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

LMU
Ludwig-
Maximilians-
Universität
München

Model Generation in Disjunctive Normal Databases

Adnan H. Yahya

http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1996-10, June 1996

Model Generation in Disjunctive Normal Databases

Adnan Yahya
Computer Science Department, University of Munich, Munich, Germany
Electrical Engineering Department, Birzeit University, Birzeit, Palestine
yahya@informatik.uni-muenchen.de

Abstract

Algorithms for computing several classes of models for disjunctive normal databases are presented. We show how to efficiently compute minimal, restricted minimal, perfect, and stable models. The common feature of the advanced algorithms is that they are based on augmenting a model generating procedure with a set of hypotheses to guide its search for acceptable models and/or to interpret negation in clause bodies. The approach is shown to be useful for different database applications including query answering under different semantics and integrity constraint enforcement. The developed algorithms are easy to implement and compare favorably with others advanced in the literature for the same purpose.

1 Introduction

Much attention has been devoted to computing models for disjunctive databases as a tool for data storage and manipulation. Several algorithms were suggested for computing different classes of models that can also be utilized for query answering and integrity constraint enforcement. It is often the case that the user has some knowledge or assumptions about the current state of the database that need to be verified. The source of this information may be the database itself or an external agent. Some examples are:

- Given an interpretation I determine if I is a (minimal) model of the database.
- Given a model M of DB find a minimal component of M that is a minimal model of DB or find the entire set of minimal models of DB that are subsets of M;
- Given a (minimal) model M of DB determine if M satisfies a set of clauses external to DB (e.g. a set of integrity constraints for DB).
- Given a (minimal) model M of DB determine if M belongs to the models of DB under a particular semantics (e.g. minimal, perfect, stable, ...).

While many other questions may be posed, we limit ourselves to checks performed on a subset of the Herbrand base. Each such set can be treated as an interpretation of the theory. We show that being able to test hypotheses about such sets of atoms can be utilized in many useful applications.

In particular, we show that such tests can be converted into a procedure for computing answers under different database semantics and for integrity constraint enforcement, among other possible uses.

One can think of several sources for the set of atoms being checked: the previous state of the database represented by the set of (minimal) models: when we need to check if they are still (minimal) models of the updated theory; a generic (minimal) model generating procedure for the database or one of its transformations (e.g. the positive database corresponding to a database with negated body literals- a normal disjunctive database) and restrictions on the (local) Herbrand base due to partitioning.

We consider cases when the set of hypotheses is kept constant during the reasoning process or is allowed to change to reflect the progress of the computation.

The rest of the paper is organized as follows. In the next section we give the basic notation and background material and point to the problems associated with computing perfect and stable models using conventional model generation. In Section 3 we develop a procedure for computing restricted models and show how to use it to check for model minimality. In Section 4 we show how to compute the set of *perfect* models of a stratified database. In Section 5 we outline a procedure for computing *stable* models for general normal databases. In Section 6 we show the potential utility of the developed procedures for various database applications, compare our approach with others reported in the literature and point to some possible directions of further research.

2 Notation and Background:

In the following we assume familiarity with the basic concepts of disjunctive logic programming as in [12]. We present only the most relevant definitions in a condensed form.

2.1 General:

Definition 2.1 *A* disjunctive database DB *is a set of clauses of the form:*

$$C = A_1 \vee \cdots \vee A_m \leftarrow B_1, \ldots, B_n, notD_1, \ldots, notD_k,$$

where $k, m, n \geq 0$ *and the As, Bs and Ds are atoms in a First Order Language (FOL)* $\mathcal{L}$ *with no function symbols and "not" is the default negation operator.* DB *is: A disjunctive deductive database (DDDB) if* k *is always = 0 and is a disjunctive normal database (DNDB) otherwise.*

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*[1] 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. We also employ two additional atoms: $\top$ to represent the atom true in all interpretations and $\perp$ to represent the atom false in all interpretations. To maintain the clause implication form (head-body representation of clauses) we assume that $\perp$ is the head of a clause with an empty head and $\top$ is the body of a clause

[1] As is common in the field, we identify the interpretation by the set of ground atoms assigned *true* in that interpretation. All other atoms of the Herbrand Base are assigned *false*.

with an empty body. Clearly all models (implicitly) have the atom $\top$ and none can have $\bot$. By $Head(C)$ ($Body(C)$) we denote the set of literals in the head (body) of clause C.

Definition 2.2 (**positive transformation**) *Given a disjunctive database DB we define its positive transformation as the DDDB DB^+ such that:*

$$DB^+ = \{A_1 \vee \cdots \vee A_m \vee D_1 \ldots \vee D_k, \leftarrow B_1, \ldots, B_n | A_1 \vee \cdots \vee A_m \leftarrow B_1, \ldots, B_n, not D_1, \ldots, not D_k \in DB\}$$

Definition 2.3 *A clause C is range restricted (safe) if every variable occurring in the head (a negated atom of the body) of C also appears in a positive atom of the body of C. A database is range restricted (safe) iff all its clauses are range restricted (safe).*

Clearly, the positive transformation of a range restricted and safe normal database is a range restricted DDDB.

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

1. *The* extensional part *(E_{DB}) a positive disjunctive database corresponding to base relations.*
2. *The* intensional part *(I_{DB}) corresponding to view definitions The rules of I_{DB} can be used to derive new pieces of information from the extensional part of the database.*
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).*

The following is an extension of the concept of *supported* interpretation/model [1, 20] to the disjunctive case:

Definition 2.5 (**supported interpretation**) *[1, 20] An interpretation I of a disjunctive normal database, DB, is supported if for each atom $A \in I$ there is a ground instance of a clause in DB, say C, such that the body of C is true in I and $A \in Head(C)$.*

A supported interpretation that satisfies all clauses of DB is a supported model of DB.

Definition 2.6 *If $C = A_1 \vee ... \vee A_n$ is a disjunction of atoms, then by $Neg(C)$ we denote the finite set of clauses in implication form $Neg(C) := \{A_1 \rightarrow \bot, ..., A_n \rightarrow \bot\}$. On the other hand if $M = \{A_1, ..., A_n\}$ is a finite model (interpretation) then $Neg(M)$ denotes the set of clauses in implication form $Neg(M) := \{A_1 \wedge ... \wedge A_n \rightarrow \bot\}$.*

2.2 Stratification and Perfect Models:

Processing DNDB is complicated by the negations in their clause bodies. However, for the special class of stratified and locally stratified databases negation is well behaved. The semantics of these databases can be defined by their set of perfect models. This is a subset of the set of the minimal models of the theory defined to reflect the hierarchical structure of (locally) stratified databases.

Definition 2.7 (**stratification**) *[18] A disjunctive normal database DB is stratified (locally stratified) if there is a level mapping of its predicates (ground atoms) to nonnegative integers so that for each clause as in Definition 2.1 above: $L(A_i) = L(A_j)$, $L(A_i) \leq L(B_j)$ and $L(A_i) > L(D_j)$, where $L(A)$ is the level or stratum of the predicate of atom A (of ground atom A).*

Given a stratified (locally stratified) database, DB, we can talk about the stratification (local stratification) of DB as $\mathcal{S} = \{S_1, ...S_n\}$, where S_i is the set of predicates (ground atoms) in stratum i or as $\{DB_1, ...DB_n\}$, where DB_i is the set of clauses with heads in stratum (local stratum) i. That is, DB_i comprises the clauses defining elements of S_i. Elements of the lowest (first) stratum are always elements of EDB. Note that $\{DB_1, ..., DB_n\}$ is sufficient to specify the theory and its stratification.

Definition 2.8 (**perfect models**) *Let DB be a disjunctive normal database with levels $\{1, ..., n\}$. M is a perfect model of DB if and only if for all $i \leq n$ the set of atoms of level i in M is minimal among all models which agree with M on all atoms of level $j < i$.*

The set of perfect models of a DNDB, DB, is denoted by $Perfect(DB)$. It was shown that computing $Perfect(DB)$ can be done by combining the results obtained from generating the minimal models for individual strata of DB in the order $1, ..., n$ [7].

2.3 Stable Model Semantics:

Stable models are a subset of the minimal models of the database and are defined using the following transformation called the Gelfond-Lifschitz (G-L) transformation.

Definition 2.9 (**G-L transformation**) *[8] Given a DNDB DB and an interpretation I*

$$\begin{aligned} DB^I = \{(A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_n)\theta : \theta \textit{ is ground and} \\ (A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_n, notD_1, \ldots, notD_m) \in DB \\ \textit{and } \{D_1\theta, \ldots, D_m\theta\} \cap I = \emptyset\}. \end{aligned}$$

DB^I is the Gelfond-Lifschitz *(GL) transformation of DB with respect to I, where the As, Bs and Ds are atomic formulae. Note that DB^I is a DDDB.*

Definition 2.10 (**stable models**) *[17] Let DB be a disjunctive normal database. M is a stable model of DB iff M is a minimal model of DB^M.*

The set of stable models of a DNDB, DB, is denoted by $Stable(DB)$. For stratified databases stable and perfect models are the same. For DDDBs stable, perfect and minimal models are the same.

2.4 Model Generation:

The main results of this paper are based on using model generating procedures with certain properties [3, 21]. We extensively utilize denial clauses (rules with empty heads) to restrict the search space for models. Unless explicitly stated otherwise we assume that denial clauses have only positive body literals and therefore represent purely negative clauses.

