Annals of Mathematics and Artificial Intelligence 8(1993)449–474

Disjunctive LP + integrity constraints = stable model semantics

José Alberto Fernández*, Jorge Lobo☆, Jack Minker*,†

and

V.S. Subrahmanian*,†

*Department of Computer Science, University of Maryland, College Park, MD 20742, USA

†Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA

☆Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, IL 60680, USA

Abstract

We show that stable models of logic programs may be viewed as minimal models of programs that satisfy certain additional constraints. To do so, we transform the normal programs into disjunctive logic programs and sets of integrity constraints. We show that the stable models of the normal program coincide with the minimal models of the disjunctive program that *satisfy* the integrity constraints. As a consequence, the stable model semantics can be characterized using the *extended generalized closed world assumption* for disjunctive logic programs. Using this result, we develop a bottom-up algorithm for function-free logic programs to find all stable models of a normal program by computing the perfect models of a disjunctive stratified logic program and checking them for consistency with the integrity constraints. The integrity constraints provide a rationale as to why some normal logic programs have no stable models.

1. Introduction

We consider the problem of finding the stable models of normal logic programs using iterative techniques. To do this, we transform the normal programs into disjunctive logic programs and sets of integrity constraints. We show that the stable models of the normal program coincide with the minimal models of the disjunctive program that are consistent with the integrity constraints. This result implies that it is possible to define a fixpoint characterization of the stable models of a program independently of the Gelfond–Lifschitz transform. An important consequence of this is the fact that the stable model semantics can be characterized using the

extended generalized closed world assumption for disjunctive logic programs. This characterization sheds new light on the issue of what it means for a normal program to be inconsistent with respect to the stable model semantics.

In many cases, a normal program consists of two or more parts. Some parts are stratifiable and others are not. We improve the result by transforming only the non-stratifiable part of the normal logic program as described above. The resulting program is a stratified disjunctive logic program where we compute its perfect models. This improves over the case where the entire normal logic program is transformed since, in many cases, it reduces the number of models that must be checked for stability.

Using this result and the fixpoint and bottom-up algorithm of Fernández and Minker [6] for disjunctive stratified databases, we develop a bottom-up algorithm for function-free logic programs that finds all the stable models of a normal program by computing the minimal models of a stratified disjunctive logic program and checking for their consistency with respect to the integrity constraints. Moreover, the integrity constraint check can provide the rationale as to why some normal logic programs have no stable models.

This work is structured as follows. Section 2 defines disjunctive normal logic programs and integrity constraints. It also presents a characterization of constraint satisfaction in a disjunctive logic program in terms of the extended generalized closed world assumption (EGCWA) [20]. Section 3 presents the *evidential transformation* and describes how it relates to the stable model semantics of normal logic programs. Section 4 reviews the results of Fernández and Minker in computing the perfect models of disjunctive stratified databases. Finally, in section 5 we use the bottom-up algorithm of Fernández and Minker [6] to compute the perfect models of stratified disjunctive deductive databases, and in this way generate the stable models of normal deductive databases. We extend the algorithm to incorporate integrity constraint checking.

2. Background

A normal disjunctive logic program P is a finite set of clauses of the form

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

where $k > 0$, $n, m \geq 0$ and the A_i, B_j and D_l are atoms. The operator *not* represents nonmonotonic *negation as failure.*

If $k = 1$ for all clauses in P, the program is called a *normal logic program.*[1] When $m = 0$ for all clauses in P, the program is said to be negation-free and is called simply a *disjunctive logic program.*

[1] We will use the term *non-disjunctive logic program* when we emphasize that a program is not disjunctive.

A ground atom or positive clause C is a logical consequence of a negation-free disjunctive program P iff C is true in all minimal Herbrand models of P. Since minimal models are not unique in the disjunctive case, the meaning of a disjunctive program P is characterized by its set of minimal Herbrand models $\mathcal{M}_P$. Definite programs have a unique minimal model, usually denoted by M_P. The minimal models of definite or disjunctive logic programs can be computed iteratively using the fixpoint operators T_P and T_P^M for definite and disjunctive programs, respectively [18,4].

For normal programs, the minimal models semantics is not always natural, since some of these models do not seem to reflect the intended meaning of the program. An example of this is represented by the program $\{p \leftarrow not\ q\}$. The program has two minimal Herbrand models: $\{p\}$ and $\{q\}$. Only the model $\{p\}$ seems to be a natural model of the meaning of the program since p is supported by $not\ q$ in $\{p\}$, but there is no similar support for q.

