Representations for Disjunctive Deductive Databases

Adnan Yahya[1,3*] Jack Minker[1,2]

[1]Institute for Advanced Computer Studies

[2]Computer Science Department
University of Maryland
College Park, MD 20742

[3]Electrical Engineering Department
Birzeit University
Birzeit, West Bank

Abstract

The concepts of the *dual* and *complementary* databases as alternative representations for a given disjunctive deductive database, DB, are introduced. The *dual* database results from interchanging the occurrences of OR and AND operators in the positive clause representation of DB. The *complementary* database is defined to have as minimal models the complements of the minimal models of DB relative to the underlying Herbrand base, HB_{DB}. The properties of these derivative databases are studied. In particular, we show that the minimal models of DB correspond to the minimally derivable clauses of the *dual* database and that the *complementary* database defines the *Extended Generalized Closed World Assumption* ($EGCWA$) *completion*, and consequently the *Generalized Closed World Assumption* ($GCWA$) *completion*, of DB. A tree structure for minimally derivable clauses is defined and algorithms for constructing and performing update and search operations on such trees are given.

1 Introduction

Minimal models and minimally derivable clauses are basic concepts in much of the work on disjunctive deductive databases ($DDDBs$). Either of these forms can be used to represent a $DDDB$ [16, 21]. Equivalent semantic (in terms of minimal models), and syntactic (in terms of minimal derivations) definitions were given for the *Generalized Closed World Assumption* ($GCWA$) [13] and the *Extended Generalized Closed World Assumption* ($EGCWA$) [21] that are used to assume negative information in $DDDBs$. These negation rules served to extend the *Closed World Assumption*, originally introduced to deal with negative information in Horn databases, to the non-Horn case [13, 17, 21].

*This work was done while visiting at the University of Maryland Institute for Advanced Computer Studies.

Model trees and ordered model trees were used as structure sharing approaches to represent information in $DDDBs$ [7, 20]. Informally, a model tree for a $DDDB$, DB, is a tree structure in which nodes are labeled by atoms of the Herbrand base of DB. With each branch (i.e. path from the root to a leaf node) we associate the set of atoms encountered on that branch. Branches represent models of DB. A minimal model tree is a model tree for which there is a one to one correspondence between the minimal models of DB and the tree branches. Algorithms for constructing and updating model trees as well as query answering algorithms operating on model trees were presented in [7] and [20]. Whereas there may be many different model trees for a given disjunctive database, there can be only one ordered model tree. Ordered model trees were introduced as a normal form for disjunctive deductive databases. They can also be used to facilitate the computation of minimal model trees and answering queries for disjunctive theories by exploiting an order on the Herbrand base of the theory. There are several criteria that may be used to select an ordering of atoms of the Herbrand base, such as the length of a clause in which the atom appears, the frequency of occurrence of the atom in the database, as well as other criteria. We do not discuss this topic in this paper.

In this paper, for a given disjunctive deductive database, DB, we define its *dual* database, DB^d, and its *complementary* databases, DB^c. The *dual* database is constructed by interchanging the occurrences of logical OR and logical AND operations in the positive clause representation of DB. The *complementary* database is constructed by complementing the minimal models of DB relative to the Herbrand base of DB. We study the properties of these derivative databases. In particular, we show that the minimal models of DB correspond to the minimally derivable clauses of its dual database and that the complementary database defines the $EGCWA$ *completion*, and consequently the $GCWA$ *completion*, of DB. A tree structure for minimally derivable clauses is defined and algorithms for performing update and search operations on such trees are given.

The subject matter of disjunctive deductive databases is concerned with incomplete information. Such information arises in every day experience. We know only a portion of the world and not all of the facts. Thus, we may know that a certain person is a scientist, but we do not know for sure which field the individual is in. We may know that the person is either a computer scientist or an engineer. In a medical database, it may be known that a person may have one or more diseases, but we do not know the precise disease except that it may be one of several different diseases. In analyzing images from space, a certain object observed may be one of several possibilities. Current relational and deductive database technologies do not adequately handle such data. In addition, it has been shown that many problems that deal with nonmonotonic and commonsense reasoning can be translated into disjunctive deductive databases [15]. It will be necessary to be able to understand the meaning of disjunctive databases and how to respond to queries posed to them. A great deal is known, however, about the semantics of disjunctive deductive databases [12, 5]. In this paper we address alternative possible representations of such databases. There have been a number of papers devoted to disjunctive deductive databases, see [5, 4, 6, 10, 11, 9, 14, 19] for a number of such papers. A survey of the work described in many of these papers is contained in [12, 5].

The paper is organized as follows. In section 2 we give some definitions and background material related to disjunctive deductive databases. In section 3 we define the *dual* database, investigate its properties and give algorithms for the construction and update of minimal clause trees for positive $DDDBs$ and $DDDBs$ with rules. In section 4 we define the *complementary* database and study its properties. We conclude by discussing the possibilities for utilizing the reported results and the possible directions for future research.

2 Notation and Background

A *disjunctive deductive database (DDDB)* DB is a set of clauses of the form:

$$A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_n$$

where $n \geq 0, k > 0$ and the A_i and B_j are atoms, defined using a FOL $\mathcal{L}$ that does not contain function symbols. In this work we will assume that all clauses in DB are ground.

If $n = 0$ for all clauses in DB, the database is called a *positive disjunctive database*. Otherwise DB is said to contain rules. If $k = 1$ for all clauses in DB, the database is called a *a Horn deductive database*. If $k > 1$ for some clauses in DB, the database is called a *a non-Horn or a disjunctive deductive database*.

Where no confusion arises, a clause C will be referred to either as the set of its constituent literals or as the disjunction of the elements of this set. So, the above clause will be referred to as the set

$$\{A_1, \cdots, A_k, \neg B_1, \cdots, \neg B_n\}.$$

or as the disjunction

$$A_1 \vee \cdots \vee A_k \vee \neg B_1 \vee \cdots \vee \neg B_n.$$

Definition 2.1 *Let A be a ground atom, let DB be a $DDDB$ and let C be a clause. Then*

- *A occurs* positively *(*negatively*) in C iff A ($\neg A$) is a literal of C.*
- *A occurs* positively *(*negatively*) in DB iff A occurs* positively *(*negatively*) in a clause of DB.*
- *A occurs in C (DB) iff A occurs* positively *or* negatively *in C (DB).*
- *A is* purely positive *(*purely negative*) in DB iff A occurs only positively (negatively) in DB.*

Given a $DDDB$, DB, the Herbrand base of DB, HB_{DB}, is the set of all ground atoms that can be formed using the predicate symbols and constants in the FOL $\mathcal{L}$. A *Herbrand interpretation* is any subset of HB_{DB}. A Herbrand model of DB, M, is a Herbrand interpretation such that $M \models DB$ (i.e. all clauses of DB are *true* in M). M is minimal if no proper subset of M is a model of DB.

Definition 2.2 *By $\mathcal{M}(DB)$ we denote the set of all models (not necessarily minimal) of DB and by $\mathcal{MM}(DB)$ we denote the set of all minimal models of DB.*

$\mathcal{M} = \{M : M \models DB\}$

$\mathcal{MM}(DB) = \{M : M \in \mathcal{M}(DB)$ *and* $\forall M' \subset M, M' \notin \mathcal{M}(DB)\}$

Clearly $\mathcal{MM}(DB) \subseteq \mathcal{M}(DB)$. In addition for every element $M \in \mathcal{M}(DB)\, \exists M' \in \mathcal{MM}(DB)$: $M' \subseteq M$.

Definition 2.3 *A positive clause (possibly non-ground) $\exists Q(\vec{x})$ is called a* query *and is considered a* logical consequence *of a $DDDB$, DB, iff there exists a set of ground substitutions $\{\theta_1, \ldots, \theta_n\}$ such that $Q(\vec{x})\theta_1 \vee \cdots \vee Q(\vec{x})\theta_n$ is* true *in all* minimal Herbrand models *of DB, $\mathcal{MM}(DB)$. $Q(\vec{x})\theta_1 \vee \cdots \vee Q(\vec{x})\theta_n$ or equivalently $\{\theta_1 + \ldots + \theta_n\}$ is said to be an answer to $\exists Q$. We also write $Q(\vec{x})\theta_1 \vee \cdots \vee Q(\vec{x})\theta_n$ is an answer to $Q(\vec{x})$.*

The notation used in Definition 2.3 follows that used by Reiter [17]. In general, a query may be a disjunct of conjuncts. If the query is $Q(\vec{x}) = Q_1(\vec{x}) \vee \ldots \vee Q_k(\vec{x})$, and each of the $Q_i(\vec{x})$, $i = 1 \ldots k$ are conjuncts, we may replace the query by $Q(\vec{x}) = q_1(\vec{x}) \vee \ldots \vee q_k(\vec{x})$ and enter the rules:

$$q_1(\vec{x}) \leftarrow Q_1(\vec{x})$$
$$\vdots$$
$$q_k(\vec{x}) \leftarrow Q_k(\vec{x})$$

in the database.

As noted in the definition of a query, predicates may have arbitrary arity. We limit the examples to monadic predicates for simplicity.

Example 1 *Let* $DB = \{P(a) \vee P(b), P(c) \vee P(d)\}$ *and let* $Q(x) = \exists(P(x))$. *Then* $x = \{a + b\}$ *and* $x = \{c + d\}$ *are answers to* $Q(x)$.

Definition 2.4 *Let* DB_1 *and* DB_2 *be disjunctive deductive databases. Then*

$$DB_1 =_{mm} DB_2 \text{ iff } \mathcal{MM}(DB_1) = \mathcal{MM}(DB_2)$$

DB_1 *and* DB_2 *are said to be* minimal model equivalent.

Clearly $(=_{mm})$ is an equivalence relation.

Since most of the properties discussed in this paper are based on minimal model concepts the terms *equivalent* and *minimal model equivalent* are used interchangeably. However, databases with different syntax may be minimal model equivalent:

Example 2 $DB_1 = \{P(a) \vee P(b), P(c) \vee P(d), \leftarrow P(a) \wedge P(c)\}$.
and let
$DB_2 = \{P(a) \vee P(b), P(c) \vee P(d), P(b) \vee P(d)\}$.

$$DB_1 =_{mm} DB_2.$$

Definition 2.5 *Let* DB *be a disjunctive deductive database. By* $\mathcal{S}(DB)$ *we denote the set of positive clauses derivable from* DB. *That is,* $\mathcal{S}(DB) = \{C : C$ *is positive and* $DB \vdash C$, *(* C *is* true *in every minimal model of* $DB)\}$. *Using the terminology of [16],* $\mathcal{S}(DB)$ *is a model state of* DB.

Definition 2.6 *Let* DB *be a disjunctive deductive database. By* $\mathcal{MS}(DB)$ *we denote the set of positive clauses minimally derivable from* DB. *That is,* $\mathcal{MS}(DB) = \{C : C$ *is positive and* $DB \vdash C$ *and* $\forall\ C' \subset C,\ DB \nvdash C'\}$. $\mathcal{MS}(DB)$ *is the* minimal model state *of* DB *[16].*

Clearly, for a given database, DB, $\mathcal{MS}(DB) \subseteq \mathcal{S}(DB)$. In addition $\forall C \in \mathcal{S}(DB)\ \exists C' \in \mathcal{MS}(DB)$ such that $C' \subseteq C$.

