INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67, D–80538 München

Ludwig-
Maximilians-
Universität
München

A Goal-Driven Approach to Efficient Query Processing in Disjunctive Databases

Adnan H. Yahya

An earlier version was presented at the Dagstuhl Seminar on
Disjunctive Logic Programming and Databases: Nonmonotonic Aspects,
1-5 July, 1996, Dagstuhl, Germany, Report Nr.150.
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1996-12, July 1996

A Goal-Driven Approach to Efficient Query Processing in Disjunctive Databases

Adnan Yahya
Institut für Informatik, Ludwig-Maximilians-Universität München, Germany
Electrical Engineering Department, Birzeit University, Birzeit, Palestine
yahya@informatik.uni-muenchen.de

Abstract

Generally, proof procedures based on model generation perform bottom-up processing of clauses. Several algorithms for generating (minimal) models for disjunctive theories were advanced in the literature. Used for query answering, bottom-up procedures tend to explore a much larger search space than is strictly needed. On the other hand, top-down processing usually has a more focused search space which can result in more efficient query answering. In this paper we establish a strong connection between model generation and clause derivability that allows us to use a (minimal) model generating procedure for evaluating queries in a top-down fashion. In contrast to other methods our approach requires no extensive rewriting of the input theory and introduces no new predicates. Rather, it is based on a certain *duality* principle for interpreting logical connectives achieved by reversing the direction of implication connectives in the clauses representing both the theory and the negation of the query. The application of a generic (minimal) model generating procedure to the transformed clause set results in top-down query answering. We explain the reasoning behind the transformation and show how the duality approach can be utilized for refined query answering by specifying the conditions under which the query becomes derivable from the theory. Our initial testing points to a clear efficiency advantage of the advanced approach as compared to traditional bottom-up processing for the class of positive queries against a disjunctive database.

1 Introduction

Model generation received much attention in the literature both as a basis for refutation procedures and to generate model representations for logic programs under different semantics [17, 9, 16, 26]. Several efficient implementations were reported in the literature that can serve as a basis for (minimal) model-based reasoning [5, 11, 20]. Current work is directed at introducing enhancements to get more general systems in terms of the class of theories being treated. Model generation is generally based on the bottom-up evaluation of clauses. When used for refutations (e.g. query answering) model generation procedures tend to explore a search space much larger than that required to generate a refutation. In a sense they tend to generate answers to all possible queries rather than to the query under consideration [23]. Improvements directed towards early detection of contradictions were incorporated into the procedure resulting in substantial performance improvements [5, 11, 20, 24].

On the other hand, top-down methods for query answering tend to perform more focused search for refutations by exploiting the information contained in the query, a desirable property. In a deductive database context this can mean substantial time savings in query answering. Several approaches based on transforming the set of clauses or using certain data structures and special algorithms to achieve top-down processing of the theory for a given query are available. Many of the transformations involve the introduction of new predicates and/or the rewriting of clauses to enable the more focused search [1, 7, 21, 23, 27].

In this paper we offer an alternative method to achieve a top-down evaluation of queries posed to disjunctive theories using a generic bottom-up model generating procedure. The method is based on a certain concept of *duality* and utilizes the implicit modification of the interpretation of logical connectives resulting from the reversal of the implication sign of clauses. Otherwise, no modification of the theory is needed and effectively the same model generating procedure (with a manual or automatic selection of direction) can be used for both bottom-up and top-down processing for query answering.

Our selection of a model generation proof procedure is motivated by the wealth of information it returns even in cases when no refutation is found. This will help us explore the potential of the advanced approach. For example, we will utilize the information returned by the proof procedure to allow for refined query answering by specifying the optimal conditions under which the query becomes derivable from the theory after an update. In addition, by virtue of using a generic minimal model generating procedure, the advanced approach will be able to benefit from all the efficiency enhancement modifications and generalizations introduced to the model generating process. In the context of a deductive database, we show that our approach will make it possible to separate the query answering process into two stages: the first is the generation of the checks needed to ensure the derivability of the query, which is based on the interaction of the query with the Intentional part of the database (IDB), from the theory. The second is the actual look-up of these conditions in the Extensional part of the database (EDB). In this regard it behaves like a compiled approach to query answering [10]. Alternatively, one could integrate the two stages to get shallower computations with the context and associated costs determining the exact choice.

Since our approach is based on model generation, it suffers also from all the shortcomings of such procedures. We address these limitations and discuss some of the ways they can be lifted.

The remainder of the paper is organized as follows. In the next section we give some relevant definitions and background material. In Section 3 we define our procedure for a restricted class of disjunctive theories: that of ground databases with no denial rules. In Section 5 we offer some interpretation and implementation notes on the advanced approach to explain the sources of its performance, potential and limitations. In Section 4 we show how to relax the restrictions on our procedure and the problems involved. In Section 6 we compare our approach with others advanced in the literature and point to the possible directions of further research and development.

2 Preliminaries and Background Material

In this section we review some of the concepts related to query answering in disjunctive deductive databases. We assume familiarity with the basic concepts as outlined in [15] and therefore limit ourselves to the basic material needed for the results presented in this paper.

Definition 2.1 *A* disjunctive deductive database (DDDB)*, DB, is a set of clauses of the form:*

$$C = A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \ldots \wedge B_n,$$

where $m, n \geq 0$ and the As and Bs are atoms in a First Order Language (FOL) $\mathcal{L}$ with no function symbols.

By $Head(C)$, ($Body(C)$) we denote the set of atoms in the head (body) of a clause C of DB. We use the atom $\perp$ (false) to refer to the empty head and atom $\top$ (true) to refer to the empty body.

At the expense of a slightly abused notation we write a clause $S_1 \rightarrow S_2$ where $S_1 = \{B_1, \ldots, B_n B_n\}$, $S_2 = \{A_1, ..., A_m\}$ are sets of atoms and interpret it as the conjunction of atoms in S_1 implies the disjunction of atoms of S_2 ($B_1 \wedge ... \wedge B_n \rightarrow A_1 \vee \cdots \vee A_m$).

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 $\mathcal{L}$. A *Herbrand interpretation* is any subset of HB_{DB}. A Herbrand model of DB, M, is a Herbrand interpretation such that $M \models DB$ (all clauses of DB are *true* in M). M is *minimal* if no proper subset of M is a model of DB. The set of all minimal models of DB is denoted by $\mathcal{MM}(DB)$.

In this paper we use (possibly subscripted) x for variables, a, b, c, d, e, f, g, h for constants, P, R, S, U, V for predicates (of different arities), Q to denote a query. In examples where only ground atoms are used, we may replace the atom by a constant (e.g. replace $P(a)$ by a) to avoid obscuring the relevant material.

Definition 2.2 *A clause C is range restricted if every variable occurring in the head of C also appears in the body of C. A database is range restricted iff all its clauses are range restricted.*

Definition 2.3 *If $C = A_1 \vee ... \vee A_n$ is a disjunction of atoms, then by $Neg(C)$ we denote the set of clauses in implication form $Neg(C) := \{A_1 \rightarrow \perp, ..., A_n \rightarrow \perp\}$. If $M = \{A_1, ..., A_n\}$ is a finite interpretation then $Neg(M)$ denotes the clause in implication form $Neg(M) = A_1 \wedge ... \wedge A_n \rightarrow \perp$.*

Definition 2.4 *A DDDB, DB, can be partitioned into three sets of clauses:*

1. *The* extensional part *(EDB) a positive disjunctive database corresponding to base relations and containing facts (clauses with empty bodies, positive clauses).*
2. *The* intensional part *(IDB) corresponding to view definitions. The rules of IDB can be used to derive new pieces of information from the extensional part of the database. They have the form of clauses with nonempty heads and bodies.*

3. *The* integrity constraints *(IC_{DB}). This is a set of rules that are used to ensure that the theory consisting of the first two components satisfies certain properties. These can be denial rules (clauses with empty heads) or general rules (clauses with nonempty heads)*[1]*.*

The main results of this paper are presented using a model generator with certain properties as the proof procedure [5, 29]. We extensively utilize denial clauses (rules with empty heads) to restrict the search space for models. Since denial clauses have only positive body literals they represent purely negative clauses.

Definition 2.5 *Given a DDDB, DB and a model generating procedure $\mathcal{P}$, by $\mathcal{P}(DB)$ we denote the result returned by $\mathcal{P}$ run with DB as input. We say that $\mathcal{P}$ is*

1. *Sound: if it returns only models of DB: $\forall M \in \mathcal{P}(DB), M \models DB$.*
2. *Minimal-Model sound if it returns only minimal models of its input: $\mathcal{P}(DB) \subseteq \mathcal{MM}(DB)$.*
3. *Complete: if it returns all the minimal models of DB: $\mathcal{MM}(DB) \subseteq \mathcal{P}(DB)$.*

To make the paper complete, next we give a brief description of successively refined model generating procedures that are sound and complete [5]. Given a DDDB, DB, each of these procedures constructs a tree (model tree) with the ground unit clauses in each root-to-leaf branch representing a model of DB. The completeness implies that the tree has at least one branch representing each minimal model of DB.

Starting from $\top$ as the root, the procedure expands a tree for a range restricted DDDB, DB, by applying the following expansion rules:

Definition 2.6 (expansion rules) *Let DB be a DDDB. If the elements above the horizontal line are in a branch then it can be expanded by the elements below the line.*

Positive Unit Hyper-Resolution (PUHR) Rule:

$$\frac{\begin{array}{c} B_1 \\ \vdots \\ B_n \end{array}}{E\sigma}$$

Splitting Rule:

$$\frac{E_1 \vee E_2}{E_1 \quad | \quad E_2}$$

where σ is a most general unifier of the body of a clause $(A_1 \wedge ... \wedge A_m \rightarrow E) \in DB$ with $(B_1, ..., B_n)$. $\{A_1, ..., A_m\}\sigma = \{B_1, ..., B_n\}$.

Note that the splitting rule is always applied to *ground* disjunctions. This is possible since the theory is range restricted. The head is always ground when the body is ground (or empty).

Definition 2.7 (model tree construction) *A Model Tree for a DDDB, DB, is a tree whose nodes are sets of ground atoms and disjunctions of ground atoms constructed as follows:*

[1]Without loss of generality, one may assume that the sets of predicates in EDB and IDB do not intersect. That is, DB has no hybrid predicates [15]. We don't make this assumption here but discuss how such a separation can improve the efficiency of the algorithms presented in this paper.

1. *$\{\top\}$ is the top (root) node of the tree.*

2. *If T is a leaf node in the tree being constructed for DB such that an application of the PUHR rule (respectively splitting rule) is possible to yield a formula E (respectively, two formulas E_1 and E_2) not subsumed by an atom already in the branch, then the branch is extended by adding the child node $\{E\}$ (respectively the two child nodes $\{E_1\}$ and $\{E_2\}$) as successor(s) to T.*

While the above definition imposes no order on atom expansion, we elect to maintain an order that will later be exploited for defining the properties of the generated tree.

Definition 2.8 *(***conventions for model generation***) When expanding a model tree we assume that the procedure adheres to the following rules*[2]*:*

1. *Always select E_1 of a disjunction to be atomic.*

2. *Expand the leftmost atom of a disjunction first.*

3. *As a result of items 1 and 2 atoms of the clause are expanded from left to right (by adding the remainder of the clause, if any, to the top of the theory to be processed in the sibling branch).*

We always expand left branches of the model tree first. Our interest is only in branches with no occurrences of $\perp$ (open branches). The branch expansion is stopped when *false* ($\perp$) is added to a branch (the branch closes). Only (ground) disjunctions that are not subsumed in the branch are expanded to avoid unnecessary expansions. A branch represents the interpretation in which all (ground) unit clauses on that branch are assigned the truth value *true*. For the class of of range restricted disjunctive deductive databases with finite models the tree defined by such a procedure is *sound* in the sense that it generates only models of the theory and *complete* in the sense that it has branches representing all minimal models of DB. However, not all branches represent minimal models [5].

Example 1 *Let DB be the following set of clauses:*

$$\begin{array}{ll} \top \rightarrow P(a) \vee P(b) \qquad\qquad & P(a) \rightarrow P(b) \vee P(d) \\ \top \rightarrow P(a) \vee P(c) & P(b) \rightarrow P(a) \vee P(d) \end{array}$$

Figure 1 is a model tree for DB. The minimal model $\{P(a), P(b)\}$ of DB is generated twice. The tree also has a branch with the nonminimal model $\{P(a), P(b), P(c)\}$. Among others, all minimal models of DB, i.e. $\{P(a), P(b)\}$, $\{P(a), P(d)\}$, and $\{P(b), P(c), P(d)\}$ are generated.

Further, it was shown that replacing the splitting rule by the following one called *Complement Splitting Rule* preserves the completeness and soundness of the model generating procedure [5].

Definition 2.9 (complement splitting rule)

[2]These conventions are adopted in the implementation reported in [5]. They correspond to a left-to-right, depth-first traversal of the search space. However, this is not the only possible expansion ordering. Breadth-first search can be adopted for the same purpose. Comparing the merits of these two approaches is beyond the scope of this paper.

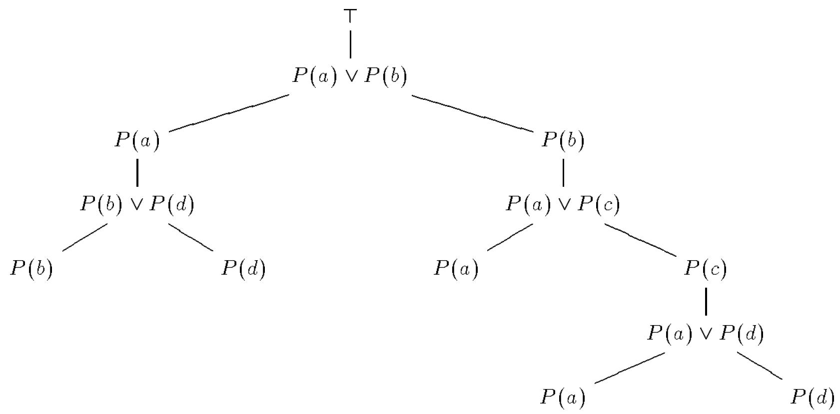

Figure 1: A Model Tree for Example 1 (with nonminimal and duplicate models).

$$\begin{array}{l|r} \multicolumn{2}{c}{E_1 \vee E_2} \\ \hline E_1 & E_2 \\ Neg(E_2) & \end{array}$$

The adoption of this rule tends to reduce the search space by closing (adding *false* to) branches before they grow into complete nonminimal or duplicate models. Besides, the first (leftmost) model generated using this rule is minimal.