Different semantics have been proposed for particular classes of programs to reflect their intended meaning. For the class of stratified programs, one semantics is the *perfect models semantics*. A positive clause C is considered a logical consequence of P iff C is true in all *perfect models* of a disjunctive logic program P. The perfect models semantics produces a unique perfect model in the case of non-disjunctive stratified programs [13], and it is equivalent to the prioritized circumscription of the predicates on each strata of the program [10,13]. A disjunctive logic program P is stratified iff there exists a partition of the predicate symbols defined by the program $\{S_1, \ldots, S_r\}$ such that if a clause defines a predicate in S_i, then the atoms on the right-hand side have predicate symbols belonging to $\cup_{j=1}^{i-1} S_j$ if the atom occurs negatively, or belonging to $\cup_{j=1}^{i} S_j$ if it occurs positively.

The partition or stratification of the predicate symbols induces a stratification $\{P_1, \ldots, P_r\}$ of the clauses of the program defining each predicate. This structure of the program can be used to iteratively compute its perfect models. The perfect models semantics coincides with the minimal models semantics when computed for negation-free programs. The notation $\mathcal{M}_P$ will refer to the perfect models when P is stratified.

For larger classes of programs, different semantics have been proposed. We are interested in the study of how to compute effectively the *stable model semantics* [8]. In particular, we are interested in how to compute the stable models iteratively, as in the case of negation-free or stratified programs.

The stable models semantics characterizes the meaning of a normal program by a set of minimal models called *stable models*, which are defined using the Gelfond–Lifschitz transformation. This transformation in general is defined as follows.

DEFINITION 2.1 [8]

Let P be a disjunctive normal logic program and let I be an interpretation:

$$P^I = \{(A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_n) : (A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_n,\ not\ D_1, \ldots, not\ D_m)$$

is a ground instance of a clause in P and $\{D_1, \ldots, D_m\} \cap I = \emptyset\}$.

P^I is the Gelfond–Lifschitz transformation of P with respect to I, where the A_i, B_j and D_l are atomic formulae.

The result of the Gelfond–Lifschitz transformation is a negation-free disjunctive (definite) program. Stable models for non-disjunctive logic programs may now be defined as follows.

DEFINITION 2.2 [8]

Let P be a definite normal program. M is a stable model of P iff M is the unique minimal model of P^M.

For disjunctive programs, which usually have more than one minimal model, stable models are formally defined as follows.

DEFINITION 2.3 [14]

Let P be a disjunctive normal program. M is a stable model of P iff M is a minimal model of P^M.

Hence, under the stable models semantics, a ground atom or positive clause C is a logical consequence of a disjunctive (definite) normal logic program P iff C is true in all the stable models of P.

For non-ground function-free programs, the use of these definitions to check if an interpretation I is stable is expensive.[2)] A new program P^I must be constructed and its least fixpoint computed. Moreover, non-stratified definite normal programs can have more than one stable model or no stable models at all. If we are interested in finding all the stable models of P, it is important to reduce the search space (i.e. interpretation to be tested) and the verification process (i.e. stability verification) as much as possible.

2.1. KNOWLEDGE BASES AND INTEGRITY CONSTRAINTS

Our approach to stable models consists in finding a set of models that cover them; from this set we test each model for stability. At the same time, we reduce the verification of stability in the model to a test for integrity constraint satisfaction.

Integrity constraints (*IC*) describe the correct states that a knowledge base (*KB*) can take. A *KB* is considered to be correct (or possible) if it satisfies the integrity constraints. There are several approaches to describe integrity constraint satisfaction [9, 16, 17]; we consider only the first two approaches below.

[2)] Without the function-free restrictions, the task becomes undecidable.

DEFINITION 2.4 (CONSISTENCY) [9]

KB satisfies *IC* iff $KB \cup IC$ is consistent.

DEFINITION 2.5 (ENTAILMENT) [16]

KB satisfies *IC* iff $KB \models IC$.

If we restrict our knowledge base to be a definite logic program P and assume that its meaning is given by its minimal Herbrand model (i.e. the meaning of P is represented by the theory $P \cup CWA(P)$), the two concepts of consistency and entailment coincide. This can be trivially proved since $P \cup CWA(P)$ is a complete theory.