Lemma 1 *[13, 16, 21] Let* DB *be a disjunctive deductive database.* M *is a model (minimal model) for* DB *if and only if* M *is a model (minimal model) for* $\mathcal{MS}(DB)$.

Corollary 1 *Let DB_1 and DB_2 be DDDBs. Then:*

$$DB_1 =_{mm} DB_2 \ iff \ \mathcal{MS}(DB_1) = \mathcal{MS}(DB_2).$$

Lemma 2 *Let DB be a disjunctive deductive database. Then [13, 21]:*

- *A clause $C \in \mathcal{MS}(DB)$ iff $DB \vdash C$ and $\forall C' \subset C$, C' is falsified by at least one minimal model of DB.*
- *If a clause $C = A_1 \vee A_2 \vee \cdots \vee A_n \in \mathcal{MS}(DB)$ then $\forall i \in \{1, 2, ..., n\}$, $\exists\ M \in \mathcal{MM}(DB)$ such that $A_i \in M$ and $\forall_{j \neq i},\ A_j \notin M$.*

Definition 2.7 *[20] Let DB be a ground disjunctive deductive database. A* cluster *$\mathcal{C}$ is a subset of DB such that if a clause $C \in \mathcal{C}$ and C contains an occurrence of atom A then every clause of DB that contains an occurrence of A is also in $\mathcal{C}$.*

A cluster is *minimal* if no proper subset of it is a cluster. Any nonminimal cluster is the union of more than one minimal cluster. In addition, minimal clusters are *disjoint*. No ground atom can occur in two minimal clusters. Therefore, a minimal cluster can be referred to by one of the clauses or atoms occurring in it [22].

Since clauses with pure negative literals do not contribute to the minimal models of a database, we can remove these clauses and obtain a minimal model equivalent database [1].

We assume that a $DDDB$, DB is divided into two different sets of clauses: the *extensional part (E_{DB})* and the *intensional part (I_{DB})*. It is easy to see that any $DDDB$ that has predicates that may be both intensional and extensional can be transformed into one in which the predicates are either entirely extensional or entirely intensional. The division is such that:

1. E_{DB} is a positive disjunctive database.
2. $\forall A$ that occurs in E_{DB}, A occurs purely negatively or does not occur at all in I_{DB}.
3. E_{DB} is the largest positive disjunctive database that fulfills conditions 1–3.

The extensional part of DB can be equated to base relations of a relational database, it is the place where the information is stored. The intensional part can be equated to view definitions.

Model Trees: Given that the meaning of a $DDDB$ is defined by the set of its minimal Herbrand models, Fernández and Minker [7] defined an abstract data structure, called the *model tree*, to represent the minimal models of the $DDDB$. They provide algorithms for the computation of queries using model trees, where the minimal models of the $DDDB$ are computed incrementally. From the resulting model tree the answers to the query can be extracted by constructing an answer tree representing the minimal answers. Formally, a model tree can be defined as follows:

Definition 2.8 *[7] Let $\mathcal{I}$ be a finite set of Herbrand interpretations (models) over a FOL $\mathcal{L}$.* An interpretation (model) tree for *$\mathcal{I}$ is a tree structure where*

- *The root is labeled by the special symbol ε.*

- *Other nodes are labeled with atoms in $\mathcal{I}$ or the special symbol $\not\in$.*
- *A path from the root to a leaf node is called a* branch
- *No atom occurs more than once in a branch.*
- *$I \in \mathcal{I}$ iff $\exists b_I\ I = \{A :\ A\ \in\ b_I\} - \{\not\in\}$ where b_I is a branch in the tree.*

The symbol $\not\in$ is meant to stand for the absence of an atom for a node.

To capture the semantics of a $DDDB$, DB, a model tree must contain the minimal models of DB.

Definition 2.9 *Let DB be a $DDDB$ and let $\mathcal{M}$ be a set of models of DB. Then $\mathcal{T}_{\mathcal{M}}$ is called a* model tree of DB *iff*

$$\forall M' \models DB, \exists M \in \mathcal{M},\ M \subseteq M'$$

We use the notation $\mathcal{T}_{DB}$ to refer to a model tree of DB.

Ordered model trees were introduced in [20] as a normal form to represent a $DDDB$. Next we define this data structure:

Definition 2.10 *Let S be a set of atoms. An order ($>$) on S is an assignment of a unique integer $\mathcal{O}(A)$ to every ground atom A of S.*

- *$A > B$ iff $\mathcal{O}(A) > \mathcal{O}(B)$ for A and B in S.*
- *$B < A$ iff $A > B$.*
- *A sequence of atoms $\langle A_1, \ldots, A_N \rangle$ is ordered iff $\forall_{i<j} A_i > A_j$ for A_i and A_j in S.*
- *R^o denotes the ordered sequence containing the atoms in the set R, where $R \subseteq S$.*

We use the term *ordered set R* to refer to the ordered sequence R^o.

Definition 2.11 *Given an order ($>$) on a set S and ordered sets of atoms $R_1 = \langle A_1, A_2, \ldots, A_N \rangle$ and $R_2 = \langle B_1, B_2, \ldots, B_M \rangle$ with the A_is and B_js in S. Then:*

- *$R_1 > R_2$ iff $\exists k > 0$ such that $\forall_{i<k} A_i = B_i$ and either $A_k > B_k$ or $M < k \leq N$.*
- *$R_2 < R_1$ iff $R_1 > R_2$.*

Definition 2.12 *Given an order ($>$) on $HB_{\mathcal{L}}$, let $\mathcal{I} = \{I_1, \ldots, I_n\}$ be a set of interpretations (models or minimal models) over $\mathcal{L}$. Then, the ordered set of interpretations (models or minimal models) of $\mathcal{I}$ is the sequence $\mathcal{I}^o = \langle I^o_{s_1}, \ldots, I^o_{s_n} \rangle$ such that $\forall_{i<j} I^o_{s_i} > I^o_{s_j}$. I^o_i is referred to as the ordered interpretation (model or minimal model) I_i.*

We use the term *ordered Herbrand base $HB_{\mathcal{L}}$*, to refer to the ordered sequence $HB^o_{\mathcal{L}}$. It completely specifies the total order ($>$).

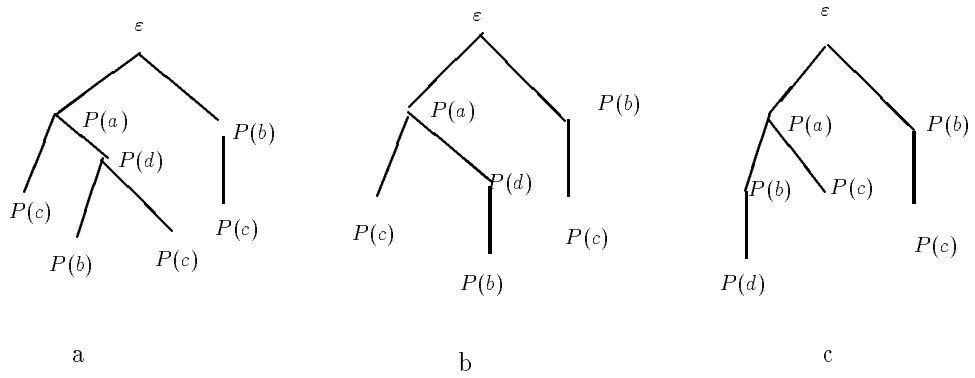

Figure 1: Ordered and Unordered Model Trees for Example 4

Example 3 *Given a set of interpretations* $\mathcal{I} = \{\{C, A, B\}, \{A, D, B\}, \{\}, \{A, B\}\}$ *over the ordered Herbrand base* $\langle D, C, B, A\rangle$*, then the corresponding ordered set of interpretations of* $\mathcal{I}$*,* $\mathcal{I}^o = \langle\langle D, B, A\rangle, \langle C, B, A\rangle, \langle B, A\rangle, \langle\rangle\rangle$*.*

Definition 2.13 *Given an order (>) on* $HB_{\mathcal{L}}$*, let* $\mathcal{I}$ *be a set of interpretations (models, minimal models) over* $\mathcal{I}$*. Let* $\mathcal{T}_{\mathcal{I}}$ *be a model tree for* $\mathcal{I}$*. Let* $\langle b_1, \ldots, b_n\rangle$ *be a right-to-left enumeration of the branches in* $\mathcal{T}_{\mathcal{I}}$ *and let* S_i *be the sequence* $\langle A_{i1}, \ldots, A_{im_i}\rangle$ *corresponding to a root-to-leaf traversal of the branch* b_i*. Then,* $\mathcal{T}_{\mathcal{I}}$ *is ordered iff* $\mathcal{I}^o = \langle S_1, \ldots, S_n\rangle$*.* $\mathcal{T}_{\mathcal{I}}$ *is called the* ordered interpretation (model, minimal model) tree for $\mathcal{I}$*, and is denoted as* $\mathcal{T}^o_{\mathcal{I}}$*.*

Example 4 *Let* $DB = \{P(a) \vee P(b), P(a) \vee P(c), P(c) \vee P(d), P(b) \vee P(c)\}$*. Figure 1 shows: (a) a nonminimal unordered model tree , (b) a minimal unordered model tree and (c) a minimal ordered model tree for DB with the order* $P(a) > P(b) > P(c) > P(d)$*.*

For a given order, the ordered model tree of a $DDDB$ is unique [20]. In [20] algorithms are given for building and updating minimal model trees for positive $DDDBs$ and $DDDBs$ with rules.

Negation in $DDDBs$**:** Usually there is asymmetry in the treatment of positive and negative information in deductive databases. In most cases, negative information is not explicitly stored in the database but is inferred using default rules for negation. One rule for definite (Horn) databases is the *Closed World Assumption* (CWA). Under the CWA an atom, A, is assumed to be *false* if and only if A is not in the *unique* Herbrand model of the database [17]. The CWA is not applicable to $DDDB$ since it may produce inconsistent results. For $DDDBs$, the rule used to define negated atoms is the *Generalized Closed World Assumption (GCWA)* which is an extension of the CWA rule to the disjunctive case [13]. The $GCWA$ is able to consistently define those *atoms* whose negation can be assumed to be *true* in the database. The $GCWA$ is not sufficient for determining the truth or

falsity of a *conjunction of atoms*. In this case the *Extended Generalized Closed World Assumption (EGCWA)* can be used [21]. The default rules for disjunctive deductive databases are formally defined as follows:

Definition 2.14 *[13, 21] Let DB be a DDDB. Then:*

$$GCWA(DB) = \{\neg A : A \in HB_{DB} \text{ and } \nexists M \in \mathcal{MM}(DB) \text{ such that } A \in M\}.$$

$$EGCWA(DB) = \{\neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_n : A_i \in HB_{DB} \text{ and } n > 0 \text{ and } \nexists M \in \mathcal{MM}(DB) \text{ such that } A_i \in M \ \forall i \in \{1, 2, ..., n\}.$$

By $MEGCWA(DB)$ we denote the minimal elements in $EGCWA(DB)$. That is,

$$MEGCWA(DB) = \{C \in EGCWA(DB) \text{ and } \forall C' \subset C, C' \notin EGCWA(DB)\}.$$

Minimal Model Representation of Databases: We discuss the transition from the minimal model representation of a $DDDB$ to its clausal form representation.

