Cumulative Updates

S. M. Sripada
European Computer-Industry Research Centre
Arabellastrasse 17
81925 Munich, Germany
sripada@ecrc.de

B. Wüthrich
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
beat@cs.ust.hk

Abstract

When an update to a view is requested by a user, there may be no unique way of updating the stored relations in the database to realize the requested update. Chosing one of the alternatives for updating stored relations may not reflect the change that has actually taken place in the real world; in the presence of other derived views, the database may actually present a very wrong model of the world to the user. The problem is even more severe in the case of deductive databases. For avoiding this problem, we introduce a new notion of view updates, called *cumulative updates*. The key idea behind cumulative updates is that update mechanisms should wait for further update requests to resolve ambiguities. Equivalently, current update requests must also take into account previous requests made to the knowledge base. Cumulative updates, therefore, subsume conventional updates in which only the current update request is considered. In this paper, we motivate the need for cumulative updates and formally define the notion of such updates as well as the different classes therein. We then give methods for computing one particular class of cumulative updates.

**Proceedings of the 20th VLDB Conference
Santiago, Chile, 1994**

1 Introduction

When a view update is requested by a user, there may be no unique way of updating the stored relations in the database to realize the requested update. However, chosing one of the alternatives for updating base relations may not reflect the change that has actually taken place in the real world. For example, consider two stored relations *empdept* and *deptMgr* that record the department in which each employee works and the manager for each department respectively. Assume that a view *empMgr* has been defined on this relational schema which is obtained by joining the two base relations:

$$empMgr(X,Y) \leftarrow empdept(X,Z), deptMgr(Z,Y)$$

Let us assume that in the current database, "Tim is the manager of the sales department" and "Tom works in the sales department". Therefore, the view *empMgr* contains the tuple "manager of Tom is Tim". Now consider an update request on the view which states that "Smith is the manager of Tom". There are at least two ways in which this update can be realized: (1) Tom has now moved into a new department which is managed by Smith, or (2) Smith has replaced Tim as the manager of the sales department. Depending upon which alternative is chosen, updates to the base relations empdept and deptMgr will be made. However, each choice may bring with it additional consequences. For example, it may be that Tim has been fired and therefore should not receive his next salary cheque. The database has no way of knowing which alternative would be best. But at least it should make sure that it does not misrepresent the state of affairs in the real world. On the other hand, the user may not have enough information to resolve this conflict either. Indeed, he may come to know of what had actually happened at a later date, and this information will then come in as a new update request. Therefore, it would be nice to have an update mechanism that would allow the database to record the ambiguous state of affairs concerning its model of the world and let the user be warned. It would also be nice if the database would wait for further update requests to resolve the ambiguity. The notion of *cumulative updates* that we propose in this paper is precisely aimed at achieving this objective.

The problem is even more serious in the case of deductive databases where derived relations are recursively defined. Sometimes, the number of alternatives for an update may be infinite. We shall now consider another example to illustrate the problem of *side effects* caused by other derived views.

Consider the following propositional knowledge base KB:

$$\begin{array}{lcl} p & \leftarrow & m1, m2 \\ p & \leftarrow & n1, m2 \\ q & \leftarrow & a, m2 \\ r & \leftarrow & b, m1 \\ s & \leftarrow & b, n1 \\ & a & \\ & b & \end{array}$$

Now consider the update request to add p. This update may be realized in two ways:

(i) add $m1$ and add $m2$

(ii) add $n1$ and add $m2$.

However, both alternatives have side-effects. The first alternative causes r to be true in addition to p. The second alternative causes s to be true in addition to p. In addition, both alternatives cause q to be true.

It is not possible to come up with a "belief revision" semantics that would allow us to identify, in all situations, which of the two alternatives in the above example is better [AGM85, Dal88, EG92, FKUV86, Gar88, Gra91, KM91, Mar91, MS86, Neb91, RdK87, Win90]. So, normally we are forced to make a choice, and almost all the update mechanisms proposed in the literature make a choice or leave it to the user to make a choice, where s/he may have no way of knowing *apriori* which is the correct choice [BKSW91, Dec90, GL90, KM90, ML91, RB92, SI91, Tom88, TA91, Wüt93].

There are severe problems with making a choice. Suppose we choose alternative (i) in our example. Then, we also believe that r is true. However, suppose that after some time, more information concerning the world is known. Say, an update request to add s is received. It can be seen that, if choice (ii) were made in first place, then the second update request would have been automatically satisfied. If $n1$ is now added to KB (so as to realize "add s"), then the KB updates do not satisfy the minimal change criterion (which seems to be the most natural one). In order to satisfy this criterion, $m1$ should be deleted and $n1$ should be added to the KB. It is not at all straightforward to discover that that the above two operations, when performed, would result in a minimal change.

In addition to the criterion of minimal change, the second problem with forcing an option is that r may be believed to be true when in reality it is not the case. In other words, *the update we request is not the update we get.*

Of course, one may argue that instead of converting an update on a view (a derived relation) to that of an update on a base relation, one may just "update" the derived relation. That is: "p" is added to the database. Such an approach is proposed in [LLS93]. However, there are serious problems with this approach too, the main ones being the following: (1) It may not be meaningful or indeed impossible to "store" some views. For example, let us assume that the date's of birth of all employees are recorded in a base relation *employeeDob* and that a view of the ages of all employees *employeeAge* is defined over employeeDob, where age is obtained by subtracting the date of birth from the current date. When a request to update the view employeeAge is received, it is not meaningful to store the age of the employee explicitly since it changes every day. (2) If p is stored directly in the KB of our example, then it would not be possible to know that q is also true in the world. (3) Undisciplined addition and deletion of base and derived relations to a database results in inconsistencies and anomalies. For example, when an employee (eg. Smith) is fired, he may still - according to the database - be the manager of some employees (eg. Tom) since the view empMgr is now explicitly stored!

For the above reasons, we believe that a better mechanism for realising updates in a database is required. The purpose of this work is to propose a new concept of updates, called *cumulative updates*, which overcomes the above problems. In particular, one would like to have an update mechanism such that after the addition of p in our example

1. p is true in (the new) KB
2. $m1$ and $n1$ are possibly true in KB since there is no way to say which is true for sure
3. r and s are possibly true in KB (follows from 2)
4. $m2$ is true in KB (since $m2$ will have to be added no matter which choice is made)
5. q is true in KB (follows from 4)

Similarly, after a further addition of s,

1. p is still true in (the new) KB
2. $n1$ is true in the new KB but $m1$ is not
3. s is true in the new KB but r is not (follows from 2)
4. $m2$ is still true in KB
5. q is still true in KB

In other words, cumulative updates behave as though always the right choice is made. This means, however, that updates have to take into account previous update requests as well as the current one.

