Updating Disjunctive Databases via Model Trees *

John Grant
Computer and Information Sciences, Towson State University
Jarosław Gryz
Computer Science Department, University of Maryland, College Park
Jack Minker
UMIACS and Computer Science Department, University of Maryland, College Park

Abstract

In this paper we study the problem of updating disjunctive databases, which contain indefinite data given as positive disjunctive clauses. We give correct algorithms for the insertion of a clause into and the deletion of a clause from such databases. Although the algorithms presented here are oriented towards model trees, they apply to any representation of minimal models.

1 Introduction

Updating relational databases is a relatively straightforward problem. In this paper we are concerned with updating disjunctive databases, which contain indefinite data given as positive disjunctive clauses. In such databases it is not a straightforward process to update data in certain representations. We consider the update problem for disjunctive databases represented by *model trees*. The concept of a model tree was introduced in [4] to provide a compact representation for disjunctive databases in terms of minimal models.

*This research has been supported by the following grants: NSF IRI-9200898, NSF IRI-8916059, and AFOSR 910350

Model trees were shown to be a useful data structure for the representation of disjunctive databases and query processing. In this paper we give correct algorithms for the insertion of a clause into and the deletion of a clause from a disjunctive database. Although the algorithms presented here are oriented towards model trees, they apply to any representation of minimal models. These operations are important not only for the manipulation of the database but also because the solution of the view update problem (the update of intensional predicates) for deductive disjunctive databases presupposes the capability to perform such updates. Algorithms for the view update problem for normal disjunctive databases are presented in [5] and [3].

The content of the rest of the paper is as follows. Section 2 contains basic definitions and notation. An algorithm for inserting a clause, including a proof of its correctness, is given in Section 3. Section 4 contains an algorithm for deletion, along with a proof of its correctness and analysis of its complexity. In Section 5 several special cases of updates are reviewed. The paper is concluded in Section 6.

2 Preliminaries

We define a *disjunctive database* DB to contain a set of ground clauses of the form $a_1 \vee ... \vee a_n$, $n \geq 1$, where the a_i's, $1 \leq i \leq n$ are atomic formulae. The reason for not including any rules is that we only deal with updates of the extensional database. We assume that DB does not contain redundant clauses, that is clauses C_1 and C_2 where $C_1 \models C_2$. A *model tree* (for a disjunctive database DB) is a tree whose nodes are labeled by atoms of the Herbrand base of DB (except the root) in such a way that the branches correspond exactly to the minimal models of the database, $\mathcal{MM}_{DB}$. A model is *minimal* if it is not contained in any other model. The *minimization* of a set of models consists of removing non-minimal models from the set. For a set of models $\mathcal{M}$, we write $min(\mathcal{M}) = \{M \in \mathcal{M} | \not\exists M' \in \mathcal{M},\ M' \subset M\}$. In particular, if $Mod(DB)$ is the set of models of DB, then $\mathcal{MM}_{DB} = min(Mod(DB))$.

Example 1 *Let the database DB contain the following clauses: $\{a \vee c,\ a \vee b \vee f,\ b \vee c \vee d,\ b \vee c \vee e,\ b \vee e \vee f\}$. After minimization the model tree looks as shown in Figure 1.*

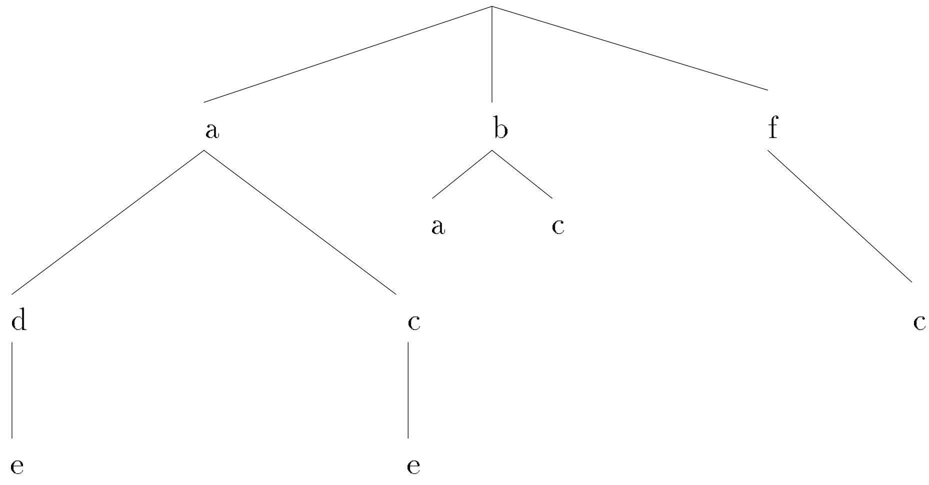

Figure 1: Model tree representation for DB

Here $\mathcal{MM}_{DB} = \{\{a,b\},\{b,c\},\{c,f\},\{a,c,e\},\{a,d,e\}\}$. *Sometimes it is also useful to represent* DB *as a set of clauses, for example, we write* $DB = \{[a,c],[a,b,f],[b,c,d],[b,c,e],[b,e,f]\}$, *where the notation* $[a_1,...,a_n]$ *denotes the clause* $a_1 \vee ... \vee a_n$. *Although written using brackets instead of braces, it denotes a set, thus all set operations with clauses as operands are legal.*

There may be many different model trees constructed for a single database[1]; we try to minimize the number of nodes by allowing branches to share atoms, such as for $\{a,c,e\}$ and $\{a,d,e\}$ in Figure 1.

Another way of minimizing the size of model trees is to represent unrelated knowledge in separate trees. Sometimes, a database can be partitioned into sets of related clauses called *clusters* each of which can be represented by a separate model tree. The representation of the database is then called a *model forest*. For a formal description of model forest see [2]. The following is an example of a database partitioned as a model forest.

Example 2 *Let* $DB' = \{[a,b,c],[b,d],[e,f],[f,g],[e,h]\}$. DB' *consists of two unrelated sets of clauses (databases):* $DB'_1 = \{[a,b,c],[b,d]\}$, *and* $DB'_2 =$

[1]For ordered model trees such representation is unique for a given database. See [1] for a description of ordered model trees.

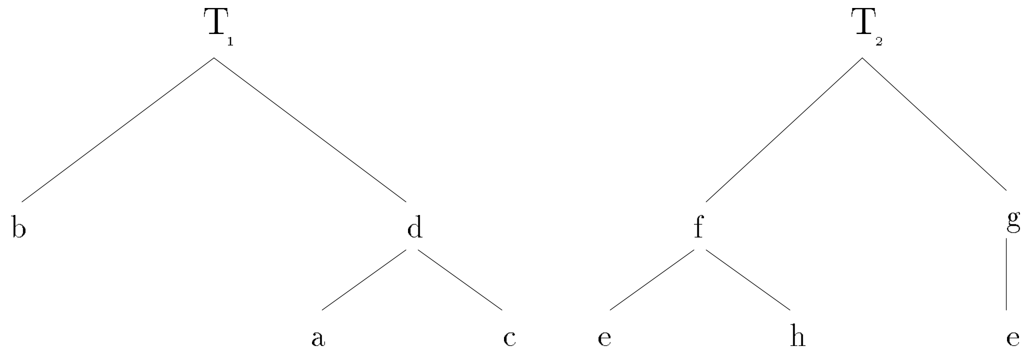

Figure 2: Model forest representation of DB'

$\{[e, f], [f, g], [e, h]\}$. *They can be represented by two model trees shown in Figure 2. Notice that if DB' were represented in one tree, T_1 could be attached to every leaf of T_2 (or vice versa) with the resulting tree substantially larger than the number of nodes of T_1 and T_2.*

By the *insertion* of a clause C into DB we mean $DB \cup \{C\}$. In this case we assume $DB \not\models C$. By the *deletion* of a clause C from DB we mean $DB - \{C\}$. In this case we assume $C \in DB$. We use the following convention: DB^- is the database resulting from the deletion of C from DB, and DB^+ is the database resulting from the insertion of C into DB. Other types of deletion will be discussed in Section 5.