Theorem 1 *Let DB_s be a DDDB and $\mathcal{MM}(DB) = \{M_1, M_2, ..., M_n\}$. Let $A_{i,j}$ $\forall j \in \{1, 2, ..., k_i\}$, be the atoms of M_i. Let $DB_t = (A_{1,1} \wedge A_{1,2} \wedge ... \wedge A_{1,k_1}) \vee (A_{2,1} \wedge A_{2,2} \wedge ... \wedge A_{2,k_2}) \vee ... \vee (A_{n,1} \wedge A_{n,2} \wedge ... \wedge A_{n,k_n})$. Then: $DB_s =_{mm} DB_t$.*

Proof: DB_t is in its disjunctive normal form (DNF). A model of DB_t must satisfy at least one of the disjuncts. The minimal models of DB_t are those models that satisfy exactly one of the disjuncts of DB_t for otherwise it will be possible to shrink the model to satisfy only one of the disjuncts contradicting the minimality condition. The minimal models of DB_t satisfying exactly one of the disjuncts of DB_t are the minimal models of DB_s namely the set $\{M_1, M_2, ..., M_n\}$.

Every minimal model, M_i, of DB_s must have all $A_{i,j}$ for $j \in \{1, 2, ..., k_i\}$. It satisfies DB_t by making exactly one of its disjuncts *true*. All minimal models are distinct and no subset of M_t satisfies a disjunct in DB_t by construction. ■

We can transform DB_t from DNF into DB_f in conjunctive normal form (CNF), or clausal form, by forming all the possible clauses that include an element from each disjunct of DB_t. That is, every clause of DB_f must contain at least one element of every minimal model of DB_s. Equivalently, every clause of DB_f contains an atom from every root-to-leaf branch of the minimal model tree of DB_s and every such clause is either in DB_f or subsumed by a clause in DB_f.

DB_f is positive and $DB_f =_{mm} DB_s$. Every clause of DB_f is satisfied by the minimal models of DB_s by construction. For every interpretation that is not a model for DB_s there is a clause of DB_s falsified by that interpretation.

The transformation from a minimal model tree to clausal form can be performed directly by generating all the possible clauses resulting from including an atom from every minimal model (branch) of the tree.

Corollary 2 *Let DB_s be a DDDB. The set $\mathcal{MM}(DB)$ defines a positive DDDB, DB_f, minimal model equivalent to DB_s, ($DB_s =_{mm} DB_f$).*

Proof: The procedure outlined in Theorem 1 describes the construction process of the positive database DB_f. ■

3 Dual Databases

In this section we emphasize the strong connection between the models of a database and the set of clauses derivable from it. We introduce the concept of the *dual* database corresponding to a given disjunctive deductive database and study its properties.

Definition 3.1 *Let DB be a disjunctive deductive database with $\mathcal{MM}(DB) = \{M_1, M_2, ..., M_n\}$. Let $A_{i,j}$ for $j = 1, 2, ..., k_i$, be the atoms in M_i. $DB = (A_{1,1} \wedge A_{1,2} \wedge ... \wedge A_{1,k_1}) \vee (A_{2,1} \wedge A_{2,2} \wedge ... \wedge A_{2,k_2}) \vee ... \vee (A_{n,1} \wedge A_{n,2} \wedge ... \wedge A_{n,k_n})$. Define the dual database as $DB^d = (A_{1,1} \vee A_{1,2} \vee ... \vee A_{1,k_1}) \wedge (A_{2,1} \vee A_{2,2} \vee ... \vee A_{2,k_2}) \wedge ... \wedge (A_{n,1} \vee A_{n,2} \vee ... \vee A_{n,k_n})$. Another way to define DB^d is that it is the database we get by replacing every logical $\vee$ by a logical $\wedge$ and vice versa in the positive clause representation of DB.*

If DB is inconsistent (has no models) then its dual is the empty database. If DB has the empty set as its only minimal model then its dual is the inconsistent database (the empty clause).

We may also elect to perform minimization on DB^d by deleting duplicate atoms in individual clauses and removing subsumed clauses.

3.1 Properties of Dual Databases

Recall that we can refer to a clause either as a disjunction of its literals or as the set of these literals. Therefore, the set of minimal clauses derivable from a disjunctive deductive database, DB

$\mathcal{MS}(DB) = \{C : DB$ minimally derives $C\}$ or

$\mathcal{MS}(DB) = \{\{A_1, A_2, ..., A_m\} : DB$ minimally derives $C = A_1 \vee A_2 \vee ... \vee A_m\}$.

Theorem 2 *Let DB be a DDDB and let DB^d be its dual database. Let $\mathcal{MS}(DB)$ denote the set of minimal positive clauses derivable from DB. Then:*

$$\mathcal{MM}(DB^d) = \mathcal{MS}(DB) \text{ and } \mathcal{MM}(DB) = \mathcal{MS}(DB^d).$$

Or, equivalently $M = \{A_1, A_2, ..., A_m\} \in \mathcal{MM}(DB)$ iff $C = A_1 \vee A_2 \vee ... \vee A_m \in \mathcal{MS}(DB^d)$ and $C = A_1 \vee A_2 \vee ... \vee A_m \in \mathcal{MS}(DB)$ iff $M = \{A_1, A_2, ..., A_m\} \in \mathcal{MM}(DB^d)$.

Proof: Part 1: $\mathcal{MM}(DB^d) = \mathcal{MS}(DB)$.

We first prove that $\mathcal{MM}(DB^d) \subseteq \mathcal{M}(DB)$

Let $M = \{A_1, A_2, ..., A_m\} \in \mathcal{MM}(DB^d)$. M satisfies every clause in DB^d. By definition of DB^d, M must contain an atom from every minimal model of DB. $DB \vdash A_1 \vee A_2 \vee ... \vee A_m$ and $A_1 \vee A_2 \vee ... \vee A_m \in \mathcal{S}(DB)$. Let M be a nonminimal clause of DB. There exists $M' \subset M$ such that $DB \vdash M'$. M' has an atom from every minimal model of DB. M' satisfies every clause in DB^d. M is not a minimal model of DB^d. A contradiction.

Now we show that $\mathcal{MS}(DB) \subseteq \mathcal{MM}(DB^d)$.

Let $C = A_1 \vee A_2 \vee ... \vee A_m \in \mathcal{MS}(DB)$.

We show that $\{A_1, A_2, ..., A_m\} \in \mathcal{MM}(DB^d)$.

Since C is minimally derivable from DB, by Lemma 2 there must exist a set of minimal models of DB, $\{M_{i_k} : A_k \in M_{i_k}$ such that $A_j \notin M_{i_k} \forall j \neq k\}$. That is, for each atom A_i there is a minimal model in which only A_i is *true* and all the other atoms of the clause C are *false*.

Let these minimal models be $M_{i_1}, M_{i_2}, ..., M_{i_m}$. All of them are clauses in DB^d by construction and therefore they must be *true* in every minimal model of DB^d. C satisfies the clauses in DB^d resulting from the models $M_{i_1}, M_{i_2}, ..., M_{i_m}$.

As for the other models of DB each of them contains one or more A_i and are therefore satisfied by C. So C is a model of DB^d. Removing atom A_k from this model will falsify the clause of DB^d corresponding to the model M_{i_k}. Therefore C is a minimal clause for DB^d.

Part 2: We prove that $\mathcal{MM}(DB) = \mathcal{MS}(DB^d)$.

We first show that $\mathcal{MM}(DB) \subseteq \mathcal{MS}(DB^d)$.

Let $M = \{A_1, A_2, ..., A_m\} \in \mathcal{MM}(DB)$. We show that $M \in \mathcal{MS}(DB^d)$.

$M \in \mathcal{S}(DB^d)$ by construction. We need only to show that $M \in \mathcal{MS}(DB^d)$. Assume $M \notin \mathcal{MS}(DB^d)$. There exists $M' \subset M$ such that $M' \in \mathcal{MS}(DB^d)$. Every minimal model of DB^d contains an atom of M'. As a result of the first part of the ongoing proof, every clause in $\mathcal{MS}(DB)$ is a minimal model of DB^d and therefore must contain an atom of M'. M' is a model of DB. A contradiction.

Finally we show that $\mathcal{MS}(DB^d) \subseteq \mathcal{MM}(DB)$.

Let $C = A_1 \vee A_2 \vee ... \vee A_m \in \mathcal{MS}(DB^d)$. We show that $C \in \mathcal{MM}(DB)$. C is *true* in every minimal model of DB^d since it has at least one element from every minimal model of DB^d and consequently C is a model of DB. Let C be a nonminimal model of DB. $\exists\ C' \subset C$ *such that* $C' \in \mathcal{MM}(DB)$. The clause $C' \subset C$ is in $\mathcal{S}(DB^d)$. A contradiction. ■

Example 5 *Let*

$$\begin{aligned} DB = \quad & \{P(a) \vee P(b), \\ & P(a) \vee P(c), \\ & P(c) \vee P(d), \\ & P(b) \vee P(c)\} \end{aligned}$$

$\mathcal{MM}(DB) = \{\{P(a), P(b), P(d)\}, \{P(a), P(c)\}, \{P(b), P(c)\}\}$.
$DB^d = \{P(a) \vee P(b) \vee P(d), P(a) \vee P(c), P(b) \vee P(c)\}$.

Theorem 3 *Let DB be a disjunctive deductive database and let DB^d be its dual database. Then:*

$$\mathcal{M}(DB^d) = \mathcal{S}(DB) \text{ and } \mathcal{M}(DB) = \mathcal{S}(DB^d).$$

Proof: The proof follows directly from Theorem 2 and the observation that every model contains a minimal model and every derivable clause contains a minimally derivable clause. ■

Theorem 4 *Let DB_1 and DB_2 be disjunctive deductive databases. Then:*

$$DB_1 =_{mm} DB_2 \text{ iff } DB_1^d =_{mm} DB_2^d.$$

Proof: $DB_1^d =_{mm} DB_2^d$ iff both have the same set of minimal models $\mathcal{MM}(DB_1^d) = \mathcal{MM}(DB_2^d)$. By Theorem 2 this means that $\mathcal{MS}(DB_1) = \mathcal{MS}(DB_2)$ and consequently, by Lemma 2 $DB_1 =_{mm} DB_2$.

