A Common Framework for Classifying and Specifying Deductive Database Updating Problems

(preliminary version)

Ernest Teniente
Toni Urpí

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

Abstract

Several problems may arise when updating a deductive database. Up to now, the general approach of the research related to deductive database updating problems has been to provide specific methods for solving particular problems. However, all these methods are explicitly or implicitly based on a set of rules that define the changes that occur in a transition from an old state of a database to a new, updated state. Therefore, these rules provide the basis of a framework for classifying and specifying these problems.

In this paper we propose to use the event rules [Oli91], which explicitly define the insertions and deletions induced by an update, for such a basis. We also define two interpretations of these rules which provide a common framework for classifying and specifying deductive database updating problems such as view updating, materialized view maintenance, integrity constraints checking, integrity constraints maintenance, repairing inconsistent databases, integrity constraints satisfiability or condition monitoring. Moreover, these interpretations allow us to identify and to specify some problems that have received little attention up to now like enforcing or preventing condition activation. By considering only a unique set of rules for specifying all these problems, we want to show that it is possible to provide general methods able to deal with all these problems as a whole.

July 1994

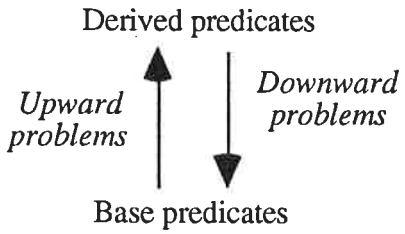
1 Introduction

Deductive databases 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, deductive databases 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.

Several problems may arise when updating a deductive database [Abi88]. During last years much research has been devoted to different database updating problems like view updating [Dec90, KM90, GL91, TA91, TO92], materialized view maintenance [CW91, HD92, UO92, GMS93], integrity constraints checking [BDM88, SK88, DW89, Küc91, Oli91, GCM+94], integrity constraints maintenance [CW90, ML91, Wüt93] or condition monitoring [RCB+89, HCK+90, QW91].

Up to now, the general approach of the research related to deductive database updating problems has been to provide specific methods for solving particular problems. However, all these methods are explicitly or implicitly based on a set of rules that define the changes that occur in a transition from an old state of a database to a new one. Therefore, these rules provide the basis of a framework for classifying and specifying these problems. We propose to use the event rules [Oli91], which explicitly define the insertions and deletions induced by an update, for such a basis.

In this paper we define two interpretations of the event rules: the upward interpretation and the downward one, which allow us to classify deductive database updating problems in two different types: upward problems and downward ones. *Upward problems* are concerned with computing the changes on derived predicates induced by a transaction which consists of a set of changes on base facts. On the other hand, *downward problems* are concerned with determining the possible transactions that satisfy a given set of changes on derived predicats. The following figure illustrates this idea:



We show how to specify deductive database updating problems in terms of the upward and the downward intepretations of the event rules. By considering only a unique set of rules for specifying all these problems, we want to show that it is possible to provide general methods able to deal with

all these problems as a whole. Therefore, we could uniformly integrate view updating, materialized view maintenance, integrity constraints checking, integrity constraints maintenance, condition monitoring and other deductive database updating problems into an update processing system.

The interpretations reported here can be seen as parallel to the notions of backward and forward reasoning [Kow83]. Their main differences rely on the fact that backward and forward reasoning are defined in terms of deductive rules and are used for query processing, while upward and downward interpretations are defined in terms of the event rules and are used for update processing.

Finally, it is important to point out that we are not proposing a new method for change computation. What we propose is to use the event rules and their interpretation as a common framework for classifying and specifying deductive database updating problems. A particular implementation of these interpretations would produce a particular method for change computation.

This paper is organized as follows. Next section reviews basic concepts of deductive databases. Section 3, shortly reviews and adapts the concepts of event, transition rules and event rules as presented in [Oli91]. Section 4 describes the upward and downward interpretations of the event rules. In section 5, we use these interpretations in order to classify and specify upward and downward deductive database updating problems. Finally, in section 6 we present our conclusions and point out future work. 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 deductive databases [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 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 deductive database 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 deductive database 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 $Ic1 \leftarrow L_1 \wedge ... \wedge L_n$.

A *deductive database* D is a triple D = (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 deductive database and the sets DR and IC is called the *intensional* part.

We assume that deductive database 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 deductive database can be defined in this form [BR86].

As usual, we require that the deductive database 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.

3 Transition and Event Rules

In this section, we shortly review and adapt the concepts and terminology of *events*, transition and event rules, as presented in [Oli91]. The event and transition rules, and their interpretation (described in the next section) constitute the common framework for classifying and specifying deductive database updating problems. In a later section, we will discuss the use of these rules for specifying these problems.

3.1 Events

Let D^o be a deductive database, T a transaction and D^n the updated deductive database. We say that T induces a transition from D^o (the old state) to D^n (the new state). We assume for the moment that T consists of an unspecified set of base facts to be inserted and/or deleted.

Due to the deductive, T may induce other updates on some derived predicates. Let P be one of such predicates, and let P^o and P^n denote the same predicate evaluated in D^o and D^n, respectively. Assuming that a fact $P^o(\mathbf{C})$ holds in D^o, where **C** is a vector of constants, two cases are possible:

a.1. $P^n(\mathbf{C})$ also holds in D^n (both $P^o(\mathbf{C})$ and $P^n(\mathbf{C})$ are true).
a.2. $P^n(\mathbf{C})$ does not hold in D^n ($P^o(\mathbf{C})$ is true but $P^n(\mathbf{C})$ is false).

