resp. $\delta P(\mathbf{C})$ to a deletion), then $P(\mathbf{C})$ must not hold in the current knowledge base (resp. $P(\mathbf{C})$ must hold).

Our method is able to translate view update requests which 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 which 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 [Llo87], 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 will describe the Events Method for a particular kind of knowledge bases. More precisely, we will assume that all variables appearing in the body of a deductive rule appear also in its conclusion. That is, and following the terminology defined in section 3, we will consider that the knowledge base contains only rules that do not have an existential part ($E(P_i) = \varnothing$). For instance, the rule $P(x) \leftarrow Q(x) \wedge R(x,y) \wedge \neg S(y)$ does not satisfy this condition since $E(P) = R(x,y) \wedge \neg S(y)$.

Example 4.1: (adopted from [SK88]) Let K be a knowledge base with the following base and view predicates:

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 to this knowledge base are:

T.1	$Rr'_{1,1}(x) \leftarrow Alien(x) \wedge \neg\delta Alien(x) \wedge \neg Cr(x) \wedge \neg\iota Cr(x)$	
T.2	$Rr'_{1,2}(x) \leftarrow Alien(x) \wedge \neg\delta Alien(x) \wedge \delta Cr(x)$	
T.3	$Rr'_{1,3}(x) \leftarrow \iota Alien(x) \wedge \neg Cr(x) \wedge \neg\iota Cr(x)$	
T.4	$Rr'_{1,4}(x) \leftarrow \iota Alien(x) \wedge \delta Cr(x)$	
T.5	$Rr'_{2,1}(x) \leftarrow Cit(x) \wedge \neg\delta Cit(x)$	
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	$\iota Rr(x) \leftarrow Rr'_{1,j}(x) \wedge \neg Rr_2(x)$	$j = 2 \ldots 4$
I.4	$\iota Rr(x) \leftarrow Rr'_{2,2}(x) \wedge \neg Rr_1(x)$	
D.1	$\delta Rr(x) \leftarrow \delta Alien(x) \wedge \neg Cr(x) \wedge \neg Rr'_2(x)$	
D.2	$\delta Rr(x) \leftarrow Alien(x) \wedge \iota Cr(x) \wedge \neg Rr'_2(x)$	
D.3	$\delta Rr(x) \leftarrow \delta Cit(x) \wedge \neg Rr'_1(x)$	

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) $\cup$ {$\leftarrow \iota Rr$ (Mary)} succeed. This is shown in figure 4.1 (circled labels appearing in the left of a derivation are references to the rules of the method, defined in section 4.4).

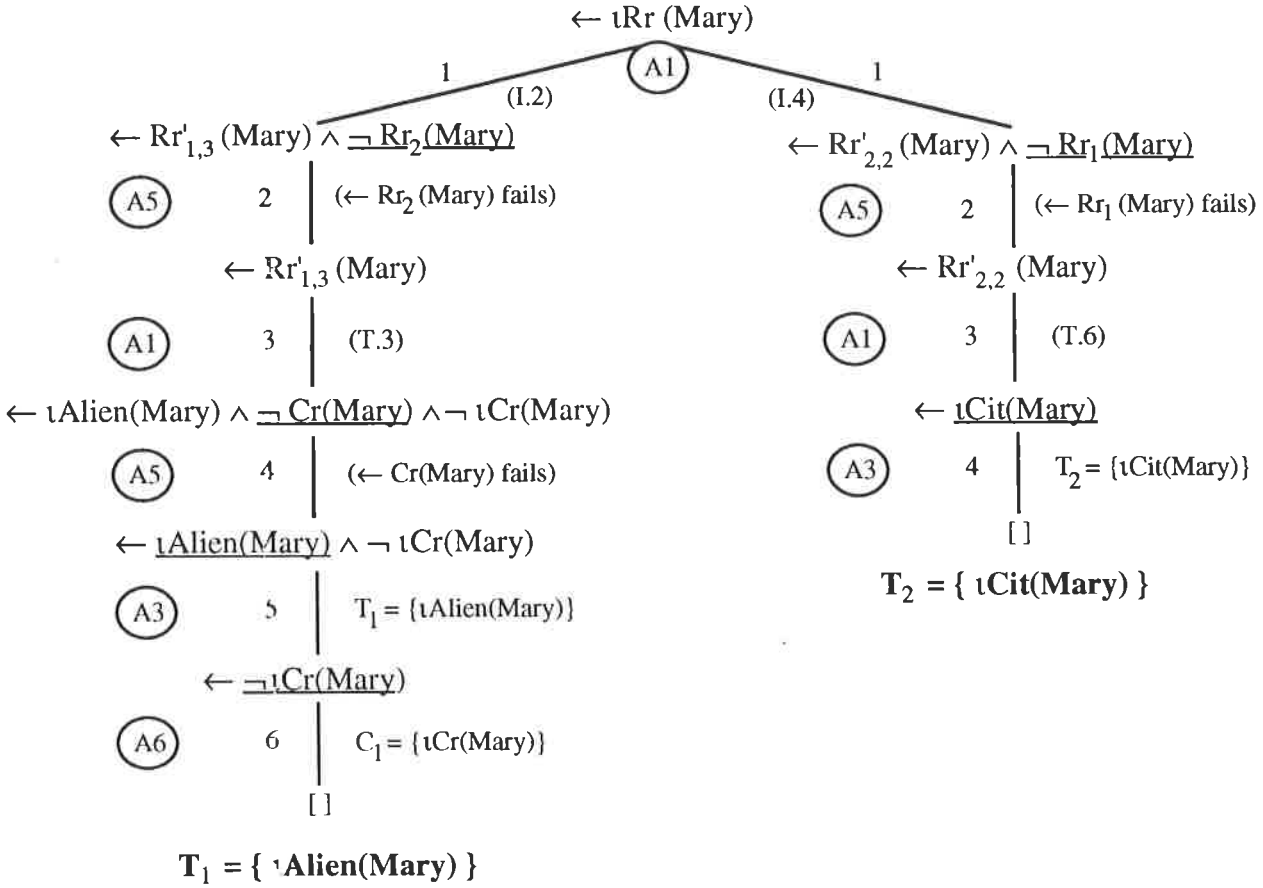


Fig. 4.1

Steps 1 to 4 in the left derivation are SLDNF resolution steps. At step 5, the selected literal is ιAlien (Mary), which is a positive base event. In order 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 ¬ιCr(Mary). In order to get success for this branch, ιCr(Mary) must fail, which implies that it must not belong to T. We use an auxiliary set C, which we call *condition set*, in order to check that later on, during the derivation process, the event ιCr(Mary) will not be included in T. Thus, in step 6, ιCr(Mary) is included in C.

In general, a condition set C is a set of base events that can not be included in the translation T. Due to the opposite meaning of T and C, before including a base event in T (resp. in C) we must check that it does not belong to the subset of C (resp. 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 nor in 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 that T_1={ιAlien(Mary)}. Then, 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 inserting the fact that Mary is registered as an alien.

In a similar way, the right derivation reaches the goal ←ιCit(Mary), that can be succeeded by including ιCit(Mary) in the input set (step 4). Thus, another possible translation that satisfies the request is T_2={ιCit(Mary)}.

Selecting clauses I.1 and I.3 in step 1 of the previous example 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 to delete Cr(Mary) which is not possible. Thus, no other translation can be obtained from them.

Example 4.2: Consider again the knowledge base defined in Example 4.1, 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) ∪ {←δRr (John)} succeed. One of these derivations is shown below:

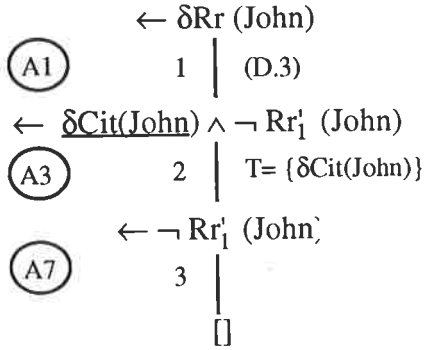


Fig. 4.2

Step 1 is an SLDNF resolution. At step 2, the selected base event δCit (John) is included in T. After this step, we get the goal ← ¬Rr'_1 (John). This derivation will succeed if we ensure that the subsidiary tree rooted at ←Rr'_1 (John) fails finitely. Part of this subsidiary tree is shown below:

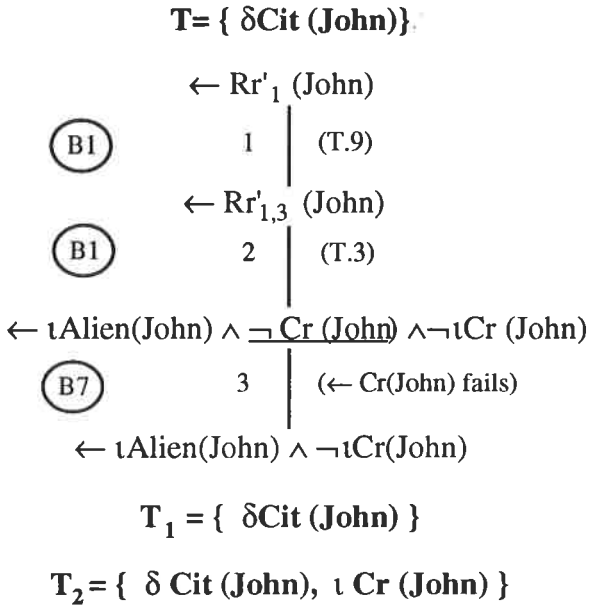


Fig. 4.3

Steps 1, 2 and 3 are SLDNF resolution steps. After this step, we get the goal ← ιAlien (John) ∧ ¬ιCr (John). Then, all possible ways in which this goal can fail must be taken into account. We will obtain as many translations as different ways to make this goal fail exist. In this case, we have two options: to add ιAlien (John) to the condition set C or to include ιCr (John) in the translation set T.

Selecting clauses T.7, T.8 and T.10 in the first step of the previous tree, 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 4.2 we get the empty clause and the derivation is finished obtaining two different translations: T_1 ={δCit(John)} and T_2={δCit(John), ιCr(John)}. Notice that T_2 is not a minimal translation, because there exist a subset of it (i.e. T_1) which is itself a translation. As we are interested only in minimal translations, T_2 would be refused.

4.3 General Case

In the previous section, we have 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. Then, the procedure for verifying that all conditions are satisfied is restricted to check 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. Then, 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. In order 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 4.3: 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.