Definition 2.11 *(***model trees:***) Let DB be a database with the Herbrand base HB_{DB}.* A model tree *for DB is a tree structure where*

- *The root is labeled by the special atom $\top$.*
- *Other nodes are labeled with atoms of HB_{DB} or disjunctions of atoms.*
- *A path from the root to a leaf node is called a* branch
- *No unit clause labels more than one node in a branch.*
- *If $\mathcal{B}$ is a branch of the tree then $Units(\mathcal{B})$ denotes the set of positive unit clauses of $\mathcal{B}$.*
- *If $\mathcal{B}$ is a branch of the tree then $M = Units(\mathcal{B})$ is a model for DB.*

Lemma 1 *Let I be an interpretation and let $\mathcal{C}$ be a set of denial rules. Then if $\mathcal{C}$ is violated in I then it is also violated in all supersets of I. If $\mathcal{C}$ is satisfied in I then it is also satisfied in all subsets of I.*

Proof: Straight forward. ∎

As a counterexample for the case of nondenial constraints consider the single rule $a \rightarrow b$ and the interpretations $\{a\}, \{a, b\}$. Only $\{a, b\}$ satisfies the constraint and $\{a\} \subset \{a, b\}$.

Theorem 1 *Let DB be a DDDB and let $\mathcal{C}$ be a set of denial constraints. Then:*

- *If M is a model for $DB \cup \mathcal{C}$ then M is a model for DB alone: (If $M \models (DB \cup \mathcal{C})$ then $M \models (DB)$).*
- *If M is a model for DB such that for all $C_i \in \mathcal{C}$, $M \cap Body(C_i) \neq Body(C_i)$ then M is a model for $DB \cup \mathcal{C}$:*
- *The model tree for $DB \cup \mathcal{C}$ is the model tree for DB except that branches firing an element of $\mathcal{C}$ are deleted.*

Proof:

- Straightforward since $DB \subset DB \cup \mathcal{C}$.
- M satisfies all elements of $\mathcal{C}$ since their bodies are always falsified (not contained entirely in M).
- Branches of the model tree for DB firing no clause of $\mathcal{C}$ are reproduced in model generation for $DB \cup \mathcal{C}$. While for others, firing a denial constraint will close the branch resulting in its removal.

∎

Note that if $\mathcal{C}$ includes nondenial rules then there can be minimal models of $DB \cup \mathcal{C}$ that are not minimal for $\mathcal{C}$ alone. This is the case in the just given example.

Definition 2.12 *Given a database, 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* DB: $\mathcal{P}(DB) \subseteq \mathcal{MM}(DB)$.
3. *Complete: if it returns all the minimal models of* DB: $\mathcal{MM}(DB) \subseteq \mathcal{P}(DB)$.

Corollary 1 *Let* DB *be a DDDB,* $\mathcal{C}$ *be a set of denial rules and* $\mathcal{P}$ *be a model generating procedure*

1. $\mathcal{P}(DB \cup \mathcal{C}) \subseteq \mathcal{P}(DB)$.
2. $\mathcal{MM}(DB \cup \mathcal{C}) \subseteq \mathcal{MM}(DB)$.
3. *If* $\mathcal{P}$ *is complete then all minimal models of* DB *satisfying the constraints in* $\mathcal{C}$ *are returned by* $\mathcal{P}(DB \cup \mathcal{C})$.

Proof: Straight forward. ■

Next we give a brief description of successively refined model generating procedures that are sound and complete [3]. 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 2.13 (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:

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

Splitting Rule:

$$\begin{array}{ccc} & E_1 \vee E_2 & \\ \hline E_1 & \mid & E_2 \end{array}$$

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 our theory is range restricted. The head is always ground when the body is ground (or empty).

Definition 2.14 (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. $\{\top\}$ *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 2.15 **(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 *false* (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 [3].

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

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

Figure 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 [3].

Definition 2.16 **(complement splitting rule)**

$$\frac{E_1 \vee E_2}{\begin{array}{lcr} E_1 & | & E_2 \\ Neg(E_2) & | & \end{array}}$$

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.

[2] These conventions are adopted in the implementation reported in [3]. They correspond to a left-to-right, depth-first traversal of the seach 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.

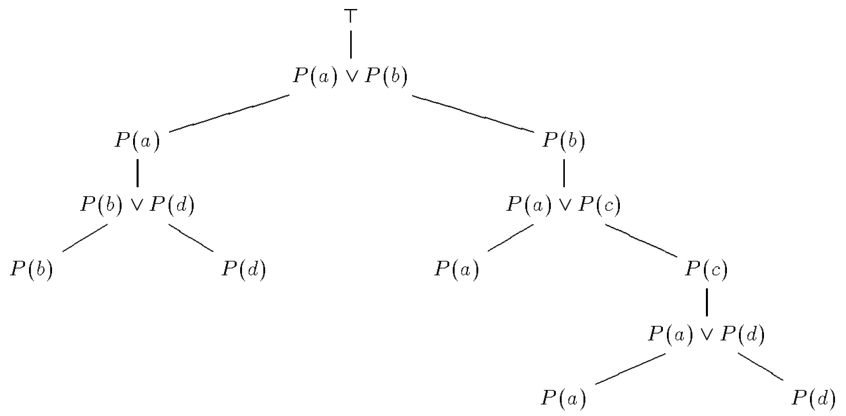

Figure 1: A Model Tree for Example 1 (with nonminimal and duplicate models).

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

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

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, ($Neg(M)$), 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.:*

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

Note that all models returned by the procedure are minimal.

In [21] and [3] sound and complete minimal model generation procedures were given for ground and RR theories, respectively. In [5] it was shown how to modify such a procedure to compute the perfect models of a stratified database and stable models for general DNDBs. [3] 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 [13], CS-Satchmo for the implementation with complement splitting and MM-Satchmo for the implementation with model minimization (by including negation of generated minimal models). All through this paper we assume that the DDDB under consideration is range restricted and has only finite models.

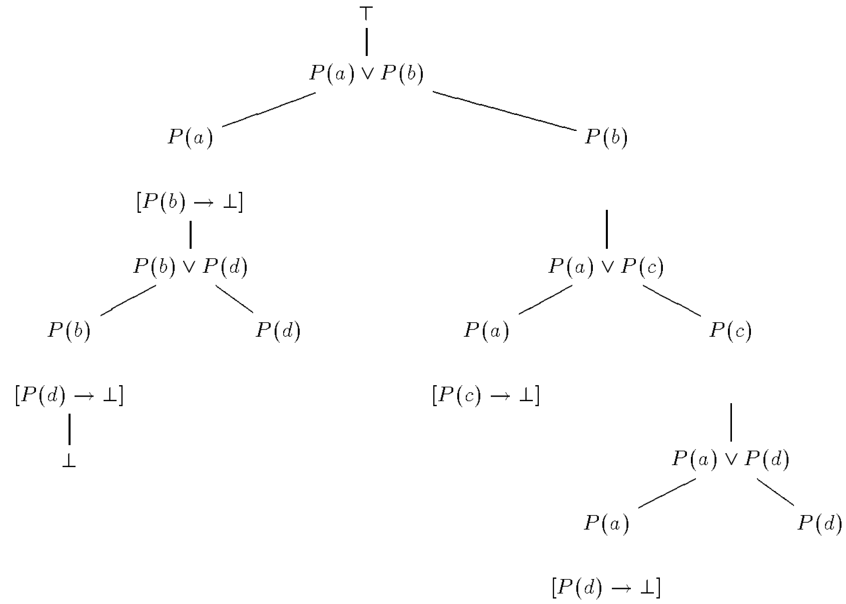

Figure 2: The Model Tree with Complement Splitting for Example 2.

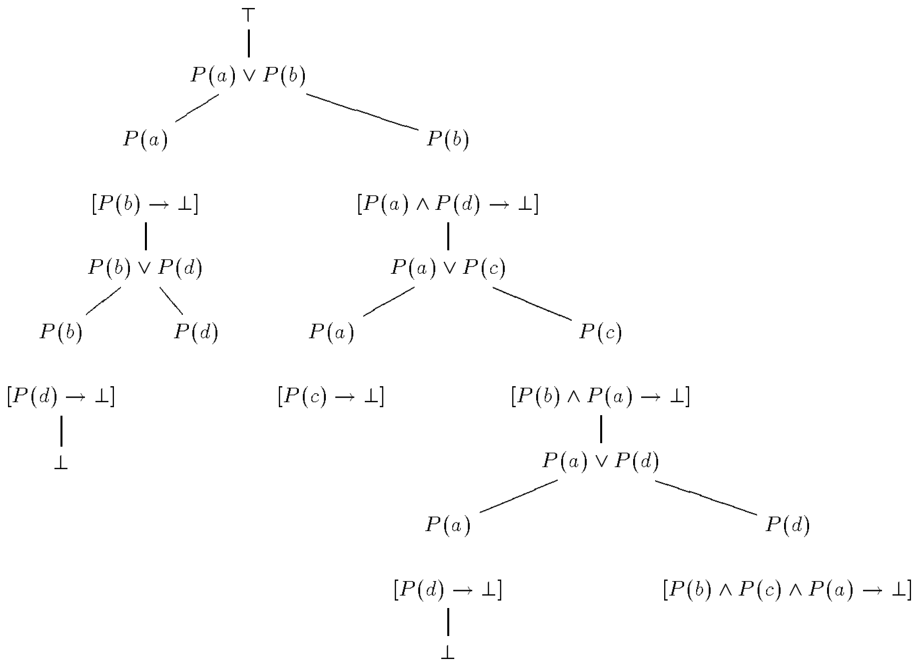

Figure 3: A Run of the Minimal Model Generation Procedure MM-Satchmo for Example 3.