Example 2 *Let DB be the set of clauses of Example 1, i.e.:*

$$\begin{array}{ll} \top \rightarrow P(a) \vee P(b) & \qquad P(a) \rightarrow P(b) \vee P(d) \\ \top \rightarrow P(a) \vee P(c) & \qquad P(b) \rightarrow P(a) \vee P(d) \end{array}$$

Figure 2 gives the model tree for DB. Clauses not in the original theory are given in square brackets. The models of this tree are $\{P(a), P(d)\}$, $\{P(b), P(c), P(a)\}$, $\{P(b), P(a)\}$, and $\{P(b), P(c), P(d)\}$. Note that although some are not minimal, no duplicates are returned and the first model is minimal.

If additionally, for each minimal model, M, generated so far we augment the theory by the negation of M, $(Neg(M) = A_1 \wedge ... \wedge A_m \rightarrow \perp \ if \ M = \{A_1, ..., A_m\})$, then we achieve a model generating procedure that is *minimal model sound* and *complete*. It returns all and only minimal models of its input theory.

Example 3 *Figure 3 gives the search spaces of the minimal model generation procedure for the set of clauses of Examples 1 and 2, i.e.:*

$$\begin{array}{ll} \top \rightarrow P(a) \vee P(b) & \qquad P(a) \rightarrow P(b) \vee P(d) \\ \top \rightarrow P(a) \vee P(c) & \qquad P(b) \rightarrow P(a) \vee P(d) \end{array}$$

Note that all models returned by the procedure are minimal.

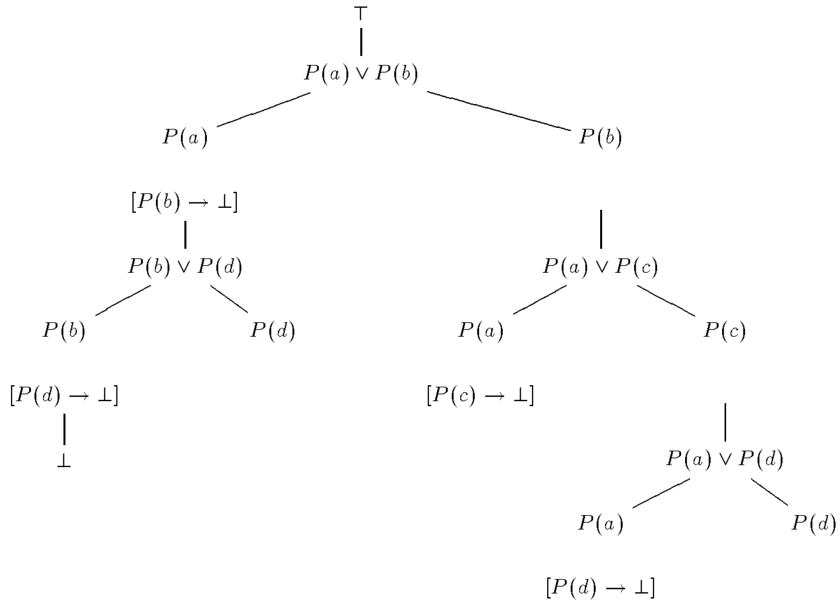

Figure 2: The Model Tree with Complement Splitting for Example 2.

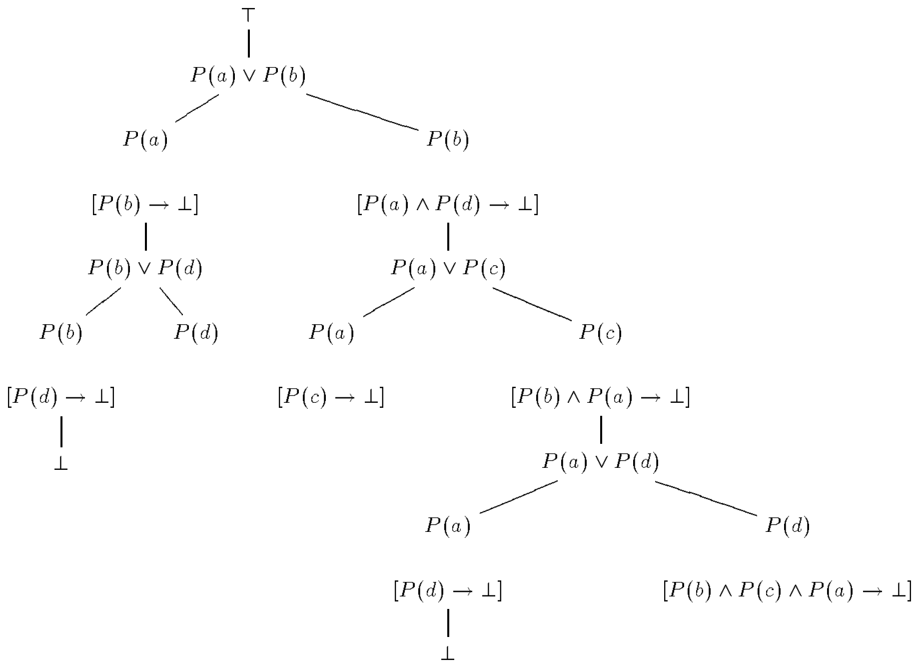

Figure 3: A Run of the Minimal Model Generation Procedure MM-Satchmo for Example 3.

In [29] and [5] sound and complete minimal model generation procedures were given for ground and RR theories, respectively. [5] contains a Prolog implementation of a series of procedures, for the class of RR theories with finite minimal models and no body negation, called: Satchmo for the program with splitting [17], CS-Satchmo for the implementation with complement splitting and MM-Satchmo for the implementation with model minimization (by including negation of generated minimal models).

3 The Duality Approach

We are interested in answering queries in the context of a DDDB. We limit our consideration to the class of *positive* queries. This includes *disjunctive* queries which are disjunctions of atomic queries and *conjunctive* queries which are conjunctions of atomic queries. Atomic queries are a special case of either of these classes. Other positive queries can be reduced to a conjunction of disjunctive queries (positive clauses). For now, queries are assumed ground and have *yes/no* answers. Additionally, we treat both clauses and models as sets of ground atoms. A set of atoms is interpreted disjunctively when it is referred to as a clause and conjunctively when it is referred to as a model. We can even talk about equality of a clause and a model which is to be interpreted as their having the same underlying set. Unless otherwise stated, we assume that the database under consideration is consistent.

3.1 Disjunctive Queries

In this section we present the main results of our approach for the case of disjunctive queries: queries that are disjunctions of atoms. Later we show that the results easily extend to other positive queries.

3.1.1 Goal Set Expansion

Theorem 1 *Let DB be a ground Disjunctive Deductive Database (with no denial constraints) and $Q = q_1 \vee ... \vee q_n$ be a disjunctive query. If $DB \vdash Q$ then a clause subsuming Q occurs in the head of some clause of DB.*

Proof: Q is derivable from DB iff it is *true* in every model of DB. Assume that Q is derivable from DB and there exists no clause $C \in DB$ such that $Head(C)$ subsumes Q. To show contradiction we construct a model in which no atom of Q is present. Consider $M = HB(DB) \setminus \{q_1, ..., q_n\}$. It satisfies all clause heads since none of them consists entirely of atoms in Q. M is a model of DB. ■

That Theorem 1 doesn't hold in the presence of denial constraints is clear from the following example:

Example 4 *Consider $DB_1 = \{P(d) \rightarrow P(a) \vee P(b) \vee P(c), P(d)\}$ and $DB_2 = \{P(d) \rightarrow P(a) \vee P(b) \vee P(c), P(d), P(c) \rightarrow\}$ and $Q = P(a) \vee P(b)$. Clearly $DB_2 \vdash Q$ but $DB_1 \not\vdash Q$.*

In view of Theorem 1, if a subclause of Q is in the (Extensional) database then the query is *true*. This can be verified by inspection. If not, the process of answering Q will require more work. We need to look for a clause C with a subclause of Q as (instance of) the head and show that C will derive Q at least in all models of DB in which Q may not be satisfied, ensuring the derivability of Q. We can do that, for example, by showing that atoms of the body of C are all elements of the EDB. However, this is too strong a condition: it is sufficient but not necessary.

A necessary and sufficient condition is given by the following rephrasing of results reported in [7, 14, 21].

Definition 3.1 *Let DB be a ground Disjunctive Deductive Database (with no denial constraints), $Q = q_1 \vee ... \vee q_n$ be a disjunctive query against DB and $C = B_1 \wedge ... \wedge B_k \longrightarrow A_1 \vee ... \vee A_l$ be a clause in DB. We define $\mathcal{G}_C^Q$, the goal clause set of Q relative to C, as:*

$$\mathcal{G}_C^Q = \begin{cases} \{q_1 \vee ... \vee q_n \vee B_1, ..., q_1 \vee ... \vee q_n \vee B_k\} & \text{if } \{A_1, ..., A_l\} \subseteq \{q_1, ..., q_n\} \\ \{q_1 \vee ... \vee q_n\} & \text{if } \{A_1, ..., A_l\} \not\subseteq \{q_1, ..., q_n\} \end{cases}$$

By $Body(C) \vee Q$ we denote the set $\{q_1 \vee ... \vee q_n \vee B_1, ..., q_1 \vee ... \vee q_n \vee B_k\}$

In essence, $\mathcal{G}_C^Q$ is meant to define a set of clauses, that need to be derivable from DB to prove that Q follows from DB in as far as clause C is concerned. C may contribute to the derivability of Q only when its head subsumes Q. While we are replacing the provability of one clause Q by the provability of several, the latter are potentially longer disjunctions and, therefore, each of them has a better chance of being proved than Q itself.

Clearly a set of clauses is derivable from DB if and only if all of its members are derivable from DB. Since a clause is always satisfied whenever any clause subsuming it is satisfied it is possible to delete all nonminimal elements from the clause set $\mathcal{G}_C^Q$. In particular, note that if $\{A_1, ..., A_l\} \subseteq \{q_1, ..., q_n\}$ and $\{B_1, ..., B_k\} \cap \{q_1, ..., q_n\} \neq \emptyset$ as well then $\mathcal{G}_C^Q$ is subsumed by the single clause $\{q_1 \vee ... \vee q_n\}$ and therefore $\mathcal{G}_C^Q = \{q_1 \vee ... \vee q_n\}$ in this case too. We make this explicit when defining the consequence operator $\mathcal{T}_{DB}^g$ later in this section (Definition 3.2).

Theorem 2 *Let DB be a ground Disjunctive Deductive Database (with no denial constraints) and $Q = q_1 \vee ... \vee q_n$ be a disjunctive query. $DB \vdash Q$ if and only if there exists a clause C in DB such that $Head(C)$ subsumes Q and the formula $(Body(C) \vee Q)$ is derivable from DB. Or equivalently if and only if $\{(B \vee Q) \mid B \in Body(C)\}$ is derivable from DB, where B is an atom in $Body(C)$.*

Proof: Assume that Q is not *true* in some model of DB, say M. Then from $M \models (Body(C) \vee Q)$ we get that $Body(C) \subseteq M$ and consequently $Head(C) \subseteq Q$ is *true* in M. Q is *true* in M by an application of clause C, a contradiction.

The other direction follows directly from Theorem 1. Assume that for all clauses C of DB, $(Body(C) \vee Q)$ is not derivable from DB. Consider a suitable model M of DB: there is a clause $(B \vee Q)$ for some $B \in Body(C)$ such that both B and Q are false in M. Q is not derivable from DB (it is false in M). ■

Corollary 1 *[14] Under the conditions of Theorem 2:*

1. *$(Body(C) \vee Q)$ is derivable from DB iff $(Body(C) \vee Q)$ is derivable from $DB \setminus \{C\}$.*

2. *If* $Body(C) \cap Q \neq \emptyset$ *then* $DB \vdash Q$ *iff* $DB \setminus \{C\} \vdash Q$.

3. $DB \not\vdash Q$ *if and only if for any clause* C *in* DB *such that* $Head(C)$ *subsumes* Q, $(Q \vee Body(C))$ *is not derivable from* DB.

Proof: 1. If $Body(C) \vee Q$ is derivable from $DB \setminus \{C\}$ then Q is derivable from DB by the application of clause C.
Let $Body(C) \vee Q$ be derivable from DB. Assume that $DB \setminus \{C\} \not\vdash (Body(C) \vee Q)$. There exists a model M such that $M \models (DB \setminus \{C\})$ but $M \not\models (Body(C) \vee Q)$. $Body(C) \cap M = \emptyset$. $M \models C$ and consequently $M \models DB$ and therefore $M \models (Body(C) \vee Q)$. A contradiction.

2. Immediate since in this case $Body(C) \vee Q$ is subsumed by Q. That is $Q \in \{(B \vee Q) \mid B \in Body(C)\}$.

3. straightforward.

■

If all the resulting clauses are subsumed by elements in EDB then the task is over. Otherwise attempts must be made to prove the new goal set by repeated application of Theorem 2. Consider the example:

Example 5 *Let* $DB_3 = \{C_1 = a \vee b, C_2 = c \vee d, C_3 = a \wedge d \rightarrow e \vee b, C_4 = b \wedge d \rightarrow f \vee c, C_5 = c \rightarrow g\}$. *Let* $Q_1 = b \vee e \vee g$, $Q_2 = b \vee c \vee f$ *and* $Q_3 = g \vee f$.