Cumulative updates can vary widely in complexity depending upon whether updates involve only the addition and deletion of facts or they involve the addition and deletion of rules or they involve the induction of new rules.

Several methods for updating knowledge bases have been proposed in the literature [BKSW91, Dec90, RB92, GL90, KM90, Tom88, TA91, Wüt93]. Although some of the methods consider composite updates (i.e. several atomic updates at a time) e.g. [Wüt93], they do not consider situations where the knowledge base is queried in between the atomic updates. These approaches require that the user or the system make a choice to realize an update. As we have outlined above, there are severe problems with making choices. Kakas and Mancarella [KM90] express some of the sentiments found in this paper regarding choices. They are however concerned with unsatisfiability of update requests and with backtracking over previous choices in case an update request proves unsatisfiable. They suggest the use of a truth maintenance system for backtracking over previous update choices. We are not aware of any work in the literature which deals with the problem of cumulative updates.

The main aim of this paper is to define the notion of cumulative updates in the more general setting of knowledge bases, identify the different classes of cumulative updates, and to propose a method for a specific class of cumulative updates for knowledge bases. The method we propose is sound but not always complete. The method is straightforward to implement for the relational case.

The rest of the paper is organized as follows. We describe different classes of cumulative updates in section 2. In section 3, we define the notions of correctness and completeness for techniques dealing with cumulative updates. We provide an exhaustive example in section 4 and motivate a technique to realize cumulative updates. The technique consists of two parts described in sections 5 and 6 respectively. The first part concerns the creation of some new rules from the original rules in the KB (section 5). This is basically a rewrite technique such as Magic Sets. The second part, described in section 6, concerns the maintenance of a table of ambiguous facts. In section 7, we outline how the proposed technique may be implemented for the relational case to realize cumulative updates. Conclusions and future work are outlined in section 8.

2 Cumulative Updates

Different classes of cumulative update problems are conceivable depending upon the syntax and the semantics of a knowledge base, the interpretation of what a sequence of updates means, the allowed modifications of the knowledge base, and so on. We list six criteria according to which the problem of dealing with cumulative updates can be classified. Each criterion is based on a particular assumption. Each assumption results in a different instance of the problem. Therefore, there is a whole class of problems of cumulative updates.

We assume to have a knowledge base $KB = (F, R, C)$ consisting of a set of ground atoms or facts F, a set of rules R and a possibly empty set of integrity constraints C. Furthermore, we assume a sequence of transactions $T_1, T_2, T_3, \ldots$ coming in that order. Each transaction T_i consists of a sequence of updates $u_{i,1}, \ldots, u_{i,n_i}$. Hence a transaction sequence $T_1, \ldots, T_k$ is also an update sequence $u_{1,1}, \ldots, u_{1,n_1}, \ldots, u_{k,n_k}$. Six criteria have a significant influence on the problem of cumulative updates. Below, we list these criteria with possible values for each criterion.

1. Syntactic restrictions on KB
   (a) Propositional rules and constraints
   (b) Datalog rules without negation [Ull88]
   (c) Datalog rules with stratified negation [Ull88]
   (d) Rules with function symbols [Llo87]
   (e) Datalog rules without recursion (equivalently, views defined by relational algebra expressions) [Ull82]
2. Semantics
   (a) Well founded model semantics [GRS91]
   (b) Negation as failure [Llo87]
   (c) Least model semantics [vEK76]
   (d) Minimal model [ABW88]
   (e) Clark completion [Cla78]
   (f) Classical logical consequence [Men87]
3. Information which may be changed by an update
   (a) The facts F

(b) The facts F and the rules R

4. How to decide between different potential solutions

   (a) Choose the appropriate solution among the set of all potential solutions

   (b) Dealing with semi-modalities (i.e. a fact can also be "possibly true" or "possibly false" respectively) and definitely confirming a modification only when it is unique (i.e. the modification is required by all potential solutions)

5. Semantics of the sequence of updates

   (a) Update $u_{i,j}$ is independent from update $u_{l,h}$ if $i \neq l$ or $j \neq h$

   (b) Transactions are independent of each other; the intention of a single transaction T_i is to make $\wedge_j u_{i,j}$ true

   (c) All updates throughout all transactions cooperate, the user would like to make $\wedge_{i,j} u_{i,j}$ true, but knows the information at different points in time (hence has to put the updates into different transactions)

6. What's a solution

   (a) A solution is a fact (and/or rule) modification which makes the update true and makes minimal changes on the set F (and the set R)

   (b) A solution is a fact (and/or rule) modification which makes the update true and minimally changes the information deducible

The following example illustrates criterion 5 under the assumptions 1(a), 2(d), 3(a), 4(b) and 6(a).

We consider the propositional knowledge base consisting of the constraint

$$\leftarrow q, c$$

(i.e. q and c cannot hold simultaneously) and the rules

$$\begin{array}{lcl} p & \leftarrow & a, b \\ p & \leftarrow & c, b \\ q & \leftarrow & a \\ q & \leftarrow & d \\ r & \leftarrow & d, c \end{array}$$

We consider two sequential transactions consisting of one update each:

1. add p
2. add q

Let us first assume 5(b). From the request to add p we can conclude that b holds since b is true in all alternatives that make p true (one alternative is to add $\{a, b\}$ and the other alternative is to add $\{c, b\}$). Hence we confirm this information by adding the fact b to the knowledge base. Whilst p should be true after the transaction, a and c should be possibly true. A subsequent request to add q results in no further confirmations (one alternative is to add a and the other alternative is to add d). Whilst q should be true after the transaction, a and d should be possibly true. Combining the individual consequences of the two updates together will make r possibly true (since c is possibly true by the first update and d is possibly true by the second update). So after both transactions p, q and b should be true, and a, c, d and r should be possibly true.

On the other hand, if we assume 5(c) then the conclusions are different. Having received the update to make p true, we know that b is true. Hence it is added to the knowledge base. We also know that a and c are possibly true. Then we receive the transaction add q. So we have to look for solutions making $p \wedge q$ true. The only minimal solution is to add a (since adding $\{d, c\}$ is forbidden by the constraint). Hence we confirm a by adding it to the knowledge base. This now leaves r definitely false and makes p, q, a and b true.