3 Insertion

This section contains our algorithm for the insertion of a clause into a database. The algorithm is shown to be correct and is illustrated by an example. For the insertion algorithm we assume $\mathcal{MM}_{DB}$ and $C = a_1 \vee ... \vee a_m$ are given, where $DB \not\models C$.

Procedure **Insertion** *($\mathcal{MM}_{DB}$, C) returns $\mathcal{MM}_{Ins}$*

1. $\mathcal{MM} := \emptyset$

2. For each model $M \in \mathcal{MM}_{DB}$
3. If $\exists i \; 1 \leq i \leq m, \; s.t. \; a_i \in M$
4. Then $\mathcal{MM} := \mathcal{MM} \cup \{M\}$
5. Else $\mathcal{MM} := \mathcal{MM} \cup (\bigcup_{i=1}^{m} \{M \cup \{a_i\}\})$
6. $\mathcal{MM}_{Ins} := min(\mathcal{MM})$

We trace this algorithm on Example 1 where DB does not contain the clause $a \vee c$. So let $DB = \{[a, b, f], [b, c, d], [b, c, e], [b, e, f]\}$ and C=[a,c]. Figure 3 shows a model tree representation for DB; $\mathcal{MM}_{DB}$=$\{\{a, d, e\}, \{a, c, e\}, \{b\}, \{d, e, f\}, \{c, f\}\}$. $\mathcal{MM}$ is modified in step 2 as follows:
$\mathcal{MM} = \emptyset$

$M = \{a, d, e\}$
$\mathcal{MM} = \{\{a, d, e\}\}$

$M = \{a, c, e\}$
$\mathcal{MM} = \{\{a, d, e\}, \{a, c, e\}\}$

$M = \{b\}$
$\mathcal{MM} = \{\{a, d, e\}, \{a, c, e\}, \{a, b\}, \{b, c\}\}$

$M = \{d, e, f\}$
$\mathcal{MM} = \{\{a, d, e\}, \{a, c, e\}, \{a, b\}, \{b, c\}, \{a, d, e, f\}, \{c, d, e, f\}\}$

$M = \{c, f\}$
$\mathcal{MM} = \{\{a, d, e\}, \{a, c, e\}, \{a, b\}, \{b, c\}, \{a, d, e, f\}, \{c, d, e, f\}, \{c, f\}\}$

Finally, after minimization we obtain: $\mathcal{MM}_{Ins} = \{\{a, b\}, \{b, c\}, \{c, f\}, \{a, c, e\}, \{a, d, e\}\}$.

We note that this algorithm was implicitly given in [4]. The next Lemma is taken from [3] and thus presented here without a proof.

Lemma 3.1 *Let DB_1 and DB_2 be disjunctive databases. Then, $DB_2 \models DB_1$ iff $\forall M \in \mathcal{MM}_{DB_2} \; \exists M' \in \mathcal{MM}_{DB_1}$ s.t. $M' \subseteq M$.*

Theorem 3.1 *Procedure* **Insertion** *is correct.*

PROOF: We must show that $\mathcal{MM}_{Ins} = \mathcal{MM}_{DB\cup\{C\}}$. That is, for every clause D, $DB_{Ins} \models D\ iff\ DB\cup\{C\} \models D$. (We use the fact that $DB\cup\{C\} \models D\ iff\ DB \models D\ or\ C \models D$).

($\Leftarrow$)

By Lemma 3.1, if $DB \models D$ then $DB_{Ins} \models D$ because $\forall M \in DB_{Ins} \exists M' \in DB\ s.t.\ M' \subseteq M$ by the construction. Also, $DB_{Ins} \models C$ because $\forall M \in DB_{Ins} \exists a_i,\ 1 \leq i \leq m\ s.t.\ a_i \in M$ by the construction, hence if $C \models D$, then $DB_{Ins} \models C$.

($\Rightarrow$)

Suppose that $DB \not\models D$ and $C \not\models D$ where $D = b_1 \vee ... \vee b_n$. To obtain the contrapositive, we show that $DB_{Ins} \not\models D$. By the assumptions $\exists M_0 \in \mathcal{MM}_{DB}$ s.t. $\forall j,\ 1 \leq j \leq n\ b_j \notin M_0$ and $C = [a_1, ..., a_m] \not\subseteq [b_1, ..., b_n] = D$. If $M_0 \in \mathcal{MM}_{Ins}$, then $DB_{Ins} \not\models D$. Otherwise, for every $a_i,\ 1 \leq i \leq m$, either $M_0 \cup \{a_i\} \in \mathcal{MM}_{Ins}$ or $\exists M_i \in \mathcal{MM}_{DB}$ such that $M_i \subset M_0 \cup \{a_i\}$ and so $M_i \in \mathcal{MM}_{Ins}$. Let i be such that $[a_i] \not\subseteq [b_1, ..., b_n]$. Then, $\forall j,\ 1 \leq j \leq n,\ b_j \notin M \cup \{a_i\}$, hence $DB_{Ins} \not\models D$.

$\square$

It is easy to see that the complexity of this algorithm is determined by the minimization procedure of line 6. Since the number of models of $\mathcal{MM}$ before the execution of line 6 is no greater than m^n (where m is the average clause size and n is the number of clauses in DB^+), the worst case complexity of the minimization procedure, and hence of the algorithm, is $O(m^{2n})$.

4 Deletion

In this section we take deletion to mean the deletion of a clause together with all of its consequences that are not implied by other clauses. This type of deletion generalizes deletion in relational databases where the deletion of a tuple removes the tuple from the database. Thus, in particular, there is no reason to assume that any consequence of the deleted clause (atom) - not implied independently by the remaining clauses - should be derivable from the database after deletion. Consequently, the database representation (in our case in the form of a model tree) should be brought to the same form it would have had if the database did not contain the clause to be deleted.

