COMPUTING PERFECT AND STABLE MODELS USING ORDERED MODEL TREES

Article *in* Computational Intelligence · February 1995

DOI: 10.1111/j.1467-8640.1995.tb00024.x · Source: DBLP

Adnan Yahya
Birzeit University
41 PUBLICATIONS **439** CITATIONS

Computing Perfect and Stable Models Using Ordered Model Trees*

José Alberto Fernández[2] Jack Minker[1,2] Adnan Yahya[1,3†]

[1]Institute for Advanced Computer Studies

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

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

Abstract

Ordered Model trees were introduced as a normal form for disjunctive deductive databases. They were also used to facilitate the computation of minimal models for disjunctive theories by exploiting the order imposed on the Herbrand base of the theory. In this work we show how the order on the Herbrand base can be used to compute perfect models of a disjunctive stratified finite theory. We are able to compute the stable models of a general finite theory by combining the order on the elements of the Herbrand base with previous results that had shown that the stable models of a theory T can be computed as the perfect models of a corresponding disjunctive theory $\mathcal{E}T$ resulting from applying the so called evidential transformation to T.

While other methods consider many models that are rejected at the end, the use of atom ordering allows us to guarantee that every model generated belongs to the class of models being computed. As for negation-free databases, the ordered tree serves as the canonical representation of the database.

1 Introduction

Minimal model trees were introduced in [6] as a structure sharing approach to represent information in disjunctive deductive databases (DDDBs). They were shown to capture the semantics of this class

*Financial support provided by the National Science Foundation under the grant Nr. IRI-89-16059 and by the Air Force Office of Scientific Research under the grant Nr. AFOSR-91-0350

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

of databases. In [20] *ordered minimal model trees* were introduced as a normal form for model trees. Informally, given a total order on the atoms of the Herbrand base of a DDDB, DB, an ordered model tree for DB is a tree structure in which nodes are labeled by atoms of HB_{DB}. With each branch (i.e. path from the root to a leaf node) of the tree we associate the set of atoms encountered in that branch. Branches represent ordered models of DB and the branches in a left-to-right traversal of the tree are ordered. An ordered minimal model tree is an ordered model tree for which there is a one to one correspondence between minimal models and branches in the tree. Algorithms for constructing and updating ordered minimal model trees were presented in [20]. Ordered model trees treat the minimization problem of model tree computation by exploiting the order in the tree for restricting the search space.

Perfect models were defined in [17] as the proper semantics for disjunctive stratified theories. For such theories, perfect model semantics uses the hierarchical structure of the theory to deal with negative information. In this work we exploit this hierarchical structure to define an order on the Herbrand base of the theory. This order is used to construct the ordered model tree. The manipulation of the order allows us to modify the algorithms of [20] to compute perfect, rather than minimal, models of the theory under consideration.

Furthermore, we use the algorithms we develop for computing perfect models and the results reported in [5], relating to the computation of the stable models of non-stratified theories, to extend our work to the class of disjunctive normal databases. We show that the algorithms developed in this paper have distinct advantages over existing algorithms for computing perfect and stable models. We can summarize the advantages of our algorithm as follows.

- As opposed to other approaches, our algorithm integrates in one process the generation of models and the test for minimization.
- The acceptance of a model as perfect or stable is decided as soon as the model is generated. Other approaches must wait until all models are generated until finally deciding the status of individual models.
- The algorithm exploits to a great extent the internal structure of the disjunctive database to be able to direct the computation of stable models.
- Our transformational approach to stable model semantics is conservative in the sense that it diminishes the amount of extra models that need to be inspected.
- Finally, the algorithm is parametrized for different semantics which allows us to use the same algorithm to compute minimal, perfect and stable models on the same program depending on the parameters.

The paper is organized as follows. In section 2 we give definitions and background material related to DDDBs and their ordered model trees. In section 3 we describe algorithms based on ordered model tree concepts to compute the perfect models of a DDDB and to construct its unique ordered perfect model tree. In section 4 we address the issue of how to modify the algorithms in section 3 to compute stable, instead of perfect models. We conclude by comparing the results of this paper with other approaches reported in the literature. We also discuss possible directions for future research.

2 Notation and Background

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

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

where $n \geq 0, k > 0$ and the A_i and B_j are atoms defined using a FOL $\mathcal{L}$ that does not contain function symbols. We assume that all clauses in DB are ground. Each clause of DB represents the following first order formula:

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

If $k = 1$ for all clauses in DB, the database is called a *definite deductive database*.[1] A clause C is positive (negative) iff $n = 0$ ($k = 0$).

Definition 2.1 *[20] Let DB be a DDDB and let A be a ground atom in the Herbrand base of DB. Let C be a clause of DB. Then*

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

2.1 Minimal Models and Model Trees

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

Definition 2.2 *[20] Let DB_1 and DB_2 be DDDBs and let $\mathcal{MM}(DB_1)$ and $\mathcal{MM}(DB_2)$ be the sets of their minimal models, respectively. Then*

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

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

Definition 2.3 *[6] Let $\mathcal{I}$ be a finite set of Herbrand interpretations (models) over a FOL $\mathcal{L}$.* An interpretation (model) tree for *$\mathcal{I}$ is a tree structure where*

- *The root is labeled by the special symbol ε. Other nodes are labeled with atoms in $\mathcal{I}$ or the special symbol $\not\epsilon$.*

[1] We will use the term *non-disjunctive deductive database* to emphasize the fact that the database is not disjunctive.

- *A path from the root to a leaf node is called a* branch. *No atom occurs more than once in a branch.*
- $I \in \mathcal{I}$ *iff* $\exists b_I \; I = \{A : \; A \; \in b_I\} - \{\, \varepsilon, \not\varepsilon\}$ *where* b_I *is a branch in the tree.*

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

In general, to capture a semantics of a disjunctive database, DB, a model tree needs to represent the set of models that characterize DB under that semantics. Under the minimal model semantics, model trees of databases are defined as follows.

Definition 2.4 *Let* DB *be a disjunctive deductive database and let* $\mathcal{MM}(DB)$ *be the minimal models of* DB*. Then* $\mathcal{T}_{\mathcal{M}}$ *is called a* model tree of DB *iff*

$$\mathcal{MM}(DB) = \mathcal{M}.$$

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

2.2 Ordering the Herbrand Base

In what follows, we present the concept of order on the Herbrand base of a DDDB and the extensions of this order to more complex structures: interpretations, sets of interpretations and model trees.

Definition 2.5 *[20] Let* S *be a countable set of atoms. An order* $(>)$ *on* S *is defined by the function* $\mathcal{O} : S \to \{1, ..., card(S)\}$*, one to one and onto and such that:*

- $A > B$ *iff* $\mathcal{O}(A) > \mathcal{O}(B)$ *for* A *and* B *in* S*.*
- $B < A$ *iff* $A > B$*.*

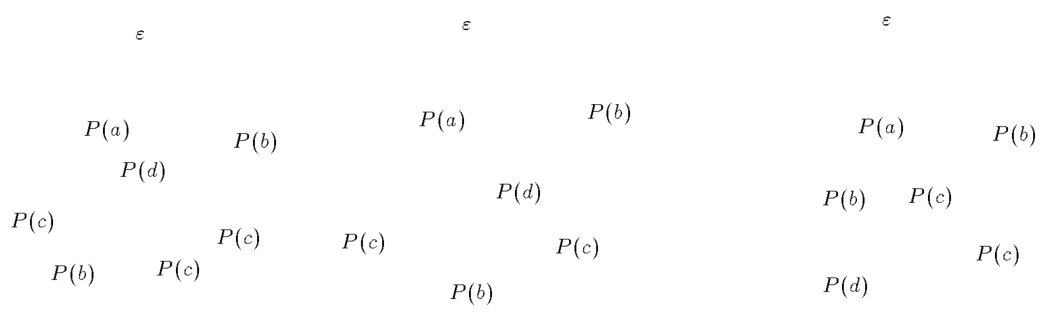
A sequence of atoms $\langle A_1, \ldots, A_N\rangle$ *is ordered iff* $\forall_{i<j} A_i > A_j$ *for* A_i *and* A_j *in* S*. We denote by* R^o *the ordered sequence containing the atoms in the set* R*, where* $R \subseteq S$*.*

We use the term *ordered set* R to refer to the ordered sequence R^o. In particular, we use the term *ordered Herbrand base,* $HB_{\mathcal{L}}$*,* to refer to the ordered sequence $HB^o_{\mathcal{L}}$ which completely specifies the total order $(>)$ of the atoms of the Herbrand base of $\mathcal{L}$.