2.5 Model Generation and Negation:

In this paper we are concerned with model computations for disjunctive databases in which negative atoms appear only in rule bodies (normal disjunctive databases). The natural way to extend our model generating procedure to the case of DNDB is to replace the PUHR-rule used for deriving new clauses to account for body negation:

Definition 2.17 (UHR-expansion rules) *Let DB be a DNDB and $\mathcal{B}$ be a root-to-leaf branch of the model tree being constructed. UHR-rule adds a new (leaf) node to branch $\mathcal{B}$ with label E_σ, where σ is a most general unifier of the body of a clause $(not D_1 \wedge ... \wedge not D_l \wedge A_1 \wedge ... \wedge A_m \rightarrow E) \in DB$ such that $\{A_1, ..., A_m\}\sigma = \{B_1, ..., B_n\}$, $\{B_1, ..., B_n\} \subseteq Units(\mathcal{B})$ and $\{D_1, ..., D_l\}\sigma \cap Units(\mathcal{B}) = \emptyset$.*

Note that the UHR-rule reduces to the PUHR-rule for DDDBs. The splitting rule is not changed and it is always applied to *ground* disjunctions. The safety condition guarantees that negated body atoms are also ground when the rule is applied.

The modification fits the practice of defining interpretations through their positive (*true*) atoms and interpreting all atoms not explicitly mentioned in the interpretation as *false*. The implementation of all model generating procedures reported in [3] simulate this UHR-rule when applied to a DNDB.

However, even with this modification, the model generation procedures developed for the DDDB case are not readily applicable to normal databases. The problem with model computations in disjunctive normal databases is the treatment of negated atoms in rule bodies. Negative literals may get treated on the same level as positive literals, without taking into account the nonmonotonic nature of default negation. Having a negative literal satisfied in the current (intermediate, still developing) interpretation, by the virtue of its absence from the branch, doesn't guarantee that the respective atom will not be asserted at a later stage, thus invalidating earlier inferences. Since our procedure retains only the head of the clause for further processing, retraction becomes a problem. Consider the following example:

Example 4 *Let $DB_1 = \{not P(a) \rightarrow P(b), P(a)\}$. Starting from $I_1 = \{\}$ and treating the first clause first will produce $I_2 = \{P(b)\}$. Now treating the second clause will produce $I_3 = \{P(b), P(a)\}$. While I_3 is a model of DB_1 it is not minimal. Additionally, the grounds for deriving $P(b)$ (having $P(a)$ false) are no more in I_3 although they existed in I_1. I_3 is the only model returned by $MM-Satchmo$.*

Let $DB_2 = \{not P(a) \rightarrow Q(b), S(c), P(a) \rightarrow R(d), S(c) \rightarrow P(a)\}$. The only model generated is $M_1 = \{S(c), Q(b), P(a), R(d)\}$ which is not minimal. The only minimal model of DB_2 is $M_2 = \{S(c), P(a), R(d)\}$.

That is, the procedure employing the UHR-rule is not complete for the class of DNDBs[3].

Using the UHR-expansion rule for a DNDB may still suppress certain minimal models of the input theory. The reason for this behavior is that the UHR-rule asserts the head of a clause when its body is satisfied as a way of satisfying the entire clause. For negation free clauses this is not problematic

[3]Note that completeness is important for refutational soundness of a model generating procedure and such applications as query answering [3]. Additionally, we are interested in generated all minimal models since the set of stable models is a subset of the set of minimal models for a DNDB. Note that in Example 4 M_2 is stable for DB but is not returned.

since once the body of a clause is satisfied in a branch it remains so when the branch expands by adding positive atoms. Asserting the head is the only way to satisfy the clause. For clauses with negation things are different. Subsequent atom additions to a branch may falsify the body of the clause by asserting atoms corresponding to its negative body literals. One way to go around this problem is to modify the UHR-expansion rule so that it makes it possible to expand branches so that take into account the possibility of the clause body becoming *false* by future additions of atoms. That is we modify the UHR-rule given in Definition 2.17 as follows:

Definition 2.18 (Complete UHR-expansion rule) *Let DB be a DNDB and $\mathcal{B}$ be a root-to-leaf branch of the model tree being constructed. The Complete UHR-rule adds a new (leaf) node to branch $\mathcal{B}$ with label $(E \vee D_1 \vee ... \vee D_l)\sigma$, where σ is a most general unifier of the body of a clause*[4] $(notD_1 \wedge ... \wedge notD_l \wedge A_1 \wedge ... \wedge A_m \rightarrow E) \in DB$ *such that* $\{A_1, ..., A_m\}\sigma = \{B_1, ..., B_n\}$, $\{B_1, ..., B_n\} \subseteq Units(\mathcal{B})$ *and* $\{D_1, ..., D_l\}\sigma \cap Units(\mathcal{B}) = \emptyset$.

UHR-expansion Rule:	Complete UHR-expansion Rule:
Branch $\mathcal{B}$	Branch $\mathcal{B}$

$$\begin{array}{c} B_1 \\ \vdots \\ B_n \\ \hline E\sigma \end{array} \qquad\qquad \begin{array}{c} B_1 \\ \vdots \\ B_n \\ \hline (E \vee D_1 \vee ... \vee D_l)\sigma \end{array}$$

where σ is an MGU of the body of a clause $(notD_1 \wedge ... \wedge notD_l \wedge A_1 \wedge ... \wedge A_m \rightarrow E) \in DB$ such that $\{A_1, ..., A_m\}\sigma = \{B_1, ..., B_n\}$, $\{B_1, ..., B_n\} \subseteq Units(\mathcal{B})$ and $\{D_1, ..., D_l\}\sigma \cap Units(\mathcal{B}) = \emptyset$.

We have the immediate following result:

Theorem 2 *Let DB be a DNDB and DB^+ its positive transformation. Consistently applying the Complete UHR-expansion rule to DB is equivalent to applying the PUHR-expansion rule to DB^+.*

Proof: By definition of the Complete UHR-expansion rule and the PUHR-expansion rule applied to $C = (notD_1 \wedge ... \wedge notD_l \wedge A_1 \wedge ... \wedge A_m \rightarrow E) \in DB$ and $C^+ = A_1 \wedge ... \wedge A_m \rightarrow E \vee D_1 \vee ... \vee D_l$, respectively they both add the same clause: $(E \vee D_1 \vee ... \vee D_l)\sigma$. Both rules are activated in exactly the same branches $\{\mathcal{B}$ such that the $\{A_1, ..., A_m\}\sigma = \{B_1, ..., B_n\}$, $\{D_1, ..., D_l\}\sigma \cap Units(\mathcal{B}) = \emptyset$ and $\{B_1, ..., B_n\} \subseteq Units(\mathcal{B})\}$ Other branches will disable both C abd C^+ either by falsifying their bodies or satisfying their heads. ∎

Clearly, if the PUHR-expansion rule in a model generating procedure that is complete for DDDBs is replaced by the Complete UHR-expansion rule then the modified procedure will be minimal model complete for the class of DNDBs.

[4] Note that we select this order of atoms in the resultant clause so that an algorithm employing Complement Splitting will first consider cases corresponding to *stable* instances of the clause: those in which $E\sigma$ is *true* and the $D\sigma$s are all *false*. The hope is that such a choice will facilitate the early genration of stable models.

Therefore, instead of modifying the normal database to get its positive transformation to compute the entire set of minimal models one can modify the procedure to operate with the DNDB and still retain the completeness of the procedure.

Definition 2.19 (**set for resolving negation**) *Given a DNDB, DB, $\mathcal{RN} \subseteq HB_{DB}$ is a set for resolving negation for DB if negative atom occurrences in the body of a clause of DB are all interpreted according to $\mathcal{RN}$. That is, a body occurrence of (notA), has the truth value* false *if $A \in \mathcal{RN}$ and* true *otherwise.*

$\mathcal{RN}$ is not necessarily used to assign truth values to positive occurrences of body atoms or head atoms. This is so since the $\mathcal{RN}$ may be *provisional* in the sense that we may want to retract some of the assignments for elements of $\mathcal{RN}$ in the model to be generated.

Theorem 3 *Let DB be a DNDB and $\mathcal{RN} \subseteq HB_{DB}$ be a set for resolving negation. Let $\mathcal{P}$ be a complete model generating procedure and $\mathcal{P}(DB, \mathcal{RN})$ be the operation of $\mathcal{P}$ on DB with negative body literals always interpreted according to $\mathcal{RN}$. $\mathcal{P}(DB, \mathcal{RN})$ returns all the minimal models of $DB^{\mathcal{RN}}$, among other models of $DB^{\mathcal{RN}}$ ($\mathcal{P}$ is sound and complete for $DB^{\mathcal{RN}}$).*

Proof: $\mathcal{RN}$ is the set used to interpret negative body literals of DB. Lat $C \in DB$ be a clause and $notA$ be in $Body(C)$. If $A \in \mathcal{RN}$ then C is satisfied (by the falsity of its body). It never fires and has no contribution to the model generation process. $A \notin \mathcal{RN}$ then $notA$ can be removed from the body of C. This is equivalent to the G-L transformation of C. Applying it to all clauses of DB wil have the effect of working with $DB^{\mathcal{RN}}$. Since the model generating procedure is complete it will return all the minimal models of $DB^{\mathcal{RN}}$.