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

b.1 $P^o(\mathbf{C})$ also holds in D^o (both $P^o(\mathbf{C})$ and $P^n(\mathbf{C})$ are true).
b.2. $P^o(\mathbf{C})$ does not hold in D^o ($P^n(\mathbf{C})$ is true but $P^o(\mathbf{C})$ 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 asscciate to each derived P an *insertion event predicate ιP* and a *deletion event predicate δP*, defined as:

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

where **x** is a vector of variables. From the above, we then have the equivalencies [Urp93]:

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

If P is a derived predicate, ιP and δP facts represent induced insertions and induced deletions, respectively. 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. Therefore, we assume from now on that T consists of an unspecified set of insertion and/or deletion base event facts.

3.2 Transition Rules

Let us consider a derived predicate P of the deductive database. The definition of P consists of the rules in the deductive database having P in the conclusion. Assume that there are m ($m \geq 1$) such rules. For our purposes, we rename predicate symbols in the conclusions of the m rules by $P_1, ..., P_m$, change the implication by an equivalency and add the clause:

$$(5) \quad P \leftrightarrow \bigvee_{i=1}^{i=m} P_i$$

Consider now one of the rules $P_i(\mathbf{x}) \leftrightarrow L_1 \wedge ... \wedge L_n$. When this rule is to be evaluated in the new state, its form is $P^n{}_i(\mathbf{x}) \leftrightarrow L^n{}_1 \wedge ... \wedge L^n{}_n$, where $L^n{}_j$ (j = 1... n) is obtained by replacing the predicate Q of L_j by Q^n. 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 $P^n{}_i$ in terms of old state predicates and events.

More precisely, if $L^n{}_j$ is a positive literal $Q^n{}_j(\mathbf{x}_j)$ we apply (3) and replace it by:

$$(Q^o{}_j(\mathbf{x}_j) \wedge \neg \delta Q_j(\mathbf{x}_j)) \vee \iota Q_j(\mathbf{x}_j)$$

and if $L^n{}_j$ is a negative literal $\neg Q^n{}_j(\mathbf{x}_j)$ we apply (4) and replace it by:

$$(\neg Q^o{}_j(\mathbf{x}_j) \wedge \neg \iota Q_j(\mathbf{x}_j)) \vee \delta Q_j(\mathbf{x}_j)$$

After distributing $\wedge$ over $\vee$, we get the transition rule for $P^n{}_i$, whose body is expressed in disjunctive normal form. Notice that there are 2^{ki} disjunctands (where k_i is the number of literals in the $P^n{}_i$ rule) and that literals in each disjuctand can be of three types: old database literals, base event literals and derived event literals.

Example 3.1: Consider the rule $P_1(x) \leftrightarrow Q(x) \wedge \neg R(x)$. In the new state, this rule has the form $P^n{}_1(x) \leftrightarrow Q^n(x) \wedge \neg R^n(x)$. Then, replacing $Q^n(x)$ and $\neg R^n(x)$ by their equivalent expressions given by (3) and (4) we get:

$$P^n{}_1(x) \leftrightarrow [(Q^o(x) \wedge \neg \delta Q(x)) \vee \iota Q(x)] \wedge [(\neg R^o(x) \wedge \neg \iota R(x)) \vee \delta R(x)]$$

and, after distributing $\wedge$ over $\vee$, we get the following transition rule:

$$P^n_1(x) \leftrightarrow [\ (Q^o(x) \wedge \neg\delta Q(x) \wedge \neg R^o(x) \wedge \neg\iota R(x)) \vee$$
$$(Q^o(x) \wedge \neg\delta Q(x) \wedge \delta R(x)) \vee$$
$$(\iota Q(x) \wedge \neg R^o(x) \wedge \neg\iota R(x)) \vee$$
$$(\iota Q(x) \wedge \delta R(x))\ ]$$

The body of the transition rule for P^n is a disjunctive normal form formed by the union of the disjunctive normal forms corresponding to all P^n_i.

For simplicity of the presentation, we will omit the sub index when P is defined by only one rule.

3.3 Insertion and Deletion Event Rules

Let P be a derived predicate. Insertion and deletion event rules of predicate P are defined respectively as:

(6) $\iota P(\mathbf{x}) \leftrightarrow P^n(\mathbf{x}) \wedge \neg P^o(\mathbf{x})$

(7) $\delta P(\mathbf{x}) \leftrightarrow P^o(\mathbf{x}) \wedge \neg P^n(\mathbf{x})$

where P^n refers to the transition rule of P and P^o refers to the current (old) state of the database. The event rules define the induced changes that happen in a transition from an old state of a database to a new, updated state.

We would like to point out that these rules can be intensively simplified, as described in [Oli91, UO92, UO94]. However, for the purpose of this paper it is sufficient to consider them as expressed above.

4. Interpretation of the Event Rules

The event rules define the induced changes that happen in a transition from an old state of a database to a new, updated state. These rules can be interpreted in two different ways according to the direction in which the equivalence is considered. As we will see, left implication defines the changes on derived predicates induced by changes on base predicates. On the other hand, right implication defines the changes on base predicates needed to satisfy changes on derived predicates. We call them upward and downward interpretation, respectively.

We would like to point out that we are not proposing methods for evaluating the event rules, but interpretations of them. Thus, a particular implementation of these interpretations could be based either on a top-down or on a bottom-up query evaluation procedure.

4.1 Upward Interpretation

The upward interpretation of the event rules defines the changes on derived predicates induced by changes on base predicates. We assume that these changes on base predicates are given by a transaction, which consists of a set of base event facts.