Definition 2.6 *[20] Given an order* $(>)$ *on* S*, let* $R_1 = \langle A_1, \ldots, A_N\rangle$ *and* $R_2 = \langle B_1, \ldots, B_M\rangle$ *be ordered sets of atoms with the* A_i*s and* B_j*s in* S*. Then:*

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

Definition 2.7 *[20] Given an order* $(>)$ *on* $HB_{\mathcal{L}}$*, let* $\mathcal{I} = \{I_1, \ldots, I_n\}$ *be a set of interpretations (models or minimal models) over* $\mathcal{L}$*. Then the ordered set of interpretations (models or minimal models) of* $\mathcal{I}$ *is the sequence* $\mathcal{I}^o = \langle I^o_{s_1}, \ldots, I^o_{s_n}\rangle$ *such that* $\forall_{i<j} I^o_{s_i} > I^o_{s_j}$*.* I^o_i *is referred to as the ordered interpretation (model or minimal model)* I_i*.*



a: Nonminimal Unordered Tree b: Minimal Unordered Tree c: Minimal Ordered Tree

Figure 1: Ordered and Unordered Model Trees with $P(a) > P(b) > P(c) > P(d)$

Example 1 *Given a set of interpretations* $\mathcal{I} = \{\{C, A, B\}, \{A, D, B\}, \{\}, \{A, B\}\}$ *over the ordered Herbrand base* $\langle D, C, B, A\rangle$*, the corresponding ordered set of interpretations of* $\mathcal{I}$*,* $\mathcal{I}^o = \langle\langle D, B, A\rangle, \langle C, B, A\rangle, \langle B, A\rangle\langle\rangle\rangle$*.*

Definition 2.8 *[20] Let* $HB_{\mathcal{L}}$ *be an ordered Herbrand base and let* $\mathcal{I}$ *be a set of interpretations (models, minimal models) over* $HB_{\mathcal{L}}$*. Let* $\mathcal{T}_{\mathcal{I}}$ *be a model tree for* $\mathcal{I}$*. Then*

1. *A left-right traversal of* $\mathcal{T}_{\mathcal{I}}$ *is a sequence* $\langle S_1, \ldots, S_n\rangle$ *such that* $\langle b_1, \ldots, b_n\rangle$ *is a left-to-right enumeration of the branches in* $\mathcal{T}_{\mathcal{I}}$ *and* S_i *is the sequence* $\langle A_{i,1}, \ldots, A_{i,m_i}\rangle$ *corresponding to a root-to-leaf traversal of the branch* b_i*.*

2. $\mathcal{T}_{\mathcal{I}}$ *is ordered iff* $\mathcal{I}^o = \langle S_1, \ldots, S_n\rangle$*, the left-right traversal of* $\mathcal{T}_{\mathcal{I}}$*.* $\mathcal{T}_{\mathcal{I}}$ *is called the* ordered interpretation (model, minimal model) tree for $\mathcal{I}$*, and is denoted as* $\mathcal{T}^o_{\mathcal{I}}$*.*

Example 2 *Let* $DB = \{P(a) \vee P(b), P(a) \vee P(c), P(c) \vee P(d), P(b) \vee P(c)\}$*. Figure 1 shows: (a) a nonminimal unordered model tree , (b) a minimal unordered model tree and (c) a minimal ordered model tree for DB with the order* $P(a) > P(b) > P(c) > P(d)$*.*

Theorem 2.9 *[20] Let* DB_1 *and* DB_2 *be databases with a common ordered Herbrand base,* $HB^o_{\mathcal{L}}$*, and let* $\mathcal{T}^o_{DB_i}$ *be the ordered minimal model tree for* DB_i*. Then,* $DB_1 =_{mm} DB_2$ *iff the left-right traversals of* $\mathcal{T}^o_{DB_1}$ *and* $\mathcal{T}^o_{DB_2}$ *coincide.*

That is the ordered minimal model tree representation is unique for a given database and order.

2.3 Building Ordered Model Trees

Here we give a brief description of the algorithm used in [20] to construct the minimal ordered model tree for a disjunctive deductive database, DB.

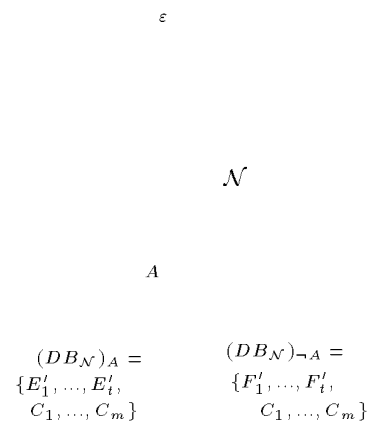

ε
$\mathcal{N}$
A
$(DB_{\mathcal{N}})_A = \{E'_1, \ldots, E'_t, C_1, \ldots, C_m\}$
$(DB_{\mathcal{N}})_{\neg A} = \{F'_1, \ldots, F'_t, C_1, \ldots, C_m\}$

Figure 2: A step in the Tree Construction Process

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

Assume that we are dealing with a ground disjunctive deductive database with the ordered Herbrand base $\langle A_1, \ldots, A_N \rangle$. Let A be the largest atom that occurs positively in the current database, DB. Let DB be of the form $\{F_1, \ldots, F_n, E_1, \ldots, E_t, C_1, \ldots, C_m\}$ where the F_i are the clauses that contain A, the E_k are the clauses containing $\neg A$, and the C_j are clauses that do not contain occurrences either of A or of $\neg A$. Let $F'_i = F_i - A$ (i.e. the result of removing the literal A from the clause F_i) and $E'_k = E_k - \neg A$ (i.e. the result of removing the literal $\neg A$ from the clause E_i). We assume that no tautologies are present in DB (i.e. no clause contains both A and $\neg A$).

The first order theory represented by DB can be rewritten as $\{(A \vee (F'_1 \wedge \cdots \wedge F'_n)), (\neg A \vee (E'_1 \wedge \cdots \wedge E'_t)), C_1, \ldots, C_m\}$. Hence, after expanding A, the decomposition of DB creates the new databases,

$$\begin{aligned} DB_A &= \{E'_1, \ldots, E'_t, C_1, \ldots, C_m\} \\ DB_{\neg A} &= \{F'_1, \ldots, F'_n, C_1, \ldots, C_m\} \end{aligned}$$

In each step of the construction, the current database $DB_{\mathcal{N}}$ is decomposed to construct subtrees *underneath* a particular node $\mathcal{N}$ of the tree (initially ε, the root). At this point, no atom in the path from the root to $\mathcal{N}$ occurs in $DB_{\mathcal{N}}$. The leftmost child of $\mathcal{N}$ is the node A with the database $(DB_{\mathcal{N}})_A$ to be expanded *underneath* A. The database $(DB_{\mathcal{N}})_{\neg A}$ will be expanded *underneath* $\mathcal{N}$ and to the right of A (it will generate the subtrees of $\mathcal{N}$ to the right of A). The process is recursively applied to these two databases. Figure 2 illustrates the expansion process. $(DB_{\mathcal{N}})_A$ and $(DB_{\mathcal{N}})_{\neg A}$ are smaller than $DB_{\mathcal{N}}$ in the sense that $(DB_{\mathcal{N}})_A$ has less clauses than DB, and some clauses of $DB_{\neg A}$ contain fewer literals than the clauses in DB. Two terminating conditions are possible:

1. No atom occurs positively in $DB_{\mathcal{N}}$ (e.g. $DB_{\mathcal{N}}$ is empty). The branch represents a model, not

necessarily minimal, of the original database.

2. $DB_{\mathcal{N}}$ contains the empty clause, $\Box$, (the clause with no literals). In this case, $DB_{\mathcal{N}}$ is inconsistent, it has no model, and the node $\mathcal{N}$ is labeled *nil*.

After a recursive expansion of a node $\mathcal{N}$ is completed, any *nil* child of $\mathcal{N}$ is removed from the tree. If all children of $\mathcal{N}$ are *nil*, then the node itself is labeled *nil*. If the root node becomes *nil*, then the original database has no models. It is inconsistent.

One of the most important properties of the ordered model tree construction algorithm presented in [20] is that it guarantees that the rightmost branch of the resulting tree corresponds to a minimal model of the database. Based on this result, it is possible to devise an algorithm that generates only minimal models of DB.

After a minimal model of DB, $M \in \mathcal{MM}(DB)$, is found, $DB =_{mm} M \vee (DB \cup \{M^{\neg}\})$, where $M^{\neg} = (\bigvee_{A \in M} \neg A)$ (the negative clause containing negative occurrences of all and only the atoms of M). The idea is that, we are suppressing the minimal models already generated and their supersets by including the negation of the model M as a denial $M^{\neg}$ in the remaining database. The process terminates when all minimal models are found and the database, together with the added negative clauses, is inconsistent. The order in the tree makes it unnecessary to pass all atoms of an already found minimal model. It is sufficient to pass only the negative clause corresponding to the components in that model not shared by the (potential) model being constructed [20].

Algorithm 1 presents formally the recursive construction of the ordered minimal model tree incorporating this model minimization approach. The parameter $NegSet$ in the algorithm is an internal output parameter the purpose of which is to pass minimization information between tree branches during the ordered minimal model tree construction process. Initially, Algorithm 1 is called as `BuildTree(`$DB, \varepsilon, NegSet$`)`, where DB is a disjunctive deductive database and ε is the root node for the tree (the output value of $NegSet$ is considered irrelevant in this call).

Example 3 *Let*

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

After translating the rules into clausal form

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

We build an ordered minimal model tree for DB assuming the following order $P(a) > P(b) > P(c) > P(d) > P(e) > P(f)$.

Algorithm 1 *Constructing Ordered Minimal Model Trees for Disjunctive Deductive Databases.*

```
Procedure BuildTree($DB$, $Root$; Out:$NegSet$) returns tree;
If $\Box \in DB$ then Return $nil$ with $NegSet$ := $\{\}$;
If $no\ atom\ occurs\ positively\ in\ DB$ then
     return Root with $NegSet$ := $\{\Box\}$;
Let $A$ := $largest\ atom\ that\ occurs\ positively\ in\ DB$;
Let $T_R$ := BuildTree($DB_{\neg A}$, $Root$, $Neg_R$);
Let $T_L$ := BuildTree($DB_A \cup Neg_R$, $A$, $Neg_L$);
Let $NegSet$ := $Neg_R \cup \{(\neg A \vee C) | C \in Neg_L\}$;
If $T_L = nil$ then return $T_R$;                              /*$Neg_L = \{\}$*/
Else if $T_R = nil$ then return AttachLeftNode($Root$, $T_L$);     /*$Neg_R = \{\}$*/
Else return AttachLeftNode($T_R$, $T_L$)
```

We start with $P(a)$

$$\begin{aligned} DB_1 = DB_{P(a)} &= \{P(e) \vee \neg P(b) \vee \neg P(c);\ P(b);\ P(c) \vee P(d) \vee P(f);\ P(f) \vee \neg P(d)\} \\ DB_1' = DB_{\neg P(a)} &= \{P(e) \vee \neg P(b) \vee \neg P(c);\ P(b);\ P(c);\ P(c) \vee P(d) \vee P(f)\}. \end{aligned}$$

The expansion of DB_1' on $P(b)$ yields

$$\begin{aligned} DB_1'' = (DB_1')_{P(b)} &= \{P(c);\ P(e) \vee \neg P(c)\} \\ (DB_1')_{\neg P(b)} &= \{\Box;\ P(c);\ P(c) \vee P(d) \vee P(f)\} \end{aligned}$$

$P(a)$ and $P(b)$ are all the siblings. Further expansion is detailed in Figure 3-a.