Clearly if $\mathcal{P}$ is minimal model sound and complete then it returns exactly the set of minimal models of $DB^{\mathcal{RN}}$, $\mathcal{MM}(DB^{\mathcal{RN}})$. ∎

Note that neither in the definition of $\mathcal{RN}$ nor in Theorem 3 do we *require* $DB^{\mathcal{RN}}$ to change even when additional atoms are computed. Recall also that $DB^{\mathcal{RN}}$ is a DDDB and has no negation in the bodies of its rules. In the remainder of this paper we always assume that the model generating procedure is defined as outlined above, and detailed in [3]. The plain reference to a model generating procedure $\mathcal{P}$ assumes nothing about the type of splitting rule employed or model minimization. We use the prefix "CS" to emphasize the use of complement splitting and "MM" presence of a model minimization component of the procedure.

Next we show how to utilize $\mathcal{RN}$ to compute certain classes of models for disjunctive databases.

3 Restricted Model Generation:

In this section we consider two problems: Given a set of ground atoms I and a disjunctive theory DB, find all the minimal models of DB that are subsets of I, if any; (find the set $\{M | M \in \mathcal{MM}(DB) \textit{ and } M \subseteq I\}$) and a special case of that when the given interpretation is a model that needs to be checked for minimality.

One may solve this problem by generating all the minimal models of the theory then check for those included in I [3]. This may be too excessive. It is desirable to localize the decision: to decide model minimality based on the content of I alone without generating unneeded models. We can then employ such an approach for minimality checking in a model generating procedure [3, 15]. Our approach is to constrain the (minimal) model generating procedure so that it returns the set of minimal models contained in I alone. The details are outlined below.

3.1 Generating Restricted Models:

We run our model generating procedure $\mathcal{P}$ and restrict it to expanding atoms in I. This is equivalent to augmenting DB with the negations of all the atoms that are not in I. That is, having $\mathcal{P}$ operate on $(DB \cup \{A \rightarrow \perp,\ for\ all\ A \notin I\})$ rather than on DB alone. The use of an atom not in I will cause a branch to close (can't be a model). If the procedure returns models then these are models of DB contained in I. If $\mathcal{P}$ is complete, the returned set is empty then I contains no models of DB.

To achieve this goal an additional termination condition is added to $\mathcal{P}$: *fail* when there are no atoms of I to expand and some clauses are still not satisfied. If this situation is reached then a termination with failure will be reported on that expansion branch. Note, however, that we don't exclude the use of the negation of atoms not in I during the model construction process (for restricting the search space as in e.g. CS-$\mathcal{P}$).

Definition 3.1 *Let $\mathcal{P}$ be a model generating procedure. By $\mathcal{P}(DB)|_I$ we denote the result returned by $\mathcal{P}$ operating on a disjunctive theory DB with atom expansion restricted to I. Only atoms of I are selected for expansion and a branch terminates with failure (closes) if an expansion is needed but not possible on a atom of I.*

Theorem 4 *Let I be a subset of the Herbrand Base of a DDDB, DB, and $\mathcal{P}$ be a model generating procedure. $\mathcal{P}(DB)|_I$ returns exactly the set $\{M : M \in \mathcal{P}(DB)$ and $M \subseteq I\}$.*

Proof:

- $\mathcal{P}(DB)|_I$ is $\mathcal{P}(DB \cup \mathcal{C})$, where $\mathcal{C} = \{A \rightarrow \perp,\ for\ all\ A \notin I\}$. By Theorem 1 any model returned is also a model for DB alone.
- Assume that M is a model of DB and $M \subseteq I$. If $M \not\models \mathcal{C}$ then there must exist an atom $A \notin I$ such that $A \in M$. A contradiction.

■

Corollary 2 *Let I be a subset of the Herbrand Base of a DDDB, DB. Then:*

- *A $CS - \mathcal{P}(DB)|_I$ returns the set $\{M : M \in \mathcal{MM}(DB)$ and $M \subseteq I\}$ among others and its first model returned is in $\mathcal{MM}(DB)$.*
- *A $MM - \mathcal{P}(DB)|_I$ returns exactly the set $\{M : M \in \mathcal{MM}(DB)$ and $M \subseteq I\}$.*

Note that any models that can contribute to the nonminimality of models in I are themselves in I and therefore will be generated by the modified procedure. That is, the test for minimality for generated models is always complete (sufficient).

Example 5 *Consider $DB = \{P(a), P(b) \vee P(c), P(b) \vee P(d), P(e) \vee P(c)\}$ and the set $I = \{P(a), P(b), P(c)\}$.*

The set of minimal models returned is $\{\{P(a), P(b), P(c)\}\}$.

The other minimal models $\{\{P(a), P(b), P(e)\}\}$ and $\{\{P(a), P(c), P(d)\}$ are not generated.

3.2 Checking for Model Minimality:

If I is known to be a model and we need to check for its minimality (or to find a single minimal model of I even when it is not known to be a model) we can improve on the above approach.

Theorem 5 *Let M be a model of DB. Let M' be the first minimal model returned by the model generating procedure $\mathcal{P}(DB)|_M$. Then M is minimal if and only if $M = M'$.*

Proof: Clearly, by Corollary 1, M' is a minimal model for DB and the result follows immediately. ∎

However, we already showed that the first model generated by CS-$\mathcal{P}$ is minimal for its input theory. Therefore, we have the following result:

Corollary 3 *Let M be a model of DB. Then:*

1. *Let M' be the first model returned by $CS - \mathcal{P}(DB)|_M$. Then M is minimal if and only if $M = M'$.*
2. *Let M' be the first model returned by $MM - \mathcal{P}(DB)|_M$. Then M is minimal if and only if $M = M'$.*

Corollary 4 *Let M be a model of DB. Then:*

1. *M is minimal if and only if $CS - \mathcal{P}(DB \cup \{\neg M\})|_M = \emptyset$ (returns no models).*
2. *M is minimal if and only if $MM - \mathcal{P}(DB \cup \{\neg M\})|_M = \emptyset$ (returns no models).*

Proof: We prove the first assertion. The second is proved along the same lines. Assume that $CS - \mathcal{P}(DB \cup \{Neg(M)\})|_M \neq \emptyset$. Each element of $CS - \mathcal{P}(DB \cup \{Neg(M)\})|_M$ is a model of DB and is a proper subset of M. At least one of these models (the first generated) is minimal. M is not minimal.

Now assume $CS - \mathcal{P}(DB \cup \{Neg(M)\})|_M = \emptyset$. M is a model of DB and any proper subset of M is a model of $Neg(M)$. There is no subset of M that is a model of DB for otherwise it will be a model for $DB \cup \{Neg(M)\}$ contradicting our assumption. M is a minimal model of DB. ∎

Example 6 *Let DB be the following set of clauses:*

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

Figure 1 is a model tree for DB. Assume we want to test the minimality of the models: $M_1 = \{P(a), P(b), P(c)\}$ and $M_2 = \{P(b), P(c), P(d)\}$.

Restricting the tree to M_1 will result in the first minimal model generated $\{P(a), P(b)\} \neq M_1$. M_1 is not minimal. Note that adding $Neg(M_1) = \{P(a) \wedge P(b) \wedge P(c) \rightarrow \bot\}$ will still allow the generation of $\{P(a), P(b)\}$ by both $CS - \mathcal{P}|_{M_1}$ and $MM - \mathcal{P}|_{M_1}$.

Restricting the tree to M_2 will result in the first minimal model generated $\{P(b), P(c), P(d)\} = M_2$. M_2 is not minimal. Note that adding $Neg(M_2) = \{P(b) \wedge P(c) \wedge P(d) \rightarrow \bot\}$ will suppress the generation of $\{P(a), P(b)\}$ by both $CS - \mathcal{P}|_{M_2}$ and $MM - \mathcal{P}|_{M_2}$ and make them return the empty set.

3.3 Implementation Issues:

Incorporating restrictions into the model generating process is a straight forward operation. One needs only to check that the atom to be asserted to satisfy a clause belongs to the given set. If so then things proceed as usual otherwise the corresponding branch closes. The fact that the test is performed on the set I and not its complement is important since the complement may be prohibitively large. This is so although we included denial rules corresponding to atoms not in I when defining the restricted model generation procedure.

For minimality checking one can utilize a procedure generating only minimal models (e.g. MM-Satchmo) or one returning a single minimal model (e.g. CS-Satchmo). The choice of which to use will depend on the type of check to be performed. A single test for minimality is better performed using the less expensive CS-$\mathcal{P}$. A search for all minimal models included in a set of atoms can be better accomplished by MM-$\mathcal{P}$.

One can also use the minimality testing approach to convert a CS-$\mathcal{P}$ procedure into a minimal model generating procedure by successively generating models and then checking for their minimality[3, 15]. This may be preferable to collecting constraints corresponding to already generated minimal models to ensure minimality of subsequent models as done in [3, 21].

The approach is easily extendible to the cases when testing is to be performed on a combination of sets of atoms. Given I, to test for minimal models containing no element of I can be achieved by changing the test of the check of whether an atom belongs to I to its complement: the atom not belonging to I. Once more the test is performed on I itself. Other combinations can be treated in the same spirit.

