Journal of Automated Reasoning **11**: 249–267, 1993.

View Updates in Stratified Disjunctive Databases*

JOHN GRANT[2,4], JOHN HORTY,[2,3] JORGE LOBO,[5] and JACK MINKER[1,2]
[1] *Computer Science Department,* [2] *Institute for Advanced Computer Studies,* [3] *Philosophy Department, University of Maryland, College Park, MD 20742, U.S.A.*
[4] *Department of Computer and Information Sciences, Towson State University, Towson, MD 21204, U.S.A.*
[5] *Department of Electrical Engineering and Computer Science, The University of Illinois at Chicago, Chicago, IL 60680, U.S.A.*

(Received: 1 February 1993)

Abstract. The view update problem is considered in the context of deductive databases where the update of an intensional predicate is accomplished by modifying appropriately the underlying relations in the extensional database. Two classes of disjunctive databases are considered. The first class contains those disjunctive databases which allow only definite rules in the intensional database and disjunctive facts in the extensional database. The second class contains stratified disjunctive databases so that in addition to the first class, negation is allowed in the bodies of the rules, but the database must be stratified. Algorithms are given both for the insertion of an intensional predicate into and the deletion of an intensional predicate from the database. The algorithms use SLD resolution and the concept of minimal models of the extensional database. The algorithms are proved to be correct and best according to the criterion of causing minimal change to the database, where we give first priority to minimizing deletions.

Key words. Disjunctive deductive database, view update, stratified database.

1. Introduction

This paper is devoted to the problem of view updates in deductive databases. In this context, derived or *intensional* predicates correspond to the views of traditional relational databases. The view update problem is thus the problem of accomplishing the update of an intensional predicate by modifying appropriately the underlying relations in the *extensional* part of the database.

In the case of purely defined deductive databases, it is sometimes hard to see how the extensional relations should be modified to accomplish certain view updates. As an example, consider the simple database

$$P(x) \leftarrow A(x)$$
$$P(x) \leftarrow B(x),$$

where P is an intensional predicate and A and B are extensional. Suppose we want to update this database with the information that $P(c)$. If we are restricted to definite clauses, there are only three plausible ways to do this: add $A(c)$, add $B(c)$, or add both $A(c)$ and $B(c)$. Each of the first two options seems arbitrary; the third results

*Research supported by the National Science Foundation under grant numbers IRI-8916059, IRI-8921591, IRI-9200898, and IRI-9210220.

in an update that is too strong. If we allow disjunctive information into the database, however, we can accomplish the update by adding $A(c) \vee B(c)$; and this option appears intuitively to be correct.

We define here general algorithms for accomplishing view updates (both insertions and deletions) in disjunctive deductive databases; and we provide a semantic justification of the updates accomplished by these algorithms, making precise a sense in which they represent 'minimal modifications' of the underlying database. We consider two kinds of updates: those involving the insertion or deletion of information into and from a disjunctive deductive database, and those involving the insertion or deletion of information into and from a stratified normal disjunctive database. Insertions into normal deductive databases may require the insertion of negative information. (Of course, a database cannot contain negative information explicitly; but it can yield such information through nonmonotonic reasoning techniques, such as closed world reasoning [6, 7].)

There has recently been a good deal of work devoted to the view update problem in deductive databases. Fagin *et al.* [2] provide a semantic characterization of correctness that forms the basis of our own. That paper works with a fully expressive logical language, however, and so it cannot contain algorithms for computing the appropriate updates; in addition, it does not deal with negative information concluded through closed world reasoning. More recently, the problem has been investigated by Guessoum and Lloyd [4,5]; but they limit their treatment to definite databases, and so the algorithms they provide do not agree with the semantics of [2]. They also allow changes in the intensional database.

The work that is closest to our own is that of Rossi and Naqvi [8]. This work does deal with disjunctive information, and it provides algorithms; however, unlike the approach presented here, it does not insert the disjunctive information directly into the extensional database (which is supposed to remain definite), but keeps it aside in a filter that is then to be used in the process of query evaluation.

2. The Language

A deductive database, DB, is divided into an intensional component, I_{DB}, and an extensional component, E_{DB}. L is the background language of the database; the predicates occurring in L are partitioned into two sets: the L_I predicates are intensional, and the L_E predicates are extensional. Those predicates occurring in E_{DB} belong to L_E; the predicates occurring in the heads of I_{DB} rules belong to L_I. For simplicity, constants are substituted for variables in the database, so we can deal with ground atoms rather than predicates; these atoms are said to be intensional or extensional depending on the kind of predicate they contain. There are no denials or integrity constraints.

Updates must involve modifications of the extensional database only; the rules are not modified. Hence we assume, without further elaboration, that a view update to DB does not change I_{DB}. We also assume that E_{DB} does not contain redundant information, such as $A \leftarrow$ and $A \vee B \leftarrow$. Hence we include a subsumption elimination step in our algorithms. This is not a crucial point, however, and subsumption elimination may be omitted in all cases. Since in our case there are no variables or function symbols, subsumption checking involves only checking for substrings.

In Section 3 we focus on a special kind of disjunctive database: the intensional part consists of definite rules and the extensional part consist of positive clauses which may be disjunctive. The definition is given below. In Section 4 we extend Definition 2.1 to obtain stratified disjunctive databases.

DEFINITION 2.1. Let L be a function-free, first-order language. Let DB be a first-order theory in L. Then DB is called a *disjunctive database* iff:

1. $DB = E_{DB} \cup I_{DB}$ and $E_{DB} \cap I_{DB} = \emptyset$;
2. $C \in E_{DB}$ iff C is a positive ground clause;
3. $C \in I_{DB}$ iff C is a clause of the form $A \leftarrow B_1, \ldots, B_m$, with $A, B_1, \ldots, B_m$ ground atoms and $m > 0$;
4. The predicate symbols occurring in E_{DB} do not appear in the head of any rule in I_{DB}.

3. Updating Disjunctive Databases

This section describes techniques for updating disjunctive databases. Here, the algorithms for insertion and deletion are described in terms of proof trees constructed with SLD derivations.

3.1. INSERTIONS

We first define a special kind of SLD-tree, as follows.

DEFINITION 3.1. Let L be a function-free first order language. Let DB be a disjunctive database in L. Let P be an atom (in this case, P is ground) such that P is intensional. Assume R to be a computation rule that only selects atoms from L_I. A restricted *SLD-tree* for the goal $G = \leftarrow P$, given DB, is a tree satisfying the following conditions:

1. The root of the tree is G;
2. each node in the tree is a (possibly empty) goal;
3. a goal γ is a child of a node Λ if γ can be SLD-derived from Λ via R.

A restricted SLD-tree may contain infinite branches. However, since variables are not included in the goals, it is possible to check for repeated ancestors of a selected literal before expanding it.