1. $Q_1 = b \vee e \vee g$: *Applying the clause* C_3 *to* Q_1 *yields the set* $\{C_6 = b \vee e \vee g \vee a, C_7 = b \vee e \vee g \vee d\}$. C_6 *is derivable from EDB (subsumed by* C_1*) while* C_7 *is not so we apply clause* C_5 *to* C_7 *to yield* $\{C_8 = b \vee e \vee g \vee d \vee c\}$ *which is derivable from EDB (subsumed by* C_2*) and consequently so is* Q_1.

2. $Q_2 = b \vee c \vee f$: *Applying the clause* C_4 *to* Q_2 *yields the set* $\{C_9 = c \vee f \vee b, C_{10} = b \vee c \vee f \vee d\}$. C_{10} *is derivable from EDB while* C_9 *is not. All attempts to solve* C_9 *(through* C_4 *fail and therefore* $DB \not\vdash Q_2$. *Note that we could have generated* C_9 *alone since* $Q_2 \cap Body(C_4)$ *is not empty without changing the final result.*

3. $Q_3 = g \vee f$: *Applying the clause* C_5 *to* Q_3 *yields the set* $\{C_{11} = c \vee g \vee f\}$. C_{11} *is not derivable from EDB. Applying the clause* C_4 *to* C_{11} *yields the set* $\{C_{12} = c \vee g \vee f \vee b, C_{13} = c \vee g \vee f \vee d\}$. C_{13} *is derivable from EDB while* C_{12} *is not. No clauses are available to extend* C_{12} *so* $DB \not\vdash Q_2$. *It is easy to see how the results of Corollary 1 hold for Example 5.*

In effect, Theorem 2 and Corollary 1 suggest a simple approach to proving Q. Keep generating clauses subsumed by Q until a sufficient group of such clauses is subsumed by clauses in DB. The following points need to be emphasized in this regard:

- Each of the clauses of the goal clause set, $\mathcal{G}_C^Q$, is at least as long as Q. The derivability of the query is expressed in terms of the derivability of other positive clauses. Since, by definition, the only component of the database that contains positive facts is the EDB then the derivability of the query must ultimately be reduced to simple searches in EDB for clauses subsuming each of the elements of a *final* goal set. The expansion is continued until the goal clause set is found in the EDB or until no more applications of the rules of DB are possible.

- The top-down nature of the expansion process since the clauses are selected on their (head) matching the goal clause (current query). This may be instrumental in reducing the search space as compared with the case of using a bottom-up query evaluation procedure e.g. based on minimal model generation.
- The clause (instance) used for expansion is applied only once and it can be deleted from the database during the rest of the search process. This results in reducing the size of the IDB component of the database at the expense of increasing the size/number of the clauses that need to be tested for (simultaneous) derivability from EDB.

It is clear that derivability of the answer Q is being *reduced* to the derivability of a set of larger clauses with the final aim of having these larger clauses looked up in the extensional part of the database (EDB). As far as the nature of $\mathcal{G}_C^Q$ content is concerned, it is preferable to have the testing performed on longer clauses. If this set contains clauses subsuming each other then we can limit our testing to the minimal components alone. If the test succeeds, subsumed clauses will be derivable as well.

Definition 3.2 *Let $DB = EDB \cup IDB$ be a ground Disjunctive Deductive Database (with no denial constraints), C be a ground positive clause and $\mathcal{S}$ be a set of ground positive clauses. We define the consequence operator $\mathcal{T}_{DB}^g$ that maps sets of positive clauses into sets of positive clauses of the disjunctive Herbrand base of DB as follows:*

$$\mathcal{T}_{DB}^g(\{C\}) = \begin{cases} \{C \vee B| & \exists C_1 \in IDB \ s.t. Head(C_1) \subseteq C, Body(C_1) \cap C = \emptyset, B \in Body(C_1)\}. \\ \{C\} & Otherwise \end{cases}$$

$$\mathcal{T}_{DB}^g(\mathcal{S}) = \bigcup_{C \in \mathcal{S}} \mathcal{T}_{DB}^g(\{C\})$$

$$\mathcal{T}_{DB}^g \uparrow 0(\mathcal{S}) = \mathcal{S},$$

$$\mathcal{T}_{DB}^g \uparrow \alpha(\mathcal{S}) = \mathcal{T}_{DB}^g(\mathcal{T}_{DB}^g \uparrow (\alpha - 1)(\mathcal{S})) \text{ for successor ordinal } \alpha,$$

$$\mathcal{T}_{DB}^g \uparrow \alpha(\mathcal{S}) = lub\{\mathcal{T}_{DB}^g \uparrow \beta(\mathcal{S}) : \beta < \alpha\} \text{ for limit ordinal } \alpha,$$

$$lfp(\mathcal{T}_{DB}^g) = \mathcal{T}_{DB}^g \uparrow \omega(\mathcal{S}), \text{ where } \omega \text{ is the first limit ordinal[8, 12, 25].}$$

We may assume that subsumed clauses are deleted (minimization is performed) at each stage. This is so since subsumed clauses are automatically derivable from DB when the subsuming clauses are[3]. However, we elect not to adopt this assumption to make our discussion as general as possible. Note that our expansion process differs from that of [21] by explicitly prohibiting the generation of subsumed goals through applicable clauses with nonempty intersection of the clause body and the current goal.

[3]Note, however, that the partial order on sets of interpretations $\mathcal{I}$ and $\mathcal{J}$ is defined through the set inclusion relationship ($\subseteq$) between the *minimal elements* of the sets. That is, $\mathcal{J} \sqsubseteq \mathcal{I}$ if and only if $\forall I \in Min(\mathcal{I}), \exists J \in Min(\mathcal{J}), J \subseteq I$, where $Min(\mathcal{I}) = \{I | I \in \mathcal{I}, \ \not\exists I' \in \mathcal{I} \ s.t. \ I' \subset I\}$. So, rather than minimizing at every step, we minimize after the complete computation. Clearly, nonminimal elements will have no effect on the partial order and the results of [8] hold here as well.

Example 6 *Consider* $DB =$
$\{C_1 = a \vee b, C_2 = c \vee d, C_3 = a \wedge d \rightarrow e \vee b, C_4 = b \wedge d \rightarrow f \vee c, C_5 = c \rightarrow g\}$.
Let $Q_1 = b \vee e \vee g$, $Q_2 = b \vee c \vee f$ *and* $Q_3 = g \vee f$ *as in Example 5.*

$\mathcal{T}^g_{DB} \uparrow 1(\{Q_1\}) = \{C_6 = b \vee e \vee g \vee a, C_7 = b \vee e \vee g \vee d, C_8 = b \vee e \vee g \vee c\}$.
$\mathcal{T}^g_{DB} \uparrow 2(\{Q_1\}) = \{C_9 = b \vee e \vee g \vee a \vee c, C_{10} = b \vee e \vee g \vee d \vee c\}$.
$\mathcal{T}^g_{DB} \uparrow \omega(\{Q_1\}) = \mathcal{T}^g_{DB} \uparrow 2(\{Q_1\})$.

$\mathcal{T}^g_{DB} \uparrow 1(\{Q_2\}) = \{C_{11} = b \vee c \vee f\}$.
$\mathcal{T}^g_{DB} \uparrow \omega(\{Q_2\}) = \mathcal{T}^g_{DB} \uparrow 1(\{Q_2\})$.

$\mathcal{T}^g_{DB} \uparrow 1(\{Q_3\}) = \{C_{12} = g \vee c \vee f\}$.
$\mathcal{T}^g_{DB} \uparrow 2(\{Q_3\}) = \{C_{13} = g \vee c \vee f \vee b, C_{14} = g \vee c \vee f \vee d\}$.
$\mathcal{T}^g_{DB} \uparrow \omega(\{Q_3\}) = \mathcal{T}^g_{DB} \uparrow 2(\{Q_3\})$.

Note that the expansion can produce nonminimal elements. For example, if we let $DB' = DB \cup \{C_0 = f \wedge d \rightarrow a \vee g\}$ *then applying* C_0 *to* C_9 *gives* $\mathcal{T}^g_{DB'} \uparrow \omega(\{Q_1\}) = \{C_{14} = b \vee e \vee g \vee a \vee c \vee d, C_{15} = b \vee e \vee g \vee a \vee c \vee f, C_{10} = b \vee e \vee g \vee d \vee c\}$.
C_{14} *is nonminimal. It is subsumed by* C_{10}.

Later we prove that if we start with clause (disjunctive query) Q then $\mathcal{T}^g_{DB} \uparrow \omega(\{Q\})$ is the set of clauses that all need to be subsumed by EDB in order to prove Q.

3.1.2 The Duality Transformation

As outlined above, the expansion procedure has many of the elements of model generation. Our aim is to employ a (bottom-up) model generating procedure to simulate the (top-down) computation suggested by the proved theorems. However, since the elements we are dealing with in the theorems are clauses and clause subsumption rather than models and clause satisfiability, some changes are needed to account for this difference.

As a first step, starting from the query to be proved, Q, we will try to construct a set of ground clauses (*clause construction*) that are to be tested for being subsumed by elements of EDB (*subsumption checking*) to verify the derivability of Q from DB. Later we incorporate the subsumption checking part into the clause construction process to get the required procedure. To explain our approach we need to recall some terminology and definitions.

We recall that a clause $C = Body(C) \rightarrow Head(C)$ is always treated as reading the conjunction of atoms in the set $Body(C)$ implies the disjunction of atoms in the set $Head(C)$. Changing the implication direction therefore will have the automatic effect of exchanging $\vee$ by $\wedge$ and vise versa.

Definition 3.3 *(***dual clause***) Let* $C = Body(C) \rightarrow Head(C)$ *be a clause of a DDDB, DB. We define the dual clause of* C, $C^d = Head(C) \rightarrow Body(C)$. *Clearly,* $Head(C) = Body(C^d)$ *and* $Body(C) = Head(C^d)$. *Note that* $\perp$ *(*$\top$*) in the head (body) of* C *replaced by* $\top$ *(*$\perp$*) in the body (head) of* C^d. *The dual of a set of clauses is the set of the duals of each of its members. In particular,* $IDB^d = \{C^d | C \in IDB\}$ *is the dual of* IDB.

The intended meaning of C^d is that in order to show that the disjunction of atoms in the head of a clause C is derivable from DB one needs to show that all the disjunctions of an element of the body with the head of C are derivable from DB. So read C^d as "to prove the disjunction subsumed by $Body(C^d)$ requires ($\rightarrow$) that the clauses corresponding to each element of $Head(C^d)$ added to that disjunction are provable from DB. This is in line with the interpretation of a disjunctive clause $C = Body(C) \rightarrow Head(C)$ as implying $Body(C) \vee D \rightarrow Head(C) \vee D$ for a positive clause D, under the restriction that $Head(C)$ subsumes D [21, 27]. A different reading that reflects the procedure being developed in this paper will be advanced in paragraph 5.1.

Now, to start the search for the derivability of $Q = q_1 \vee ... \vee q_n$ in DB we take the set $Q^d = \{q_1, ..., q_n\}$[4] and look for a clause $C_i^d \in IDB^d$ such that $Q^d \cap Body(C_i^d) = Body(C_i^d)$. (To use the interpretations terminology,$Body(C_i^d)$ is satisfied in the partial interpretation Q^d). We extend Q^d by one element of $Head(C_i^d)$ at a time to create the new set $\{\{Q^d, B\} | B \in Head(C_i^d)\}$. Now, all the clauses corresponding to the elements of the new set have to be derivable from EDB. If not the same procedure is applied to those which are not. We may, however, keep generating the new elements until all those potentially needed are produced (a fixpoint is reached in the computation) and do the checking at that time. As suggested by Corollary 1 we may also remove a clause of the theory as soon as it is used to generate its goal clause set (expired) or we can keep it in the database. With a view on expanding our procedure to the case of nonground theories we elect to postpone the subsumption checking in the EDB and to keep expired clauses in the database. Note that the outlined procedure has all the elements of model generation for the dualized database.

Formally, we have the following results:

Lemma 1 *Let DB be a ground Disjunctive Deductive Database (with no denial constraints),* $Q = q_1 \vee ... \vee q_n$ *be a disjunctive query against DB and* $C = B_1 \wedge ... \wedge B_k \rightarrow A_1 \vee ... \vee A_l$ *be a clause in DB. Then*

- $\mathcal{MM}(\{C^d\} \cup Q^d) \subset \mathcal{G}_C^Q$
- $\mathcal{MM}(\{C^d\} \cup Q^d) = Min(\mathcal{G}_C^Q)$

Proof: By definition $C^d = A_1 \wedge ... \wedge A_l \rightarrow B_1 \vee ... \vee B_k$. Recall the definition of $\mathcal{G}_C^Q$. Three cases are possible:

1. If $Q^d \cap \{A_1, ..., A_l\} \neq \{A_1, ..., A_l\}$ then $\mathcal{MM}(\{C^d\} \cup Q^d) = \{Q^d\} = \mathcal{G}_C^Q$.
2. If $Q^d \cap \{A_1, ..., A_l\} = \{A_1, ..., A_l\}$ and $Q^d \cap \{B_1, ..., B_k\} \neq \emptyset$ then $\mathcal{MM}(\{C^d\} \cup Q^d) = \{Q^d\} \subset \mathcal{G}_C^Q$ and $Min(\mathcal{G}_C^Q) = \{Q^d\}$ and $\mathcal{MM}(\{C^d\} \cup \{Q^d) = Min(\mathcal{G}_C^Q)$.
3. If $Q^d \cap \{A_1, ..., A_l\} = \{A_1, ..., A_l\}$ and $Q^d \cap \{B_1, ..., B_k\} = \emptyset$ then $\mathcal{MM}(\{C^d\} \cup Q^d) = \{\{Q^d \cup \{B_1\}\}, ...\{Q^d \cup \{B_k\}\}\} = \mathcal{G}_C^Q = Min(\mathcal{G}_C^Q)$.

