Incorporating Modification Requests in Updating Consistent Knowledge Bases

(Preliminary Version)

Enric Mayol
Ernest Teniente

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

Abstract

An important problem that arises when updating a knowledge base is related to integrity constraints maintenance. That is, given a consistent knowledge base and an update request, to obtain all possible ways of updating the base facts such that the request is satisfied and consistency is maintained.

In this paper we present a new method for providing automatic support in these situations. Our method takes into account information about the keys of the predicates and it is able to handle updates which consist of a set of insertions, deletions and/or modifications of base and derived predicates.

KEYWORDS: knowledge base updating, integrity constraints maintenance, modification requests

July 1993

1 Introduction

A knowledge base is called *consistent* if it satisfies the set of its integrity constraints. When performing an update, knowledge base consistency may be violated. That is, the update, together with the current contents of the knowledge base, may falsify some integrity constraint. One possible way of resolving this conflict consists of trying to repair constraints violations by considering additional updates which restore the consistency of the knowledge base . Then, it is guaranteed that the knowledge base state resulting from applying the update does not violate any integrity constraint and that it satisfies the update requested by the user. This problem is usually known as *integrity constraints maintenance* or *enforcement*.

Two basic approaches have been proposed to solve this problem. The first one consist of generating repairs at *execution-time* by taking into account the information provided by the update request and the contents of the knowledge base [ML91, Wüt93, TO93]. The second approach is concerned with preparing these extra updates at *compile-time*. In this case, production rules, ECA rules or transaction specifications are generated from a knowledge base schema and a proposed update description. They allow the specification of data manipulation operations that are automatically executed whenever certain events occur and/or certain conditions are met [CW90, CFPT92, Pas92].

We propose here a new method for integrity constraints maintenance following the execution-time approach. 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 events rules, which explicitly define insertions, deletions and modifications induced by a knowledge base update. These rules are then used for updating a knowledge base while maintaining its consistency.

We take into account information about the keys of base and view predicates. This allows us to deal with knowledge base updates which consist of insertions, deletions and modifications of base and view predicates, where a modification is understood as a change in some non-key argument of a predicate. View updates are automatically translated by our method into correct updates of the underlying base facts. Moreover, maintenance of key integrity constraints is guaranteed by the own definition of the method, which substantially improves its efficiency.

The paper is organized as follows. Next section reviews basic concepts of knowledge bases. Section 3, which is based on [UO92, Urp93], reviews the concepts of event, transition rules and events rules. Section 4 discusses the application of these rules to updating consistent knowledge bases. In sections 5 we stress the differences between our method and the Events Method [TO93] and in setion 6 we compare our method with related work. Finally, we present in section 7 our conclusions and point out future research.

2 Knowledge Bases

A knowledge base KB consists of three finite sets: a set F of facts, a set R of deductive rules, and a set I of integrity constraints. A fact is a ground atom. The set of facts is called the *extensional* part of the knowledge base and the set of deductive rules and integrity constraints 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].

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

2.1 Deductive Rules

A deductive rule is a formula of the form:

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

where P is an atom denoting the conclusion, and L_1, ..., L_n are literals representing conditions. Any variable in P, L_1,..., L_n is assumed to be universally quantified over the whole formula. A derived predicate P may be defined by means of one or more deductive rules.

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

We deal with *stratified* knowledge bases [ABW88] and, as usual, we require the knowledge base to be *allowed* [Llo87]. That is, any variable that occurs in a deductive rule has an occurrence in a positive condition of an ordinary predicate. This ensures that all negative conditions can be fully instantiated before they are evaluated by the "negation as failure" rule.

2.2 Integrity Constraints

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

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

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].

The above are called *static* integrity constraints, since they must be satisfied in any state of the knowledge base. Constraints involving two consecutive knowledge base states, which are called *transition* integrity constraints, will be considered in section 4.

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 rewrite the former denial as $Ic_n \leftarrow L_1 \wedge ... \wedge L_n$.

To enforce the concept of key we assume that associated to each P(**k**,**x**) there is a primary key integrity constraint that we define as:

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

Like with primary keys, to enforce the concept of alternate key, we also assume that there may exist some alternate key integrity constraint associated to each base predicate P(**ka**,**x**), where **ka** is a vector of variables that form an alternate key of P. These constraints are defined as:

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

The main difference between primary and alternate key constraints is that the former are used to define and simplify the events rules defined in next section, while the latter are only used to simplify them.

3 Transition and Events Rules

In this section, we shortly review the concept of *event* and the procedure for automatically deriving the *transition* and *events rules* for a given knowledge base, as presented in [UO92, Urp93]. These rules depend only on the deductive rules, and are independent from the base facts stored in the knowledge base. In a later section, we will discuss the use of these rules for updating consistent knowledge bases.

3.1 Events

Let K^o be a knowledge base, P a predicate, U an update, K^n the updated knowledge base and P^n the predicate P evaluated in K^n. We say that U induces a transition from K^o (the current state) to K^n (the new, updated state). We assume for the moment that U consists of an unspecified set of base and/or derived facts to be inserted, deleted and/or modified. In general,

the application of U will change the extension of knowledge base predicates. We formalize these changes with the concept of *event*.

For each predicate P of the knowledge base, we define two meta predicates ιP and δP (called *insertion* and *deletion event predicates*) as follows:

(1) $\forall \mathbf{k},\mathbf{x}\ (\iota P(\mathbf{k},\mathbf{x}) \leftrightarrow P^n(\mathbf{k},\mathbf{x}) \wedge \neg\ \exists \mathbf{y} P^o(\mathbf{k},\mathbf{y}))$
(2) $\forall \mathbf{k},\mathbf{x}\ (\delta P(\mathbf{k},\mathbf{x}) \leftrightarrow P^o(\mathbf{k},\mathbf{x}) \wedge \neg\ \exists \mathbf{y} P^n(\mathbf{k},\mathbf{y}))$
where **k** and **x** are vectors of variables.

Furthermore, we define a *modification event predicate* for each derived predicate P with non-key arguments:

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

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

From the above, we then have the equivalencies:

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

(5) $\forall \mathbf{k},\mathbf{x}\ (\neg P^n(\mathbf{k},\mathbf{x}) \leftrightarrow (\neg P^o(\mathbf{k},\mathbf{x}) \wedge \neg\ \iota P(\mathbf{k},\mathbf{x}) \wedge \neg\ \mu P(\mathbf{k},\mathbf{x}',\mathbf{x}))$
$\vee\ \delta P(\mathbf{k},\mathbf{x})$
$\vee\ \mu P(\mathbf{k},\mathbf{x},\mathbf{x}'))$
which relate the new state with the old state and the events induced in the transition.

If P is a base predicate, then ιP, δP and μP facts represent insertions, deletions and modifications of base facts, respectively. If P is a derived predicate, then ιP, δP and μP facts represent induced insertions, deletions and modifications, respectively.