In the sequel, we will assume 5(c) as the semantics of a sequence of updates. Let $u_{1'}, \ldots, u_{l'}$ be those requests of $u_{1,1}, \ldots, u_{k,n_k}$ not yet uniquely realized (i.e. not made true by confirmed information alone). Then we try to deduce the information necessarily needed to make the not yet uniquely realized updates $u_{1'}, \ldots, u_{l'}$ *simultaneously* true. The modifications necessarily needed to make $u_{1'} \wedge \ldots \wedge u_{l'}$ true are then confirmed and the base relations are modified accordingly. In course of time, more and more of the remaining updates $u_{1'}, \ldots, u_{l'}$ will be realized in the wake of new information arriving in form of update requests on extensional or intensional data.

In the rest of the paper we discuss the problem of cumulative updates under the assumptions 1(b), 2(d), 3(a), 4(b), 5(c) and 6(a).

The novelty of our study stems from the following facts:

- We define the semantics of a sequence of update transactions (no other study so far considers the case of a sequence of update transactions).
- We propose not to force a choice in case of update ambiguity (choice 4(a)) but instead to deal with the problem through semi-modalities (choice 4(b)).

Introducing semi-modalities creates another assumption dimension, namely, whether a transaction is allowed to make updates on possibly true and possibly false information. Updates on "possible" information may be interpreted in two ways.

- The first interpretation arises when the user is not aware of the fact that the knowledge base is assimilating updates in a cumulative fashion. It is then reasonable to assume that an update request to delete a "fact" arises in the world only if that "fact" was previously true in the world. Then the update request to delete "fact" when *pos(fact)* holds (i.e. when *fact* is possibly true) in the knowledge base implies that "fact" should have been true in the knowledge base prior to the current update request. This kind of reasoning can be achieved only when both the history of events occurring in the real-world as well as a history of the database updates are both recorded in the database [Sri88, Rei92]. Recording such histories enables one to reason about the two notions of time, namely valid time and transaction time, that are required in advanced knowledge base applications [Sri93].
- The second interpretation is plausible when the user is aware of the fact that the knowledge base is assimilating updates in a cumulative fashion. It is then reasonable to assume that the update request to delete "fact" when *pos(fact)* holds in knowledge base implies that the user would actually like to confirm that "fact" is not true in the real-world. In other words, the user is trying to reduce the ambiguity or uncertainty of the information contained in the knowledge base by explicitly ruling out some possibilities.

Both interpretations are equally plausible. However, the first one requires a more complicated reasoning and an explicit treatment of time. For the sake of simplicity, we adopt the second interpretation in this paper.

The assumptions 3(a) and 6(a) we made are natural and more or less standard in the literature. The update methods proposed so far [BKSW91, Dec90, GL90, KM90, LLS93, RB92, Tom88, TA91, Wüt93] are mostly concerned with criteria 1 and 2; they only differ in the methods proposed for solving the update problem.

We now give the formal definitions needed for a precise understanding of the notions presented in this paper.

3 Definitions

Let $\mathcal{L}$ be a first order language built from a finite alphabet $\mathcal{A}$ containing three disjoint classes of predicate symbols: intensional predicates IP, extensional predicates EP and possible predicates PP. $\mathcal{A}$ satisfies that for each n-ary predicate symbol p in EP or IP, there are the two n-ary predicate symbols ***possibly_p*** and ***possibly_not_p*** in PP. The predicates ***possibly_p*** and ***possibly_not_p*** will be written as $pos(p)$ and $pos(\neg p)$ respectively. $\mathcal{A}$ contains also a finite set of skolem constants $Sk = \{c1, c_2, ...\}$.

A *knowledge base*, denoted KB and sometimes subscripted, is a triple (F, R, C) of sets: F is a set of facts or ground atoms, R is a set of rules and C is a set of constraints. A *rule* is an implicitly universally closed formula of the form $H \leftarrow L_1, ..., L_n$ where H is an atom built from a predicate in IP and L_i $(1 \leq i \leq n)$ is a literal. A *constraint* is a universally closed, range-restricted formula. In this paper we assume Horn rules, i.e. each literal in any rule body is positive. This is assumption 1(b).

An *update u*, sometimes called update request, is a conjunction of ground literals. Sometimes we also say that fact A is to be deleted or should be made false while referring to the update $\neg A$.

As mentioned earlier in this paper, we will restrict ourselves to the case where only modifications on facts are allowed. This reflects assumption 3(a). By m we denote a *modification* which is a pair (S^+, S^-) of sets of facts built from predicates in EP. If KB is a knowledge base then $m(KB) = (KB \cup S^+) - S^-$. A modification can also be viewed as a formula. Each skolem constant in m is replaced by a new variable symbol, the atoms in S^- are existentially closed, the conjunction of all atoms in S^+ plus the negation of the atoms in S^- are conjunctively connected, and finally, the resulting formula is existentially closed and the quantifiers moved in. For example, the modification $m = (\{p(1), q(c1, c_2), r(c1, 2)\}, \{t(c_3, 4)\})$ represents the formula $p(1) \wedge (\exists X (\exists Y q(X, Y) \wedge r(X, 2))) \wedge \neg \exists Z t(Z, 4)$.

We introduce a two-valued function α mapping a knowledge base and a closed formula to the set $\{t, f\}$. That is, for an update u we have $\alpha(KB, u) = t$ iff $\xi \models u$ and $\alpha(KB, u) = f$ iff $\xi \not\models u$, where ξ denotes the chosen semantics under criterion 2. In this paper, we assume that $\xi = M_{KB}$ where M_{KB} denotes the minimal model of the rules and the facts in KB. This reflects our assumption 2(d). Therefore, in this paper, α is the minimal model semantics of a database consisting of Horn rules and facts.

Let m be a modification, u be an update and $KB = (F, R, C)$ be a knowledge base. We say that m is a *solution* (wrt u and KB) iff

- $\alpha(m(KB), u) = t$,
- $\alpha(m(KB), \wedge_{c \in C} c) = t$, and

- no proper subformula of m is logically equivalent to m.

A solution is therefore a fact modification which makes the update true, leaves the constraints satisfied and is minimized with respect to facts involving skolem constants. Notice that the definition of solution is independent of any specific assumptions that one may make i.e. it applies to any combination of assumptions.

We assume a belief revision function β (6(a) and 6(b) are belief revision functions). The set of all possible solutions wrt u and KB under the belief revision function β is denoted by $\beta(KB, u)$. In our case, we assume β to be 6(b), so that $\beta(KB, u)$ is the set containing exactly the solutions which are minimal according to the partial order $\sqsubseteq$ defined as follows. Let m_1 and m_2 be two modifications viewed as formulae. Then, m_2 is *as big as* m_1, denoted $m_1 \sqsubseteq m_2$, iff m_2 logically implies m_1. The two modifications are *equal*, denoted $m_1 \simeq m_2$, iff $m_1 \sqsubseteq m_2$ and $m_2 \sqsubseteq m_1$. The *intersection* m of m_1 and m_2, denoted $m_1 \sqcap m_2 (= m)$, is the biggest modification such that $m \sqsubseteq m_1$ and $m \sqsubseteq m_2$.