EXAMPLE 3.1. Let DB be the disjunctive database:

$P \leftarrow A, B$

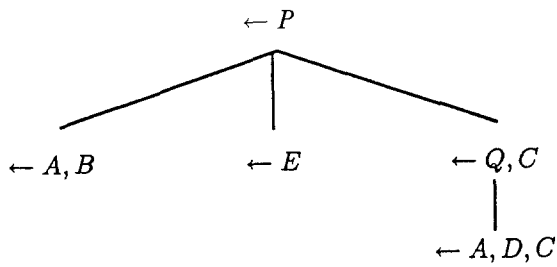
$P \leftarrow E$

$P \leftarrow Q, C$

$Q \leftarrow A, D$

$A \vee B \vee C \leftarrow$

Let $\{A, B, C, D, E\}$ be the set of extensional atoms and $\{P, Q\}$ be the set of intensional atoms. The restricted SLD-tree for $\leftarrow P$ in DB is the following:



A statement P is a logical consequence of DB if any of the conjunctions of atoms in the leaves in a restricted SLD-tree for the goal $G = \leftarrow P$ is *true* in DB. Take, for example the leftmost leaf in Example 3.1. If A and B are true in DB, then P is also true in DB. Since it is desirable to 'minimize' the changes to DB, the insertion algorithm needs to consider the weakest formula that achieves the insertion. In the example, this formula is given by the disjunction of these leaves: $(A \wedge B) \vee (E) \vee (A \wedge D \wedge C)$.

Since we are limited to modifying only the extensional part of the disjunctive database, there are some updates that are not possible.

EXAMPLE 3.2. Let DB be the disjunctive database:

$P \leftarrow Q, C$

Let $\{C\}$ be the set of extensional atoms and $\{P, Q\}$ the set of intensional atoms. The insertion of P into DB requires the insertion of Q. The only possible modification to the extensional part of DB is to insert C. There is no way to insert Q.

DEFINITION 3.2. Let DB be a disjunctive database such that $DB \nvdash P$. An insertion of an atom P is a *possible insertion* into DB iff there exists a disjunctive database DB' such that $DB' \vdash P$.

Algorithm 1 (Insertion of an intensional atom P into a disjunctive database – Insertion 1). Given an atom P and a disjunctive database DB such that $DB \nvdash P$, the

algorithm computes a database DB' such that $DB' \vdash DB \wedge P$ whenever the insertion is possible.

1. Let $DB' = DB$.
2. Construct a restricted SLD-tree for $\leftarrow P$ from DB.
3. Let $\leftarrow F_1, \ldots, \leftarrow F_n$ be the leaf nodes containing only conjunctions of extensional atoms. If $n = 0$ the insertion fails. Otherwise, construct the conjunctive normal form of $F_1 \vee \cdots \vee F_n$, writing it as $C_1 \wedge \cdots \wedge C_m$, where subsumed clauses are omitted from the conjunctive normal form.
4. For each C_i, $1 \leq i \leq m$, insert C_i into DB'.
5. Delete all subsumed clauses from DB'.

EXAMPLE 3.3. From the restricted SLD-tree of Example 3.1. we obtain $F_1 = A \wedge B$, $F_2 = E$ and $F_3 = A \wedge D \wedge C$. Then $DB \vdash P$ if $DB \vdash F_1 \vee F_2 \vee F_3$. The conjunctive form of $F_1 \vee F_2 \vee F_3$ is $C_1 \wedge C_2 \wedge C_3$, where $C_1 = A \vee E$, $C_2 = B \vee D \vee E$, $C_3 = B \vee C \vee E$. Then, the updated database DB' is:

$$P \leftarrow A, B$$

$$P \leftarrow E$$

$$P \leftarrow Q, C$$

$$Q \leftarrow A, D$$

$$A \vee B \vee C \leftarrow$$

$$A \vee E \leftarrow$$

$$B \vee C \vee E \leftarrow$$

$$B \vee D \vee E \leftarrow$$

In order to justify this algorithm, we must show first that it actually accomplishes the desired insertion.

LEMMA 3.1. *Let DB be a disjunctive database in L. Let P be an atom in L_I and assume that the insertion of P into DB is possible, and $DB \not\vdash P$. Then, if DB' is the deductive database constructed by Algorithm* 1, $DB' \vdash DB \wedge P$.

Proof. $DB' \vdash DB$ follows from Steps 1, 4, and 5. Now, let M be a model of DB'. By construction, each C_i, $1 \leq i \leq m$, as developed by Algorithm 1, is *true* in M. Hence $F_1 \vee \cdots \vee F_n$ is *true* in M. By the soundness of SLD-derivations and the intensional parts of rules of DB', P is *true* in M. ■

But, of course, this is not enough; in addition, we must require that the insertion should modify the original database as little as possible, where this can be defined as follows.

DEFINITION 3.3. Let DB be a disjunctive database in L. Let P be an atom in L_I and assume $DB \not\vdash P$. A *minimal insertion* of P into DB is a minimal DB' such that

$DB' \vdash DB \wedge P$. That is, there is no database DB'' such that $DB' \vdash DB'' \vdash DB \wedge P$ and $DB'' \neq DB'$.

As the following lemma shows, Algorithm 1 does in fact produce a minimal insertion.

LEMMA 3.2. *Let DB be a disjunctive database in L. Let P be an atom in L_I and assume that the insertion of P into DB is possible, and $DB \nvdash P$. Let DB' be the deductive database that is obtained from Algorithm* 1 *after inserting P. If $DB' \vdash DB'' \vdash DB$ and $DB'' \nvdash DB'$ then $DB'' \nvdash P$.*

Proof. By construction, $DB' = DB \cup \{C_i \in \{C_1, \ldots, C_m\} | DB \nvdash C_i\}$, where the C_i are obtained in step 4 of Algorithm 1. Therefore, since $DB'' \nvdash DB'$ and $DB'' \vdash DB$, $\exists i$, $1 \leq i \leq m$, such that $DB'' \nvdash C_i$. Hence, $DB'' \nvdash F_1 \vee \cdots \vee F_n$. Hence, $DB'' \nvdash P$. ■

Unfortunately, however, the minimality condition alone does not imply uniqueness, and so it cannot be used as a criterion to justify the insertion algorithm. This should be obvious already from the example used in the introduction; there, the addition of either $A(c)$ or $B(c)$ yields a minimal modification of the database. In order to force a unique result, we require not only that the update should be minimal, but also that it must yield the weakest modification of the database, in the following sense.

DEFINITION 3.4. Let DB_1 and DB_2 be disjunctive databases. DB_1 is weaker than DB_2 iff $DB_2 \vdash DB_1$. Let $\mathscr{DB}$ be a set of disjunctive databases. DB is the *weakest* disjunctive database in $\mathscr{DB}$ iff for any other disjunctive database DB' in $\mathscr{DB}$, DB is weaker than DB'.

The following theorem shows that Algorithm 1 constructs the weakest of the minimal disjunctive databases that accomplish the desired update.