Therefore, we assume from now on that U consists of an unspecified set of insertion and/or deletion and/or modification events. We say that an event (ιP, δP, μP) is base (resp. derived) if P is base (resp. derived). It is only required that the events contained in U be mutually exclusive. That is:

(6) $\forall \mathbf{k},\mathbf{x}\ (\iota P(\mathbf{k},\mathbf{x}) \rightarrow \neg\ \exists \mathbf{y}\ (\iota P(\mathbf{k},\mathbf{y}) \wedge \mathbf{x} \neq \mathbf{y}))$

(7) $\forall \mathbf{k},\mathbf{x}\ (\delta P(\mathbf{k},\mathbf{x}) \rightarrow \neg\ \exists \mathbf{y}\ (\mu P(\mathbf{k},\mathbf{x},\mathbf{y}) \wedge \mathbf{x} \neq \mathbf{y}))$

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

If P is an inconsistency predicate, then ιP facts represent violations of its integrity constraint. In this case, δP and μP facts are not defined since we assume that the knowledge base is consistent before the update.

3.2 Transition Rules

Let P be a derived or inconsistency predicate of the knowledge base. The definition of P consists of the rules in the knowledge base having P in the conclusion. Assume that there are m (m≥1) such rules. For our purposes, we require to rename the predicate symbol in the conclusions of the m rules by P_1, ..., P_m and add the set of clauses:

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

Consider now one of the rules $P_i(\mathbf{k,x}) \leftarrow L_1 \wedge ... \wedge L_n$. When this rule is to be evaluated in the new state its form is $P^n_i(\mathbf{k,x}) \leftarrow L^n_1 \wedge ... \wedge L^n_n$ where L^n_r (r = 1...n) is obtained by replacing the predicate Q^o of L_r by Q^n**.** Now, if we replace each literal in the body by its equivalent definition given in (4) or (5), we get a new rule, called a *transition rule*, which defines predicate P^n_i (new state) in terms of old state predicates and events.

More precisely, if L^n_r is an ordinary positive literal $Q^n_r(\mathbf{k_r,x_r})$ we apply (4) and replace it by:

$$(Q^o_r(\mathbf{k_r,x_r}) \wedge \neg\, \delta Q_r(\mathbf{k_r,x_r}) \wedge \neg\, \mu Q_r(\mathbf{k_r,x_r,x'_r})) \vee \iota Q_r(\mathbf{k_r,x_r}) \vee \mu Q_r(\mathbf{k_r,x'_r,x_r})$$

and if L^n_r is an ordinary negative literal $\neg Q^n_r(\mathbf{k_r,x_r})$ we apply (5) and replace it by:

$$(\neg Q^o_r(\mathbf{k_r,x_r}) \wedge \neg\, \iota Q_r(\mathbf{k_r,x_r}) \wedge \neg\, \mu Q_r(\mathbf{k_r,x'_r,x_r})) \vee \delta Q_r(\mathbf{k_r,x_r}) \vee \mu Q_r(\mathbf{k_r,x_r,x'_r})$$

If L^n_r is an evaluable predicate, we just replace L^n_r by its old state version L^o_r. After distributing ∧ over ∨, we get the set of transition rules for predicate P^n_i.

3.3 Insertion Events Rules

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

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

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

$$(9)\quad \iota P(\mathbf{k,x}) \leftarrow P^n_i(\mathbf{k,x}) \wedge \neg\, \exists \mathbf{y} P^o{}_1(\mathbf{k,y}) \wedge ... \wedge \neg\, \exists \mathbf{y} P^o{}_i(\mathbf{k,y}) \wedge ... \wedge \neg\, \exists \mathbf{y} P^o{}_m(\mathbf{k,y}) \qquad i=1...m$$

Rules (9) are called *insertion events rules* of predicate P. They allow us to deduce which ιP facts (induced insertions) happen in a transition.

3.4 Deletion Events Rules

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

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

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

$$(10)\quad \delta P(\mathbf{k,x}) \leftarrow P^o{}_i(\mathbf{k,x}) \wedge \neg\, \exists \mathbf{y} P^n{}_1(\mathbf{k,y}) \wedge \ldots \wedge \neg\, \exists \mathbf{y} P^n{}_i(\mathbf{k,y}) \wedge \ldots \wedge \neg\, \exists \mathbf{y} P^n{}_m(\mathbf{k,y}) \qquad i=1\ldots m$$

Rules (10) are called *deletion events rules* of predicate P. They allow us to deduce which δP facts (induced deletions) happen in a transition.

3.5 Modification Events Rules

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

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

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

$$(11)\quad \mu P(\mathbf{k,x,x'}) \leftarrow P^n{}_i(\mathbf{k,x'}) \wedge P^o{}_h(\mathbf{k,x}) \wedge \mathbf{x} \neq \mathbf{x'} \qquad i,h=1\ldots m$$

Rules (11) are called *modification events rules* of predicate P. They allow us to deduce which μP facts (induced modifications) happen in a transition.

Insertion, deletion and modification rules can be automatically transformed into a set of equivalent and simplified rules by using the knowledge provided by primary key and alternate key constraints. These simplified rules can be evaluated more efficiently. These simplifications are presented in [UO92, Urp93]. In this paper we will use the simplified version of events rules.

3.5 The Augmented Knowledge Base

Let K be a knowledge base. We call *augmented knowledge base*, and we denote it by A(K), the knowledge base consisting of K, its transition rules and its events rules. It has been proved in [Urp93] that if the properties of K are such that SLDNF resolution is complete in K and no recursive predicate in K has non-key arguments, then SLDNF resolution will also be complete for A(K).

4 Updating Consistent Knowledge Bases

In this section we will apply the transition and event rules defined in the last section to update a knowledge base while maintaining its consistency. The *integrity constraints maintenance problem* can be stated as follows: given a consistent knowledge base and an update, obtain all possible changes of the base facts such that the update request is satisfied and consistency of the KB is mantained.

4.1 Our Method

In this preliminary version of the method, we have only considered the *universal key case*. That is, if P is a derived or an inconsistency predicate, keys of the literals appearing in the body of the deductive rules for P must be a subset of (or equal to) the key of P.

Our method deals with *update requests* that consist of a set of insertions, deletions and/or modifications of base and derived predicates. As we mentioned in previous section, it is only required that the events composing the update request be mutually exclusive.

Given an update request, our method obtains automatically all possible ways of changing the knowledge base such that the update becomes true and integrity constraints remain satisfied. We consider *solutions* only that involve updates of the extensional part of the knowledge base, that is, insertions, deletions and modifications of base facts. In general, several solutions may exist. Our approach consists in generating all *minimal* solutions. We require the solutions to be minimal in the sense that no proper subset of them is itself a solution.

In our method, an insertion (resp. deletion) request corresponds to a fact $\iota P(\mathbf{k},\mathbf{x})$ (resp. $\delta P(\mathbf{k},\mathbf{x})$), where $P^n(\mathbf{k},\mathbf{x})$ is the fact that must hold (resp. must not hold) in the new state of the knowledge base. A modification request corresponds to a fact $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'})$, where $P^n(\mathbf{k},\mathbf{x'})$ must hold and $P^n(\mathbf{k},\mathbf{x})$ must not hold in the new state of the knowledge base. Due to the definition of the events, we require that if $\iota P(\mathbf{k},\mathbf{x})$ is an insertion ($\delta P(\mathbf{k},\mathbf{x})$ a deletion or $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'})$ a modification) request, then $P(\mathbf{k},\mathbf{x})$ must not hold ($P(\mathbf{k},\mathbf{x})$ must hold or $P(\mathbf{k},\mathbf{x})$ must hold and $\mathbf{x}\neq\mathbf{x'}$) in the current knowledge base.