It is possible to further optimize the search by removing from the input set all instances of clauses with some non-I elements in the body. We can also replace by *false* elements with no I atoms in the head. These steps may be performed as preprocessing stage and may be helpful when I is not changing for a large number of queries.

Our testing of a prototype implementation points to gains of up to orders of magnitude in run time when using the restricted algorithm as compared to that of the plain one. The exact numbers depended on how restrictive the set I was (the number of models returned by the restricted procedure, which in turn depends on the size of the restriction set I) and the type of check performed. The time was larger for tests based on negative predicates (not member of) than for the positive predicates (member of). The improvement was better felt for problems with large running times. The explanation is that restricting the search did not only restrict the number of models generated but also the number of constraints that correspond to these models.

4 Computing Perfect Models:

The idea of stratification is to have the theory partitioned in such a way so that decisions on negated atoms in clause bodies are "permanent" in the sense that they are not affected by future decisions of the model generating procedure. This is done by having all the negative body atoms assigned a permanent truth value at an earlier stratum before treating the atoms in the head of the clause in which they occur.

Definition 4.1 *Given a disjunctive normal database DB with a (local) stratification* $\mathcal{S} = \{S_1, ..., S_r\}$ *we define* $St - \mathcal{P}(DB, \mathcal{S})$ *to be the complete model generating procedure* $\mathcal{P}$ *that never expands an element of* S_j *before* S_i *for* $i < j$*.*

Note that working with $St - \mathcal{P}(DB, \mathcal{S})$ is equivalent to letting $\mathcal{RN}_i = (\cup_{j<i} S_j) \cap M$, where M is the computed part of the current model. That is the set for resolving negation at level i consists of all atoms of the current model M of strata $j < i$. The atoms of $\mathcal{RN}_i$ already have their truth values assigned and will not change in the interpretation being developed. By Theorem 3, at level i we always compute the minimal models of $DB_i^{\mathcal{RN}_i}$. Note also that $\mathcal{P}$ can employ splitting or complement splitting: the main thing is that it be complete.

Theorem 6 (perfect model completeness) $St - \mathcal{P}(DB, \mathcal{S})$ *is a complete perfect model generating procedure for the class of stratified disjunctive normal databases. That is, given a stratified DNDB, DB, and (local) stratification* $\mathcal{S}$, $St - \mathcal{P}(DB, \mathcal{S})$ *returns all the perfect models of DB, among others.*

Proof: We recall that for a perfect model of DB, M, $M_i = M \cap S_i$ is a minimal model for S_i.

Let M be a perfect model of DB. We prove that it will be generated by $St - \mathcal{P}(DB, \mathcal{S})$ by showing that $M_i = M \cap S_i$ is returned at level i for all i. The proof is by induction on i:

Base: $i = 1$: $St - \mathcal{P}(DB, \mathcal{S})$ operates with an empty $\mathcal{RN}$ and therefore returns the minimal models of DB_1, $M_1 = M \cap S_1$ among them.

Induction step: Assume for all levels $j < i$, M_j was already generated (if nonempty). At level i, $St - \mathcal{P}(DB, \mathcal{S})$ has $\mathcal{RN}_i = \cup_{j<i} M_j$. It returns all the minimal models of $DB_i^{\mathcal{RN}_i}$, among others, (by Theorem 3). That is, it returns M_i. ∎

While guaranteeing the correct (and timely) assignment of truth values to negative body literals, such a change in the model generating procedure is not sufficient to ensure the generation of only perfect models (soundness). This remains the case even if a minimal model sound procedure is used as $\mathcal{P}$. The following example demonstrates this point:

Example 7 *Let* $DB = \{(\top \rightarrow P(a)), (\top \rightarrow P(b) \vee P(c)), (P(a) \wedge P(c) \rightarrow P(b)), (notP(c) \wedge P(b) \rightarrow R(d) \vee R(e)), (P(b) \wedge P(c) \rightarrow R(e) \vee R(f))\}$. *Clearly DB is stratified with* $S_1 = \{P(a), P(b), P(c)\}$ *and* $S_2 = \{P(d), R(e), R(f)\}$. *Treating the clauses in the order they are given, which is compatible with the stratification, gives the set of models:*
$\{\{P(a), P(b), R(d)\}, \{P(a), P(b), R(e)\}, \{P(a), P(c), P(b), R(e)\}, \{P(a), P(c), P(b), R(f)\}\}$. *Even among the minimal models there is one that is not perfect, namely* $\{P(a), P(c), P(b), R(f)\}$. *This is so since there are other models with less atoms of the first stratum (the first two). The model tree for this example corresponding to this run of a model generating procedure is given in Figure 4.*

It is not difficult to see that for a stratified database, DB, if the clauses are ordered and processed by a model generating procedure according to their stratification then the procedure returns only supported models of DB. This is so since first: a clause with no body negation is expanded only if its body atoms are all in the current interpretation so there is support for all (ground) elements in the head of that clause. Second: an instance of a clause C with negated body atoms at level i with $notA \in Body(C)$, where A is an atom of an earlier stratum $j < i$, some A is already assigned its final

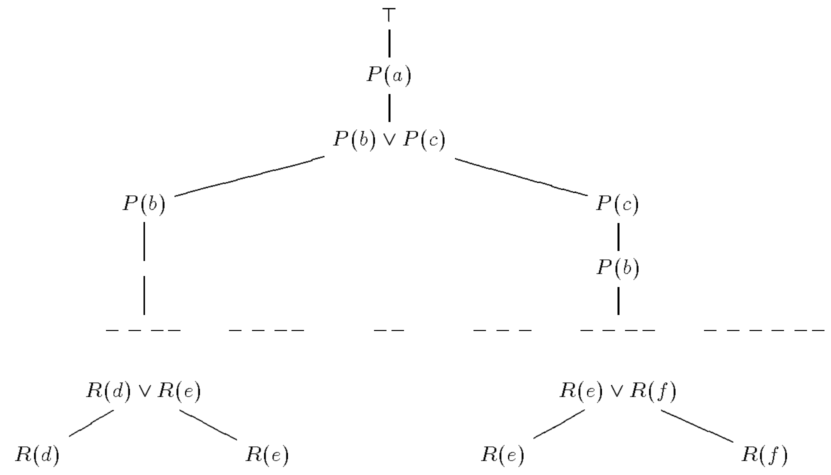

Figure 4: A Tree with Nonperfect Models for Example 7.

truth value in the current interpretation. C has two possibilities: either A is *true* and C will not fire in the current branch since its body is not satisfied, (and neither will C's positive counterpart with A moved to the head). The other case is when all As are *false* and C will act as a support for all of its head atoms in the currently developing interpretation. Not all the returned models are minimal and not all minimal models returned are perfect as was shown in Example 7. So even when we restrict our procedure to generate minimal models of the input theory nonperfect models will still be returned. The reason for generating the extra minimal but not perfect models is that to compute perfect models we need to minimize within individual strata [17, 5] rather that on the level of the entire model of the theory. As will be stated formally later, if we use complement splitting then the first model generated (leftmost) is perfect since each of its components are minimal in their respective strata.

However, to explicitly minimize in individual strata implies having all the potential perfect models of the theory under development simultaneously. Another method offered in [5] is to incrementally pass constraints corresponding to minimization in individual strata from already generated models. That approach is applicable here. It is based on incremental collection of constraints corresponding to individual strata of generated models starting from the highest stratum (last in the processing order). While that approach is desirable from efficiency of computations point of view it is more difficult for implementation, especially non-incremental ones. Next we offer still an alternative approach that is more suited for easy implementation and is in line with the conventional approach to model minimization adopted in this paper and in [3]. The approach is based on temporarily interpreting atoms in strata higher than the current stratum during the model generation process.

Lemma 2 *Let DB be a (locally) stratified database with (local) stratification $\{DB_1, ..., DB_n\}$. Then if when processing stratum i all ground atoms of strata $j > i$ are assigned "true" then for any $C = A_1 \vee \cdots \vee A_m \leftarrow B_1, \ldots, B_n, notD_1, \ldots, notD_k \in DB_i$ we have:*

1. *When $m = 0$ and D_l is false for all $l = \{1, ..., k\}$, the clause C is equivalent to: $\{\bot \leftarrow B_1^i \wedge \ldots \wedge B_{n_i}^i | \{B_1^i, \ldots, B_{n_i}^i\} = \{B_1, \ldots, B_n\} \cap (\cup_{j \le i} S_j)\}$. That is, the assumption is equivalent to stripping away all elements of strata higher than i in C.*

2. *The assumption has no effect when $m > 0$ or D_l is true for some $l \in \{1, ..., k\}$ (for clauses with nonempty head).*

Proof: 1. When processing headless clauses (denial rules) at level i, atoms of lower levels have already been processed and assigned final truth values. Body atoms at higher strata can be deleted since they are *true*. The negative body literals can be dropped since they all evaluate to *true* in the lower strata.

2. By definition of stratification, when processing the head of a clause at level i all atoms of the body are at level i or lower. No assumption is made about levels less than i: all assigned truth values at these levels are final, and therefore such a clause is not affected. Note that the D's are in lower strata than i and are therefore interpreted correctly. If any of the negative body literals is *false* then the clause never fires and can be ignored anyway.