The other side follows directly from the definition of the *dual* database. ■

Theorem 5 *Let DB be a DDDB and let DB^d be its dual database. Then $(DB^d)^d =_{mm} DB$. That is, the double application of the duality rules will always lead to a database equivalent to the original.*

Proof: The dual transformation was defined in terms of minimal models. Double application of the dual transformation will define a database with the same minimal models as DB. ■

Theorem 6 *Let DB be a DDDB. Let DB have the partition $\{DB_1, DB_2, ..., DB_k\}$. That is, $DB = DB_1 \cup DB_2 \cup ... \cup DB_k$ and $DB_i \cap DB_j = \emptyset \ \forall\ i \neq j$ and DB_i is a cluster of $DB\ \forall\ i \in \{1, ..., k\}$. Let DB_i^d be the dual of the partition blocks DB_i. Then: $DB^d = DB_1^d \vee DB_2^d \vee ... \vee DB_k^d$. That is, $DB^d = \{C :\ C = C_1 \vee C_2 \vee ... \vee C_k,\ where\ C_i \in DB_i^d\}$.*

Proof: $\mathcal{MM}(DB) = \{M : M = \cup_{i=1}^{i=k} M_i,\ where M_i \in \mathcal{MM}(DB_i)\ \forall i \in \{1, 2, ..., k\}\}$. That is, the set of minimal models of DB is the *cartesian union* of the minimal models of its partitions [22].

The proof follows immediately. ■

Example 6 *Let $DB = \{P(a) \vee P(b), P(a) \vee P(c), P(d) \vee P(f), P(e) \vee P(f)\}$.*
$DB_1 = \{P(a) \vee P(b), P(a) \vee P(c)\}$ and $DB_1^d = \{P(a), P(b) \vee P(c)\}$.
$DB_2 = \{P(d) \vee P(f), P(e) \vee P(f)\}$ and $DB_2^d = \{P(d) \vee P(e), P(f)\}$.
$DB^d = \{P(a) \vee P(d) \vee P(e), P(a) \vee P(f), P(b) \vee P(c) \vee P(d) \vee P(e), P(b) \vee P(c) \vee P(f)\}$.

Definition 3.2 *A disjunctive deductive database DB is self-dual iff $DB^d =_{mm} DB$.*

For a self-dual database DB we have $\mathcal{MM}(DB) = \mathcal{MS}(DB)$ and $\mathcal{MM}(DB^d) = \mathcal{MS}(DB^d)$

Example 7 *Let $DB_1 = \{P(a)\}$.*
$\mathcal{MM}(DB_1) = \{\{P(a)\}\}$

Let $DB_2 = \{P(a) \vee P(b), P(a) \vee P(c), P(b) \vee P(c)\}$.
$\mathcal{MM}(DB_2) = \{\{P(a), P(b)\}, \{P(a), P(c)\}, \{P(b), P(c)\}\}$.
Let

$$
\begin{aligned}
DB_3 = \ & \{P(a) \vee P(b) \vee P(c),\\
& P(a) \vee P(b) \vee P(d),\\
& P(a) \vee P(b) \vee P(e),\\
& P(a) \vee P(c) \vee P(d),\\
& P(a) \vee P(c) \vee P(e),\\
& P(a) \vee P(d) \vee P(e),\\
& P(b) \vee P(c) \vee P(d),\\
& P(b) \vee P(c) \vee P(e),\\
& P(b) \vee P(d) \vee P(e),\\
& P(c) \vee P(d) \vee P(e)\}
\end{aligned}
$$

$\mathcal{MM}(DB_3) = \{\{P(a), P(b), P(c)\}, \{P(a), P(b), P(d)\}, \{P(a), P(b), P(e)\}, \{P(a), P(c), P(d)\}, \{P(a), P(c), P(e)\}, \{P(a), P(d), P(e)\}, \{P(b), P(c), P(d)\}, \{P(b), P(c), P(e)\}, \{P(b), P(d), P(e)\}, \{P(c), P(d), P(e)\}\}$.

Let $DB_4 = \{P(a) \vee P(b), P(a) \vee P(c), P(a) \vee P(d), P(b) \vee P(c) \vee P(d)\}$.
$\mathcal{MM}(DB_4) = \{\{P(a), P(b)\}, \{P(a), P(c)\}, \{P(a), P(d)\}, \{P(b), P(c), P(d)\}\}$.

Let $DB_5 = \{P(a) \vee P(b)\}$.
$\mathcal{MM}(DB_5) = \{\{P(a)\}, \{P(b)\}\}$

DB_1, DB_2, DB_3 and DB_4 are self-dual. Their minimal clauses and minimal models coincide when treated as sets of atoms. DB_5 is not self-dual.

Given a $DDDB$, DB, and $S \subset HB_{DB}$ such that $\mid S \mid = n$, where n is an odd number, it is possible to construct a self dual database consisting of all possible clauses of length $(n+1)/2$. To see this note that each clause has to contain an atom of every other clause since two sets of $(n+1)/2$ atoms each, have at least one atom in common for otherwise $\mid S \mid > n$. So, every clause is also a model of this database. On the other hand, minimal models of this database are all of cardinality $(n+1)/2$. This is so since $S' \subset S$, such that $\mid S' \mid = (n+1)/2$ will satisfy all database clauses because it intersects with each of them. Any $S'' \subset S'$ (an interpretation of size $m < n+1$) will falsify the clauses consisting entirely of atoms not in S''. Another interesting property of this class of databases is its extremely bad behavior in terms of the number of minimal models generated. However, this is not the only class of self dual databases as demonstrated by DB_4 in Example 7.

The class of self-dual databases is characterized by the interchangeability of their models and clauses. Given any of these representations it is straightforward to determine the other. However, it is not trivial to determine self-duality of a given database.

3.2 Tree Structures for Dual Databases

Given a disjunctive deductive database, DB, the above discussion establishes the strong relationship between its set of minimal models $\mathcal{MM}(DB)$, and its set of minimally derivable positive clauses $\mathcal{MS}(DB)$ through the *dual* database DB^d.

Model trees were introduced as a structure sharing approach to represent information in disjunctive deductive databases in the form of minimal models [7]. The efficiency of answering queries and performing update operations in disjunctive deductive databases depends on the nature of the databases, the type of operations and queries considered and the database representation selected. Queries and updates expressed in terms of models can be handled efficiently using minimal model trees. However, queries and updates expressed in terms of database clauses are inherently more difficult to handle in model trees. They have to be translated into minimal model operations before accessing the tree structure. A way around this problem is to have a minimal clause tree representation for the disjunctive deductive database. This is a structure sharing approach to specifying the set of clauses minimally derivable from a database. Informally, a clause tree for a disjunctive deductive database DB is a tree structure in which nodes are labeled by atoms of the Herbrand base of DB. With each branch of the tree we associate the set of atoms encountered on that branch.

Branches from the root to leaf nodes represent clauses derivable from DB. A minimal clause tree is a clause tree for which there is a one to one correspondence between minimally derivable clauses (elements of $\mathcal{MS}(DB)$) and branches from the root to leaf nodes. The tree is ordered if the root-to-leaf traversal of each branch produces the ordered sequence corresponding to that branch and a left-to-right enumeration of the branches produces the ordered sequence corresponding to the set of ordered branches (see Definition 2.13).

Given a disjunctive deductive database, DB, two approaches to build the ordered minimal clause tree for DB, $\mathcal{T}^o_{\mathcal{MS}(DB)}$, are possible: an indirect, two step solution, by constructing the ordered minimal model tree for the dual database DB^d, using the algorithms developed for constructing minimal model trees in [5, 20] and the properties of dual databases mentioned above. The other approach is to directly construct the minimal clause tree corresponding to DB.

The following two step algorithm can be used to construct the minimal clause tree of a $DDDB$, DB, using the first approach:

- *Step 1*: Build the ordered minimal model tree for DB using any of the algorithms given in [20]. Call the resulting tree $\mathcal{T}^o_{\mathcal{MM}(DB)}$. The branches of $\mathcal{T}^o_{\mathcal{MM}(DB)}$ correspond to the clauses of the positive database DB^d.
- *Step 2*: Use the clauses of DB^d represented by the branches of $\mathcal{T}^o_{\mathcal{MM}(DB)}$ to construct the minimal ordered model tree of DB^d, $\mathcal{T}^o_{\mathcal{MM}(DB^d)}$, using the algorithm given in [20]. From the results on the properties of dual databases it is easy to see that the resulting tree is the ordered minimal clause tree for DB, $\mathcal{T}^o_{\mathcal{MS}(DB)}$.

Note that this approach works for positive and nonpositive $DDDBs$. We do not need to materialize DB^d in order to build its model tree. We can operate directly on the branches of $\mathcal{T}^o_{\mathcal{MM}(DB)}$. Theorem 2, together with the correctness of the minimal model tree building algorithms given in [20], guarantee the correctness of the minimal clause tree construction process.

Note also that in the second step we always deal with the positive database DB^d. The rules are taken care of in step 1. Therefore, the model tree construction process is generally simplified. We can also exploit the order in the tree constructed at the first stage, $\mathcal{T}^o_{\mathcal{MM}(DB)}$, to achieve better performance for Step 2.

Note also that the (minimal) model tree can be viewed as an OR-AND tree of the database DB, where the AND is between the a node and each of its children and the OR is between the siblings of a node. The same structure, when viewed as an AND-OR tree (the roles of the logical operators reversed), represents the clause tree of the dual database DB^d.

The second approach is discussed in the following paragraphs.

3.3 Building Ordered Clause Trees

Here we describe the minimal clause tree construction process directly from the clauses of DB. We consider two classes of disjunctive deductive databases. The first class is positive disjunctive deductive databases where we assume that the database is ground with no negative occurrences of atoms. The second class is disjunctive deductive databases with rules. When translated into clausal form the rules will generate negative literals.

3.3.1 Positive Databases

In the construction of a clause tree for a positive database, we recursively decompose the database by extracting, on each step, the largest atom (in the order $>$) that occurs in the database, and generating two new smaller databases to be processed further.

At each step, the current database, DB, is decomposed to construct subtrees *underneath* a particular node $\mathcal{N}$ of the tree (initially ε, the root). Moreover, no atom in the path from the root to $\mathcal{N}$ occurs in DB.

Let DB be a ground positive $DDDB$ and let $\langle A_1, \ldots, A_N \rangle$ be the ordered Herbrand base underlying DB. Assume that $A \in \{A_1, \ldots, A_N\}$ is the largest atom that occurs positively in DB and $\mathcal{N}$ is the node under expansion ($\mathcal{N} > A$ or $\varepsilon = \mathcal{N}$).

Let $DB = \{F_1, \ldots, F_n, C_1, \ldots, C_m\}$ where the F_i are the clauses that contain A, the C_j are clauses free of A and let $F_i' = F_i - A$ (i.e. the result of removing the literal A from the clause F_i).

We extract the common literal A from the F_i clauses by rewriting DB as $\{(A \vee (F_1' \wedge \cdots \wedge F_n')), C_1, \ldots, C_m\}$ and decompose DB into the two subsets of clauses $DB_{\neg A} = \{C_1, \ldots, C_m\}$ and $DB_A = \{F_1', \ldots, F_n'\}$.

The leftmost child of $\mathcal{N}$ is the node A with the database DB_A to be expanded *underneath* A. The database $DB_{\neg A}$ will be expanded *underneath* $\mathcal{N}$ and to the right of A (it will generate the subtrees of $\mathcal{N}$ to the right of A). The process is recursively applied to these two databases (Figure 2-a).