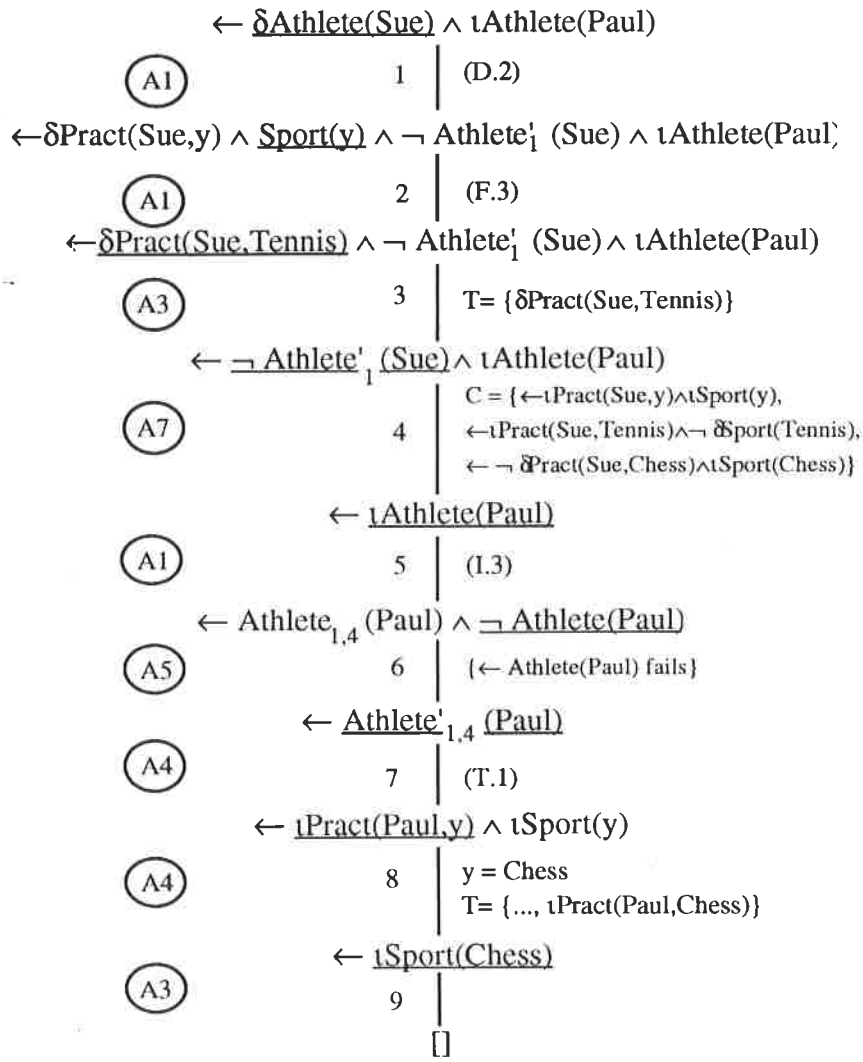
The current content of the knowledge base is:

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

Transition and event rules associated to 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	$\text{Athlete}'_1(x) \leftarrow \text{Athlete}'_{1,j}(x)$ $\quad j = 1\ldots4$
I.1..3	$\iota\text{Athlete}(x) \leftarrow \text{Athlete}'_{1,j}(x) \wedge \neg\text{Athlete}(x)$ $\quad j = 2\ldots4$
D.1	$\delta\text{Athlete}(x) \leftarrow \text{Pract}(x,y) \wedge \delta\text{Sport}(y) \wedge \neg\text{Athlete}'_1(x)$
D.2	$\delta\text{Athlete}(x) \leftarrow \delta\text{Pract}(x,y) \wedge \text{Sport}(y) \wedge \neg\text{Athlete}'_1(x)$

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) ∪ {← δAthlete(Sue) ∧ ιAthlete(Paul)} succeed. One of these derivations is shown below:



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

Fig. 4.4

Steps 1 to 7 correspond to steps already described in previous examples. Subsidiary derivation associated to the literal ¬Athlete'$_1$(Sue) (step 4) is not shown in the above figure, but it implies the inclusion of three conditions: ←ιPract(Sue,y) ∧ ιSport(y), ←ιPract(Sue,Tennis) ∧ ¬δSport(Tennis) and ←¬δPract(Sue,Chess)∧ ιSport(Chess) into C.

After step 7 we get the goal ← ιPract(Paul,y) ∧ ιSport(y). This goal can be succeeded if we instantiate the variable 'y' to some value 'Y' for which ιPract(Paul,Y) and ιSport(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 ιPract(Paul,Chess) is included in the translation set T.

At step 9, the selected literal ιSport(Chess) must be added to T. However, we have to verify that this inclusion satisfies C = {←ιPract(Sue,y) ∧ ιSport(y), ←ιPract(Sue,Tennis) ∧ ¬δSport(Tennis), ←¬δPract(Sue,Chess) ∧ ιSport(Chess)}. This verification is performed by ensuring that, for each condition C_i of C, the SLDNF search space of A(K) ∪ 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. We show in the next figure the necessity of adding new base events to T in order to maintain satisfaction of the condition ← ¬δPract(Sue,Chess) ∧ ιSport(Chess). Consistency of the other conditions is not affected by the inclusion of ιSport(Chess) into T.

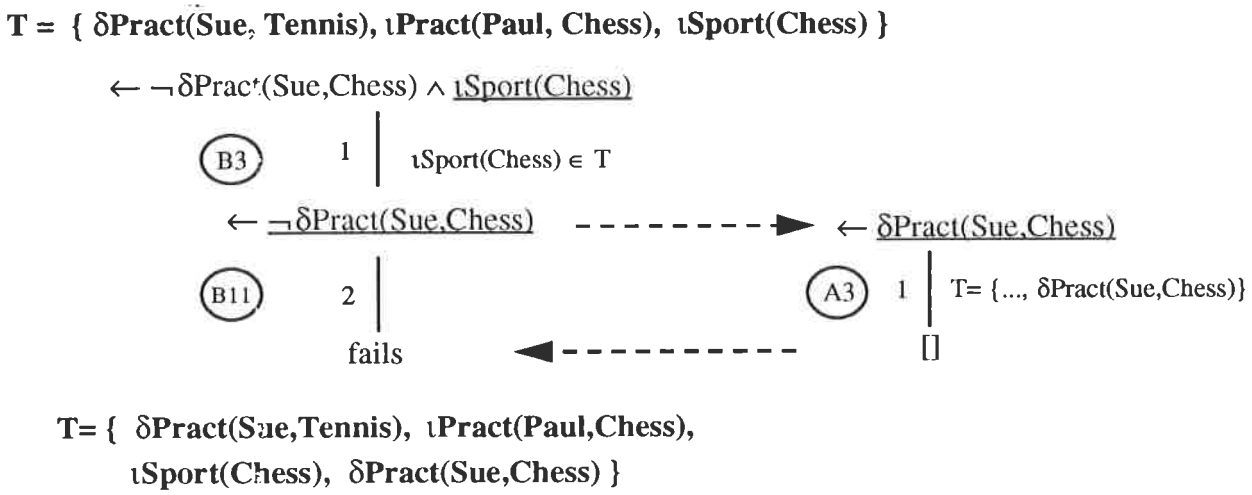


Figure 4.5

Steps 1 is an SLDNF resolution step. At step 2, the selected literal is ¬δPract(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 the above figure and it causes the inclusion of δPract(Sue,Chess) in the translation set T.

Then, we get the empty clause in the initial derivation of figure 4.4 and, hence, the translation process finishes obtaining the translation T={δPract(Sue,Tennis),ιPract(Paul,Chess),ιSport(Chess), δPract(Sue,Chess)}. Notice that the addition of ιSport(Chess) to T in step 9 of figure 4.4 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 ← ¬δPract(Sue,Chess) ∧ ιSport(Chess)) and to perform necessary repair actions (which motivates the inclusion of δPract(Sue,Chess) in T).

Selecting rule D.2 in step 1 of figure 4.4, we would obtain another translation T_2 = {δSport(Tennis), ιPract(Paul,Chess), ιSport(Chess), δPract(Sue,Chess)}. This derivation is not shown in the tree above.

4.4 Formalization of the 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 conditions that deductive rules must satisfy is allowedness.

As has been pointed out in the 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 exists a constructive derivation from ($\leftarrow$u $\varnothing$ $\varnothing$) 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 in order to satisfy the view update request. In order 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 can not 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: δPract, ιPract, δSport, ιSport
- derived events: δAthlete, ιAthlete
- 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. We summarize in figure 4.6 which rule is applied in each case:

Constructive	Positive	Negative
Old base pred. Old derived pred.	A1	A5
Base event	A2, A3, A4	A6
Derived event New predicate	A1	A7

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

Fig. 4.6

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 [Llo87] is a sequence:

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

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:

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 $G_{i+1}=G_i \backslash L_j$, $T_{i+1} =T_i$ and $C_{i+1}=C_i$.

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

A4) If L_j is a non ground positive base event, σ is a substitution such that:
1) If L_j is a non ground positive base event 'ιP', then σ is a substitution of variables of P such that $P\sigma$ does not hold in the current knowledge base.
2) If L_j is a non ground positive base event 'δP', then σ is a substitution of variables of P such that $P\sigma$ holds in the current knowledge base.
and there exists a consistency derivation from $(C_i\ T_i \cup \{L_j\sigma\}\ C_i)$ to $(\{\}\ T'\ C')$, then $G_{i+1}=G_i\sigma \backslash L_j\sigma$, $T_{i+1}=T'$ and $C_{i+1}= C'$

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

A6) If L_j is a ground negative base event and $\neg L_j \notin T_i$, then $G_{i+1} = G_i \backslash L_j$, $T_{i+1} =T_i$ 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 exists a consistency derivation from $(\{\leftarrow \neg L_j\}\ T_i\ C_i)$ to $(\{\}\ T'\ C')$, then $G_{i+1}=G_i \backslash L_j$, $T_{i+1}=T'$ 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, in order 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 $C_i = \varnothing$, 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, then, we have to instantiate it (substeps A4.1 and A4.2). In A4.1) this is done either by assigning default values or by asking the user. In A4.2) 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 in order to ensure that they will not be included in the translation set afterwards. In rule A7), we get 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:

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

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 $F_{i+1}=H_i \backslash L_j \cup F'_i$, $T_{i+1}=T_i$ 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 the goal $\leftarrow\neg L_j$ fails finitely, then $F_{i+1} = H_i \backslash L_j \cup F'_i$, $T_{i+1}= T_i$ and $C_{i+1}= C_i$.

B8) If L_j is a base or derived predicate, negative and old, and there exists 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 ground base event, $\neg L_j \notin T_i$ and k >1, then $F_{i+1} = H_i \backslash L_j \cup F'_i$, $T_{i+1} = T_i$ and $C_{i+1} = C_i$.

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

B13) If L_j is a negative, new or derived event, predicate, and there exists 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 in order 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 exists a constructive derivation for the negation of the selected literal. In rules B13) and B12) the current branch will be dropped or not depending on whether exists 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.