As stated before, the upward interpretation considers the left implication of the equivalence in the event rules. Therefore, in this interpretation the event rules corresponding to a derived predicate P are expressed in the following way:

$$\iota P(\mathbf{x}) \leftarrow P^n(\mathbf{x}) \wedge \neg P^o(\mathbf{x})$$
$$\delta P(\mathbf{x}) \leftarrow P^o(\mathbf{x}) \wedge \neg P^n(\mathbf{x})$$

whose intended meaning is that there will be an induced insertion (deletion) of a fact of P if the body of its corresponding event rule evaluates to true in the transition. _

The result of upward interpreting an event rule corresponding to a derived predicate P (succinctly, the upward interpretation of $\iota P(\mathbf{x})$ or $\delta P(\mathbf{x})$) is a set of derived event facts. Each of them corresponds to a change of a derived fact induced by the transaction.

To obtain this result, literals in the body of ιP and δP have to be interpreted in the following way. Old database literals ($P^o(\mathbf{x})$ and $\neg P^o(\mathbf{x})$) correspond to a query that must be performed in the current (old) state of the database, while new database literals ($P^n(\mathbf{x})$ and $\neg P^n(\mathbf{x})$) are handled by upward interpreting the transition rule of predicate P.

To upward interpret a positive new database literal P^n, literals in each disjunctand of the body of P^n must be interpreted as follows:

- An old database literal corresponds to a query that must be performed in the current state of the database. If there are no solutions for this query, the disjunctand does not define any induced change.

- A base event literal corresponds to a query that must be applied to the transaction. If there are no solutions for this query, the disjunctand does not define any induced change.

- A derived event literal defines the induced changes on a derived predicate and, therefore, it must be handled by upward interpreting its corresponding event rule. In the particular case of a negative derived event, e.g. $\neg\iota P(\mathbf{x})$, its upward interpretation corresponds to a condition whose truth value is given by the result of the upward interpretation of $\iota P(\mathbf{x})$. This condition will be true if the latter result does not contain any derived event fact and false otherwise.

The upward interpretation of a negative new database literal $\neg P^n$ corresponds to a condition whose truth value is given by the upward interpretation of the corresponding positive new database literal, as for negative derived events.

The following example illustrates the upward interpretation.

Example 4.1: Consider the following deductive database:

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

and assume a transaction T which consists of the deletion of the base fact R(B). In our notation, $T=\{\delta R(B)\}$. As it can be easily seen, this transaction induces only an insertion of the derived fact P(B). We are going to show how the upward interpretation of the event rules of P defines this induced insertion.

The event rule of P expressed in the upward interpretation is:

$$\iota P(x) \leftarrow P^n(x) \wedge \neg P^o(x)$$

Induced insertions on P are given by upward interpreting the literals in $(P^n(x) \wedge \neg P^o(x))$. The first literal is a new literal. Then, we have to consider the upward interpretation of its corresponding transition rule:

$$\begin{aligned} P^n(x) \leftarrow [\ & (Q^o(x) \wedge \neg\delta Q(x) \wedge \neg R^o(x) \wedge \neg\iota R(x)) \vee \\ & (Q^o(x) \wedge \neg\delta Q(x) \wedge \delta R(x)) \vee \\ & (\iota Q(x) \wedge \neg R^o(x) \wedge \neg\iota R(x)) \vee \\ & (\iota Q(x) \wedge \delta R(x))\ ] \end{aligned}$$

Consider the second disjunctand of $P^n(x)$: $Q^o(x) \wedge \neg\delta Q(x) \wedge \delta R(x)$. The first literal, $Q^o(x)$, is a database literal and, thus, it corresponds to a query that must be performed in the current state of the database. In this example, two possible values of x (x=A and x=B) exist such that Q(x) holds. On the other hand, $\neg\delta Q(x)$ is a negative base event literal and, since no base event $\delta Q(X)$ belongs to the transaction, this condition also holds. The third literal, $\delta R(x)$, is a positive base event literal and since the event $\delta R(B)$ belongs to the transaction, this query holds for x=B. Therefore, the whole disjunctand evaluates to true for x=B and, thus, $P^n(x)$ holds for x=B.

The second literal in the insertion event rule is an old database literal, $\neg P^o(x)$, which holds for in the current state for the value x=B that satisfies $P^n(x)$. Therefore, transaction T induces $\iota P(B)$. It can be similarly seen that this transaction does not induce any other change. Thus, the result of upward interpreting $\iota P(x)$ is the set $\{ \iota P(B) \}$.

4.2 Downward Interpretation

The downward interpretation of the event rules defines the changes on base predicates needed to satisfy changes on derived predicates. We assume that these changes on derived predicates are given by a set of derived event facts. In general, the downward interpretation may not be unique. That is, several sets of changes on base predicates that satisfy changes on derived predicates may exist. Each possible set constitutes a possible transaction that applied to the current state of the database will accomplish the required changes on derived predicates.

As stated before, the downward interpretation considers the right implication of the equivalence in the event rules. Therefore, in this interpretation the event rules corresponding to a derived predicate P are expressed in the following way:

$$\iota P(\mathbf{x}) \rightarrow P^n(\mathbf{x}) \wedge \neg P^o(\mathbf{x})$$
$$\delta P(\mathbf{x}) \rightarrow P^o(\mathbf{x}) \wedge \neg P^n(\mathbf{x})$$

whose intended meaning is that if there is an insertion (deletion) of a fact of P then it is necessary that the body of its corresponding event rule evaluates to true in the transition.

The result of downward interpreting an event rule corresponding to a derived predicate P with respect to a derived event fact $\iota P(\mathbf{X})$ or $\delta P(\mathbf{X})$ (succinctly, the downward interpretation of $\iota P(\mathbf{X})$ or $\delta P(\mathbf{X})$) is a disjunctive normal form, where each disjunctand defines an alternative to satisfy a change on this derived predicate[1]. Each disjunctand may contain positive base events facts, which constitute a possible transaction to be performed, and negative base events facts, representing requirements that the transition must satisfy.