THEOREM 3.3. *Let DB be a disjunctive database in L. Let P be an atom in L_I and assume that the insertion of P into DB is possible, and $DB \nvdash P$. Let DB' be the disjunctive database that is obtained from Algorithm* 1 *after inserting P. Then DB' is the weakest disjunctive database DB'' such that $DB'' \vdash DB \wedge P$.*

Proof. Let $DB'' \vdash DB \wedge P$. If $F \in DB$ then $DB'' \vdash F$. Now suppose that $F \in (DB' - DB)$. Then $F = C_i$, for some i, $1 \leq i \leq m$, where C_i is a clause formed by Algorithm 1. Since $DB'' \vdash P$, by the construction of the C_is, and the completeness of SLD-resolution, $DB'' \vdash C_i$. ■

Another interesting characterization of the weakest minimal insertion is given by the disjunction of all minimal insertions.

DEFINITION 3.5. Let $DB_1, \ldots, DB_n$ be disjunctive databases with identical intensional portion I_{DB}. Then

$$\bigvee_{i=1}^{n} DB_i = \{C_1 \vee \cdots \vee C_n | C_i \in E_{DB_i},\ 1 \leq i \leq n\} \cup I_{DB}.$$

THEOREM 3.4. *Let DB be a disjunctive database in L. Let P be an atom in L_I and assume that the insertion of P into DB is possible, and $DB \nvdash P$. Let DB' be the disjunctive database that is obtained from Algorithm 1 after inserting P. Let $DB_1, \ldots, DB_n$ be all the minimal updates that insert P into DB. Then DB' is logically equivalent to $\vee_{i=1}^{n} DB_i$.*

Proof. Since DB' is the weakest insertion, $DB_i \vdash DB'$, therefore, $\vee_{i=1}^{n} DB_i \vdash DB'$. Conversely, since DB' is a minimal insertion, DB' is one of the DB_i. Therefore, $DB' \vdash \vee_{i=1}^{n} DB_i$. ■

This final characterization of the appropriate result of insertion updates can be derived easily from that of Fagin *et al.* [2]; Theorem 3.3 thus shows how the characterization can be reached through a slightly different route.

3.2. DELETIONS

We now present the algorithm for updating disjunctive databases by deleting information from the database. We note that in a definite database when an intensional atom P is deleted, $\neg P$ becomes *true* in the updated database. The reason for this is that a definite database has a unique minimal model where either P or $\neg P$ is *true*. However, a disjunctive database may have several minimal models. The deletion of P may be accomplished by deleting P from some but not all of the minimal models. Hence the deletion of P does not necessarily make $\neg P$ *true*.

DEFINITION 3.6. Let DB be a disjunctive database such that $DB \vdash P$. A deletion of an atom P from DB results in a disjunctive database DB' such that $DB' \nvdash P$.

Note that the deletion of an atom, unlike insertion, is always possible.

Algorithm 2 (Deletion of an intensional atom P from a disjunctive database – Deletion 1). Given an atom P and a disjunctive database DB such that $DB \vdash P$. The algorithm computes a database DB'' such that $DB'' \nvdash P$.

1. Construct a restricted SLD-tree for $\leftarrow P$ from DB.
2. Let $\leftarrow F_1, \ldots, \leftarrow F_n$ be the leaf nodes containing only conjunctions of extensional atoms. Construct the conjunctive normal form of $F_1 \vee \cdots \vee F_n$, writing it as $C_1 \wedge \cdots \wedge C_m$, where subsumed clauses are omitted in the conjunctive normal form.
3. Let S_i be the sets of clauses in DB which subsume C_i, $1 \leq i \leq m$. For each j, $1 \leq j \leq m$ such that there is no i, $i \neq j$ and $1 \leq i \leq m$, such that $S_i \subset S_j$, obtain $DB_j = (DB - S_j)$. Without loss of generality assume that the remaining DB_j are $DB_1, \ldots, DB_k$.
4. Let DB'' be $\vee_{j=1}^{j=k} DB_j$.
5. Delete all subsumed clauses from DB''.

EXAMPLE 3.4. Let DB be the disjunctive database:

$P \leftarrow A, B$

$P \leftarrow E$

$P \leftarrow Q, C$

$Q \leftarrow A, D$

$A \leftarrow$

$B \vee D \leftarrow$

$B \vee E \leftarrow$

I_{DB} is the same as in Example 3.1. Hence the restricted SLD-tree for $\leftarrow P$ is the same, so $C_1 = A \vee E$, $C_2 = B \vee D \vee E$, $C_3 = B \vee C \vee E$. Hence, $S_1 = \{A\}$, $S_2 = \{B \vee D, B \vee E\}$, $S_3 = \{B \vee E\}$. Since $S_3 \subseteq S_2$, S_2 is eliminated, leaving S_1 and S_3. From step 3 we obtain the disjunctive databases $DB_1 = I_{DB} \cup \{B \vee D, B \vee E\}$ and $DB_3 = I_{DB} \cup \{A, B \vee D\}$. Then, DB'' is:

$I_{DB} \cup$

$\{B \vee D \leftarrow$

$A \vee B \vee E \leftarrow\}$.

The following lemma shows that Algorithm 2 accomplishes the desired deletion.

LEMMA 3.5. *Let DB be a deductive database in L. Let P be an atom in L_I and assume $DB \vdash P$. Then the deductive database DB'' constructed by Algorithm 2 does not imply P. That is, $DB'' \nvdash P$.*

Proof. By construction and the soundness and completeness of SLD-resolution for all i, $1 \leq i \leq m$, $(DB - S_i) \nvdash C_i$, where C_i is obtained from Algorithm 2. Therefore $(DB - S_i) \nvdash P$. Hence $DB'' \nvdash P$. ■

In the previous case of inserting information we required that the update should modify the original database as little as possible; this led us to present in Definition 3.3 the concept of a minimal insertion. In the case of deletions, the counterpart idea of a minimal modification can be defined as a maximal subset of the original database that accomplishes the desired deletion.

DEFINITION 3.7. Let DB be a disjunctive database in L. Let P be an atom in L_I and assume $DB \vdash P$. A *minimal deletion* of P from DB is a maximal subset DB_i of DB that does not imply P. That is, $DB_i \subseteq DB$, $DB_i \nvdash P$ and for any other disjunctive database $DB' \subseteq DB$ such that $DB_i \subset DB'$, $DB' \vdash P$.

Again, however, there may be several such minimal deletions; and so, in accordance with the views of Fagin *et al.* [2], it seems that a semantically correct algorithm should yield a result equivalent to their disjunction. The following theorem shows that Algorithm 2 is correct in this sense.

THEOREM 3.6. *Let DB be a disjunctive database in L. Let P be an atom in* L_I *and assume that* $DB \vdash P$. *Let* DB'' *be the disjunctive database that is obtained from Algorithm 2 by deleting P. Let* $DB_1, \ldots, DB_n$ *be all the minimal updates that delete P from DB. Then* DB'' *is logically equivalent to* $\vee_{i=1}^{n} DB_i$.

Proof. The proof follows from the observation that the minimal deletions of P are precisely the $(DB - S_i)$ obtained in step 3 of Algorithm 2. ■

As it turns out, there is an interesting asymmetry between insertion and deletion. In the case of insertion, the disjunction of minimal insertions is actually equivalent to a particular one of the minimal insertions; but in the case of deletion, the disjunction of minimal deletions need not itself lie among the minimal deletions.