We will show in section 6 that our method obtains all minimal translations that satisfy a given update request. However, as we have seen in example 4.2, 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 exists 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 in order 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 (eventually empty) of translations that do not necessarily correspond to a subset of the translations obtained by view updating alone. This approach is explained in 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, afterwards, we would 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. This approach is explained in 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 ιIcn 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 ιIc as: $\iota Ic \leftarrow \iota Ic1(\mathbf{x_1}), \ldots, \iota Ic \leftarrow \iota Icn(\mathbf{x_n})$, (where $\mathbf{x_i}$, i =1...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 exists a constructive derivation (as formally defined in section 4.4) from $(\{\leftarrow u \wedge \neg \iota Ic\}\ \varnothing\ \varnothing)$ to ([] T C). During this derivation, the literal $\neg \iota Ic$ will be selected. As it corresponds to a negative derived event predicate (rule A7), the next step of the constructive derivation will be reached if there exists a consistency derivation from $(\{\leftarrow \iota Ic\}\ T'\ C')$ to ({ } T C), where T' and C' are the subsets of T and C 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 5.1: Consider again the knowledge base defined in Example 4.3, 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 to 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)

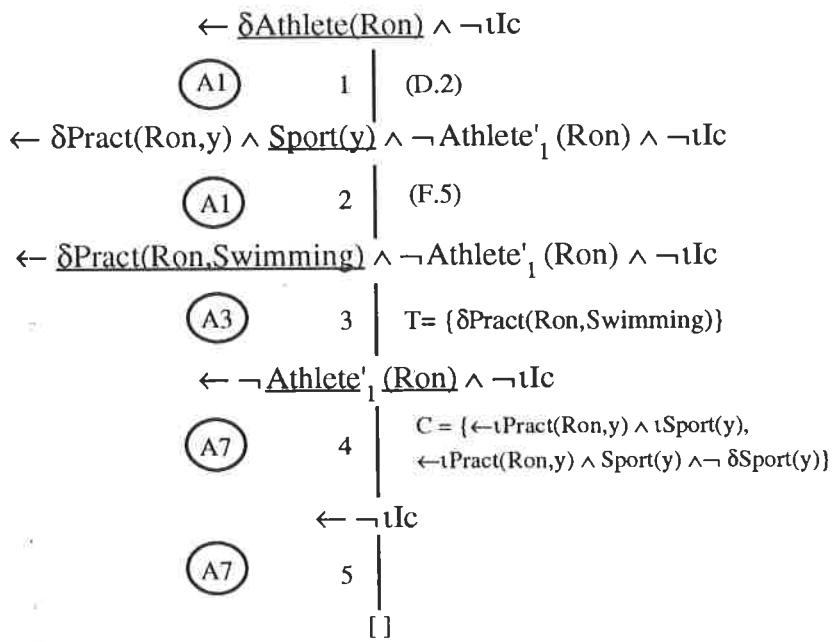
T.10 $Ic1'_{1,2} \leftarrow \neg$Pract(Ron,Swimming) $\wedge \neg\iota$Pract(Ron,Swimming) $\wedge \delta$Pract(Ron,Climbing)

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 $\iota Ic1 \leftarrow Ic1'_{1,j}$ $\quad j = 2...4$

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 exists a constructive derivation from ($\{\leftarrow\delta$Athlete(Ron) $\wedge \neg\iota$Ic$\}$ $\varnothing$ $\varnothing$) to ([] T C). Part of this derivation is shown in figure 5.1.



T= { δPract(Ron,Swimming), ιPract(Ron,Climbing), δSport(Climbing) }

Fig. 5.1

Steps 1 and 2 are SLDNF resolution steps. At step 3, the selected literal δPract(Ron,Swimming) is included in the translation set T. At step 4, there exists a consistency derivation associated to the goal ←Athlete'$_1$(Ron). This derivation is not shown in the above figure, but it implies the inclusion of two conditions: ←ιPract(Ron,y) ∧ ιSport(y), ←ιPract(Ron,y) ∧ Sport(y) ∧ ¬δSport(y) into the condition set C. At step 5, the next goal is reached because there exists a consistency derivation associated to ←ιIc. Part of this derivation is shown below:

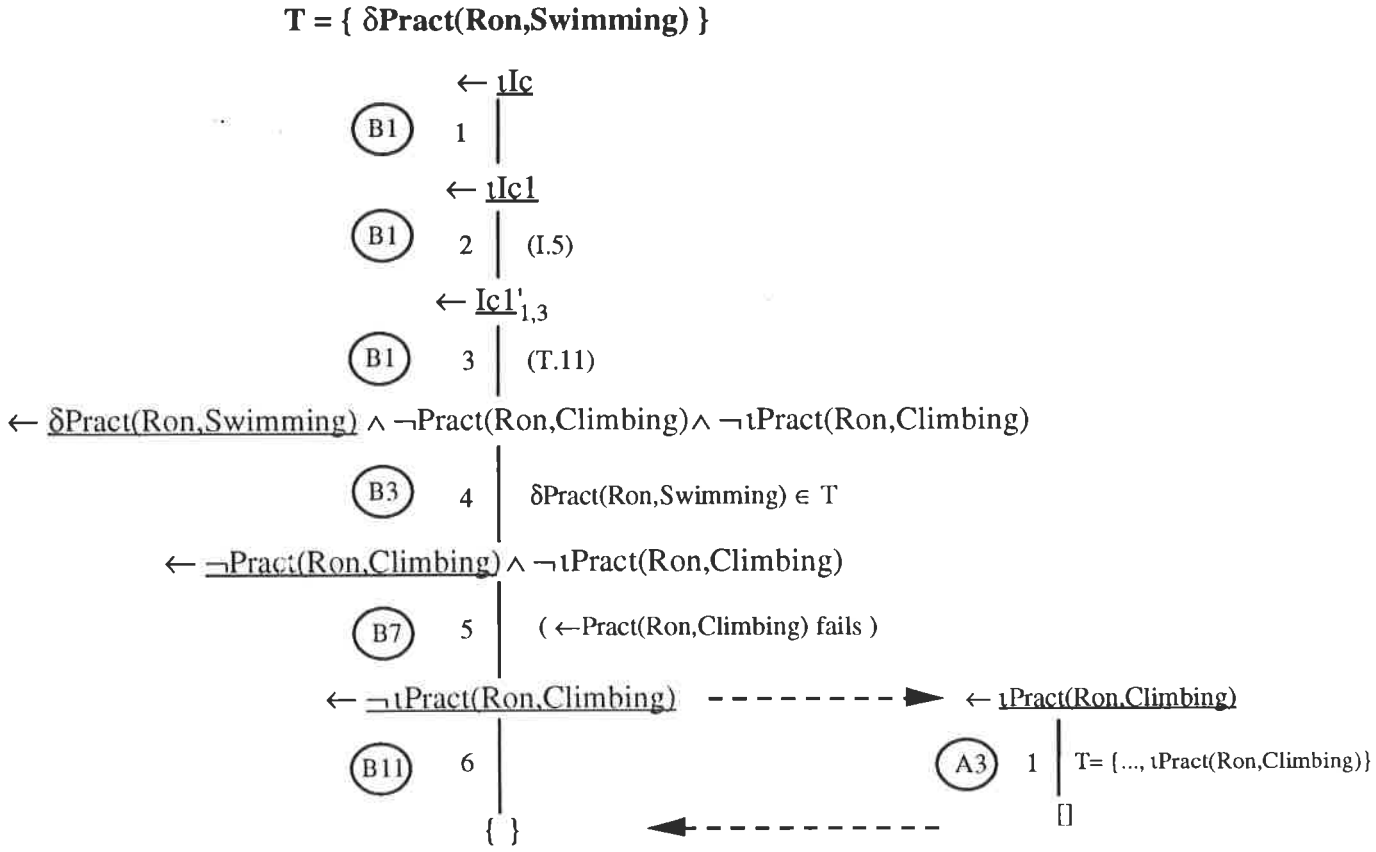


Fig. 5.2

Steps 1 to 5 are SLDNF resolution steps. At step 6, this branch can be removed because there exists a constructive derivation for the negation of the selected literal. This derivation is shown in the right part of the above figure.

At step 1 of this constructive derivation ιPract(Ron,Climbing) is included in T. Note that this inclusion is necessary in order not to violate Ic1. Moreover, we have to verify that this inclusion satisfies C = {←ιPract(Ron,y) ∧ ιSport(y), ←ιPract(Ron,y) ∧ Sport(y) ∧ ¬δSport(y)}. In this case, the event δSport(Climbing) must be included in T in order to maintain consistency of the second condition (this is done in a similar way to figure 4.5).

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 (this derivations are not shown in the tree above).

Then, in the initial derivation of figure 5.1, we get the empty clause and the derivation is finished obtaining T = {δPract(Ron,Swimming), ιPract(Ron,Climbing), δSport(Climbing)}, which satisfies the view update request and maintains knowledge base consistency. Selecting the clause D.1 at step 1 of figure 5.1 we would obtain another solution T_2 = {δSport(Swimming)}.

As we have shown in the previous example, in our method integrity constraints maintenance is incorporated into the translation process. 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 could also 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 5.1. In the first step, we would translate the view update request into base updates as explained in section 4, obtaining two translations: T_1 = {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 can also 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 in [Oli91] (which is also based on the concept of event) or any other method for integrity constraints checking.

As it has been pointed out recently [CW90, ML91], 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: $T_1 = \{\delta Pract(Ron,Swimming)\}$ and $T_2 = \{\delta Sport(Swimming)\}$. 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 nor Climbing. Thus, this translation would be rejected and the only solution $T_2 = \{\delta Sport(Swimming)\}$ would be obtained following the integrity constraints satisfaction approach.

In order to finish this section, we would like to point out here 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 [Oli91] for the details of transition and event rules in this case.

6 Soundness and Completeness of the Events Method

In this section, we summarize the main results concerning the soundness and completeness of the Events Method. The detailed proofs are given in Appendix B.

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 (that is, 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 exists an SLDNF refutation of $A(K) \cup T \cup \{\leftarrow u\}$.

As can be seen, the lemma relates the constructive derivation ($\{\leftarrow u\}$ $\varnothing$ $\varnothing$) to ([] T C) of our method to an SLDNF refutation of $A(K) \cup T \cup \{\leftarrow u\}$. Given that SLDNF resolution has been proved sound [Cla78], then the following theorem follows:

Theorem 6.1: *(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 does also exist in the reverse direction. This is stated in the following theorem (see Appendix B):

Theorem 6.2: 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, gives a refutation of $A(K) \cup T \cup \{\leftarrow u\}$. Then, there exists a constructive derivation from ($\{\leftarrow u\}$ $\varnothing$ $\varnothing$) to ([] T C).

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

Theorem 6.3: *(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 goal $\{\leftarrow u\}$. Then, there exists a constructive derivation from ($\leftarrow u$ $\varnothing$ $\varnothing$) to ([] T C) for any translation T that satisfies that u is a logical consequence of $comp(A(K) \cup T)$.

In Appendix C we review the cases for which completeness results of SLDNF-resolution have been proved.

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 exists a constructive derivation from ($\leftarrow u$ $\varnothing$ $\varnothing$) 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.