A translation of an update request, denoted by T, defines a set of insertions, deletions and/or modifications of base facts such that if u is an insertion request $\iota P(\mathbf{k},\mathbf{x})$ (resp. deletion request $\delta P(\mathbf{k},\mathbf{x})$), $P^n(\mathbf{k},\mathbf{x})$ is (resp. is not) a logical consequence of the completion of the knowledge base, once updated according to T. If u is a modification request $\mu P(\mathbf{k},\mathbf{x},\mathbf{x'})$, $P^n(\mathbf{k},\mathbf{x'})$ is a logical consequence of the completion of the updated knowledge base, and $P^n(\mathbf{k},\mathbf{x})$ is not .

Let K be a knowledge base, A(K) the augmented knowledge base, u an update consisting in a set of base and/or derived events, T a translation consisting of a set of base events. In our

method, T will be a solution if, using SLDNF resolution, the goal $\{\leftarrow u \wedge \neg \iota Ic\}$ succeeds from input set $A(K) \cup T$. T is obtained by having some failed SLDNF derivation of $A(K) \cup \{\leftarrow u \wedge \neg \iota Ic\}$ succeed. This is effected by including in the input set of clauses a ground instance of each literal corresponding to a positive base event selected during the derivation.

Enforcement of primary and alternate key integrity constraints is included in the management of the updates during the translation process. Then, we do not need to define key integrity constraints nor to derive their transition and event rules. The only thing that needs to be done is to specify the key arguments of each base predicate. For this reason, predicate ιIc is defined as $\iota Ic \leftarrow \iota Ic1, \ldots, \iota Ic \leftarrow \iota Icn$, where n is the number of non-key integrity constraints and each ιIcj has its respective arguments.

The possible ways in which a failed derivation may succeed correspond to the different solutions T_i that satisfy the update request and maintain knowledge base consistency. If we get the empty clause, then the translation set T will contain the base event facts selected during the derivation. If no translation T is obtained, the update cannot be satisfied by changing only the extensional part of the knowledge base. In this case, changes to the intensional knowledge base (mainly updates of deductive rules and/or integrity constraints) should be considered. These changes are not obtained by our method, and we delegate this decision to the knowledge base administrator.

We use an auxiliary set C, which we call *condition set* to ensure that the solution T is consistent with the update request and the integrity constraints of the knowledge base. In general, C contains conditions of the form $\leftarrow L_1 \wedge \ldots \wedge L_n$ that must be satisfied by the events contained in the translation set T. C is also obtained during the derivation process and it contains some of the goals reached in the derivations. Before adding a base event to T, we have to check the consistency of each condition of C, that itself can cause the inclusion of new elements in T.

We would like to note that *transition integrity constraints* can also be maintained in our method. These constraints involve two consecutive knowledge base states. Transition integrity constraints have the same form as static integrity constraints (see in section 2), but literals can be knowledge base predicates as well as transition and event predicates. A typical example could be a constraint stating that salaries cannot decrease. See [Oli91] for details of transition and events rules in this case.

4.2 Example

The following example illustrates our approach. We give the formal definition of our method in section 4.3. Let K be a knowledge base with the following base and derived predicates:

Cont (k,x)	teacher k has a contract with the university x.
Teach (k,x)	teacher k is teaching at the university x.
Enq (k)	teacher k is enquired.
Visits (k,x)	any teacher k that is teaching at a university x, but does not have a contract with it, is a visiting teacher.
Ic1(k,x)	an enquired teacher can not teach at the UGT (University of Good Teachers).

The current contents of the knowledge base is:

F.1 Teach (Tom, UAB).
F.2 Enq (Tom).
F.3 Cont (Tom, UGT).
F.4 Cont (Julie, UAB).
DR. $Visits(k,x) \leftarrow Teach(k,x) \wedge \neg Cont(k,x)$
IC. $Ic1(k,x) \leftarrow Enq(k) \wedge Teach(k,x) \wedge x=UGT$

Notice that, as we mentioned in section 4.1, we do not need to define key integrity constraints because their maintenance is ensured by the own definition of our method.

Relevant event rules are the following:

M.2 $\mu Visits(k,x,x') \leftarrow \mu Teach(k,x,x') \wedge \delta Cont(k,x')$
M.3 $\mu Visits(k,x,x') \leftarrow \mu Teach(k,x,x') \wedge \mu Cont(k,x',z)$

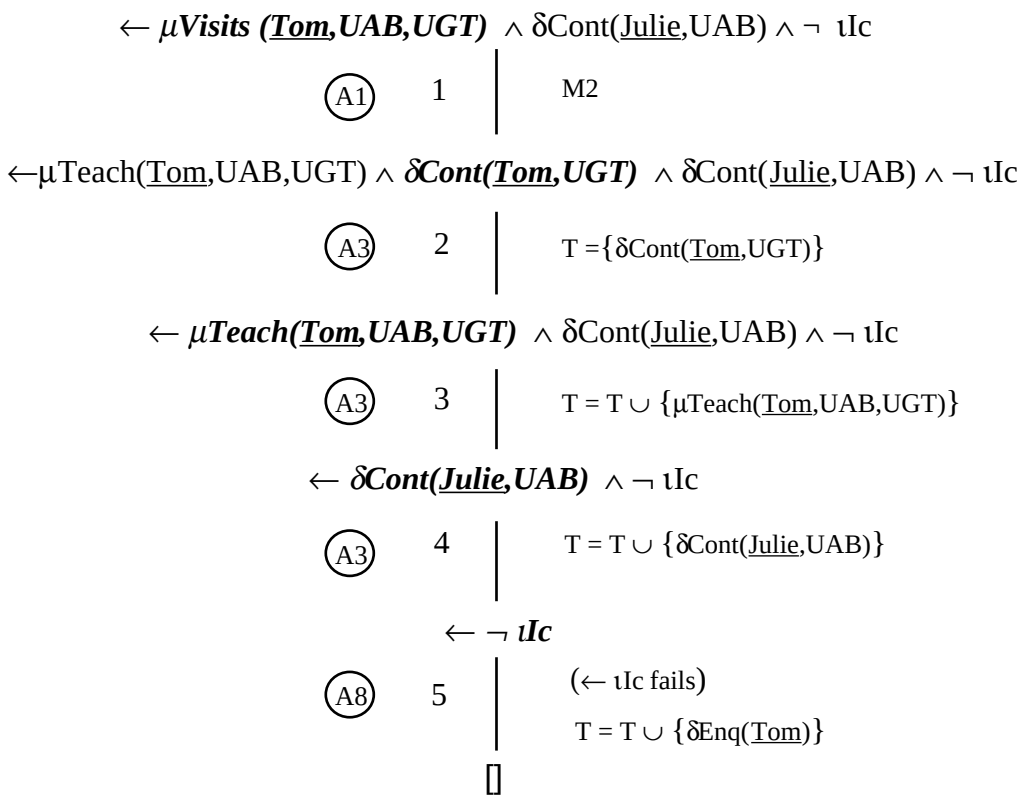
IC.0 $\iota Ic \leftarrow \iota Ic1(k,x)$
IC.1 $\iota Ic1(k,x) \leftarrow Enq(k) \wedge \neg \delta Enq(k) \wedge \iota Teach(k,x) \wedge x=UGT$
IC.2 $\iota Ic1(k,x) \leftarrow Enq(k) \wedge \neg \delta Enq(k) \wedge \mu Teach(k,x',x) \wedge x=UGT$

Note that we have eliminated their sub and super indexes to be more readable.