The natural criterion for making a fact modification due to an update request *definite* is that whenever a submodification m is required by all (or equivalently all minimal) modifications realizing an update then and only then m has enough support to be made definite. We call this strategy *confirmation by uniqueness*. This strategy reflects assumption 4(b). We say that a modification m is *confirmed* (made definite) by update u and knowledge base KB iff $m = \sqcap_{m' \in \beta(KB,u)} m'$. Note that m is syntactically determined only up to equality ($\simeq$) which means that its representation is determined up to renaming of skolem constants.

We now define correctness and completeness of techniques dealing with cumulative updates under the assumption 5(c). Let KB_0 be a given, initial knowledge base and $u_{1,1}, ..., u_{k,n_k}$ be a sequence of updates received from a sequence of transactions $T_1 = u_{1,1}, ..., u_{1,n_1}$, $T_2 = u_{2,1}, ..., u_{2,n_2}$, ..., $T_k = u_{k,1}, ..., u_{k,n_k}$. Then the knowledge base sequence $(KB_0, KB_1, ..., KB_k)$ is said to be *correct* iff for all l $(1 \leq l \leq k)$ and for each ground atom A the following four conditions hold:

1. (a) We set KB'_0 to KB_0 and define KB'_l to be $m(KB'_{l-1})$ where m is the modification confirmed by $u_{1'} \wedge ... \wedge u_{l'}$ and KB'_{l-1}.
   (b) Let $u_{1'}, ..., u_{l'}$ be all updates in $u_{1,1}, ..., u_{l,n_l}$ such that $\alpha(KB'_{l-1}, u_{i'}) = f$ holds [1].
2. $\alpha(KB_l, A) = t \qquad \Longrightarrow$
   (a) $\alpha(KB'_l, A) = t$; or
   (b) $\alpha(KB'_{l-1}, A) = t$ and there are m_1 and m_2 in $\beta(KB'_{l-1}, u_{1'} \wedge ... \wedge u_{l'})$ such that $\alpha(m_1(KB'_{l-1}), A) = t$ and $\alpha(m_2(KB'_{l-1}), A) = f$.
3. $\alpha(KB_l, pos(A)) = t \qquad \Longrightarrow$
   (a) there is an $m \in \beta(KB'_{l-1}, u_{1'} \wedge ... \wedge u_{l'})$ such that $\alpha(m(KB'_{l-1}), A) = t$, and
   (b) $\alpha(KB_l, A) = f$.
4. $\alpha(KB_l, pos(\neg A)) = t \qquad \Longrightarrow$
   (a) there is an $m \in \beta(KB'_{l-1}, u_{1'} \wedge ... \wedge u_{l'})$ such that $\alpha(m(KB'_{l-1}), A) = f$, and
   (b) $\alpha(KB_l, A) = t$.

[1]Note that 1(a) and 1(b) recursively define a confirmed knowledge base sequence KB'_0 to KB'_k.

The knowledge base sequence is said to be *complete* iff condition 1 holds and the if-counterparts ($\Longleftarrow$) of conditions 2, 3 and 4 (i.e. the when the implications in 2, 3 and 4 above are reversed) hold.

These definitions of correctness and completeness also describe what happens in the case of an unrealizable update: it leads to a situation where no changes can be made, regardless of what further update requests arrive. We now give a comprehensive example.

4 Example

Consider the database of a company. The basic information is given through two base relations: $h(X,Y)$ (Y is head of department X) and $w(X,Y)$ (Y is a worker of department X). From these base relations some information is deduced intensionally through rules: $d(D,X)$ (X is an employee in department D), $sd(X,Y)$ (X and Y work in the same department), $m(X)$ (X is a manager), and $p(X,Y)$ (X and Y are peers). Furthermore, three integrity constraints (denoted ICs) assure that nobody is in more than one department, that no department has two heads, and that nobody is a worker and a department head at the same time.

$$\begin{array}{rlr} IC: & d(D1,X), d(D2,X) \rightarrow D1 = D2 & (1) \\ & h(D,X), h(D,Y) \rightarrow X = Y & (2) \\ & \neg(\exists D1 w(D1,X), \exists D2 h(D2,X)) & (3) \end{array}$$

$$\begin{array}{rcll} d(D,X) & \leftarrow & w(D,X) & (4) \\ d(D,X) & \leftarrow & h(D,X) & (5) \\ sd(X,Y) & \leftarrow & d(D,X), d(D,Y) & (6) \\ m(X) & \leftarrow & h(D,X) & (7) \\ p(X,Y) & \leftarrow & m(X), m(Y) & (8) \\ p(X,Y) & \leftarrow & w(D,X), w(D,Y) & (9) \end{array}$$

The information acquisition and inference scenario that we would like to achieve, given the above rules and constraints, is illustrated in the following. Since an update (a piece of information) can not always be explained uniquely we would sometimes like to deduce facts such as $pos(\neg fact)$ denoting that $fact$ is possibly false, or $pos(fact)$ denoting that $fact$ is possibly true. Therefore, we would like to reason with semi-modalities in a two valued logic. The logic is still two valued since a fact such as $pos(fact)$ is either deducible, hence true, or not deducible, hence false. We will discuss a knowledge assimilation scenario consisting of three sequential updates stemming from three different transactions:

1. add $sd(john, simon)$
2. add $w(1, simon)$
3. add $h(1, john)$

Considering the above transactions, first of all we get the information that *john* and *simon* belong to the same department. Hence we would like to make *sd(john,simon)* deducible. In order to realize this we would like to simply physically insert the fact *add(sd(john,simon))* into the system. This should enable the system to deduce *sd(john,simon)*, *sd(simon,john)*, *pos(w(c1,john))*, *pos(w(c1,simon))*, *pos(h(c1,john))*, *pos(h(c1,simon))*, *pos(m(john))*, *pos(m(simon))*, *pos(p(john,simon))*, *pos(p(simon,john))*, *pos(d(c1,john))*, and *pos(d(c1,simon))* [2]. Note that *c1* is a skolem constant (simply a constant not used anywhere else) replacing an existentially quantified variable (hence a skolem constant can also be seen as a NULL value [Imi86, IL84]).

Secondly, we get the information that *simon* works in department 1, thus we insert *add(w(1,simon))* into the database. This should make *sd(john,simon)*, *sd(simon,john)*, *pos(h(1,john))*, *pos(w(1,john))*, *pos(m(john))*, *pos(p(john,simon))*, *w(1,simon)*, *pos(p(simon,john))*, *d(1,simon)*, and *pos(d(1,john))* deducible[3].