7 Other Kinds of Updates

In the previous sections, we have faced the problem of view updating and discussed how it could be combined with integrity checking and integrity maintenance. We have also shown how to apply integrity maintenance when considering updates of base facts. However, knowledge bases also allow other kinds of updates: i.e. updates of deductive or integrity rules, which may also 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 will be 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 ιP 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 $\iota P(x) \leftarrow P'(x)$. Transforming, as indicated in section 3, P'(x) into its equivalent set of rules, we get the insertion event rules that give the ιP facts produced during the transition. Note that, in this particular case, we would not need to maintain consistency since, being P 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, ιP facts produced by the update are given by : $\iota P(x) \leftarrow P'_k(x) \wedge \neg P(x)$, and transforming $P'_k(x)$ into its equivalent set of transition rules we get the insertion event rules that give the ιP 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 exists a constructive derivation from, ({←¬ιIc} ∅ ∅) to ([] T C). If T = ∅, 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 7.1: 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) ← Cont (x)
IC.1	Ic1 (x) ← Works (x) ∧ Unemp (x)
T.1	$Ic'_{1,3}$ (x) ← ιWorks (x) ∧ Unemp (x) ∧ ¬δUnemp (x)
I.1	ιIc1 (x) ← $Ic1'_{1,3}$ (x)
I.2	ιIc ← ιIc1 (x)

Let the update be the insertion of the deductive rule:

Works (x) ← 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 $Works_2$. The transition rules associated to this predicate are:

T.2	$Works'_{2,1}$ (x) ← Grant (x) ∧ ¬δGrant (x)
T.3	$Works'_{2,2}$ (x) ← ιGrant (x)

The new insertion event rules are:

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

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

Note that the insertion of this deductive rule violates integrity constraint Ic1. Then, some updates of the base facts must be performed in order to maintain knowledge base consistency. T will be a solution if, with these new rules, there exists a constructive derivation from ($\{\leftarrow\neg\iota Ic\}\ \varnothing\ \varnothing$) to ([] T C). This derivation is shown in figure 7.1.

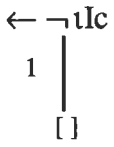
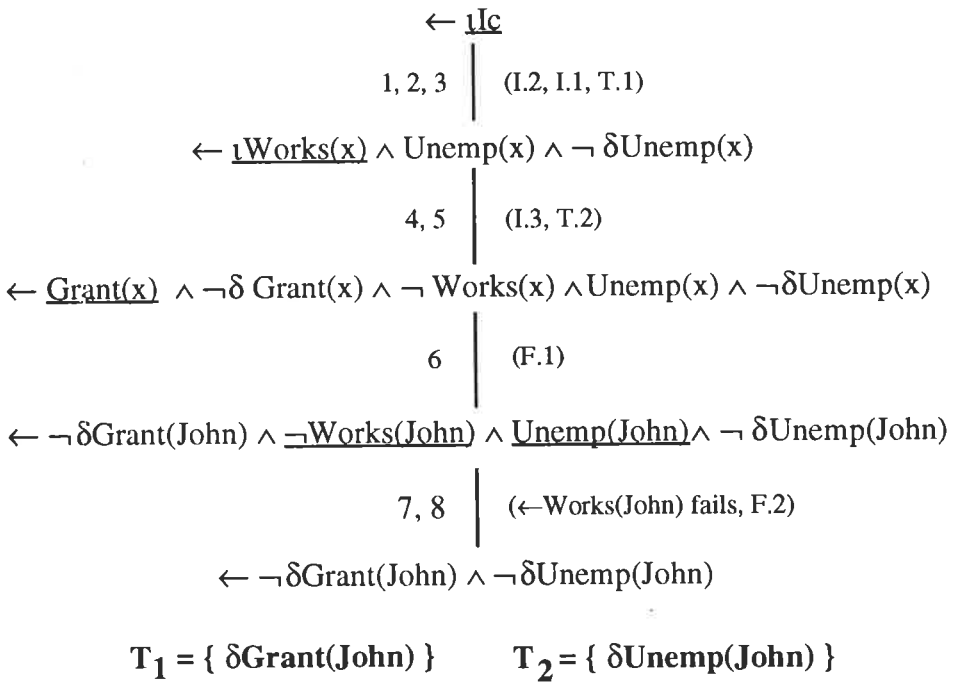


Fig. 7.1

The empty clause is reached because there exists a consistency derivation associated to $\leftarrow\iota Ic$. Part of this derivation is shown in figure 7.2.



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

Fig. 7.2

To save space we sometimes group two or more steps into a single one like in the above figure for 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 $\leftarrow \neg\delta Grant(John) \wedge \neg\delta Unemp(John)$ can fail must be studied. In this case, we will obtain two solutions: $T_1 = \{\delta Grant(John)\}$ and $T_2 = \{\delta Unemp(John)\}$. Notice that both solutions maintain knowledge base consistency when inserting the rule Works (x) ← 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 again the same knowledge base of example 7.1 and assume that its current content is exactly the same. Let the update request be the insertion of the integrity constraint: $Ic2(x,y) \leftarrow Assig(x,y) \wedge \neg Dept(y)$. In our method, T will be a solution if, considering the transition and event rules associated to Ic2, there exists a constructive derivation from $(\{\leftarrow \neg \iota Ic\}\ \varnothing\ \varnothing)$ to $([]\ T\ C)$. In this case, our method would obtain 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 would first determine the input events produced by each update and then apply the integrity maintenance approach as described in sections 5.1, 7.1 and 7.2. If there exists some translation, the knowledge base is updated and, eventually, the transition and event rules are modified.

8 Additional Features of Our Method

8.1 Modification Requests

In order to simplify the presentation, we have considered in previous sections 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 one 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 allow 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) afterwards. In our method, T will be a solution if there exists a constructive derivation from ({←δEdm (John,D1,Mary) ∧ ιEdm (John,D1,Sue)} ∅ ∅) to ([] T C). In this case, we would obtain the solution T = {δDm (D1, Mary), ιDm (D1,Sue)}.

The second approach for dealing with modification requests would consist on adapting the framework described in [UO92, Urp93]. In this proposal it is assumed that each predicate has a non-null vector of arguments that form the key for that predicate. Then, the kind of events considered in [Oli91] is extended by considering not only insertions and deletions, but also modifications of base and derived predicates, and deriving accordingly the modification event rules. Incorporating the concept of key (not considered in this paper), we could adapt our method to use the modification event rules in order to handle modifications of view and base predicates.

8.2 Evaluable Predicates

In order to simplify the presentation, we have considered that all predicates appearing in the body of deductive rules were ordinary (that is, base or derived). However, we can also 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 [Ull88]. 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

We have proved in section 6 that the Events Method is sound and complete. However, we should note that, 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 in order to avoid this problem.

In general, loop checking techniques [BAK91, Bol93] 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 would prevent our method to enter into an infinite loop and, thus, it would always terminate.

8.4 Preventing Side Effects

Due to the deductive rules, non requested updates may be induced on some derived predicates when performing an update. We say that a *side effect* occurs when this happens. The Events Method is able to prevent side effects, if desired. They can be prevented 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 $\neg\iota P_i$ (resp. $\neg\delta Q_j$) is added to the root goal for each fact P_i (resp. Q_j) for which we want induced insertions (resp. deletions) not to occur. Then, the root goal will have 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. Then, 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 case of side effect involving inconsistency predicates.

Being able to prevents side effects is particularly important because it allows us to follow the *constant complement approach* [BS81] in view updating. This elegant approach consists on defining for each view a complement which 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, for instance in order 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 for dealing with this problem.

As we have seen in section 2, we associate to 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. Then, 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 described in section 3 in order 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.

Then, in order 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 will contain base facts updates needed in order to repair all integrity constraints violations.

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

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

We can use our method for restoring consistency of this knowledge base. In this case, predicate Ic is defined as $Ic \leftarrow \iota c1(x)$, because there is only one integrity constraint. Then, we have to derive its transition and event rules:

$Ic'_{1,1} \leftarrow Ic1(x) \wedge \neg\delta Ic1(x)$
$Ic'_{1,2} \leftarrow \iota Ic1(x)$
$Ic'_1 \leftarrow Ic'_{1,j} \qquad j = 1, 2$
$\iota Ic \leftarrow Ic'_{1,2} \wedge \neg Ic$
$\delta Ic \leftarrow \delta Ic1(x) \wedge \neg Ic'_1$

T will be a solution if there exists a constructive derivation from $(\{\leftarrow\delta Ic\}\ \varnothing\ \varnothing)$ to $([]\ T\ C)$. In this example, we would 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 paper, we have assumed that the knowledge base is consistent before the update in order 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. Then, the Events Method would be applied in the same way as defined in sections 5.1 and 7, with the only difference that the literal $\neg\iota Ic$ would be replaced by δIc 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 [Tom88] 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. Then, these annotations are used for guiding the translation process. The general idea is to explore only the branches which are permitted by the annotations.

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

8.7 An Optimization of the Events Method

In this paper, we have presented a version of the Events Method where the translation process is completely performed at execution time, 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* [LlS91], thus increasing efficiency of our method.

In the Events Method, we could partially evaluate the intensional part of the augmented knowledge base (that is, deductive, transition and event rules) with respect to the update request, thus obtaining at compile time a set of equivalent rules which 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 for solving the view update problem and methods which 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 within the first group. In section 10, we will 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. In order to clarify the contribution of our method, we select here a representative set of existing methods and provide a comparison with them (see also [Ten92]). All these methods deal with the same class of knowledge bases and view update requests than our. However, none of them is able to handle transition integrity constraints nor prevention of side effects on other views.

9.1 Decker's Method [Dec90]

Let D be a deductive database and u an update request. In Decker's method view updates for delete requests are obtained 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 input clause (negative literals are treated as subsidiary insert requests).

In insert requests, Decker needs to define the concept of *view update trees* in order to obtain the translations impeded by the selection function employed in the SLDNF resolution. Informally, the basic idea of these trees consists on 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 differences between this method and the method proposed here. First, the main problem present in Decker's method 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).

In Decker's method two different solutions would be obtained: 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, Decker's method must be iterated a number of times in order 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 proposes 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. Third, this method only allows integrity checking. On the contrary, our method can also deal with integrity constraints maintenance.

9.2 Guessoum and Lloyd's Method [GL90, GL91]

In the same way as Decker's method, Guessoum and Lloyd's method uses SLDNF resolution in order to obtain the translations that satisfy a view update request. In a first version of the method [GL90] procedures for deleting an atom from a normal program and inserting an atom into a normal program are presented. In a later version [GL91], these procedures are 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 the knowledge base, there exists an SLDNF refutation associated to the view update request.

It is not difficult to note that this method and Decker's method are very close. Nevertheless, there are some differences. A first one is that Guessoum and Lloyd's method checks the updated knowledge base in order to determine whether a solution is valid. Because of this, this method does not present the problem of solutions invalidated due to negation. However, we would like to remark that 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 one. For insert requests, Decker's method needs to define "view updates trees" in order to obtain all translations impeded by the selection function used in the SLDNF resolution. However, Guessoum and Lloyd do not propose any alternative to this problem. Then, in general they cannot obtain all translations that satisfy an insert request. Our method always gets all valid solutions

Finally, in some cases this method should also be iterated a number of times in order to obtain a solution without knowing how many iterations should be performed. On the contrary, the Events Method does not needs to be iterated.

9.3 Kakas and Mancarella's Method [KM90]

This method studies the view update problem within an elegant abductive approach. It distinguishes clearly two steps to obtain the translations. In the first step, the update request is translated into several sets Δ_i. Each Δ_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 Δ_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 Δ_i. Then, some

Δ_i expressing requirements that the current database already satisfies may be obtained. Because of this, the number of Δ_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 Δ_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 that fact 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 would be obtained: $T_1=\{\delta Q\}$ and $T_2=\{\delta R\}$. Notice that they correspond to the valid solutions.

Another important difference is the way of handling integrity constraints maintenance. In Kakas and Mancarella's method integrity constraints can be dynamically maintained 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 constraint:

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

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 would obtain the solution $T=\{\iota Q(A), \iota R(A)\}$ which satisfies both the update request and the integrity constraint.