■

[4] There is a slight abuse of notation here. However, the correspondence is clarified if one looks at Q as the clause $q_1 \vee ... \vee q_n \rightarrow \bot$ the dual of which is $\top \rightarrow q_1 \wedge ... \wedge q_n$ which is equivalent to $Q^d = \{q_1, ..., q_n\}$. The same reasoning applies to motivate having $Q^d = q_1 \vee ... \vee q_n$ when $Q = q_1 \wedge ... \wedge q_n$. Paragraph 5.1 offers a way of looking at the meaning of such a transformation.

Note that only the minimal (relative to set inclusion) elements of $\mathcal{G}_C^Q$ are relevant since they subsume all other clauses of $\mathcal{G}_C^Q$. The nonminimal clauses will be derivable whenever the minimal ones are. The goal clause set can be further extended by applying the elements of IDB to the recently generated set. The process can be continued until no further extension is possible.

Definition 3.4 *Let DB be a ground Disjunctive Deductive Database (with no denial constraints) and $Q = q_1 \vee ... \vee q_n$ be a disjunctive query. Let $IDB_Q^d = Q^d \cup IDB^d = \{q_1, ..., q_n\} \cup \{C^d | C \in IDB\}$. By $\mathcal{MM}(IDB_Q^d)$ we denote the set of minimal models of IDB_Q^d.*

The result of Lemma 1 can be extended further by the following:

Theorem 3 *Let DB be a ground Disjunctive Deductive Database (with no denial constraints) and $Q = q_1 \vee ... \vee q_n$ be a disjunctive query. Let $IDB_Q^d = \{q_1, ..., q_n\} \cup \{C^d | C \in IDB\}$. Let $\mathcal{MM}(IDB_Q^d)$ be the set of minimal models of IDB_Q^d. Then:*

- $\mathcal{MM}(IDB_Q^d) \subseteq \mathcal{T}_{DB}^g \uparrow \omega(\{Q\})$.
- $\mathcal{MM}(IDB_Q^d) = Min(\mathcal{T}_{DB}^g \uparrow \omega(\{Q\}))$.

Proof: (Sketch). $M \in \mathcal{MM}(IDB_Q^d)$. $Q^d \subseteq M$. $\mathcal{T}_{DB}^g \uparrow 0(\{Q\}) = \{Q\}$.
$\mathcal{T}_{DB}^g \uparrow 1(\{Q\}) = \{Q \vee B | \exists C \in IDB \ s.t. Head(C) \subseteq Q, Body(C) \cap Q = \emptyset \ and \ B \ in \ Body(C)\}$. By Lemma 1 the minimal models of $\{C^d\} \cup Q^d$ are in the set $\mathcal{T}_{DB}^g \uparrow 1(\{Q\})$. Using this as the base step i, an induction step can be constructed by applying the operator $\mathcal{T}_{DB}^g$ to the set $\mathcal{T}_{DB}^g \uparrow 1(\{Q\})$ and using Lemma 1 to show that the minimal models of the set consisting of the atoms of each clause generated at step i and a matching clause in IDB^d are among the elements returned by the new application of the operator $\mathcal{T}_{DB}^g$. ■

$M \in \mathcal{MM}(IDB_Q^d)$ is saturated (depth-wise) with regard to the the rules of IDB^d. No rule of IDB^d can be applied to M to extend it further. Whenever a rule is applicable by having its body atoms in M at least one head literal of that rule has an atom of M as well. This follows from the definition of a model.

The set $\mathcal{MM}(IDB_Q^d)$ is also saturated (width-wise) in the sense that all possible expansions are attempted whenever the body of a rule C^d in IDB^d is satisfied by the atoms in a branch. The branch containing no atom of $Head(C^d)$ is developed into a set of branches each of which extends the branch by a single head atom of $Head(C^d)$. If the branch already has an atom of $Head(C^d)$ no extension is performed in accordance with Corollary 1.

Because no further application of the rules of IDB^d is possible, it follows that a clause corresponding to an element of the set $\mathcal{MM}(IDB_Q^d)$ is derivable from DB if and only if it is subsumed by a clause in EDB. That is, Q is derivable from DB if and only if all elements of $\mathcal{MM}(IDB_Q^d)$ are subsumed by clauses in EDB. Note also that $\mathcal{MM}(IDB_Q^d)$ depends only on Q and the generally static IDB and is independent from the dynamically changing EDB.

Theorem 4 *Let DB be a ground Disjunctive Deductive Database (with no denial constraints) and $Q = q_1 \vee ... \vee q_n$ be a disjunctive query. Let $IDB_Q^d = \{q_1, ..., q_n\} \cup \{C^d | C \in IDB\}$. Let $\mathcal{MM}(IDB_Q^d)$ be the set of minimal models of IDB_Q^d. Then: Q is derivable from DB if and only if EDB derives clause M for all $M \in \mathcal{MM}(IDB_Q^d)$: ($EDB \vdash C_M = A_1 \vee ... \vee A_l$ if $M = \{A_1, ..., A_l\}$).*

Proof: • Let M be in $\mathcal{MM}(IDB^d_Q)$ and Q be derivable from DB. We show that C_M, the clause corresponding to M, is derivable from EDB.

Assume that EDB does not derive C_M. There exists a model M' of EDB such that $M' \cap M = \emptyset$. Clearly $M' \not\models Q$, since $Q^d \subseteq M$. We extend M' into a model of DB, M'', such that $M'' \not\models Q$.

All clauses in IDB the dual of which participated in generating M (fired during the generation of M) are trivially satisfied in M' since at least one of their body atoms is not in M'. That is $\{\forall C | C \in IDB, Head(C) \subseteq M\}$, an atom A of $Body(C)$ is in M and therefore $A \notin M'$ since $M' \cap M = \emptyset$.

For any other clause of IDB, say C, if $Body(C)$ is satisfied in M' and $Head(C)$ is not then add an atom $A \notin M$ of $Head(C)$ to M'. Such an A exists since otherwise C would have participated in the derivation of M (and is already satisfied in M'). The resulting M'' is a superset of M', a model of DB, and $M'' \not\models Q$ contradicting the derivability of Q from DB.

- Assume that for all $M \in \mathcal{MM}(IDB^d_Q)$, C_M is derivable from EDB. We show that in this case Q is derivable from DB.

  Let $N \in \mathcal{MM}(DB)$ such that $N \not\models Q$. We describe how we can extend Q^d into an $M \in \mathcal{MM}(IDB^d_Q)$ such that $N \cap M = \emptyset$ and therefore $N \not\models C_M$.

  1. Let $i := 0$ and Let $M^0 := Q^d$;
  2. Clearly, $N \cap M^i = \emptyset$ (recall that Q is a disjunctive query).
     If $M^i \in \mathcal{MM}(IDB^d_Q)$ then exit with $M = M^i$.
     Otherwise there exists a clause $C_i \in IDB$ such that $Head(C_i) \cap M^i = Head(C_i)$.
     Since $N \not\models Body(C_i)$ there must be an atom $A_i \in Body(C_i)$ such that $A_i \notin N$.
     Now let $M^{i+1} := M^i \cup \{A_i\}$; $i := i+1$ and go to step 2.

  Since all models are finite, the process terminates generating an $M \in \mathcal{MM}(IDB^d_Q)$ such that $N \cap M = \emptyset$, $N \models DB$ and $N \not\models M$ contradicting our assumption that all such M are derivable from DB.

■

We apply this theorem to the database of Example 5:

Example 7 *Let* $DB = \{C_1 = a \vee b, C_2 = c \vee d, C_3 = a \wedge d \rightarrow e \vee b, C_4 = b \wedge d \rightarrow f \vee c, C_5 = c \rightarrow g\}$.
Let $Q_1 = b \vee e \vee g$, $Q_2 = b \vee c \vee f$ *and* $Q_3 = g \vee f$.
$IDB = \{C_3 = a \wedge d \rightarrow e \vee b, C_4 = b \wedge d \rightarrow f \vee c, C_5 = c \rightarrow g\}$.
$IDB^d = \{C^d_3 = e \wedge b \rightarrow a \vee d, C^d_4 = f \wedge c \rightarrow b \vee d, C^d_5 = g \rightarrow c\}$.

$$IDB^d_{Q_1} = IDB^d \cup Q^d_1 = \{C^d_3 = e \wedge b \rightarrow a \vee d, C^d_4 = f \wedge c \rightarrow b \vee d, C^d_5 = g \rightarrow c, b, e, g\}$$

$$IDB^d_{Q_2} = IDB^d \cup Q^d_2 = \{C^d_3 = e \wedge b \rightarrow a \vee d, C^d_4 = f \wedge c \rightarrow b \vee d, C^d_5 = g \rightarrow c, b, c, f\}.$$

$$IDB^d_{Q_3} = IDB^d \cup Q^d_3 = \{C^d_3 = e \wedge b \rightarrow a \vee d, C^d_4 = f \wedge c \rightarrow b \vee d, C^d_5 = g \rightarrow c, g, f\}.$$

$\mathcal{MM}(IDB^d_{Q_1}) = \{\{a,b,c,e,g\},\{b,c,d,e,g\}\}$.
Both $a \vee b \vee c \vee e \vee g$ *and* $b \vee c \vee d \vee e \vee g$ *are subsumed by clauses in EDB.* Q_1 *is an answer.*

$\mathcal{MM}(IDB^d_{Q_2}) = \{\{b,c,f\}\}$. *Clause* $b \vee c \vee f$ *is not derivable from EDB and therefore* Q_2 *is not an answer.*

$\mathcal{MM}(IDB^d_{Q_3}) = \{\{b,c,f,g\},\{c,d,f,g\}\}$.
$c \vee d \vee f \vee g$ *is subsumed by* $(c \vee d) \in EDB$ *while* $b \vee c \vee f \vee g$ *is not and so* Q_3 *is not an answer.*

This approach has the advantage of separating the stage of generating the checks from the stage of actual checking. It may be helpful in cases when the IDB is in the core memory while the EDB is in secondary storage. Gains may be achieved from optimizing access to external memory. However, this is not the only way of testing for derivability. Recognizing that certain clauses may become derivable from EDB long before the full model is generated, the processes of model generation and checking can be integrated so that derivability is detected as soon as it occurs, even before the generation of the entire model.

In the following theorem we show how to integrate the checking for clause satisfiability into the model generation process.

Theorem 5 *Let DB be a ground Disjunctive Deductive Database (with no denial constraints) and* $Q = q_1 \vee ... \vee q_n$ *be a disjunctive query. If* $DB^d_Q = \{q_1, ..., q_n\} \cup \{C^d | C \in IDB\} \cup \{Head(C) \rightarrow \perp | C \in EDB\} = IDB^d_Q \cup \{Head(C) \rightarrow \perp | C \in EDB\}$. *Then: Q is derivable from DB if and only if* $\mathcal{MM}(DB^d_Q) = \emptyset$.

Proof: A clause corresponding to $M \in \mathcal{MM}(DB^d_Q)$ is derivable from DB if and only if it is subsumed by a clause C in EDB. M will be eliminated by the presence $Head(C) \rightarrow \perp$ in DB^d_Q. Since M is arbitrary, $\mathcal{MM}(DB^d_Q) = \emptyset$. The result follows immediately from Theorem 4[5]. ∎

Corollary 2 *Let DB be a ground Disjunctive Deductive Database (with no denial constraints) and* $Q = q_1 \vee ... \vee q_n$ *be a disjunctive query. Let* $\mathcal{MM}(DB^d_Q) \neq \emptyset$ *be the nonempty set of minimal models of* DB^d_Q. *Then:*

1. *Q is not derivable from DB.*
2. $DB_u \vdash Q$ *where* $DB_u = DB \cup \mathcal{C}$ *and* $\forall M \in \mathcal{MM}(DB^d_Q) \exists C \in \mathcal{C}$ *s.t. C subsumes M. That is,* DB_u *is achieved by adding to DB the set of clauses* $\mathcal{C}$ *subsuming all the minimal models of* IDB^d_Q.
3. $\mathcal{S} = \mathcal{MM}(DB^d_Q)$ *is the weakest such set that can be added to DB to guarantee the derivability of Q from the updated database* DB_u.

[5]Note that for the class of theories and queries considered here, the emptiness of minimal model set is equivalent to the existence of a refutation. In this case, the results extend naturally to other refutationally complete proof procedures. In the absence of a refutation, Corollary 2, however, utilizes the results returned by the model generation procedure for refined query answering.

Proof: 1. Immediate.

2. Needed to guarantee the condition of theorem 5.

3. Consider $M \in \mathcal{MM}(DB^d_Q)$. Clearly any clause C that is subsumed by M, $(M \subset C)$, will not remove M from the model set when its corresponding denial $(Head(C) \rightarrow \bot)$, is added to DB^d_Q. Any clause that subsumes M $(C \subset M)$ can be weakened by augmenting it with the remaining elements of M (elements of $M \setminus C$) and still guarantee the removal of M from the model set.

∎

Example 8 *For the database and queries of Example 7,*
$DB = \{C_1 = a \vee b, C_2 = c \vee d, C_3 = a \wedge d \rightarrow e \vee b, C_4 = b \wedge d \rightarrow f \vee c, C_5 = c \rightarrow g\}$.
Let $Q_1 = b \vee e \vee g$, $Q_2 = b \vee c \vee f$ *and* $Q_3 = g \vee f$.

$$DB^d = \{C^d_1 = a \wedge b \rightarrow \bot, C^d_2 = c \wedge d \rightarrow \bot, C^d_3 = e \wedge b \rightarrow a \vee d, C^d_4 = f \wedge c \rightarrow b \vee d, C^d_5 = g \rightarrow c\}.$$

$DB^d_{Q_1} = DB^d \cup Q^d_1 =$
$\{C^d_1 = a \wedge b \rightarrow \bot, C^d_2 = c \wedge d \rightarrow \bot, C^d_3 = e \wedge b \rightarrow a \vee d, C^d_4 = f \wedge c \rightarrow b \vee d, C^d_5 = g \rightarrow c, b, e, g\}$

$DB^d_{Q_2} = DB^d \cup Q^d_2 =$
$\{C^d_1 = a \wedge b \rightarrow \bot, C^d_2 = c \wedge d \rightarrow \bot, C^d_3 = e \wedge b \rightarrow a \vee d, C^d_4 = f \wedge c \rightarrow b \vee d, C^d_5 = g \rightarrow c, b, c, f\}$.