From now on, we will indicate the rule of the method applied at each step of a derivation by means of a circled label. Moreover, the selected literal of the goal will be in cursive and bold style, and key arguments of each literal will be underlined.

Let the update request be: to change Tom status of visiting teacher from university UAB to the university UGT ($\mu Visits(Tom,UAB,UGT)$) and to cancel the contract of Julie with university UAB ($\delta Cont(Julie,UAB)$).

Solutions that satisfy this request and maintain knowledge base consistency are obtained by having some failed SLDNF derivation of $A(K) \cup \{ \leftarrow \mu Visits(Tom,UAB,UGT) \wedge \delta Cont(Julie,UAB) \wedge \neg \iota Ic \}$ succeed. Part of this derivation is shown in figure 4.1.



T= { δCont(Tom,UGT), μTeach(Tom,UAB,UGT), δCont(Julie,UAB), δEnq(Tom)}

Fig. 4.1

Step 1 corresponds to an SLDNF resolution step resolving with the modification event rule M2. At step 2, to get a successful derivation, we have to include the selected positive base event (δCont(Tom,UGT)) in the translation set T and use it as an input clause. Before including it in T, we have to make sure that Cont(Tom,UGT) holds in the current knowledge base. Step 3 is similar to step 2, but we also must check that old value of the non-key argument (UAB) is different to the new value (UGT). At step 4 we proceed in the same way of the second one.

At step 5 the selected literal is ¬ ιIc. To get a success for this branch ιIc must not hold. Then, we have to ensure that the subsidiary tree rooted by ←ιIc fails finitely. This is equivalent to guarantee that the translation T={δCont(Tom,UGT), μTeach(Tom,UAB,UGT), δCont(Julie,UAB)} mantains KB consistency. Notice that this is not the case because the current translation violates integrity constraint Ic1. Thus, we must add the event δEnq(Tom) to the set T to restore KB consistency. In figure 4.2 we show how this is performed in our method.

We show only one branch of the derivation because the other branches fail finitely without modifying the translation set.

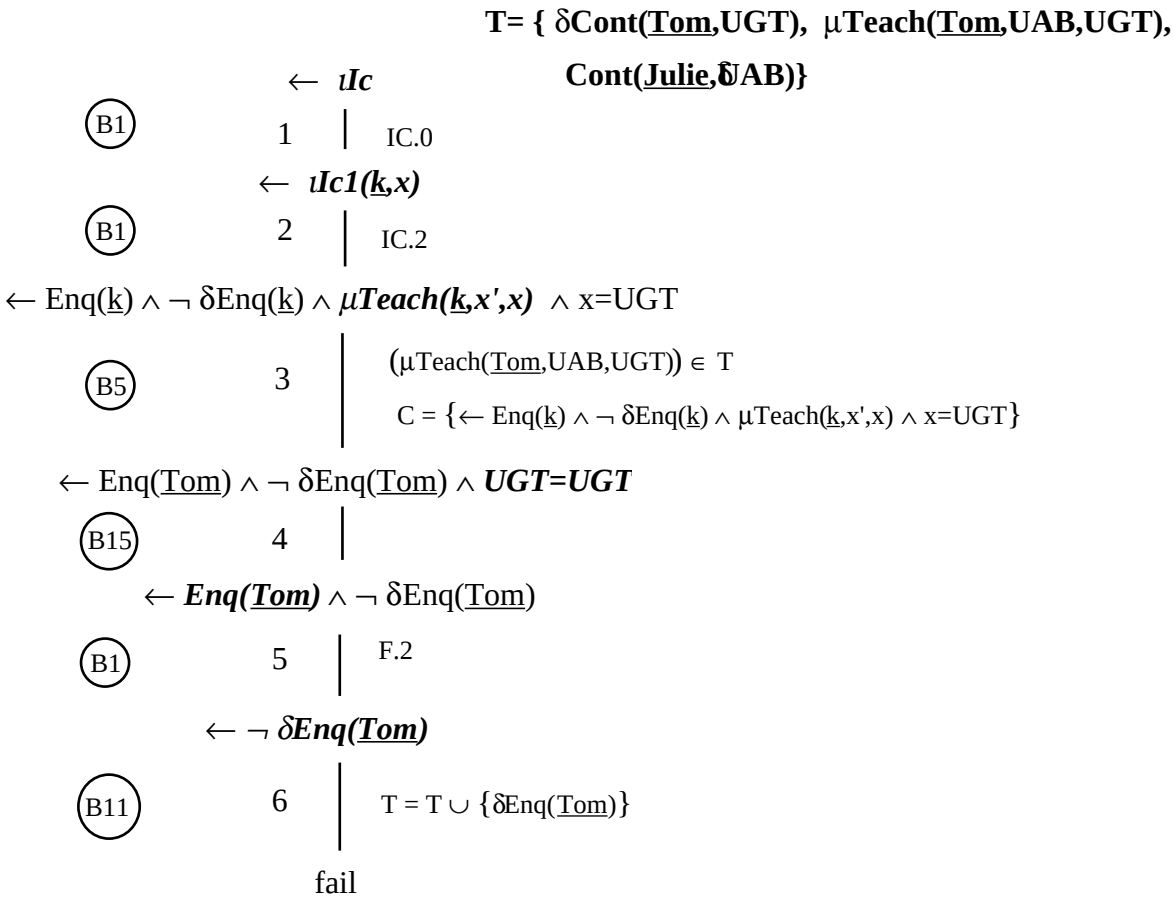


Fig. 4.2

Steps 1 and 2 are SLDNF derivations. At step 3, the selected literal is unified with the event μTeach(Tom,UAB,UGT) of the translation set T. To ensure that further additions into T do not make this goal succeed, we also include the goal in the condition set C. Steps 4 and 5 are SLDNF resolution steps resolving an evaluable predicate to true and a base predicate that holds in the KB. In the last step, the only way to have the goal fail is to force the base event δEnq(Tom) which is done by means of a subsidiary derivation rooted at ← δEnq(Tom). This derivation is not shown in the above example, but it causes the inclusion of δEnq(Tom) in the translation T.

Then, in the initial derivation of figure 4.1 the empty clause is reached, and the derivation finishes obtaining the solution T={δCont(Tom,UGT), μTeach(Tom,UAB,UGT), δCont(Julie,UAB), δEnq(Tom) }. Note that this solution satisfies the update request and maintains knowledge base consistency.

There is a different way, not shown in figure 4.1, in which the empty clause can be reached. We can resolve with the second modification event rule (M3), and we obtain the solution T={μCont(Tom,UGT,UPC), μTeach(Tom,UAB,UGT), δCont(Julie,UAB), δEnq(Tom) }, where "UPC" is a value given by the user or assigned by default.

4.3 Formalization of Our Method

In this section, we give a formal definition of our method for obtaining all minimal solutions that satisfy an update request and maintain knowledge base consistency. As it has been pointed out in the previous example, the Events Method is an interleaving of two activities: 1) satisfying an update by including base events in the translation set, and 2) ensuring that the updates induced by these base events are not contradictory with the update nor with the integrity constraints. These two activities are performed during *constructive* and *consistency derivations*, respectively, as defined below.