9.4 Atzeni and Torlone's Method [AT92, TA91]

Atzeni and Torlone's method deals with view updating in definite deductive databases. That is, databases where view predicates can only be defined by means of function free Horn rules, without negation. Despite this expressive limitation, we would like to remark some interesting features of this proposal.

Their main contribution is to provide a formalization of a declarative semantics of updates of facts. Among the possible translations of an insertion they consider as important the *minimal potential results*, which include only the information that is strictly needed to satisfy the insert request. Then, they distinguish between deterministic and non deterministic updates depending on the existence of a minimal potential result. Semantics for deletions is defined in a similar way.

They do also propose 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 [TA91], this method is 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 Atzeni and Torlone's method and the Events Method. First, they consider 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 where 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 apply the integrity checking approach when dealing with functional dependencies, while the Events Method follows the integrity constraints maintenance approach. Next example (adopted from [TA91]) 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, then, Smith cannot teach both CS101 and CS202. However, in our method the translation T = {δTeaches(Smith,CS101), ιTeaches(Smith,CS202), ιAttends(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. We would like to remark first that none of these methods deals with insertions and deletions of deductive rules nor with insertions of integrity constraints, while we do.

10.1 Moerkotte and Lockemann's Method [ML91]

Moerkotte and Lockemann's approach 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 which must be appended to the original transaction in order to regain consistency.

Moerkotte and Lockemann's method distinguishes clearly three steps to obtain the repairs. In a 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 a second step, a set of *causes* is generated from the symptoms. Causes correspond to base (stored) facts that give raise to a symptom. Finally, the causes are transformed into *repairs* by syntactic modification.

Moerkotte and Lockemann's method inspired our approach to integrity constraints maintenance. However, their method is restricted to definite deductive databases and they consider only flat transactions, that is, 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.

10.2 Ceri et al.'s Method [CFPT92]

The approach taken by Ceri et al. is to provide automatic repair of inconsistent databases by using *production rules*. For each constraint, production rules are used in order 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 in [CW90]. In this paper, 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, with what the process of restoring knowledge base consistency could not be completely automatic. Ceri et al. [CFPT92] do not have this problem because in their proposal production rules can be automatically generated.

Ceri et al.'s 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 method the analysis is completely performed at execution time, although we could do some preparatory work at compile time by means of partial evaluation (see section 8.6).

A second important difference is that Ceri et al.'s method 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 paper, we have presented 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 performs the translation of a view update request into appropriately updates of the underlying base facts. We have proved soundness and completeness in this case.

Our method handles uniformly both insert and delete requests. Complex updates, such as mixed multiple updates and modifications, can also be requested. We have presented several important additional features of our method like allowing prevention of side effects on other views, repairing inconsistent knowledge bases and maintenance of 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 interest in this paper 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 [Ten92], but we plan to continue working with the aim of developing 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 have 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.

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 ... m and j = 2 ... 2^{ki}

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

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

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

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

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

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

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 $\neg([Q_h(\mathbf{x_h}) | \neg Q_h(\mathbf{x_h})] \wedge \beta)$ 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 ... m and j = 2 ... 2^{ki}

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

(A.4) $P'_{i,j}(\mathbf{x}) \leftrightarrow [Q_1(\mathbf{x_1}) \wedge \neg\delta Q_1(\mathbf{x_1}) | \neg Q_1(\mathbf{x_1}) \wedge \neg\iota Q_1(\mathbf{x_1})] \wedge \ldots \wedge$
$[Q_q(\mathbf{x_q}) \wedge \neg\delta Q_q(\mathbf{x_q}) | \neg Q_q(\mathbf{x_q}) \wedge \neg\iota Q_q(\mathbf{x_q})] \wedge E(P'_{i,j})$

On the other hand we have:

$P_i(\mathbf{x}) \leftrightarrow U(P_i) \wedge E(P_i)$
$P_i(\mathbf{x}) \leftrightarrow U(P_i) \wedge E\_P_i(\mathbf{x})$
(A.5) $P_i(\mathbf{x}) \leftrightarrow [Q_1(\mathbf{x_1}) | \neg Q_1(\mathbf{x_1})] \wedge \ldots \wedge [Q_q(\mathbf{x_q}) | \neg Q_q(\mathbf{x_q})] \wedge E\_P_i(\mathbf{x})$

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

(A.6) $\iota P(\mathbf{x}) \leftarrow [Q_1(\mathbf{x_1}) \wedge \neg\delta Q_1(\mathbf{x_1}) | \neg Q_1(\mathbf{x_1}) \wedge \neg\iota Q_1(\mathbf{x_1})] \wedge ... \wedge$
$[Q_q(\mathbf{x_q}) \wedge \neg\delta Q_q(\mathbf{x_q}) | \neg Q_q(\mathbf{x_q}) \wedge \neg\iota Q_q(\mathbf{x_q})] \wedge E(P'_{i,j}) \wedge$
$\neg P_1(\mathbf{x}) \wedge ... \wedge \neg P_{i-1}(\mathbf{x}) \wedge$
$\neg([Q_1(\mathbf{x_1}) | \neg Q_1(\mathbf{x_1})] \wedge ... \wedge [Q_q(\mathbf{x_q}) | \neg Q_q(\mathbf{x_q})] \wedge E\_P_i(\mathbf{x})) \wedge$
$\neg P_{i+1}(\mathbf{x}) \wedge ... \wedge \neg P_m(\mathbf{x})$

from where we can remove literals $[Q_1(\mathbf{x_1}) | \neg Q_1(\mathbf{x_1})] \wedge ... \wedge [Q_q(\mathbf{x_q}) | \neg Q_q(\mathbf{x_q})]$, 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) $P'_i(x) \leftrightarrow P'_{i,1}(x) \vee ... \vee P'_{i,2^k i}(x)$ and replacing above:

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

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

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

Assume $P_i(x) \leftrightarrow L_1 \wedge ... \wedge L_n$, 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) $P'_{i,1}(\mathbf{x}) \leftrightarrow U(P_i) \wedge [\neg \delta Q_1(x_1) | \neg \iota Q_1(x_1)] \wedge ... \wedge [\neg \delta Q_q(x_q) | \neg \iota Q_q(x_q)] \wedge E(P'_{i,1})$

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) $\delta P(\mathbf{x}) \leftarrow P_i(\mathbf{x}) \wedge [\delta Q_j(x_j) | \iota Q_j(x_j)] \wedge \beta \wedge \alpha$ 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 $P_i(\mathbf{x}) \rightarrow U(P_i)$. In (A.15) literals L_j of $P_i(\mathbf{x})$ and β can also be removed, since $[\delta Q_j(x_j) | \iota Q_j(x_j)] \rightarrow L_j$, $\delta Q_j(x_j) \rightarrow \neg \iota Q_j(x_j)$ and $\iota Q_j(x_j) \rightarrow \neg \delta Q_j(x_j)$ and we get rules (19).

In rules (A.16) replacing β and distributing $\wedge$ over $\vee$ we get:

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

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

Appendix B - Soundness and Completeness of the Events Method

In this Appendix we prove the soundness and the completeness of our method [Ten92].

a) Soundness

In order to prove soundness of the Events Method we need to define the concepts of constructive and consistency derivations of level k ([1]).

Definition: Let G be a goal, T and T' translation sets and C and C' condition sets. A *consistency derivation of level 0* from (G T C) to ({} T' C') is a consistency derivation that does not call any constructive derivation nor any consistency derivation.

Definition: Let G be a goal, T and T' translation sets and C and C' condition sets. A *constructive derivation of level 0* from (G T C) to ([] T' C') is a constructive derivation that does not call any consistency derivation, or it calls only consistency derivations of level 0.

Definition: Let G be a goal, T and T' translation sets and C and C' condition sets. A *consistency derivation of level k+1* from (G T C) to ({} T' C') is a consistency derivation that calls some constructive or consistency derivation of level k.

Definition: Let G be a goal, T and T' translation sets and C and C' condition sets. A *constructive derivation of level k+1* from (G T C) to ([] T' C') is a constructive derivation that calls some consistency derivation of level k+1.

Let u be an update request. Lemma 1 states that there exists an SLDNF refutation of $A(K) \cup T \cup \{\leftarrow u\}$ for every translation T obtained by our method.

Lemma 1: Let K be a knowledge base, A(K) the augmented knowledge base, u an update request and T a minimal translation such that there exists a constructive derivation from $(\leftarrow u \varnothing \varnothing)$ to ([] T C). Then, there exists an SLDNF refutation of $A(K) \cup T \cup \{\leftarrow u\}$.

Proof: We have to proof that the steps used in constructive and in consistency derivations correspond to SLDNF resolution steps, where clauses in $A(K) \cup T$ act as input clauses. The proof is by induction on the level k of these derivations.

Let G be a goal, T and T' translation sets and C and C' condition sets and suppose that $k=0$. We first prove that a consistency derivation corresponds to a finitely failed SLDNF tree. This result is used afterwards to prove that a constructive derivation corresponds to an SLDNF refutation.

---

[1] Note that the concept of level is different to the concept of rank of an SLDNF derivation as defined by Lloyd [Llo87]

(1) Let CS be a *consistency derivation* of level 0 from (G T C) to ({} T' C'). Then, the SLDNF tree of $A(K) \cup T' \cup \{G\}$ fails finitely[1]. Note that these derivations do not modify the translation set T, and therefore, T=T'.

- Steps B1), B2), B7) and B8) are SLDNF resolution steps where A(K) acts as input set.
- Steps B3), B4), B5), B6), B9) and B10) correspond to an SLDNF resolution step where T' acts as input set.

(2) Let CT be a *constructive derivation* of level 0 from (G T C) to ([] T' C'). Then, there exists an SLDNF refutation of $A(K) \cup T' \cup \{G\}$.

Case1: No consistency derivation is called.

- Steps A1) and A5) are SLDNF resolution steps where A(K) acts as input set.
- Steps A2), A3) and A6) correspond to an SLDNF resolution step where T' acts as input set. Note that, in this case, no consistency derivation is performed in case A3).
- In step A4), the selected base event, once fully instantiated, is included in the translation set. Then, this is a resolution step where T' acts as input set.

Case2: Some consistency derivations of level 0 are called.

- Steps A1) and A5) are SLDNF resolution steps where A(K) acts as input set.
- Steps A2) and A6) correspond to an SLDNF resolution step where T' acts as input set.
- Let L_j be the selected literal and C_i be the condition set when step A3) or A4) is applied. We get the next goal in the constructive derivation if there exist a consistency derivation of level 0 from (C_i $T_i \cup \{L_j\sigma\}$ C_i) to ({} $T_i \cup \{L_j\sigma\}$ C_j). Therefore, steps A3) and A4) are equivalent to:
  - an SLDNF step where L_j or $L_j\sigma$ acts as input clause.
  - one step of application of the negation as failure rule. Note that the selected base event will only be added to T if this inclusion does not alter the failure of the consistency derivations previously considered.
- In step A7) it is checked that there exists a consistency derivation of level 0 from ($\neg L_j$ T_i C_i) to ({} T_i C_j), where L_j is the selected literal. We have proved in (1) that the existence of this derivation corresponds to the failure of the goal $\neg L_j$ and, thus, step A7) corresponds to the negation as failure rule.