After the first minimal model $M_1 = \{P(b), P(c), P(e)\}$ is found we retain it and add the clause $(1) = \neg P(b) \vee \neg P(c) \vee \neg P(e)$ to DB_1. When the second minimal model $M_2 = \{P(a), P(b), P(f)\}$ is generated, only the relevant portions of it, clause $(3) = \neg P(f)$, is added to $(DB_1 \cup \{(1)\})_{P(b))P(c)}$, which corresponds to the submodel $\{A \in M_2 : P(c) > A\}$. Figure 3-a details the full expansion which produces the minimal ordered model tree of Figure 3-b.

3 Computing Perfect Models

In this section and the next, we extend the algorithms for computing minimal ordered model trees to the case of *disjunctive normal databases.* That is, databases with clauses of the form

$$A_1 \vee \ldots A_l \leftarrow B_1, \ldots, B_n,\ not\ D_1, \ldots,\ not\ D_m.$$

The D_k atoms are said to be *negated* in the clause as opposed to positive (A_i) and negative (B_j) atoms.

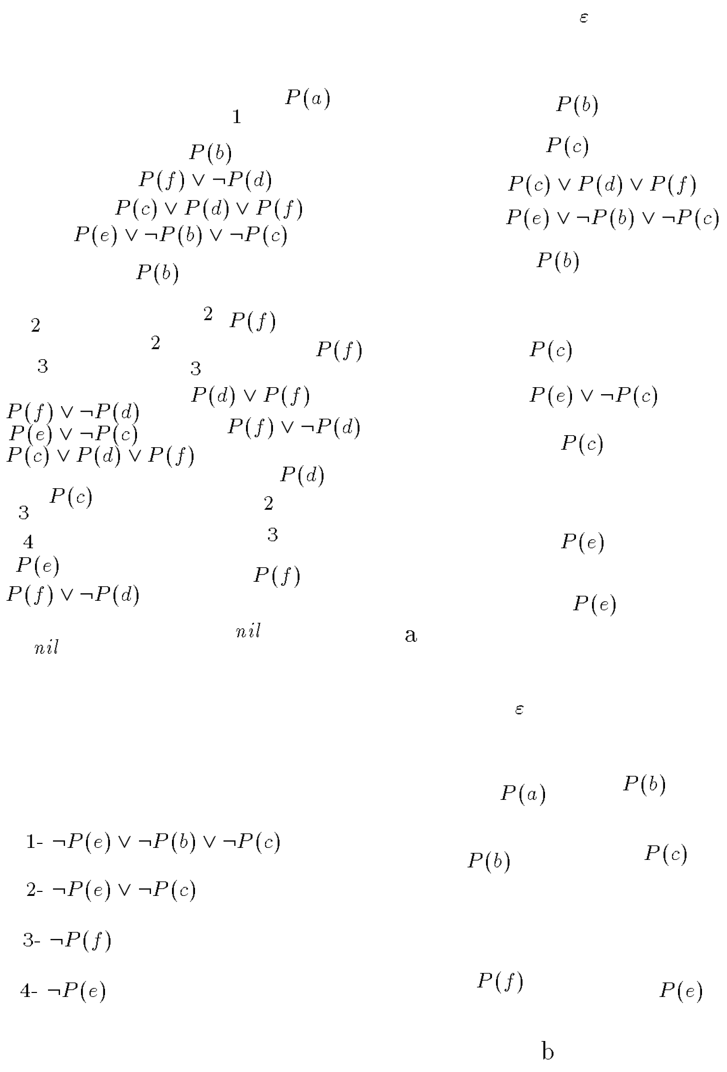


Figure 3: Generating Ordered Minimal Model Trees

For disjunctive normal databases, the usage of minimal model semantics to describe the meaning of databases does not produce the expected set of logical consequences. Let the database contain the single clause $DB = \{P(a) \vee P(b) \leftarrow \mathit{not}\, P(c)\}$. If we interpret the operator "*not*" as denoting logical negation ("$\neg$"), DB has three minimal models $\{P(a)\}, \{P(b)\}$ and $\{P(c)\}$. By the minimal models semantics, the only logical consequence of DB is the clause $P(a) \vee P(b) \vee P(c)$. But, from the syntax, the intended meaning of DB is that $P(a) \vee P(b)$ is a logical consequence if $P(c)$ is *false*, which follows since there is no rule with $P(c)$ on its left hand side. Hence in our intended meaning, the positive clause $P(a) \vee P(b)$ must be a logical consequence, which implies that the semantic characterization of the database must be based only on the two minimal models $\{P(a)\}$ and $\{P(b)\}$. The minimal model $\{P(c)\}$ must be excluded from the semantic characterization. The *perfect model semantics* for stratified databases [17], and the stable model semantics —presented in the next section— for normal databases [9, 18] resolve this situation.

In this section we present the algorithms for the class of disjunctive stratified databases (DSDB) and in the next we discuss algorithms for the general class of disjunctive normal databases (DNDB).

Definition 3.1 *[10, 17] A disjunctive normal database, DB, is called a* disjunctive stratified database (DSDB) *if it is possible to partition the (finite) set S of all predicate symbols in DB into sets $S_1, \ldots, S_r$, called* strata*, such that for every clause of DB*

$$A_1 \vee \ldots A_l \leftarrow B_1, \ldots, B_n, \mathit{not}\, D_1, \ldots, \mathit{not}\, D_m \qquad (l \geq 0)$$

$\exists c$ a constant, $1 \leq c \leq r$, such that $\forall_{1 \leq i \leq l}$, $Stratum(A_i) = c$, $\forall_{1 \leq j \leq n}$, $Stratum(B_j) \leq c$ and $\forall_{1 \leq k \leq m}$, $Stratum(B_k) < c$. Where $Stratum(A) = i$ iff the predicate symbol of the atom A is in stratum S_i. Any partition $S_1, \ldots, S_r$ of S satisfying the above conditions is called a stratification of DB.

To verify that a database is stratified, Definition 3.1 calls for the analysis of the database only at the predicate level. For finite ground databases it is possible to go further, to the atomic level. That is, to let the set $S = HB_{DB}$ and have S_i form the partition blocks of the Herbrand base. This is called *local stratification* [17]. In what follows, we will use local stratification to present our results. Nevertheless, the results also apply to the case of regular stratification since regular stratified databases are also locally stratified (but not vice versa) [10, 17].

Definition 3.2 *[17] Let DB be a disjunctive normal database with the local stratification $S_1, \ldots,$ S_r. Let $A \in S_i$ and $B \in S_j$ be two distinct atoms of the Herbrand base of HB_{DB}. Then B has a higher priority (for minimization) than A ($A \prec B$) if $i > j$.*

Definition 3.3 *[17] Let M and N be two distinct models of a normal disjunctive database, DB. N is preferable to M ($N \prec\!\prec M$) If for every ground atom $A \in (N \setminus M)$ there is a ground atom $B \in (M \setminus N)$ such that $A \prec B$. A model M is perfect if no model of DB is preferable to M.*

Perfect models can also be define in de following way.

Definition 3.4 *[17] Let DB be a disjunctive normal database and let $S_1, \ldots, S_r$ be a local stratification of DB. M is a perfect model of DB iff for every $j \leq r$ the set of atoms of S_j in M (the set $S_j \cap M$) is minimal among all models N of DB for which the atoms of $\cup S_i : i < j$ are the same for M and N.*

Lemma 3.5 *[17] Let M and N be models of a disjunctive deductive database (negation-free). Then $N \prec\prec M$ iff $N \subset M$.*

Lemma 3.5 points to the fact that the concepts of a perfect and minimal model for disjunctive deductive databases coincide. In disjunctive normal databases, these concepts are different.
In general, every perfect model is a minimal model but not vice versa [17]. Consider the following example:

Example 4 *Let $DB = \{P(d) \leftarrow P(b), P(b) \leftarrow \textit{not}\, P(a)\}$. $\{P(a)\}$ and $\{P(b), P(d)\}$ are the two minimal models of DB only the second of which is perfect.*

The algorithms developed in [20] and presented in Section 2 construct ordered model trees for the class of disjunctive deductive databases. A normal program, DB, can be converted into a negation-free DDDB, using the following transformation.

Definition 3.6 *Let DB be a disjunctive normal database. Then,*

$$DB^+ = \{A_1 \vee \ldots A_l \vee D_1, \ldots, \vee D_m \leftarrow B_1, \ldots, B_n : \\ A_1 \vee \ldots A_l \leftarrow B_1, \ldots, B_n, \textit{not}\, D_1, \ldots, \textit{not}\, D_m \in DB\}.$$

That is, DB^+ is the disjunctive deductive database obtained by substituting the default negation (*not*) by the logical operator ($\neg$) and moving the resulting negative literal in the body by their positive counterpart in the head ($\neg\neg A = A$).

Lemma 3.7 *Let DB be a disjunctive normal database. Let DB^+ be the disjunctive deductive database achieved by the transformation in Definition 3.6. Then $DB^+ =_{mm} DB$.*

Proof: The proof is immediate since the transformation preserves logical equivalence. ∎

We would like to have our algorithms operate on the modified programs (DB^+) and manipulate the order selection process to generate the perfect models of the normal database from the negation-free database corresponding to it. To achieve this, the order assigned to an atom needs to reflect the hierarchical structure of the database. Atoms occurring (negated) in rule bodies need to be processed before atoms appearing in the head of that particular rule and the order of the atoms during the tree construction process must reflect this property as it propagates throughout the database. This approach can be utilized to compute the perfect model tree for DB.