Thirdly, we account for the new and complementing information that *john* is head of department 1 by inserting the fact $add(h(1, john))$. We would then like to have the facts $sd(john, simon)$, $sd(simon, john)$, $m(john)$, $w(1, simon)$, $h(1, john)$, $d(1, john)$, and $d(1, simon)$ deducible.

[2]There is a weak form of incompleteness in our techniques proposed in sections 5 and 6 which is illustrated through this example: it would be strongly complete to deduce *d(c1,john)* and *d(c1,simon)* (there is a department to which *john* and *simon* belong) instead of *pos(d(c1,john))*, and *pos(d(c1,simon))* respectively.

[3]The same weak incompleteness occurs here with *pos(d(1,john))*.

Our technique, described in sections 5 and 6, which achieves the reasoning illustrated above consists of two components.

The first component generates new rules and integrity constraints from the original set of rules and constraints using rewriting techniques. Standard query evaluation mechanisms are then applied to the new rules to answer queries. Under our assumptions, the new rules will be evaluated under the minimal model semantics for stratified Datalog programs [ABW88]. The second component maintains the set of base facts, which now include semi-modalities such as $pos(m(john))$ and meta-facts such as $add(m(john))$. This component is invoked immediately after each update transaction to prepare the database for the correct evaluation of subsequent queries.

In section 5 we describe how to generate the new rules from the original set of rules. In section 6 we describe the method for maintaining the base facts.

5 Rule Rewriting

In this section, we describe the logic behind the generation of new rules and show how new rules and integrity constraints are generated from the original set of rules.

If m is a predicate symbol and X is a vector of terms, three kinds of facts $m(X)$, $pos(m(X))$ and $pos(\neg m(X))$, will be visible (and accessible) to the DB user. In addition to the above, some auxiliary facts (not visible to the user) are needed corresponding to the intensional predicates: $m^*(X)$, $pos^*(\neg m(X))$, $pos^*(m(X))$, $nt(m(X))$, $nf(m(X))$, $add(m(X))$ and $del(m(X))$ (if m is an extensional predicate then only $pos^*(m(X))$ and $pos^*(\neg m(X))$ are needed). The fact $m^*(X)$ is deducible exactly when $m(X)$ follows from the confirmed (and hence unique) information alone. The fact $pos^*(m(X))$ (resp. $pos^*(\neg m(X))$) shows that $m(X)$ possibly holds irrespective of whether it is not yet definitely deducible and irrespective of whether it is necessarily false (a weak inference). Hence the (user-visible) fact $pos(m(X))$ can be defined by $pos(m(X)) \leftarrow pos^*(m(X)), \neg m(X), \neg nf(m(X))$. The fact $pos(\neg m(X))$ is analogously defined. The fact $nt(m(X))$ (resp. $nf(m(X))$) expresses that from the update request it follows that $m(X)$ necessarily holds (resp. necessarily does not hold).

An update request of the form add $m(john)$ (resp. delete $m(john)$) will be transformed into a physical insertion of the tuple $add(m(john))$ into the database (resp. a physical insertion of the tuple $del(m(john))$) (see also step 1 of the algorithm given in section 6). However, such tuples are no longer needed when for instance $m^*(john)$ becomes deducible and thus $m(john)$ is derivable without the explicit fact $add(sd(john, simon))$. So two constraints,

$\leftarrow m^*(X), add(m(X))$ and $\leftarrow \neg m^*(X), del(m(X))$, are required to ensure that facts such as $add(m(X))$ and $del(m(X))$ are deleted when they are no longer necessary. Hence a truth maintenance [Mar91] is achieved with these constraints (section 6 explains when these integrity constraints are checked). A fact such as $m(john)$ may become deducible without an explicit fact $add(m(john))$ from the more complete information that has been assimilated in the meantime through subsequent transactions. There are two ways of assimilating more information on base relations:

- Obtain new data by means of explicit base relation updates given by the user.
- Uniquely deduce (modulo skolem constant renaming) through an update request on an intensional predicate that certain base facts must be added or deleted.

Considering the example again, we now give the rules and integrity constraints generated from the rules (4) to (9). The newly generated rules and constraints as well as the original rules and constraints are stored in the database since they are needed by the fact maintenance component (see section 6). The query evaluation itself uses only the new rules ((16 to (45)).

$$\begin{array}{rcll}
ICs: & \leftarrow & m^*(X), add(m(X)) & (10)\\
& \leftarrow & sd^*(X,Y), add(sd(X,Y)) & (11)\\
& \leftarrow & p^*(X,Y), add(p(X,Y)) & (12)\\
& \leftarrow & \neg m^*(X), del(m(X)) & (13)\\
& \leftarrow & \neg sd^*(X,Y), del(sd(X,Y)) & (14)\\
& \leftarrow & \neg p^*(X,Y), del(p(X,Y)) & (15)
\end{array}$$

$$\begin{array}{rcll}
d(D,X) & \leftarrow & d^*(D,X), \neg nf(d(D,X)) & (16)\\
d(D,X) & \leftarrow & nt(d(D,X)) & (17)\\
sd(X,Y) & \leftarrow & sd^*(X,Y), \neg nf(sd(X,Y)) & (18)\\
sd(X,Y) & \leftarrow & nt(sd(X,Y)) & (19)\\
m(X) & \leftarrow & m^*(X), \neg nf(m(X)) & (20)\\
m(X) & \leftarrow & nt(m(X)) & (21)\\
p(X,Y) & \leftarrow & p^*(X,Y), \neg nf(p(X,Y)) & (22)\\
p(X,Y) & \leftarrow & nt(p(X,Y)) & (23)
\end{array}$$

$$\begin{array}{rcll}
d^*(D,X) & \leftarrow & w(D,X) & (24)\\
d^*(D,X) & \leftarrow & h(D,X) & (25)\\
sd^*(X,Y) & \leftarrow & d^*(D,X), d^*(D,Y) & (26)\\
m^*(X) & \leftarrow & h(D,X) & (27)\\
p^*(X,Y) & \leftarrow & m^*(X), m^*(Y) & (28)\\
p(X,Y) & \leftarrow & w(D,X), w(D,Y) & (29)
\end{array}$$