A peculiar property of the deletion algorithm is that logically equivalent databases that are syntactically different can be transformed into non-equivalent databases after the deletion of an atom from the database. Consider, for example the disjunctive database DB in Example 3.4. Extend DB with the extensional fact $A \vee B$. The new disjunctive database is equivalent to DB since A subsumes $A \vee B$. The deletion of P from this new database using Algorithm 2 results in the disjunctive database:

$$DB' = I_{DB} \cup \{B \vee D;\ A \vee B\}.$$

DB'' from Example 3.4 is weaker than this database. The minimality condition does not guarantee the preservation of equivalence after updating equivalent databases. In the case of insertion, in addition to minimality, the weakness condition was used to select the best update. The weakest insertion was the insertion selected. A similar condition can be imposed on deletion. We call it the strongness condition.

DEFINITION 3.8. Let DB_1 and DB_2 be disjunctive databases. DB_1 *is stronger than* DB_2 iff $DB_1 \vdash DB_2$. Let $\mathscr{DB}$ be a set of disjunctive databases. DB is the *strongest disjunctive database* in $\mathscr{DB}$ iff for any other disjunctive deductive database DB' in $\mathscr{DB}$, DB is stronger than DB'.

In general, there is no strongest disjunctive database that does not imply an atom P and is weaker than DB. Moreover, there is not a direct correspondence between minimal and strong deletions. The following algorithm non-deterministically selects a disjunctive database DB' that: (1) accomplishes the deletion, (2) is weaker than DB, and (3) is stronger than or incomparable with any other disjunctive database containing I_{DB} that does not imply P.

Algorithm 3 (Deletion of an intensional atom P from a disjunctive database – Deletion 2). Given an atom P and a disjunctive database DB such that $DB \vdash P$. The algorithm computes a database DB' such that $DB' \nvdash P$.

1. Construct a restricted SLD-tree for $\leftarrow P$ from DB.
2. Let $\leftarrow F_1, \ldots, \leftarrow F_n$ be the leaf nodes containing only conjunctions of extensional atoms. Construct the conjunctive normal form of $F_1 \vee \cdots \vee F_n$, writing it as $C_1 \wedge \cdots \wedge C_m$ where subsumed clauses are omitted.
3. Let S_i be the sets of clauses in DB which subsume C_i, $1 \leq i \leq m$. Choose j, $1 \leq j \leq m$ such that there is no i, $i \neq j$ and $1 \leq i \leq m$, $S_i \subset S_j$. Delete the set S_j from DB.
4. For each clause $C \in S_J$, form all disjunctions $C \vee Q$ where Q is an extensional atom that does not appear in C.
5. Add to DB all clauses $C \vee Q$ which do not subsume C_j. Call the resulting database DB'.

EXAMPLE 3.5. Using the same disjunctive deductive database as in Example 3.4, there are two answers for the deletion of P using Algorithm 3 depending on the choice of S_1 or S_3 (see Example 3.4). These are (omitting the intensional part which is the same in all cases):

(1) $\{A \vee B;$
$A \vee C;$
$A \vee D;$
$B \vee D;$
$B \vee E\}.$

(2) $\{A;$
$B \vee D\}.$

■

The following lemma shows that Algorithm 3 accomplishes the deletion.

LEMMA 3.7. *Let DB be a disjunctive deductive database in L. Let P be an intensional atom in L_I. Assume that $DB \vdash P$. Then the deductive database DB', constructed by Algorithm 3, does not imply P. That is, $DB' \nvdash P$.*

Proof. By construction, $DB' \nvdash C_j$, where C_j is derived in Algorithm 3. Therefore, $DB' \nvdash C_1 \wedge \cdots \wedge C_m$ and hence $DB' \nvdash P$. ■

Theorem 3.8 shows that there is no stronger deductive database than DB' that deletes P.

THEOREM 3.8. *Let DB be a disjunctive deductive database in L. Let P be an intensional atom in L_I. Assume $DB \nvdash P$. Let DB' be the deductive database that is obtained from Algorithm 3 after deleting P. Then there is no deductive database, DB'', stronger than DB' and weaker than DB such that $DB'' \nvdash P$.*

Proof. Since the clauses added to the database in step 5 are logically implied by DB, $DB \vdash DB'$. Now suppose that $DB \vdash DB'' \vdash DB'$ and assume $DB' \nvdash DB''$. Let $C_1 \wedge \cdots \wedge C_m$ be the conjunctive formula obtained in step 2 of Algorithm 3. Let S_j be the set selected in step 3 for deletion. Since for all i, $i \neq j$, $DB' \vdash C_i$, then

$DB'' \vdash C_i$, where C_i is derived in Algorithm 3. But since $DB \vdash DB''$ and $DB' \nvdash DB''$, by the construction, $DB'' \vdash C_j$, so $DB'' \vdash P$. ■

If we prefer to have a unique disjunctive database after a deletion as in the case of Algorithm 2, we can take the disjunction of all possible disjunctive databases obtained with Algorithm 3. This database is stronger than the disjunction of the minimal updated databases. A disadvantage of strong databases over minimal databases is that they are language dependent. New disjunctions are formed in step 4, to be included in the updated database; one disjunction for each extensional atom in the underlying language of the database. So the algorithm produces non-equivalent databases over different languages from two logically equivalent databases

4. Updating Stratified Disjunctive Databases

4.1. NORMAL INSERTIONS

In this section we present an algorithm to insert information into a subclass of normal disjunctive databases. A normal disjunctive database, DB, is a disjunctive database in which the clauses in the I_{DB} can be of the form $A \leftarrow L_1, \ldots, L_m$ with A a ground atom, $L_1, \ldots, L_m$ ground literals (i.e. atoms and negated atoms) and $m \geq 0$. We consider insertions into normal databases that are stratified. The definitions of stratified databases, stratification, and the stratum of an atom are as defined in [1].

The insertion of an atom into a normal disjunctive database may require the 'insertion' of negative information into the database. Consider, for example, a database DB with $I_{DB} = \{P(x) \leftarrow \neg A(x)\}$. Assume that A is an extensional predicate. The insertion of $P(a)$ into DB must modify DB into a new database DB' such that the new database implies $\neg A(a)$. Also, because of this duality between the insertion of atoms and negated atoms into a normal disjunctive database, the algorithm below can be used for both kinds of insertions, positive and negative.

Negative information can be derived from databases using nonmonotonic reasoning techniques, such as closed world reasoning. Because we are working with disjunctive databases, we use the generalized closed world assumption (GCWA), devised by Minker [6]. According to this rule, a ground formula, $\neg F$, is derivable from a disjunctive database, DB, if F is false in all minimal models of that database. In that case, we say $GCWA(DB) \vdash \neg F$.