Definition 3.8 *Let DB be a disjunctive normal database with the ordered Herbrand base HB^o_{DB}. We say that the order ($>$) of the Herbrand base is compatible with the hierarchical structure of DB iff it is possible to subdivide the range of the order function, $\{1, \ldots, card(HB_{DB})\}$, into disjoint intervals, $INT_1, \ldots, INT_r$, such that for every clause of DB of the form:*

$$A_1 \vee \ldots \vee A_l \leftarrow B_1, \ldots, B_n, \textit{not}\, D_1, \ldots, \textit{not}\, D_m$$

1. *$\mathcal{INT}(A_i) \geq \mathcal{INT}(B_j)$.*
2. *$\mathcal{INT}(A_i) > \mathcal{INT}(D_k)$, $(A_i > D_k)$.*

3. *$\mathcal{INT}(A_i) = \mathcal{INT}(A_k)$. That is, The elements in the head belong to the same interval.*

where $\mathcal{INT}(A) = c$ iff $\mathcal{O}(A) \in INT_c$.

Definition 3.9 *Let DB be a disjunctive stratified database with the ordered Herbrand base HB^o_{DB} and local stratification $S_1, \ldots, S_r$. We say that the order ($>$) is compatible with the stratification iff $\forall_{i<j}$ when $A \in S_i$ and $B \in S_j$ then $\mathcal{O}(A) > \mathcal{O}(B)$, $(A > B)$.*

The following theorem establishes the relationship between local stratifiability of a disjunctive normal database, DB, and the ability to assign an order to the underlying Herbrand base HB_{DB} compatible with the hierarchical structure of DB.

Theorem 3.10 *Let DB be a normal deductive database with the Herbrand base HB_{DB}. It is possible to assign an order to HB_{DB} compatible with the hierarchical structure of DB iff DB is locally stratifiable.*

Proof: Trivial. ∎

3.1 Ordered Models and Preferable Models

We now define the relation between the order relation ($>$) and the preference relation ($\prec\!\prec$) between models as presented in Definition 3.3.

Theorem 3.11 *Let DB be a normal disjunctive database with the local stratification $S_1, \ldots, S_r$. Let the order of HB_{DB} ($>$) be compatible with the stratification of DB. That is, atoms with higher priority for minimization (lower strata) are always expanded first using, say Algorithm 1. Then for any two ordered models M and N of DB, if $M \prec\!\prec N$ then $M < N$.*

Proof: Assume that $M \prec\!\prec N$ and the conditions of the theorem hold. $M \neq N$. Assume that $M > N$. By Definition 2.6 there is an atom $A \in M$ and $B \in N$ such that $A > B$ and all elements in the models less than A or less than B are equal. Since the models are ordered, N cannot have A as all elements below B in the ordered model are less than B. If A and B belong to different strata then $N \prec\!\prec M$. A contradiction. If A and B belong to the same stratum, S_i, then M cannot be preferable to N since the elements of M in S_i (the set $M \cap S_i$) cannot be a subset of the elements of S_i in N (the set $N \cap S_i$) since $A \in (M \cap S_i)$ but $A \notin (N \cap S_i)$. That is, it cannot be the case that $(M \cap S_i) \subset (N \cap S_i)$. A contradiction. ∎

Theorem 3.11 can be used to characterize the type of models generated using Algorithm 1 (with or without minimization).

Corollary 3.12 *Let DB be a normal deductive database and let the order of expansion be compatible with the stratification of DB. Let DB^+ be the disjunctive deductive database corresponding to DB. Then*

1. *If $M \prec\!\prec N$ then M is generated before N (M appears to the right of N) in the ordered model tree expansion of DB^+.*

2. The first model generated, M_p, (the rightmost model in the tree expansion of DB^+) is a perfect model of DB.

3. If DB has one perfect model then the tree construction algorithm will generate the perfect model as the rightmost model of the ordered model tree for DB^+.

Proof: 1. The proof is straight forward from the definition of the tree (Definition 2.8), the relationship between the models in the tree and Theorem 3.11.

2. A result of the M_p being minimal as shown in [20]. Any model preferable to M_p will appear to the right of M_p. But M_p has no right siblings and consequently no minimal models preferable to it. M_p is a perfect model of DB.

3. A direct consequence of 2.

■

Corollary 3.12 guarantees only that the first model generated is perfect. This result can be helpful if we are interested in an individual perfect model. In this case, we can treat normal programs as regular disjunctive programs with no negation in the body and construct the perfect model of DB as long as we restrict ourselves to expanding the atoms in the order of their stratification. Elements of the same stratum can be expanded in arbitrary order.

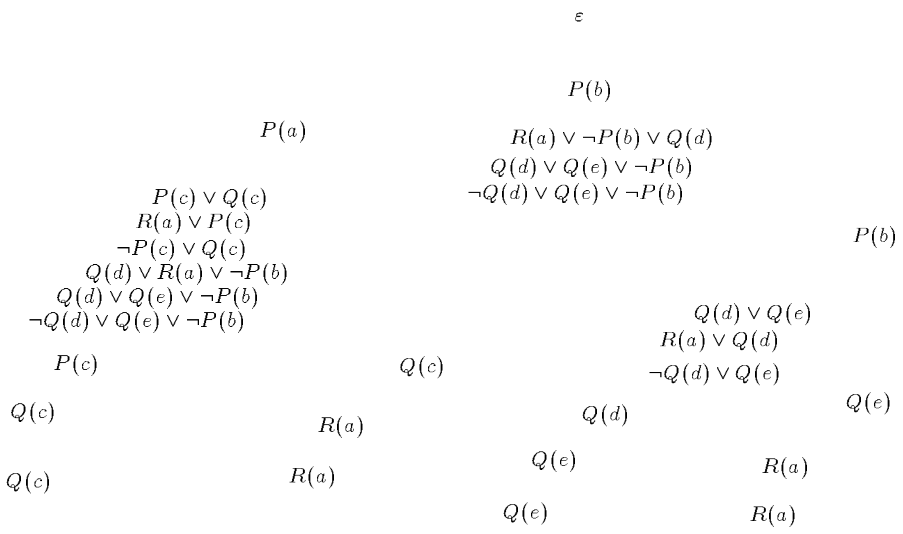
However, if the database has more than one perfect model then, using the tree construction algorithm (Algorithm 1), other nonperfect models can be generated as perfect models need not be consecutive in the ordered minimal model tree corresponding to DB. This is demonstrated by the following example:

Example 5 *Let DB be the following disjunctive stratified database,*

$$
\begin{aligned}
DB = \{ \quad & Q(d) \vee Q(e) \leftarrow P(b), \\
& Q(e) \leftarrow Q(d) \wedge P(b), \\
& P(a) \vee P(b), \\
& P(c) \vee Q(c) \leftarrow P(a), \\
& Q(c) \leftarrow P(c) \wedge P(a), \\
& ------- \\
& R(a) \leftarrow P(a) \wedge \textit{not } P(c), \\
& R(a) \leftarrow P(b) \wedge \textit{not } Q(d) \quad \}
\end{aligned}
$$

Transforming into the corresponding DDDB, and in clausal form, we get

$$
\begin{aligned}
DB^+ = \{ \quad & Q(d) \vee Q(e) \vee \neg P(b), \\
& Q(e) \vee \neg Q(d) \vee \neg P(b), \\
& P(a) \vee P(b), \\
& P(c) \vee Q(c) \vee \neg P(a), \\
& Q(c) \vee \neg P(c) \vee \neg P(a), \\
& ------- \\
& R(a) \vee P(c) \vee \neg P(a), \\
& R(a) \vee Q(d) \vee \neg P(b) \quad \}
\end{aligned}
$$



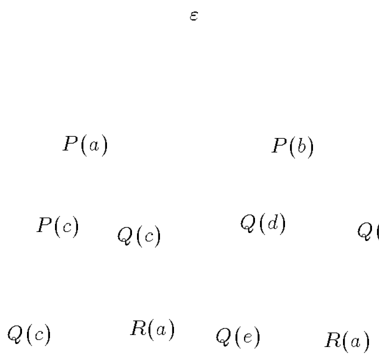


Figure 4: The Minimal Model Tree for Example 5

Let $HB^o_{DB} = \langle P(a), P(b), P(c), Q(c), Q(d), Q(e), R(a)\rangle$. The construction process and the final ordered minimal model tree is shown in Figure 4. The minimal models are $\{P(a), P(c), Q(c)\}$, $\{P(a), Q(c), R(a)\}$, $\{P(b), Q(d), Q(e)\}$, $\{P(b), Q(e), R(a)\}$ from left-to-right. The second and last are the only perfect models.

This can be attributed to the fact that the tree construction process assigns a total order to the atoms of the Herbrand base. All atoms are treated equally as far as minimization is concerned. The construction is oriented to minimizing the total number of atoms in a model rather than the atoms of a higher stratum at the expense of the atoms of lower strata. We would like to adapt the minimization approach used in [20] and discussed briefly in Section 2 to account for the priorities assigned to the different atoms and in this way ensure that no nonperfect model is generated in the tree construction process.

Definition 3.4 and Corollary 3.12 suggest an approach to do this. Given two models M and N, $M \prec\!\not\prec N$ iff $\exists_i$ such that $\forall_{j<i}(S_j \cap M) = (S_j \cap N)$ and $(S_i \cap N) \not\subseteq (S_i \cap M)$ and $(S_i \cap M) \not\subseteq (S_i \cap N)$. It will be sufficient to suppress a model, M, when there is an already found perfect model, N, such that $\exists_i\ \forall_{j<i}(S_j \cap M) = (S_j \cap N)$, $(S_i \cap N) \subset (S_i \cap M)$.