$$\begin{array}{rcl}
pos(w(D,X)) & \leftarrow & pos^*(w(D,X)), \neg w(D,X),\\
& & \neg nf(w(D,X))\\
pos(h(D,X)) & \leftarrow & pos^*(h(D,X)), \neg h(D,X),\\
& & \neg nf(h(D,X))\\
pos(d(D,X)) & \leftarrow & pos^*(d(D,X)), \neg d(D,X),\\
& & \neg nf(d(D,X))\\
pos(sd(X,Y)) & \leftarrow & pos^*(sd(X,Y)), \neg sd(X,Y),\\
& & \neg nf(sd(X,Y))\\
pos(m(X)) & \leftarrow & pos^*(m(X)), \neg m(X),\\
& & \neg nf(m(X))\\
pos(p(X,Y)) & \leftarrow & pos^*(p(X,Y)), \neg p(X,Y),\\
& & \neg nf(p(X,Y))\\
pos(\neg w(D,X)) & \leftarrow & pos^*(\neg w(D,X)), w(D,X),\\
& & \neg nt(w(D,X))\\
pos(\neg h(D,X)) & \leftarrow & pos^*(\neg h(D,X)), h(D,X),\\
& & \neg nt(h(D,X))\\
pos(\neg d(D,X)) & \leftarrow & pos^*(\neg d(D,X)), d(D,X),\\
& & \neg nt(d(D,X))\\
pos(\neg sd(X,Y)) & \leftarrow & pos^*(\neg sd(X,Y)), sd(X,Y),\\
& & \neg nt(sd(X,Y))\\
pos(\neg m(X)) & \leftarrow & pos^*(\neg m(X)), m(X),\\
& & \neg nt(m(X))\\
pos(\neg p(X,Y)) & \leftarrow & pos^*(\neg p(X,Y)), p(X,Y),\\
& & \neg nt(p(X,Y))
\end{array}$$

$$\begin{array}{rcl}
pos^*(d(D,X)) & \leftarrow & pos^*(w(D,X))\\
pos^*(d(D,X)) & \leftarrow & pos^*(h(D,X))\\
pos^*(sd(X,Y)) & \leftarrow & pos^*(d(D,X)), d(D,Y)\\
pos^*(sd(X,Y)) & \leftarrow & d(D,X), pos^*(d(D,Y))\\
pos^*(sd(X,Y)) & \leftarrow & pos^*(d(D,X)), pos^*(d(D,Y))\\
pos^*(m(X)) & \leftarrow & pos^*(h(D,X))\\
pos^*(p(X,Y)) & \leftarrow & pos^*(m(X)), m(Y)\\
pos^*(p(X,Y)) & \leftarrow & m(X), pos^*(m(Y))\\
pos^*(p(X,Y)) & \leftarrow & pos^*(m(X)), pos^*(m(Y))\\
pos^*(p(X,Y)) & \leftarrow & pos^*(w(D,X)), w(D,Y)\\
pos^*(p(X,Y)) & \leftarrow & w(D,X), pos^*(w(D,Y))\\
pos^*(p(X,Y)) & \leftarrow & pos^*(w(D,X)), pos^*(w(D,Y))\\
pos^*(\neg d(D,X)) & \leftarrow & pos^*(\neg w(D,X))\\
pos^*(\neg d(D,X)) & \leftarrow & pos^*(\neg h(D,X))\\
pos^*(\neg sd(X,Y)) & \leftarrow & pos^*(\neg d(D,X)), d(D,Y)\\
pos^*(\neg sd(X,Y)) & \leftarrow & d(D,X), pos^*(\neg d(D,Y))\\
pos^*(\neg m(X)) & \leftarrow & pos^*(\neg h(D,X))\\
pos^*(\neg p(X,Y)) & \leftarrow & pos^*(\neg m(X)), m(Y)\\
pos^*(\neg p(X,Y)) & \leftarrow & m(X), pos^*(\neg m(Y))
\end{array}$$

$$pos^*(\neg p(X,Y)) \leftarrow pos^*(\neg w(D,X)), w(D,Y)$$
$$pos^*(\neg p(X,Y)) \leftarrow w(D,X), pos^*(\neg w(D,Y))$$

$$nt(d(D,X) \leftarrow add(d(D,X)) \quad (30)$$
$$nt(sd(X,Y)) \leftarrow add(sd(X,Y)) \quad (31)$$
$$nt(m(X)) \leftarrow add(m(X)) \quad (32)$$
$$nt(p(X,Y)) \leftarrow add(p(X,Y)) \quad (33)$$
$$nf(d(D,X) \leftarrow del(d(D,X)) \quad (34)$$
$$nf(sd(X,Y)) \leftarrow del(sd(X,Y)) \quad (35)$$
$$nf(m(X)) \leftarrow del(m(X)) \quad (36)$$
$$nf(p(X,Y)) \leftarrow del(p(X,Y)) \quad (37)$$

$$nt(sd(X,Y)) \leftarrow nt(d(D,X)), d^*(D,Y) \quad (38)$$
$$nt(sd(X,Y)) \leftarrow d^*(D,X), nt(d(D,Y)) \quad (39)$$
$$nt(sd(X,Y)) \leftarrow nt(d(D,X)), nt(d(D,Y)) \quad (40)$$
$$nt(p(X,Y)) \leftarrow nt(m(X)), m^*(Y) \quad (41)$$
$$nt(p(X,Y)) \leftarrow m^*(X), nt(m(Y)) \quad (42)$$
$$nt(p(X,Y)) \leftarrow nt(m(X)), nt(m(Y)) \quad (43)$$
$$nt(d(c1,X)) \leftarrow nt(sd(X,Y)) \quad (44)$$
$$nt(d(c1,Y)) \leftarrow nt(sd(X,Y)) \quad (45)$$

Some properties and remarks on the transformation technique are the following:

- The information given by the user in the form of extensional relations (or predicates) is not altered at all by the rewriting technique.

- The technique is also applicable for recursive rules although we did not explicitly show an example with recursion.

- If queries does not refer to possibly true and possibly false facts then most of the rules generated will not be involved in the query evaluation process at all. In our example, only the rules (16) to (29) and (30) to (45) are needed in such a case. Most of these rules are of a simple form (no joins are involved) and work only on facts introduced by update requests (*add*(*fact*) and *del*(*fact*)). Therefore, performance degradation is minimal when semi-modal information is not queried.

- All the rules generated are range restricted if the original rules are range restricted (each variable of a rule occurs in one of its positive body literals). The generated rules are thus amenable to set-oriented bottom-up query evaluation.

- The number of rules generated is linear in the number of the original rules.

- The technique is also applicable for the case when the original rules involve stratified negation (assumption 1(c)). The rules generated thereof are also stratified. Because of limitations of space this aspect is not illustrated in the paper.