LEMMA 2.6

$P \cup CWA(P) \cup IC$ is consistent iff $P \cup CWA(P) \models IC$.

However, the situation changes when the knowledge base is indefinite. Let P be a disjunctive logic program. A similar characterization of its minimal Herbrand models can be described by the extended generalized closed world assumption.

DEFINITION 2.7 [20]

$$EGCWA(P) = \{\neg A_1 \vee \cdots \vee \neg A_n : A_i \in HB_P,\ n > 0$$
$$\neg A_1 \vee \cdots \vee \neg A_n \text{ is } \textit{true}$$
$$\text{in every minimal Herbrand model of } P\}.$$

Yahya and Henschen [20] show that this set is maximally consistent in the following sense:

> No new disjunctions of negated atoms can be added to $EGCWA(P)$ without being able to prove, from $P \cup EGCWA(P)$, new positive disjuncts that were not provable from P.

This observation leads to the following lemma.

LEMMA 2.8 [20]

M is a minimal Herbrand model of P iff M is a model of $P \cup EGCWA(P)$.

Since a disjunctive logic program P may have more than one minimal model, $P \cup EGCWA(P) \cup IC$ is consistent iff some minimal Herbrand model of P is a model of IC. On the other hand, $P \cup EGCWA(P) \models IC$ iff every minimal Herbrand model of P is a model of IC.

EXAMPLE 1

Let $P = \{a(1), a(2) \vee b(2), b(1) \vee b(2)\}$. Let the integrity constraint be the formula $(\forall X)\,(\neg a(X) \vee \neg b(X))$.

$$\mathcal{M}_P = \{\{a(1), a(2), b(1)\}, \{a(1), b(2)\}\}.$$

Only the model $\{a(1), b(2)\}$ satisfies the constraint; hence, the program does not entail the constraint but is consistent with it.

Intuitively, under the consistency interpretation, integrity constraints can be used as filters for models that are not interesting (meaningful) in the *KB*.

2.2. INTEGRITY CONSTRAINTS FOR DISJUNCTIVE LOGIC PROGRAMS

We represent integrity constraints by clauses of the form

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

where $k + n > 0$ and the A_i and B_j are atoms. The use of "$\Leftarrow$" instead of "$\leftarrow$" is only syntactical to allow us to differentiate between clauses of a program and integrity constraints. Integrity constraints add semantic information about the kind of models that are meaningful with respect to the intended meaning of the program.

DEFINITION 2.9

Let P be a disjunctive logic program and let $\mathcal{M}_P$ be the set of its minimal models. Let IC be a set of integrity constraints; then

$$\mathcal{M}_P^{IC} = \{M \in \mathcal{M}_P : M \models IC\}.$$

$\mathcal{M}_P^{IC}$ is the set of minimal models of P that satisfy IC.

The use of Kowalski's [9] definition of integrity constraints for disjunctive logic programs means that the model semantics of disjunctive programs with integrity constraints is characterized by the set of model $\mathcal{M}_P^{IC}$.

LEMMA 2.10

M is a model of $P \cup EGCWA(P) \cup IC$ iff $M \in \mathcal{M}_P^{IC}$.

Proof

$M \models P \cup EGCWA(P) \cup IC$ iff $M \models P \cup EGCWA(P)$ and $M \models IC$ iff $M \in \mathcal{M}_P$ by lemma 2.8, and $M \models IC$ iff $M \in \mathcal{M}_P^{IC}$. □

In the following section, we will show how integrity constraints can be used to select the stable models of a normal logic program.

3. Stable models and the evidential transformation

Our first approach to computing the stable models of a normal program P transforms the problem into the computation of the minimal models of a negation-free disjunctive program $P^{\mathcal{E}}$ and the selection of those minimal models that satisfy a set of integrity constraints IC_P. To verify the stability of the models of P that correspond to the minimal models of $P^{\mathcal{E}}$, it is not necessary to apply the Gelfond–Lifschitz transformation.

This new program $P^{\mathcal{E}}$ uses a new set of predicate symbols called *evidence* (they are the reason for the name "evidential transformation"). For each predicate symbol p, we introduce a new predicate symbol $\mathcal{E}p$ whose intended meaning is "there is evidence of p". Although these new predicate symbols have an autoepistemic flavor, their use in the transformed program is completely classical. Given an atom $A \equiv p(\boldsymbol{x})$, we will denote the atom $\mathcal{E}p(\boldsymbol{x})$ by $\mathcal{E}A$.