DB_A and $DB_{\neg A}$ are smaller than DB in the sense that DB_A has less clauses and/or fewer literals in its clauses than DB, and $DB_{\neg A}$ contains fewer clauses than DB. Two terminating conditions are possible:

1. No atom occurs in DB (e.g. DB is empty). The branch represents a clause, not necessarily minimal, of the original database.

2. DB contains the empty clause, $\Box$, (the clause with no literals). In this case, some clauses are nonminimal and no further expansion is needed.

The resulting tree is ordered. This results from expanding the children of a particular node from left to right in an increasing order of the atoms and processing all clauses in a subtree containing that atom simultaneously so that the atom occurs in all clauses in the left subtree and is absent from all clauses in the right subtree. This guarantees that depth first search will always yield the ordered clauses of the database.

The resulting tree is not necessarily minimal. A minimization step is needed. The tree construction process guarantees that a clause is nonminimal if and only if it is a superset of a clause appearing to its right in the tree. This is so since the tested clause contains at least one atom (the top node) not present in clauses to its right. A clause in the tree may subsume other clauses to its left only. The rightmost clause generated is minimal since it can have no clauses to subsume it. If we elect to generate right clauses first then we can limit our checking to already found minimal clauses.

Example 8 *Let* $DB = \{P(a) \vee P(c) \vee P(d), P(a) \vee P(b) \vee P(c), P(c) \vee P(d), P(b) \vee P(d)\}$.

We build an ordered model tree for DB *assuming the following order of atoms* $P(a) > P(b) > P(c) > P(d)$.

We start with $P(a)$.

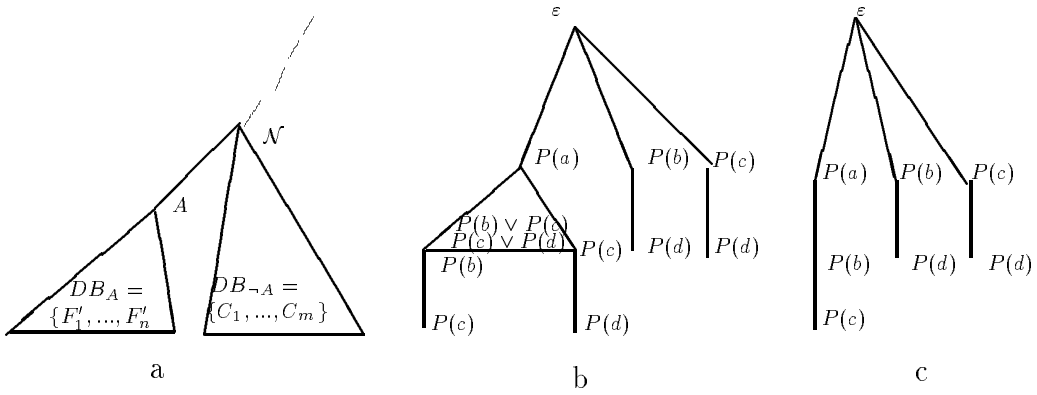

Figure 2: Building the Clause Tree for Positive Databases (a-General; b,c-for Example 8)

$DB1 = DB_{P(a)} = \{P(c) \vee P(d), P(b) \vee P(c)\}$ *and*
$DB2 = DB_{\neg P(a)} = \{P(c) \vee P(d), P(b) \vee P(d)\}$.

Next we expand DB2 on $P(b)$ *to get* $DB2_{P(b)} = \{P(d)\}$ *and* $DB3 = DB2_{\neg P(b)} = \{P(c) \vee P(d)\}$. $DB3_{P(c)} = \{P(d)\}$ *and* $DB3_{\neg P(c)} = \{\}$. *So the children of the root are P(a), P(b) and P(c) and the associated subtrees are* $\{P(c) \vee P(d), P(b) \vee P(c)\}$, $\{P(d)\}$ *and* $\{P(d)\}$, *respectively. The tree that needs further expansion is* $DB1$.

$DB1_{P(b)} = \{P(c)\}$ *and*
$DB4 = DB1_{\neg P(b)} = \{P(c) \vee P(d)\}$.
$DB4_{P(c)} = \{P(d)\}$, *and* $DB4_{\neg P(c)} = \{\}$

In the resulting tree we have one nonminimal clause $P(a) \vee P(c) \vee P(d)$ *which is to be deleted as shown in Figure 2-b,c.*

3.3.2 Clause Tree Construction for Databases with Rules

To construct the minimal clause tree for a database with rules, DB, we first build the minimal clause tree for the positive part of DB (DB without the rules, E_{DB}). Next, for every rule C in DB we try resolving C against each of the branches of the current clause tree. The resolvent clauses will have fewer negative literal. If a positive clause is generated as a result of these resolutions then it is added to the minimal ordered clause tree using the clause addition algorithm to be described later. If no resolutions are possible or if the resolutions result in no positive clauses then no update is performed on the minimal clause tree. After each positive clause addition the rule application process is repeated until no new positive clauses are possible and therefore no new updates to the tree are generated. The order in the tree can be exploited to facilitate the resolution process.

Example 9 *Let*

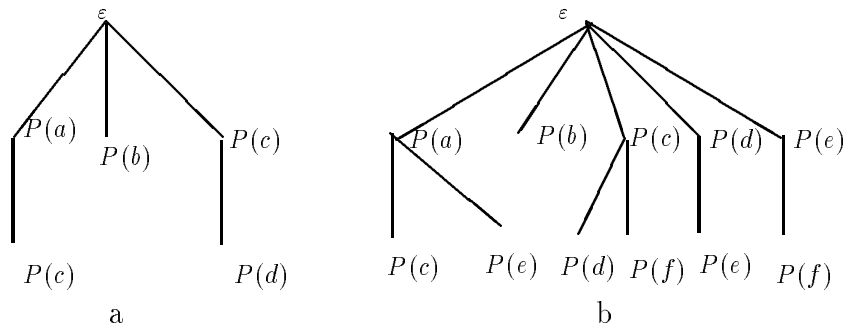

Figure 3: Building the Clause Tree for Databases with Rules for Example 9

$$
\begin{aligned}
DB = \quad & \{P(e) \leftarrow P(b) \wedge P(c), \\
& P(a) \vee P(b), \\
& P(a) \vee P(c), \\
& P(c) \vee P(d), \\
& P(b), \\
& P(f) \leftarrow P(a) \wedge P(d)\}
\end{aligned}
$$

We build an ordered model tree for DB assuming the alphabetical order of atoms which consists of the original clauses minus the clause $P(a) \vee P(b)$ which is subsumed by $P(b)$.

Employing the rules we generate the additional clauses
$P(a) \vee P(e), P(d) \vee P(e), P(f) \vee P(c), P(f) \vee P(e)$
which are added to the original tree to get the tree shown Figure 3.

It is worthwhile to note that in constructing DB^d the only relevant atoms of the HB_{DB} are those occurring in DB. Other atoms of HB_{DB} do not contribute to the minimal model structure or to the minimal clause structure of DB or DB^d. In addition, *definite* atoms of HB_{DB} (atoms occurring in every minimal model) will occur in every clause of DB^d and vice versa. Therefore, we need to pay attention to the set of *indefinite* atoms of HB_{DB} (atoms present in some, but not all, minimal models; atoms occurring in minimal indefinite clauses). The *indefinite* atoms are the determining factor in the size of the resulting minimal model trees and the minimal clause trees for DB and DB^d. They define the branching of the tree and therefore its complexity. The *definite* atoms on the other hand constitute the common part of all branches. Under the proper order selection each of them can be made to appear only once in the tree. Note also that the set of *definite* atoms and the set of *indefinite* atoms of a $DDDB$ are disjoint (their intersection is empty).

3.4 Operations on Ordered Clause Trees

In this section we discuss the problem of performing update operations on ordered minimal clause trees. We assume that the tree is minimal before the start of the update. Each branch represents a minimally derivable clause in DB and every minimally derivable clause in DB has a root to leaf branch in the minimal clause tree.

The operations we discuss are adding a clause to DB and removing a clause from DB. The decision to add or remove a clause is external to our update algorithms. It can result, among other factors, from applying the rules to the extensional database or update operations to the database.

We may elect to use the standard algorithms reported in [7, 20] to update the minimal *model* tree of the database and then reconstruct the minimal *clause* tree to reflect the introduced updates. However, this approach may be costly and here we describe how to perform updates directly on the minimal clause tree.

3.4.1 Clause Addition

Let C be the clause to be added to the $DDDB$, DB. Order the atoms in C and search for an exact match in the branches of the minimal clause tree for DB.

If C is a branch in the tree then do nothing. The clause is already in the tree.

If C is not a clause in the tree then add C at the proper location in the tree to preserve the order and perform clause minimization to remove any nonminimal clauses.

If C is added to the tree then check for compliance with the rules, if any. Add the appropriate clauses until no further additions are needed.

Example 10 *Let* $DB = \{P(a) \vee P(b), P(a) \vee P(c), P(f) \leftarrow P(d)\}$.

First add the clause $P(e) \vee P(d)$.

$DB^1 = \{P(a) \vee P(b), P(a) \vee P(c), P(e) \vee P(d), P(f) \leftarrow P(d)\}$.

The application of the rule to DB^1 *will result in the addition of the clause* $P(e) \vee P(f)$.

$DB^2 = \{P(a) \vee P(b), P(a) \vee P(c), P(f) \leftarrow P(d), P(e) \vee P(d), P(e) \vee P(f)\}$

which complies with the database rules.

Next add the unit clause $P(d)$ *to* DB^2. *The clause* $P(e) \vee P(d)$ *becomes nonminimal and is therefore deleted to give*

$DB^3 = \{P(a) \vee P(b), P(a) \vee P(c), P(f) \leftarrow P(d), P(d), P(e) \vee P(f)\}$.

The application of the rule to DB^3 *results in the additional clause* $P(f)$.

$DB^4 = \{P(a) \vee P(b), P(a) \vee P(c), P(f) \leftarrow P(d), P(d), P(e) \vee P(f), P(f)\}$.

The clause $P(e) \vee P(f)$ *is not minimal and is therefore deleted to obtain the final* $DB^5 = \{P(a) \vee P(b), P(a) \vee P(c), P(f) \leftarrow P(d), P(d), P(f)\}$. *The steps are shown in Figure 4.*

3.4.2 Clause Deletion

Let C be the clause to be deleted from the database. Order the atoms in C and search for an exact match in the tree branches. If C is not in the tree then do nothing.

If there is a branch in the tree corresponding to C then delete that branch.