To obtain this result, literals in the body of ιP and δP have to be interpreted in the following way. Old database literals ($P^o(\mathbf{x})$ and $\neg P^o(\mathbf{x})$) correspond to a query that must be performed in the current (old) state of the database, while new database literals ($P^n(\mathbf{x})$ and $\neg P^n(\mathbf{x})$) are handled by downward interpreting the transition rule of predicate P.

To downward interpret a positive new database literal, literals in each disjunctand of the body of P^n must be interpreted as follows:

---

[1] Sometimes, no set of changes of base predicates is obtained when considering the downward interpretation of a derived event ιP (resp. δP). In this case, two possible things may happen. First, P may (resp. may not) hold in the current state of the database and, then, the requested change on P does not make sense since it is already satisfied. Second, if P does not hold (resp. holds) and no set of changes on base events is obtained, the requested change on P can not be satisfied by considering only changes of base facts. In this latter case, updates of the extensional database should be considered in order to satisfy the requested change.

- An old database literal corresponds to a query that must be performed in the current state of the database. If there are no solutions for this query, the disjunctand does not define any alternative.

- A base event literal defines different alternatives of base fact updates to be performed, one for each possible way to instantiate this event. Note that, as we consider finite domains, the number of alternatives is always finite. In particular, a ground positive base event literal corresponds to a change on a base fact that must be performed, provided that the event definition is satisfied. Negative base event literals correspond to changes that must not be performed.

- A derived event literal corresponds to a set of changes on a derived predicate, one for each possible way to instantiate this event, that must be handled by downward interpreting its corresponding event rule. The downward interpretation of a negative derived event is defined as the disjunctive normal form of the logical negation of the result obtained by downward interpreting the corresponding positive derived event.

The downward interpretation of a negative new database literal $\neg P^n$ is defined as the disjunctive normal form of the logical negation of the result obtained by downward interpreting the corresponding positive new database literal, as for negative derived events.

The downward interpretation of a set of event facts is defined as the disjunctive normal form of the logical conjunction of the result of downward interpreting each event in the set.

The following example illustrates the downward interpretation.

Example 4.2: Consider again the same database as in example 4.1 :

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

and assume now that the insertion of the derived fact P(B) is requested. In our notation, $\iota P(B)$. Intuitively, it can be seen that the change on base predicates needed to satisfy this change on P(B) is $\delta R(B)$. We are going to show how the downward interpretation of the event rules of P define this change.

The insertion event rule of P expressed in the downward interpretation is:

$$\iota P(x) \rightarrow P^n(x) \wedge \neg P^o(x)$$

Then, changes on base predicates needed to satisfy $\iota P(B)$ are given by downward interpreting the literals in $(P^n(B) \wedge \neg P^o(B))$. The second literal is an old database literal which holds in the current state. On the other hand, the first literal is a new literal. Then, we have to consider the downward interpretation of its corresponding transition rule:

$$\begin{aligned} P^n(B) \rightarrow [\ & (Q^o(B) \wedge \neg\delta Q(B) \wedge \neg R^o(B) \wedge \neg\iota R(B)) \vee \\ & (Q^o(B) \wedge \neg\delta Q(B) \wedge \delta R(B)) \vee \\ & (\iota Q(B) \wedge \neg R^o(B) \wedge \neg\iota R(B)) \vee \\ & (\iota Q(B) \wedge \delta R(B))\ ] \end{aligned}$$

Consider the second disjunctand of this rule. The first literal, $Q^o(B)$ is a database literal which holds in the current state. The second literal, $\neg\delta Q(B)$, is a negative base event and, thus, it corresponds to a change that must not be performed. Finally, $\delta R(B)$ is a positive base event literal and, thus, it corresponds to a base fact update that must be performed. Therefore, from this disjunctand we obtain the alternative $(\delta R(B) \wedge \neg\delta Q(B))$. In a similar way, it can be seen that no other alternatives are obtained by considering the other disjunctands.

Thus, the final result of downward interpreting $\iota P(B)$ is $(\delta R(B) \wedge \neg\delta Q(B))$. Therefore, the application of the transaction $T=\{\delta R(B)\}$ to the current state of the database will accomplish the insertion of P(B).

5. Classifying and Specifying Deductive Database Updating Problems

In this section we show how to use the event rules and their interpretation for classifying and specifying deductive database updating problems. First, we need to endow a derived predicate with a concrete semantics. Many authors [DaW89, RCB89+, KüC91] have proposed to express an integrity constraint, a view (materialized or not) and a condition to be monitored as a derived predicate. In all cases, they have the same form as a deductive rule, but a concrete interpretation. Thus, in our example the derived predicate P can be expressed as:

$$\begin{aligned} Ic1(x) &\leftarrow Q(x) \wedge \neg R(x) \\ View(x) &\leftarrow Q(x) \wedge \neg R(x) \\ Cond(x) &\leftarrow Q(x) \wedge \neg R(x) \end{aligned}$$

where we consider that Ic1, View and Cond are interpreted accordingly to their concrete semantics.

Furthermore, we assume that there exists a global inconsistency predicate Ic, defined as Ic ← Ic1(x_1),..., Ic ← Icn(x_n), where x_i, i =1...n, is a vector of terms and n is the number of integrity constraints. Notice that if Ic holds the database is inconsistent because some integrity constraint is violated, whereas if Ic does not hold the database is consistent, that is, all integrity constraints are satisfied.