Although this seems to be the most natural interpretation of deletion, it is also the most expensive to implement. The brute force approach discards

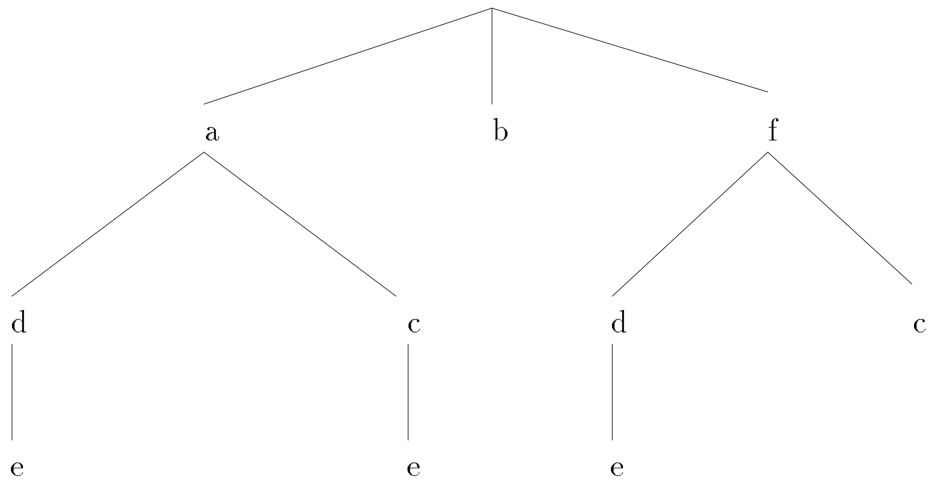

Figure 3: The minimized model tree for clauses DB^-

the model tree for DB and creates a new one for $DB - \{C\}$. But even more subtle techniques presented in the algorithm below have complexity exponential in the number of clauses in the database. The main priority in the algorithm presented below is to retain as much as possible of the tree structure representing the database before deletion.

We show how the algorithm works on Example 1 presented in Section 1, whose model tree is shown in Figure 1. We delete $C = (a \vee c)$ from the database described in that example. Figure 3 shows a correct model tree for the remaining clauses: $DB^- = \{[a,b,f],[b,c,d],[b,c,e],[b,e,f]\}$.

The two trees of Figure 1 and Figure 3 differ in two ways. First, a and c are deleted from some of the models of the tree of Figure 1. Second, a new minimal model, $\{d,e,f\}$ is added to the tree in Figure 3.

Before we describe the algorithm, we define several concepts.

Definition 4.1 *Let C be a clause not in DB, and $M \in \mathcal{MM}_{DB}$. The* **C-extension** *of M, $Ext_C(M)$ is defined as follows:*

$$Ext_C(M) = \begin{cases} \{M' | M' = M \cup \{p\},\ p \in C\} & if\ M \cap C = \emptyset \\ \{M\} & if\ M \cap C \neq \emptyset \end{cases}$$

In Example 1, $Ext_{(a \vee c)}(\{b\}) = \{\{b, a\}, \{b, c\}\}$.

Observation 4.1 *Let C be a clause deleted from DB, and $M \in \mathcal{MM}_{DB^-}$. Then $\forall M' \in Ext_C(M),\ M' \models DB$.*

Note that for $M' \in Ext_C(M),\ M' \models DB$, but M' need not be a minimal model of DB . In Example 1 $\{d, e, f\}$ is a minimal model of DB^-, but neither $\{d, e, f, a\}$ nor $\{d, e, f, c\}$ is a minimal model of DB.

The next observation, which follows immediately from Lemma 3.1, shows that we never have to remove more than one atom from a model in $\mathcal{MM}_{DB}$ to make it a model of $\mathcal{MM}_{DB^-}$.

Observation 4.2 *Let C be a clause deleted from DB. Then, $\forall M' \in \mathcal{MM}_{DB}$ $\exists M \in \mathcal{MM}_{DB^-}$, s.t. $M' \in Ext_C(M)$.*

From these observations it follows that, in general, $\mathcal{MM}_{DB^-}$ is obtained from $\mathcal{MM}_{DB}$ by deleting one atom each from some elements of $\mathcal{MM}_{DB}$ and adding new models. The algorithm described below performs separately these tasks. There are, however, special cases with only atom deletion and no model addition. We call such cases *simple deletions.*

Definition 4.2 *Let C be a clause deleted from DB. The deletion is* **simple** *if $\forall M \in \mathcal{MM}_{DB^-}$ $\exists M' \in Ext_C(M)$ s.t. $M' \in \mathcal{MM}_{DB}$.*

Example 3 *Consider the database DB=(DB'_2 in Example 2)=$\{[e, f], [f, g], [e, h]\}$ from which $[e, h]$ is to be deleted. Then $\mathcal{MM}_{DB^-} = \{\{f\}, \{e, g\}\}$. The $[e, h]$-extensions of these models are respectively $\{\{e, f\}, \{f, h\}\}$ and $\{\{e, g\}\}$. Since $\mathcal{MM}_{DB} = \{\{e, f\}, \{e, g\}, \{f, h\}\}$, this is a simple deletion.*

It is easy to see (Observation 4.2) that in the case of a simple deletion a model tree of DB^- can be created out of a model tree of DB by removing at most one atom from every model of DB.

The next lemma shows two properties that a minimal model of DB^- must have if the deletion of C from DB is not simple.

Lemma 4.1 *Let $C \in DB$, and $M \in \mathcal{MM}_{DB^-}$. Then, $\forall M' \in Ext_C(M),\ M' \notin \mathcal{MM}_{DB}$*

iff

1. $M \cap C = \emptyset$

and

2. $\forall a \in C\ \exists M_1 \subset M$ s.t. $M_1 \cup \{a\} \in \mathcal{MM}_{DB^-}$.

PROOF

($\Rightarrow$) By the contrapositive.

1. Assume that $M \cap C \neq \emptyset$. Then $M' = M$, hence $M' \in \mathcal{MM}_{DB}$.

2. Assume that $\exists a \in C\ s.t.\ \forall M_1 \subset M\ M_1 \cup \{a\} \notin \mathcal{MM}_{DB^-}$ and $M \cap C = \emptyset$. Consider $M_2 = M \cup \{a\}$. Since M is a model of DB^-, M_2 is also a model of DB. Let $M_3 \subset M_2$. If $a \in M_3$, then by the assumption M_3 is not a model of DB, while if $a \notin M_3$, then $M_3 \cap C = \emptyset$ and again M_3 is not a model of DB. Thus, $M_2 \in \mathcal{MM}_{DB}$, but also $M_2 \in Ext_C(M)$.

($\Leftarrow$)

By 1., M does not contain any atoms of C, hence $\forall M' \in Ext_C(M)\ M \subset M'$, *but* $M \neq M'$. But also, by 2., $\forall a \in C\ \exists M_1 \subset M\ s.t.\ M_1 \cup \{a\} \in \mathcal{MM}_{DB^-}$, so $\forall M' \in Ext_C(M)\ \exists M''\ (M'' = M_1 \cup \{a\},\ a \in C),\ s.t.\ M'' \subset M'$. Hence, $M' \notin \mathcal{MM}_{DB}$.

$\square$

Example 4 *The only model of DB^- in Figure 3 whose $[a, c]$-extension does not contain any models in DB (see Figure 1) is $\{f, d, e\}$. Note that 1: it does not contain any atoms of the deleted clause $[a, c]$ and 2: $[a, d, e]$ and $[f, c]$ are in $\mathcal{MM}_{DB^-}$. Also, this is the only model of DB^- that satisfies conditions 1 and 2 of the above Lemma.*

Finding all minimal models that need to be added to the model tree when a deletion is executed is a very expensive procedure; it may require computing minimal models of a large part of the database. Thus, an early recognition that the deletion is simple would substantially reduce the amount of work that needs to be done. However, the only sufficient condition we have is when the clause to be deleted contains an atom which does not belong to any other clause in the database. The following lemma states this fact formally.

Lemma 4.2 *Let C be a clause in DB. If $\exists b \in C\ s.t.\ \forall C' \in DB\ C' \neq C,\ b \notin C'$, then*

$M \in \mathcal{MM}_{DB^-}$ *iff* $\exists M' \in \mathcal{MM}_{DB},\ s.t.\ M' \in Ext_C(M)$.