For the insertion algorithm into stratified databases we need to extend the definition of restricted SLD-trees to cover negation. The new trees will be called restricted stratified SLD-trees. We use normal goals where literals of the form $\neg C$ may appear. Restricted stratified SLD-trees are defined inductively on the stratum of the atom associated with the tree.

DEFINITION 4.1. Let DB be a stratified disjunctive database in L. Let P be an atom (in this case, P is ground) such that P is intensional. Assume R to be a computation rule that only selects literals with atoms from L_I. A restricted stratified SLD-tree for the goal $G = \leftarrow P$ given DB is a restricted SLD-tree for $\leftarrow P$ if the stratum of P is 1. Otherwise, assume all restricted stratified SLD-trees for atoms in a stratum less than n are defined and let the stratum of P be n. A restricted stratified SLD-tree for $\leftarrow P$ is a tree satisfying the following conditions:

1. The root of the tree is G.
2. Each node in the tree is a (possibly empty) goal.
3. Let L_i be the selected literal in the node $\Lambda = \leftarrow L_1, \ldots, L_{i-1}, L_i, L_{i+1}, \ldots, L_k$.
   (a) If L_i is positive, a normal goal γ is a child of Λ if γ is SLD-derived from Λ via R.
   (b) If $L_i = \neg Q$ is negative, let $\leftarrow F_1, \ldots, \leftarrow F_n$ be the leaf nodes in the restricted stratified SLD-tree for $\leftarrow Q$. Construct the conjunctive normal form of $F_1 \vee \cdots \vee F_n$, writing it as $C_1 \wedge \cdots \wedge C_m$. A normal goal $\leftarrow L_1, \ldots, L_{i-1}, \gamma, L_{i+1}, \ldots, L_k$ is a child of Λ iff γ is the conjunction of literals obtained from $\neg C_i$, for some C_i, $1 \leq i \leq m$.

EXAMPLE 4.1. Let DB be the stratified disjunctive database:

$Q \leftarrow A, \neg R$

$R \leftarrow B, C$

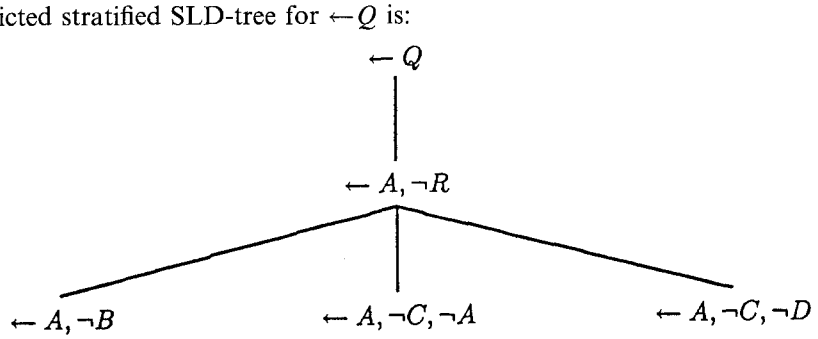
$R \leftarrow A, S$

$S \leftarrow B, D$

$A \vee C \leftarrow$

$A \vee D \leftarrow$

The conjunctive normal form obtained from the restricted stratified SLD-tree for $\leftarrow R$ (in this case just restricted SLD-tree) is $B \wedge (C \vee A) \wedge (C \vee D)$. Therefore, the restricted stratified SLD-tree for $\leftarrow Q$ is:



Algorithm 4 (Insertion of an intensional atom P into a stratified disjunctive database – Insertion 2). Suppose that P is an intensional atom, and let DB be a stratified disjunctive database such that $GCWA(DB) \not\vdash P$. The algorithm computes a database DB' such that $GCWA(DB') \vdash P$.

1. Let $\mathcal{M}$ be the minimal models of E_{DB}, the extensional component of DB.
2. Construct a restricted stratified SLD-tree for $\leftarrow P$ from DB.
3. Let $\leftarrow F_1, \ldots, \leftarrow F_n$ be the leaf nodes containing only conjunctions of (possibly negated) extensional atoms. If $n = 0$ the insertion fails. Otherwise, construct the conjunctive normal form of $F_1 \vee \cdots \vee F_n$, writing it as $C = C_1 \wedge \cdots \wedge C_m$, where subsumed clauses are omitted from the conjunctive normal form.
4. If $E_{DB} \cup \{C\}$ is consistent then order C by placing positive clauses first and negative clauses last. For each C_j, $1 \leq j \leq m$, do
   (a) If C_j is a positive clause, $C_j = A_1 \vee \cdots \vee A_l$ then:
   For each $M \in \mathcal{M}$ such that for all A_i, $1 \leq i \leq l$, $A_i \notin M$ let $M_i = M \cup \{A_i\}$, $1 \leq i \leq l$, and let $\mathcal{M} = (\mathcal{M} - \{M\}) \cup \{M_i | 1 \leq i \leq l\}$.
   Eliminate from $\mathcal{M}$ every M for which there exists $M' \in \mathcal{M}$, such that $M' \subset M$.
   (b) If C_j is a mixed clause, $C_j = A_1 \vee \cdots \vee A_l \vee \neg E_1 \vee \cdots \vee \neg E_s$, then:
   For each $M \in \mathcal{M}$ such that for all A_i, $1 \leq i \leq l$, $A_i \notin M$ and for all E_k, $1 \leq k \leq s$, $E_k \in M$ let $M_i = M \cup \{A_i\}$, $1 \leq i \leq l$, and let $\mathcal{M} = (\mathcal{M} - \{M\}) \cup \{M_i | 1 \leq i \leq l\}$.
   Eliminate from $\mathcal{M}$ every M for which there exists $M' \in \mathcal{M}$, such that $M' \subset M$.
   This step must be repeated until there is a complete pass through all the mixed clauses which causes no change to $\mathcal{M}$.
   (c) If C_j is a negative clause, $C_j = \neg E_1 \vee \cdots \vee \neg E_l$ then:
   For each $M \in \mathcal{M}$ such that for all E_i, $1 \leq i \leq l$, $E_i \in M$ eliminate M from $\mathcal{M}$.
5. Build DB' from the new $\mathcal{M}$.
   (*Comment.* If the DB is represented by using model trees as discussed in [3], it is not necessary to reconstruct all the clauses in $E_{DB'}$).
6. Else ($E_{DB} \cup \{C\}$ is inconsistent)
   (a) Construct all DB_i such that each E_{DB_i} is a maximal subset of $\mathrm{Cn}(E_{DB})$ (the set of positive logical consequences of E_{DB}) with the property that $E_{DB_i} \cup \{C\}$ is consistent. (This can be done by a process where first single elements of $\mathrm{Cn}(E_{DB})$ are omitted to obtain E_{DB_j} and the consistency of $E_{DB_j} \cup \{C\}$ is checked; when the latter is inconsistent, the process is iterated with E_{DB_j} substituted for $\mathrm{Cn}(E_{DB})$; the maximal sets E_{DB_i} obtained this way are retained. The consistency of $E_{DB_j} \cup \{C\}$ can be checked by doing all possible resolutions using the elements of E_{DB_j} and C_i [the clauses in C]; the empty clause indicates inconsistency.) At the end, subsumed clauses are omitted from DB_i. Assume without loss of generality that these are $DB_1, \ldots, DB_k$.
   (b) For each DB_i, $1 \leq i \leq k$, apply steps 4 and 5 to obtain DB_i'.