Let u be an update request. T will be a solution if there exists a *constructive derivation* from ($\{\leftarrow u \wedge \neg \iota Ic\}$ $\varnothing$ $\varnothing$) to ([] T C). Base events contained in T correspond to the updates (insertions and/or deletions and/or modifications) of base facts that must be performed on the extensional part of the knowledge base in order to satisfy the update request and maintain knowledge base consistency.

In order to define constructive and consistency derivations, we need to introduce the following convention. Let G_i be a goal that has the form $\leftarrow L_1 \wedge \ldots \wedge L_k$. We denote by $G_i \backslash L_j$, goal G_i where the literal L_j has been removed. Then, $G_i \backslash L_j = \leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge L_{j+1} \wedge \ldots \wedge L_k$. Note that if $G_i = \leftarrow L_j$ then $G_i \backslash L_j = []$.

Constructive Derivation Description

A *constructive derivation* from (G_1 T_1 C_1) to (G_n T_n C_n) via a safe selection rule R 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 and it is not a base event, then $G_{i+1} = S$, $T_{i+1} = T_i$ and $C_{i+1} = C_i$, where S is the resolvent of some clause in A(K) with G_i on the selected literal L_j

A2) If L_j is a positive ground 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 positive ground base event and $L_j \notin T_i$:

A3.1) $L_j = \iota P(\mathbf{k},\mathbf{x})$: If $\neg \exists \mathbf{y}\ \iota P(\mathbf{k},\mathbf{y}) \in T_i$ and $\neg \exists \mathbf{y}\ P(\mathbf{k},\mathbf{y})$ that holds in the current KB.

A3.2) $L_j = \delta P(\mathbf{k},\mathbf{x})$: If $\neg \exists \mathbf{x'}\ \mu P(\mathbf{k},\mathbf{x},\mathbf{x'}) \in T_i$ and $P(\mathbf{k},\mathbf{x})$ holds in the current KB.

A3.3) $L_j = \mu P(\mathbf{k},\mathbf{x},\mathbf{x'})$: If $\delta P(\mathbf{k},\mathbf{x}) \notin T_i$ and $\neg \exists \mathbf{y'}\ \mu P(\mathbf{k},\mathbf{x},\mathbf{y'}) \in T_i$, $P(\mathbf{k},\mathbf{x})$ holds in the current KB and $\mathbf{x} \neq \mathbf{x'}$.

If $C_i = \{\leftarrow Q_1, \ldots, \leftarrow Q_k, \ldots, \leftarrow Q_n\}$ and there exist consistency derivations

from ($\{\leftarrow Q_1\}$ $T_i \cup \{L_j\}$ C_i) to ({} T^1 C^1), ……,

from ($\{\leftarrow Q_n\}$ T^{n-1} C^{n-1}) to ({} T^n C^n)

then $G_{i+1} = G_i \backslash L_j$, $T_{i+1} = T^n$ and $C_{i+1} = C^n$.

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 $\iota P(\mathbf{k},\mathbf{x})$ (or $\delta P(\mathbf{k},\mathbf{x})$ or $\mu P(\mathbf{k},\mathbf{x},\mathbf{x}')$) and $\exists \mathbf{y}\ \iota P(\mathbf{k},\mathbf{y}) \in T_i$ (or $\exists \mathbf{y}\ \delta P(\mathbf{k},\mathbf{y}) \in T_i$ or $\exists \mathbf{y},\mathbf{y}'\ \mu P(\mathbf{k},\mathbf{y},\mathbf{y}') \in T_i$), then $G_{i+1}=(G_i \backslash L_j)\{\mathbf{y}\backslash\mathbf{x},\mathbf{y}'\backslash\mathbf{x}'\}$, $T_{i+1}=T_i$ and $C_{i+1} = C_i$.

A5) If L_j is a non ground positive base event, and σ is a substitution such that:

A5.1) If $L_j=\iota P(\mathbf{k},\mathbf{x})$, then $\neg\ \exists \mathbf{y}\ P(\mathbf{k}\sigma,\mathbf{y})$ must hold in the current knowledge base.

A5.2) If $L_j=\delta P(\mathbf{k},\mathbf{x})$ and $\neg\ \exists \mathbf{x}'\ \mu P(\mathbf{k}\sigma,\mathbf{x}\sigma,\mathbf{x}') \in T_i$, then $P(\mathbf{k},\mathbf{x})\sigma$ must hold in the current knowledge base.

A5.3) If $L_j=\mu P(\mathbf{k},\mathbf{x},\mathbf{x}')$, $\delta P(\mathbf{k},\mathbf{x})\sigma \notin T_i$ and $(\mathbf{x}\neq\mathbf{x}')\sigma$, then $P(\mathbf{k},\mathbf{x})\sigma$ must hold in the current knowledge base.

If $C_i = \{\leftarrow Q_1, \ldots, \leftarrow Q_k, \ldots, \leftarrow Q_n\}$, and there exist consistency derivations

from $(\{\leftarrow Q_1\}\ T_i \cup \{L_j\sigma\}\ C_i)$ to $(\{\}\ T^1\ C^1)$, ……,

from $(\{\leftarrow Q_n\}\ T^{n-1}\ C^{n-1})$ to $(\{\}\ T^n\ C^n)$

then $G_{i+1} = G_i \backslash L_j\sigma$, $T_{i+1} = T^n$ and $C_{i+1}= C^n$.

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

A6) If L_j is a base or derived predicate, negative and old, then $G_{i+1} = G_i \backslash L_j$, $T_{i+1} = T_i$ and $C_{i+1} = C_i$ if, using SLDNF resolution the goal $\leftarrow \neg L_j$ fails finitely.

A7) If L_j is a 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 $C_{i+1} = C_i \cup \{\leftarrow \neg L_j\}$.

A8) If L_j is a negative, new or auxiliary 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 $C_{i+1}= C'$.

A9) If L_j is an evaluable predicate and its evaluation is true, then $G_{i+1}= G_i \backslash L_j$, $T_{i+1}= T_i$ and $C_{i+1}=C_i$.

Steps corresponding to rules A1), A2), A4) and A6) are SLDNF resolution steps. In case A3), the selected base event is included in the translation set T_i if there is no incompatible events with it in T_i, and if we can enforce that this base event does not falsify the consistency of any condition in C_i. In some cases, this test may cause the addition of new elements to T and/or to C. Note that if $C_i = \emptyset$, consistency derivations must not be performed because there is no condition to satisfy.

In case A5) a non fully ground positive base event is selected. We have to ground it in order to delimit the base update to which it is referred (sub-steps A5.1, A5.2 and A5.3). In A5.1) it can be done by assigning default values or asking the user. In A5.2) it must be done seeking for facts in the knowledge base that can be unified with $P(\mathbf{k},\mathbf{x})$. In A5.3) it can be done assigning default values or asking the user for the **x'** value, and looking at the knowledge base for the **x** value. Once the base update is fully delimited, we proceed in the same way as in case A3).

In case A7) selected base events are added to the condition set C in order to enforce that they will not be included in the translation set afterwards. In the case corresponding to rule A8), we

get the next goal if we can enforce consistency for the selected literal. This may also cause new additions to T_i and/or to C_i. Auxiliary predicates are needed in order to maintain the A(K) allowed.

Case A9) is also an SLDNF resolution step where the selected literal is a ground evaluable predicate that must be valued true.