■

One can look at this as processing through a sliding window on the partitions of the Herbrand base induced by the (local) stratification. At any moment atoms of the current stratum are under the window and are being examined to get assigned truth values. Atoms of lower strata are already known and those in higher strata are assumed *true* while they wait for processing. Since all the body atoms of the nondenial clause being treated at level i are in stratum i or lower, they are not affected by this assumption (their values are already fixed). This has a substantial effect when treating constraints (denials) resulting from already generated minimal (perfect) models. Such clauses may contain atoms of different strata. Assuming that atoms of higher strata are *true* will have the effect of minimizing within the current stratum. This is needed to ensure that the generated models are perfect[5]. We modify our model generating procedure so that it acts this way.

Definition 4.2 *By $MSt - \mathcal{P}(DB, \mathcal{S})$ we denote the operation of $St - \mathcal{P}(DB, \mathcal{S})$ with the following properties:*

1. *The procedure employs complement splitting.*

2. *The procedure is modified so that for any generated model M a clause $Neg(M)$ is added to DB in all further processing steps.*

3. *The temporary assignment of true to all atoms in strata above the current one (not yet reached) is made.*

We have the following theorem:

Theorem 7 (perfect model soundness) *Let DB be a (locally) stratified disjunctive normal database with the (local) stratification $\mathcal{S} = \{S_1, ...S_r\}$. Running $MSt - \mathcal{P}(DB, \mathcal{S})$ returns exactly the set of perfect models of DB: $Perfect(DB) = MSt - \mathcal{P}(DB, \mathcal{S})$. That is $MSt - \mathcal{P}(DB, \mathcal{S})$ is sound and complete for perfect model generation.*

Proof: (Sketch)

- The elements of a disjunctive head of a clause are always (by definition) in the same stratum and therefore will be treated on the same level and are not affected by the truth assignments to higher strata. The complement splitting constraints will guarantee that left submodels (in the stratum) are never supersets of their right siblings [21, 3].
- The first model generated is perfect. This is so since all elements of individual strata are minimal in their respective stratum [5].
- Under the assumptions of Definition 4.2, the constraints resulting from already generated perfect models, according to Lemma 2, correspond to minimization in the current stratum. The newly generated sub-model will therefore be minimal in the current stratum since it cannot be subsumed by subsequent sub-models in the current stratum and cannot be a superset of already computed models by the presence of the constraints corresponding to these (minimal) models. This results in all models generated being perfect (soundness).
- Since, by Theorem 6, $St - \mathcal{P}(DB, \mathcal{S})$ is perfect model complete, and no perfect model is removed as a result of minimization in individual strata (a negative clause corresponding to a minimal model is always satisfied in any other minimal model), it follows that the procedure is complete.

■

Example 8 *Consider* $DB = \{(\top \rightarrow P(a)), (\top \rightarrow P(b) \vee P(c)), (\top \rightarrow P(b)), (not P(c) \wedge P(b) \rightarrow P(d) \vee P(e)), (P(b) \wedge P(c) \rightarrow P(e) \vee P(f))\}$*, as in Example 7. The first model generated is* $\{P(a), P(b), P(d)\}$ *which is perfect. The induced constraint is* $P(a) \wedge P(b) \wedge P(d) \rightarrow \bot$*. The next perfect model (at stratum 2),* $\{P(a), P(b), P(e)\}$ *is not affected by this constraint. Now moving to stratum 1 with the two constraints the models* $\{P(a), P(c), P(b), P(e)\}$ *and* $\{P(a), P(c), P(b), P(f)\}$ *will be suppressed by the clause* $P(a) \wedge P(b) \rightarrow \bot$ *(*$P(d), P(e) \in S_2$ *are both assumed* true*). As shown in Figure 5 all models generated are perfect and all the perfect models are produced.*

We want to emphasize the importance of processing individual strata of the database separately. Ignoring this aspect prevents procedure from being perfect model complete. While processing individual clauses in the order of their strata is equivalent to processing each stratum separately, the discrepancy between the positive case and the case with negation need not be overlooked in implementations[5]. For example, while the set of clauses violated in a certain interpretation can be satisfied as a set by asserting atoms of their heads without the danger of sacrificing completeness, the presence of negation may be problematic. Consider Example 4 once more with the clause order reflecting the stratification:

Example 9 *Let* $DB = \{(\top \rightarrow P(a)), (not P(a) \rightarrow P(b)\}$*. Starting from* $I_1 = \{\}$ *we observe that both clauses are violated: their bodies are satisfied but not their heads. So in* I_1 *we need to satisfy the set* $\{P(a), P(b)\}$*. We get* $I_2 = \{P(a)\}$ *to satisfy the first and, if we try to process the other head then we get* $I_3 = \{P(b), P(a)\}$*. Clearly* I_3 *is a not a perfect model of* DB *since it is not minimal. The reason for this behavior is the "nonmonotonic nature" of body satisfiability in the presence of*

[5] Can't apply `findall` violated clauses to the entire theory but to individual strata or clauses.

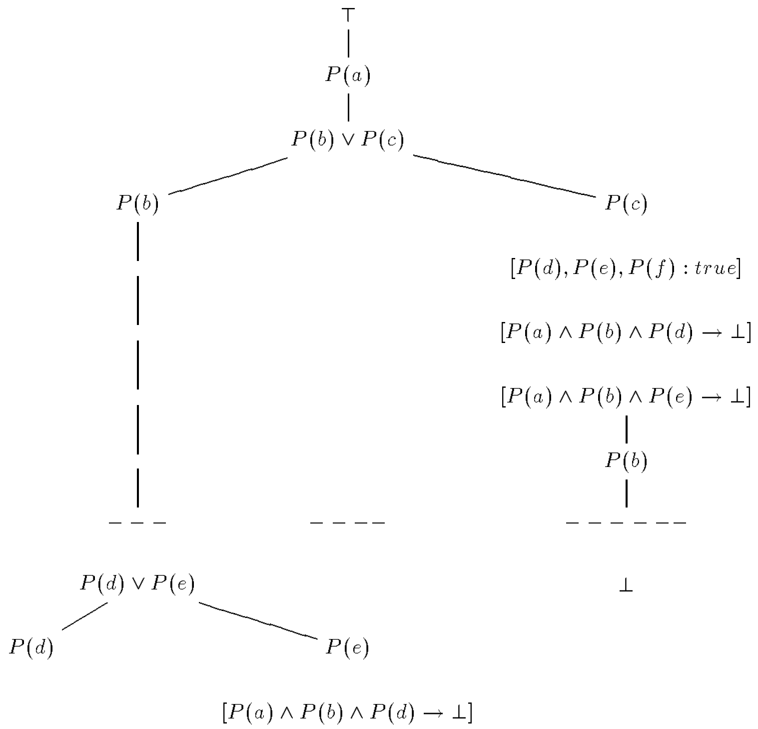

Figure 5: A Perfect Model Tree for Example 8.

negation. Adding more atoms to the interpretation may violate the body and therefore satisfy the entire clause. This is not the case for DDDBs. The morale of this example is that the use of "set" operations should be carefully examined.

5 Computing Stable Models:

While stratified databases constitute a major class of real life theories and received much attention in the literature, there are many useful databases that are not stratified. The perfect model semantics is not applicable for this class of theories.

Stable models are an important means of defining semantics for non-stratified databases. As stated earlier, the set of stable models is a subset of the set of minimal models of the DNDB. When the theory is stratified then its stable and perfect models coincide. For DDDBs stable, perfect and minimal models are the same. Therefore, it will be sufficient to compute the stable models of the theory and the perfect model computation algorithm is a side product.

Our approach to computing stable models is based on computing a *superset* of the set of stable models of a DNDB, DB, that includes the set of minimal models (*model generation pass*) then checking each computed model for stability (*stability checking pass*). The model generation pass is performed using DB^+, the positive transformation of DB (cf. Definition 2.2). Since our theory is safe and RR, $DB^+$is range restricted. A complete model generating procedure can be applied to DB^+ to find a superset of the minimal models of DB ($\mathcal{MM}(DB) = \mathcal{MM}(DB^+)$).

The stability test for a computed model M is based on using M to determine the truth value assignment for the negative body atoms of clauses in DB. The precessing during the stability test is performed on the original DNDB, DB, and not on its positive transformation DB^+. Since M is a model of DB it satisfies all clauses of DB. To avoid the premature assignment of *false* to atoms negated in clause bodies we define the set of resolving negation $\mathcal{RN}$ (Definitions 2.19 and 2.18) to be the already computed model, M. Negative literals in a clause body of DB are interpreted according to M: only atoms not in M can assume the truth value *false* while checking for stability. Atoms in M cannot be assumed to be *false*.

On the other hand, given M, some of the clauses of DB are satisfied in M by simply having a negative atom of the body as an element of M. Such clauses will not fire, although they may have contributed to the content of M during the model generation pass (on DB^+). The stability check pass can produce a result different from M. The stability test can be interpreted as verifying that every atom of M was constructively included in M to satisfy a head of a clause of DB the body of which is satisfied in M. This test is accomplished by letting $\mathcal{RN} = M$ which has the effect of deleting all clauses of DB with "*notA*" in their bodies for some $A \in M$ and deleting all negative body occurrences of atoms in M. This is the Gelfond-Lifschitz transformation DB^M of DB as was shown in Theorem 3. Additionally, the procedure for checking the minimality of models is run restricted to the elements of M. If M proves to be minimal for DB^M then M is a stable model for DB. Otherwise it is not. Formally we have the following:

Lemma 3 *Let DB be a DNDB and DB^+ be its positive transformation. Given a complete model generating procedure $\mathcal{P}$ and a model M of DB^+ then:*

- *M is a stable model of DB if and only if $\{M\} = \mathcal{P}(DB, M)|_M$. That is, M is stable if and only if it is the only model returned by $\mathcal{P}(DB, M)|_M$. Or equivalently,*
- *M is a stable model of DB if and only if $\mathcal{P}(DB \cup \{Neg(M)\}, M)|_M = \emptyset$.*

Proof:
- $\mathcal{RN} = M$. Since $\mathcal{P}$ is complete then by Theorems 3 and 4, $\mathcal{P}(DB, M)|_M$ returns, among others, all the minimal models of DB^M that are subsets of M. Clearly if M is minimal for DB^M (stable for DB) then it must be the only element of $\mathcal{P}(DB, M)|_M$.