Check for database compliance with the rules, if any. Modify the database accordingly. We may need to remove additional clauses to accomplish the update. This process may be nondeterministic:

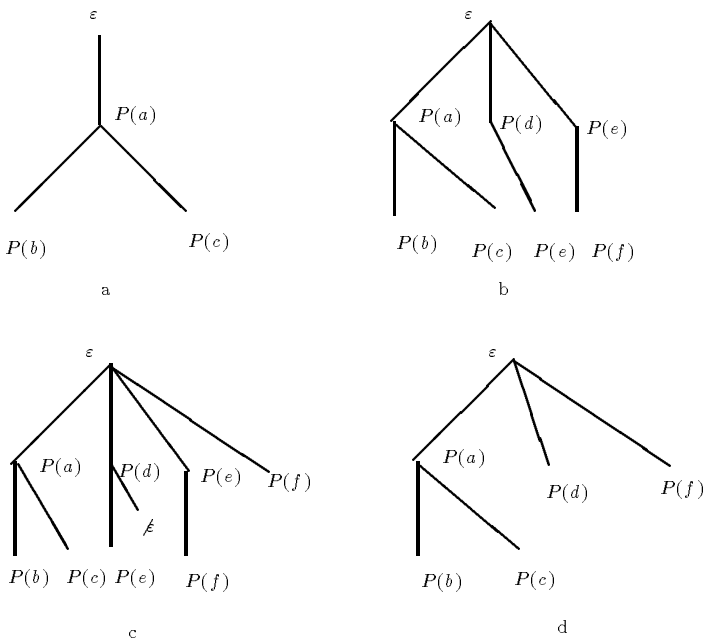

Figure 4: Clause Addition to a Minimal Clause Tree for Example 10

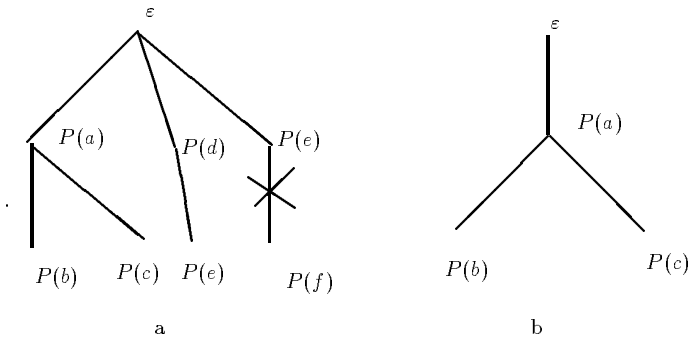

Figure 5: Clause Deletion from a Minimal Clause Tree for Example 11

more than one choice can accomplish the deletion. However, we may elect to reject the deletion if the resulting database does not obey the rules. The choice depends on the application under consideration.

If as a result, additional clauses were added to the tree then check for clause minimality in the tree and remove any nonminimal clauses. No minimality check is needed if deletion was the only operation performed. We may elect to reject the deletion of a clause that necessitates further updates.

Example 11 *Let* $DB = \{P(a) \vee P(b), P(a) \vee P(c), P(f) \leftarrow P(d), P(e) \vee P(d), P(e) \vee P(f)\}$.

First delete the clause $P(e) \vee P(f)$. *The resulting database is*

$DB^1 = \{P(a) \vee P(b), P(a) \vee P(c), P(f) \leftarrow P(d), P(e) \vee P(d)\}$.

DB^1 *does not comply with the rule* $P(f) \leftarrow P(d)$.

To make it comply we need either to add the clause $P(e) \vee P(f)$ *and thus undo the update or to delete the additional clause* $P(e) \vee P(d)$ *and preserve the update and get the final database*

$DB^2 = \{P(a) \vee P(b), P(a) \vee P(c), P(f) \leftarrow P(d)\}$ *which accomplishes the required deletion and complies with the rules. If we are restricted to the deletion of the single clause* $P(e) \vee P(f)$ *then the update cannot be accomplished. The original and resulting trees when the update succeeds are shown in Figure 5-a and Figure 5-b respectively. We could also delete the rule itself. However, since the rules generally serve as view definitions we do not pursue this option.*

The following example demonstrates that clause deletion can be performed in more than one way.

Example 12 *Let* $DB = \{P(a), P(b), P(c) \leftarrow P(a) \wedge P(b)\}$. *The original tree contains the definite clauses* $P(a), P(b)$, *and* $P(c)$. *Deleting* $P(c)$ *can be accomplished by removing the* $P(c)$ *branch and one (or both) of the branches corresponding to* $P(a)$ *and* $P(b)$.

Note also that in general the clause tree update operations need not be reversible. Adding a clause and deleting it may not take us back to the original database as a result of update propagation.

For example if $DB = \{P(a) \vee P(b)\}$ and the clause $P(b)$ is added then the resulting database will be $DB^1 = \{P(b)\}$. Clearly deleting $P(b)$ will result in the empty database. Our interpretation of updates is not the only one possible. The detailed discussion of the topic of clause addition and deletion is beyond the scope of this paper. Several papers on database updates address this issue [2, 8, 18].

3.5 Query Answering in Minimal Clause Trees

In [5] algorithms were described to extract answers to queries from minimal model trees. While the same algorithms are applicable to minimal ordered model trees [20] performance improvements can be achieved by modifying the algorithms to account for the order in the tree.

We would like to develop answer extraction mechanisms for minimal clause trees. We consider two cases: disjunctive queries and conjunctive queries.

3.5.1 Disjunctive Queries

A disjunctive query $Q(x)$ is a query of the form $Q(x) = P_1(x) \vee P_2(x) \vee ... \vee P_n(x)$, where P_i is a predicate in DB for all $i \in \{1, 2, ..., n\}$. To answer such a query we need to find a clause (branch) in the minimal clause tree that is an instance of this query or that subsumes an instance of the query. The substitution set resulting from all such tree branches is the answer set to the query $Q(x)$. Alternatively, we can negate the query $Q(x)$ to get $\neg P_1(x) \wedge \neg P_2(x) \wedge ... \wedge \neg P_n(x)$. Every substitution that generates the empty clause from any one of the tree branches is an answer to $Q(x)$. If no derivation of the empty clause is possible then the query has no answers.

Basically the disjunctive query answering process in minimal clause trees is a tree search for the answers in the branches of the tree.

Example 13 *Let* $DB = \{P(e), P(a) \vee R(d), P(a) \vee P(c), R(a) \vee R(b)\}$ *and let* $Q(x) = P(x) \vee R(x)$. *Then,* $e, a + d, a + c, a + b \in ANSWER(Q(x))$.

Note that the empty clause $\square$ *is generated from the negation of the query* $\neg P(x) \wedge \neg R(x)$ *and the clauses* $P(e), P(a) \vee R(d), P(a) \vee P(c)$, *and* $R(a) \vee R(b)$, *respectively, to generate the above answers. The indefinite answer,* $a + d$, *to the query* $Q(x)$ *denotes that* $P(a) \vee R(a) \vee P(d) \vee R(d)$ *is an answer. That is, it is entailed by the database since it is subsumed by* $P(a) \vee R(d)$, *which is a fact in the database. If the user desires to know the smallest answer that satisfies the query, it could be determined by asking if subqueries of the answer were satisfied.*

3.5.2 Conjunctive Queries

A conjunctive query $Q(x)$ is a query of the form $Q(x) = P_1(x) \wedge P_2(x) \wedge ... \wedge P_n(x)$. The process of conjunctive query evaluation is more complex. A simple search is not sufficient.

To answer $Q(x)$ we add a new rule to the database with the head corresponding to the query $Q(x) \leftarrow P_1(x) \wedge P_2(x) \wedge ... \wedge P_n(x)$. We update the minimal clause tree to add all the positive clauses containing the new predicate Q. Clauses pure in predicate Q are all the possible answers to the query $Q(x)$. Mixed clauses generate no answers to $Q(x)$.

Example 14 *Let* $DB = \{P(a) \vee P(b), P(a) \vee R(b), R(a) \vee R(b), P(b) \vee R(a), P(a) \vee P(c), R(a) \vee R(c)\}$.
The new rule added is $Q(x) \leftarrow P(x) \wedge R(x)$.
The new positive clauses generated are $\{P(b) \vee Q(a), P(a) \vee Q(b), R(b) \vee Q(a), R(a) \vee Q(b), P(b) \vee Q(a) \vee P(c), R(b) \vee Q(a) \vee P(c), P(b) \vee Q(a) \vee R(c), R(b) \vee Q(a) \vee R(c)\}$.
Further application of the rule generates the additional new clauses $\{Q(a) \vee Q(b)\}$. *Clauses pure in* Q *generate the answer* $a + b$. *No other answers are generated.*

4 Complementary Databases

Given a $DDDB$, DB, we define another derivative database corresponding to it, the *complementary database*, DB^c. The definition of DB^c is based on the minimal model representation of DB. We study the properties of the *complementary database* and its possible utility for computing database *completions*.

Definition 4.1 *Let* DB *be a* $DDDB$ *with the set of minimal models* $\{M_1, M_2, ..., M_n\}$. *The complementary database,* DB^c, *is the database with the set of minimal models:* $\{M_1', M_2', ..., M_n'\}$, *where* $M_i' = HB_{DB} \setminus M_i$ *for all* $i \in \{1, 2, ..., n\}$.

Note that all elements in $\mathcal{MM}(DB^c) = \{M_1', M_2', ..., M_n'\}$, are distinct and minimal for otherwise there must exist two models M_i' and M_j' such that $i \neq j$ and $M_i' \subseteq M_j'$. In this case $(HB_{DB} \setminus M_j') \subseteq (HB_{DB} \setminus M_i')$. $M_j' \subseteq M_i'$ contradicting the minimality of the elements of $\mathcal{MM}(DB)$. Therefore, DB^c is well defined. We may also elect to perform minimization on DB^c by deleting duplicate atoms in individual clauses and removing subsumed clauses.

Example 15 *Let* $DB = \{P(a) \vee P(b), P(a) \vee P(c), P(c) \vee P(d), P(b) \vee P(c)\}$.
$\mathcal{MM}(DB) = \{\{P(a), P(b), P(d)\}, \{P(a), P(c)\}, \{P(b), P(c)\}\}$.
$HB_{DB} = \{P(a), P(b), P(c), P(d)\}$.
$\mathcal{MM}(DB^c) = \{\{P(c)\}, \{P(b), P(d)\}, \{P(a), P(d)\}\}$.
$DB^c = P(c) \vee (P(b) \wedge P(d)) \vee (P(a) \wedge P(d))$.
Expanding we get $DB^c = \{P(a) \vee P(b) \vee P(c), P(c) \vee P(d)\}$.

Definition 4.2 *Let* DB *be a* $DDDB$. *By* $DB^\neg$ *we denote the set of clauses resulting from replacing all atom occurrences in* DB *by their negations.*

If DB is positive then the negative database $DB^\neg$ has the empty set, $\emptyset$, as its only minimal model. In this case, it may be more reasonable to talk about maximal models of $DB^\neg$. That is, the set $\mathcal{XM}(DB^\neg) = \{M : M \models DB^\neg \text{ and } \forall M' \supset M, M' \not\models DB^\neg\}$.

For a positive database, DB, it is easy to show that $\mathcal{XM}(DB^\neg) = \mathcal{MM}(DB^c)$. To see that note that the maximal models of $DB^\neg$ are exactly the complements of the minimal models of DB relative to HB_{DB}.

Example 16 *For the database in Example 15*
$DB^\neg = \{\neg P(a) \vee \neg P(b), \neg P(a) \vee \neg P(c), \neg P(c) \vee \neg P(d), \neg P(b) \vee \neg P(c)\}$.
$\mathcal{XM}(DB^\neg) = \{\{P(c)\}, \{P(b), P(d)\}, \{P(a), P(d)\}\} = \mathcal{MM}(DB^c)$.

