DATABASE UPDATES THROUGH ABDUCTION

A.C. Kakas* and P. Mancarella**

*Department of Computing, Imperial College,180 Queen's Gate, London SW7 2BZ, UK.
**Dipartimento di Informatica, Universitàt di Pisa, Corso Italia, 40, I-56125 Pisa, Italy.

ABSTRACT:

The problem of view updates in deductive databases is studied by casting this in a naturally associated abductive framework. It is shown that this abductive approach deals successfully, in a simple yet powerful way, with the difficulties related to the presence of negation in the database and provides a uniform common procedure for insert and delete update requests. This procedure is formally defined and its correctness and completeness is investigated.

The abductive formalization of the update problem allows for a natural generalization of the basic update procedure in various ways. One important such extension is the fact the integrity checking asssociated with any update request can be dynamically incorporated into the update procedure so that potential inconsistent solutions to the request are trapped and rejected during their generation. It is also possible to extend the abductive approach to handle general non ground requests using constructive abduction and an associated form of constructive negation.

1. Introduction and Motivation

This paper is concerned with the problem of updating a deductive database. Given an update request insert(φ) (resp. delete(φ)), where φ is a closed first order formula, the task of updating is to change the database in such a way that the new modified database satisfies the update request. This problem is a generalization of the view update problem for relational databases (see e.g. [Banchilhon and Spyratos 81], [Cosmadakis and Papadimitriou 84], [Dayal and Bernestein 82], [Fagin et al. 83], [Furtado and Casanova 85], [Gottlob et al. 88]). At the same time it is a special case of a more general problem, namely that of Knowledge Assimilation (see e.g. [Kowalski 79]) where the new information to be assimilated is the truth (resp.falsity) of φ.

Proceedings of the 16th VLDB Conference
Brisbane, Australia 1990

It has been argued that a new piece of information should not necessarily be assimilated into the knowledge base by the explicit addition of this information. Instead, it may be useful to add a set of (defeasible) hypotheses that together with the current state of the knowledge base is sufficient to prove the new information. These hypotheses are also required to be consistent with the current state of the knowledge base. In other words, the new information is assimilated implicitly by the explicit addition of the hypotheses.

There are several reasons why such an implicit assimilation of information is desirable. In some cases a new piece of information may carry with it other implicit information that would be lost if this is just added explicitly to the knowledge base. For example, consider a knowledge base about families organized primarily in terms of the Parent relation. Then a new piece of information such as "Tom is the brother of Bob", if added explicitly as a fact does not capture other implicit information such as the fact that Tom and Bob have the same grandparents. On the other hand, if this is assimilated by adding a common parent for Bob and Tom then the knowledge base would contain implicitly the fact that they have the same grandparents. Another related reason for assimilating information implicitly, is the fact that relevant new information may not always be acquired in terms directly related to the way the knowledge base itself is organized. Such new pieces of information need first to be "translated" into a form that reflects the organization of the information within the knowledge base, before they can be included in the knowledge base, if this is to maintain its original structure.

The problem of updating a deductive database, that concerns us in this paper, is an important example where these ideas are relevant. In a deductive database DB[1] the knowledge is separated into two parts:

(a) the *extensional* database (EDB) - a set of ground facts on *base* relations (base predicates) which describes the state of the world.

[1] In this paper we will regard a deductive database as a logic program.

(b) the *intensional* database (IDB) - a set of rules (view) on *view* relations (view predicates) which are regarded as reasoning mechanisms with which new facts can be derived.

Different users may have different sets of rules (view) IDB with which they can manipulate the data in the common extensional part of the database. Whenever a new piece of information on a view relation arrives at a particular user, this is assimilated by appropriately changing the EDB rather than adding it explicitly into the view. In other words, an update request insert(φ) or delete(φ) on DB $\equiv$ IDB $\cup$ EDB, where φ is in terms of view predicates - a *view update* request - is accomplished by drawing suitable transactions on EDB only. In this way different users share the new view information received by each one, and the structure of the database maintains its original form.

There is another important reason that motivates why a view update request should be effected by changing the EDB only. This is the fact that the rules (view) are generally regarded to be complete and non-defeasible and thus should not be changed, whereas the EDB is regarded as containing incomplete and defeasible knowledge about the world, i.e. data on base relations may be added or deleted in the future. From this point of view it is natural to consider the base predicates as *abducible* predicates (predicates whose instances can be assumed when required) and the problem of view updates as an abductive problem. In this abductive framework an update request insert(φ) (resp. delete(φ)) is regarded as a new observation that needs to be explained, i.e. explain(φ) (resp. explain($\neg\varphi$)), in terms of possible hypotheses on the abducible (assumable) base predicates. Note that at first sight this appears to differ from the way the view update problem is tackled in the relation database case where both the view and the database are modified by an update. However, in our case for deductive databases the view is modified indirectly through the changes in the extensional database.

In the next section we clarify further the connection with abduction and present the basic ideas behind this approach. In section 3 the details of the abductive framework and the basic strategy of the update procedure are presented with particular emphasis on the problem of negation. The main features of this procedure are illustrated in section 4 before this is formally defined. Its correctness and completeness are investigated in section 5. Finally, in the last section important extensions of the update procedure that follow naturally from the abductive approach are discussed.

The problem of view updates in deductive databases has recently received much attention by various authors. Work related to the approach taken here can be found in [Bry 90] where the update request is effected by generating a suitable model that can support it. For definite databases (i.e. with no negation) this has been tackled in [Nicolas and Yazdanian 83] [Tomasic 88]. The problems with the presence of negation are studied in [Guessoum and Lloyd 90a] [Guessoum and Lloyd 90b] [Decker 89] in terms of SLDNF derivations. In [Fagin et al. 83] the problem of dealing with the different ways in which an update can be effected has been studied. Their work has been extended in [Rossi and Naqvi 89]. Other work on the update problem can be found in [Ross 85] [Manchanda and Warren 88] [Abiteboul 88] and references therein.

2. Basic ideas

In the previous section it has been argued that the problem of updating a deductive database with an update request on view predicates (in IDB) should be effected by changing appropriately the extensional part (EDB) of the database. This motivates the translation of the original update request on view predicates to an equivalent update request on base predicates alone. In this section we present the basic strategy of an update procedure based on such a translation of the problem and explain how abduction can be used to provide this translation. It is important to note that despite this distinction between the view and the extensional parts of the database it is possible to handle within the same abductive approach view update requests that can not be satisfied by any set of hypotheses on the base predicates and hence require changes in the view. This will be discussed in section 6.