  If $\{M\} = \mathcal{P}(DB, M)|_M$ then by completeness M is a minimal model of DB^M and is therefore a stable model for DB.
- If $\mathcal{P}(DB \cup \{Neg(M)\}, M)|_M = \emptyset$ then no proper subset of M is a model for DB^M since it would have been returned as $\mathcal{P}$ is complete. That is, M is a minimal model for DB^M and is therefore a stable model for DB.

  If M is stable then it is minimal for DB^M. Only models that contain a proper subset of M can satisfy $Neg(M)$. Such interpretations cannot be returned by $\mathcal{P}(DB \cup \{Neg(M)\}, M)|_M$ in light of Theorem 3, (they are not models of DB^M).

∎

Theorem 8 *(***stable model soundness and completeness***): Let DB be a DNDB and DB^+ be its positive transformation. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two complete model generating procedures (for DDDBs). Then:*

- *$Stable(DB) = \{M | M \in \mathcal{P}_1(DB^+)$ and $\{M\} = \mathcal{P}_2(DB, M)|_M\}$. Or equivalently,*
- *$Stable(DB) = \{M | M \in \mathcal{P}_1(DB^+)$ and $\mathcal{P}_2(DB \cup \{Neg(M)\}, M)|_M = \emptyset\}$.*

Proof: By completeness $\mathcal{P}(DB^+)$ returns all the minimal models of DB including all the stable models. All and only models passing the test of Lemma 3 are stable. The result follows. ∎

The suggested procedure is quite simple. Use a complete model generating procedure to compute, among others, the set of minimal models of DB by operating on its positive transformation DB^+. Test all generated models for stability using the model itself as the set for resolving negation in DB, $\mathcal{RN}$. Models passing the test are the only members of the set of stable models of DB, $Stable(DB)$.

Example 10 *Let DB be the following set of clauses:*

$$
\begin{array}{ll}
notP(a) \rightarrow Q(a) & notS(a) \wedge P(a) \rightarrow R(a) \\
Q(a) \rightarrow R(a) & notS(a) \rightarrow P(a) \\
\top \rightarrow Q(a) \vee T(a) & notT(a) \wedge P(a) \rightarrow S(a)
\end{array}
$$

DB^+ is the following set of clauses:

$$
\begin{array}{ll}
\top \rightarrow Q(a) \vee P(a) & P(a) \rightarrow R(a) \vee S(a) \\
Q(a) \rightarrow R(a) & \top \rightarrow P(a) \vee S(a) \\
\top \rightarrow Q(a) \vee T(a) & P(a) \rightarrow S(a) \vee T(a)
\end{array}
$$

$\mathcal{MM}(DB^+) = \{M_1 = \{P(a), S(a), T(a)\}, M_2 = \{P(a), R(a), T(a)\}, M_3 = \{Q(a), R(a), S(a)\}\}$.

$DB^{M_1} = \{Q(a) \rightarrow R(a), \top \rightarrow Q(a) \vee T(a)\}$.

$\mathcal{MM}(DB^{M_1}) = \{\{Q(a), R(a)\}, \{T(a)\}\}$ *and* $M_1 \notin \mathcal{MM}(DB^{M_1})$. *Therefore* M_1 *is not stable.*

$DB^{M_2} = \{P(a) \rightarrow R(a), Q(a) \rightarrow R(a), \top \rightarrow P(a), \top \rightarrow Q(a) \vee T(a)\}$.

$\mathcal{MM}(DB^{M_2}) = \{\{P(a), Q(a), R(a)\}, \{P(a), R(a), T(a)\}\}$ *and* $M_2 \in \mathcal{MM}(DB^{M_2})$. *Therefore* M_2 *is stable.*

$DB^{M_3} = \{\top \rightarrow Q(a), Q(a) \rightarrow R(a), \top \rightarrow Q(a) \vee T(a), P(a) \rightarrow S(a)\}$.

$\mathcal{MM}(DB^{M_3}) = \{\{Q(a), R(a)\}\}$ *and* $M_2 \notin \mathcal{MM}(DB^{M_3})$. *Therefore* M_3 *is not stable.*

Aside from completeness, Theorem 8 places no restrictions on the model generating procedures used. The choice is left to the user. Using complement splitting will restrict the search space and prevent multiple generation of the same model. This generally improves performance. Utilizing a sound minimal model generating procedure will have the effect of limiting the set of candidate models to the minimal models of DB. However, the cost of model minimization must be weighed against the cost of testing for stability. This reasoning applies to the procedure used for generating candidate models and for the one used in the testing for stability. We don't require that the same procedure be used for both model generation and testing .

While the outlined procedure for computing stable models is applicable for computing perfect models if DB is stratified (and for computing the minimal models if DB is a DDDB) it is more economical to use the procedures developed for that purpose in cases when the (local) stratification of the database is given or is not very expensive to compute. Only if doubts about the class of the database (whether it is stratified or not) are present then the stable model procedure is utilized.

Before concluding this section we would like to observe a "conceptual" connection between our approaches to computing perfect and stable models. A way to view the computation of perfect models for a stratified database is as the process in which negative literals in the body of a clause always get assigned truth values before using them to determine the truth values for head atoms. Abandoning potential nonperfect models is achieved by minimizing in each stratum separately (or using a technique with an equivalent effect as described earlier). The resulting perfect model is the set union of atoms in all strata. With this in mind, we can view the computation of stable models for a nonstratified database DB as computing the perfect model of an artificially stratified database $DB^s = DB^+ \cup DB$. The first stratum is DB^+, which we call the *phantom* stratum, and the second stratum is DB itself. As a DDDB, DB^+ has no negative body literals. Therefore, computing in the phantom stratum DB^+ proceeds normally (with the empty $\mathcal{RN}$). Minimizing in the first stratum (DB^+) will always yield minimal models of DB^+ (and of DB). The computed (minimal) model of DB^+ (the phantom stratum), say M, is used as $\mathcal{RN}$ to determine the values for negative body literals in DB. This is equivalent to applying the minimal model generating procedure to DB^M Since a minimal model of DB^+ is a minimal model of DB the only possible cases that can happen is that a proper subset of M may be a minimal model of DB^M. In this case M is not stable. Otherwise M is a stable model of DB.

Of course, the stratification is somehow hypothetical since both strata may define the same set of atoms. That is the reason why we call the first a *phantom* stratum. An alternative would be to rename the predicates in one of the strata so that to achieve real stratification and to use negative literals from the first stratum in the second. This comes close to other approaches that will be compared with ours in the next section. We argue that while such an approach is interesting it does little to the efficiency of computing stable models.

6 Remarks and Conclusions:

The procedures developed in this article can be utilized in the various aspects of processing disjunctive databases. The efficient computation of restricted models is of importance for distributed databases where only a subset of the Herbrand base is of interest at a particular location. Additionally, this approach can be helpful in checking properties of interpretations including the minimality of models using only the context of the model being tested and without reference to other models of the theory [15]. This approach was also shown to be helpful for computing stable models were the test for stability is constrained to the atoms of the model being tested.

Clearly, a complete procedure for computing stable/perfect models can be incorporated into a system for answering queries under the respective semantics. The query answering process is basically a search for instances of the query in each of the computed models. A query answering procedure can use model generation to construct the model tree under the proper semantics and searches can be performed on the tree. The (static) model tree is modified or reconstructed when the database is updated. Alternatively, a run of the model generating procedure can be performed for each query and each model is checked (searched) as soon as it is generated to see if it satisfies the query. The exact approach will depend on the frequency of updates as compared to query answering requests. One should be especially careful with the monotonicity issue when dealing with query answering under perfect and stable semantics. As opposed to reasoning under, say, minimal model semantics, even a positive query may be *true* in the current state of the database DB and *false* in DB^u achieved by adding a positive clause to DB. This point is demonstrated by the following simple example[6]:

Example 11 *Let* $DB = \{notP(a) \rightarrow S(a)\}$*. The only stable (and perfect) model of* DB *is* $\{S(a)\}$*.* $Q = S(a)$ *is true (derivable) under the stable (and perfect) model semantics. However, for* $DB^u = \{\top \rightarrow P(a), notP(a) \rightarrow S(a)\}$ *the only stable (and perfect) model of* DB^u *is* $P(a)$*.* $Q = S(a)$ *is not true (not derivable) under the stable (and perfect) model semantics in* DB^u *and* $DB \subset DB^u$*.*

Another potential use is in integrity constraints checking in disjunctive deductive databases. Integrity constraints are rules that describe properties of a database state. Only models of the (completed) theory that satisfy the constraints are admissible. As such, integrity constraints do not participate constructively in the model generation process. No atom is added to a model for the sole purpose of satisfying a constraint. The clauses representing the constraints are put in denial form, possibly with negative body occurrences, and the resulting rules are checked after the model generation procedure finishes its work. Only models satisfying the constraints are accepted. All the others are pruned. A simple approach to checking constraints will be to add the integrity constraints as an additional stratum of the database and use a perfect model generating procedure to compute the perfect models of the thus modified database. These are the models of DB satisfying the constraints.