From its intended meaning, it is clear that $\mathcal{E}A \leftarrow A$ must be true for every atom of the program. The role of the "evidence" is to separate the positive use of an atom q in the body of a clause, which has a classical meaning, from the use of its negation (i.e. *not* q) where the *negation as failure to prove* introduces a nonmonotonic usage of the same atom.

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

$$A \leftarrow B_1, \ldots, B_n, \neg\mathcal{E}D_1, \ldots, \neg\mathcal{E}D_k,$$

where A is true if the B_i are true and "there is no evidence of the D_i" atoms. The use of classical negation in this rule eliminates any nonmonotonic property of the program. To recover it, we must take full account of what it means for $\mathcal{E}A$ to provide evidence of A.

In an interpretation I, we say that an atom $\mathcal{E}A$ is consistent with its intended meaning, iff

$$\mathcal{E}A \in I \Rightarrow A \in I.$$

In other words, if "there is evidence of A" in the interpretation, then A must be present in the interpretation. An interpretation where this is not the case is inappropriate since one should not claim to have evidence of A (claim to have "a proof of A") without A being true.

3.1. EVIDENTIAL TRANSFORMATION

Using the ideas presented above, the evidential transformation can be defined formally as follows.

DEFINITION 3.1

Let P be a normal logic program. The *evidential transformation of* P defines a disjunctive logic program $P^{\mathcal{C}}$ and a set of integrity constraints IC_P such that

1. For each clause $A \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, not\ D_k$ in P, the clause $A \vee \mathcal{C}D_1 \vee \cdots \vee \mathcal{C}D_k \leftarrow B_1, \ldots, B_n$ belongs to $P^{\mathcal{C}}$.
2. For each predicate symbol p of P, the clause $\mathcal{C}p(\boldsymbol{x}) \leftarrow p(\boldsymbol{x})$ belongs to $P^{\mathcal{C}}$.
3. For each predicate symbol p of P, IC_P contains an integrity constraint of the form $p(\boldsymbol{x}) \Leftarrow \mathcal{C}p(\boldsymbol{x})$.

Nothing else belongs to either $P^{\mathcal{C}}$ or IC_P.

Note that definition 3.1 starts with a normal logic program that does not contain integrity constraints. The evidential transformation alters the normal logic program to a disjunctive logic program and introduces a set of integrity constraints. The disjunctive clauses are equivalent to $A \leftarrow B_1, \ldots, B_n, \neg\mathcal{C}D_1, \ldots, \neg\mathcal{C}D_k$, since we are using classical negation instead of negation as failure. In step 2, we add clauses that define the *evidence* predicates: "there is evidence of an atom A" if the atom A is true. The integrity constraints IC_P, introduced in step 3, restrict the models to those representing the intended meaning of the program: if "there is evidence of an atom A", then it should be the case that A is true.

Interpretations in $HB_{P^{\mathcal{C}}}$ have a richer structure than interpretations in HB_P. In the first, there are two different classes of objects: evidences and objective atoms. This simple structure is rich enough to characterize syntatically those models of $P^{\mathcal{C}}$ that satisfy IC_P.

LEMMA 3.2

Let N be a model of $P^{\mathcal{C}}$. Then: N satisfies IC_P iff $N = M \cup \mathcal{C}M$ for some $M \subseteq HB_P$, where $\mathcal{C}M = \{\mathcal{C}A: A \in M\}$.

Proof

Let M and M' be subsets of HB_P such that $N = M \cup \mathcal{C}M'$. Since N is a model of $P^{\mathcal{C}}$, then $N \models \mathcal{C}A \leftarrow A$ for all $A \in HB_P$ and therefore $M \subseteq M'$. N is a model of IC_P iff $N \models A \Leftarrow \mathcal{C}A$ for all $A \in HB_P$ and therefore $M \supseteq M'$. Hence, $N = M \cup \mathcal{C}M$ for some $M \subseteq HB_P$. □

The importance of this lemma, as will be seen after the next theorem, is that it allows us to reduce the stability test of a model to a sequential check of its elements. The relation between the stable models of P and the minimal models of $P^{\mathcal{C}}$ is clear from the following theorem.