Once the base case has been proved, we now assume that the result is true for derivations of level k. We are going to prove that the lemma also holds for derivations of level k+1.

(3) Let CS be a *consistency derivation* of level k+1 from (G T C) to ({} T' C'). Then, the SLDNF tree of $A(K) \cup T' \cup \{G\}$ fails finitely.

---

[1] When G is a set of n goals, n>1, the root of the tree is an implicit goal $\leftarrow G_1$ with n descendant branches, one for each goal in G.

- Steps B1), B2), B7) and B8) are SLDNF resolution steps where A(K) acts as input set.
- Steps B3), B9) and B10) correspond to an SLDNF resolution step where clauses in T' act as input clauses.
- Let G_i be the goal and T_i and C_i the translation and condition sets when steps B4), B5) or B6) are applied. We are going to prove that these steps correspond to an SLDNF resolution step where T' acts as input set.
  - by the own definition of these steps, they correspond to an SLDNF resolution step where clauses of T_i act as input clauses.
  - let $T_p = T' - T_i$. That is, T_p contains the base events added to the translation set after the application of these steps. These base events will have been included in T_p in a step A3) or A4) of a constructive derivation of level k or below. As in steps B4), B5) and B6), G_i is added to the condition set C_i, in A3) and A4) failure for this condition is verified. Then, applying the induction hypothesis, steps B4), B5) and B6) correspond to an SLDNF resolution step where clauses of T_p act as input clauses.

  As $T' = T_i \cup T_p$, there is a corresponding SLDNF resolution step where clauses in T' act as input clauses to steps B4), B5) and B6).
- In steps B11) and B13) a constructive derivation of level k or below is called. Applying the induction hypothesis, there will be a refutation of the subsidiary tree associated to the selected literal. Then, the current branch fails.
- In step B12) a consistency derivation of level k or below is called. Applying the induction hypothesis, the subsidiary tree associated to the selected literal fails finitely. Then, this step corresponds to the negation as failure rule of SLDNF resolution.

(4) Let CT be a *constructive derivation* of level k+1 from (G T C) to ([] T' C'). Then, there exists an SLDNF refutation of $A(K) \cup T' \cup \{G\}$.

- Steps A1) and A5) are SLDNF resolution steps where A(K) acts as input set.
- Steps A2) and A6) correspond to an SLDNF resolution step where T' acts as input set.
- Let L_j be the selected literal and C_i be the condition set when step A3) or A4) is applied. We get the next goal in the constructive derivation if there exists a consistency derivation of level k+1 or below from (C_i $T_i \cup \{L_j\sigma\}$ C_i) to ({} T^n C^n). As we have proved in (3), the existence of this derivation corresponds to the failure of the goal C_i. Then, steps A3) and A4) correspond to a SLDNF step where L_j or $L_j\sigma$ acts as input clause and to one step of application of the negation as failure rule.
- In step A7) it is verified that there exists a consistency derivation of level k+1 or below from ($\neg L_j$ T_i C_i) to ({} T_j C_j), where L_j is the selected literal. As we have proved in (3), the existence of this derivation corresponds to the failure of the goal $\neg L_j$ and, thus, this step corresponds to the negation as failure rule of SLDNF resolution. □

Theorem 6.1: *(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) ∪ T).

Proof: From lemma 1, there exists an SLDNF refutation of A(K) ∪ T ∪ {←u} if there exists a constructive derivation from (←u ∅ ∅) to ([] T C). Then, by the soundness of the SLDNF resolution, it follows that u is a logical consequence of comp(A(K) ∪ T). □

b) Completeness

In this section, we relate the completeness of the Events Method to that of the SLDNF resolution. Let K be a knowledge base, A(K) the augmented knowledge base, u an update request and T a minimal translation such that u is satisfied in the updated knowledge base. Assume that there exists an SLDNF refutation for A(K) ∪ T ∪ {←u}. We prove in this section (theorem 6.2) that there will be a constructive derivation from (←u ∅ ∅) to ([] T C). We first prove lemmas 2, 3 and 4 which are the basis for theorem 6.2.

We use the concept of rank of an SLDNF refutation and rank of a finitely failed SLDNF tree as defined in [Llo87].

Lemma 2: Let K be a knowledge base, A(K) the augmented knowledge base, G and H goals, T, T' and T'' translation sets and C, C' and C'' condition sets. Then, the two following results hold:

a) Let an SLDNF refutation of rank n of A(K) ∪ T ∪ {G} then
- ∀ T' such that T' ⊆ T, and
- ∀ C' such that ∀ C ∈ C' the SLDNF tree for A(K) ∪ T ∪ {C} fails finitely and has rank n-1

there exists a constructive derivation from (G T' C') to ([] T'' C'') such that:
- T'' ⊆ T,
- ∀ C ∈ C'', the SLDNF tree for A(K) ∪ T'' ∪ {C} fails finitely and has rank n-1.

b) Let a finitely failed SLDNF tree of rank n for A(K) ∪ T ∪ {H} then
- ∀ T' such that T' ⊆ T, and
- ∀ C' such that ∀ C ∈ C' the SLDNF tree for A(K) ∪ T ∪ {C} fails finitely and has rank n-1

there exists a consistency derivation from (H T' C') to ({} T'' C'') such that:
- T'' ⊆ T, and
- ∀ C ∈ C''- C', the SLDNF tree for A(K) ∪ T ∪ {C} fails finitely and has rank n.

Proof: the proof is by induction over the rank n of the refutation or the finitely failed tree.

Case n = 0:

a) In this case, the goal G contains only positive literals, and we have to show that there exists a constructive derivation from (G T' ∅) to ([] T'' ∅), for any T' ⊆ T. The idea is to associate to each SLDNF refutation step a corresponding step in a constructive derivation, and prove that in any intermediate step we have (G_i T_i ∅) with $T_i \subseteq T$. Initially, $G_i = G$ and $T_i = T'$.

Let L_j be the selected literal in the refutation. The step applied in the constructive derivation depends on the type of L_j, as follows:

- L_j is not a base event. We apply A1) and obtain (S T_i ∅).
- L_j is a ground base event:
  - If $L_j \in T_i$, we apply A2) and get ($G_i \backslash L_j$ T_i ∅).
  - If $L_j \notin T_i$, we apply A3) and get ($G_i \backslash L_j$ $T_i \cup \{L_j\}$ ∅).
    Note that $L_j \in T$ and therefore $T_i \cup \{L_j\} \subseteq T$.
- L_j is a non ground base event, instantiated by SLDNF with substitution σ. We apply A4) and get ($G_i \backslash L_j\sigma$ $T_i \cup \{L_j\sigma\}$ ∅). Note again that $L_j\sigma \in T$ and then $T_i \cup \{L_j\sigma\} \subseteq T$.

The derivation ends with the empty clause [].

b) As before, the goal H of the tree contains only positive literals, and we have to show that there exists a consistency derivation from (H T' ∅) to ({} T'' C''), for any T' ⊆ T. Again, the idea is to associate to an SLDNF derivation step a corresponding step in a consistency derivation, and prove that in any intermediate step we have (F_i T_i C_i) with $T_i \subseteq T$ and ∀ $C \in C_i$ the SLDNF tree for $A(K) \cup T \cup \{C\}$ fails finitely and has rank 0. Initially, F_i = H, T_i = T' and $T_i = \emptyset$. Note that, in general, F_i is a set of goals that corresponds to a subset of the nodes of the SLDNF tree.

Let $H_i = \{\leftarrow L_1 \wedge \ldots \wedge L_k\}$ be a node in the SLDNF tree and L_j the selected literal. Let $F_i = H_i \cup F'_i$. The step applied in the consistency derivation depends on the type of L_j, as follows:

- L_j is not a base event and S' is the set of all resolvents of clauses in A(K). We apply B1) and obtain ($S' \cup F'_i$ T_i C_i).
- L_j is not a base event and there is no clause in A(K) whose head unifies with L_j. We apply B2) and obtain (F'_i T_i C_i).
- L_j is a ground base event:
  - If $L_j \in T_i$, and $k > 1$, we apply B3) obtaining ($H_i \backslash L_j \cup F'_i$ T_i C_i).
    Note that if k=1, the tree would not be failed.

- If $L_j \notin T_i$, we use B4) and get $(F'_i\ T_i\ C_i \cup \{H_i\})$.
  In this case, the SLDNF tree for $A(K) \cup T \cup \{H_i\}$ has rank 0 and fails finitely since H_i is the current node. If $L_j \notin T_i$ but $L_j \in T$, the SLDNF tree will contain additional nodes, but all of them will end with a failure.

- L_j is a non ground base event and S' is the set of all resolvents of clauses in T_i, we apply B5) and get $(S' \cup F'_i\ T_i\ C_i \cup \{H_i\})$. As before, the SLDNF tree for $A(K) \cup T \cup \{H_i\}$ fails finitely and has rank 0.
  Note that if $T_i \subset T$, the SLDNF tree will contain additional nodes, but all of them will end with a failure.

- L_j is a non ground base event and there is no clause in T_i that can be unified with L_j, we apply B6) and get $(F'_i\ T_i\ C_i \cup \{H_i\})$. Again, the SLDNF tree for $A(K) \cup T \cup \{H_i\}$ fails finitely and has rank 0.
  Note that if $T_i \subset T$, the SLDNF tree will contain additional nodes, but all of them will end with a failure.

The derivation ends with {}.

General case: Assume that the result holds for SLDNF refutations and finitely failed SLDNF trees of rank n-1. We are going to prove that it also holds for refutations and finitely failed trees of rank n.

a) In this case, the goal G may contain positive and negative literals, and we have to show that there exists a constructive derivation from (G T' C') to ([] T'' C''), for any $T' \subseteq T$. Let L_j be the selected literal in the refutation. The step applied in the constructive derivation depends on the type of L_j, as follows:

<u>L_j is positive</u>:

- L_j is not a base event. We apply A1) and obtain $(S\ T_i\ C_i)$.
- L_j is a ground base event:
  If $L_j \in T_i$, we apply A2) and get $(G_i \backslash L_j\ T_i\ C_i)$.
  If $L_j \notin T_i$. We apply A3) which adds L_j to T_i and verifies that there exists a consistency derivation of $(C_i\ T_i \cup \{L_j\}\ C_i)$. The SLDNF tree for $A(K) \cup T \cup \{C_i\}$ fails finitely and has rank n-1. Then, applying the induction hypothesis, there exists a consistency derivation from $(C_i\ T_i \cup \{L_j\}\ C_i)$ to $(\{\}\ T'\ C')$ with $T' \subseteq T$ and for each $C \in C' - C_i$, the SLDNF tree for $A(K) \cup T \cup \{C\}$ fails finitely and has rank n-1.
- L_j is a non ground base event, we apply A4) which proceeds in a similar way that in the previous case once the event has been fully instantiated.