Consistency Derivation Description

A *consistency derivation* from $(F_1\ T_1\ C_1)$ to $(F_n\ T_n\ C_n)$ via a safe selection 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 $H_i \cup F'_i$ and, *for some* j=1…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, and k>1, then $F_{i+1} = H_i \backslash L_j \cup F'_i$, $T_{i+1} = T_i$ and $C_{i+1} = C_i$ if, using SLDNF resolution, the goal $\leftarrow \neg L_j$ fails finitely.

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 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 negative 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 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 auxiliary 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 auxiliary 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 $C_{i+1} = C'$.

B14) If L_j is an evaluable predicate and its evaluation is true and k>1, then $F_{i+1} = H_i \backslash L_j \cup F'_i$, $T_{i+1} = T_i$ and $C_{i+1} = C_i$.

B15) If L_j is an evaluable predicate and its evaluation is false, then $F_{i+1} = F'_i$, $T_{i+1} = T_i$ and $C_{i+1} = C_i$.

Steps corresponding to rules B1), B2), B3), B7), B8), B9), B10), B14) and B15) are SLDNF resolution steps. In case B4) the current branch is dropped from the consistency derivation because the subset of T already determined enforces failure for it. Moreover, the current goal H_i must be included in the condition set C_i to guarantee that new additions to T_i will not make this branch succeed.

In cases B5) and B6) new conditions are added to C_i if they are consistent with the subset of T already determined. Concretely, in case B5) this consistency is verified, while in case B6) the current branch already fails and it can be dropped. In case B11) the current branch is dropped if there exists a constructive derivation for the negation of the selected literal. In cases B12) and B13) 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. Once the set of translations has been obtained, a simple step removes from it all those translations that are subsumed by other.

5 Comparison with the Events Method [TO93]

The method proposed in this paper extends the work reported in [TO93] in three main directions:

a) We take into account information about the keys of base and derived predicates.
b) We do not only deal with insertions and deletions, but also with modifications of base and view predicates.
c) Maintenance of key integrity constraints is guaranteed by the own definition of the method.

The differences between our method and the Events Method are mainly due to the fact that we deal with modifications of base and view predicates as a new basic update operation.This extension has mainly been possible because we consider the procedure proposed in [TO92, Urp93] in spite of that of [Oli91] in order to automatically derive the transition and event rules for a given knowledge base.

The differences between both methods can be stated in three main directions. First, the semantic of insertion and deletion update requests has changed with respect to their semantic in

the Events Method. Second, non-minimal solutions in the Events Method may become minimal solutions in our method. Third, due to the powerful simplifications applied when deriving the augmented knowledge base and to the inclusion of key constraints maintenance into the translation process our method is, in general, more efficient than the Events Method.

These differences are described in sections 5.1, 5.2 and 5.3, respectively. In section 5.4 we point out other features of the Events Methed and discuss whether they can be applied in our method or not.

5.1 Comparing Basic Updates (insertion, deletion and modification)

In our method we have incorporated the modification as a new type of basic update request, while in the Events Method they are handled as a deletion and an insertion. This has caused addition induced changes in the semantic of the existing update requests (insertion and deletion) with respect to their meaning in the Events Method. These changes are due to the different definition of insertion and deletion events with respect to their definition in of the Events Method.

As an example, consider the following knowledge base:

$$P(\underline{A},1)$$
$$R(\underline{k},x) \leftarrow P(\underline{k},x) \wedge \neg Q(\underline{k},x)$$

A user update requesting the inclusion of the fact Q($\underline{B}$,2) would correspond to an event ιQ($\underline{B}$,2) in both methods. That is, this request is understood as an insertion request in both cases. In a similar way, a request foe deleting the fact P($\underline{A}$,1) would correspond to a deletion event δP($\underline{A}$,1) in both cases.

However, a user request for including the fact P($\underline{A}$,3) would correspond to an insertion event ιP(A,3) in the Events Method and to a modification event μP($\underline{A}$,1,3) in our method because the fact P($\underline{A}$,1) holds in the current knowledge base. This difference is caused by taking into account the information provided by the keys of base and derived predicates, what has implied changes in the definition of the basic events. Thus, we can state that insertion and deletion requests in our method are more specialized (focused) than in the Events Method because we make use of the modification request in some specific cases. Updates of derived facts behave in a similar way.

5.2 Minimal Solutions

Considering a modification as a new basic update causes also changes respect to the minimal solutions obtained by our method and by the Events Method. In both cases, a solution T is considered to be minimal if no proper subset of T is itself a solution. Then, and due to the modification event, a minimal solution in our method may not be considered minimal in the Events Method. For this reason, the number of solutions obtained may differ in some cases.

As an example, assume that we have obtained two solutions $T_1 = \{\delta P(\underline{A},1)\}$ and $T_2 = \{\mu P(\underline{A},1,3)\}$. As no proper subset of T_1 nor T_2 is itself a solution, both are considered minimal in our method. In the Events Method, these two solutions correspond to $T_1 = \{\delta P(A,1)\}$ and $T_2 = \{\delta P(A,1), \iota P(A,3)\}$. In this case, T_2 is not minimal because there is a subset of it (T_1) that is itself a solution. Then the Events Method would only generate the solution $T_1 = \{\delta P(A,1)\}$.

5.3 Comparison with Respect to Extensional Knowledge Base Accesses

In this section we will compare the number of accesses to the knowledge base needed by our method to obtain all solutions with respect to the accesses needed by the Events Method.

We consider only accesses to the extensional part of the knowledge base (KB). That is, accesses to the base facts. We do not count accesses to deductive rules nor integrity constraints because the intensional part of the KB may be resident in main memory or because some partial evaluation work can be done at compilation time [LlS91].

In order to make this comparison, we show an example where we have considered insertion, deletion and modification requests separately.

Example 5.3.1: *Example without non-key integrity constraints*

In order to simplify the comparison, we will consider in this example a knowledge base that contains only key integrity constraints. However, the conclusions that will be stated apply also for the case where non-key integrity constraints are taken into account. Let the intensional KB of this example, and relevant event rules be:

DR.1 $P(\underline{k},x) \leftarrow S(\underline{k},x) \wedge \neg R(\underline{k},x)$

M.1 $\mu P(\underline{k},x,x') \leftarrow \mu S(\underline{k},x,x') \wedge \neg R(\underline{k},x') \wedge \neg \iota R(\underline{k},x') \wedge \neg Aux1(\underline{k},x') \wedge \neg R(\underline{k},x).$
M.2 $\mu P(\underline{k},x,x') \leftarrow \mu S(\underline{k},x,x') \wedge \delta R(\underline{k},x').$
M.3 $\mu P(\underline{k},x,x') \leftarrow \mu S(\underline{k},x,x') \wedge \mu R(\underline{k},x',y).$
A.2 $Aux1(\underline{k},x) \leftarrow \mu R(\underline{k},z,x)$

Modification request: assume that the previous knowledge base contains the following facts:

F.1 S (A, 1).
F.2 R (A, 3).
F.3 S (B, 1).

and let the the modification request be $\mu P(\underline{A},1,2)$.

In *our method*, the solutions are obtained by having a failed SLDNF derivation of $A(K) \cup \{\leftarrow \mu P(\underline{A},1,2)\}$ succeed. Figure 5.1, which shows part of this derivation, helps us to explain the number of accesses to the extensional part of the knowledge base needed to obtain all the solutions.