In the foregoing, we have explained the key ideas behind the transformation technique by means of an example. The rewriting technique for Horn rules can be generalized from the example. The suggested rule transformation for deducing information that is necessarily true is not always complete. This occurs in situations where reasoning by cases is required. Therefore, the rules (38) to (45) of our example are not always complete although most situations are handled satisfactorily by these rules (see, for instance, the running example). In the following, we describe how to feed the new rules with appropriate factual information to achieve the desired cumulative view update behaviour.

6 Fact Maintenance

The fact maintenance described in this section feeds the rules generated in section 5 with appropriate factual information in order to achieve the desired behavior. The fact maintenance uses the constraints generated in section 5, the original constraints and the originally given rules. For each extensional predicate p we need a table or relation $pos^*(p)$ (the possibly true table) and a table $pos^*(\neg p)$ (the possibly false table).

For each transaction received, the following four steps are carried out:

1. Let $T_i = u_{i,1}, ..., u_{i,n_i}$ be the latest transaction received. For each $j (1 \leq j \leq n_i)$ insert the fact $add(u_{i,j})$ (resp. the fact $del(u_{i,j})$) in the DB if T_i requires the ground atom $u_{i,j}$ to be added (resp. to be deleted).

2. Let $u_{1'}, ..., u_{l'}$ be all update requests received so far for which there is still a tuple $add(u_{i'})$ or $del(u_{i'})$ kept internally. Generate all minimal fact modifications $m_1, ..., m_n$ realizing $u_{1'}, ..., u_{l'}$ by taking into account only the facts, the original set of rules, and the original set of constraints. This can be done using the method given in [Wüt93], for instance. (In our example of section 4, we generate the minimal solutions for these updates from the rules (4) to (9) and the constraints (1) to (3).)

3. Determine the information which can be confirmed, i.e. compute the maximal modification $m = (S^+, S^-) = \sqcap_{1 \leq i \leq n} m_i$ which is in all $m_i = (S_i^+, S_i^-)$. Add the tuples in S^+ to the base relations and delete the tuples S^- from the base relations. Empty the possibly true and possibly false

tables and store the tuples $(\cup_{1\leq i\leq n}S_i^+)-S^+$ in the possibly true tables and the tuples $(\cup_{1\leq i\leq n}S_i^-)-S^-$ in the possibly false tables.

4. Check the new integrity constraints ((10) to (15) in our example) and remove an internal tuple $add(u_{i'})$ (resp. $del(u_{i'})$) if an instance of a constraint $\leftarrow u_{i'}^*, add(u_{i'})$ (resp. $\leftarrow \neg u_{i'}^*, del(u_{i'})$) is violated.

Note that no additional integrity checking is need. The original constraints are taken into account when generating the minimal solutions in step 2 by relying on the method [Wüt93]; and the internal constraints are maintained in step 4.

This method is correct and complete for base facts. However, in the presence of recursive rules, the generation of all minimal fact modifications in step 2 may be too expensive. For pragmatic reasons , therefore, one should not generate all but only generate some of the minimal fact modifications, as proposed in [Wüt93]. This makes the technique somewhat incomplete also for base facts, but still leaves it correct.

The illustration of the whole view update method with the interaction between the rule transformation and the fact maintenance components was already anticipated in section 4. Processing the knowledge base $(\emptyset, \{(4),...,(9)\}, \{(1),...,(3)\})$ and the update sequence $sd(john, simon), w(1, simon), h(1, john)$ with the method given in sections 5 and 6 results in the cumulative view update behavior described in section 4.

7 Cumulative View Updates in Relational Databases

Application of the techniques presented in sections 5 and 6 to the relational case is straightforward. In the relational case, every rule of section 5 is represented as a view. Since views in the relational case are non-recursive, step 2 of fact maintenance which computes all minimal solutions for a given update is simplified. For non-recursive views there is always a finite number of minimal solutions, hence all of them can be computed as part of step 2 of fact maintenance. Consequently, it is possible to give a complete method for the relational case.

The methods given in section 5 and 6 for the deductive case suffer from incompleteness for two reasons. Firstly, there may be an infinite number of minimal solutions (for step 2 of fact maintenance, section 6) not all of which can be computed. However, for the relational case, the views are non-recursive. Therefore, there are always a finite number of minimal solutions for step 2 (of fact maintenance). Hence our techniques do not suffer from this form of incompleteness for the relational case.

Secondly, the techniques of section 5 and 6 are sometimes incomplete for the case where possibly true and possibly false facts originating from disjunctive views are also used in the derivation of other disjunctive views. For example, consider the following two view definitions.

```
CREATE VIEW ENGINEERINGDEP AS
SELECT EMP
FROM EMPDEP
WHERE DEP = 'DEVELOPEMENT'
OR DEP = 'RESEARCH'
```

```
CREATE VIEW ENGINEERINGMGR AS
SELECT EMP
FROM EMPDEP A, MGR B
WHERE (A.DEP = 'DEVELOPEMENT'
OR A.DEP = 'RESEARCH')
AND (A.EMP = B.EMP)
```

On receiving the information that Tim became a member of the engineering department whilst already knowing (or whilst receiving within the same transaction) the information that Tim is a manager, one can definitely deduce that Tim is also an engineering manager. Our methods, however, only deduce that Tim is *possibly* an engineering manager. This form of incompleteness can be removed for the relational case, making our techniques sound and complete.

A sound and complete method for the relational case is obtained by replacing step 3 of fact maintenance (section 6) by the following:

3.(a) Look for information which can be confirmed, i.e. compute the maximal modification $m = (S^+, S^-) = \sqcup_{1\leq i\leq n} m_i$ which is in all $m_i = (S_i^+, S_i^-)$. We add the tuples in S^+ to the base relations and delete the tuples S^- from the base relations. Empty the possibly true and possibly false tables and store the tuples $(\cup_{1\leq i\leq n}S_i^+) - S^+$ in the possibly true tables and the tuples $(\cup_{1\leq i\leq n}S_i^-) - S^-$ in the possibly false tables.

3.(b) For each modification m_i and each fact $pos(q)$ deducible from the rules given in section 5 such that q is an intensional predicate, check whether q is deducible under each modification m_i. If so, then add the fact $add(q)$, otherwise don't do anything.

This method, although complete, is highly inefficient. It has exponential time complexity. However, we believe that more efficient methods could be devised for the relational case.

8 Conclusions

We presented a novel idea, called cumulative updates, for intelligently assimilating updates in knowledge bases. The biggest advantage of cumulative updates compared to conventional updates is that in the presence of update ambiguities a misleading model of the world is never presented to the user by the knowledge base. Instead, the user is made aware of the information that s/he should acquire in order to make the database reflect the world being modeled more accurately, without ambiguities. However, when such information is not forthcoming, cumulative updates provide a mechanism for coping gracefully with the situation.