The upward and downward interpretation of the event rules corresponding to Ic, View and Cond allow us to classify the deductive database updating problems that have been addressed in the past years. Moreover, these interpretations allow us to identify some deductive database updating problems that, to our knowledge, have not been addressed up to now. This is summarized in Table 4.1. In the rest of this section, we will briefly review all these deductive database updating problems and we will explain how can they be specified in terms of the upward and the downward interpretations.

<table>
<tr><th colspan="2"></th><th>View</th><th>Ic</th><th>Cond</th></tr>
<tr><td rowspan="2">Upward
Interpretation</td><td>ιP</td><td rowspan="2">Materialized view maintenance</td><td>IC checking</td><td rowspan="2">Condition monitoring</td></tr>
<tr><td>δP</td><td>IC violation removal checking</td></tr>
<tr><td rowspan="4">Downward
Interpretation</td><td>ιP</td><td rowspan="2">View updating
View validation</td><td>Ensuring IC satisfaction</td><td rowspan="2">Enforcing condition activation
Condition validation</td></tr>
<tr><td>δP</td><td>Repairing inconsistent DB
IC satisfiability</td></tr>
<tr><td>T, ¬ιP</td><td rowspan="2">Preventing side effects</td><td>IC maintenance</td><td rowspan="2">Preventing condition activation</td></tr>
<tr><td>T, ¬δP</td><td>Maintaining DB inconsistency</td></tr>
</table>

Table 4.1

5.1 Upward Problems

5.1.1 Integrity Constraints Checking

There exists a large cumulative effort in the field of integrity constraints checking [BDM88, SK88, DW89, Küc91, Oli91, GCM+94]. The problem can be summarized in the following terms. Given a consistent database state and a transaction that consists of a set of base fact updates, the problem is to determine incrementally whether this transaction violates the integrity constraints.

In our framework this problem can be specified as the upward interpretation of ιIc, provided that Ic^o does not hold. The result of this upward interpretation is a set which either contains ιIc or is empty. If ιIc belongs to result then the transaction induces an insertion of Ic and, therefore, it violates some integrity constraint. Otherwise, the transaction does not violate any integrity constraint.

Example 5.1: Consider a deductive database with the following predicates:

La (x)	x is in labour age.
Works (x)	x works for some company.
U_benefit (x)	x receives an unemployment benefit.
Unemp (x)	x is unemployed. Every person that is in labour age and does not work is unemployed.
Ic1	Inconsistency predicate which states that all unemployees must receive an unemployment benefit.

Assume that the current content of the deductive database is the following:

La(Dolors)
U_benefit(Dolors)
$Unemp(x) \leftarrow La(x) \wedge \neg Works(x)$
$Ic1 \leftarrow Unemp(x) \wedge \neg U\_benefit(x)$

Relevant transition and event rules associated to this database are:

$\iota Unemp(x) \leftrightarrow Unemp^n(x) \wedge \neg Unemp^o(x)$
$\delta Unemp(x) \leftrightarrow Unemp^o(x) \wedge \neg Unemp^n(x)$
$\iota Ic1 \leftrightarrow Ic1^n \wedge \neg Ic1^o$

$$Unemp^n(x) \leftrightarrow [\ (La^o(x) \wedge \neg\delta La(x) \wedge \neg Works^o(x) \wedge \neg\iota Works(x)) \vee (La^o(x) \wedge \neg\delta La(x) \wedge \delta Works(x)) \vee (\iota La(x) \wedge \neg Works^o(x) \wedge \neg\iota Works(x)) \vee (\iota La(x) \wedge \delta Works(x))\ ]$$

$$Ic1^n \leftrightarrow [\ (Unemp^o(x) \wedge \neg\delta Unemp(x) \wedge \neg U\_benefit^o(x) \wedge \neg\iota U\_benefit(x)) \vee (Unemp^o(x) \wedge \neg\delta Unemp(x) \wedge \delta U\_benefit(x)) \vee (\iota Unemp(x) \wedge \neg U\_benefit^o(x) \wedge \neg\iota U\_benefit(x)) \vee (\iota Unemp(x) \wedge \delta U\_benefit(x))\ ]$$

Assume a transaction T={δU_benefit(Dolors)}. The upward interpretation of ιIc1 allows us to determine whether a transaction violates this integrity constraint. In this case, both $Ic1^n$ and $\neg Ic1^o$ hold and, therefore, the final result is {ιIc1}. Thus, Ic1 is violated and the transaction must be rejected.

On the other hand, there is also another problem that belongs to this category. Given an inconsistent database and a transaction that consist of a set of base fact updates, the problem is to check whether these updates restore the database to a consistent state. In our framework this problem can be specified as the upward interpretation of δIc, provided that Ic^o holds. In this case, the result of this upward interpretation is a set which either contains δIc or is empty. If δIc belongs to the result, then the transaction induces a deletion of Ic and, therefore, restores the consistency of the database. Otherwise, the database keeps the inconsistency.

Notice that both problems can also be complementary specified as the upward interpretation of $\neg\iota Ic$ and $\neg\delta Ic$, respectively. Thus, while the upward interpretation of ιIc checks whether the transaction violates some integrity constraint, the upward interpretation of $\neg\iota Ic$ checks whether the transaction does not violate any integrity constraint. In a similar way, we can interpret $\neg\delta Ic$.

5.1.2 Condition Monitoring

As stated before, general conditions can also be expressed as derived predicates. Condition monitoring refers to the problem of incrementally monitoring the changes on a condition induced by a transaction that consist of a set of base fact updates [RCB+89, HCK+90, QW91].

In our framework, changes induced in a given condition, *Cond(x)*, are specified as the upward interpretation of $\iota Cond(\mathbf{x})$ and $\delta Cond(\mathbf{x})$. The former, $\iota Cond(\mathbf{x})$, defines the changes meaning that **x** satisfy the condition after the application of the transaction, but not before. The latter, $\delta Cond(\mathbf{x})$, defines the changes meaning that **x** satisfy the condition before the application of the transaction, but not after.