PROOF:

($\Rightarrow$) Assume that this is not the case. Then there is an $M \in \mathcal{MM}_{DB^-}$ s.t. there is no $M' \in Ext_C(M),\ s.t.\ M' \in \mathcal{MM}_{DB}$. Then by 2. of Lemma 4.1, $\exists M_1 \subset M\ s.t.\ M_1 \cup \{b\} \in \mathcal{MM}_{DB^-}$. But that is impossible since b does not appear in DB^-.

($\Leftarrow$) This follows immediately from Observation 4.2.

$\square$

Lemma 4.2 does not provide a necessary condition for a deletion to be simple, i.e. a deleted clause does not have to contain an atom not shared by any other clause in the database for the deletion to be simple. Here is an example.

Example 5 *Let $DB = \{a \vee b, b \vee c, a \vee c\}$. Then, $\mathcal{MM}_{DB} = \{\{a, c\}, \{a, b\}, \{b, c\}\}$. After deleting $a \vee c$, $\mathcal{MM}_{DB^-}$ becomes $\{\{a, c\}, \{b\}\}$. This is a simple deletion which does not satisfy the condition of Lemma 4.2.*

Next we give our deletion algorithm. We start by explaining the notation. Let DB be a database containing the following disjunctions: $C_0, C_1, ..., C_n$ and let $C_0 = (a_1 \vee a_2 \vee ... \vee a_m)$ be the clause to be deleted. Let DB^0 be the set of all clauses of DB from which all atoms of C_0 have been removed, i.e.

$DB^0 = \{C | C = C' - C_0, \ where \ C' \in DB\}$.

Let $\mathcal{C}^{a_j}$ be the set of clauses in DB^0 which were obtained from clauses of DB that contained a_j, $1 \leq j \leq m$, i.e.

$\mathcal{C}^{a_j} = \{C | C = C' - C_0, \text{ where } C' \in DB, \ a_j \in C'\}$.

For Example 1, where we delete $a \vee c$ from DB, $\mathcal{C}^c = \{b \vee d, \ b \vee e\}$.

The following two lemmas are used in the algorithm; they relate DB^0 and $\mathcal{C}^{a_i}$ to $\mathcal{MM}_{DB^-}$. The first lemma connects the clauses of $\mathcal{C}^{a_i}$ with the elements of $\mathcal{MM}_{DB}$ that contain exactly one $a_i \in C_0$. The second lemma shows that every element of $\mathcal{MM}_{DB^-}$ not in $\mathcal{MM}_{DB}$ is in $\mathcal{MM}_{DB^0}$. In the first lemma we continue with the notation from the definition of $\mathcal{C}^{a_i}$ that for $C \in \mathcal{C}^{a_i}$, $C = C' - C_0$ with $C' \in DB^-$.

Lemma 4.3 *Let $a_i \in M \in \mathcal{MM}_{DB}$ and suppose that for all a_j, $1 \leq i \neq j \leq m$, $a_j \notin M$. Then, $M - \{a_i\} \in \mathcal{MM}_{DB^-}$ iff $\forall C \in \mathcal{C}^{a_i}, \ M \models C$.*

PROOF: ($\Rightarrow$) By the contrapositive. Assume that $C \in \mathcal{C}^{a_i}$ and $M \not\models C$. Then, $a_i \in C'$ and a_i is needed in M for C', hence $M - \{a_i\} \notin \mathcal{MM}_{DB^-}$.

($\Leftarrow$) Assume that $\forall C \in \mathcal{C}^{a_i}, \ M \models C$. Then, for all $C' \in DB^-$ corresponding to C, a_i is not needed in M, hence $M - \{a_i\} \in \mathcal{MM}_{DB^-}$.

$\square$

Lemma 4.4 *$\mathcal{MM}_{DB^0} = \mathcal{MM}_{DB^-} - \mathcal{MM}_{DB}$.*

PROOF:

($\subseteq$) Let $M \in \mathcal{MM}_{DB^0}$. Clearly, M is a model of DB^- and $M \cap C_0 = \emptyset$. Since $M \cap C_0 = \emptyset$, M is not a model of DB and M is a minimal model of DB^-. Thus, $M \in \mathcal{MM}_{DB^-} - \mathcal{MM}_{DB}$

($\supseteq$) Let $M \in \mathcal{MM}_{DB^-}$ and $M \cap C_0 \neq \emptyset$. Then $M \in \mathcal{MM}_{DB}$. So, if $M \in \mathcal{MM}_{DB^-} - \mathcal{MM}_{DB}$, then $M \cap C_0 = \emptyset$. Therefore $M \in \mathcal{MM}_{DB^0}$.

$\square$

Lemma 4.4 suggests an algorithm to find $\mathcal{MM}_{DB^-}$: compute $\mathcal{MM}_{DB^0}$ and minimize $\mathcal{MM}_{DB} \cup \mathcal{MM}_{DB^0}$. Instead, our algorithm uses Lemmas 4.3 and 4.4 for reasons to be explained later when we analyze the complexity of deletion algorithms.

Procedure **Deletion** *(DB, $\mathcal{MM}_{DB}$, C_0) returns $\mathcal{MM}_{Del}$*

1. Construct DB_0 and $\mathcal{MM}_{DB^0}$
2. For each model $M \in \mathcal{MM}_{DB}$
3. If M contains exactly one $a_i \in C_0$
4. If $\forall C \in \mathcal{C}^{a_i}, M - \{a_i\} \models C$, then
5. $\mathcal{MM}_{DB} := \mathcal{MM}_{DB} - M$
6. $\mathcal{MM}_{Del} := \mathcal{MM}_{DB} \cup \mathcal{MM}_{DB^0}$

Example 6 *We apply this algorithm to the database of Example 1, where we are deleting $a \vee c$. Then, in step 1 we obtain $DB^0 = \{[b,f],[b,d],[b,e],[b,e,f]\}$, $\mathcal{C}^a = \{[b,f]\}$, $\mathcal{C}^c = \{[b,d],[b,e]\}$, $\mathcal{MM}_{DB^0} = \{\{b\},\{f,d,e\}\}$.*

In step 2 the algorithm considers the model $\{a,b\}$. Since all clauses in $\mathcal{C}^a$ are true in $\{a,b\}$, $\{a,b\}$ is removed from $\mathcal{MM}_{DB}$. Similar action is performed on $\{b,c\}$. For all other models the loop is executed vacuously. After leaving the loop $\mathcal{MM}_{DB^0}$ is added to the modified tree of DB.

Theorem 4.1 *Procedure* **Deletion** *is correct, i.e. $\mathcal{MM}_{Del} = \mathcal{MM}_{DB^-}$.*

PROOF: Recall from the comment after Lemma 4.4 that $\mathcal{MM}_{DB^-}$ can be obtained by minimizing $\mathcal{MM}_{DB} \cup \mathcal{MM}_{DB^0}$. We need to show that this is achieved in step 5. So, let $M \in (\mathcal{MM}_{DB} \cup \mathcal{MM}_{DB^0}) - \mathcal{MM}_{DB^-}$. By Lemma 4.4, $M \in \mathcal{MM}_{DB}$. By the comment immediately preceding

Observation 4.2 and by Lemma 4.3 exactly one atom must be removed from M to obtain an element of $\mathcal{MM}_{DB^-}$. Since the resulting model does not contain any atoms of C_0 it is already in $\mathcal{MM}_{DB^0}$. Hence, M can be safely removed from $\mathcal{MM}_{DB}$.