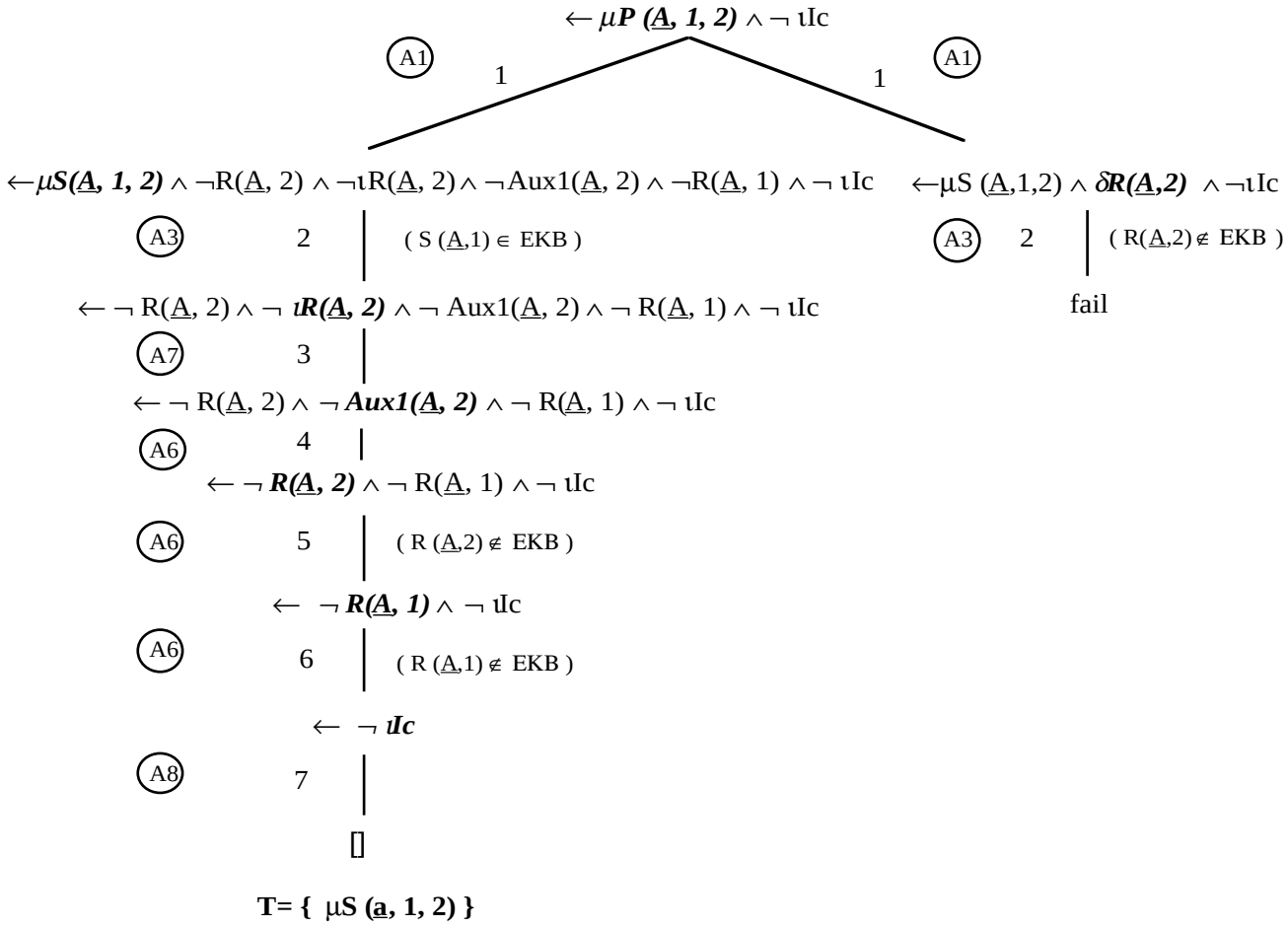


Fig. 5.1

Step 1 in the left derivation requires an access to the intensional KB and, then, it is not computed. At step 2, before adding the modification base event to T, it must be checked if S(A,1) holds in the current KB. Thus, an access to the extensional KB is necessary. At step 3, to incorporate the condition ←ιR(A,2) to C, no accesses to the extensional KB are needed. Step 4 only requires access to the intensional part. At steps 5 and 6 we have negative base facts, their falsity must be checked by looking in the extensional KB. At step 7, we only have to access the intensional KB to notice that no rule exists for ιIc. Then, we get the empty clause and the solution T={ μS(A,1,2) } is obtained. Our method has accessed 3 times to the extensional knowledge base (steps 2, 5 and 6).

The right branch fails because the event δR(A,2) is not feasible, but it needs to access once to the extensional KB to prove that R(A,2) is false. Selecting clause M.3 at step 1 of the previous derivation we get a failed derivation that also needs to access once to the extensional KB (this derivation is not shown in the above example).

Then, there is only one solution T={ μS(A,1,2) } that satisfies the update request μP(A,1,2) and maintains knowledge base consistency in the given knowledge base. Our method needs to access 5 times to the extensional part in order to obtain this solution.

Modification requests are handled in the *Events Method* by means of a deletion and an insertion. Moreover, this method needs to define an integrity constraint for each non-key argument of base and derived predicate. When applying the Events Method to the previous update request, 25 accesses are needed to obtain the solution $T^*=\{\delta S(A,1), \iota S(A,2)\}$. Note that this solution is equivalent to the solution obtained by our method, but the number of accesses needed is much greater (in this example, our method improves an 80% on the number of accesses).

Deletion request: Consider the same extensional knowledge base than in the previous update request, and assume that the user requests the deletion of the fact $\delta P(\underline{A},1)$. In this case, *our method* obtains three different minimal solutions: $T_1=\{\mu R(\underline{A},3,1)\}$, $T_2=\{\delta S(\underline{A},1)\}$ and $T_3=\{\mu S(\underline{A},1,3)\}$. To get them, it needs to access 10 times to the extensional knowledge base.

Applying the *Events Method* to the same delete request, only two minimal solutions: $T^*_1 = \{\delta R(A,3), \iota R(A,1)\}$ and $T^*_2=\{\delta S(A,1)\}$ are obtained, and 7 accesses to the extensional knowledge base are needed. Thus, in this case the Events Method is a little bit more efficient than ours (it needs 7 accesses in spite of 10).

As we mentioned in section 5.2, the number of solutions obtained in our method and in the Events Method may differ because minimal solutions in our method may not be considered minimal in the Events Method. This is a typical example where the solution $T^*_3=\{\delta S(A,1), \iota S(A,3)\}$ (corresponding to our solution T_3) is not considered in this last method because it is not minimal.

As a conclusion, we can state from this example that in general our method needs more accesses to the extensional KB than the Events Method when dealing with deletion requests of derived predicates. The reason is that we have an alternative way to translate this deletion by considering the modification as a basic update operation, which implies a growth in the number of event rules that can be resolved with. Moreover, this is the reason why the number of obtained solution is bigger in our method.

Insertion request: Consider the same intensional knowledge base as in example 5.3.1, and assume the insertion request $\iota P(\underline{C},1)$. In this case, we have detected that the efficiency comparison depends on the contents of the extensional knowledge base. Thus, we show a table for each method where, given that insertion request, we analyse the result of the translation process in three different extensional KB.