First, let us state some assumptions that we will make. We will assume, unless stated otherwise, that all update requests are of the form insert(φ) or delete(φ) where φ is a ground instance of a view predicate (see section 6 for comments about the case where φ is a general first order formula). For the most part of this paper, we will assume that the database is locally stratified [Przymusinski 88]. The technical results will be presented with respect to the stable model semantics [Gelfond and Lifschitz 88] for negation as failure associated with such databases. One of the main motivations for choosing this semantics is the fact that it allows a natural formulation of various extensions of our basic approach, as we will discuss in the last section. Before proceeding with the update problem let us briefly overview the stable model semantics and its suitability for deductive databases.

Let P be a logic program and I a Herbrand interpretation. We denote by Π^I the following (possibly infinite) set of ground definite Horn clauses:

$\Pi^I = \{H \leftarrow B_1,\dots,B_k \mid H \leftarrow B_1,\dots,B_k, \neg L_1, \dots, \neg L_m$ is a clause in ground(P) and $L_i \notin I$ for each $i=1,\dots,m\}$

where ground(P) denotes the set of all ground instances of clauses in P. Notice that if a clause has no negative literal in its body, then all its ground instances belong to Π^I, for any I; moreover if a clause in ground(P) contains a negative literal $\neg L$ such that $L \in I$, then such a clause does not contribute to the set Π^I.

Definition ([Gelfond and Lifschitz88])
Let P be a logic program and M a Herbrand interpretation. M is a stable model of P iff M is equal to the minimal Herbrand model of Π^M.

Definition ([Gelfond and Lifschitz88])
The stable model semantics is defined for a logic program P if P has exactly one stable model M and it declares M to be the meaning of P.

Proposition ([Gelfond and Lifschitz88])
Let P be a locally stratified logic program. Then P has a unique stable model and hence a stable model semantics.

Stable model semantics for logic programs with negation as failure finds its roots in Moore's autoepistemic logic [Moore85], for *beliefs*. A stable model M of P is a *closed* rational set of beliefs given P as the premises. If M is the set of atoms that we believe to be true then any rule that has $\neg L$ with $L \in M$ in its conditions is useless; any condition $\neg L$ with $L \notin M$ is believed to be true and hence can be dropped. The result of this process is to transform the rules in ground(P) in a set of definite clauses, Π^M. If M is the set of atomic logical consequences of Π^M then M is rational.

example
The program (view)

$p \leftarrow \neg q$
$q \leftarrow B$
$r \leftarrow B_1$

has a unique stable model {p}.

For deductive databases, it is appropriate to regard a view, IDB, as representing a *collection* of stable models, each one defined by a particular state, E, of the extensional database (see[Kakas and Mancarella 90a]). In the previous example this correspondence between the state of the extensional database and stable models of the whole database is:

E: $\emptyset$	M= {p}
E: B.	M= $\{q\} \cup \{B\}$
E: B_1.	M= $\{r, p\} \cup \{B_1\}$
E: B. B_1.	M = $\{r, q\} \cup \{B, B_1\}$

Roughly speaking, given an update request abduction chooses a subclass of suitable stable models by defining a set of conditions that the corresponding extensional database must satisfy.

The task of a view update procedure is to define a suitable set of transactions Tr on the extensional database so that the resulting database satisfies the update request. In this paper, by the satisfaction of an update request insert(φ) (resp. delete(φ)) on a database DB, we mean that φ (resp. $\neg\varphi$) is true in the stable model of DB. We will be primarily interested in finding a specification for these transactions Tr. We will not study here in detail how these transactions would be effected on the extensional database. This would depend on various assumptions that the database may have (e.g. domain assumptions). In any case, the generated specification reduces the problem from updating a deductive database to updating its extensional part, which can be solved by existing methods for relational databases.

In order to translate the original view update request into an update problem for the extensional database, we will use abduction within a naturally associated abductive framework. Abduction has recently received much attention in Artificial Intelligence and has been recognized as an appropriate form of non-monotonic reasoning with incomplete and changing knowledge (see e.g. [Poole 88] [Eshghi and Kowlaski 89]). This theoretical and mainly abstract idea can be simplified and applied in a natural way to the view update problem as we shall see below. In fact, its usefulness to this problem gives it practical value and can help in the further development of abduction. Let us briefly review the basic notion of abduction. By an *abductive framework* <T,A,I> we mean (see [Eshghi and Kowlaski 89], [Kakas and Mancarella 90a]) a theory (logic program) T where a collection of predicates in T, A, are designated as abducibles, together with a (possibly empty) integrity theory I. The abducible predicates have no definition in T but are assumable, and given an observation Q the aim of abduction is to *explain* Q by finding a consistent set Δ of hypotheses involving the abducibles, which together with T imply Q. Intuitively the abducible predicates represent the incomplete information in the theory and can be used at any time (observation) to partially complete the theory. Furthermore the hypotheses in Δ are defeasible and would be withdrawn or replaced if they become inconsistent with later observations.

Returning now to our problem of view updates, it is clear from the above description of abduction that these are closely related. At first, let us consider a definite database DB $\equiv$ EDB $\cup$ IDB, i.e. where the rules in IDB have no negative conditions. A naturally associated abductive framework is <IDB, A, $\emptyset$> where A is the set of base predicates (base predicates will always be denoted by a capital B, e.g. B, B_1, B_2, ...). An update request, insert(p) say, can be equivalently regarded as the abductive query to explain the observation p in <IDB, A, $\emptyset$>. The set Δ generated by abduction for p is a specification of a set of sufficient requirements that the extensional database, EDB, should satisfy for the original request to be effected. Thus abduction provides a translation of the original view update request on the whole database, DB, into an update request on the extensional database, EDB, defined through Δ. To illustrate these ideas and the basic strategy of an update procedure let us consider a simple example.

example 1

IDB: $p(x) \leftarrow B(x)$ EDB:

$B(c)$.

$B(d)$.

Update: insert(p(a))

The first step, step A, of our update procedure is to solve abductively the query p(a) on IDB alone irrespective of the information in EDB. Reasoning backwards from $\leftarrow$ p(a) we obtain $\leftarrow$ B(a) where B(a) is an abducible. To complete the proof we can assume B(a) since it is consistent to do so thus giving a successful abductive proof with $\Delta=\{B(a)\}$. This in effect translates the insert(p(a)) request on DB to the update request insert(B(a)) in EDB. Schematically:

$$\begin{tikzcd}\boxed{\begin{array}{c}\text{insert(p(a))}\\ \text{in DB}\end{array}} \arrow[r, "\text{Abduction}", "\Delta"'] & \boxed{\begin{array}{c}\text{insert(B(a))}\\ \text{in EDB}\end{array}}\end{tikzcd}$$

The second step, step B, of the update procedure would involve solving the update problem on EDB generated in step A. In this example this would be achieved by adding B(a) to EDB. (As mentioned earlier, in this paper we will not be concerned with how step B is accomplished).