$\square$

Before analyzing the complexity of the deletion algorithm we provide another characterization of $\mathcal{MM}_{DB^-}$, one that is based entirely on models and does not involve clauses. We start with definitions leading to $\mathcal{M}^*$, a set that contains the elements of $\mathcal{MM}_{DB^-} - \mathcal{MM}_{DB}$. First, let $\mathcal{M}^i$ be the set of elements of $\mathcal{MM}_{DB}$ containing only one atom of C_0, namely a_i, i.e. $\mathcal{M}^i = \{M \in \mathcal{MM}_{DB} | M \cap C_0 = \{a_i\}\}$, $1 \leq i \leq m$, say $\mathcal{M}^i = \{M_1^i, ..., M_{k_i}^i\}$. Let $M_{<j_1,...,j_m>} = \bigcup_{i=1}^m M_{j_i}^i - \{a_1, ..., a_m\}$ and $\mathcal{M}^* = \{M_{<j_1,...,j_m>} | 1 \leq j_i \leq k_i\}$. Thus each element of $\mathcal{M}^*$ is obtained by picking an element from each $\mathcal{M}^i$, taking the union of these models and deleting the atoms of C_0.

Example 7 *For Example 1 (using letters as superscripts),* $\mathcal{M}^a = \{\{a, b\}, \{a, d, e\}\}$ *and* $\mathcal{M}^c = \{\{b, c\}, \{c, f\}\}$. *Hence* $\mathcal{M}^* = \{\{b\}, \{b, f\}, \{b, d, e\}, \{d, e, f\}\}$ *and* $min(\mathcal{M}^*) = \{\{b\}, \{d, e, f\}\}$.

Lemma 4.5 $min(\mathcal{M}^*) = \mathcal{MM}_{DB^-} - \mathcal{MM}_{DB}$.

PROOF: If $M \in \mathcal{M}^*$, then M is a model of DB^- and $M \cap C_0 = \emptyset$, hence M is not a model of DB. Therefore, it suffices to show that if $M \in \mathcal{MM}_{DB^-} - \mathcal{MM}_{DB}$ then $M \in \mathcal{M}^*$. So let $M \in \mathcal{MM}_{DB^-} - \mathcal{MM}_{DB}$. It follows from the comment after Observation 4.2 that there are two cases:

1. $\forall a_i \in C_0$, $M \cup \{a_i\} \in \mathcal{MM}_{DB}$. In this case $M \cup \{a_i\} \in \mathcal{M}^i$, $1 \leq i \leq m$, hence $M \in \mathcal{M}^*$.
2. By Lemma 4.1, $\forall a_i \in C_0$ $\exists M_i \subset M$ $s.t.$ $M_i \cup \{a_i\} \in \mathcal{MM}_{DB^-}$. Therefore, $M_i \cup \{a_i\} \in \mathcal{MM}_{DB}$. Hence $M = \bigcup_{i=1}^m M_i \in \mathcal{M}^*$.

$\square$

Lemmas 4.4 and 4.5 provide two different characterizations of $\mathcal{MM}_{DB^-} - \mathcal{MM}_{DB}$, the first one is obtained by modifying the database while the second involves only manipulation of minimal models. Lemma 4.5 suggests another algorithm to find $\mathcal{MM}_{DB^-}$: compute $\mathcal{M}^*$ and minimize $\mathcal{MM}_{DB} \cup \mathcal{M}^*$. As we will show later, this algorithm is also more complex than the algorithm given earlier.

The **Deletion** algorithm presented earlier is superior to the brute force approach of building a new model tree for DB^- from scratch in two ways. First, a substantial part of the tree structure together with all auxiliary tools (such as indexing schemes) for the new tree is already stored in secondary memory. Hence, all that needs to be done is *modification*, not *creation* of both the tree structure and associated tools. Second, the amount of computation required to find out which models have to be added to the tree and which atoms can be deleted from it is substantially less than the brute force approach. The average case analysis of the complexity of the algorithm is hard to present since the result depends on too many parameters of the database (distribution of atoms of C_0 among clauses of DB^-, size of the Herbrand base, etc.). Hence, we show only that in the worst case the complexity of the **Deletion** algorithm is still better than recomputing minimal models of DB^- from scratch.

Assume that the database DB^- contains n clauses and that the average clause size is m. Clearly, the number of all models before minimization is m^n. The creation of a new tree involves minimization which amounts to $O(m^{2n})$ operations (each model needs to be compared with all other models), where the unit of operation is comparison of two models. **Deletion** involves two distinct types of operations: (a) the computation of $\mathcal{MM}_{DB^0}$; (b) checking whether all clauses of $\mathcal{C}^{a_i}$ are true in some modified models of DB (line 4 of the algorithm).

(a) To have the largest number of models of DB^0, the distribution of the atoms of C_0 should be such that one atom of C_0 occurs in exactly one clause of DB. In that case there are m clauses containing $m-1$ atoms (these are the clauses with the atoms of C_0 removed from them); the remaining $n-m$ clauses of DB are unchanged. Then, there are $m^{n-m} * (m-1)^m$ models of DB^0, minimization of which requires $[m^{n-m} * (m-1)^m]^2$ operations.

(b) Since DB contains n clauses each with m atoms, there are $m * n$ atoms (not necessarily distinct) in DB. Thus each $\mathcal{C}^{a_i}$ can contain at most $m * n$ atoms. Since we defined the unit of operation as the comparison of two sets (models) each of which can contain up to n atoms, line 4 of the algorithm which is, roughly speaking, a comparison of a set with $n-1$ atoms (a model) with a set of $m * n$ atoms (the atoms of $\mathcal{C}^{a_i}$) will take less than m such operations for each model. Hence, for the entire set of minimal models of DB, the execution of lines 2-5 will take $m^n * m = m^{n+1}$ operations.

The total number of operations is then equal to:

$[m^{n-m} * (m-1)^m]^2 + m^{n+1} =$

$[m^{2n-2m} * (m-1)^{2m}] + m^{n+1} =$

$m^{2n}[\frac{(m-1)^{2m}}{m^{2m}} + \frac{1}{m^{n-1}}] < m^{2n}$, for all $m, n > 1$.

Note that if the algorithm simply implemented the suggestion following Lemma 4.4, its complexity would be substantially higher: it would involve not only the construction and minimization of $\mathcal{MM}_{DB^0}$ (which the **Deletion** algorithm also does) but also the minimization of $\mathcal{MM}_{DB} \cup \mathcal{MM}_{DB^0}$. Even though it is not a complete minimization i.e. the models of $\mathcal{MM}_{DB}$ need to be compared only with the models of $\mathcal{MM}_{DB^0}$, it still requires $m^{n+1} * m^{n-m} * (m-1)^m$ (i.e. the number of elements of $\mathcal{MM}_{DB}$ times the number of elements of $\mathcal{MM}_{DB^0}$) operations. This value is higher than the value we obtained in (b) above.

We conclude this analysis by noting that the deletion algorithm based purely on minimal models (i.e. the one computing $\mathcal{M}^*$) is also worse than the **Deletion** algorithm. If the atoms of C_0 are evenly distributed among the models of DB and each such model contains exactly one atom of C_0, then each $\mathcal{M}^i$ has $\frac{1}{m} * m^n = m^{n-1}$ elements. $\mathcal{M}^*$, which is the set of all combinations of unions of these models, thus has $(m^{n-1})^m$ elements. Clearly, minimization of $\mathcal{M}^*$ is of order $O(((m^{n-1})^m)^2)$, which is already higher (for $n > m > 1$) than the cost of constructing $\mathcal{MM}_{DB^-}$ from DB^-.