(c) Let $DB' = \vee_{i=1}^{k} DB_i'$.

The next example illustrates the case where DB' is obtained using steps 4 and 5.

EXAMPLE 4.2. Let DB be the stratified disjunctive database:

$P \leftarrow \neg A, C, R$

$P \leftarrow \neg D, R$

$R \leftarrow B$

$R \leftarrow E$

$A \vee B \leftarrow$

$A \vee D \leftarrow$

$C \vee D \leftarrow$

$C \vee E \leftarrow$

The clauses obtained by step 3 after the construction of the restricted stratified SLD-tree for $\leftarrow P$ are C_1: $\neg A \vee \neg D$, $C_2 : C \vee \neg D$, $C_3 : B \vee E$.

The minimal models of the extensional database of DB are $MM(E_{DB}) = \{\{A, C\}, \{A, D, E\}, \{B, D, E\}, \{B, C, D\}\}$. In this case E_{DB} is consistent with the C_is. The algorithm first modifies $MM(E_{DB})$ to make C_3 true. This produces the new set of minimal models $\{\{A, B, C\}, \{A, C, E\}, \{A, D, E\}, \{B, D, E\}, \{B, C, D\}\}$. Next, the set of minimal models is modified to make C_2, true. The algorithm produces the set $\{\{A, B, C\}, \{A, C, E\}, \{A, C, D, E\}, \{B, C, D, E\}, \{B, C, D\}\}$. The models $\{A, C, D, E\}$ and $\{B, C, D, E\}$ are removed since they are not minimal. Finally, C_1 is true in the three remaining models. Then, the minimal models of the new extensional database are $MM(E_{DB'}) = \{\{A, B, C\}, \{A, C, E\}, \{B, C, D\}\}$. Building the extensional database from these models we obtain:

$A \vee B \leftarrow$

$A \vee D \leftarrow$

$C \leftarrow$

$B \vee E \leftarrow$

The following example illustrates the case where step 6 must be used, that is, where $E_{DB} \cup \{C\}$ is inconsistent.

EXAMPLE 4.3. Let DB be the stratified disjunctive database:

$P \leftarrow R, \neg C, \neg D$

$P \leftarrow A, \neg B, \neg D$

$P \leftarrow B, \neg C, \neg A$

$R \leftarrow A$

$R \leftarrow B$

$S \leftarrow B, D$

$A \vee B \leftarrow$

$A \vee C \leftarrow$

$D \leftarrow$

The conjunction of clauses obtained by step 3 after the construction of the restricted stratified SLD-tree for $\leftarrow P$ is $(A \vee B) \wedge (\neg B \vee \neg C) \wedge (\neg A \vee \neg D)$. This conjunct is inconsistent with E_{DB}. E_{DB} has two minimal models $\{A, D\}$ and $\{B, C, D\}$. There are three maximal subsets of $\mathrm{Cn}(E_{DB})$ consistent with the conjunction $C_1 \wedge C_2 \wedge C_3 : E_{DB_1} = \{A \vee B,\ A \vee C,\ A \vee D,\ B \vee D\}$, $E_{DB_2} = \{A \vee B,\ A \vee C,\ A \vee D,\ C \vee D\}$, and $E_{DB_3} = \{A \vee B, D\}$. Applying the insertion to each of these databases we obtain $E'_{DB_1} = \{A, B\}$, $E'_{DB_2} = \{A, C\}$ and $E'_{DB_3} = \{B, D\}$. The disjunction of these three new databases together with the rules produces $DB' = I_{DB} \cup \{A \vee B, A \vee D, B \vee C\}$.

The simple characterization of minimality from the treatment of positive updates does not apply directly to this more complicated context, where nonmonotonic reasoning is involved and insertions and deletions from and into the extensional database may be needed to accomplish the insertion of an intensional atom. However, there is a sense in which the insertion defined by Algorithm 4 is minimal. The new database first minimizes the deletion of positive conclusions from the original database and then, subject to that constraint, minimizes the addition of new positive conclusions. That is, insertions are preferred over deletions. As it turns out, this condition is enough to guarantee uniqueness. The following theorems characterize this property. The first theorem shows the minimality of DB' with respect to DB and C.

THEOREM 4.1. *Let C be the conjunction of the clauses obtained in step 3 of Algorithm 4 and assume that $E_{DB} \cup \{C\}$ is consistent. Then, for any positive clause D, $DB' \vdash D$ iff $DB \cup \{C\} \vdash D$, where DB' is obtained in step 5 of Algorithm 4.*

Proof. Since, by the construction of DB', every (minimal) model of DB' is a model of $DB \cup \{C\}$, it follows that $DB \cup \{C\} \vdash D$ implies that D is true in every model of DB'. Therefore $DB' \vdash D$. To prove the other direction by the contrapositive, let $DB \cup \{C\} \nvdash D$. Then there must be a minimal model of $DB \cup \{C\}$, say $M = \{A_1, \ldots, A_k\}$, where D is false. Let D be $A_{k+1} \vee \cdots \vee A_n$. M must be a model of DB. Since M is also a model of C, by the construction of DB', if M is a minimal model of DB then M remains a minimal model of DB'. Hence, $DB' \nvdash D$. Otherwise M is a non-minimal model of DB. Therefore, there is $M' \subset M$, say $M' = \{A_1, \ldots, A_i\}$, such that M' is minimal model of DB. M' is not a model of $DB \cup \{C\}$ because it is a proper subset of a minimal model of $DB \cup \{C\}$. Consider what happens to M' in step 4 of Algorithm 4. For any C_j, part of C, that is true in M'

the algorithm does nothing. Consider now some C_j false in M'. If C_j is positive or mixed then M' is expanded in a minimal way to satisfy C_j. The expansion, $M'' \subseteq M$, since M is a minimal model of DB. Hence M'' cannot be a model of D. In the case of negative C_j, since M is a model of C_j, M' must already be a model of C_j. Therefore M (or a subset of M) is a minimal model in $\mathcal{M}$ from which DB' is built. Hence, $DB' \not\vdash D$. ■

Theorem 4.1 suggests an alternative approach to obtaining the result of step 4 since the algorithm obtains the logical consequences of $E_{DB} \cup \{C\}$: do all possible resolutions using elements of E_{DB} and C_i, $1 \leq i \leq m$. Obtain the set of all positive clauses at the bottom of the resolution trees and omit subsumed clauses from the set. The result is $E_{DB'}$. This result shows an interesting connection between resolution and minimal model manipulation.

The following lemma is a technical result, needed later to characterize the minimal models of $\vee_{i=1}^{k} DB_i$.