We gave motivating examples for cumulative updates and formally defined the notion of such updates in a very general setting. We also identified different classes of cumulative updates and defined the notions of correctness and completeness for cumulative updates. The simplest class of cumulative updates comprises of update requests that involve only the addition or deletion of base and derived facts. For this class, we proposed a method for computing cumulative updates and showed that the method imparts an "intelligent knowledge assimilation" behavior to the knowledge base. The method involves rewriting of the logical rules defining the views. Such a rewriting approach has never been proposed before in the literature for view updates. The rewriting technique may also be used for achieving traditional (i.e. non-cumulative) view updates in databases. Cumulative updates are also very relevant to the relational case as shown in this paper. A sound and complete method for cumulative view updates in relational databases is also given in this paper.

We view the work presented in this paper as laying the foundations for cumulative updates in data and knowledge bases. The solutions proposed herein are considered only as a first attempt. Many extensions are conceivable. For instance, we restricted ourselves to one particular class cumulative update problems where updates are made only on base and derived facts. Techniques should be developed for other classes as well. A study of the complexity for various classes of cumulative updates is also required. The rewriting technique for cumulative updates presented in this paper is a novel idea and may also be applied to realize non-cumulative updates.

Acknowledgements

We would like to thank some anonymous referees for detailed and valuable comments. Many thanks are also due to Mudasser Wyne for his kind help in the timely submission of this paper.

References

[ABW88] K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann Publishers, 1988.

[AGM85] C.E. Alchourron, P. Gardenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[BKSW91] T. Barsalou, A. Keller, N. Siambela, and G. Wiederhold. Updating relational databases through object-based views. In J. Clifford and R. King, editors, *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 248–257, 1991.

[Cla78] K. L. Clark. *Logic and Databases*, chapter Negation as Failure. Plenum Press, 1978.

[Dal88] M. Dalal. Investigations into a theory of knowledge base revision: Preliminary report. In *Proc. AAAI-88*, pages 475–479, 1988.

[Dec90] H. Decker. Drawing updates from derivations. In S. Abiteboul and P. C. Kanellakis, editors, *Proc. Int. Conf. on Database Theory (ICDT 90)*, pages 437–451, 1990.

[EG92] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactuals. In *Proc. PODS'92*, 1992.

[FKUV86] R. Fagin, G. M. Kuper, J. D. Ullman, and M. Y. Vardi. Updating logical databases. *Advances in Computing Research*, 3:1–18, 1986.

[Gar88] P. Gardenfors. *Knowledge in Flux*. MIT Press, 1988.

[GL90] A. Guessoum and J. W. Lloyd. Updating knowledge bases. Technical Report 89-05, Univ. Bristol Comp. Sc., 1990.

[Gra91] G. Grahne. Updates and counterfactuals. In *Proc. KR'91*, pages 269–276, 1991.

[GRS91] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *J.ACM*, 38(3), 1991.

[IL84] T. Imilienski and W. Lipski. Incomplete information in relational databases. *J. ACM*, 33(4):761–791, 1984.

[Imi86] T. Imilienski. Query Processing in Deductive Databases with Incomplete Information. In *Proc SIGMOD'86*, pages 268–280, 1986.

[KM90] A. C. Kakas and P. Mancarella. Database updates through abduction. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *Proc. 16th Int. Conf. on Very Large Data Bases*, pages 650–661, 1990.

[KM91] H. Katsuno and A.O. Mendelzon. On the difference between updating a knowledge base and revising it. In *Proc. KR'91*, pages 387–395, 1991.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 2nd edition, 1987.

[LLS93] D. Laurent, V. Phen Luong, and N. Sypratos. Updating intensional predicates in deductive databases. In *Proc. 9th IEEE Int. Conf. on Data Engineering*, 1993.

[Mar91] J.P. Martins. The truth, the whole truth, and nothing but the truth. an indexed bibliography to the literature of truth maintenance systems. *AI Magazine*, 11(5):417–428, 1991.

[Men87] E. Mendelson. *Introduction to Mathematical Logic.* Wadsworth & Brooks, 3 edition, 1987.

[ML91] G. Moerkotte and P. C. Lockemann. Reactive consistency control in deductive databases. *ACM TODS*, 16(4):670–702, 1991.

[MS86] J.P. Martins and S.C. Shapiro. Theoretical foundations for belief revision. In J.Y. Halpern, editor, *Proc. Theoretical Aspects of Reasoning about Knowledge*, 1986.

[Neb91] B. Nebel. Belief revision and default reasoning: Syntax-based approaches. In *Proc. KR'91*, pages 417–428, 1991.

[RB92] L. De Raedt and M. Bruynooghe. Belief updating from integrity constraints and queries. *Artificial Intelligence Journal*, 53:291–307, 1992.

[RdK87] R. Reiter and J. de Kleer. Foundations of assumption-based truth maintenance systems: Preliminary report. In *Proc. AAAI-87*, pages 183–188, 1987.

[Rei92] R. Reiter. Formalizing database evolution in the situation calculus. In *Proc. Int. Conf. on Fifth Generation Computer Systems (FGCS-92)*, 1992.

[SI91] K. Satoh and N. Iwayama. Computing abduction by using the tms. In K. Furukawa, editor, *Proc. 8th Int. Conf. on Logic Programming (ICLP 91)*, pages 505–518, 1991.

[Sri88] S. Sripada. A logical framework for temporal deductive databases. In *Proc. Int. Conf. on Very Large Data Bases*, August 1988.

[Sri93] S. Sripada. A metalogic programming approach to reasoning about time in knowledge bases. In *Proc. Int. Joint Conf. on Artificial Intelligence, IJCAI-93*, August 1993.

[TA91] R. Torlone and P. Atzeni. Updating deductive databases with functional dependencies. In *Proc. 2nd Int. Conf. on Deductive and Object-Oriented Databases (DOOD 91)*, 1991.

[Tom88] A. Tomasic. View update translation via deduction and annotation. In M. Gyssens, J. Paredanens, and D. Van Gucht, editors, *Proc. Int. Conf. on Database Theory (ICDT 88)*, pages 338–352, 1988.

[Ull82] J. D. Ullman. *Principles of Database Systems.* Computer Science Press, 2 edition, 1982.

[Ull88] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1,2. Computer Science Press, 1988.

[vEK76] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, 1976.

[Win90] M. Winslett. *Updating Logical Databases.* Cambridge University Press, 1990.

[Wüt93] B. Wüthrich. On Updates and Inconsistency Repairing in Knowledge Bases. In *9th IEEE Int. Conf. on Data Engineering*, pages 608–616, 1993.