5 Extensions

5.1 Model tree split after deletion

As stated in Section 2, clauses belonging to separate clusters should have their models represented in separate trees. Thus, whenever a deletion divides a database into clusters, the original tree should be split.

For the algorithm below, we assume that after the deletion of C, the database DB is split into two separate clusters DB_1 and DB_2. We also assume that this information, including the clauses in DB_1 and DB_2, is known to us *before* the execution of the deletion. The algorithm yields a model tree for DB_1; it can be modified to yield a model tree for DB_2 as well.

We start with definitions about splitability and an observation about the *min* operator (defined in Section 2). We write *At(DB)* for the set of atoms in the clauses of DB. We call DB *C-splitable into* DB_1 *and* DB_2 if $DB = DB_1 \cup \{C\} \cup DB_2$ and $At(DB_1) \cap At(DB_2) = \emptyset$.

Observation 5.1 *If* $min(T) \subseteq S \subseteq T$*, then* $min(S) = min(T)$.

Now we show the connections between $\mathcal{MM}_{DB_1}$, $\mathcal{MM}_{DB_2}$, $and \mathcal{MM}_{DB}$.

Lemma 5.1 *Suppose that* DB *is C-splittable into* DB_1 *and* DB_2*. Then*
(a) $\mathcal{MM}_{DB_1} = min(\{M \cap At(DB_1) | M \in \mathcal{MM}_{DB}\})$
(b) $\mathcal{MM}_{DB_2} = min(\{M \cap At(DB_2) | M \in \mathcal{MM}_{DB}\})$
(c) $\mathcal{MM}_{DB} = min(\{M_1 \cup M_2 | M_1 \in \mathcal{MM}_{DB_1},\ M_2 \in \mathcal{MM}_{DB_2},\ \exists a \in C,\ a \in M_1 \cup M_2\} \cup \{M_1 \cup \{a\} \cup M_2 | M_1 \in \mathcal{MM}_{DB_1},\ M_2 \in \mathcal{MM}_{DB_2},\ a \in C,\ (M_1 \cup M_2) \cap C = \emptyset\})$

PROOF
(a) We wish to use Observation 5.1 with $T = Mod(DB_1)$ and $S = \{M \cap At(DB_1) | M \in \mathcal{MM}_{DB}\}$. So we must show that $\mathcal{MM}_{DB_1} \subseteq \{M \cap At(DB_1) | M \in \mathcal{MM}_{DB}\} \subseteq Mod(DB_1)$. The second inclusion, $S \subseteq Mod(DB_1)$. is clear from the definitions. So we need to show that if $M_1 \in \mathcal{MM}_{DB_1}$, then $M_1 \in S$. There are two cases:
(1) $M_1 \cap C \neq \emptyset$. Let $M_2 \in \mathcal{MM}_{DB_2}$ such that $M_2 \cap C = \emptyset$ Then $M_1 \cup M_2 \in \mathcal{MM}_{DB}$ and $M_1 = (M_1 \cup M_2) \cap At(DB_1)$, hence $M_1 \in S$.
(2) $M_1 \cap C = \emptyset$. Let $M_2 \in \mathcal{MM}_{DB_2}$ such that $M_2 \cap C \neq \emptyset$ Again, $M_1 \cup M_2 \in \mathcal{MM}_{DB}$ and $M_1 = (M_1 \cup M_2) \cap At(DB_1)$, hence $M_1 \in S$.

(b) Similar to (a) with DB_1 and DB_2 reversed.

(c) We wish to use Observation 5.1 with $T = Mod(DB)$ and
$S = \{M_1 \cup M_2 | M_1 \in \mathcal{MM}_{DB_1},\ M_2 \in \mathcal{MM}_{DB_2},\ \exists a \in C,\ a \in M_1 \cup M_2\} \cup \{M_1 \cup \{a\} \cup M_2 | M_1 \in \mathcal{MM}_{DB_1},\ M_2 \in \mathcal{MM}_{DB_2},\ a \in C,\ (M_1 \cup M_2) \cap C = \emptyset\}$.
So we must show that $\mathcal{MM}_{DB} \subseteq S \subseteq Mod(DB)$. The second inclusion, $S \subseteq Mod(DB)$ is clear from the definitions. Thus, we need to show that if $M \in \mathcal{MM}_{DB}$, then $M \in S$. There are four cases:
(1) $M \cap At(DB_1) \cap C \neq \emptyset$ and $M \cap At(DB_2) \cap C \neq \emptyset$. Let $M_1 = M \cap At(DB_1)$ and $M_2 = M \cap At(DB_2)$. Then $M_1 \in \mathcal{MM}_{DB_1}$ and $M_2 \in \mathcal{MM}_{DB_2}$. Also, $\exists a \in C,\ a \in M_1 \cup M_2$ and $M = M_1 \cup M_2$. Hence $M \in S$.

(2) $M \cap At(DB_1) \cap C \neq \emptyset$ and $M \cap At(DB_2) \cap C = \emptyset$. Let $M_2 = M \cap At(DB_2)$. Then $M_2 \in \mathcal{MM}_{DB_2}$. If $M \cap At(DB_1) \in \mathcal{MM}_{DB_1}$, then the rest of the proof is as in case (1). If $M \cap At(DB_1) \notin \mathcal{MM}_{DB_1}$, then $\exists a \in (C \cap M \cap At(DB_1))$ s.t. $(M \cap At(DB_1)) - \{a\} \in \mathcal{MM}_{DB_1}$. Let $M_1 = (M \cap At(DB_1)) - \{a\}$. Then $M = M_1 \cup \{a\} \cup M_2$, hence $M \in S$.

(3) Similar to (2) with DB_1 and DB_2 reversed.

(4) $M \cap At(DB_1) \cap C = \emptyset$ and $M \cap At(DB_2) \cap C = \emptyset$. This is possible only if $\exists a \in C$ s.t. $a \notin At(DB_1) \cup At(DB_2)$. Let $M_1 = M \cap At(DB_1)$ and $M_2 = M \cap At(DB_2)$. Then $M \in \mathcal{MM}_{DB_1}$ and $M_2 \in \mathcal{MM}_{DB_2}$. So $M = M_1 \cup \{a\} \cup M_2$, hence $M \in S$.

□

This lemma shows that the minimal models of DB_1 are obtained from the minimal models of DB by restricting them to the atoms of DB_1 and that the minimal models of DB are obtained from the unions of minimal models of DB_1 and DB_2 with an atom of C, if needed. The lemma suggests the following algorithm for obtaining $\mathcal{MM}_{DB}$: traverse the tree for $\mathcal{MM}_{DB}$, eliminate from each branch the atoms not in DB_1, and then minimize. Our algorithm presented below, is more efficient. It is based on the following Corollary to Lemma 5.1.