The first table refers to the application of *our method*.

Ext. KB	empty	R(c,1)	S(c,2), R(c,2)
accesses	7	9	8
solutions	T={ ιS(c,1) }	T_1={ ιS(c,1), δR(c,1)}	T={ μS(c,2,1) }
	–	T_2={ιS(c,1),μR(c,1,9)}	–

In this example, our method may require 7, 9 or 8 accesses to insert the derived fact P(C,1), depending on the contents of the extensional KB. Moreover, we obtain only one solution for the first and third case, but two solutions for the second one.

The table below corresponds to the application of the *Events Method.*

Ext. KB	empty	R(c,1)	S(c,2), R(c,2)
accesses	6	11	8
solutions	T={ ιS(c,1) }	T={ ιS(c,1), δR(c,1)}	T={ δS(c,2), ιS(c,1)}

The Events Method needs 6, 11 or 8 accesses to satisfy the update request. In this case, it only obtains one solution for each case.

As for delete requests, our method obtains some additional minimal solutions not obtained by the Events Method. Moreover, for insertion requests there is not a clear improvement of efficiency of accesses to the extensional KB because depending on the contents of the extensional KB, our method may need more or less accesses than the Events Method.

5.4 Additional Aspects

In order to conclude the comparison between our method and the Events Method, we would like to comment some additional aspects. First, the Events Method can handle updates of base and view facts and also other kinds of updates like qualified updates and insertion and deletion of deductive rules and integrity constraints. At the moment, our method is only able to deal with updates of base and view predicates. The other types of updates are left for further work.

Second, in [TO93] several important additional features of the Events Method like prevention of side effects on other views and repairing inconsistent knowledge bases are presented. All these additional features can also be applied in our method.

6 Comparison With Previous Work

In this section, we compare in detail our method for updating consistent knowledge bases with the approaches taken by some of the methods mentioned in the introduction. We will only consider these methods that, like ours, follow the execution-time approach to integrity constraints maintenance.

As a common comparison with these methods, we would like to remark that none of them deals with modifications as a basic update operation. That is, they consider only insertions and deletions. Moreover, they do not take into account the information about the keys of the predicates and, then, they do not include maintenance of key constraints in the management of updates during the translation process. Finally, none of them can handle transition integrity constraints, while we do.

6.1 Moerkotte and Lockemann's Method [ML91]

Moerkotte and Lockemann approach the problem of reactive consistency control in the context of definite deductive databases. When a transaction violates some integrity constraint, 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, causes are transformed into *repairs* by syntactic modification.

Moerkotte and Lockemann's method inspired our approach to integrity constraints maintenance. However, they restrict to a particular case of databases, where view predicates and integrity constraints can only be defined by means of Horn rules, and they consider only flat transactions, that is, transactions that consist on updates of base facts. On the contrary, our method allows negation to appear in the body of clauses and deals with updates of base facts and view updates.

6.2 Wüthrich's Method [Wüt93]

Wüthrich's method distinguishes clearly two steps to obtain the solutions. The first step consists of unfolding the update request until no more unfolding can be performed. In the second step, the unfolded formula $\hat{U}$ is considered and it tries to deduce updates $< I, D >$ of the facts F

of the old knowledge base such that Û becomes true with regard to (F ∪ I) - D. The generation of these updates is done by means of two special procedures: Insert and Delete. Knowledge base consistency is taken into account by conjunctively appending the integrity constraints to the update.

A first problem of this method is that, in some cases, it can not generate some valid solution. This is mainly due to the fact that it assumes there exists an ordering for dealing with the deductive rules and integrity constraints involved in the update, which will lead to the generation of a solution. However, this ordering does not always exist. As an example consider a consistent knowledge base containing the following base facts and integrity constraints:

Node (A)
Node (B)
Edge (A,B)
Edge (B,A)
Ic1 ← Node (x) ∧ ¬ ∃y Edge (x,y)
Ic2 ← Node (x) ∧ ¬ ∃z Edge (z,x)
Ic3 ← Edge (x,y) ∧ ¬ Node (x)
Ic4 ← Edge (x,y) ∧ ¬ Node (y)

and let the update request be the insertion of Edge (A,C). In this case, Wüthrich's method could not obtain the solution T ={ιEdge(A,C),ιNode(C),ιEdge(C,D),ιNode(D),ιEdge(D,B)}. On the contrary, our method always gets all valid solutions. We would like to remark that this problem will mainly appear when the knowledge base contains referential integrity constraints, such as the above ones.

A second problem of this method is that it does not necessarily generate minimal solutions because it does not test whether a base or view fact is already present in the old knowledge base when suggesting to add or delete it. This problem may be important because in some cases Wüthrich's method could not obtain the minimal solutions. Moreover, it may also imply a lot of unnecessary work. As an example consider the following knowledge base:

S (A,B)
P (x) ← Q (x) ∧ R (x)
R (x) ← S (x,y)

and let the update request be the insertion of P(A). In this method only the solution T={ιq(A),ιs(A,D)} may be obtained where D is a value given by the user or assigned by default. On the contrary, our method would only obtain the minimal solution T = {ιq(A)}.

7 Conclusions and Further Work

In this paper, we have proposed a new method for updating knowledge bases following the execution-time approach for integrity constraints maintenance. That is, given a consistent knowledge base and an update request, our method automatically obtains at run time all possible changes of the base facts such that the update is satisfied and consistency is maintained.

Our method can deal with updates which consist of a set of insertions, deletions and/or modifications of base and derived predicates. And it includes enforcement of primary and alternate key constraints in the management of updates during the translation process. We would like to remark that these features are not present in none of the methods we know of within the same approach to integrity constraints maintenance.

With respect to the future work, we want to extend our method to the general case (*existential key case*) where keys of the literals appearing in the body of the deductive rules for a derived predicate P must not be a subset of the key of P.

Furthermore, we plan to further improve the efficiency of our method by considering new optimizations (mainly for delete request) and by incorporating maintenance of more general constraints into the definition of our method.

We would also like to extend the kinds of updates we can deal with by considering qualified updates and insertions and deletions of deductive rules and integrity constraints.

Acknowledgements

We are grateful to D. Costal, C. Martín, A. Olivé, J. A. Pastor, C. Quer, M.R. Sancho, J. Sistac and T. Urpí for many useful comments and discussions.

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

References

[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.

[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.

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

[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.

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

[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.

[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.

[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.

[Pas92] Pastor, J.A. "Deriving Consistency-preserving Transaction Specifications for (View)Updates in Relational Databases", Third Int. Workshop on the Deductive Approach to Information Systems and Databases, Roses, Catalonia, 1992, pp. 275-300.

[Ten92] Teniente, E. "El mètode dels esdeveniments per a l'actualització de vistes en bases de dades deductives", PhD Thesis, 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.

[TO93] Teniente, E.; Olivé. A. "Updating Consistent Knowledge Bases", LSI Internal Report, Barcelona, 1993.

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

[Urp93] Urpí, T. " El mètode dels esdeveniments interns per al càlcul de canvis en Base de Dades Deductives", PhD Thesis, Barcelona, 1993 (in catalan).

[Wüt93] Wüthrich, B. "On Updates and Inconsistency Repairing in Knowledge Bases", Int. Conf. on Data Engineering, Vienna, 1993, pp. 608 - 615.