$DB^d_{Q_3} = DB^d \cup Q^d_3 =$
$\{C^d_1 = a \wedge b \rightarrow \bot, C^d_2 = c \wedge d \rightarrow \bot, C^d_3 = e \wedge b \rightarrow a \vee d, C^d_4 = f \wedge c \rightarrow b \vee d, C^d_5 = g \rightarrow c, g, f\}$.

$\mathcal{MM}(DB^d_{Q_1}) = \emptyset$. *$Q_1$ is an answer.*

$\mathcal{MM}(DB^d_{Q_2}) = \{\{b, c, f\}\}$. *$b \vee c \vee f$ is not derivable from EDB and therefore Q_2 is not an answer. It will be an answer if we add $b \vee c \vee f$ to DB.*

$\mathcal{MM}(DB^d_{Q_3}) = \{\{b, c, f, g\}\}$. *$Q_3$ is not an answer but adding $b \vee c \vee f \vee g$ will make Q_3 an answer.*

3.2 Conjunctive Queries

In the previous discussion we considered only ground disjunctive queries. The basic idea in treating a disjunctive query Q was to take its *dual*, Q^d (the set of atoms in Q) and apply the minimal model generation algorithm to Q^d together with the dual of the database itself. Atomic queries are a special case of disjunctive queries and no further treatment is needed. Conjunctive queries can be reduced to the set of their atomic components. Each of these atoms can be run alone and the query succeeds if all its atomic components are successful. This approach, however, may prove inappropriate for two reasons:

- There may be some duplication in the various runs of the procedure for the individual atoms. The goal clause sets of individual atoms need not be disjoint.

- This approach cannot be lifted automatically to the case of more complex ground queries and nonground queries when the separation of individual atoms may result in a different interpretation (and consequently different answers) from the intended one.

It turns out that the same approach developed for disjunctive queries is applicable to conjunctive queries as demonstrated by the following theorem:

Theorem 6 *Let DB be a ground Disjunctive Deductive Database (with no denial constraints) and $Q = q_1 \wedge ... \wedge q_n$ be a conjunctive query. Let $IDB_Q^d = Q^d \cup IDB^d = \{q_1 \vee ... \vee q_n\} \cup \{C^d | C \in IDB\}$. Let $\mathcal{MM}(IDB_Q^d)$ be the set of minimal models of IDB_Q^d. Then: Q is derivable from DB if and only if EDB derives clause M for all $M \in \mathcal{MM}(IDB_Q^d)$.*

Proof: For $Q = q_1 \wedge ... \wedge q_n$ to succeed, $q_1, ..., q_n$ must all succeed simultaneously. By Theorem 4 this happens if and only if EDB derives clause M for all $M \in \mathcal{MM}(IDB_{q_i}^d)$ for all $i = 1, ..., n$.

To prove the theorem we need only to show that the elements of $\mathcal{MM}(IDB_Q^d)$ subsume all the elements of $\cup_{i=1}^{i=n} \mathcal{MM}(IDB_{q_i}^d)$. That is for any $M \in \cup_{i=1}^{i=n} \mathcal{MM}(IDB_{q_i}^d)$ there exists $M' \in \mathcal{MM}(IDB_Q^d)$ such that $M' \subseteq M$.

This is the case since $\mathcal{MM}(IDB_Q^d) = Min(\cup_{i=1}^{i=n} \mathcal{MM}(IDB_{q_i}^d))$, where $Min(S)$ returns only the minimal elements of a set of models S. Clearly, any minimal model of $IDB_{q_i}^d$ is also a (not necessarily minimal) model of IDB_Q^d since it satisfies IDB^d which is common to both as well as Q by having its q_i as an element. ∎

The following is an immediate result:

Corollary 3 *Let DB be a ground Disjunctive Deductive Database (with no denial constraints) and $Q = q_1 \wedge ... \wedge q_n$ be a conjunctive query. If $DB_Q^d = Q^d \cup \{C^d | C \in IDB\} \cup \{Head(C) \rightarrow \perp | C \in EDB\}$ Then:*

1. *Q is derivable from DB if and only if $\mathcal{MM}(DB_Q^d) = \emptyset$.*
2. *If $\mathcal{MM}(DB_Q^d)$ is nonempty then:*
   (a) *Q is not derivable from DB.*
   (b) *Q becomes derivable from the updated database DB_u achieved by adding to DB the set of clauses $\mathcal{S}$ such that $\forall M \in \mathcal{MM}(DB_Q^d) \exists C \in \mathcal{S}$ s.t. C subsumes M.*
   (c) *$\mathcal{S} = \mathcal{MM}(DB_Q^d)$ is the weakest such set that can be added to DB to guarantee the derivability of Q from the updated database DB_u.*

Example 9 *Let $DB = \{C_1 = a \vee b, C_2 = c \vee d, C_3 = a \wedge d \rightarrow e \vee b, C_4 = b \wedge d \rightarrow f \vee c, C_5 = c \rightarrow g, C_{21} = d \rightarrow h, C_{22} = c \rightarrow i, C_{23} = e \rightarrow i, C_{24} = f \rightarrow i, C_{25} = g \rightarrow h\}$. Let $Q_4 = h \wedge i$, $Q_5 = h \wedge e$.*

$IDB^d = \{C_3^d = e \wedge b \rightarrow a \vee d, C_4^d = f \wedge c \rightarrow b \vee d, C_5^d = g \rightarrow c, C_{21}^d = h \rightarrow d, C_{22}^d = i \rightarrow c, C_{23}^d = i \rightarrow e, C_{24}^d = i \rightarrow f, C_{25}^d = h \rightarrow g\}$.

$IDB^d_{Q_4} = IDB^d \cup Q^d_4 = \{C^d_3 = e \wedge b \rightarrow a \vee d, C^d_4 = f \wedge c \rightarrow b \vee d, C^d_5 = g \rightarrow c, C^d_{21} = h \rightarrow d, C^d_{22} = i \rightarrow c, C^d_{23} = i \rightarrow e, C^d_{24} = i \rightarrow f, C^d_{25} = h \rightarrow g, h \vee i\}$.

$IDB^d_{Q_5} = IDB^d \cup Q^d_2 = \{C^d_3 = e \wedge b \rightarrow a \vee d, C^d_4 = f \wedge c \rightarrow b \vee d, C^d_5 = g \rightarrow c, C^d_{21} = h \rightarrow d, C^d_{22} = i \rightarrow c, C^d_{23} = i \rightarrow e, C^d_{24} = i \rightarrow f, C^d_{25} = h \rightarrow g, h \vee e\}$.

$\mathcal{MM}(IDB^d_{Q_4}) = \{\{c, d, g, h\}, \{a, b, c, e, f, i\}, \{c, d, e, f, i\}\}$. *All of these clauses are subsumed by clauses in EDB.* Q_4 *is an answer.*

$\mathcal{MM}(IDB^d_{Q_5}) = \{\{c, d, g, h\}, \{e\}\}$. *Clause* e *is not derivable from EDB and therefore* Q_5 *is not an answer.*

On the other hand: $\mathcal{MM}(IDB^d_{Q_4} \cup \{C^d_1, C^d_2\}) = \emptyset$ *since* Q_4 *is an answer while* $\mathcal{MM}(IDB^d_{Q_5} \cup \{C^d_1, C^d_2\}) = \{e\}$ *since* Q_5 *is not an answer. Adding* e *to DB will make* Q_5 *an answer.*

3.3 Compound Queries

So far we dealt with conjunctive and disjunctive queries on the assumption that they constitute the major type of queries encountered in applications. The results of this section can be extended to a more general class of queries we call positive queries: those that can be translated into a set of ground positive clauses (a conjunction of a set of disjunction queries). Clearly an answer to such as query is affirmative if every clause has a *yes* answer and is negative otherwise.

To answer such a query one could run each clause separately as a disjunctive query and combine the results. Alternatively, one may run a single process with the elements of each clause being the starting set for individual branches. The resulting set of minimal models, which may be more compact than the union of minimal model sets of individual clauses, will represent the clauses that need to be true in EDB for the compound query to have a *yes* answer. The compactness is the result of exploiting the shared pieces of information between the processes corresponding to individual disjunctive components of the query. Consider the following example:

Example 10 *Let* $DB = \{C_1 = a \vee b, C_2 = c \vee d, C_3 = a \wedge d \rightarrow e \vee b, C_4 = b \wedge d \rightarrow f \vee c, C_5 = c \rightarrow g\}$. *Let* $Q_6 = a \vee g \vee f$ *and* $Q_7 = a \vee c \vee f$.
$\mathcal{MM}(IDB^d_{Q_6}) = \{\{a, b, c, f, g\}, \{a, c, d, f, g\}\}$.
$\mathcal{MM}(IDB^d_{Q_7}) = \{\{a, b, c, f\}, \{a, c, d, f\}\}$.
$(Q_6 \wedge Q_7)^d = \{a, f, g \vee c\}$.
$\mathcal{MM}(IDB^d_{Q_6 \wedge Q_7}) = \mathcal{MM}(IDB^d_{Q_7}) = \{\{a, b, c, f\}, \{a, c, d, f\}\}$.

However, if we run the procedure in refutation mode (Theorem 5) then it may be more advantageous to operate each process separately since the failure of one of the disjunctive queries will indicate a failure of the entire query. Refined query answering where we can augment the theory with the constraints needed to make the query derivable is still applicable as in the case of disjunctive and conjunctive queries.

4 Removing Restrictions

In our discussion so far two major restrictions were imposed on the type of theory being treated. The first is the requirement that the query and the database be ground and the second is that the database has no denial rules (clauses with empty heads). Some of these restrictions simplified the discussion while others are common in procedures dealing with DDDB. However, we would like our approach extended beyond these restrictions.

4.1 Denial Constraints

Denial constraints are rules with empty heads of the form $C = Body(C) \rightarrow \perp$. In [5] we prove the following result:

Lemma 2 *Let $\mathcal{S}$ be a set of clauses and $A_1, ..., A_n (n \geq 1)$ be atoms.*

1. *If M is a minimal Herbrand model of $\mathcal{S}$ such that $M \not\models A_1 \wedge ... \wedge A_n$, then M is a minimal Herbrand model of $\mathcal{S} \cup \{A_1 \wedge ... \wedge A_n \rightarrow \perp\}$.*
2. *If M is a minimal Herbrand model of $\mathcal{S} \cup \{A_1 \wedge ... \wedge A_n \rightarrow \perp\}$, then M is also a minimal Herbrand model of $\mathcal{S}$.*

Lemma 2 shows that adding denial rules to the theory contribute to the minimal model structure of the database only by removing those minimal models which satisfy the body of a denial rule. That is, no minimal models of the database DB are extended nor new minimal models are created for the new theory $DB \cup \mathcal{C}$, where $\mathcal{C}$ is a set of denial rules and additionally, $\mathcal{MM}(DB \cup \mathcal{C}) \subseteq \mathcal{MM}(DB)$.

Clearly, if a positive query Q is derivable from DB then it is also derivable from $DB \cup \mathcal{C}$. However, it is possible that Q is not derivable from DB but is derivable from $DB \cup \mathcal{C}$ due to the fact that the rules of $\mathcal{C}$ remove all the minimal models of the set $\mathcal{MM}(DB) \setminus \mathcal{MM}(DB \cup \mathcal{C})$. This was demonstrated by Example 4. So, in a sense, the presence of denial rules must enhance the potential derivability from the database (for positive queries). The form this enhancement can take is to expand the clauses in the goal clause set of the query so as to contain more atoms.

Indeed, that is what happens. Formally we have the following result:

Theorem 7 *Let DB be a ground Disjunctive Deductive Database. Let $\mathcal{C}$ be the set of denial rules in DB and Q be a positive query. Then: $DB \vdash Q$ if and only if the formula $(Body(C) \vee Q)$ is derivable from DB for some $C \in \mathcal{C}$. Or equivalently, if and only if $\{(B \vee Q) | B \in Body(C)\}$ is derivable from DB.*

Proof: If $DB \vdash (Body(C) \vee Q)$ for some $C \in \mathcal{C}$ then the minimal models of DB in which Q is not satisfied will have to satisfy $Body(C)$ and will be nonmodels of DB by an application of C making $DB \vdash Q$.

Assume $DB \vdash Q$. Then by Lemma 2, for any model $M \in \mathcal{MM}(DB \setminus \mathcal{C})$ and $M \notin \mathcal{MM}(DB)$ there must exist a clause $C \in \mathcal{C}$ such that $Body(C) \subseteq M$. Clearly, $M \models (Body(C) \vee Q)$. ∎

This theorem is basically suggesting that denial rules have the effect of expanding the goal clause set of the query unconditionally by extending the query Q with atoms from $Body(C)$, one at a time. Since denial rules of DB will convert into positive clauses in the dual database DB^d according to our definitions, it is clear that these rules can be treated on the same footing as other rules of the theory. It is straightforward to extend all the results established so far to the case of databases containing denial rules. The result is summarized in the following theorem[6]:

Theorem 8 *Let DB be a ground Disjunctive Deductive Database with the set of denial rules $\mathcal{C}$: $DB = EDB \cup IDB \cup \mathcal{C}$ and Q be a positive query.*

- *Let $EDB^d = \{C^d | C \in EDB\}$.*
- *Let $\mathcal{C}^d = \{C^d | C \in \mathcal{C}\}$.*
- *Let $IDB^d = \{C^d | C \in IDB\}$.*
- *Let $IDB^d_Q = Q^d \cup IDB^d = Q^d \cup \{C^d | C \in IDB\}$.*
- *Let $DB^d = EDB^d \cup IDB^d \cup \mathcal{C}^d$.*
- *Let $DB^d_Q = Q^d \cup EDB^d \cup IDB^d \cup \mathcal{C}^d$.*

Then

1. *$DB \vdash Q$ if and only if EDB derives clause M for all $M \in \mathcal{MM}(\mathcal{C}^d \cup IDB^d_Q)$. That is, $EDB \vdash C_M = A_1 \vee ... \vee A_l$ if $M = \{A_1, ..., A_l\}$.*
2. *$DB \vdash Q$ if and only if $\mathcal{MM}(DB^d_Q) = \emptyset$.*
3. *If $\mathcal{MM}(DB^d_Q)$ is nonempty then:*
   (a) *Q is not derivable from DB.*
   (b) *Q becomes derivable from the updated database DB_u achieved by adding to DB the set of clauses $\mathcal{S}$ such that $\forall M \in \mathcal{MM}(DB^d_Q) \exists C \in \mathcal{S}$ s.t. C subsumes M.*
   (c) *$\mathcal{S} = \mathcal{MM}(DB^d_Q)$ is the weakest such set that can be added to DB to guarantee the derivability of Q from the updated database.*

Proof: Along the lines of earlier proofs. ■

Consider the following example:

[6] The result is quite natural. The empty head $\perp$ of the denial clause trivially subsumes every positive clause and therefore the goal set of the query can be expanded using such a clause, unconditionally. As expected, in the transformed theory such clauses are converted into facts where they can be used for this unconditional expansion.