In general, the existence of negation complicates the view update problem but we can maintain the close connection with abduction by adapting and extending the abductive approach introduced in [Eshghi and Kowalski 89] for simulating and generalizing negation by failure in logic programming. The related abductive framework now allows negations, as well as base predicates, to be treated as abducibles. Then as before, an update request insert(p) is translated to an update request on EDB through the result Δ of abduction for explain(p).

Finally, for delete view update requests, e.g. delete(p), the abductive approach for negation will allow, as we shall see in the next section, to treat these as insert requests, e.g. insert(not p) and thus provide a completely uniform framework with a single procedure for both insert and delete requests.

3. Non-definite databases

The existence of negation in IDB introduces added complications to the view update problem. In [Guessoum and Lloyd 90a] and [Decker 89] it has been pointed out that it is possible for some solution of an update request drawn from SLDNF derivations, to be invalidated due to negation. One way to avoid this problem of invalidation, proposed in [Guessoum and Lloyd 90a], is to impose the strictness condition [Kunen 87] on IDB. In [Decker 89] this problem is tackled by trying to find all possible ways in which an update request can be effected thus ensuring that any valid solutions would be amongst the ones found. In this section we present the full abductive framework that we will be using which incorporates negations as abducibles and describe how the basic ideas and strategy for the update procedure presented in the previous section carry through for the case where negation is present. In the following sections we will see how the complications of negation are handled within this abductive approach.

Given a general logic program with negative conditions, P, the abductive framework for negation [Eshghi and Kowlaski 89] associated with P is <P*, A*, IC*> where:
- P* is the (definite) logic program obtained from P by replacing each negative literal, $\neg q(t)$, by a new positive literal $q^*(t)$;
- A* is the set of predicate symbols, q*, introduced in P*;
- IC* is the set of denials

  $\leftarrow q(x), q^*(x)$

and meta-level statements

  $\text{Demo}(P^*\cup\Delta, q(x)) \vee \text{Demo}(P^*\cup\Delta, q^*(x))$

for all predicate symbols, q*, introduced in P*.

Here IC* is to be thought of as an integrity theory which any generated set of hypotheses Δ together with P* must satisfy. In the semantics of stable models that we associate to P (and generalized stable models for <P*, A*, IC*> see [Kakas and Mancarella 90a]) the meta-level integrity constraint $\text{Demo}(P^*\cup\Delta, q(x)) \vee \text{Demo}(P^*\cup\Delta, q^*(x))$ means truth of the disjunction $q(x) \vee q^*(x)$ in these models and can be replaced by this object-level disjunctive integrity constraint.

For the problem at hand where P is IDB we need to adapt this framework (due to delete requests) by introducing a new predicate q* for all predicates in IDB. Furthermore, this framework is extended by treating the base predicates and their negations as abducibles, giving the abductive framework

  <IDB*, Ab, IC*>

where Ab is the set of all base predicates together with a new predicate, α^*, for every view or base predicate α, and IC* is defined as above for every predicate symbol α^*. Note that the extensional database EDB is not taken into account in the definition of this abductive framework. For the following example:

example 2

IDB: $p(x) \leftarrow \text{not}\, q(x)$

$q(x) \leftarrow B(x)$

the associated abductive framework is:

IDB*:	$p(x) \leftarrow q^*(x)$	
	$q(x) \leftarrow B(x)$	
Ab:	$\{B, B^*, p^*, q^*\}$	
IC*:	$\leftarrow q(x), q^*(x)$	$q(x) \vee q^*(x)$
	$\leftarrow p(x), p^*(x)$	$p(x) \vee p^*(x)$
	$\leftarrow B(x), B^*(x)$	$B(x) \vee B^*(x)$

With this framework the basic strategy for the update procedure remains as before: Given a view update request insert(p), the first step, step A, tries to solve abductively the query $\leftarrow p$ *in <IDB*, Ab, IC*> generating* a set of abducibles Δ such that

IDB* ∪ Δ entails p

IDB* ∪ Δ is consistent and does not violate the integrity constraints in IC*.

This Δ translates the original update request into an update problem, Update(Δ, EDB), on EDB, i.e.

$$\begin{tikzcd} \boxed{\begin{matrix}\text{insert(p)}\\ \text{in DB}\end{matrix}} \arrow[r, "\text{Abduction}", "\Delta"'] & \boxed{\text{Update}(\Delta, \text{EDB})} \end{tikzcd}$$

Then the second step, step B, of the update procedure will solve the Update(Δ,EDB) problem by drawing a suitable set of transactions, Tr, on EDB such that Tr(EDB) entails any assumption in Δ involving only base predicates.

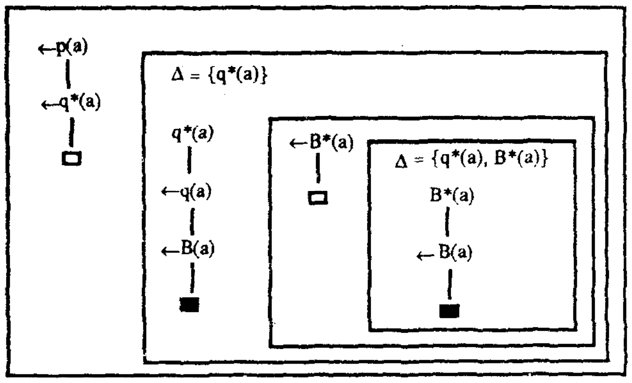
One useful way to view these steps of the update procedure is to regard step A as an investigation of whether the update request can indeed be supported by IDB, by finding a sufficient set Δ of "beliefs" about the base predicates that could achieve the request. Then step B can be regarded as a revision of "beliefs" with priority on the beliefs expressed by Δ over those present in EDB, whenever a conflict exists between the two. At the end the new set of "beliefs" Tr(EDB) ensures that the update request is satisfied.

In the case where step A fails to return any Δ this indicates that the update request, insert(p), can *not* be supported by IDB, no matter how EDB is changed. Hence failure to find a Δ means that the update request in not "well-posed" and the update procedure would "fail". For the update request to succeed it will be necessary to change the view IDB itself. In section 6 we will indicate how this can be done, without changing the form of the update procedure, by suitably extending the set of abducibles Ab.

An abductive solution, Δ, is usually required to be minimal in the sense that no proper subset of Δ is itself a solution. Despite this minimality condition it is possible that there could exist many different minimal abductive solutions, $\Delta_1,...,\Delta_n$, each of which will define a different way in which the update request can be satisfied. In this paper, we will not study the important problem of defining a preference among these different choices and only make a few comments on this problem here.