LEMMA 4.2. *Let $DB = \vee_{i=1}^{k} DB_i$, $\mathcal{M}$ the set of minimal models of DB, and $\mathcal{M}_i$ the set of minimal models of DB_i, $1 \leq i \leq k$. Then $M \in \mathcal{M}$ iff $M \in \mathcal{M}_i$ for some i, $1 \leq i \leq k$, and there is no $M' \in \mathcal{M}_j$ for some j, $1 \leq j \leq k$, such that $M' \subset M$.*

Proof. ($\Leftarrow$) Suppose that $M \in \mathcal{M}_i$ for some i, $1 \leq i \leq k$, and there is no $M' \in \mathcal{M}_j$ for some j, $1 \leq j \leq k$, such that $M' \subset M$. Clearly, M is a model of DB. If M is not a minimal model of DB then there is an $M' \subset M$, a model of DB which is minimal. This would require that $M' \in \mathcal{M}_j$ for some j, $1 \leq j \leq k$ contradicting the hypothesis.

($\Rightarrow$) We show the contrapositive. Suppose that $M \notin \mathcal{M}_i$ for any i. If M is a superset of any $M' \in \mathcal{M}_i$, then M is not minimal since M' is a model of DB. On the other hand if for any set $M' \in \mathcal{M}_i$, either $M \subset M'$ or $M \bigcap M' \neq M'$ then there must exist a clause, D_i, in each DB_i such that M does not model D_i. Then $D_1 \vee \cdots \vee D_k$ is not modeled by M hence M is not a model of DB. ■

THEOREM 4.3. *Let DB be a stratified deductive database in L. Let P be an atom in L_I and assume $DB \not\vdash P$. Then the deductive database DB', constructed by Algorithm* 4, *implies P. That is, $DB' \vdash P$. Moreover, $DB' = \bigvee\{DB'' | DB'' \vdash P$ and $\nexists DB'' \vdash P$, and $\nexists DB''' \vdash P$ such that $\mathrm{Cn}(E_{DB''}) \bigcap \mathrm{Cn}(E_{DB}) \subset \mathrm{Cn}(E_{DB'''}) \bigcap \mathrm{Cn}(E_{DB})\}$.*

Proof. Assume first that $E_{DB} \cup \{C_1, \ldots, C_m\}$ is consistent. Since, by Theorem 4.1, DB' is equivalent to $\mathrm{Cn}(DB \cup \{C_1, \ldots, C_m\})$ it follows that $DB' \vdash P$. Now, since $E_{DB} \cup \{C_1, \ldots, C_m\}$ is consistent, again from Theorem 4.1 $\mathrm{Cn}(E_{DB'}) \supseteq \mathrm{Cn}(E_{DB})$. Therefore, $\nexists DB''' \vdash P$, such that $\mathrm{Cn}(E_{DB'}) \bigcap \mathrm{Cn}(E_{DB}) \subset \mathrm{Cn}(E_{DB'''}) \bigcap \mathrm{Cn}(E_{DB})$. Assume that there exists another DB'' such that $DB'' \vdash P$ and $\nexists DB''' \vdash P$, such that $\mathrm{Cu}(E_{DB''}) \bigcap \mathrm{Cn}(E_{DB}) \subset \mathrm{Cn}(E_{DB'''}) \bigcap \mathrm{Cn}(E_{DB})$. Then it is also true that $\mathrm{Cn}(E_{DB''}) \supseteq \mathrm{Cn}(E_{DB})$; otherwise we can choose DB' as DB''' and show that $\mathrm{Cn}(E_{DB''}) \bigcap \mathrm{Cn}(E_{DB}) \subset \mathrm{Cn}(E_{DB'''}) \bigcap \mathrm{Cn}(E_{DB})$. Let M be a model of DB''. Then M is also a model of DB and a model of $\{C_1, \ldots, C_m\}$. But by Theorem 4.1,

DB' contains all the possible positive resolvents of DB with $\{C_1, \ldots, C_m\}$. Therefore, by the soundness and completeness of resolution, M is also a model of DB'. Therefore, $DB'' \vdash DB'$. Hence, the disjunction of all DB'', $\bigvee DB''$ is equivalent to DB'.

If $E_{DB} \cup \{C_1, \ldots, C_m\}$ is inconsistent, consider a DB_i as constructed in step 6a. By the argument above, for each i, $1 \leq i \leq k$, $DB_i' \vdash P$. That is, in every minimal model of each DB_i', $1 \leq i \leq k$, each C_j, $1 \leq j \leq m$, is true. By Lemma 4.2 and step 6b the same is true for DB'. Also, the constructions of the DB_i' assume that they play the role of DB'' in the statement of the theorem. Hence the result follows from the first part. ■

4.2. NORMAL DELETIONS

Algorithm 5 (Deletion of an intensional atom from a stratified disjunctive database – Deletion 3). Suppose that P is an intensional atom, and let DB be a stratified disjunctive database such that $GCWA(DB) \vdash P$. The algorithm computes a database DB' such that $GCWA(DB)' \nvdash P$.

1. Let $\mathcal{M}$ be the set of minimal models of E_{DB}, the extensional component of DB.
2. Construct a restricted stratified SLD-tree for $\leftarrow P$ from DB.
3. Let $F_1, \ldots, F_n$ be the leaf nodes containing only conjunctions of (possibly negated) extensional atoms. Construct the conjunctive normal form of $F_1 \vee \cdots \vee F_n$, writing it as $C_1 \wedge \cdots \wedge C_m$, where subsumed clauses are omitted from the conjunctive normal form.
4. Select a disjunction C_j, $1 \leq j \leq m$. The selection is made giving preference to C_j which satisfies step 5 or the first part of step 7, if possible, to minimize deletions.
5. If C_j is a negative clause, $C_j = \neg E_1 \vee \cdots \vee \neg E_l$ then:
   For each $M \in \mathcal{M}$, let $e(M)$ be the number of atoms in M from $\{E_1, \ldots, E_l\}$. Let $m = \max\{e(M)\}$ for all $M \in \mathcal{M}$. Select $M \in \mathcal{M}$ such that $e(M) = m$. Let $M' = M \cup \{E_1, \ldots, E_l\}$. Let $\mathcal{M} = (\mathcal{M} - \{M\}) \cup \{M'\}$. Eliminate from $\mathcal{M}$ any proper subset of M'.
6. If C_j is a positive clause $C_j = A_1 \vee \cdots \vee A_l$ then:
   Let $M' = HB_{E_{DB}} - \{A_1, \ldots, A_l\}$. If $M' = \emptyset$ let $\mathcal{M} = \emptyset$; otherwise let $\mathcal{M} = \mathcal{M} \cup \{M'\}$. Eliminate from $\mathcal{M}$ any proper superset of M'.
7. If C_j is a mixed clause, $C_j = A_1 \vee \cdots \vee A_l \vee \neg E_1 \vee \cdots \vee \neg E_s$ then:
   If there exists a model in $\mathcal{M}$ that contains no A_i, $1 \leq i \leq l$, then, for each $M \in \mathcal{M}$ let $e(M)$ be the number of atoms in M from $\{E_1, \ldots, E_s\}$. Let $m = \max\{e(M)\}$ for all $M \in \mathcal{M}$. Select $M \in \mathcal{M}$ such that $e(M) = m$. Let $M' = M \cup \{E_1, \ldots, E_s\}$. Let $\mathcal{M} = (\mathcal{M} - \{M\}) \cup \{M'\}$.
   Eliminate from $\mathcal{M}$ any proper subset of M'.
   Otherwise, let $M' = HB_{DB} - \{A_1, \ldots, A_1\}$. Let $\mathcal{M} = \mathcal{M} \cup \{M'\}$. Eliminate from $\mathcal{M}$ any proper superset of M'.
8. Build DB' from the new $\mathcal{M}$.