Example 11 *Let* $DB =$
$\{C_1 = a \vee b, C_2 = c \vee d, C_3 = a \wedge d \rightarrow e \vee b, C_4 = b \wedge d \rightarrow f \vee c, C_5 = c \rightarrow g, C_6 = a \wedge d \rightarrow \bot\}$. *Let* $Q_1 = b \vee e \vee g$, $Q_2 = b \vee c \vee f$ *and* $Q_3 = g \vee f$. $\mathcal{C} = \{C_6 = a \wedge d \rightarrow \bot\}$.

$IDB^d \cup \mathcal{C}^d = \{C_3^d = e \wedge b \rightarrow a \vee d, C_4^d = f \wedge c \rightarrow b \vee d, C_5^d = g \rightarrow c, C_6^d = \top \rightarrow a \vee d\}$.

$IDB^d_{Q_1} \cup \mathcal{C}^d = IDB^d \cup Q_1^d \cup \mathcal{C}^d =$
$\{C_3^d = e \wedge b \rightarrow a \vee d, C_4^d = f \wedge c \rightarrow b \vee d, C_5^d = g \rightarrow c, C_6^d = \top \rightarrow a \vee d, b, e, g\}$

$IDB^d_{Q_2} \cup \mathcal{C}^d = IDB^d \cup Q_2^d \cup \mathcal{C}^d =$
$\{C_3^d = e \wedge b \rightarrow a \vee d, C_4^d = f \wedge c \rightarrow b \vee d, C_5^d = g \rightarrow c, C_6^d = \top \rightarrow a \vee d, b, c, f\}$.

$IDB^d_{Q_3} \cup \mathcal{C}^d = IDB^d \cup Q_3^d \cup \mathcal{C}^d =$
$\{C_3^d = e \wedge b \rightarrow a \vee d, C_4^d = f \wedge c \rightarrow b \vee d, C_5^d = g \rightarrow c, C_6^d = \top \rightarrow a \vee d, g, f\}$.

$\mathcal{MM}(IDB^d_{Q_1} \cup \mathcal{C}^d) = \{\{a, b, c, e, g\}, \{b, c, d, e, g\}\}$.
Both $a \vee b \vee c \vee e \vee g$ *and* $b \vee c \vee d \vee e \vee g$ *are subsumed by clauses in EDB.* Q_1 *is an answer and is not affected by the added constraint* C_6.

$\mathcal{MM}(IDB^d_{Q_2} \cup \mathcal{C}^d) = \{\{a, b, c, f\}, \{b, c, f, d\}\}$. *Both clauses are derivable from EDB and therefore* Q_2 *is an answer as a result of adding* C_6.

$\mathcal{MM}(IDB^d_{Q_3} \cup \mathcal{C}^d) = \{\{a, b, c, f, g\}, \{c, d, f, g\}\}$.
$c \vee d \vee f \vee g$ *is subsumed by* $(c \vee d) \in EDB$ *and* $a \vee b \vee c \vee f \vee g$ *is subsumed by* $(a \vee b) \in EDB$ *and so* Q_3 *is an answer.*
Note that $\mathcal{MM}(DB^d_{Q_1}) = \mathcal{MM}(DB^d_{Q_2}) = \mathcal{MM}(DB^d_{Q_3}) = \emptyset$. *Comparing the results with Example 8 shows how adding the constraints contributed to deriving more yes answers.*

4.2 Nonground Terms

We considered only ground databases and queries. While one may argue that a DDDB can always be grounded so that the results are applicable, grounding can result in huge databases. So it is always better to be able to deal with DDDBs with variables and perform instantiations only when necessary. In this regard we consider two issues: nonground queries and nonground rules.

4.2.1 Nonground Queries

For definite databases only atomic answers are allowed and possible, DDDBs allow the derivation of indefinite answers to queries. While one copy of the query is needed in the derivation of an answer to an atomic query in a definite database, more copies may be necessary to derive an indefinite answer. This stems from the definition of an answer to a query in a DDDB. $a_1 + a_2 + ... + a_n$ is an answer to $Q(x)$ iff $Q(a_1) \vee ... \vee Q(a_1)$ is derivable from DB.

So in our procedure, when failing to derive the empty clause with a single ground substitution for the variables in the query attempts should be taken with additional instantiations. The additions will in general help the refutation process by generating longer clauses (models) that are to be subsumed by the elements of EDB (closed by clauses the transformed EDB).

In the case of disjunctive queries the issue of insuring the compatibility of substitutions with the original query must be accounted for. But since all the elements (atoms) of the query are added to the same branch, they can be instantiated simultaneously. In case additional variables remain they have to inherit the naming so that compatibility of subsequent substitutions can be guaranteed. For conjunctive queries elements (atoms) of the transformed query (Q^d) constitute a clause and therefore the occurrences of a variable are instantiated simultaneously.

In a sense variables in transformed (dualized) queries are treated as universally quantified. They can have multiple instantiations in the same branch resulting in longer clauses to be checked against the EDB. Of course such multiple instantiations will result in disjunctive answers to the query which is in line with treating elements of a single branch as disjunctions of the corresponding atoms.

Example 12 *Let* $DB_1 = \{C_1 = S \vee R, C_2 = R \rightarrow P(a), C_3 = S \rightarrow P(b)\}$.
Let $DB_2 = \{C_4 = T \vee U, C_5 = T \rightarrow P(c) \vee P(d), C_6 = U \rightarrow P(d) \vee P(e)\}$.

Let $Q = P(x)$. $IDB_1^d = \{C_2^d = P(a) \rightarrow R, C_3^d = P(b) \rightarrow S\}$. *The only minimal model of* ${IDB_1}_Q^d$ *derivable from* EDB_1 *is* $\{P(a), R, P(b), S\}$ *which includes two instances of the query.* $a + b$ *is an answer to* Q *in* DB_1.

On the other hand $IDB_2^d = \{C_5^d = P(c) \wedge P(d) \rightarrow T, C_6^d = P(d) \wedge P(e) \rightarrow U\}$. *The only minimal model of* ${IDB_2}_Q^d$ *derivable from* EDB_2 *is* $\{P(c), P(d), T, P(e), U\}$ *which includes three instances of the query.* $c + d + e$ *is an answer to* Q *in* DB_2.

Both answers generated are minimal. Note, however, that combining the two databases in different orders will generate both minimal and nonminimal answers even when derivability of clauses is detected as soon as it appears. For example for $DB = \{C_1, C_4, C_5, C_2, C_3, C_6\}$ *will generate the nonminimal answer* $c + d + a + b$.

4.2.2 Nonground Rules

While we described our procedure relative to a ground DDDB, the extension to the case of nonground databases is possible if the procedure for minimal model computations can find all the minimal models of the theory starting from the set of facts. The procedure we use to implement our approach is complete and sound for the class of theories that are range restricted and have only finite minimal models. For non-range restricted theories our procedure needs to make them so by a simple transformation. This transformation is a substitute for, and can generally be more efficient than, blind instantiation [5].

In our approach, given a DDDB, DB, the minimal model generating procedure is applied to the dual DB^d. As far as finiteness of minimal models this property is always guaranteed in the absence of function symbols and under the finiteness of underlying domains of the theory in the case of DDDBs. As for range restrictedness, even if the original database is range restricted, the transformed (dual) one need not be so. This happens when a variable in the body of a rule doesn't occur in the head of that same rule. In this case we'll need to apply the range restriction transformation to the dualized theory to make our algorithm applicable [5]. Of course there may be instances when the original theory is not range restricted and the dual is. No action is needed in this instance. In all cases we need to make sure that the range restriction transformation is not taken into account during the dualization process but only after the dualization is completed.

One point to emphasize here is that a multiple use of a nonrange restricted dual clause will result in the variables in the body being treated as existentially quantified. Each copy produces a possible candidate for an answer to the query and the number of such copies used is a reflection of the indefiniteness of the answer. To be able to perform testing after model generation, variables resulting from different application of the same rule may be given different place-holding names (say subscripted variables) that can be matched against elements of the EDB. Different occurrences of the same variable in different branches need to maintain the same naming to ensure compatible future substitutions. Of course such an approach tends to obscure the concept of model minimality which is defined in terms of ground atoms. Range restrictedness of the dual database implies that whenever the query is used to ground the body of a dual clause, the head of that clause is also grounded. This removes the issue of maintaining substitution compatibility in different model tree branches. Of course if the query itself is nonground then range restrictedness will be less helpful. Issues of answer minimality are raised by the brute force instantiation. Consider the following example:

Example 13 *[14, 12]*

- *Let* $DB_1 = \{C_1 = R(a) \vee S(b), C_2 = S(c), C_3 = S(d) \vee P(e),$ $C_4 = R(x) \rightarrow P(x), C_5 = S(x) \rightarrow R(x)\}$ *and let* $Q_1 = P(a) \vee P(b), Q_2 = P(d), Q_3 = P(x)$.

  $IDB_1^d = \{C_4^d = P(x) \rightarrow R(x), C_5^d = R(x) \rightarrow S(x)\}$.

  The only minimal model of ${IDB_1}^d_{Q_1}$ *is* $\{P(a), P(b), R(a), R(b), S(a), S(b)\}$ *which is derivable from EDB.* Q_1 *is a yes answer.*

  The only minimal model of ${IDB_1}^d_{Q_2}$ *is* $\{P(d), R(d), S(d)\}$ *which is not derivable from EDB.* Q_2 *is a no answer.*

  The only minimal model of ${IDB_1}^d_{Q_3}$ *is one with any number of instances of* $\{P(x), R(x), S(x)\}$. *The derivable components return the answers. It is easy to see that the answers to* Q_3 *are* $a{+}b$, c *and* $d{+}e$ *corresponding to the instances:* $\{P(a), P(b), R(a), R(b), S(a), S(b)\}$ $\{P(c), R(c), S(c)\}$ *and* $\{P(d), P(e), R(d), R(e), S(d), S(e)\}$.

  Note that DB_1^d *is range restricted and that not being able to detect derivability as soon as it occurs or using a different clause ordering may result in generating nonminimal answers to nonground queries. Splitting didn't occur here since the dual database is Horn. In case of splitting on nonground disjunctions substitutions compatibility needs to be guaranteed.*

- *Let* $DB_2 = \{C_1 = Person(x) \wedge Cold(x) \rightarrow Sneeze(x), C_2 = Person(x) \wedge HayFever(x) \rightarrow Sneeze(x), C_3 = \top \rightarrow Person(Tom), C_4 = Person(x) \wedge Cold(x) \wedge R(x) HayFever(x) \rightarrow \bot\}$.

  Let $Q = Sneeze(Tom)$. $Q^d = Sneeze(Tom)$.

  $DB_2^d = \{C_1^d = Sneeze(x) \rightarrow Person(x) \vee Cold(x), C_2^d = Sneeze(x) \rightarrow Person(x) \vee HayFever(x),$ $C_4^d = \top \rightarrow Person(x) \vee Cold(x) \vee HayFever(x), C_3^d = Person(Tom) \rightarrow \bot\}$.

  The only minimal model of ${DB_2}^d_Q$ *is* $\{Sneeze(Tom), Cold(Tom), Hayever(Tom)\}$. Q *is not a yes answer and* $Sneeze(Tom) \vee Cold(Tom) \vee HayFever(Tom)$ *is the possible update. Note that* $\{Cold(Tom) \vee HayFever(Tom)\}$ *is the nontrivial update.*

Since DB_2^d is not range restricted, having more constants in the Herbrand base would have generated longer (and more) minimal models. Once one knows that the branch will not close the decision on when to stop the expansion may be influenced by the type of update desired.

5 Interpretation and Implementation Issues

In this section we briefly discuss the reasoning behind the duality approach adopted in this paper to achieve top-down reasoning using a bottom-up procedure and argue that working with the transformed theory is as *natural* as working with the original clause set. We also address some implementation issues.

5.1 Interpretations of the Duality Approach

We give some thoughts to explain what is happening in the transition from the theory to its dual as the input to the model generation procedure. We show that this process can be viewed as a switch to reasoning in a theory with reversed polarities of literals in which the dual clauses are used to propagate a truth value different from that propagated by the clauses of the original theory. The hope is to get some insight into the change in efficiency resulting from the application of the proof procedure to DB^d and Q^d instead of to DB and $\neg Q$.