Some of these choices may be invalidated due to the integrity theory of the database (see section 6) but it is still possible for more than one solution to remain. It may be that domain specific priorities on the abducibles (base predicates) exist which can be used to provide a preference between these choices. Another way to choose between different solutions is to prefer those that have fewer additional consequences when effected in the database. Note that due to the presence of negation as failure in the rules of IDB it is possible for a non minimal solution to be prefered over its minimal subet under this criterion. Yet another way is to use some facility, like query-the-user, to generate tests that would discriminating between these choices by rejecting those that are incompatible with the results of these tests. Unfortunately, even in the case where all of these methods are used it is not always possible to get a unique prefered choice or indeed if such a choice exists that this will be the correct one. In view of this we would propose an alternative complementary approach to this problem whereby a method for recovery from inconsistency that allows backtracking to previous update requests is developed. Then if in the face of new information, eg. new update requests, the database becomes inconsistent due to a wrong choice of solution at some earlier update we would be able to recover by backtracking and choosing another solution. Such a truth maintenance system can be developed in a natural way within our abductive framework by recording and manipulating the abductive solutions (see [Kakas and Mancarella 90b]). Together with this recovery mechanism we could also generalize the query evaluation methods of the database to allow conditional answers to queries that make explicit the dependency of any answer to a query on the choice of solutions to the update requests prior to the query. In this way the database can recover from a wrong choice when this is made apparent while in the meantime it safeguards against mistaken choices by considering any piece of information extracted from the database that depends on a choice of solution to an earlier update request to be conditional on that choice being correct.

In order to illustrate the basic structure of the abductive procedure let us consider example 2 with EDB = {B(a)} and the update request insert(p(a)). The abductive search space of step A is shown below:



giving $\Delta = \{ q^*(a), B^*(a) \}$ where □ denotes success of a branch and ■ failure. Any part of the search space enclosed in a bold outlined box shows the consistency checking of the new hypothesis that is added to the Δ. This is done by adapting and extending the consistency method of [Sadri and Kowalski 88]. Note that in such a consistency tree success in showing the consistency of the abducible at the top requires that all its branches end up in failure. In the large bold box the hypothesis $q^*(a)$ is checked against the integrity constraint $\leftarrow q(x), q^*(x)$. At the tip of the branch the required failure of $B(a)$ triggers the query $\leftarrow B^*(a)$ due to the integrity constraint $B(a) \vee B^*(a)$ which in turn requires $B^*(a)$ to be added to the Δ.

From the generated $\Delta = \{q^*(a), B^*(a)\}$ the original view update request has been reduced to Update $(B^*(a), EDB)$ which can be effected by removing $B(a)$ from EDB, i.e. a minimal set of transactions Tr would be Tr = {remove(B(a))}.

Finally, delete requests such as delete(p) are handled by transforming them to an insert request, i.e.

delete(p) $\Leftrightarrow$ insert(not p) $\Leftrightarrow$ insert(p*)

In the previous example 2 the delete request, delete(q(a)), will be transformed to the abductive query $\leftarrow q^*(a)$. Its search space will be the same as that shown for insert(p(a)) without the first resolution step and subsequently the solution will be the same, i.e. Tr = { remove(B(a)) }. In this way inserts and deletes are treated uniformly in the same way within one procedure. This allows us to generalize in a trivial way the set of possible updates by allowing mixed multiple requests of the form

insert(p_1) and ... insert(p_n) and delete(q_1) and ... and delete(q_k).

Such an update request is transformed to the abductive query

$\leftarrow p_1, \ldots, p_n, q_1^*, \ldots, q_k^*$

for step A.

4. Main features of the update procedure

In this section we illustrate the main features of the update procedure and the way it handles various problems, in particular those related to negation discussed in the previous section, through a series of examples. These will also be helpful to explain further the definition of the abduction procedure of step A before its formal definition, which will be given in the next section.

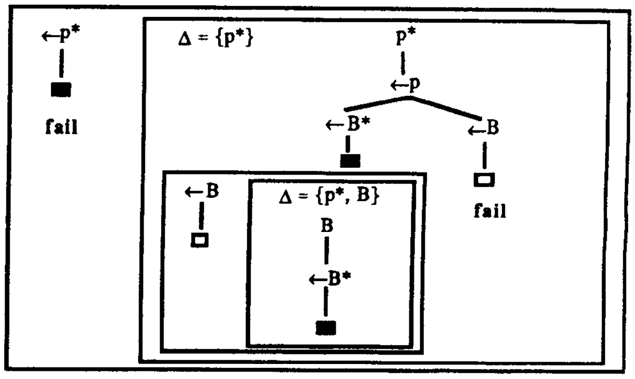
example 3

IDB*: $p \leftarrow B^*$ EDB: B.

$p \leftarrow B$

Update: delete(p).

The abductive search space for the query $\leftarrow p^*$ is:



The abductive procedure fails to find a Δ. The reason for this is the fact that for the consistency checking of any abducible all branches in the consistency tree must end in ■. Here the second branch of the consistency search space for p* ends in □ due to the fact that for the first one to end in ■ B has to be added in the Δ.

The fact that the abductive procedure fails to find a Δ indicates that the update request can not be supported by any set of beliefs in EDB with the given IDB: if we want to satisfy the update request we must change IDB in some way (see section 6). Thus we see that the naive "solution" of removing B from EDB which is invalidated by the presence of negation is not drawn within our approach. In general, the invalidation due to negation is subsumed by the abductive procedure of step A. As a result, it is not necessary to impose the restriction of strictness onto IDB. Consider the following non-strict related example:

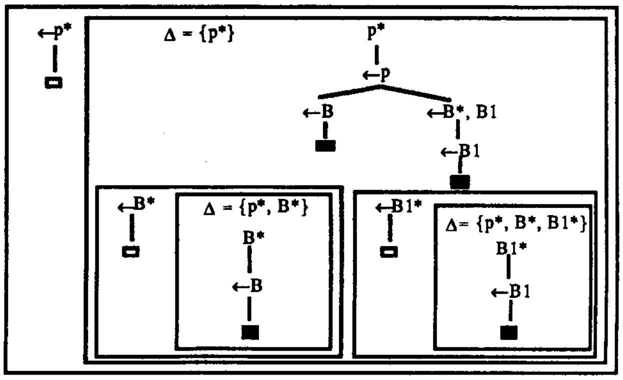
example 4

IDB*: $p \leftarrow B^*, B_1$ EDB: B.

$p \leftarrow B$ B_1.