EXAMPLE 4.4. Let DB be the stratified disjunctive database:

$$P \leftarrow \neg Q$$
$$Q \leftarrow \neg C, \neg D$$
$$Q \leftarrow A, R$$
$$R \leftarrow B, C$$
$$R \leftarrow, A, \neg D$$
$$A \vee C \leftarrow$$
$$B \vee D \leftarrow$$
$$C \vee D \leftarrow$$

Three conjuncts are obtained from the restricted SLD-tree for $\leftarrow P$: $C_1 = C \vee D$, $C_2 = \neg A \vee \neg B \vee \neg C$, and $C_3 = \neg A \vee D$. The minimal models of the extensional part of the database are: $\mathcal{M} = \{\{A, D\}, \{B, C\}, \{C, D\}\}$.

Assume now that we want to delete P from the database. Suppose we select C_2 for the deletion. The algorithm changes the model $\{B, C\}$ to $\{A, B, C\}$. Now $\mathcal{M} = \{\{A, D\}, \{A, B, C\}, \{C, D\}$. The effect on DB is to add $A \vee D$.

Alternatively, if C_1 is selected, the new minimal model $\{A, B\}$ is added. The new $\mathcal{M} = \{\{A, B\}, \{A, D\}, \{B, C\}, \{C, D\}\}$. The effect on DB is to delete $C \vee D$. Selecting C_3 has the same effect as selecting C_2. ■

The following lemma shows that Algorithm 5 accomplishes the desired deletion.

LEMMA 4.4. *Let DB be a stratified deductive database in L. Let P be an atom in L_I and assume $DB \vdash P$. Then the deductive database DB', constructed by Algorithm 5 does not imply P. That is, $DB' \nvdash P$.*

Proof. The construction creates a database with a minimal model where the clause C_i selected in step 4 is not *true*. ■

The next result shows in what sense the deletion is minimal. First of all, minimality is defined with respect to the C_j that is chosen in step 4. Observe that if step 5 or the first part in step 7 is the step executed to obtain DB' then $E_{DB'} \vdash E_{DB}$. Hence, if insertions are preferred over deletions, a C_j should be chosen for which step 5 or the first part of step 7 is applicable. Under these considerations there is no database $E_{DB''}$ that does not imply the chosen C_j that is weaker than E_{DB} but stronger than $E_{DB'}$.

THEOREM 4.5. (1) *There does not exist DB'', such that $DB'' \nvdash C_j$ and* $\mathrm{Cn}(E_{DB'}) \bigcap \mathrm{Cn}(E_{DB}) \subset \mathrm{Cn}(E_{DB''}) \bigcap \mathrm{Cn}(E_{DB})$. (2) *There does not exist DB'', such that $DB'' \nvdash C_j$,* $\mathrm{Cn}(E_{DB'}) \bigcap \mathrm{Cn}(E_{DB}) = \mathrm{Cn}(E_{DB''}) \bigcap \mathrm{Cn}(E_{DB})$, $E_{DB'} \vdash E_{DB''}$, $E_{DB''} \nvdash E_{DB'}$.

Proof. Case 1. C_j is negative, $\neg E_1 \vee \cdots \vee \neg E_k$. In this case $E_{DB'} \bigcap E_{DB} = E_{DB}$,

hence statement 1 is satisfied automatically. For statement 2, the existence of such an $E_{DB''}$ would require some atom missing from a minimal model of $E_{DB'}$, which is impossible due to the way the algorithm constructs $E_{DB'}$.

Case 2. C_j is mixed, $\neg E_1 \vee \cdots \vee \neg E_k \vee A_1 \vee \cdots \vee A_l$. There are two possibilities. First, if there exists a minimal model that contains no A_j, $1 \leq j \leq l$, then the proof is similar to case 1. Second, if originally $E_{DB} \vdash A_1 \vee \cdots \vee A_l$, then $A_1 \vee \cdots \vee A_l$ now must be deleted. The algorithm replaces $A_1 \vee \cdots \vee A_l$ by $A_1 \vee \cdots \vee A_l \vee B$, for all possible ground atoms B not in the disjunction. Clearly this is minimal (see Deletion 2, Algorithm 3) and conditions 1 and 2 are satisfied.

Case 3. C_j is positive. Same argument as for the second part of case 2. ■

5. Conclusion

We have developed algorithms for the insertion and deletion of intensional atoms into and from stratified disjunctive databases. We have also shown in what sense these algorithms are optimal. There are a variety of ways in which this work can be extended. For example, it would be useful to consider Skolem constants, as in [8], to apply this approach to programs with variables; and it may be possible to use the tree representation of the minimal models of a disjunctive database described in [3] to implement the insertion of information into the stratified database. Such algorithms are currently under investigation. The most technically challenging project, however, would be to extend the algorithms developed here to a richer class of disjunctive databases which are not stratified, and also rules that contain disjunctions in their heads.

Acknowledgement

We wish to thank Jose Alberto Fernandez, Chiaki Sakama, and the referee for helpful comments.

References

1. Apt, K. R., Blair, H. A. and Walker, A., 'Towards a theory of declarative knowledge', in *Foundations of Deductive Databases and Logic Programming*, (ed. J. Minker), Morgan Kaufmann, pp. 89–148 (1987).
2. Fagin, R., Ullman, J. and Vardi, M., 'On the semantics of updates in databases', in *Proceedings of the Second ACM Symposium on the Principles of Database Systems* pp. 352–365 (1983).
3. Fernandez, J. and Minker, J., 'Bottom up evaluation of disjunctive deductive databases', Submitted.
4. Guessoum, A. and Lloyd, J., 'Updating knowledge bases', *New Generation Computing* **8**, 71–89 (1990).
5. Guessoum, A. and Lloyd, J., 'Updating knowledge bases II', *New Generation Computing* **10**, 73–100 (1991).
6. Minker, J., 'On indefinite databases and the closed world assumption', in *Lecture Notes in Computer Sciences*, Vol. 138, Springer Verlag, pp. 292–308 (1982).
7. Reiter, R., 'On closed world data bases', in *Logic and Data Bases*, (eds. H. Gallaire and J. Minker), Plenum, pp. 119–140 (1978).
8. Rossi, F. and Naqvi, S., 'Contributions to the view update problem', In *Proceeding of the Sixth International Conference on Logic Programming*, pp. 398–415 (1989).