THEOREM 3.3

Let P be a normal program. An interpretation M is a stable model of P iff

$$(M \cup \mathcal{C}M) \in \mathcal{M}_{P^{\mathcal{C}}}^{IC_P}.$$

Proof

Given a program P, we should look at the transformed programs P^M and $P^{\mathcal{C}}$. For a ground instance of a clause $C = (A \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, not\ D_k) \in P$, we denote the transformed versions of the clause as follows:

- $C^M = (A \leftarrow B_1, \ldots, B_n)$,
- $C_{\mathcal{C}} = (A \vee \mathcal{C}D_1 \vee \cdots \vee \mathcal{C}D_k \leftarrow B_1, \ldots, B_n)$.

($\Rightarrow$) First we prove that $M \cup \mathcal{C}M$ is a model of $P^{\mathcal{C}}$ and of IC_P. Then we prove that it is minimal. Using lemma 3.2, we only need to prove that it is a model for the transformed clauses of P.
Let $C = (A \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, not\ D_k)$ be a ground instance of a clause in P.
Assume $\nexists j\ D_j \in M$. Since $M \models C^M$, then $A \in M$ or $\exists i\ B_i \notin M$. Hence $(M \cup \mathcal{C}M) \models C_{\mathcal{C}}$.
Assume $\exists j\ D_j \in M$, then $C^M \notin P^M$. But $\mathcal{C}D_j \in \mathcal{C}M$, hence $(M \cup \mathcal{C}M) \models C_{\mathcal{C}}$.
Therefore $(M \cup \mathcal{C}M) \models P^{\mathcal{C}}$ and $(M \cup \mathcal{C}M) \models IC_P$.

Minimality: Assume $\exists N \subset M \cup \mathcal{C}M$ such that $N \models P^{\mathcal{C}}$ and let $N = M' \cup \mathcal{C}M''$. We prove that in this case M' is a model of P^M, contradicting the assumption that M is the minimal model of P^M.
Since $M' \cup \mathcal{C}M'' \models \mathcal{C}A \leftarrow A$, $\forall A \in HB_P$, then $M' \subseteq M'' \subseteq M$ and $M' \subset M$.
Let $C = (A \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, not\ D_k)$ be a ground instance of a clause in P. Then $M' \cup \mathcal{C}M'' \models C_{\mathcal{C}}$.
Assume $\nexists j\ D_j \in M$, then $A \in M'$ or $\exists i\ B_i \notin M'$. Hence, $M' \models C^M$.
Assume $\exists j\ D_j \in M$, then $C^M \notin P^M$.
Hence, $M' \models P^M$ since $\forall C^M \in P^M\ M' \models C^M$. ($\Rightarrow\Leftarrow$)

Therefore, $M \cup \mathcal{C}M \in \mathcal{M}_{P^{\mathcal{C}}}^{IC_P}$.

($\Leftarrow$) As before, we prove that M is a model of P^M and then that it is minimal.
Let $C = (A \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, not\ D_k)$ be a ground instance of a clause in P.
Assume $\nexists j\ D_j \in M$. Since $M \cup \mathcal{C}M \models C_{\mathcal{C}}$, $A \in M$ or $\exists i\ B_i \notin M$. Hence, $M \models C^M$.
Assume $\exists j\ D_j \in M$, then $C^M \notin P^M$.
Therefore, $\forall C^M \in P^M\ M \models C^M$, hence $M \models P^M$.

Minimality: Assume $\exists M' \subset M$ such that $M' \models P^M$. We prove that in this case $M' \cup \mathcal{C}M$ is a model of $P^{\mathcal{C}}$, contradicting the assumption that $M \cup \mathcal{C}M$ is a minimal model of $P^{\mathcal{C}}$.

Let $C = (A \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, not\ D_k)$ be a ground instance of a clause in P.

Assume $\exists j\ D_j \in M$, then $C^M \notin P^M$ but $\mathcal{C}D_j \in \mathcal{C}M$. Hence, $M' \cup \mathcal{C}M \models C_{\mathcal{C}}$.

Assume $\nexists j\ D_j \in M$, $C^M \in P^M$ and $M' \models C^M$. Hence, $A \in M'$ or $\exists i\ B_i \notin M'$, therefore $M' \models C_{\mathcal{C}}$.

$(M' \cup \mathcal{C}M) \models \mathcal{C}A \leftarrow A$, $\forall A \in HB_P$, since $M' \subset M$.