L_j is negative:

- L_j is an old predicate, base or derived. We apply A5) and obtain ($G_i \backslash L_j$ T_i C_i).
- L_j is a ground base event. We apply A6) and get ($G_i \backslash L_j$ T_i $C_i \cup \{\leftarrow L_j\}$). Note that $A(K) \cup T \cup \{\leftarrow L_j\}$ is a tree of rank 0 (and then, of rank n-1).
- L_j is a new or derived event predicate. We apply A7) which verifies that there exists a consistency derivation from ($\leftarrow\neg L_j$ T_i C_i) to ({} T' C'). The SLDNF tree for $A(K) \cup T \cup \{L_j\}$ fails finitely and has rank n-1. Then, applying the induction hypothesis, there exists a consistency derivation from ($\leftarrow\neg L_j$ T_i C_i) to ({} T' C') where T' i C' satisfy the conditions of the theorem.

The derivation ends with [].

b) As before, the goal H of the tree may contain positive and negative literals, and we have to show that there exists a consistency derivation from (H T' C') to ({} T'' C''), for any $T' \subseteq T$. Let $H_i = \{\leftarrow L_1 \wedge \ldots \wedge L_k\}$ be a node in the SLDNF tree and L_j the selected literal. Let $F_i = H_i \cup F'_i$. The step applied in the consistency derivation depends on the type of L_j, as follows:

L_j is positive:

- L_j is not a base event and S' is the set of all resolvents of clauses in A(K). We apply B1) and obtain ($S' \cup F'_i$ T_i C_i).
- L_j is not a base event and there is no clause in A(K) whose head unifies with L_j. We apply B2) and obtain (F'_i T_i C_i).
- L_j is a ground base event:
  - If $L_j \in T_i$ and k >1, we apply B3) obtaining ($F_i \backslash L_j \cup F'_i$ T_i C_i).
    Note that if k=1, the tree would not be failed.
  - If $L_j \notin T_i$, we apply B4) and get (F'_i T_i $C_i \cup \{H_i\}$).
    In this case, the SLDNF tree for $A(K) \cup T \cup \{H_i\}$ has rank n and fails finitely since H_i is the current node. If $L_j \notin T_i$ but $L_j \in T$, the SLDNF tree will contain additional nodes, but all of them will end with a failure.
- L_j is a non ground base event and S' is the set of all resolvents of clauses in T_i, we apply B5) and get ($S' \cup F'_i$ T_i $C_i \cup \{H_i\}$). As before, the SLDNF tree for $A(K) \cup T \cup \{H_i\}$ fails finitely and has rank 0. Note that if $T_i \subseteq T$, the SLDNF tree will contain additional nodes, but all of them will end with a failure.
- L_j is a non ground base event and there is no clause in T_i that can be unified with L_j, we apply B6) and get (F'_i T_i $C_i \cup \{H_i\}$). Again, the SLDNF tree for $A(K) \cup T \cup \{H_i\}$ fails finitely and has rank 0. Note that if $T_i \subseteq T$, the SLDNF tree will contain additional nodes, but all of them will end with a failure.

L_j is negative:

- L_j is an old predicate, base or derived, and the SLDNF tree for $\leftarrow\neg L_j$ fails finitely, we apply B7) and obtain $(H_i \backslash L_j \cup F'_i\ T_i\ C_i)$.
- L_j is an old predicate, base or derived, and there exists an SLDNF refutation for $\leftarrow\neg L_j$, we apply B8) and obtain $(F'_i\ T_i\ C_i)$.
- L_j is a ground base event.
  - If $L_j \in T_i$, we apply B9) and obtain $(F'_i\ T_i\ C_i)$.
  - If $L_j \notin T_i$ and $k > 1$, we apply B10) and get $(F'_i\ T_i\ C_i \cup \{F_i\})$. Note that if $k=1$, the tree would not be failed.
  - If $L_j \notin T_i$. We apply B11) which verifies that there exists a constructive derivation from $(\leftarrow\neg L_j\ T_i\ C_i)$ to ([] T' C'). The SLDNF refutation for $A(K) \cup T \cup \{\leftarrow\neg L_j\}$ has rank n-1. Then, applying the induction hypothesis, there exists a constructive derivation from $(\leftarrow\neg L_j\ T_i\ C_i)$ to ([] T' C'), where T' and C' satisfy the conditions of the theorem.
- L_j is a new or derived event predicate.
  - We apply B12) which verifies that there exists a consistency derivation from $(\leftarrow\neg L_j\ T_i\ C_i)$ to ({} T' C'). The SLDNF tree for $A(K) \cup T \cup \{\leftarrow\neg L_j\}$ fails finitely and has rank n-1. Then, applying the induction hypothesis, there exists a consistency derivation from $(\leftarrow\neg L_j\ T_i\ C_i)$ to ({} T' C'), where T' and C' satisfy the conditions of the theorem.
  - We apply B13) which verifies that there exists a constructive derivation from $(\leftarrow\neg L_j\ T_i\ C_i)$ to ([] T' C'). There is an SLDNF refutation for $A(K) \cup T \cup \{\leftarrow\neg L_j\}$. This refutation has rank n-1. Then, applying the induction hypothesis, there exists a constructive derivation from $(\{\leftarrow\neg L_j\}\ T_i\ C_i)$ to ([] T' C') where T' and C' satisfy the conditions of the theorem. (B13)

The derivation ends with {}. □

Lemma 3: Let K be a knowledge base, A(K) the augmented knowledge base, u an update request and T a minimal translation. All refutations (main and auxiliaries) appearing in the SLDNF search space of $A(K) \cup T \cup \{\leftarrow u\}$ are reached by the constructive derivation from $(\{\leftarrow u\}\ \varnothing\ \varnothing)$ to ([] T C).

Proof: We are going to prove this lemma by contradiction. Assume that there exists an SLDNF refutation for $A(K) \cup T \cup \{G_s\}$ which is not reached by the constructive derivation from $(\{\leftarrow u\}\ \varnothing\ \varnothing)$ to ([] T C).

- It cannot be the main refutation because, by the soundness of our method, to the constructive derivation from $(\{\leftarrow u\}\ \varnothing\ \varnothing)$ to ([] T C) corresponds an SLDNF refutation for $A(K) \cup T \cup \{\leftarrow u\}$.

- Then, it must be an auxiliary refutation. Let A be the finitely failed SLDNF tree for A(K) ∪ T ∪ {G_k} that calls this refutation. By theorem 2, a consistency derivation from (G_k T_k C_k) to ({} T' C') corresponds to this tree. In this derivation, we do not get G_s because it is impeded by a step applied to some literal L_j previously selected. The steps of our method that drop the current branch are:
  - B2), B8) and B9). It cannot be any of them because they are SLDNF resolution steps where either A(K) or T act as input set.
  - B11) and B13). In this case the current branch is dropped if there exists a constructive derivation for the negation of the selected literal. But, by the soundness of our method, it corresponds an SLDNF refutation to this constructive derivation. Then, it can not be any of them.
  - B4) and B6). The current branch is dropped because L_j is a base event that does not belong to T_i, where T_i is the translation set determined when this step is applied. This event is not added to T_i later on because, in this case the consistency derivation would continue. Then, if we remove L_j of T, the tree will still fail and T-{L_j} will also be a solution, which implies that T was not minimal. □

Lemma 4: Let K be a knowledge base, A(K) the augmented knowledge base, u an update request and T a minimal translation such that, using SLDNF resolution, gives a refutation for A(K) ∪ T ∪ {←u}. Then, for each t ∈ T, t is used in the main refutation or in an auxiliary refutation of A(K) ∪ T ∪{←u}.

Proof: Let us assume that t is only used in steps of finitely failed auxiliary trees. Then, if we remove t of T, these trees will still fail and T-{t} will also be a solution, which implies that T was not minimal. □

Theorem 6.2: 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, gives a refutation for A(K) ∪ T ∪ {←u}. Then, there exists a constructive derivation from ({←u} ∅ ∅) to ([] T C).

Proof: A direct consequence of the lemma 2 is that there exists a constructive derivation from ({←u} ∅ ∅) to ([] T' C'), with T' ⊆ T. We know by lemma 3 that this constructive derivation reaches all refutations appearing in the SLDNF search space of A(K) ∪ T ∪ {←u}. Then, T' includes all facts of T used in these refutations. Finally, by lemma 4 we know that all the elements contained in T are used in at least one of these refutations. Therefore, it follows that T' = T. □

Theorem 6.3: *(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) ∪ T and goal {←u}. Then, there exists a constructive derivation from (←u ∅ ∅) to ([] T C) for any translation T that satisfies that u is a logical consequence of comp(A(K) ∪ T).

Proof:

It follows from the completeness of the SLDNF resolution that if u is a logical consequence of comp(A(K) ∪ T) then there exists an SLDNF refutation of A(K) ∪ T ∪ {←u}.

By theorem 6.2, if there exists an SLDNF refutation for A(K) ∪ T ∪ {←u}, then there exists a constructive derivation from (←u ∅ ∅) to ([] T C). □

Theorem 6.3 relates the completeness of the Events Method to that of SLDNF resolution. In Appendix C we will see that, up to now, it has been proved that SLDNF resolution is complete when A(K) is allowed, A(K) ∪ T call-consistent and A(K) ∪ T ∪ {←u} even. The corresponding properties on K are that K is allowed and stratified (or call-consistent).

Appendix C - Syntactic Properties of A(K)

In this section 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 will follow the terminology and definitions of [DC90], 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 (resp. negative) if A is positive (resp. 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* (resp., *negatively*) on F' if there is a path from F' to F containing no negative edge (resp., at least one negative edge).
c) F *depends evenly* (resp., *oddly*) on F' if there is a path from F' to F containing an even (resp., 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 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 an stratified knowledge base K and a recursive rule in K such as P←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 [Cla78] proved completeness for hierarchical and allowed knowledge bases. Cavedon and Lloyd [CL89] showed completeness for knowledge bases and goals which are allowed, strict and stratified and Kunen [Kun89] for knowledge bases and goals which are allowed, strict and cali-consistent. More recently, Decker and Cavedon [DC90] 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.

References

[Abi88] Abiteboul, S. "Updates, a new frontier", Proc. ICDT 88, Springer, 1988, pp. 1-18.

[ABW88] Apt, K.R.; Blair, H.A.; Walker, A. "Towards a Theory of Declarative Knowledge", in *Foundations of Deductive Databases and Logic Programming* (J. Minker Ed.), Morgan-Kaufman, 1988, pp. 89-148.

[AT92] Atzeni, P.; Torlone, R. "Updating Intensional Predicates in Datalog", Data & Knowledge Engineering (8), 1992, pp. 1-17.

[BAK91] Bol, R.N.; Apt, K.R.; Klop, J.W. "An Analysis of Loop Checking Mechanisms for Logic Programs", Theoretical Computer Science 86, 1991, pp. 35-79.

[BMM90] Bry, F.; Manthey, R.; Martens, B. "Integrity Verification in Knowledge Bases", ECRC report D.2.1.a, Munich, 1990.

[Bol93] Bol, R.N. "Loop Checking and Negation", Journal of Logic Programming, Vol. 15, 1993, pp.147-175.

[BR86] Bancilhon, F.; Ramakrishnan, R, "An Amateur's Introduction to Recursive Query Processing", Proc. ACM SIGMOD Int. Conf. on Management of Data, Washington D.C., 1986, pp. 16-52.

[Bry90] Bry, F. "Intensional Updates: Abduction via Deduction", Proc. 7th ICLP, Jerusalem 1990.

[BS81] Bancilhon, F; Spyratos, N. "Update Semantics of Relational Views", ACM Transactions on Database Systems, vol. 6, num. 4, 1981, pp. 557-575.