5.1.1 Reversed Polarities of Clauses

Usually, proving the query is done by trying to refute the theory augmented by the negation of the query. When all clauses are represented as disjunctions of literals, the dual transformation has the effect of consistently reversing the polarity of each literal of both the theory and the negation of the query. Positive literals are changed into their negative counterparts and vise versa. This is so since $C = Head(C) \vee \neg Body(C)$ while $C^d = \neg Head(C) \vee Body(C)$[7]. The same reasoning applies to the relationship between $\neg Q$ and Q^d. Clearly, this syntactic transformation preserves the consistency properties. So, $DB \cup \{\neg Q\} \vdash \Box$ if and only if $DB^d \cup \{Q^d\} \vdash \Box$. The change in efficiency in the transition from DB to DB^d can be attributed to the fact that the bottom-up computational procedure used in both cases (e.g. model generator) tends to treat positive and negative literals asymmetrically. For example, model generation provers are generally driven by positive facts that are then used to generate new facts through theory clauses. Negative clauses are only used to close branches when applicable. Therefore, working with the transformed theory DB^d can affect the performance of the algorithm by reducing the number of positive literal occurrences and thus limiting the number of possible expansions at the expense of increasing the number of negative literal occurrences which can speed up branch closure. The overall effect may be faster refutations under favorable circumstances.

Viewed as disjunctions of literals, the clauses of DB^d specify which atoms need to be *false* while those of DB specify which atoms are to be *true* in order for the clauses of the theory DB to hold. For example, while $C = \top \rightarrow b \vee c = b \vee c$ says that either b, c or both are to be *true* for C to hold,

[7] Recall that the heads and bodies of clauses are treated as sets of atoms. The negations are, therefore, disjunctions of negative literals.

$C^d = b \wedge c \rightarrow \perp = \neg b \vee \neg c$ says that either $\neg b$, $\neg c$ or both must be *false* to satisfy C, two equivalent statements. This property is preserved by resolution between clauses of DB^d: the resolvent of two clauses of DB^d specifies what must be *false* in the same way as the resolving clauses do. Consider the following example:

Example 14 *Let* $DB = \{C_1 = b \vee c, C_2 = b \rightarrow a \vee e, C_3 = c \rightarrow a \vee d, C_4 = b \wedge e \rightarrow \perp\}$. *Let* $Q = a \vee d$.

$DB^d = \{C_1^d = b \wedge c \rightarrow \perp, C_2^d = a \wedge e \rightarrow b, C_3^d = a \wedge d \rightarrow c, C_4^d = \top \rightarrow b \vee e\}$.

The clausal representation of $DB \cup \{Q \rightarrow \perp\}$ *and* $DB^d \cup Q^d$ *are*
$\{C_1 = b \vee c, C_2 = \neg b \vee a \vee e, C_3 = \neg c \vee a \vee d, C_4 = \neg b \vee \neg e\} \cup \{\neg a, \neg d\}$ *and*
$\{C_1^d = \neg b \vee \neg c, C_2^d = b \vee \neg a \vee \neg e, C_3^d = c \vee \neg a \vee \neg d, C_4^d = b \vee e\} \cup \{a, d\}$, *respectively.*

The correspondence between the two sets and the reversal of polarities is clear.

$Resolvent(C_1, C_2) = a \vee c \vee e$ *and* $Resolvent(C_1^d, C_2^d) = \neg a \vee \neg c \vee \neg e$.

Note that any resolution of two clauses in DB *can be simulated by a resolution of the corresponding clauses in* DB^d *with the resulting resolvents having reversed polarities of their literals.*

5.1.2 Propagating Different Truth Values

Another way to look at the dual transformation as outlined so far is to interpret it in terms of using the theory rules to (backward) propagate the *falsity* of the head atoms to the body atoms of each clause. This is in contrast to the (more usual forward) propagation of the *truth* of the body atoms to the head atoms when the bottom-up model generation approach is applied to the original clauses. The *falsity* here can be interpreted as the nonderivability of the respective atom/formula through the given rule.

Usually, the derivability of a (positive) query Q from a theory DB is proved by showing that the model tree of $\{DB, \neg Q\}$ has no open branches (models). When a model generation procedure is applied to $\{DB, \neg Q\}$ one starts from the *true* atoms of the theory (elements of the EDB, facts, positive clauses) and uses the rules (elements of IDB) to discover the new atoms that can be assigned *true* (from heads of clauses with bodies satisfied in the current branch) in the hope that some of them will close branches containing literals of the negation of the query (or constraints, original or introduced e.g. if complement splitting is employed). A branch of the tree is *closed* by having it contain two complementary literals[8]. The proof procedure expands individual branches of the tree by adding new atoms until no additions are possible or the branch closes [5]. The aim is to show that to assume that the query is *false* is inconsistent with the theory and therefore Q must be derivable from DB. If one of the branches cannot be closed then the query is not derivable from the theory ($DB \not\vdash Q$) and the atoms of an open branch represent a counter model in which both DB and $\neg Q$ are satisfied.

Clearly, changing the polarity of every literal in a branch of the tree will not change the property of the branch as being closed or open. Dualizing followed by applying the model generation procedure to the transformed theory and query ($DB^d \cup Q^d$) can be viewed as the process of consistently reversing the polarity of atoms in the model tree. However, under the duality approach we begin by applying the model generating procedure to the dual of the query, Q^d, which represents the negation of the

[8] Technically, closure happens when $\perp$ is added to the branch which has all the body atoms of a clause with an empty ($\perp$) head (representing a purely negative clause).

#	Query/Clause	Dual
1	$Q = q_1 \vee ... \vee q_n$ $\neg Q = \{\neg q_1, ..., \neg q_n\}$ All q_is are *false* if Q is *false*	$Q^d = \{q_1, ..., q_n\}$ Atom i of Q^d is *true* when atom i of $\neg Q$ is *false*
2	$Q = q_1 \wedge ... \wedge q_n$ $\neg Q = \neg q_1 \vee ... \vee \neg q_n$ Some q_is are *false* if Q is *false*	$Q^d = q_1 \vee ... \vee q_n$ Atom i of Q^d is *true* when atom i of $\neg Q$ is *false*
3	$C = B_1 \wedge ... \wedge B_k \rightarrow A_1 \vee ... \vee A_l$ If all B_is are *true* then some of the A_js are *true*	$C^d = A_1 \wedge ... \wedge A_l \rightarrow B_1 \vee ... \vee B_k$ If all A_js are *false* then some of the B_is are *false*
4	$C = \top \rightarrow A_1 \vee ... \vee A_l$ $C = A_1 \vee ... \vee A_l$ Some of the A_is are *true*	$C^d = A_1 \wedge ... \wedge A_l \rightarrow \bot$ $C^d = \neg A_1 \vee ... \vee \neg A_l$ Some of the A_is are *false*
5	$C = B_1 \wedge ... \wedge B_k \rightarrow \bot$ $C = \neg B_1 \vee ... \vee \neg B_k$ Some of the B_i are *false*	$C^d = \top \rightarrow B_1 \vee ... \vee B_k$ $C^d = B_1 \vee ... \vee B_k$ Some of the B_is are *true*

Table 1: The Dual Transformation and its Effects.

query with reversed (positive) polarities. This is equivalent to specifying the atoms that need to be *false* in order for Q to be *false* (nonderivable). Next we have the bottom-up procedure operate with the transformed rules to derive the other relevant atoms that must be *false* as a result of assuming the falsity of the query.

The search is initiated by the query elements and only relevant clauses of the transformed theory are invoked. Branches are extended by elements the *falsity* (nonderivability) of which is determined from the current elements of the branch through the propagation of falsity from heads to bodies of the original clauses (or from the bodies to the heads of the dual clauses). During this expansion process a branch may close or may remain open. Contradiction (and closing a branch) is reached when certain atoms are required to be *false* (nonderivable) in order for the query to be *false* while they are *true* in (derivable from) the theory (e.g. elements of the EDB). An open branch M corresponds to the disjunction of atoms that needs to be *false* in (not derivable from) the theory DB (by having all its atoms *false* or not derivable) so that $DB \not\vdash Q$. To close such a branch there must exist a clause of DB that consists entirely of elements of M. The dualization takes care of this by making the dual of a DB-fact (headless clause) derive the empty clause when all its atoms are present in the branch (it has $\bot$ in the head, see entry 4 of Table 1). The full details of how dualization enables a model generation procedure to interpret clauses in terms of propagating the *falsity* (nonderivability) of the head to the body rather than the *truth* of the body to the head of the clause are summarized in Table 1. Clearly, the head of a (ground) clause C is *false* only when all atoms of $Head(C)$ are *false*. For that to happen and still have C satisfied it must be the case that at least one of the atoms of $Body(C)$ is *false* (entry 3). A positive clause is *false* when all of its atoms are *false*. Otherwise the clause is satisfied (entry 4). For a negative clause, one can derive *false* only if all its atoms are *true*. Otherwise, if one of the atoms is *false* then the clause is satisfied (entry 5). When a disjunction of atoms is to be *false* it must be the case that none of its subdisjunctions is *true*. That is, all clauses with heads subsuming the disjunction must have *false* heads. This falsity has to be propagated to the bodies to maintain that such clauses hold in all states of the theory (entry 3). Note, in particular, that the roles of facts and denials are interchanged (entry 5). The relationship between the negation of the query and its dual in terms of reversed polarities is reflected in entries 1 and 2 of Table 1.

With the reversal of polarities accounted for, the correspondence between the tree structure resulting from the application of the model generation procedure to $DB^d \cup Q^d$ and general model trees is clear. An open branch of the tree represents the set of atoms that must be *false* for the query to be *false*. Each such branch represents a way in which the *falsity* (nonderivability) of the query can be proved. Only if all possible ways fail (the model tree has no open branches) then the query is shown to be *true*. A closed branch represents a failing attempt to account for the nonderivability of the query: the failure results from requirement that a single atom be assigned both *true* and *false*. A cut in the tree (the disjunction of an element from each open branch) specifies the falsity of which atoms must be ensured to guarantee the falsity of the query. By the completeness of the model generating procedure, the open branches of a saturated tree (one in which no further expansions are possible) is the counterpart of the model tree and specifies the branches that need to be closed to make the query derivable as suggested by Theorem 8. Clearly, it is sufficient to close only branches with minimal sets of atoms and other branches will close automatically.

5.2 Implementation Issues

The discussion so far points to a simple algorithm for answering a positive query Q against a disjunctive deductive database DB. Two steps are involved:

- The first is the transformation of DB and $\neg Q$ into its dual database DB^d and Q^d. This is done by reversing the direction of the implication signs in all clauses. Since Heads and Bodies of clauses are treated as sets of atoms, the change of the logical connective is sort of a side effect to this transformation. The transformation is applicable to clauses with empty heads and bodies as well.

- The second step is to apply a sound and complete minimal model generating procedure to the dual database augmented by the dual of the query being answered. That is, to the set $DB^d_Q = (DB^d \cup \{Q^d\})$. The set of minimal models of DB^d_Q carries the answer to the derivability of Q from DB as well as the conditions under which this derivability is possible/not possible.

In [5] we have developed an efficient, sound and complete minimal model generation procedure called MM-Satchmo. One way to accommodate MM-Satchmo to the present task is rewriting the procedure to account for the dualization and the changed roles of the query and the EDB to the initial partial interpretation and the set of constraints, respectively. Another is to write a small program (module) to convert the input theory into the required form (perform the dualization of DB and Q). One may even elect to modify the procedure so that the roles of $\vee$ and $\wedge$ can be exchanged and the direction of the implication symbol is interpreted in reverse depending on input parameters. For the testing of our results we used the plain version of our minimal model generator and supplied it with the dualized directly. The dualization was done manually.

Another degree of freedom in our procedure is the possibility of separating the *check generation* process from that of the *actual checking*. The first is performed on the generally static IDB and the query. The second is done against the more dynamic EDB. The separation of the two steps, which could be viewed as operating the procedure in a model generation mode, could be of value to minimize the number of accesses to the EDB component. It is also possible to *compile* the query with the IDB so as not to recompute (regenerate checks) each time the same query is posed [10]. Integrating the two steps on the other hand, which can be viewed as running the procedure in a refutation mode makes it possible to abandon models before they are fully generated. This may improve the efficiency if the cost of accessing the EDB component is not very large. The surviving models in the latter case are helpful in defining updates sufficient to make the query derivable from the database.

The advanced procedure is based on minimal model generation for a simple (syntactical) modification of the input database. The model generation process is driven by the query itself while the elements of the extensional component of the database serve as constraints on the generated models working only to reduce their number . Therefore, one would expect that the number of models having a query as basis will generally be not very large. Clauses that are not relevant to the query do not participate in the model generation process. The exact size of the model tree, however, depends on the nature of the database and the interconnections between its clauses.

Working with the dual structure on the other hand will most likely involve having many disjunctive elements representing the bodies of original clauses. This will tend to increase the size of the model structure by generating a large number of models. For example, such a situation will arise when one has a definite database with rules that have large bodies. The dual database will be heavily disjunctive resulting in a large search space for our procedure. This may contrast strongly with the single model of the original definite database.

An additional efficiency consideration is that the database resulting from the transformation may not be range restricted while the original theory is. This will mandate resorting to additional transformations to achieve range restrictedness or to other bottom-up procedures capable of handling such theories.

Of course under the best case scenario the opposite may happen. A heavily disjunctive database in which short bodies and long heads are the norm will transform into a more manageable one with a more compact model structure. An originally non-range restricted database can become range restricted after the transformation resulting in simpler processing. In all cases it is important to emphasize that the transformation itself can be implemented in linear time in the size of the database.

During the model generation process we may want to give priority to clauses that reduce the search space by pruning models as early as possible. Denial rules constitute one such class of clauses.

If checking for the derivability of clauses representing the set of minimal models is performed against the elements of EDB then only atoms occuring in EDB are relevant to this checking process. One can take this optimization a step further if the set of atoms of EDB is known. Call this set $\mathcal{S}$. Clearly only elements of the set $Min(\{M' | M \in \mathcal{MM}(IDB^d_Q)\ and\ M' = M \cap \mathcal{S}\})$ are relevant. The fact that only minimal elements need to be checked is evident. One can even incorporate this optimization into the minimal model generating process by abandoning models in which the elements of $\mathcal{S}$ is a superset of an already generated model. Additionally, any rule that cannot contribute to the $\mathcal{S}$ content of models (in a particular computation of models) may be disabled.

6 Conclusions and Future Work

We presented a simple approach to utilize an essentially forward chaining model generation procedure to process queries in a backward chaining mode. The idea is to utilize a certain version of the duality principle to reinterpret the clauses of the input theory so that the application of a model generating procedure will answer the posed query in a top-down fashion.