Update: delete(p)

The abductive search space for $\leftarrow p^*$ is:



giving a final Δ of $\{p^*, B^*, B_1^*\}$. Then the update request can be satisfied if B^* and B_1^* (or $\neg B$ and $\neg B_1$) are true in EDB which is achieved, for instance, by Tr = {remove(B), remove(B_1)}. Another non-strict example is:

example 5

IDB*: $p \leftarrow q^*, B$ EDB = $\emptyset$

$q \leftarrow B_1^*, B$

Update: insert(p)

The update request requires q* to be consistent. There are two sets of hypothesis that can support the consistency of q*, reflecting the two possible ways the goal $\leftarrow B, B_1^*$ can fail in the consistency tree of q*. At such a goal we have a forking (backtracking point) in the abductive procedure, giving $\Delta_1=\{q^*, B^*\}$ and $\Delta_2=\{q^*, B_1\}$. The success in proving p requires the additional assumption B. The consistency check of B will reject Δ_1 and give as final Δ only $\Delta = \{q^*, B_1, B\}$. Notice that the procedure takes an *active* view towards ensuring that required negations are consistent. This means that when a negation q* is required, instead of first looking for the non-failed branches of $\leftarrow q$ and then trying to make them fail, we try from the start to make all branches of $\leftarrow q$ fail. Moreover the information about the failure of each branch, which is in fact part of the solution to the update request, is dynamically recorded in the Δ and is used to ensure that the failure of subsequent branches is indeed consistent with that of the previous ones.

Whenever the update request is not ground or when local variables are involved in the definition of the view predicates the generated Δ in step A is not fully ground. Consider for example:

example 6

IDB*: $p \leftarrow B_1(x), B_2(x)$

EDB: $B_1(a)$.

Update: insert(p)

The abductive procedure of step A can be extended to a form of constructive abduction that allows existentially quantified hypotheses. During the execution of the procedure a "grounding" substitution is applied that replaces any such variable with a new (skolem) constant. The final generated Δ for this example will be $\Delta=\{\exists y(B_1(y),B_2(y))\}$ obtained by reversing the grounding substitution. Then one solution of the reduced problem, Update(Δ, EDB), that is minimal with respect to what already exists in EDB is given by $Tr_1=\{add(B_2(a))\}$ but notice that another solution would be $Tr_2=\{add(B_1(o)), add(B_2(o))\}$ where 'o' is a new individual. Similarly, for the view, IDB*: $p(x) \leftarrow B_1(x),B_2(x)$, and the update request, insert($\exists x p(x)$), the same Δ and thus the same two solutions will be generated.

Another interesting example with local variables is :

example 7

IDB*: $p \leftarrow B_1^*(x), B_2^*(x)$

EDB: $B_1(a). B_2(a). B_3(c)$

$B_1(b). B_2(b)$.

Update: delete(p)

The consistency of p* requires that no individual 'o' exists such that $B_1(o)$ and $B_2(o)$ are both false. This is handled by adding the denial $\forall x(\leftarrow B_1^*(x),B_2^*(x))$ to the abductive assumptions resulting in the final $\Delta=\{p^*, \forall x(B_1(x) \vee B_2(x))\}$. Hence the required transactions Tr are defined by the condition that Tr(EDB) must entail $\forall x(B_1(x) \vee B_2(x))$. Such abduced denials are first checked for consistency against the current assumptions in the Δ and then they act as new additional integrity constraints that must be satisfied by any subsequent additions to the Δ. This will be done in the consistency phase of the procedure alongside with that for the constraints IC* and any constraints of the database. For more discussion on non ground updates and more general integrity checking see section 6.

5. Correctness and completeness of the Update Procedure

In this section we will present the formal definition of the abductive proof procedure with theorems for its correctness and completeness. We will show that whenever the procedure succeeds there exists a state E of the extensional database such that the update request is true in the stable model of the resulting database. Similarly, under suitable conditions on IDB, the procedure is complete in the sense that if there exists a state E of the extensional database such that the update request is true in the corresponding stable model of the database, then the procedure will generate a sufficient set of assumptions to define this state. In appendix A we review the notion of stable models and discuss how these are appropriate for deductive databases and the view update problem.

As indicated by the examples of the previous sections, the abductive proof procedure is an interleaving of two activities: 1) reasoning backwards for a refutation collecting any required abductive assumption and 2) checking for the consistency of these assumptions. The procedure will be defined by suitably adapting and extending the procedure of [Eshghi and Kowalski 89] which itself can be seen as an extension to SLDNF. In this paper, in order to simplify the presentation we will define only a restricted form of the procedure where only ground abducibles are allowed to be abduced.

In the sequel, given an atom $L=p(t_1,\ldots,t_k)$ (resp. $p^*(t_1,\ldots,t_k)$) we will denote by L* the atom $p^*(t_1,\ldots,t_k)$ (resp. $p(t_1,\ldots,t_k)$). Moreover an abducible atom is called ***base abducible*** if it has the form $B(t_1,\ldots,t_k)$ or $B^*(t_1,\ldots,t_k)$, where B is a base predicate.

Definition (*safe selection rule*)
A safe selection rule R is a (partial) function which, given a goal $\leftarrow L_1, \ldots, L_k$ $k \geq 1$ returns an atom L_i, $i=1,\ldots,k$ such that:

either i) L_i is not abducible;
or ii) L_i is ground.

From now on we will refer to a *safe* selection rule R.

Definition (*abductive proof procedure*)
An ***abductive derivation*** from $(G_1\ \Delta_1)$ to $(G_n\ \Delta_n)$ via rule R is a sequence

$$(G_1\ \Delta_1), (G_2\ \Delta_2), \ldots, (G_n\ \Delta_n)$$

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

A1) If L_j is not abducible, then $G_{i+1}=C$ and $\Delta_{i+1}=\Delta_i$ where C is the resolvent of some clause in IDB* with G_i on the selected literal L_j;
A2) If L_j is abducible and $L_j \in \Delta_i$, then $G_{i+1} = \leftarrow L_1,\ldots,L_{j-1},L_{j+1},\ldots,L_k$ and $\Delta_{i+1}=\Delta_i$;
A3) If L_j is a base abducible, $L_j \notin \Delta_i$ and $L_j^* \notin \Delta_i$ then $G_{i+1} = \leftarrow L_1,\ldots,L_{j-1}, L_{j+1},\ldots,L_k$ and $\Delta_{i+1} = \Delta_i \cup \{L_j\}$;
A4) If L_j is a non-base abducible and $L_j \notin \Delta$ and there exists a consistency derivation from $(\{\leftarrow L_j^*\}\ \Delta_i \cup \{L_j\})$ to $(\{\}\ \Delta')$ then $G_{i+1} = \leftarrow L_1,\ldots,L_{j-1}, L_{j+1},\ldots,L_k$ and $\Delta_{i+1} = \Delta'$.

Steps A1) and A2) are SLD-resolution steps with the rules of IDB* and abductive hypotheses, respectively. In step A3) and A4) a new abductive hypotheses is required and it is added to the current set of hypotheses provided it is consistent. Note that in case A3) the consistency checking is trivial.