On the complementary hand, the upward interpretation of $\neg\iota Cond(\mathbf{x})$ and $\neg\delta Cond(\mathbf{x})$ checks whether the transaction does not induce any change on this condition.

5.1.3 Materialized View Maintenance

A view can be materialized by explicitly storing its extension in the extensional database. Given a transaction that consists of a set of base fact updates, the materialized view maintenance problem consists of incrementally determining which changes are needed to update accordingly the materialized views [CW91, HD92, UO92, GMS93].

In our framework, given a materialized view, View*(x)*, the materialized view maintenance problem is specified as the upward interpretation of $\iota View(\mathbf{x})$ and $\delta View(\mathbf{x})$. The former, $\iota View(\mathbf{x})$, defines the insertions that must be performed into the extension of the materialized view, while the

later defines the deletions. Thus, for instance, if ιView(**X**) belongs to the result of the upward interpretation, then View(**X**) must be inserted into the extension of the materialized view

On the complementary hand, the upward interpretation of ¬ιView(**x**) and ¬δView(**x**) checks whether the transaction does not affect to this view.

5.2 Downward Problems

5.2.1 View Updating

View updating is concerned with determining how a request to update a view should be appropriately translated into updates of the underlying base 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, Kel86, LS91] as well as in deductive databases [Dec90, KM90, GL91, TA91, TO92].

In our framework, the view update problem can be specified as the downward interpretation of ιView(**X**) or δView(**X**), where View(**X**) is the derived fact to be inserted or deleted, respectively. The former defines the possible sets of base fact updates that satisfy the insertion of View(**X**), while the latter defines the possible sets of base fact updates that satisfy the deletion of View(**X**).

In general, a view update request consists of a set of insertions and/or deletions to be performed on derived predicates. In this case, we have to consider the downward interpretation of the whole set in order to obtain the translations that satisfy the view update request.

Example 5.2: Consider again the same database as in example 5.1, and assume now that the view update δUnemp(Dolors) is requested. The downward interpretation of δUnemp(Dolors) allow us to determine the possible translations that satisfy this request. $Unemp^o$(Dolors) holds in the current state. On the other hand, the downward interpretation of ¬$Unemp^n$(Dolors) is (δLa(Dolors) ∨ ιWorks(Dolors)), which is the logical negation of the result of downward interpreting $Unemp^n$(Dolors). Therefore, the downward interpretation of δUnemp(Dolors) is (δLa(Dolors) ∨ ιWorks(Dolors)). Thus, the possible translations that satisfy the view update request are: T_1 = {δLa(Dolors)} and T_2 = {ιWorks(Dolors)}.

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 combined with integrity constraints satisfaction. This could be done either by combining view updating and integrity

constraints checking or view updating and integrity constraints maintenance. Possible ways of performing this combination will be explained in section 5.3.

The downward interpretation of ιView(**X**) or δView(**X**) can also be used for specifying the problem of *view validation.* This problem can be stated as follows. Given a derived predicate View(**x**), the problem is to obtain at least one **X** for which some set of base fact updates that satisfies ιView(**X**) or δView(**X**) exists (notice that **X** must range over the domain of the terms in View). This can be useful for providing the database designer with a tool for validating certain aspects of the database definition. For instance, s/he could validate whether it is possible to reach a state with a non empty view extension. Provided that View^o does not hold, if there exists one **X** for which ιView(**X**) defines a set of base fact updates, then, it is possible to reach such a state. Otherwise, it is not possible.

5.2.2 Preventing Side Effects

Due to the deductive rules, non desired updates may be induced on some derived predicates when applying a transaction. We say that a side effect occurs when this happens. The problem of *preventing side effects* [Ten92] is concerned with determining a set of base fact updates which, appended to a given transaction, ensure that the application of the resulting transaction to the current state of the database will not induce the undesired side effects. In general, several solutions may exist and the user must select one of them.

Assume that we want to prevent that the application of a given transaction T will induce an insertion or a deletion of a derived fact View(**X**). In our framework, this problem can be specified as the downward interpretation of the set {T,¬ιView(**X**)} or {T,¬δView(**X**)}, respectively. The former defines base fact updates needed to guarantee that the insertion of View(**X**) is not induced by T, while the latter defines changes on base predicates needed to satisfy that the deletion of View(**X**) is not induced. In general, if we want to prevent all possible side effects over View, we have to take into account all possible values of **X**.

Example 5.3: Consider again the same database as in example 5.1, and assume now a transaction T={ιLa(Maria)}. Intuitively, it can be easily seen that the application of T would induce the insertion of Unemp(Maria). Assume that we want to prevent this side effect. The downward interpretation of {ιLa(Maria),¬ιUnemp(Maria)} defines resulting transactions that prevent this side effect.

The downward interpretation of ιLa(Maria) is ιLa(Maria). On the other hand, to downward interpret ¬ιUnemp(Maria) we have to take into account the logical negation of the downward

interpretation of the event rule corresponding to ιUnemp(Maria). $\neg$Unemp^o(Maria) holds in the current state, while the downward interpretation of Unemp^n(Maria) is (ιLa(Maria) $\wedge$ $\neg$ιWorks(Maria)). Therefore, the final result of downward interpreting $\neg$ιUnemp(Maria) is ($\neg$ιLa(Maria) $\vee$ ιWorks(Maria)).