Assume as in Figure 5, that we are currently expanding atom A under node $\mathcal{N}$, where A belongs to stratum S_j ($A \in S_j$). Let M_i be a previously found perfect model of DB that has all the root-to-$\mathcal{N}$ atoms. Add the negative clause corresponding to the (possibly empty) set of atoms of stratum S_j in M_i (the set $S_j \cap M_i$), not expanded so far, to the database being processed. This will have the effect of suppressing models with components in S_j that are supersets of the S_j components of M_i and agreeing with M_i on the atoms of lower strata; that is, nonperfect models during the tree expansion under $\mathcal{N}$.

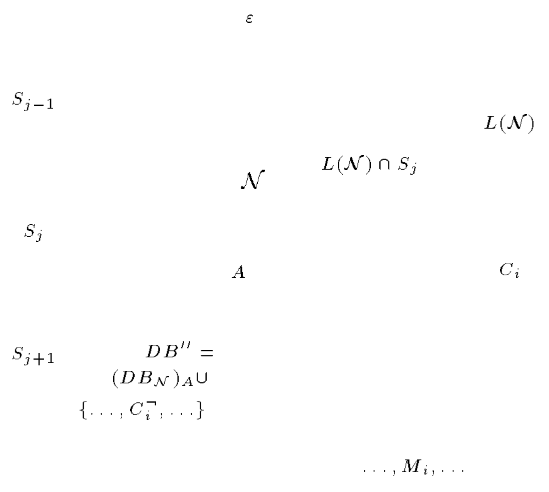


Figure 5: A Step in the Order Perfect Tree Construction

3.2 Building an Ordered Perfect Model Tree

We modify Algorithm 1 to make it construct the set of perfect models for a disjunctive stratified database, DB. The new algorithm operates on DB^+ and the order of the Herbrand base of DB is chosen to be compatible with the stratification of DB.

Assume that we are expanding atom A under node $\mathcal{N}$ as shown in Figure 5. Let the (local) stratification of DB be $S_1, \ldots, S_r$ and $A \in S_j$. Let $L(\mathcal{N})$ be the (partial) interpretation associated with the root-to-$\mathcal{N}$ path in the tree. Assume that $M_1, M_2, \ldots, M_s$ are the perfect models found so far such that $M_i \cap L(\mathcal{N}) = L(\mathcal{N})$, $\forall\ i \in \{1, 2, \ldots, s\}$, These are the perfect models that contain $L(\mathcal{N})$ as a prefix. Let $C_i = (M_i \setminus L(\mathcal{N})) \cap S_j$. That is, C_i is the set of atoms from S_j present in M_i but not processed in the current database so far.

We denote by $C_i^{\neg}$ the clause whose literals are all the negated atoms of C_i. That is, if $C_i = \{A_1, A_2, \ldots, A_m\}$ then $C_i^{\neg} = \{\neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_m\}$. If C_i is empty then $C_i^{\neg}$ is the empty clause, $\Box$. The addition of $\{C_1^{\neg}, \ldots, C_s^{\neg}\}$ to the subdatabase $(DB_{\mathcal{N}})_A$ (rather than the negative clauses corresponding to the whole models in Algorithm 1), guarantees that the models generated are perfect. The correctness of Algorithm 2 is shown by the following theorem.

Theorem 3.13 *Let DB be a normal disjunctive database. Let the order of HB_{DB} be compatible with the local stratification of DB, $S_1, \ldots, S_r$. Assume that the atom being expanded is $A \in S_j$ under node $\mathcal{N}$. Let $C_i = (M_i \cap S_j) \setminus L(\mathcal{N})$, where M_i is an already found perfect model of DB such that M_i has as prefix the partial interpretation associated with the root-to-$\mathcal{N}$ path in the tree, $L(\mathcal{N})$. Let $C_1, \ldots, C_s$ be all such sets and let $C_i^{\neg}$ be the negative clause containing occurrences of all and only atoms of C_i ($\Box$ if C_i is empty). Let $DB'' = (DB_{\mathcal{N}})_A \cup \{C_1^{\neg}, \ldots, C_s^{\neg}\}$. Then, operating on DB^+ and consistently expanding DB'' under A, Algorithm 2 will generate only perfect models of DB.*

Proof: Without loss of generality, let the next model to be generated during the tree construction process, if any, be M. $M > M_i$, $\forall\ i \in \{1, 2, \ldots, s\}$ since it is to the left of these models (M contains A and none of the models found so far contains A). M is smaller than ($<$) any model generated after it (to the right of M) for the same reason. Therefore, by Theorem 3.11 and Corollary 3.12, M can be nonperfect iff $\exists i \in \{1, 2, \ldots, s\}$ such that $M_i \prec\!\prec M$. That is, $\exists_j$ $\forall_{k<j}(S_k \cap M_i) = (S_k \cap M)$, $(S_j \cap M_i) \subset (S_j \cap M)$. The extension of atoms from S_j in M_i must be a subset of the extension of atoms from S_j in M. Let $C_i = \{B | B \in (S_j \cap M_i)\ and\ B < \mathcal{N}\}$. Equivalently, $C_i = (S_j \cap M_i) \setminus L(\mathcal{N})$. By construction, $C_i^{\neg} \in DB''$. Since $C_i \subseteq M$ then M cannot be a model of DB'' which has the negative clause $C_i^{\neg}$. A contradiction. ∎

On the other hand, since a perfect model must have the minimal extension of the first stratum on which it differs from other models (Definition 3.4), it follows that perfect models cannot be suppressed by the addition of the corresponding $C^{\neg}$ elements. These elements will be satisfied by any of the perfect models since a perfect model cannot contain all the atoms of a $C^{\neg}$ clause.

Combined with the fact that the first model generated is perfect, Algorithm 2 is guaranteed to generate all and only the perfect models of DB. Algorithm 2 constructs the *ordered* perfect model tree for DB when it is called with the parameters: `BuildTree(`DB^+`, `ε`, 0, `$NegSet$`)` (we are assuming that stratum 1 is the first set in the stratification). The tree is ordered since the atoms are expanded in the order of their position HB^o_{DB}.

The following example demonstrates the work of Algorithm 2.

Algorithm 2 *Constructing Ordered Perfect Model Trees for Disjunctive Stratified Databases.*

```
Procedure BuildTree($DB$, $Root$, $Strata$; Out:$NegSet$) returns tree ;
If $\Box \in DB$ then Return $nil$ with $NegSet$ := $\{\}$;
If $no\ atom\ occurs\ positively\ in\ DB$ then
    return Root with $NegSet$ := $\{\Box\}$;
Let $A$ := $largest\ atom\ that\ occurs\ positively\ in\ DB$;
Let $T_R$ := BuildTree($DB_{\neg A}$, $Root$, $St(A)$, $Neg_R$);
Let $T_L$ := BuildTree($DB_A \cup Neg_R$, $A$, $Strata$, $Neg_L$);
If $Strata < St(A)$ then $NegSet$ := $\{\Box\}$;
Else $NegSet$ := $Neg_R \cup \{(\neg A \vee C) | C \in Neg_L\}$;
If $T_L = nil$ then return $T_R$;
Else if $T_R = nil$ then return AttachLeftNode($Root$, $T_L$);
Else return AttachLeftNode($T_R$, $T_L$)
```

Example 6 *Let DB be as given in Example 5.*

Let $HB^o_{DB} = \langle P(a), P(b), P(c), Q(c), Q(d), Q(e), R(a)\rangle$ which is compatible with the stratification of DB. The construction process and the final ordered minimal model tree are shown in Figure 6.

As before (see Example 5), the first model generated, $M_1 = \{P(b), Q(e), R(a)\}$ is perfect. Prior to the generation of M_1 we had no perfect models and thus the current databases are not augmented by any additional negative clauses. When $\mathcal{N} = P(b)$ and $A = Q(d)$, that is, we are expanding atom $Q(d)$ under node $P(b)$. The only right sibling of $Q(d)$ is $Q(e)$. The only element of M_1 in the same stratum as $Q(d)$ is $Q(e)$. Therefore, $C_1 = Q(e)$ and $C_1^{\neg} = \neg Q(e)$. $R(a)$, which belong to M_1, was dropped from C_1 since it belongs to a higher stratum and should not be propagated any further in the tree. When added to the current database $\{Q(e)\}$, $C_1^{\neg}$ will cause suppression of the (minimal but) nonperfect model $\{P(b), Q(d), Q(e)\}$ since $\neg Q(e)$ and $Q(e)$ will generate the empty clause.

Clause 1, $\neg P(b) \vee \neg Q(e)$, is used when expanding $P(a)$ under the root. However, since neither $P(b)$ nor $Q(e)$ occurs positively in the current subtree this clause has no effect on generating the (perfect) model $M_2 = \{P(a), Q(c), R(a)\}$. When $\mathcal{N} = P(a)$ and $A = P(c)$ (expanding the leftmost branch of the tree), the clause $\neg P(b) \vee \neg Q(e)$ is retained from the previous stage and additionally the new clause C_2 is generated from the perfect model $M_2 = \{P(a), Q(c), R(a)\}$. $C_2 = Q(c)$ and $C_2^{\neg} = \neg Q(c)$. C_2 will cause suppression of the (minimal but) nonperfect model $\{P(a), P(c), Q(c)\}$.

The only models of the tree are $\{P(a), Q(c), R(a)\}, \{P(b), Q(e), R(a)\}$ which are the perfect models of DB.

It is worthwhile to note that Algorithm 2 generalizes Algorithm 1 in the sense that for a DDDB, the former computes its minimal model tree. Moreover, since for a DNDB, DB, the database DB^+ is negation-free, it can be used to compute the minimal, rather than the perfect, model tree of DB.