4.1 Properties of Complementary Databases

Theorem 7 *Let DB be a disjunctive deductive database and let DB^c be its complementary database. Then*

$$(DB^c)^c = DB.$$

Proof: The proof follows from observing that $HB_{DB} \setminus M_i' = M_i$. ∎

Theorem 8 *Let DB_1 and DB_2 be disjunctive deductive databases with the same Herbrand base, HB_{DB}. Then:*

$$DB_1 =_{mm} DB_2 \; iff \; DB_1^c =_{mm} DB_2^c.$$

Proof: $DB_1 =_{mm} DB_2$ *iff* $\mathcal{MM}(DB_1) = \mathcal{MM}(DB_2)$. $M \in \mathcal{MM}(DB_1)$ *iff* $M \in \mathcal{MM}(DB_2)$. $\{M' : M' = HB_{DB} \setminus M, \forall M \in \mathcal{MM}(DB_1)\} = \{M' : M' = HB_{DB} \setminus M, \forall M \in \mathcal{MM}(DB_2)\}$. $\mathcal{MM}(DB_1^c) = \mathcal{MM}(DB_2^c)$. $DB_1^c =_{mm} DB_2^c$. ∎

Theorem 9 *Let DB be a DDDB and let DB^c be its complementary database. Then:*
$C = A_1 \vee A_2 \vee ... \vee A_m \in \mathcal{MS}(DB^c)$ *iff* $C^\neg = \neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m \in MEGCWA(DB)$.

Proof: Let $C = A_1 \vee A_2 \vee ... \vee A_m$ be in $\mathcal{MS}(DB^c)$. We show that $C^\neg = \neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m \in MEGCWA(DB)$.

Let $M' \in \mathcal{MM}(DB^c)$. $\exists \; A \in (M \cap C)$. The corresponding minimal model of DB is $M = HB_{DB} \setminus M'$ and $A \notin M$. Therefore, $\neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m$ is *true* in M and consequently, in every minimal model of DB. By Lemma 2, for every atom A_i of C there is a minimal model of DB^c, M_i', containing A_i but none of the other atoms of C. All other minimal models of DB^c have one or more atoms of C. Assume that a subclause of $C^\neg$, say $C'^\neg$, is in $EGCWA(DB)$. Let $A_i \in (C \setminus C')$ and let M_i' be the minimal model of DB^c such that $M_i' \cap C = \{A_i\}$. The subclause $C'^\neg$ is *false* in M_i. A contradiction.

Let $C^\neg = \neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m \in EGCWA(DB)$. We show that $C = \{A_1 \vee A_2 \vee ... \vee A_m\} \in \mathcal{MS}(DB^c)$.

Let $M \in \mathcal{MM}(DB)$. $\exists \; A_i \in C$ *and* $A_i \notin M$. By definition, $A_i \in (HB_{DB} \setminus M)$. That is, every minimal model of DB^c has an atom of C in it. C is *true* in every minimal model of DB^c. We need only show that it is minimal. Assume it is not. There exists a subclause of C, say $C' \in \mathcal{MS}(DB^c)$. By the first part of the ongoing proof $C'^\neg$ is in $EGCWA(DB)$. A contradiction. ∎

Example 17 *Let* $DB = \{P(a) \vee P(b), P(a) \vee P(c), P(c) \vee P(d), P(b) \vee P(c)\}$.

$\mathcal{MM}(DB) = \{\{P(a), P(b), P(d)\}, \{P(a), P(c)\}, \{P(b), P(c)\}\}$.

$\mathcal{MM}(DB^c) = \{\{P(c)\}, \{P(b), P(d)\}, \{P(a), P(d)\}\}$.
$EGCWA(DB) = \{\neg P(a) \vee \neg P(b) \vee \neg P(c), \neg P(c) \vee \neg P(d)\}$.
$\mathcal{MS}(DB^c) = \{P(a) \vee P(b) \vee P(c), P(c) \vee P(d)\}$.
$\neg EGCWA(DB) = \mathcal{MS}(DB^c)$

Theorem 10 *Let DB be a disjunctive deductive database. Given $EGCWA(DB)$ it is possible to construct DB' such that $DB' =_{mm} DB$.*

Proof: Construct DB' such that $DB' = (MEGCWA(DB)^\neg)^c$. We show that $DB' =_{mm} DB$.

By Theorem 9 $(MEGCWA(DB)^\neg)^c = (\mathcal{MS}(DB^c))^c$.

By Corollary 1 $(\mathcal{MS}(DB^c))^c =_{mm} (DB^c)^c$.

By Theorem 7 $(DB^c)^c =_{mm} DB$.

An therefore, $DB =_{mm} (MEGCWA(DB)^\neg)^c$. ■

Example 18 *Given $EGCWA(DB) = \{\neg P(a) \vee \neg P(b) \vee \neg P(c), \neg P(c) \vee \neg P(d)\}$.*
$EGCWA(DB)^\neg = \{P(a) \vee P(b) \vee P(c), P(c) \vee P(d)\} = \mathcal{MS}(DB^c)$.
$\mathcal{MM}(DB^c) = \{\{P(c)\}, \{P(b), P(d)\}, \{P(a), P(d)\}\}$.
$\mathcal{MM}(DB) = \{\{P(a), P(b), P(d)\}, \{P(a), P(c)\}, \{P(b), P(c)\}\}$.
$DB' = \{P(a) \vee P(b), P(a) \vee P(c), P(c) \vee P(d), P(b) \vee P(c)\}$.
DB' is the same as DB in Example 17.

Theorem 11 *Let DB_1 and DB_2 be disjunctive deductive databases. Then:*

$$DB_1 =_{mm} DB_2 \text{ iff } EGCWA(DB_1) = EGCWA(DB_2).$$

Proof: $DB_1 =_{mm} DB_2$ iff $\mathcal{MM}(DB_1) = \mathcal{MM}(DB_2)$, iff $DB_1^c =_{mm} DB_2^c$ (Theorem 8), iff $\mathcal{MS}(DB_1^c) = \mathcal{MS}(DB_2^c)$ (Corollary 1), iff $EGCWA(DB_1) = EGCWA(DB_2)$ (Theorem 9). ■

Corollary 3 *Given a DDDB, DB, then DB^d, DB^c, $\mathcal{MM}(DB)$, $\mathcal{MS}(DB)$, $EGCWA(DB)$ are alternative representations of DB. Given any of these representations it is possible to reconstruct DB' such that $DB' =_{mm} DB$ and it is possible to construct any of the other representations.*

Definition 4.3 *A disjunctive deductive database DB is self-complementary iff $DB^c = DB$.*

Example 19 *Let $DB_1 = \{P(a) \vee P(b), P(a) \vee P(c)\}$. $\mathcal{MM}(DB_2) = \{\{P(a)\}, \{P(b), P(c)\}\}$*
Let $DB_2 = \{P(a) \vee P(b), P(c) \vee P(d)\}$. $\mathcal{MM}(DB_2) = \{\{P(a), P(c)\}, \{P(a), P(d)\}, \{P(b), P(c)\}, \{P(b), P(d)\}\}$.
Let $DB_3 = \{P(a) \vee P(b) \vee P(c)\}$. $\mathcal{MM}(DB_3) = \{\{P(a)\}, \{P(b)\}, \{P(c)\}\}$.
DB_1 and DB_2 are self-complementary while DB_3 is not.

It is not hard to show that given an even number of atoms n it is possible to construct a self complementary database having as its set of minimal models all the possible interpretations of length $n/2$. To see this note that the complement of each minimal model is also a minimal model since it is an interpretation of length $n/2$. However this is not the only class of self dual databases as demonstrated by DB_1 in Example 19.

Theorem 12 *Let DB be a disjunctive deductive database with the partition $\{DB_1, DB_2, ..., DB_k\}$; That is, $DB = DB_1 \cup DB_2 \cup ... \cup DB_k$, and $DB_i \cap DB_j = \emptyset \ \forall \ i \neq j$ and DB_i is a cluster of DB $\forall i \in \{1, ..., k\}$. Let $ATM(DB_i)$ denote the set of atoms that occur in DB_i (in a sense, a*

localized *Herbrand base of DB_i). Let DB_i^c, be the database with the minimal models* $\mathcal{MM}(DB_i^c) = \{M' : M \in \mathcal{MM}(DB_i)$ *and* $M' = ATM(DB_i) \setminus M\}$.

Then $DB_1^c, DB_2^c, ..., DB_k^c$ are partition blocks of DB^c. In addition DB^c has the block containing all the atoms of the set $HB_{DB} \setminus (\cup_{i=1}^{i=k} AT(DB_i))$.

Proof: The proof is along the same lines of the proof of Theorem 6. It follows from the disjointness of the local Herbrand bases of the individual blocks and the fact that each minimal model of DB contains exactly one minimal model of each block [20]. ■

Example 20 *Let* $DB = \{P(a) \vee P(b), P(a) \vee P(c), P(d) \vee P(f), P(e) \vee P(f), P(e) \vee \neg P(g)\}$.
Let $HB_{DB} = \{P(a), P(b), P(c), P(d), P(e), P(f), P(g), P(h)\}$
$DB_1 = \{P(a) \vee P(b), P(a) \vee P(c)\}$.
$\mathcal{MM}(DB_1) = \{\{P(a)\}, \{P(b), P(c)\}\}$.
$\mathcal{MM}(DB_1^c) = \{\{P(b), P(c)\}, \{P(a)\}\}$.
$DB_2 = \{P(d) \vee P(f), P(e) \vee P(f), P(e) \vee \neg P(g)\}$.
$\mathcal{MM}(DB_2) = \{\{P(d), P(e)\}, \{P(f)\}\}$.
$\mathcal{MM}(DB_2^c) = \{\{P(g), P(f)\}, \{P(g), P(d), P(e)\}$
$\mathcal{MM}(DB^c) = \{\{P(b), P(c), P(g), P(f), P(h)\}, \{P(b), P(c), P(d), P(e), P(g), P(h)\},$
$\{P(a), P(g), P(f), P(h)\}, \{P(a), P(d), P(e), P(g), P(h)\}\}$.

The size of the Herbrand base of the database DB may be much larger than the number of atoms occurring in the database. In constructing complementary databases the atoms of interest for us are those contributing to the minimal model structure of DB. Other atoms of the Herbrand base can be ignored during the processing. For example, atoms not occurring in the database or those occurring only in clauses with pure negative literals cannot contribute to the model structure of DB and therefore such atoms can be ignored [1, 20]. In Example 20 the atoms $P(g)$ and $P(h)$ appeared in every minimal model of DB^c since none of them contributed to the model structure of DB. $P(g)$ appeared only in clauses with pure negative literals and $P(h)$ did not occur in DB.

4.2 Tree Structures for Complementary Databases

Given a database DB with a corresponding minimal model tree it is possible to construct the model tree for the *complementary* database, DB^c, simply by replacing every model by its complement. Direct application of the definition of the *complementary* database shows that if the original tree is minimal then the resulting *complementary* tree is also minimal. However, even if the original tree is ordered the complementary model tree need not be ordered.