As stated in section 4.2, the downward interpretation of a set of event facts is defined as the disjunctive normal form of the logical conjunction of the result of downward interpreting each event in the set. Therefore, the downward interpretation of {ιLa(Maria),$\neg$ιUnemp(Maria)} is [(ιLa(Maria) $\wedge$ $\neg$ιLa(Maria)) $\vee$ (ιLa(Maria) $\wedge$ ιWorks(Maria))]. Thus, the only possible resulting transaction that does not induce the insertion of Unemp(Maria) is T ={ιLa(Maria), ιWorks(Maria)}.

5.2.3 Repairing Inconsistent Databases

Sometimes it may happen that a database reaches an inconsistent state. In this case, it arises the problem of *repairing inconsistent databases* [Ten92]. Given an inconsistent database state, the problem is to obtain a set of updates of base facts which restore the database to a consistent state. In general, several solutions may exist and the database administrator should select one of them.

In our framework, this problem can be specified as the downward interpretation of δIc, provided that Ic^0 holds. Sets of base fact updates obtained by downward interpreting δIc correspond to the possible transactions that would induce a deletion of Ic and, therefore, that would restore database consistency.

The downward interpretation of δIc can also be used for *integrity constraints satisfiability* [BDM88]. This problem can be stated as follows. Given a set of integrity constraints and deductive rules, the problem is to determine whether there exists a state of the extensional database that satisfies all the integrity constraints.

In our framework, this problem can also be specified as the downward interpretation of δIc provided that Ic^o holds. If the downward interpretation of δIc defines at least one transaction, then the integrity constraints are satisfiable. Note that if Ic^o does not hold, all constraints are already satisfied in the current state of the database.

There is also another problem that belongs to this category. Given a set of deductive rules and integrity constraints, the problem is to determine whether there exists a state of the extensional database that violates some integrity constraint. This could be useful, for instance, in order to help the database designer to define a database that may never reach an inconsistent state. We denote this problem as *ensuring integrity constraints satisfaction.*

In our framework, this problem can be specified as the downward interpretation of ιIc. The possible sets of base fact updates defined by this interpretation correspond to the possible ways of turning the database into an inconsistent state. At definition time, the database designer should verify whether the downward interpretation of ιIc defines some transaction or not. If it does, an inconsistent state may be reached. Otherwise, it may not.

5.2.4 Integrity Constraints Maintenance

There exists a large effort in the field of *integrity constraints maintenance* [CW90, ML91, Wüt93]. This problem can be summarized in the following terms. Given a consistent database state and a transaction that violates some integrity constraints, the problem is to find *repairs*, that is, an additional set of insertions and/or deletions of base facts to be appended to the transaction such that the resulting 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 original transaction must be rejected.

Given a transaction T, integrity constraints maintenance can be specified in our framework as the downward interpretation of {T,¬ιIc}, provided that Ic^0 does not hold. Thus, sets of base fact updates defined by the downward interpretation of {T,¬ιIc} correspond to possible transactions that maintain database consistency.

In a similar way, the downward interpretation of a negative deletion event fact ¬δIc allow us to specify the problem of *maintaining database inconsistency*. Given an inconsistent database state and a transaction T, the problem is to obtain an additional set of base fact updates to be appended to the original transaction in order to guarantee that the resulting database state remains inconsistent. In our framework, this problem can be specified as the downward interpretation of {T,¬δIc}, provided that Ic^0 holds. Although we do not see any practical application of this problem, it can be naturally classified and specified in the framework proposed in this paper.

5.2.5 Enforcing Condition Activation

As stated before, general conditions can also be expressed as derived predicates. *Enforcing condition activation* refers to the problem of obtaining a set of changes of base facts that, if they were applied to the current sate of the database they would induce an activation of a given condition.

In our framework, enforcing condition activation is specified as the downward interpretation of ιCond(**X**) or δCond(**X**), where both correspond to the conditions to be enforced. The former defines possible transactions that will induce **X** to satisfy the condition after their application, but not before. The latter, defines possible transactions that will induce **X** not to satisfy the condition after their application.

The downward interpretation of ιCond(**X**) or δCond(**X**) can also be used for specifying the problem of *condition validation*. Given a deductive database, the problem is to obtain at least one **X** for which some set of base fact updates that satisfies ιCond(**X**) or δCond(**X**) exists. This can be useful for providing the database designer with a tool for validating certain aspects of the condition definition. For instance, s/he could validate whether it is possible to find a transaction that will induce the activation of a given condition.

5.2.6 Preventing Condition Activation

The downward interpretation of the event rules corresponding to a condition predicate allow us to identify another database update problem. Given a transaction that consists of a set of base fact updates, the problem is to find an additional set of insertions and/or deletions of base facts to be appended to the original transaction such that it guarantees that no changes in the condition will occur during the transition. We denote this problem as *preventing condition activation*.In general, several solutions may exist and the user must select one of them.

Assume that we want to prevent that the application of a given transaction T will induce a change on the condition Cond(**X**). In our framework, this problem can be specified as the downward interpretation of {T,¬ιCond(**X**)} or {T,¬δCond(**X**)}, respectively. The former defines base fact updates needed to guarantee that the insertion of Cond(**X**) is not induced by T, while the latter defines changes on base predicates needed to satisfy that the deletion of Cond(**X**) is not induced. In general, if we want to prevent all possible activations of Cond, we have to take into account all possible values of **X**.

5.3 Combining Upward and Downward Problems

In order to finish this section, we would like to comment several aspects concerning the upward and downward interpretations of the event rules corresponding to integrity constraints, views and conditions.

First of all, we would like to notice that upward problems can be combined among them. All of them share a common starting-point (a transaction which consists of a set of base fact updates) and aim at the same goal (to define the changes on derived predicates induced by this transaction). The same reasons allow the combination of downward problems among them. Therefore, we can specify more complex upward and downward problems by considering possible combinations of the problems specified in sections 5.1 and 5.2.