[6] The issue of which updates may invalidate previously derived facts is beyond the scope of this paper.

The relation between stratification and circumscription [14] was studied in [11, 10, 18]. The results make it possible to apply the approach described here to generating perfect models for computing prioritized circumscription[5].

Several approaches were suggested in the literature for computing of perfect and stable models for disjunctive databases [2, 9, 6, 5, 19]. In contrast to others, the approach presented here computes models for disjunctive theories and is not limited to the ground case but deals with the class of range restricted theories with finite minimal models [2, 6, 5, 19]. Additionally, our approach integrates the different steps of generating the required class of models into a coherent system based on the simple idea of model generation. Model generation is used both for generating the set to be tested and for testing for stability. Generally, both here and in other approaches, computing perfect models is simply evaluating clauses in the order of their stratification and ensuring minimality of lower strata for retained (sub)models. In [5] minimality within individual strata is ensured by passing constraints corresponding to components of the model on that stratum to all potential models sharing atoms of lower strata and retracting these constraints when moving up the model evaluation tree. Here we achieve the same effect by asserting a constraint corresponding to the entire model generated and we never need to withdraw such a constraint. It is disabled automatically when exhausted. We believe that the approach outlined here is easier to implement and more in line with the approach adopted for minimal model generation. The cost is that one stores constraints that are larger than the minimum needed to ensure correctness of the model computation process. Of course the approach of [5] can be adopted here when deemed necessary.

In [6], [5] and [19] the approach to computing stable models is to transform the theory into a stratified one through the so called *evidential transformation*, then compute the perfect models of the transformed theory which are shown to be the stable models of the original. It is similar to the (positive) transformation adopted here but the moved negative body atoms are distinguished from their positive counterparts. In a model, the new atoms are treated as components in need of evidence. That evidence is supplied by the presence of the corresponding *objective* atoms in the model. Only models in which there is evidence for every atom that need it are accepted. The second stratum consists exclusively of denial constraints and is designed to reject models not satisfying this property. In a sense, the first stratum generates the "minimal" models of the transformed theory and the second selects those that are stable. To account for the extra atoms added only *objective* atoms of accepted models atoms are returned. A pass through each model is needed for this purpose. Of course a minimal model generating procedure is needed to ensure the minimality of the models generated for the first stratum. This requirement is implicit in demanding that accepted models be *perfect*.

The approach presented here can be viewed in a similar light. The positive transformation of the theory is used to generate the set of candidate models that are in need of evidence. Rather than evidence for individual atoms we need to provide evidence that all atoms in the candidate model produced on the first stage are supported simultaneously. The second pass is performed to provide this support. This may be more in the spirit of the basic definition of stable models provided in [8]. While our algorithm performs the stability checking pass on the original (normal) theory after the model generating pass on its positive transformation, the second pass is restricted by the models produced in the first pass, in the sense of Definition 3.1. Additionally, no separate model minimality checking is needed since clauses to represent that checking are added to the theory on the second pass. This is not a cumulative process since constraint are retracted as soon as the stability is tested.

The minimality and stability checking are integrated into one pass. As suggested by Theorem 2 we can also incorporate the effects of the positive transformation by using the Complete UHR-rule rather than physically transforming the theory. An advantage of the approach of [5] is that the first model generated is stable[7], a property not enjoyed by the approach adopted here.

The approach of Inoue et.al. [9] is also based on transforming the normal theory by adding a new set of atoms to reflect that a given fact is *believed* or *not believed*. Two schemata are added to define acceptable assumptions. Only models surviving the schemata conditions are subjected to the stability test which basically consists of making sure that every assumed atom is objectively *true*. In many respects this approach agrees with the evidential transformation approach in as far as it compares with the method discussed in this paper. In particular, both the evidential transformation and Inoue's approaches to computing stable models expand the Herbrand base of the theory by including evidences. This usually leads to a larger number of models (possibly with higher cardinality) for the transformed theory that need to be tested for stability as compared to the original database. Our approach restricts itself to the original Herbrand base.

The procedures described in [4] and [16] compute the set of stable models for definite normal theories. While these approaches can be extended to the disjunctive case, this implies the need for minimality tests during the checks for stability which introduces into these procedures all the difficulties associated with minimal model computations. After applying the G-L transformation to a normal disjunctive database for a given interpretation, the resulting theory may still be disjunctive and both minimal and nonminimal models may be returned by a model generation procedure [3, 15]. As opposed to [16] the grounding of the theory in our approach is integrated in the model generating process and is not performed as a separate task. Partially developed models guide the grounding process and generally make it more efficient by prohibiting the generation of irrelevant instances. Expanding on the entire set of atoms that occur negatively in rule bodies [4] has the advantage of resolving negation as soon as possible but may result in expanding atoms that are irrelevant to the model generation process such as those occurring in conjunction with positive atoms that occur in no clause head. In the worst case this set can be all the Herbrand base. Our approach stops as soon as a contradictory pair of literals is generated (actually the $\perp$ atom is added to the branch) rather than testing for model consistency after the complete interpretation is produced [4] or after each addition of an atom to the branch [16].

While we don't impose any order on the expansion of the atoms of a clause, such an order can be integrated so as to favor the expansion of atoms occuring negatively in clause bodies if that is deemed to be helpful or to incorporate heuristics such as incremental evaluation of closures and backward propagation of truth values. Other efficiency enhancement measures such as program reduction after updating an interpretation and the proper treatment of body atoms with different polarities during the reduction process are implicit in our algorithm. The approach allows for optimizations that take into account the properties of the theory under consideration to optimize performance. Examples are simplifications for the ground case [15, 21] and the use of semi-stratification to localize stability tests and thus reduce the search space [6].

Our approach is meant to compute only the set of stable models of the theory and has no provisions for computing the well founded model structure as is the case for [4, 16].

[7]This property is not explicitly stated in [5] but follows directly from the fact that all perfect models of the transformed theory are stable and the first model generated is perfect.

While we still generate more models than those that are stable, our generation process is sort of focused on those models that already passed one of the tests for stability: being minimal models for the original disjunctive theory. An additional advantage of our approach for computing stable models is that it requires the completeness of the model generating procedure rather than insisting on minimal model soundness. In this sense the described approach can be viewed as defining an entire set of progressively refined procedures which differ in the size of their search space for candidate models. Some (employing plain splitting) return both nonminimal and duplicate models, others (employing complement splitting) have a more restricted search space and avoid duplicates and still others return only minimal models. Any of these can be employed for generating candidate models. The tradeoff is that less costly procedures tend to return more candidate models than need to be tested. The choice of a particular procedure for implementation may depend on the class of problems to be solved.

The constructed model structure can be used to answer queries under the appropriate semantics. But since this structure generally changes (not necessarily monotonically) with database updates it is of interest to integrate the query answering process into the model generation procedures so that it returns only models satisfying the query, if any. Topics for further development also include more work on the implementation and benchmarking evaluation of the advanced algorithms. While the preliminary results were encouraging, a more systematic approach to testing will enable working out recommendations on the conditions under which the algorithms have their best performance. Another topic is to design a general parameterizable algorithm that can be utilized for computing different types of models and maybe utilized in a variety of database processing tasks.

References

[1] K.R. Apt, H.A. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann Pub., Washington, D.C., 1988.

[2] C. Bell, A. Nerode, R. Ng, and V.S. Subrahmanian. Mixed integer programming methods for computing non-monotonic deductive databases. *Journal of the ACM*, 41(6):1178–1215, 1994.

[3] 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/.

[4] W. Chen and D. Warren. Computation of stable model and its integration with logical query processing. Technical report, Department of Computer Science, SUNY at Stony Brook, Aug 1994. ftp at ftp.ms.uky.edu:pub/lpnmr/chen2.ps.

[5] J.A. Fernandez, J. Minker, and A. Yahya. Computing perfect and stable models using ordered model trees. *J. Computational Intelligence*, 11(1):141–160, 1994.

[6] J. A. Fernández, J. Lobo, J. Minker, and V.S. Subrahmanian. Disjunctive LP + integrity constraints = stable model semantics. *Annals of Mathematics and Artificial Intelligence*, 11(3-4):449–474, 1993.

[7] J. A. Fernández and J. Minker. Computing perfect models of disjunctive stratified databases. *Journal of Logic Programming*, 25(1):33–50, 1995.

[8] 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.

[9] 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.

[10] V. Lifschitz. Closed world databases and circumscription. *Artificial Intelligence*, 27(2):229–235, November 1985.

[11] V. Lifschitz. Computing circumscription. In *IJCAI 85*, pages 121–127, 1985.

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

[13] 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.

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

[15] 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.

[16] I. Niemmelä and P. Simons. efficient implementation of the well-founded and stable model semantics. Technical Report 7-96, Institut für Informatik, Univesität Koblenz, Koblenz, Germany, 1996.

[17] T. 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.

[18] T. 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.

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

[20] J.C. Shepherdson. Negation in Logic Programming. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 19–88. Morgan Kaufman Pub., 1988.

[21] 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.