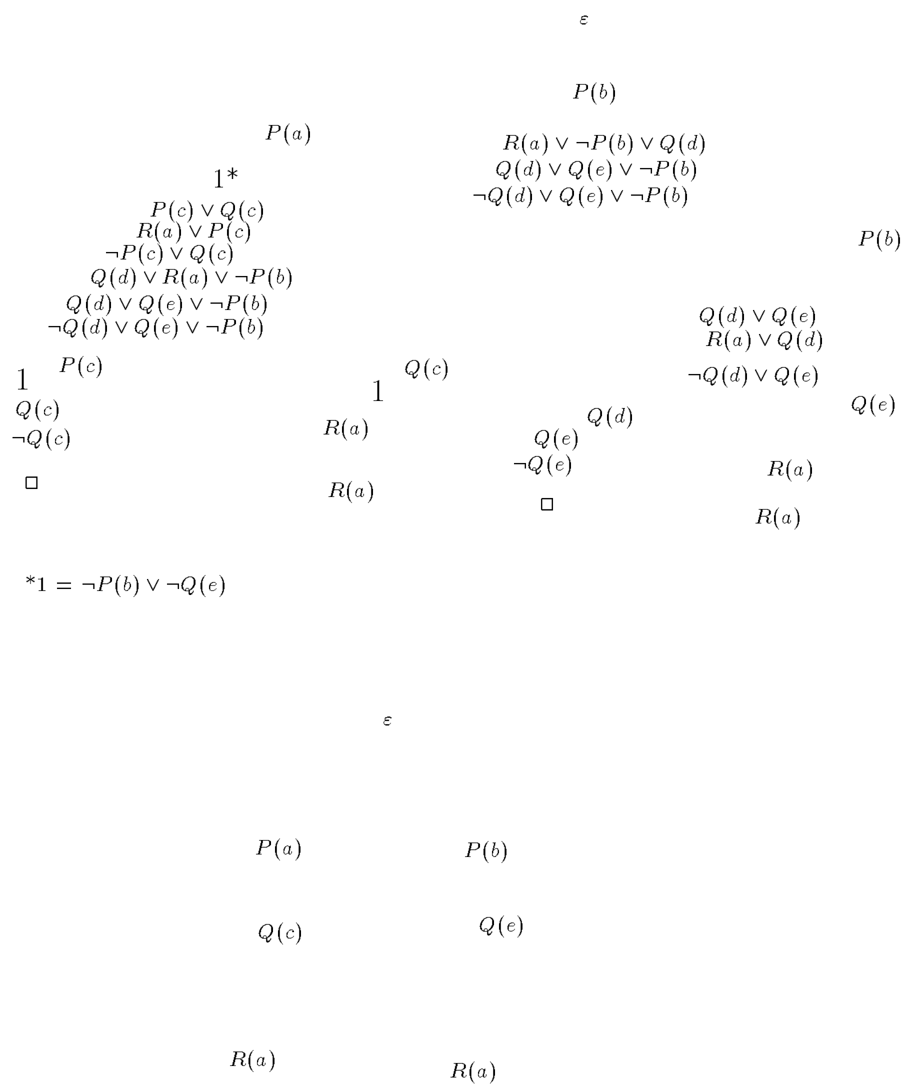


Figure 6: The Perfect Model Tree for Example 6

4 Computing Stable Models

In a disjunctive normal database there are no restrictions on the way negative literals are used. For this class, we have chosen the meaning of a disjunctive normal database, DB, to be given by the stable model semantics.[2]

Definition 4.1 *[9] Let DB be a disjunctive normal database and let I be an interpretation. Then, we define the* Gelfond-Lifschitz (GL) transformation of DB *with respect to I,*

$$\begin{aligned} DB^I = \{(A_1 \vee \cdots \vee A_l \leftarrow B_1, \ldots, B_n) : & \{D_1, \ldots, D_m\} \cap I = \emptyset \textit{ and} \\ & (A_1 \vee \cdots \vee A_l \leftarrow B_1, \ldots, B_n, not\, D_1, \ldots, not\, D_m) \textit{ is a ground instance of } DB\} \end{aligned}$$

where the A_i, B_j and D_k are atomic formulae and I is an interpretation.
M is called a stable model of DB *iff M is a minimal model of DB^M.*

DB^M is a disjunctive deductive database (negation-free) whose semantics is defined by its set of minimal models, $\mathcal{MM}(DB^M)$. Using this declarative definition, to check if an interpretation M is stable is an expensive process, we must guess the interpretation M, construct a new database DB^M, and compute its minimal models. Moreover, non-stratified definite normal databases can have more than one stable model or no stable models at all. Fernández *et al.* [5] proposed a method for reducing the search space (i.e. the interpretations to be tested) and the amount of work necessary to test for stability. They find a set of models that covers the set of stable models, and they reduce the test for stability to a test of consistency with some particular class of integrity constraints.

4.1 Stratified Evidential Transformation

The approach of [5] transforms a disjunctive normal database, DB, into a disjunctive stratified database $DB^{\mathcal{E}}$ for which we must compute the perfect models. This new database $DB^{\mathcal{E}}$ uses an additional set of predicate symbols called *evidences.* For each predicate symbol P of DB we introduce a new predicate symbol $\mathcal{E}P$ whose intended meaning is "there is evidence of P". The role of the "evidence" is to separate the positive use of an atom A in the body of a clause, which has a classical meaning, from the use of its negation (i.e. $not\, A$) where the usage is non-monotonic. Given an atom $A \equiv P(\vec{x})$, we will denote the atom $\mathcal{E}P(\vec{x})$ by $\mathcal{E}A$. Given $M \subseteq HB_{DB}$, $\mathcal{E}M = \{\mathcal{E}A : A \in M\}$.

This role separation allows us to express our clauses in the following classical way:

$$A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_n, \neg\mathcal{E}D_1, \ldots, \neg\mathcal{E}D_m$$

where some A_j is true if the B_i are true and "there is no evidence of the D_l" atoms.

Fernández *et al.* [5] also noticed that, since for disjunctive stratified databases perfect and stable models coincide [16], it was possible to apply their transformation to a stratified database and compute its stable models (i.e. perfect models). However, with the evidential transformation, one may generate a DDDB with many more minimal models than those that are stable (possibly an exponential number of extra models). The reason is that the evidential transformation disregards completely the original structure of the database. This structure is what guides the algorithms

[2]Other semantics have been proposed. See [1, 2, 19] for details.

presented in [7] to generate only perfect models. The solution of [5] was to modify the transformation to take into account the structure of the database. They compute the perfect models of the richer disjunctive stratified database produced by the following transformation.

Definition 4.2 *[5] Let DB be a disjunctive normal database. A* semi-stratification*, $S_1, \ldots, S_r$, of DB is a partition of the set of predicate symbols defined in DB such that if $P \in S_i$ then any predicate Q, on which P depends, belongs to a partition S_j where $j \leq i$ and if P depends negatively on Q then $j < i$ unless Q depends on P.*

Any disjunctive normal database has a *semi-stratification*, and all predicates involved in a recursion through negation will belong to the same semi-stratum. The *stratified evidential transformation* takes a disjunctive semi-stratified database and produces a disjunctive stratified database.

Definition 4.3 *[4, 5] Let DB be disjunctive normal database with semi-stratification $S_1, \ldots, S_r$. The* stratified evidential transformation *of DB defines a disjunctive stratified database $DB^{\mathcal{E}}$ such that:*

1. *For each clause $A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_n, \mathit{not}\, D_1, \ldots, \mathit{not}\, D_m,\ \mathit{not}\, E_1,\ \ldots,\ \mathit{not}\, E_s$ defining predicates in S_i, the clause $A_1 \vee \cdots \vee A_k \vee \mathcal{E}D_1 \vee \cdots \vee \mathcal{E}D_m \leftarrow B_1, \ldots, B_n,\ \mathit{not}\, E_1,\ \ldots,\ \mathit{not}\, E_s$ belongs to $DB^{\mathcal{E}}$ where the predicate symbols of the D_l, are defined in semi-stratum i and the predicate symbols of the E_j, are defined in the semi-strata strictly below i.*

2. *For each predicate symbol $P \in S_i$, the clause $\mathcal{E}P(\vec{x}) \leftarrow P(\vec{x})$ belongs to $DB^{\mathcal{E}}$.*

3. *For each predicate symbol $P \in S_i$, $DB^{\mathcal{E}}$ contains a clause of the form $- \leftarrow \mathcal{E}P(\vec{x}),\ \mathit{not}\, P(\vec{x})$.*

Nothing else belongs to the $DB^{\mathcal{E}}$.

The transformation only substitutes by evidences those occurrences of negated literals that cannot be dealt with by the use of stratification techniques. It only modifies normal semi-strata so that the recursions through negation are changed into an evidence. As for stratification, we can define local semi-stratifications by partitioning the Herbrand base instead of the set of predicate symbols in the obvious way.

Theorem 4.4 *[5] Let DB be a disjunctive normal database. Then $(M \cup \mathcal{E}M)$ is a perfect model of $DB^{\mathcal{E}}$ iff M is a stable model of DB.*

Models that are unstable, will contain the special symbol $-$ denoting that the model is inconsistent with the axiom $\leftarrow -$. Hence, a clause C can be defined to be a logical consequence of DB by the following test on $DB^{\mathcal{E}}$.