Corollary 5.1 *Suppose that DB is C-splitable into DB_1 and DB_2, $M_2 \in \mathcal{MM}_{DB_2}$ such that $M_2 \cap C \neq \emptyset$ and $M_1 \subseteq At(DB_1)$. Then $M_1 \in \mathcal{MM}_{DB_1}$ iff $M_1 \cup M_2 \in \mathcal{MM}_{DB}$.*

It follows from this result that to obtain $\mathcal{MM}_{DB_1}$ it is not necessary to traverse all the branches of $\mathcal{MM}_{DB}$. It suffices to traverse only those branches that contain an arbitrarily chosen $M_2 \in \mathcal{MM}_{DB_2}$ such that $M_2 \cap C \neq \emptyset$ and to delete the atoms of DB_2 from them. By Lemma 5.1 such an M_2 can be obtained from $M \in \mathcal{MM}_{DB}$ as $M_2 = M \cap At(DB_2)$ as long as $M \cap At(DB_2) \cap C \neq \emptyset$.

Procedure **Delete_and_Split** *($\mathcal{T}_{DB}$, DB_1, DB_2, C) returns $\mathcal{T}_{DB_1}$.*

1. $\mathcal{T}_{DB_1} := \emptyset$
2. Find $M_2 \in \mathcal{MM}_{DB_2}$ such that $M_2 \cap C \neq \emptyset$
3. For each branch Br of $\mathcal{T}_{DB}$

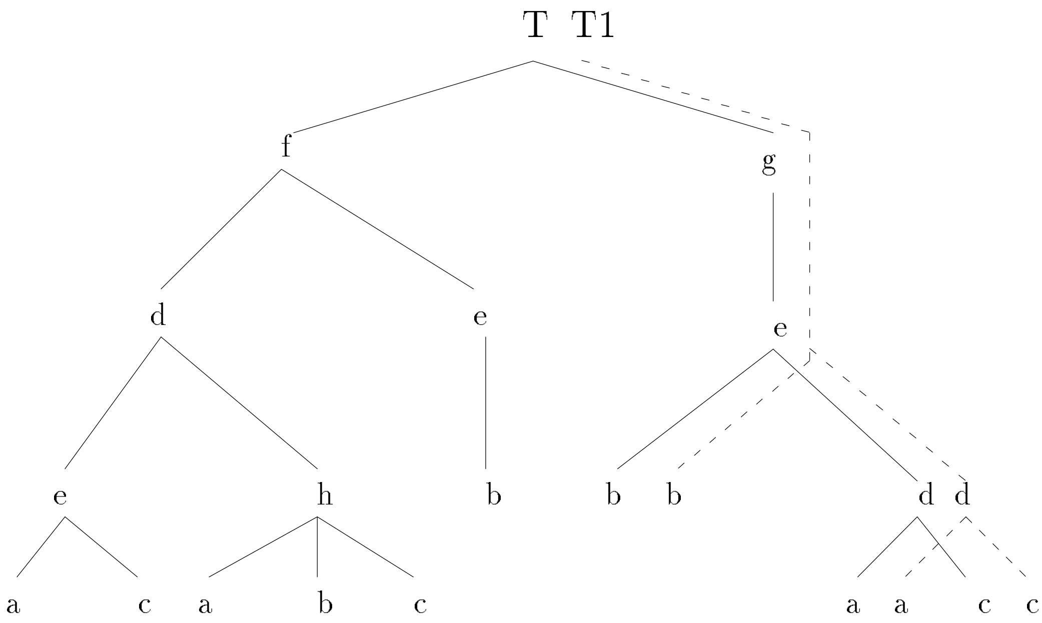

Figure 4: The model tree of DB; broken lines indicate the model tree for DB_1 returned by the **Delete_and_Split** procedure (for $M_2 = \{g, e\}$).

4. If $M_2 \subset Br$, then
5. $Br := Br - M_2$
6. $\mathcal{T}_{DB_1} := \mathcal{T}_{DB_1} \cup Br$

Example 8 *Let DB be the database containing DB_1' and DB_2' as described in Example 2 and shown in Figure 2 and an additional clause $C = (d \vee e)$ (to be deleted). If the clauses are inserted in the following order: $f \vee g, e \vee f, d \vee e, b \vee d, e \vee h, a \vee b \vee c$, then the model tree for DB is the one in Figure 4.*

If the algorithm is executed with respect to DB_1' and $M_2 = \{g, e\}$, then the returned tree would look as shown by the broken lines in the figure. All the branches to the left of the root are removed since none of them contains M_2. For the remaining branches the condition of line 4 is satisfied, hence g and e are removed.

The correctness of the Procedure **Delete_and_Split** follows from Corollary 5.1 and the remark after it.

If we assume that the number of minimal models of DB is m^{2r+1} (where m is the average clause size and $2r + 1$ is the number of clauses in DB), then the complexity of the algorithm cannot be shown to be smaller than reconstructing the two trees from DB_1 and DB_2. Assuming for simplicity that both DB_1 and DB_2 have r clauses each, then since each of them has m^r models, their minimization takes $2 * (m^{2r})$ operations (we take comparison of two models of either DB_1 or DB_2 to be the unit of operation). On the other hand, checking the condition of line 4 of the algorithm takes 2 unit operations (models of DB can be twice as large as models of DB_1 or DB_2), which with the assumption of the existence of m^{2r+1} minimal models gives $2 * m^{2r+1}$ operations.

The efficiency of the algorithm can be improved easily by using an appropriate indexing scheme that we assume exists for any implementation of model trees. Such a scheme would provide a mapping from atoms in the Herbrand base of the database to the appropriate models, indicating in which branches a given atom occurs. Then, the appropriate models for which M_2 (line 4 of the algorithm) is a subset can be computed in time smaller than required for the traversal of the entire tree (the exact figure is implementation dependent). Another advantage of the algorithm (similarly to the **Deletion** algorithm) is the possibility of inheriting the tree structure and indexing scheme for the smaller cluster from the tree before the deletion.

We conclude this section by noting that a procedure reverse to **Delete_and_Split** can also be implemented. Thus, when a clause containing atoms from separate model trees is inserted into a database, those trees need to be merged. Such an **Insert_and_Merge** algorithm may require attaching the entire tree of one cluster of the database to every leaf of the tree on another cluster then inserting the clause and minimizing the resulting tree. Clearly, the first step needs to be repeated for merging more than two trees. Various optimizations of this procedure are possible.

5.2 Other Types of Deletion

We can distinguish at least two more types of deletion in disjunctive databases.

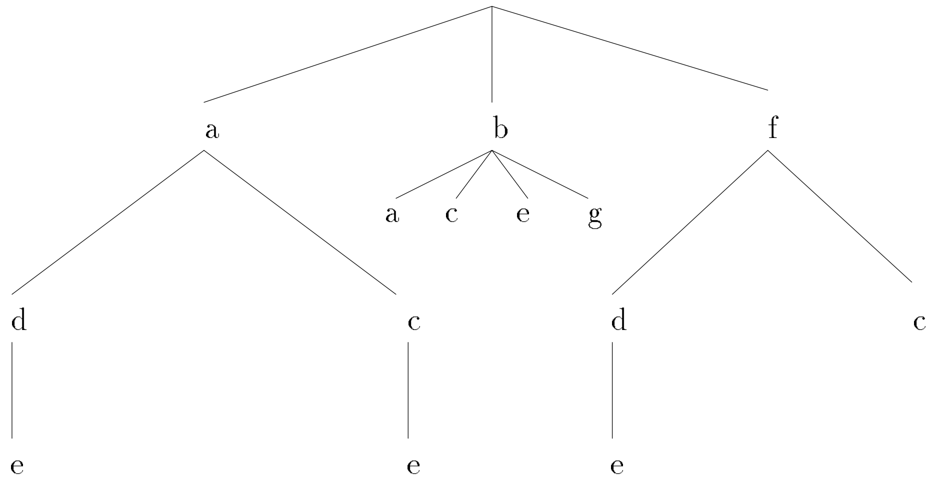

Figure 5: Model tree representation for DB^*

1. Deletion of a disjunction together with some of its consequences by means of deleting one of its atoms from an arbitrarily chosen model. Since this type of deletion has been studied by Yahya in [6], we will not consider it further.