For instance, we could combine materialized view maintenance, integrity constraints checking and condition monitoring by upward interpreting the set {ιView(**x**), δView(**x**), ιCond(**x**), δCond(**x**), ιIc}. We could also combine view updating with integrity constraints maintenance by downward interpreting the set {ιView(**X**), ¬ιIc}.

Moreover, we could also combine upward and downward problems among them. Note that the result of the downward interpretation is the same than the starting-point of the upward interpretation, that is, a transaction which consists of a set of base fact updates. Therefore, we could first deal with downward problems and, immediately after, use the obtained result for dealing with the upward ones.

For instance, we could be interested on distinguishing between integrity constraints to be maintained and integrity constraints to be checked, and on combining view updating with the treatment of both kinds of constraints. In this case, we should first downward interpret the view update and the set of integrity constraints to be maintained. Then, we should consider the resulting transactions and upward interpret the set of integrity constraints to be checked to reject those resulting transactions that violate some constraint in this set.

Finally, we would also like to notice that in our approach the specification of the upward and the downward problems is the same when considering other kinds of updates like insertions or deletions of deductive rules. In this case, we should first determine the changes on the transition and event rules caused by the update and apply then our approach in the same way as explained in sections 5.1 and 5.2.

6. Conclusions and Further Work

In this paper, we have proposed to use the event rules as the basis of a common framework for classyfing and specifying deductive database updating problems. In this sense, we have proposed two interpretations of these rules which have allowed us to identify and to specify *upward problems*, concerned with computing the changes on derived predicates induced by a transaction, and *downward problems*, concerned with determining the possible transactions that satisfy a set of changes on derived predicates.

Moreover, we have shown that upward and downward problems can be combined among them, thus being able to define more complex deductive database updating problems.

By considering only a unique set of rules for specifying all these problems, we hope to have shown that it is possible to provide general methods able to deal with all these problems as a whole.

Therefore, we could uniformly integrate view updating, materialized view maintenance, integrity constraints checking, integrity constraints maintenance, repairing inconsistent databases, condition monitoring, enforcing condition activation and other deductive database updating problems into an update processing system.

We plan to extend the results presented in this paper in two different directions. First, we plan to extend our previous work reported in [UO92, UO94, TO92] by taking into account the ideas proposed in this paper in order to provide an efficient implementation of the upward and the downward interpretations. This could be useful for dealing with all these deductive database updating problems in a real environment.

Second, we consider that the upward and the downward interpretations could also be used not only for specifying deductive database udpating problems like we have proposed in this paper, but also for specifying the methods proposed up to date for dealing with these problems. We believe that this would help to provide a common framework for comparing all these methods.

Acknowledgements

We would like to thank Dolors Costal, Carme Martín, Enric Mayol, Antoni Olivé, Joan Antoni Pastor, Carme Quer, Maria Ribera Sancho and Jaume Sistac for many useful comments and discussions.

This work has been partially supported by the CICYT PRONTIC program project TIC94-0512.

References

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

[BDM88] Bry, F.; Decker, H.; Manthey, R.. "A Uniform Approach to Constraints Satisfaction and Constraints Satisfiability in Deductive Databases", Int. Conf. on Extending Database Technology (EDBT'88), Venezia, 1988, pp. 488-505.

[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.

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

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

[CW91] Ceri, S.; Widom, J. "Deriving production rules for incremental view maintenance", Proc. of the 17th. VLDB Conf., Barcelona, 1991, pp 577-589.

[DW89] Das,S.;Williams,H. "A path finding method for constraint checking in deductive databases", Data & Knowledge Engineering, 1989, No 4, pp. 223-224.

[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.

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

[GCM+94] García, C; Celma, M.; Mota, L.; Decker, H. "Comparing and Synthesizing Integrity Checking Methods for Deductive Databases", Proc. of the 10th ICDE, Houston, USA, 1994, pp. 214-222.

[GMN84] Gallaire, H.; Minker,J.; Nicolas, J.M. "Logic and Databases: A Deductive Approach". ACM Computing Surveys, Vol. 16, N° 2, 1984, pp. 153-185.

[GMS93] Gupta, A; Mumick, I.S.; Subrahmanian, V.S. "Maintaining Views Incrementally", Int. Conf. on Management of Data (SIGMOD), Washington, 1993, pp. 157-166.

[HCK+90] Hanson, E. N.; Chaabouni, M.; Kim, C.; Wang, Y. "A predicate matching algorithm for database rule systems", Proc. ACM SIGMOD Conf. on Management of data, Atlantic City, 1990, pp. 271-280.

[HD92] Harrison, J. V.; Dietrich, S. " Maintenance of materialized views in a deductive database: an update propagation approach", in workshop on deductive databases, JICSLP, Washington, 1992, pp. 56-65.

[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.

[Kow83] Kowalski, R. "Logic Programming". Information Processing 83, Elsevier Science Publishers, 1983.

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

[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.

[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.

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

[QW91] Qian, X.; Wiederhold, G. "Incremental recomputation of active relational expressions", IEEE Trans. on knowledge and data engineering, Vol. 3, No. 3, setember 1991, pp. 337-341.

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

[SK88] Sadri, F.; Kowalski, R. "A Theorem-proving Approach to Database Integrity", In J.Minker (Ed.) "Foundations of Deductive Databases and Logic Programming", Morgan Kaufmann Pub., 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.

[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).

[UO94] Urpí, T.; Olivé, A. "Semantic Change Computation Optimization in Active Databases", Proc. of the 4th Int. Workshop on Research Issues on Data Engineering-Active Database Systems (RIDE -ADS'94), Houston, USA, 1994, pp. 19-27.

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

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