Theorem 4.5 *[4] Let DB be a disjunctive normal database, let $DB^{\mathcal{E}}$ be the stratified evidential transformation of DB, and let C be a ground positive clause. Then*

1. *DB is inconsistent iff $-$ is* true *in every perfect model of $DB^{\mathcal{E}}$.*

2. *C is* true *in every stable model of DB iff $(C \vee -)$ is* true *in every perfect model of $DB^{\mathcal{E}}$.*

Hence, we can compute the stable models of DB by computing, using Algorithm 2, the ordered perfect model tree of $DB^{\mathcal{E}}$. The addition of $-$ to the clause C in Theorem 4.5, effectively eliminates from consideration any model of $DB^{\mathcal{E}}$ that does not represent a stable model of DB. Therefore, we could directly eliminate these models from the model tree since they are useless for expressing the semantics of DB.

Note that the unstable perfect models of the transformed theory can be used to eliminate some other models that are not perfect but seem to be stable otherwise. Hence, eliminating unstable models before using them for minimization can result in the generation of models of $DB^{\mathcal{E}}$ that do not correspond to stable models of DB.

Example 7 *Let $DB = \{P(a) \leftarrow \mathit{not}\, P(a)\}$. $DB^{\mathcal{E}} = \{P(a) \vee \mathcal{E}P(a); \mathcal{E}P(a) \leftarrow P(a); - \leftarrow \mathcal{E}P(a), \mathit{not}\, P(a)\}$. The theory has the set of models $\{\{\mathcal{E}P(a), -\}, \{\mathcal{E}P(a), P(a)\}\}$, only the first of which is perfect. However, the perfect model is unstable and therefore should be removed. Note that the nonperfect model seems to be stable since it has no $-$. Eliminating unstable models first will remove the perfect model and leave $\{P(a)\}$ as a stable model for DB which clearly has no stable models.*

Given a general disjunctive normal database, DB, Algorithm 2, operating on the transformed database $DB^{\mathcal{E}}$, constructs the ordered stable model tree for DB. To achieve this, Algorithm 2 exploits the evidential transformation and the ordering of the atoms in the extended Herbrand base $(HB_{DB^{\mathcal{E}}})$ that reflects the induced semi-stratification. Inconsistency resulting from a clause with $-$ in Algorithm 2 signals that the model does not satisfy the integrity constraints introduced by the evidential transformation. However, the relevant parts of such models are retained for use in minimization, a need that was demonstrated in Example 7. The following example demonstrates the use of this approach to construct the ordered stable model tree for a nonstratified normal disjunctive database.

Example 8

$$\begin{array}{ll} \textit{Let } DB = & \{P \leftarrow \mathit{not}\, Q; Q \leftarrow \mathit{not}\, P; R \leftarrow P; R \leftarrow Q\}. \\ DB^{\mathcal{E}} = & \{P \vee \mathcal{E}Q; Q \vee \mathcal{E}P; R \leftarrow P; R \leftarrow Q; \mathcal{E}P \leftarrow P; \mathcal{E}Q \leftarrow Q; \mathcal{E}R \leftarrow R; \\ & - \leftarrow \mathcal{E}P, \mathit{not}\, P; - \leftarrow \mathcal{E}Q, \mathit{not}\, Q; - \leftarrow \mathcal{E}R, \mathit{not}\, R; \leftarrow -\}. \end{array}$$

The ordered stable model tree built using Algorithm 2 on the (stratified) database $DB^{\mathcal{E}}$ is shown in Figure 7.

5 Conclusions and Future Work

In this paper we emphasized the connection between the ordering of the Herbrand base of a disjunctive database and its hierarchical structure. We used this connection to compute the perfect model tree for stratified databases. Combining this with the evidential transformation, which transforms a nonstratified database into a stratified database and a set of integrity constraints, we were able to use the ordering of the extended Herbrand base to compute the stable model tree of a general normal database.

For the perfect and stable model tree generation, the construction proceeds in a sequential manner. To expand a node, the results of expanding nodes to its right are needed. Order and

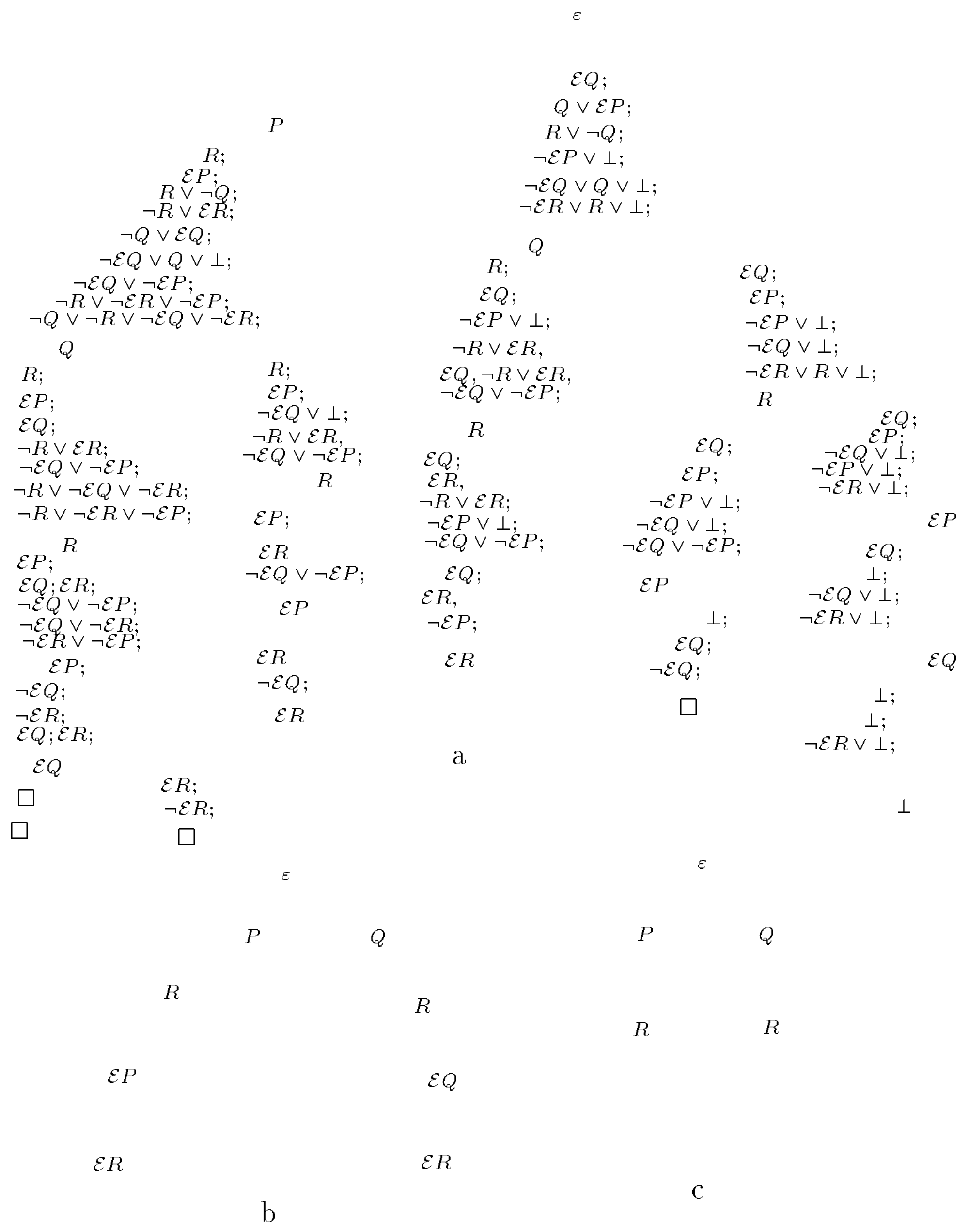

ε
$\mathcal{E}Q;$
$Q \vee \mathcal{E}P;$
$R \vee \neg Q;$
$\neg\mathcal{E}P \vee \bot;$
$\neg\mathcal{E}Q \vee Q \vee \bot;$
$\neg\mathcal{E}R \vee R \vee \bot;$
P
$R;$
$\mathcal{E}P;$
$R \vee \neg Q;$
$\neg R \vee \mathcal{E}R;$
$\neg Q \vee \mathcal{E}Q;$
$\neg\mathcal{E}Q \vee Q \vee \bot;$
$\neg\mathcal{E}Q \vee \neg\mathcal{E}P;$
$\neg R \vee \neg\mathcal{E}R \vee \neg\mathcal{E}P;$
$\neg Q \vee \neg R \vee \neg\mathcal{E}Q \vee \neg\mathcal{E}R;$
Q
$R;$
$\mathcal{E}P;$
$\mathcal{E}Q;$
$\neg R \vee \mathcal{E}R;$
$\neg\mathcal{E}Q \vee \neg\mathcal{E}P;$
$\neg R \vee \neg\mathcal{E}Q \vee \neg\mathcal{E}R;$
$\neg R \vee \neg\mathcal{E}R \vee \neg\mathcal{E}P;$
R
$\mathcal{E}P;$
$\mathcal{E}Q; \mathcal{E}R;$
$\neg\mathcal{E}Q \vee \neg\mathcal{E}P;$
$\neg\mathcal{E}Q \vee \neg\mathcal{E}R;$
$\neg\mathcal{E}R \vee \neg\mathcal{E}P;$
$\mathcal{E}P;$
$\neg\mathcal{E}Q;$
$\neg\mathcal{E}R;$
$\mathcal{E}Q; \mathcal{E}R;$
$\mathcal{E}Q$
$\square$
$\square$
$\mathcal{E}R;$
$\neg\mathcal{E}R;$
$\square$
$R;$
$\mathcal{E}P;$
$\neg\mathcal{E}Q \vee \bot;$
$\neg R \vee \mathcal{E}R,$
$\neg\mathcal{E}Q \vee \neg\mathcal{E}P;$
R
$\mathcal{E}P;$
$\mathcal{E}R$
$\neg\mathcal{E}Q \vee \neg\mathcal{E}P;$
$\mathcal{E}P$
$\mathcal{E}R$
$\neg\mathcal{E}Q;$
$\mathcal{E}R$
Q
$R;$
$\mathcal{E}Q;$
$\neg\mathcal{E}P \vee \bot;$
$\neg R \vee \mathcal{E}R,$
$\mathcal{E}Q, \neg R \vee \mathcal{E}R,$
$\neg\mathcal{E}Q \vee \neg\mathcal{E}P;$
R
$\mathcal{E}Q;$
$\mathcal{E}R,$
$\neg R \vee \mathcal{E}R;$
$\neg\mathcal{E}P \vee \bot;$
$\neg\mathcal{E}Q \vee \neg\mathcal{E}P;$
$\mathcal{E}Q;$
$\mathcal{E}R,$
$\neg\mathcal{E}P;$
$\mathcal{E}R$
$\mathcal{E}Q;$
$\mathcal{E}P;$
$\neg\mathcal{E}P \vee \bot;$
$\neg\mathcal{E}Q \vee \bot;$
$\neg\mathcal{E}R \vee R \vee \bot;$
R
$\mathcal{E}Q;$
$\mathcal{E}P;$
$\neg\mathcal{E}P \vee \bot;$
$\neg\mathcal{E}Q \vee \bot;$
$\neg\mathcal{E}Q \vee \neg\mathcal{E}P;$
$\mathcal{E}P$
$\bot;$
$\mathcal{E}Q;$
$\neg\mathcal{E}Q;$
$\square$
$\mathcal{E}Q;$
$\mathcal{E}P;$
$\neg\mathcal{E}Q \vee \bot;$
$\neg\mathcal{E}P \vee \bot;$
$\neg\mathcal{E}R \vee \bot;$
$\mathcal{E}P$
$\mathcal{E}Q;$
$\bot;$
$\neg\mathcal{E}Q \vee \bot;$
$\neg\mathcal{E}R \vee \bot;$
$\mathcal{E}Q$
$\bot;$
$\bot;$
$\neg\mathcal{E}R \vee \bot;$
$\bot$
a
ε
P
Q
R
R
$\mathcal{E}P$
$\mathcal{E}Q$
$\mathcal{E}R$
$\mathcal{E}R$
b
ε
P
Q
R
R
c