2. Deletion of a disjunction together with all of its consequences with the subsequent insertion of subsumed clauses - to be explained below

The second type of deletion applies only to databases where all the information ever acquired is retained, even if it is not stored directly in model trees. For example, if $DB^* = \{a \vee c \vee e \vee g, a \vee b \vee f, b \vee c \vee d, b \vee c \vee e, b \vee e \vee f\}$, and a new clause $a \vee c$ is inserted, then ordinarily $a \vee c \vee e \vee g$ is removed (since it is subsumed by the inserted clause) and the resulting database is identical to the one in Example 1. But, if $a \vee c \vee e \vee g$ is saved separately, then a future deletion of $a \vee c$ should prompt the insertion of $a \vee c \vee e \vee g$ and thereby the recovery of DB^*. The model tree for DB^* is shown in Figure 5. We refer to a database, like DB^*, that results from this type of deletion as $DB^{+/-}$.

The algorithm for updating a model tree in the case of such a deletion is presented below; it is a fusion of the **Deletion** and the **Insertion** algorithm (hence the proof of its correctness is omitted here). All the notation is consistent with that in the **Deletion** and **Insertion** algorithms. The algorithm is traced on Example 1.

Let DB be a database, $C_0 = [a_1, ..., a_k]$ be a clause to be deleted, $C_1 = [a_1, ..., a_k, ...a_m]$ be a clause to be added. $\mathcal{C}^{a_i}$ and $\mathcal{MM}_{DB^0}$ are defined in the same way as in the **Deletion** algorithm (with C_0 as the deleted clause).

Procedure **SemiDeletion** *(DB, C_0, C_1) returns* $\mathcal{MM}_{SemDel}$.

1. Construct DB_0 and $\mathcal{MM}_{DB^0}$
2. For each model $M \in \mathcal{MM}_{DB}$
3. If M contains exactly one $a_i \in C_0$
4. If $\forall C \in \mathcal{C}^{a_i}, M - \{a_i\} \models C$, then
5. $\mathcal{MM}_{DB} := \mathcal{MM}_{DB} - M$
6. $\mathcal{MM}_{SemDel} := \mathcal{MM}_{DB} \cup$ **Insertion** ($\mathcal{MM}_{DB^0}$, C_1)
7. Minimize $\mathcal{MM}_{SemDel}$

Lines 2-5 are exactly the same as in the **Deletion** algorithm and they serve the same purpose as described there. The only models after the **Deletion** is executed that may not be models of the database $DB^{+/-}$ are the models of DB^0. Hence **Insertion** is executed only on those models.

Example 9 *Let DB be the database described in Example 1 and let $a \vee c \vee e \vee g$ be the clause that needs to be added to the model tree after the deletion of $a \vee c$. The execution of lines 2-5 of the algorithm removes the models $\{a, b\}$ and $\{b, c\}$ from the model tree. Since $\mathcal{MM}_{DB^0}$ contains the models $\{\{b\}, \{d, e, f\}\}$, inserting $a \vee c \vee e \vee g$ in line 9 produces the set $\{\{b, a\}, \{b, c\}, \{b, e\}, \{b, g\}, \{d, e, f\}\}$ which is subsequently added to the tree. The minimization of line 7 does not change the set of minimal models.*

Yet another type of update that can be executed on a database is the insertion of a negated atom. Since a database does not store negative information explicitly, inserting a negated atom can only mean modifying information already in the database. In the case of disjunctive databases, inserting a negated atom means modifying one (or more) of the disjunctions. For example, if the database contains the disjunction $a \vee b \vee c$ and we find out that a is false, then $a \vee b \vee c$ needs to be removed and $b \vee c$ should be inserted. This is precisely the way we understand the insertion of negative information: inserting $\neg a$ into a database means removing all the clauses containing a and also inserting those same clauses with a removed from all of them. Since this procedure is the reverse of **SemiDeletion** we call it **SemiInsertion**. It turns out that **SemiInsertion** can be implemented in a very simple way, by removing all the models containing an atom whose negation is inserted. The following lemma states this fact formally.

Lemma 5.2 *Let DB' be a database DB with a deleted from all the clauses, i.e. $DB' = \{C | C = C' - \{a\}$ and $C' \in DB\}$. Then, $\mathcal{MM}_{DB'} = \mathcal{MM}_{DB} - \{M \in \mathcal{MM}_{DB} | a \in M\}$*

PROOF

($\subseteq$) Clearly, $\mathcal{MM}_{DB'} \subseteq \mathcal{MM}_{DB}$. Also, $\mathcal{MM}_{DB'}$ cannot contain any model M s.t. $a \in M$ (since a does not occur in any of the clauses of DB').

($\supseteq$) Let $M \in \mathcal{MM}_{DB}$ such that $a \notin M$. Clearly, M is a model of DB'. Since $\mathcal{MM}_{DB'} \subseteq \mathcal{MM}_{DB}$, $M \in \mathcal{MM}_{DB'}$.

6 Summary and Future Work

We have considered the update problem for disjunctive databases. We have given algorithms for the insertion of a clause into and the deletion of a clause from a disjunctive database. We have also shown the correctness of these algorithms. While our work is oriented towards the representation of disjunctive databases by model trees, a compact representation, the algorithms apply to any representation of disjunctive databases in terms of minimal models. Putting our results together with the solution of the view update problem presented in [5] and [3] in terms of updating the underlying disjunctive database, we now have a complete solution of the view update problem for various classes of normal deductive disjunctive databases.

We plan to develop an implementation of deductive disjunctive databases using the model tree representation technique. In this implementation we plan to use the algorithms given in this paper to implement the update operations. Also, by combining the algorithms here with the algorithms in [5] and [3], we will be able to implement view updates for disjunctive databases.

References

[1] J.A. Fernández, J. Minker, and A. Yahya. Computing perfect and stable models using ordered model trees. Technical Report CS-TR-3195 and UMIACS-TR-93-136, University of Maryland Institute for Advanced Computer Studies, College Park, MD 20742, December 1993. Submitted to the Journal of Computational Intelligence.

[2] José Alberto Fernández. *Disjunctive Deductive Databases*. PhD thesis, University of Maryland, Department of Computer Science, College Park, 1994.

[3] José Alberto Fernández, John Grant, and Jack Minker. Model theoretic approach to view updates in deductive databases. Technical Report CS–TR–3335, Department of Computer Science, University of Maryland, College Park, MD 20742, 1994.

[4] José Alberto Fernández and Jack Minker. Bottom-up evaluation of Hierarchical Disjunctive Deductive Databases. In Koichi Furukawa, editor, *Logic Programming Proceedings of the Eighth International Conference*, pages 660–675. MIT Press, 1991.

[5] J. Grant, J. Horty, J. Lobo, and J. Minker. View updates in stratified disjunctive databases. *Journal of Automated Reasoning*, 11:249–267, 1993.

[6] A. Yahya. Notes on updates in logic databases. (unpublished).