[CFPT92] Ceri, S.; Fraternali, P.; Paraboschi, S.; Tanca, L. "Integrity Maintenance Systems: an architecture", Third Int. Workshop on the Deductive Approach to Information Systems and Databases, Roses, Catalonia, 1992, pp. 327-344.

[CL89] Cavedon, L.; Lloyd, J. "A Completeness Theorem for SLDNF Resolution", Journal of Logic Programming, vol. 7, 1989, pp. 177-191.

[Cla78] Clark, K.L. "Negation as Failure", in Gallaire, H.; Minker, J. "Logic and Databases", Plenum Press, New York, pp. 293-322.

[CP84] Cosmadakis, S.; Papadimitriou, C. "Updates of Relational Views", Journal of the Association for Computing Machinery, vol. 31, núm. 4, 1984, pp. 742-760.

[CW90] Ceri, S.; Widom, J. "Deriving Production Rules for Constraint Maintenance", Proc. of the 16th VLDB Conference, Brisbane, Australia, 1990, pp. 566-577.

[Dat86] Date, C.J. "Updating views", in *Relational Databases: Selected Writings*, Addison - Wesley, 1986, pp.367-395.

[DB82] Dayal, U.; Bernstein, P.A. "On the Correct Translation of Update Operations on Relational Views", ACM Transactions on Database Systems, Vol. 8, Nº 3, 1982, pp. 381-416.

[DC90] Decker, H.; Cavedon, L. "Generalizing Allowedness while Retaining Completeness of SLDNF-resolution", Proc. 3rd Workshop on Computer Science Logic, Kaiserslautern, Springer-Verlag, 1990, pp. 98-115.

[Dec89] Decker, H. "The Range Form of databases or: How to avoid Floundering", Proc. 5th ÖGAI, Springer-Verlag, 1989.

[Dec90] Decker, H. "Drawing Updates from derivations", Proc. of the 3rd Int. Conf. on Database Theory (ICDT), Paris, 1990, pp. 437-451.

[FC85] Furtado, A.L.; Casanova, M.A. "Updating Relational Views", in W.Kim et el. eds., *Query Processing in Database Systems*, Springer-Verlag, 1985, pp. 127-142.

[FKUV86] Fagin, R.; Kuper, G.M.; Ullman, J.D.; Vardi, M.Y. "Updating logical databases", Advances in Computing Research, vol. 3, JAI-Press, 1986, pp. 1-18.

[FUV83] Fagin, R.; Ullman, J.D.; Vardi, M.Y. "On the Semantics of Updates in Databases", Proc. ACM PODS, 1983.

[Gär88] Gärdenfors, P. "Knowledge in Flux: Modeling the Dynamics of Epistemic States", MIT Press, 1988.

[GL90] Guessoum, A.; Lloyd, J.W. "Updating Knowledge Bases", New Generation Computing, Vol. 8, No. 1, 1990, pp. 71-89.

[GL91] Guessoum, A.; Lloyd, J.W. "Updating Knowledge Bases II", New Generation Computing, Vol. 10, 1991, pp. 73-100.

[GMN84] Gallaire, H.; Minker,J.; Nicolas, J.M. "Logic and Databases: A Deductive Approach". ACM Computing Surveys, Vol. 16, Nº 2, 1984, pp. 153-185.

[Kel85] Keller, A.M. "Algorithms for Translating View Updates to Database Updates for Views Involving Selection, Projections and Joins". Proc. 4th. ACM SIGACT-SIGMOD Symp. on Principle of Database Systems, 1985, pp. 154-163.

[Kel86] Keller, A.M. "Choosing Translator at View Definition Time". Proc. 12th VLDB Conference, Kyoto, 1986, pp. 467-474.

[KM90] Kakas, A.; Mancarella, P. "Database Updates through Abduction", Proc. of the 16th VLDB Conference, Brisbane, Australia, 1990, pp. 650-661.

[Kow92] Kowalski, R. "Database Updates in the Event Calculus". Journal of Logic Programming, vol. 12, 1992. pp. 121-146.

[Kun89] Kuner, K. "Signed Data Dependencies in Logic Programs", Journal of Logic Programming, vol. 7, 1989, pp. 231-245.

[Lan90] Langerak, R. "View Updates in Relational Databases with an Independent Schema Interface", ACM Transactions on Database Systems, Vol. 15, No 1, 1990, pp.40-66.

[Llo87] Lloyd, J.W. "Foundations on Logic Programming", 2nd edition, Springer, 1987.

[LlS91] Lloyd, J.W.; Shepherdson, J.C. "Partial Evaluation in Logic Programming", Journal of Logic Programming, No. 11, 1991, pp. 217-247.

[LlT84] Lloyd, J.W.; Topor, R.W. "Making Prolog More Expressive". Journal of Logic Programming, 1984, No. 3, pp. 225-240.

[LS91] Larson, J; Sheth, A. "Updating Relational Views Using Knowledge at View Definition and View Update Time", Information Systems, Vol. 16, No. 2, 1991, pp. 145-168.

[Mas84] Masunaga, Y. "A Relational Database View Update Translation Mechanism". Proc. of the 10th VLDB Conference, Singapore, 1984, pp. 309-320.

[ML91] Moerkotte, G; Lockemann, P.C. "Reactive Consistency Control in Deductive Databases", ACM Transactions on Database Systems, Vol. 16, No. 4, December 1991, pp. 670-702.

[MW88] Manchanda, S.; Warren, D.S. "A logic-based language for database updates" in J. Minker ed., *Foundations of Deductive Databases and Logic Programming*, Morgan-Kaufmann Pub., 1988, pp. 363-394.

[Nic82] Nicolas,J.M. "Logic for improving integrity checking in relational data bases". Technical report, ONERA-CERT, Feb. 1979. Also in Acta Informatica 1982, 18, 3, pp. 227-253.

[Oli89] Olivé, A. "On the Design and Implementation of Information Systems from Deductive Conceptual Models", Proc. of 15th VLDB Conference, Amsterdam, 1989, pp. 3-11.

[Oli91] Olivé, A. "Integrity Checking in Deductive Databases", Proc. of the 17th VLDB Conference, Barcelona, Catalonia, 1991, pp. 513-523.

[Rei84] Reiter, R. "Towards a logical reconstruction of relational database theory" in M.L. Brodie, J. Mylopoulos and J.W.Schmidt (eds.), *On Conceptual Modeling*, Springer-Verlag, 1984, pp. 191-233.

[SK88] Sadri, F.; Kowalski R. "A Theorem-Prover Approach to Database Integrity", in J. Minker ed., *Foundations of Deductive Databases and Logic Programming*, Morgan-Kaufman, 1988, pp. 313-362.

[TA91] Torlone, R.; Atzeni, P. "Updating Deductive Databases with Functional Dependencies", 2nd Int. Conf. on Deductive and Object Oriented Databases (DOOD'91), Munich, 1991, pp. 278-291.

[Ten92] 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).

[TO92] Teniente, E.; Olivé. A. "The Events Method for View Updating in Deductive Databases", Int. Conf. on Extending Database Technology (EDBT'92), Vienna, 1992, pp. 245-260.

[Tom88] Tomasic, A. "View Update Annotation in Definite Deductive Databases", Proc. Int. Conf. on Database Theory (ICDT '88), Springer-Verlag, 1988, pp. 338-352.

[Ull88] Ullman, J.D. "Principles of Database and Knowledge-Base Systems", Computer Science Press, New York, 1988.

[UO92] Urpí, T.; Olivé, A. "A Method for Change Computation in Deductive Databases", Proc. of the 18th VLDB Conference, Vancouver, Canada, 1992, pp. 225-237.

[Urp93] Urpí, A. "El Mètode dels Esdeveniments per al càlcul de canvis en bases de dades deductives", PhD Thesis, Universitat Politècnica de Catalunya, Barcelona, 1993 (in catalan).

[Win90] Winslett, M. "Updating Logical Databases", Cambridge Tracts in Theoretical Computer Science 9, 1990.

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports – 1994

LSI–94–1–R "Logspace and logtime leaf languages", Birgit Jenner, Pierre McKenzie, and Denis Thérien.

LSI–94–2–R "Degrees and reducibilities of easy tally sets", Montserrat Hermo.

LSI–94–3–R "Isothetic polyhedra and monotone boolean formulae", Robert Juan-Arinyo.

LSI–94–4–R "Una modelización de la incompletitud en los programas" (written in Spanish), Javier Pérez Campo.

LSI–94–5–R "A multiple shooting vectorial algorithm for progressive radiosity", Blanca Garcia and Xavier Pueyo.

LSI–94–6–R "Construction of the Face Octree model", Núria Pla-Garcia.

LSI–94–7–R "On the expected depth of boolean circuits", Josep Díaz, María J. Serna, Paul Spirakis, and Jacobo Torán.

LSI–94–8–R "A transformation scheme for double recursion", José L. Balcázar.

LSI–94–9–R "On architectures for federated DB systems", Fèlix Saltor, Benet Campderrich, and Manuel García-Solaco.

LSI–94–10–R "Relative knowledge and belief: SKL preferred model frames", Matías Alvarado.

LSI–94–11–R "A top-down design of a parallel dictionary using skip lists", Joaquim Gabarró, Conrado Martínez, and Xavier Messeguer.

LSI–94–12–R "Analysis of an optimized search algorithm for skip lists", Peter Kirschenhofer, Conrado Martínez, and Helmut Prodinger.

LSI–94–13–R "Bases de dades bitemporals" (written in Catalan), Carme Martín and Jaume Sistac.

LSI–94–14–R "A volume visualization algorithm using a coherent extended weight matrix", Daniela Tost, Anna Puig, and Isabel Navazo.

LSI–94–15–R "Deriving transaction specifications from deductive conceptual models of information systems", María Ribera Sancho and Antoni Olivé.

LSI–94–16–R "Some remarks on the approximability of graph layout problems", Josep Díaz, María J. Serna, and Paul Spirakis.

LSI–94–17–R "SAREL: An assistance system for writing software specifications in natural language", Núria Castell and Àngels Hernández.

LSI–94–18–R "Medición del factor modificabilidad en el proyecto LESD" (written in Spanish), Núria Castell and Olga Slávkova

LSI–94–19–R "Algorismes paral·lels SIMD d'extracció de models de fronteres a partir d'arbres octals no exactes" (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

LSI–94–20–R "Una paral·lelització SIMD de la conversió d'objectes codificats segons el model de fronteres al mdel d'octrees clàssics" (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

LSI–94–21–R "Clausal proof nets and discontinuity", Glyn Morrill.

LSI–94–22–R "A formal method for the synthesis of update transactions in deductive databases without existential rules", Joan A. Pastor.

LSI–94–23–R "On the sparse set conjecture for sets with low density", Harry Buhrman and Montserrat Hermo.

LSI–94–24–R "On infinite sequences (almost) as easy as π", José L. Balcázar, Ricard Gavaldà, and Montserrat Hermo.

LSI–94–25–R "Updating knowledge bases while maintaining their consistency", Ernest Teniente and Antoni Olivé.

LSI–94–26–R "Structural facilitation and structural inhibition", Glyn Morrill.

Copies of reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es