Figure 7: The Stable Model Tree for Example 8

stratification considerations can be used to allow for parallelism and improved efficiency in the tree construction process. When a child is in a different stratum than its parent, the computations under the two nodes can proceed independently since the child's subtree cannot affect the subtrees to the left of the parent node. The two computations can proceed in parallel. On the other hand when two siblings belong to different strata we can abandon the left sibling (the one in the lower stratum) since its models cannot be perfect.

Minimal ordered model trees were used in [20] as a normal form for representing disjunctive deductive databases. In the same way ordered perfect and ordered stable model trees can serve as normal forms to represent the corresponding class of disjunctive databases.

An issue closely related to stratification is the concept of *prioritized circumscription* [12, 13, 14, 15]. In view of the results given in [17] on the equivalence of perfect models and the prioritized circumscription of a given theory, Algorithm 2 can be readily applied to compute the prioritized circumscription of the theory as well.

The trees constructed using Algorithm 2 can be used to answer queries to the database under the relevant semantics. The algorithm presented in [8] can be used for this purpose. The order imposed on the tree can be utilized to improve the efficiency of answer extracting algorithms. These trees can also be used to compute the completions of the database and therefore to answer negative queries.

Other researchers have developed algorithms to compute perfect and stable models of disjunctive normal databases and logic programs [3, 5, 6, 11].

Fernández *et al.* [5, 6], as discussed before, use the same evidential transformation as we do. The major difference is in the way minimization is achieved. Their algorithms expand the model tree eagerly when a rule is evaluated. Once this is done a minimization step is applied, where all branches of the tree are compared against each other and non-minimal branches are eliminated. The cost of this process is polynomial in the total number of models generated. On the other hand, the cost of maintaining only minimal models in our algorithms is polynomial in the number of minimal models obtained.

Bell, Nerode, Ng and Subrahmanian [3] developed an algorithm for computing stable models of normal databases that uses *linear programming techniques* at the core of the approach. Ground clauses are translated into constraints that are evaluated to compute the set of minimal models of the database (i.e. minimal solutions to the linear problem). Stable models are then computed by applying the GL-transformation for each minimal model and solving the corresponding new linear problem. In contrast, we integrate the generation of models with the verification of their stability. Moreover, using semi-stratification, we are able to restrict the number of partial models being tested for stability to the perfect models of the transformed database rather than the entire set of minimal models as in the case of [3].

Inoue *et al.* [11] developed a different transformation to use in the computation of stable models. Using this transformation they have implemented a parallel algorithm to compute the stable models of a disjunctive normal database. Their approach requires the generation of the set of all constructive models of the transformed theory. The minimal elements of this set have to be checked for stability, as in the case of [5] and [3]. Additionally, Inoue's transformation does not take into consideration any hierarchical structure existent in the database. This results in the introduction of additional literals and rules, when compared with the stratified evidential transformation introduced by Fernández *et al.* [5]. This represents a substantial increase in the size of the database. Given the way models are screened for stability, the same comment made concerning [3] applies. An added advantage of our algorithm is that the first model generated is known to be stable. This can be of help in cases when a single model of the theory is needed. There is no need to generate all the minimal models of the theory to find a single stable model.

The approaches of [5, 6] and [11] are incremental in the sense that they process the clauses of the database one at a time. Our approach is based on processing atoms, rather than clauses. All clauses containing the atom under consideration are processed simultaneously. Together with the order imposed on atom expansion, this can help in pruning irrelevant models and consequently faster termination of the tree construction process.

The algorithms presented in this paper were implemented in Quintus Prolog. The examples of this paper were verified using this implementation. Further experimentation is planned. A topic for further research is the study of additional criteria that can be used to impose a total order on the elements of the Herbrand base and the possible effects of the resulting ordering on the tree construction process and on the structure of the resulting tree. An experimental analysis of the performance of the tree construction algorithm under various ordering criteria is planned.

Acknowledgements

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

References

[1] C. Baral, J. Lobo, and J. Minker. Non-monotonic reasoning and generalised disjunctive well-founded semantics. under preparation.

[2] Chitta Baral, Jorge Lobo, and Jack Minker. $WF^{\underline{3}}$ semantics for negation in normal and disjunctive logic programs. Draft, 1990.

[3] C. Bell, A. Nerode, R. Ng, and V.S. Subrahmanian. Computation and implementation of non-monotonic deductive databases. Technical Report CS-TR-2801, University of Maryland, 1991. Submitted for journal publication.

[4] José Alberto Fernández and Jorge Lobo. A proof procedure for stable theories. Technical Report UMIACS–TR–93–14 and CS–TR–3034, University of Maryland Institute for Advance Computer Studies, College Park, MD 20742, 1993. Submitted to the Journal of Logic Programming.

[5] José Alberto Fernández, Jorge Lobo, Jack Minker, and V.S. Subrahmanian. Disjunctive LP + integrity constraints = stable model semantics. *Annals of Mathematics and Artificial Intelli-*

gence, 1993. To appear. Preliminary version presented at the Second International Symposium on Artificial Intelligence and Mathematics, Florida, 1992.

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

[7] José Alberto Fernández and Jack Minker. Computing perfect models of disjunctive stratified databases. In Don Loveland, Jorge Lobo, and Arcot Rajasekar, editors, *Proceedings of the ILPS'91 Workshop on Disjunctive Logic Programs*, pages 110–117, San Diego, California, October 1991. An extended version has been submitted to the Journal of Logic Programming.

[8] José Alberto Fernández and Jack Minker. Semantics of disjunctive deductive databases. In *Proceedings of the International Conference on Database Theory*, pages 332–356, 1992. Invited Paper.

[9] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In R.A. Kowalski and K.A. Bowen, editors, *Proc. 5^{th} International Conference and Symposium on Logic Programming*, pages 1070–1080, Seattle, Washington, August 15-19 1988.

[10] M. Gelfond, H. Przymusinska, and T.C. Przymusinski. On the Relationship between Circumscription and Negation as Failure. *Artificial Intelligence*, 38:75–94, 1989.

[11] Katsumi Inoue, Miyuki Koshimura, and Ryuzo 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] V. Lifschitz. Closed world databases and circumscription. *Artificial Intelligence*, 27(2):229–235, November 1985.

[13] V. Lifschitz. Computing circumscription. *Proc. Ninth International Joint Conference on Artificial Intelligence*, pages 121–127, 1985. Morgan Kaufman Publishers, Inc.

[14] V. Lifschitz. On the declarative semantics of logic programs with negation. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann, 1988.

[15] J. McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1 and 2):27–39, 1980.

[16] T. C. Przymusinski. Stable semantics for disjunctive programs. *New Generation Computing Journal*, 9:401–424, 1991. Extended Abstract appeared in [18].

[17] Teodor C. Przymusinski. On the declarative semantics of deductive databases and logic programming. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 5, pages 193–216. Morgan Kaufmann Pub., Washington, D.C., 1988.

[18] Teodor C. Przymusinski. Extended stable semantics for normal and disjunctive programs. In Warren and Szeredi, editors, *Proceedings of the 7^{th} International Logic Programming Conference*, pages 459–477, Jerusalem, 1990. MIT Press. Extended Abstract.

[19] A. Van Gelder, K. Ross, and J.S. Schlipf. Unfounded Sets and Well-founded Semantics for General Logic Programs. In *Proc. 7^{th} Symposium on Principles of Database Systems*, pages 221–230, 1988.

[20] A. Yahya, J. A. Fernandez, and J. Minker. Ordered model trees: a normal form for disjunctive deductive databases. Technical Report UMIACS–TR–93–63 and CS–TR–3103, University of Maryland Institute for Advance Computer Studies, College Park, MD 20742, 1993. Submitted to the Journal of Automated Reasoning.