In constructing the minimal complementary tree we must start from a minimal tree for the original database. If the original model tree is not minimal then the resulting complementary model tree will not be minimal either. Direct minimization on the complementary tree cannot be performed. It will result in removing the needed models rather than the unnecessary ones.

Example 21 *Let* $DB = \{P(a) \vee P(b), P(a) \vee P(c), P(c) \vee P(d), P(b) \vee P(c)\}$. *The minimal model tree of DB and the minimal model tree of the complementary database DB^c are given in Figure 6.*

Consider the set of models $\{\{P(a), P(b), P(d)\}, \{P(a), P(c)\}, \{P(b), P(c)\}, \{P(a), P(b), P(c)\}\}$. *It contains the nonminimal model* $\{P(a), P(b), P(c)\}$.

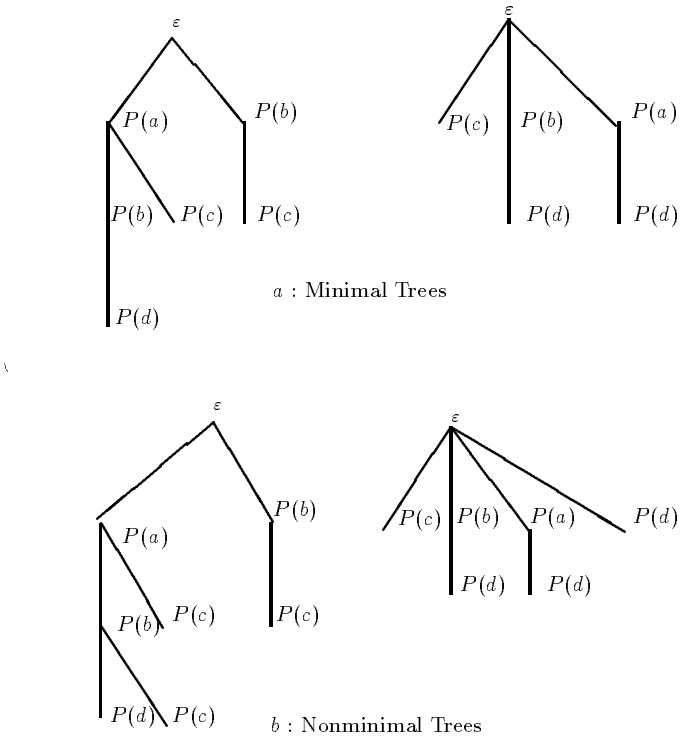

Figure 6: Model Trees for Complementary Databases for Example 21

$HB_{DB} = \{P(a), P(b), P(c), P(d)\}$. $\{M' : HB_{DB} \setminus M\} = \{\{P(c)\}, \{P(b), P(d)\}, \{P(a), P(d)\}, \{P(d)\}\}$. This set contains the nonminimal models $\{P(b), P(d)\}$ and $\{P(a), P(d)\}$. The nonminimal trees for DB and DB^c are also given in Figure 6. Minimization results in the following set of models $\{\{P(c)\}, \{P(d)\}\}$ rather than the set in Example 17.

However, if we start from a nonminimal model tree to obtain a nonminimal complementary tree we can still achieve the minimal complementary tree by removing models that are subsets of other models (in a sense *reverse* minimization). This procedure is correct since $M_1 \subset M_2$ iff $(HB_{DB} \setminus M_2) \subset (HB_{DB} \setminus M_1)$. Therefore *reverse* minimization in the complementary tree is equivalent to conventional minimization in the original tree.

Given a database DB and its minimal model tree it is possible to construct its complementary model tree as explained earlier and then construct the dual of DB^c, $(DB^c)^d$. The resulting tree has as branches the minimal elements of $EGCWA(DB)$. Clauses of length 1 are the elements of $GCWA(DB)$. The maximal depth of the tree is also the limit of the length of possible minimal negative clauses in $EGCWA(DB)$

The construction of the *complementary* tree may be complicated by the large size of the Herbrand base as compared with the number of ground atoms occurring in the model tree. We can limit our attention to the elements of the Herbrand base occurring in the model tree since all other atoms will appear as definite clauses in the complementary tree and therefore have no effect on the disjunctive components of the tree. We can even go a step further. Since definite atoms (those occurring in every minimal model) are always absent from the minimal models of DB^c and atoms not occurring at all in DB are always in every minimal model of DB^c then we can restrict our attention to atoms of the Herbrand base occurring in minimal indefinite clauses (in some, but not all, minimal models of DB). These are the source of the branching in the resulting trees.

5 Conclusion

In this paper we introduced the concepts of *dual* and *complementary* databases for a given disjunctive deductive databases, DB. We investigated the properties of these databases and their relationship to the original database and its completions. We showed that several representations can be used to specify a database DB including the dual database (DB^d), the complementary database (DB^c), the set of minimal models $\mathcal{MM}(DB)$, the set of minimally derivable clauses $\mathcal{MS}(DB)$ and the Extended Generalized Closed World Assumption $EGCWA(DB)$ and showed how to transform one representation into the other. We also described algorithms to construct minimal clause trees for a given database and to perform addition and deletion operations on these trees. The minimal clause tree and the minimal model tree for the *complementary* database can be used for efficient query evaluation and for evaluating of the *completion* of the database. Additional properties of the database such as the maximal length of minimal disjunctive answers and the maximal length of elements of $EGCWA(DB)$ can also be determined from the tree structures for dual and complementary databases.

Each of the representations discussed in this paper is sufficient to characterize the semantics of the disjunctive deductive database and each can be converted into the other. However, the transformation can be computationally expensive - exponential in the size of the number of clauses in the database.

The selection of a particular representation for a $DDDB$ depends upon the type of query most frequently requested in a particular application. For example, queries based on clause searches (disjunctive queries) can be handled easily by using the Dual (minimal ordered clause tree) representation for the database. In this case a search for an instance of the clause (query) reduces to a string matching operation in the tree branches which can be accomplished in linear time in the size of the Herbrand base (assuming an order on the underlying Herbrand base). Similar reasoning can be applied to the case of minimal model-based operations and ordered minimal model trees.

While the paper is concerned with the study of the theoretical aspects of $DDDB$ representations, the underlying motivation for the study is to enable the selection of the appropriate representation for an application under consideration.

In general, dealing with disjunctive databases is computationally expensive. At the worst, if the entire database consists exclusively of disjuncts, and there are N disjuncts of length greater than one, it may take time exponential in N to answer a query. However, we believe that most realistic databases are primarily definite, with a small fraction of the database being disjunctive. Hence, the exponential size of the database refers to the size of the set of disjuncts. Even here, the size may be reduced by considering clusters of disjuncts. The amount of time to answer a query can be the size of an individual cluster and be insignificant with respect to the database size. In addition, if the sizes of the disjuncts are restricted to at most two (that is, clauses such as $P(a) \vee P(b)$ may be in the database, but larger disjuncts such as $P(c) \vee P(d) \vee P(e)$ do not appear, answers to queries may be found in polynomial time [3].

Topics to be addressed in the future include the extension of the results to larger classes of databases and to various semantics of disjunctive deductive databases.

Acknowledgements

We greatly appreciate the financial support of the National Science Foundation, provided under the grant Nr. IRI-89-16059, the Air Force Office of Scientific Research, provided under the grant Nr. AFOSR-91-0350, and the Fulbright Scholar Program that made this work possible. This work was carried out while the first author was a visiting scientist at the University of Maryland Institute for Advanced Computer Studies (UMIACS). The support of UMIACS is also appreciated.

References

[1] C.-L. Chang and C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, 1973.

[2] R. Fagin, J. Ullman, and M. Vardi. Contributions to the view update problem. In *Proc. of the Sixth Intl. Conf. on Logic Programming*, pages 398–415, 1989.

[3] J.A. Fernandez, Z.A. Khandakar, and J. Minker. A tractable class of disjunctive deductive databases. In *Proc. Workshop on Deductive Databases, Joint International Conference and Symposium on Logic Programming (JICSLP'92)*, Washington, D.C., Nov. 1992.

[4] J.A. Fernández and J. Minker. Disjunctive deductive database. In *3rd International Conference on Logic Programming and Automated Reasoning*, pages 332–356, July 1992. Invited Paper.

[5] J.A. Fernández and J. Minker. Semantics of disjunctive deductive databases. In *Proceedings of the International Conference on Database Theory*, pages 332–356, 1992. (Invited Paper).

[6] J.A. Fernández and J. Minker. Theory and algorithms for disjunctive deductive databases. *Programmirovanie*, N 3:5–39, 1993. (also appears as University of Maryland Technical Report,CS-TR-3223, UMIACS-TR-94-17,1994. Invited Paper in Russian).

[7] 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.

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

[9] J. Grant and J. Minker. Answering queries in indefinite databases and the null value problem. In P. Kanellakis, editor, *Advances in Computing Research: The Theory of Databases*, pages 247–267. 1986.

[10] K.C. Liu and R. Sunderraman. Indefinite and maybe information in relational databases. *ACM Transactions on Database Systems*, 15(1):1–39, 1990.

[11] K.C. Liu and R. Sunderraman. On representing indefinite and maybe information in relational databases. In *Proceedings of IEEE Data Engineering*, pages 495–502, Los Angeles, 1990.

[12] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT Press, 1992.

[13] J. Minker. On indefinite databases and the closed world assumption. In *Proceedings of 6th Conference on Automated Deduction*, pages 292–308, New York, 1982.

[14] J. Minker. Toward a foundation of disjunctive logic programming. In *Proc. North American Conference on Logic Programming*, pages 1215–1235, 1989. (Invited Paper).

[15] J. Minker. An overview of nonmonotonic reasoning and logic programming. *Journal of Logic Programming*, 17(2, 3, and 4):95–126, November 1993.

[16] A. Rajasekar, J. Lobo, and J. Minker. Skeptical reasoning and disjunctive programs. In *Proceedings of First International Conference on Knowledge Representation and Reasoning*, pages 349–357. Morgan-Kaufmann, 1989.

[17] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum, New York, 1978.

[18] F. Rossi and S. Naqvi. Contributions to the view update problem. In *International Conference on Logic Programming*, pages 398–415, Lisbon, 1989.

[19] M. Suchenek. First-order syntactic characterizations of minimal entailment, and Herbrand entailment. *Journal of Automated Reasoning*, 10:237–263, 1993.

[20] A. Yahya, J. A. Fernandez, and J. Minker. Ordered model trees: a normal form for disjunctive deductive databases. Technical Report UMIACS–TR–93–14 and CS–TR–3034, University of Maryland Institute for Advance Computer Studies, College Park, MD 20742, 1993.

[21] A. Yahya and L.J. Henschen. Deduction in Non-Horn Databases. *J. Automated Reasoning*, 1(2):141–160, 1985.

[22] A. Yahya and J. Minker. Query answering in partitioned disjunctive deductive databases. Technical Report UMIACS–TR–93–14 and CS–TR–3034, University of Maryland Institute for Advance Computer Studies, College Park, MD 20742, 1993.