A ***consistency derivation*** from $(F_1\ \Delta_1)$ to $(F_n\ \Delta_n)$ is a sequence

$$(F_1\ \Delta_1), (F_2\ \Delta_2) \ldots (F_n\ \Delta_n) \text{ to } (F_n\ \Delta_n)$$

such that for each $i>1$ F_i has the form $\{\leftarrow L_1,\ldots,L_k\} \cup F_i'$ and ***for some*** $j=1,\ldots,k$ $(F_{i+1}\ \Delta_{i+1})$ is obtained according to one of the following rules:

C1) If L_j is not abducible, then $F_{i+1} = C' \cup F_i'$ where C' is the set of all resolvents of clauses in IDB* with $\leftarrow L_1,\ldots,L_k$ on the literal L_j and $\Box \notin C'$, and $\Delta_{i+1}=\Delta_i$;
C2) If L_j is abducible, L_j is ground, $L_j \in \Delta_i$ and $k>1$, then $F_{i+1}=\{\leftarrow L_1,\ldots,L_{j-1}, L_{j+1},\ldots,L_k\} \cup F_i'$ and $\Delta_{i+1}=\Delta_i$;
C3) If L_j is abducible, L_j is ground and $L_j^* \in \Delta_i$, then $F_{i+1}=F_i'$ and $\Delta_{i+1}=\Delta_i$;
C4) If L_j is a base abducible, L_j is ground, $L_j \notin \Delta_i$ and $L_j^* \notin \Delta_i$, then $F_{i+1}=F_i'$ and $\Delta_{i+1}=\Delta_i \cup \{L_j^*\}$;
C5) If L_j is a non base abducible, L_j is ground and there exists an abductive derivation from $(\leftarrow L_j^*\ \Delta_i)$ to $(\Box\ \Delta')$ then $F_{i+1}=F_i'$ and $\Delta_{i+1}=\Delta'$.

The consistency derivations do not rely on a particular selection rule, since in general all the possible ways in which a conjunction $\leftarrow L_1,\ldots,L_k$ can fail should be explored. This introduces an extra non determinism in the proof procedure. In case C1) the current branch splits into as many branches as the number of resolvents of $\leftarrow L_1,\ldots,L_k$ with the clauses in IDB* on L_j. If the empty clause is one of such resolvents the whole consistency check fails. In case C3) the current branch is already consistent under the assumptions in Δ_i, and this branch is dropped from the consistency checking. Case C4) and C5) correspond to the enforcement of the disjunctive integrity constraint $L_j^* \vee L_j$: the current branch of the consistency search space can be dropped provided $\leftarrow L_j^*$ is abductively provable. Notice that this may require new hypotheses to be added to the current Δ. In C4) this is accomplished by adding L_j^*.

The correctness and completeness results of the procedure will be stated in terms of a set of transactions on the extensional database. By a transaction we mean an action involving the addition or removal of tuples from the extensional database. As mentioned above, we are only concerned with the problem of generating a specification for the set of transactions on EDB which, if effected, would accomplish the update request. This specification is obtained from the set of assumptions Δ generated by the procedure, by dropping any non base abducible. In the sequel we will denote by $Tr(E)$ the state of the extensional database resulting by effecting on E each transaction in Tr.

Definition (*transactions associated with a* Δ)
Let Δ be a set of assumptions generated by the abductive procedure. Then Tr_Δ denotes any set of transactions such that, given a state E of the extensional database, for each base abducible $L \in \Delta$ $(L^* \in \Delta)$, $Tr_\Delta(E) \models L$ $(\neg L)$.

The soundness theorem is proved using the following lemma which justifies the above definition of transactions drawn from a successful abductive derivation[1].

Lemma 1

[1] The technical proofs are omitted from this paper due to lack of space and can be found in [Kakas and Mancarella 90c].

Let IDB be locally stratified, Q be the conjunction $A_1,\dots,A_n,\neg A_{n+1},\dots,\neg A_m$. Assume that $(\leftarrow A_1,\dots,A_n,A^*_{n+1},\dots,A^*_m\ \{\})$ has an abductive derivation to $(\square\ \Delta)$. Then for each EDB, if M is a stable model of IDB$\cup$EDB such that for each base abducible $L\in\Delta$ $(L^*\in\Delta)$ $M\models L$ $(\neg L)$, then for each non base abducible $p^*(t_1,\dots,t_n)\in\Delta$, $M\models\neg p(t_1,\dots,t_n)$.

Theorem 1 *(correctness)*
Let IDB be a locally stratified database, u be the update request insert(L) (resp. delete(L)). If $(\leftarrow L\ \{\})$ (resp. $(\leftarrow L^*\ \{\})$) has an abductive derivation to $(\square\ \Delta)$ then for any EDB, Tr_Δ(EDB) accomplishes u, i.e. L (resp. $\neg$L) is true in the stable model of IDB $\cup$ Tr(EDB).

For completeness, it is necessary to impose further restrictions on IDB. These restrictions are required to ensure that the abductive procedure does not loop indefinitly and does not flounder (see Lemma in appendix A). To do this, we define (appendix A) the properties of *acyclicity* and of *allowedness* for intensional databases, suitably adapting similar definitions for logic programs (see [Apt and Bezem 90] and [Topor 87]).

Theorem *(completeness)*
Let IDB be an acyclic and allowed intensional database, u be the update request insert(L) (resp. delete(L)). If there exists an extensional database EDB such that L (resp. $\neg$L) is true in the stable model of IDB$\cup$EDB then $(\leftarrow L\ \{\})$ (resp. $(\leftarrow L^*\ \{\})$ has an abductive derivation to $(\square\ \Delta)$ such that EDB $\models$ L' $(\neg$ L') for each base abducible $L'\in\Delta$ $(L'^*\in\Delta)$.

6. Extensions

An important feature of the particular approach to the update problem presented in this paper is the fact that this can be naturally extended in various directions. The semantics of stable models will allow a straightforward formalization of these extensions. In this section we will briefly discuss these different ways in which the update procedure can be extended. These are currently under investigation and will be presented elsewhere.