From a theoretical perspective, the results reported here constitute an elaboration on the strong connection between the seemingly separated concepts used to characterize disjunctive theories in terms of clauses and models; minimal model set and minimal model state; model trees and clausal trees; forward chaining and backward chaining; and query answering and model generation [30, 25]. From a practical point of view, the gain achieved from processing queries as proposed by our algorithm can result in substantial savings due to the limited search space explored. This is especially felt when using approaches based on model generating procedures for query answering [17, 5] which tend to expand a much larger space than required for query answering. In fact they usually process the entire set of clauses in EDB before starting to expand elements of the IDB which may be the only components directly matching the query. At the same time we achieve our aim which is to benefit from the wealth of theory and algorithms available and under development for efficient model generation to perform a goal focused search for answers to the given query [29, 9, 5, 20, 24]. Our preliminary testing points to orders of magnitude performance improvement in using a minimal model based query answering procedure on the dual theory to achieve the top-down processing mode as opposed to having the same procedure operate on the input theory in a bottom-up mode on the same query[9]. The fact that the same procedure can be used in both directions gives the user much flexibility in selecting the direction of processing depending on the theory and query under consideration.

In contrast to other approaches reported in the literature for achieving a similar effect, ours is applicable to disjunctive theories [19] and avoids the explicit introduction of new predicates into the theory for this purpose [1, 7, 23, 27]. Rather we achieve the required results by reinterpreting the clauses (and consequently the logical connectives) in a dual mode. As is the case for the original database, the transformed theory is a disjunctive deductive database (DDDB). The dual transformation is quite simple and involves only changing the direction of implications in all clauses. Only clauses relevant to the query under consideration need to be processed in this fashion. The result is the creation of a (tree) structure that consists of positive clauses the subsumption of which in the original theory (more precisely, the EDB component of the theory) implies the derivability of the query. The inclusion of the clauses corresponding to the elements of the extensional database in the model generation process makes it act in a refutation-like manner and may improve the performance by detecting clause subsumption as early as possible. The reasoning behind the duality approach was shown to be natural and based on solid logical grounds. It is equivalent to working with reversed polarities of literals and using clause to propagate particular truth values.

The method discussed in [14] uses special data structures (deduction trees) and algorithms to achieve top-down query answering without transforming the theory. But the approach there doesn't offer the refined query answering capabilities of the approach outlined here.

[9] Our testing was performed on a prototype implementation of the minimal model generator MM-Satchmo as described in [5]. The theory and the query were presented in both original form and manually transformed dual form. The gains were achieved when both ground and range-restricted (in both directions) theories were used.

[21] outlines a method based on using SLO-resolution to modify the WAM approach so that it deals with disjunctive logic programs. The modification, Disjunctive WAM (DWAM), uses clause subsumption as the basic expansion mechanism and operates in a goal oriented fashion for query answering. In contrast with the method outlined in this paper, the DWAM approach is applied to theories without constraints. However, it is straight forward to extend the DWAM methodology to the case of headless clauses, since subsumption for such clauses is trivial. As in the case here, caution should be exercised so that not to over-expand goal sets using constraints (with trivially subsumed clause heads).

A more substantive difference is that our approach avoids much of the nondeterminism that causes problems for the SLO-based DWAM approach. Rather than searching for alternative ways to subsume a goal we try all possible subsumptions in a predetermined order. Additionally, as opposed to the approach of [21], our definition of the goal set explicitly excludes a major class of irrelevant clause expansions that can cause the search space to explode. Matching clauses are expanded only if they can contribute something new to the refutational process. No clause is used for expansion more than once in a single branch and we never expand using a clause with body atoms intersecting with the set of atoms in the current branch. This makes it possible to avoid many useless expansions.

While we still have to choose from among clauses with matching heads, our expansion of the goal tree is deterministic: we always select the leftmost goal from the first (in the given clause order) potentially useful clause and insist on solving it (or failing) before moving (backtracking) to the next goal in that clause. In a sense this makes our search more focused: at every stage we are concerned with the solvability of a particular goal set. Subsequent goal sets are considered, if required, only after the decision on the current set is made (success or failure).

The fact that our approach is based on a different clause expansion paradigm makes it possible to avoid the extensive rewriting (I-code) needed for the DWAM approach to account for the nondeterminism of subsumption checking and the indexing needed to keep track of clause usage. Rather, we use the syntactic duality transformation and that alone makes it possible to utilize already existing bottom-up procedures, with their efficiency enhancing techniques, to process queries top-down.

An added advantage of our approach is that it is able to specify the conditions under which the goal set expansion can be discontinued and to interpret the cases when the expansion succeeds or fails. The results of the latter can be used to refine the query answering process.

Our approach makes it possible for the user to divide the query answering process into two stages: generating a complete (sufficient) set of clauses that need to be checked for derivability in the extensional component of the database and the actual checking process. This can be employed to achieve optimal access time to the EDB when it is stored in slower memory. In this case the procedure will be run in the model generating mode in which the elements of EDB do not participate in the first stage of query processing. Additionally, the separation of the two stages makes it possible to localize updates to individual components of the database. If one accepts the premise that updates are performed on the EDB component then the model generation process will not be affected by the update process [13]. It is even possible to use different representations for the two components. For example one may select to represent the set of clauses that need to be checked for derivability ($\mathcal{MM}(IDB^d_Q)$) in the form of a model tree which can be viewed as the compiled version of the query and IDB while representing EDB as a clausal tree [26, 10]. Alternatively, interleaving accesses to the different components of the database will make it possible to operate the procedure in the refutation mode, where the aim is to derive the empty clause. In this case the procedure will operate on a (possibly much) larger theory that includes the transformed EDB component of the database but will tend to abandon models before they are fully constructed. The choice of the mode will depend on the relative sizes of the database components and their relative access times.

While other proof procedures can be utilized to describe the utilization of the duality approach, we selected the model generation procedure due to the wealth of information it can produce that was of use in explaining the potential of the advanced method. A side effect of the approach adopted here is that it enables more refined query answering: we are no more confined to returning *yes/no* answers to the query but we can also specify the conditions under which this query becomes derivable. The set of models returned by the model generating component can be viewed as a sort of intentional answer to the query: it specifies the necessary conditions under which the query becomes derivable from the database. Even when run in the refutation mode, the surviving models still specify the conditions under which the query can be made to become derivable from the database. These conditions are minimal in the sense that they specify the least information (weakest clauses) that need to be added to the EDB to ensure the derivability of the query. This part is relevant to the view update problem in deductive databases which has been the focus of a major research effort. The various aspects for the utility of our approach to solving the update problem is to be further investigated.

While we emphasized the use of a minimal model generating procedure, that was for efficiency reasons. Any complete model generating procedure (one that returns all the minimal models of its input, among others) will do the job. A sound minimal model generating procedure is clearly superior especially when operating in the model generating mode. It returns exactly the necessary and sufficient set of clauses that need to be tested to ensure the query derivability. Nonminimal models are irrelevant if the query is derivable since the minimal components need to be checked anyway. The gains can be substantial. If the user is interested in *yes/no* answers only; then the failure to derive nonminimal clauses as well can be used as a basis for abandoning the subsequent processing and reporting a *no* answer. In fact any procedure that is refutational sound and complete can be employed to generate answers to positive queries using the duality approach outlined in this paper.

The debate over which direction for clause evaluation: bottom-up or top-down performs best was addressed extensively in the literature [6, 19, 22, 27, 28]. Our presentation here is not meant to solve this issue but rather to offer the user a choice. As a matter of fact, since $(DB^d)^d = DB$ it is immediate to note that for any theory that performs better for one approach there is a theory that performs worse. A strong argument for the top-down approach is that many deductive databases fall into the class where it is likely to perform better [27]. The main disadvantage of the approach outlined in this paper is the requirement that the transformed theory be range restricted before it is submitted to the model generating procedure. The range restricted transformation may introduce too much instantiations, especially in the cases when the model generation and checking are performed as separate stages of the query answering process. Nonground queries may also contribute to the transformed theory becoming nonrange restricted. However, interesting results are being reported in the literature to relax the need for the range restrictedness property when using model generation based procedures [3, 2, 4]. Our method will be able to utilize these results. The detailed study of this utility is a possible topic for future research. Additionally, in cases when indefinite answers are allowed the issue of answer minimality is still as problematic in our approach as is the case for other approaches [18].

The duality approach as outlined here suggests using the same direction (top-down or bottom-up) for processing all clauses of the theory. It is of interest to combine both directions for clause evaluation in a single query answering run to try to achieve *optimal* performance.

Other topics of further study are the ability of the duality approach to handle query answering under various database semantics, e.g. in the presence of negation as failure in clause bodies and multiple types of negation and the use of the defined approach in database update, abduction and answering nonpositive queries.

Acknowledgments

I thank U. Furbach, J. Lobo, I. Niemalä and D. Seipel for their useful comments on earlier drafts of this paper. This research was done while the author was visiting with the PMS-Group at the Institut für Informatik, LMU München on an Alexander von Humboldt Research Fellowship. The support of the PMS-Group and the Alexander von Humboldt Stiftung is appreciated.

References

[1] F. Bancilhon, Y. Sagiv, and J. Ullman. Magic sets and other strange ways to implement logic programs. In *Proceedings of the Fifth ACM SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–15, 1988.

[2] P. Baumgatrner, U. Furbach, and I. Niemmelä. Hyper tabluaux. Technical Report 8-96, Institut für Informatik, Univesität Koblenz, Koblenz, Germany, feb 1996.

[3] B. Beckert and J. Posegga. leanTAP: Lean tableau-based deduction. *Journal of Automated Reasoning*, 4(15):339–358, 1995.

[4] J.P. Billon. The disconnection method: a confluent integration of unification in the analytic framework. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 110–126, Palermo, Italy, May 1996. Springer-Verlag. Vol. 1071.

[5] F. Bry and A. Yahya. Minimal model generation with positive unit hyper-resolution tableaux. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 143–159, Palermo, Italy, May 1996. Springer-Verlag. Vol. 1071, Full version: http://www.pms.informatik.uni-muenchen.de/publikationen/.

[6] F. Bry. Query evaluation in recursive databases: bottom-up and top-down reconciled. *Data & Knowledge Engineering*, pages 289–312, 1990.

[7] R. Demolombe. An efficient strategy for non-horn deductive data bases. *Theoretical Computer Science*, 78:245–259, 1991.

[8] J. A. Fernández and J. Minker. Bottom-up computation of perfect models for disjunctive stratified theories. *Journal of Logic Programming*, 25(1):33–50, 1995.

[9] J. A. Fernández and J. 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.

[10] L.J. Henschen and S.A. Naqvi. On compiling queries in recursive first-order databases. *J.ACM*, 31(1):47–85, January 1984.

[11] K. Inoue, M. Koshimura, and R. Hasegawa. Embedding negation as failure into a model generation theorem prover. In *Proceedings of the Eleventh International Conference on Automated Deduction*, Saratoga Springs, NY, 1992.

[12] K. Inoue and C. Sakama. A fixpoint characterization of abductive logic programs. *Journal of Logic Programming*, 27:107–136, 1996.

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

[14] C.A. Johnson. Top down deduction in indifinite deductive databases. In *Journees Bases de Donnees Avancees*, pages 119–138, Toulosuse, France, 1993. Computer Secience Technical Report Number TR93-08, Keele University, UK.

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

[16] D.W. Loveland, D. Reed, and D. Wilson. Satchmore: Satchmo with relevancy. *J. Automated Reasoning*, 14:349–363, July 1995.

[17] R. Manthey and F. Bry. Satchmo: a theorem prover implemented in prolog. In J.L. Lassez, editor, *Proc.* 9^{th} *CADE*, pages 456–459, 1988.

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

[19] V. Neiman. Refutation search for horn sets by a subgoal-extraction method. *Journal of Logic Programming*, 9:267–284, 1990.

[20] I. Niemelä. A tableau calculus for minimal model reasoning. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 278–294, Palermo, Italy, May 1996. Springer-Verlag. Vol. 1071.

[21] A. Rajasekar and H. Yusuf. Dwam - a wam model extension for disjunctive logic programming. *Annals of Mathematics and Artifitial Intelligence*, 14:275–308, 1995.

[22] R. Ramakrishnan and S. Sudarshan. Top-down vs. bottom-up revisited. In *Proceedings of the ISLP'91*, 1991.

[23] J. Rohmer, R. Lescoeur, and J-M. Kerisit. The alexander method: a technique for the processing of recusive axioms in deductive databases. *New Generation Computing*, 4(3), 1986.

[24] H. Schüz and T. Geisler. Efficient model generation through compilation. In *Proceedings of the 13^{th} International Conference on Automated Deduction (CADE)*, New Jersey, USA, July 1996. Springer-Verlag. LNCS series.

[25] D. Seipel, J. Minker, and C. Ruiz. Model generation and state generation for disjunctive logic programs. Technical Report CS-TR-3546, UMIACS-TR-95-99, University of Maryland at College Park, Umiacs and CS department, College Park, Maryland, August 1995.

[26] D. Seipel. *Efficient Reasoning in Disjunctive Deductive Databases.* Habilitation thesis, Fakultät für Informatik, Universität Tübingen, June 1995.

[27] M. Stickel. Meta interpretation of the model elimination theorem proving procedure for deduction and abduction. *J. Automated Reasoning*, 13(2):189–210, October 1994.

[28] R. Treitel and M. Genesereth. Choosing directions of rules. *J. Automated Reasoning*, 3(2):395–431, October 1987.

[29] A. Yahya, J.A. Fernandez, and J. Minker. Ordered model trees: A normal form for disjunctive deductive databases. *J. Automated Reasoning*, 13(1):117–144, 1994.

[30] A. Yahya and J. Minker. Representations for disjunctive deductive databases. Technical Report CS-TR-3111 UMIACS-TR-93-70, Department of Computer Science and UMIACS, University of Maryland, College Park, July 1993.