Let us first consider the case where no Δ is found in step A. This means that IDB can not support the update request no matter how EDB is changed. In order to satisfy the request we need to change the view IDB but this should not be done in an ad hoc way. The fact that such situations of no Δ arise means that some (perhaps all) of the rules in IDB are defeasible and/or some predicates in IDB are incompletely defined. Such incomplete knowledge representation is typical of the problems for which abduction has been developed in AI (see e.g. [Poole 88]), [Eshghi and Kowalski 89]). Abduction provides an effective way for handling such knowledge: every defeasible rule in IDB, $p(x)\leftarrow$ Conds, can be transformed into $p(x)\leftarrow$ Conds, $ab^*(x)$ in the abductive framework, where ab is a *new* abducible predicate and, similarly, for every incomplete predicate, p(x), a new rule $p(x)\leftarrow\delta_p(x)$ is added into the abductive framework where again δ_p is a *new* abducible predicate. Intuitively, ab* can be thought of as standing for "not abnormal" or "normal" and δ_p for some "unknown" cause for p. With these transformations the structure of the abductive framework and update procedure remains unchanged. By keeping in EDB information about these new abducibles we get the effect of changing the rules in IDB without altering the rules themselves, i.e. the rules are implicitly changed in such a way that any of these changes can be undone and the original IDB recovered. An appropriate priority ([Kakas and Mancarella 89]) can be given to the ab(x), $\delta_p(x)$ abducibles relative to the base predicate abducibles so that altering the rules is used only as a last resort, i.e. when no Δ with base predicates only exists. Let us reconsider example 3 where we are given that the first rule is defeasible:

example 3'
IDB*: $p\leftarrow B^*, ab^*$ EDB: B.
$p\leftarrow B.$
Update: delete(p)
Then step A of the update procedure will generate, in exactly the same way as in example 4, $\Delta=\{p^*, B^*, ab\}$ indicating that to delete p we must remove B and "switch off" the first rule by adding ab in the database.

In section 4, we briefly discussed how the abductive procedure can handle non ground hypotheses. When such hypotheses involve only base predicates a straightforward extension of the abductive procedure given in section 5 can be defined to achieve this [Kakas and Mancarella 90c]. For the case where the required hypotheses in Δ contain abducibles corresponding to negations of view predicates, eg. $\exists\ xq^*(x)$ where q is a view predicate, we can extend further this constructive abduction procedure with a form of constructive negation that follows from our treatment of negation through abduction. For an example consider:

example 8
IDB*: $p\leftarrow q^*(x), B_1(x)$
EDB: $B_1(a)$. $B_2(a)$
$q(x)\leftarrow B_2(x)$
$B_2(b)$.
Update: insert(p)
The query $\leftarrow$ p solved by this extension of the abductive procedure will give $\Delta=\{\exists x\ (B_1(x), q^*(x), B_2{}^*(x))\}$.
Possible transactions corresponding to this Δ are:
$Tr_1=\{$ remove$(B_2(a))$ $\}$
$Tr_2=\{$remove$(B_2(b))$, add$(B_1(b))$ $\}$
$Tr_3=\{$ add$(B_1(o))\}$ for a new object 'o'.

This extension would then allow us to remove the allowdeness condition and also enable us to handle general update requests, insert(ϕ) (delete(ϕ)), where ϕ is any first order formula. This would be done by first applying a set of transformations as in [Lloyd 87] to transform the rule $p \leftarrow \phi$ where p is a new predicate symbol into a logic program and then solving the request insert(p) (delete(p)).

Another important extension that can be naturally included in our approach is the dynamic integration of integrity checking of the possible solution to the update request against an integrity theory I^{DB} that might exist with the deductive database. This integrity checking can be included as part of the already existing consistency checking with IC* in the procedure. Now the consistency checking for any required abducible will be done with respect to $IC^* \cup I^{DB}$ rather then just IC*. An example of this extension is as follows:

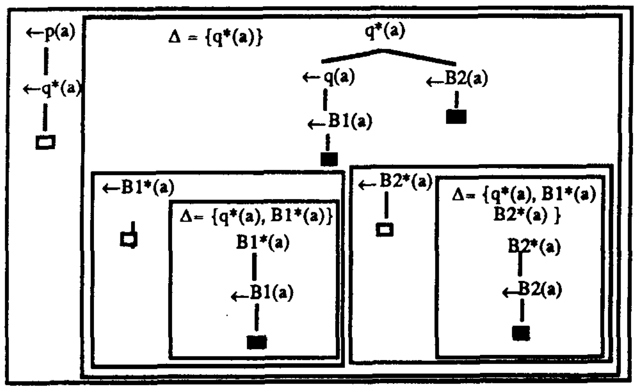
example 9

IDB*: $p(x) \leftarrow q^*(x)$
 EDB: $B_1(a)$. $B_2(a)$.
 $q(x) \leftarrow B_1(x)$
 $B_1(b)$.

I^{DB}: $\leftarrow q^*(x), B_2(x)$

Update: insert(p(a)).

The abductive search space for $\leftarrow$ p(a) is:



giving as a final Δ the set $\{q^*(a), B_1^*(a), B_2^*(a)\}$. Hence in step B apart from removing $B_1(a)$, which makes $\neg q(a)$ provable, we must also remove $B_2(a)$ to make $\neg q(a)$ consistent with I^{DB}. The semantics of stable models for logic programs P can be generalized to the case where P is replaced by an abductive framework <P,A,IC> (see [Kakas&Mancarella90]). This generalization when applied to the framework $<IDB^*, A, IC^* \cup I^{DB}>$ provides an appropriate semantics for the extended update procedure that incorporates the integrity checking of I^{DB} : when this procedure succeeds then there exists a model of DB (generalized stable model) where both the update request and I^{DB} are true.

The abductive approach for negation presented in [Eshghi and Kowalski 89] provides a way to generalize negation as failure to handle non-locally stratified programs such as

$p \leftarrow$ not q
$q \leftarrow$ not p.

and hence the abductive procedure of step A which is based on such a treatment for negation can also handle such programs. The added complication from such programs is that there exists more than one stable model, e.g. in this example there exists two stable models $M_1 = \{p\}$, $M_2 = \{q\}$. Hence in the example

IDB*: $p \leftarrow q^*, B$ EDB = ∅
 $q \leftarrow p^*$

Update: insert(p)

the generated $\Delta = \{q^*, B\}$ chooses the first stable model M_1 and this must be made explicit by recording the abducible q*. In effect, our update procedure can deal successfully with such non-locally stratified views, without any change in step A, by changing step B to allow a new type of transaction where negations of view predicates are explicitly added in the database. Technical results for this extended procedure would be expressed as before with the difference that any reference to "the stable model of DB" is replaced by "the existence of a stable model of DB".

7. Conclusions

The problem of view updates in deductive databases has been studied within an abductive approach. We have argued that this problem is naturally related to abduction and have used this relation to translate any given view update request to an equivalent update problem on the extensional (relational) part of the deductive database. A common single update procedure for both insert and delete requests is developed which handles successfully the complications arising from the presence of negations. An important feature of the approach based on abduction is the fact that this can be naturally extended along the lines discussed in section 6 in many desirable directions to give a more general solution to the problem.

Finally, in view of the fact that abduction can be used successfully for non-monotonic reasoning in a variety of problems (see e.g. [Poole88], [Eshghi and Kowalski 89] [Shanahan 89] [Kakas and Mancarella 89]) and the fact that the view update problem is closely related to abduction, an important message can be drawn: solving the view update problem (irrespective of the method of solution) will enhance the capabilities of deductive databases.

Acknowledgements

We wish to thank Robert Kowalski, Fariba Sadri and John Lloyd, Ahmed Guessoum Hendrik Decker, Francois

Bry, Luis Pereira for many useful discussions. This work has been supported under the COMPULOG Esprit Basic Research Project 3012.

APPENDIX A

In the following definitions HB(P) denotes the Herbrand base of a logic program P.

Definition *(level mapping)*
Let P be a logic program. A level mapping | | is a function | | : HB(P) $\rightarrow$ N. Given a level mapping | | its extension to ground negative literals is given by $|\neg A|=|A|$.

Definition *(acyclicity)* [Apt and Bezem 90]
A logic program P is ***acyclic*** if and only if there exists a level mapping | | such that for each clause A $\leftarrow$ $L_1,\ldots,L_n$ in ground(P) $|A|>|L_i|$ for each i=1,...,n.

Let us now define the notion of allowedness for intensional databases. In the following definitions by *local variable* we mean a variable which occurs in the body but not in the head of a clause.

Definition *(floundering)*
A derivation leading to a goal containing only non ground abducibles is said to flounder.

Definition *(allowedness)*
An intensional database IDB is ***allowed*** if for each clause H $\leftarrow$ W any variable occurring in an abducible literal of W also occurs in a non abducible literal of W.

Lemma
Let IDB be an allowed database and $\leftarrow$Q a ground goal. Then no abductive or consistency derivation resulting from this goal flounders.

References

[Abiteboul 88]
Abiteboul, S., Updates a new Frontier, Proc. ICDT, Springer Verlag, vol 326, 1988.
[Apt and Bezem 90]
Apt, K. and Bezem M., Acyclic Programs, to appear in Proc. 7th ICLP, Jerusalem 1990.
[Bancilhon and Spyratos 81]
Bancilhon, F. and Spyratos, N., Update semantics of relational views, ACM TODS, Vol 6, No4, 1981.
[Bry 89]
Bry, F., Intensional Updates: Abduction via Deduction, to appear in Proc. 7th ICLP, Jerusalem 1990.
[Cosmadakis and Papadimitriou 84]
Cosmadakis, C.C. and Papadimitriou, C.H., Updates of Relational Views, JACM 31 (4) 742-760, 1984.
[Dayal and Bernstein 82]
Dayal, U. and Bernstein P.A., On the correct translation of Update Operations on relatiuonal Views, ACM TODS, Vol 8, No 3, 1982.
[Decker 89]
Decker, H., Drawing Updates from Derivations, ECRC Technical Report, 1989.
[Eshghi and Kowlaski 89]
Eshghi, K. and Kowalski, R.A., Abduction Compared with Negation by Failure, Proc. 6th ICLP 89, MIT Press, 1989.
[Fagin et al. 83]
Fagin, R., Ullman, J. and Vardi, M., On the Semantics of Updates in Databases, Proc. 2nd ACM PODS, 1983.
[Furtado and Casanova 85]
Furtado, A.L. and Casanova, M.A., Updating Relational Views, in W. Kim et al (eds.), Query Processing in Database Systens, Springer Verlag, 1985.
[Gelfond and Lifschitz 88]
Gelfond, M. and Lifschitz, V., The Stable Model Semantics for Logic Programming, Proc. 5th ICLP, MIT Press, 1988.
[Gottlob et al 88]
Gottlob, G., Paolini, P., Zicari R., Properties and Update Sematics of Consistent Views, ACM TODS, Vol 13, No 4, 1988.
[Guessoum and Lloyd 90a]
Guessoum, A. and Lloyd, J.W., Updating Knowledge Bases, to appear in New Generation Computing 1990.
[Guessoum and Lloyd 90b]
Guessoum, A. and Lloyd, J.W., Updating Knowledge Bases II, Technical report, Department of Computer Science, University of Bristol, 1990.
[Kakas and Mancarella 89]
Kakas, A.C. and Mancarella, P., Anomalous Models and Abduction, Proc. 2nd Int. Symp. on AI, Monterrey, Mexico, 1989.
[Kakas and Mancarella 90a]
Kakas, A.C. and Mancarella, P., Generalised Stable Models: A Semantics for Abduction, to appear in ECAI-90, Stockholm, 1990.
[Kakas and Mancarella 90b]
Kakas, A.C. and Mancarella, P., On the relation between Truth Maintenance and Abduction, Imperial College preprint, 1990.
[Kakas and Mancarella 90c]
Kakas, A.C. and Mancarella, P., An Abductive Procedure for Database View Updates, Imperial College Report, 1990.
[Kowalski 79]
Kowalski, R.A. *Logic for Problem Solving*, Elsevier North-Holland, Amsterdam, 1979.
[Kunen 87]
Kunen, K., Negation in Logic Programming, in J. Logic Programming Vol 4, 1987.
[Lloyd 87]
Lloyd, J.W., Foundations of Logic Programming, second edition, Springer Verlag, 1987.
[Manchanda and Warren 88]

Manchanda, S. and Warren,D.S., Towards a Logical Theory of Database View Updates, in J. Minker (ed.) *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufman, 1988.

[Moore85]

Moore, R.C., Semantical Considerations on Nonmonotonic Logic, *Artificial Intelligence*, 25(1) (1985), pp.75-94.

[Nicolas and Yazdanian 83]

Nicolas, J.-M. and Yazdanian, K., An Outline of BDGEN: A Deductive DBMS, Proc. IFIP 83, Elsevier, 1983.

[Poole 88]

Poole, D.L., A Logical Framework for Default Reasoning, AI 36, 1988.

[Przymusinski 88]

Przymusinski, T., On the Declarative and Procedural Semantics of Stratified Deductive Databases, in J. Minker (ed.) *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufman, 1988.

[Ross 85]

Ross, B., View Updates on Deductive Databases, Department of Computer Science, University of Melbourne, 1985.

[Rossi and Naqvi 89]

Rossi, F. and Naqvi, S. A., Contributions to the View Update Problem, Proc. 6th ICLP 89, MIT Press, 1989.

[Sadri and Kowalski 88]

Sadri, F. and Kowalski, R.A., A Theorem Proving Approach to Database Integrity, in J. Minker (ed.) *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufman, 1988.

[Shanahan 89]

Shanahan, M.P., Prediction is Deduction but Explanation is Abduction, Proc. IJCAI 89, 1989.

[Tomasic 88]

Tomasic, A., View Update Annotation in Definite Deductive Databases, Proc. ICDT 88, Springer Verlag, 1988.

[Topor 87]

Topor, R.W., Domain Independent Formulas and Databases, TCS 52,3, 1987.