Hence, $(M' \cup \mathcal{C}M) \models P^{\mathcal{C}}$. $(\Rightarrow\Leftarrow)$

Therefore, M is a stable model of P. □

Using this new characterization, the process of computing the stable models of a function-free program P reduces to first iteratively computing the minimal models of the disjunctive logic program $P^{\mathcal{C}}$, and then selecting those minimal models that are consistent with the integrity constraints IC_P.

EXAMPLE 2

Let $P = \{a \leftarrow not\ b;\ b \leftarrow not\ a;\ c \leftarrow a;\ c \leftarrow b\}$. The *evidential transformation* produces the following program and set of integrity constraints:

$$P^{\mathcal{C}} = \{a \vee \mathcal{C}b; b \vee \mathcal{C}a;\ \ c \leftarrow a; c \leftarrow b;\ \ \mathcal{C}a \leftarrow a; \mathcal{C}b \leftarrow b; \mathcal{C}c \leftarrow c\},$$

$$IC_P = \{a \Leftarrow \mathcal{C}a; b \Leftarrow \mathcal{C}b; c \Leftarrow \mathcal{C}c\}.$$

The minimal models of the new program are $\mathcal{M}_{P^{\mathcal{C}}} = \{\{a, c, \mathcal{C}a, \mathcal{C}c\}, \{b, c, \mathcal{C}b, \mathcal{C}c\}, \{\mathcal{C}a, \mathcal{C}b\}\}$ and therefore $\mathcal{M}_{P^{\mathcal{C}}}^{IC_P} = \{\{a, c, \mathcal{C}a, \mathcal{C}c\}, \{b, c, \mathcal{C}b, \mathcal{C}c\}\}$.

The two stable models of P are $\{a, c\}$ and $\{b, c\}$.

From theorem 3.3 and lemma 2.10, we can characterize the stable model semantics in terms of the *EGCWA*. This definition gives us more intuition as to what it means for a model to be stable.

COROLLARY 3.4

M is a stable model of P iff $(M \cup \mathcal{C}M)$ is a model of $(P^{\mathcal{C}} \cup EGCWA(P^{\mathcal{C}}) \cup IC_P)$.

The importance of this characterization lies in the fact that it sheds light on what happens when programs do not have stable models.

COROLLARY 3.5

P does not have a stable model iff $(P^{\mathcal{C}} \cup EGCWA(P^{\mathcal{C}})$ does not satisfy the integrity constraints IC_P (i.e. $(P^{\mathcal{C}} \cup EGCWA(P^{\mathcal{C}}) \cup IC_P)$ is inconsistent).

In other words, in every minimal model of $P^{\mathcal{C}}$ there exists evidence considered true in that model and whose corresponding objective atom is false – the model claims to have evidence without the fact being true in the model.

EXAMPLE 3

Let $P = \{p \leftarrow not\ p\}$. The *evidential transformation* produces the program $P^{\mathcal{C}} = \{p \vee \mathcal{C}p; \mathcal{C}p \leftarrow p\}$ and the integrity constraints $IC_p = \{p \Leftarrow \mathcal{C}p\}$.

$$\mathcal{M}_{P^{\mathcal{C}}} = \{\{\mathcal{C}p\}\} \quad \text{and} \quad \mathcal{M}_{P^{\mathcal{C}}}^{IC_P} = \{\ \}.$$

Hence, P is inconsistent under the stable model semantics.

3.2. REDUCING THE SEARCH SPACE

The evidential transformation as defined above addresses two issues related to the problem of computing stable models. These are: to develop an iterative procedure that generates the set of models to be checked for stability and to devise a stability test for these models. In what follows, we concentrate our work on how to reduce the search space of models produced by the evidential transformation. It is well known [11] that stable models of logic programs are minimal models of the program as well (but the converse may not be true). Hence, an approach to computing stable models of a program is to first compute its minimal models and then check them for stability. However, a program may have numerous minimal models, only some of which are stable. We present below a new transformation that produces a stratified disjunctive program instead of a negation-free program. The utility of these stratified disjunctive programs in this paper is solely to use their structure to reduce the number of minimal models to be checked for stability. The obvious advantage of such a strategy is that the stability check can be performed less frequently if various minimal models are ruled out.

For stratified programs, we know that the perfect model semantics and the stable model semantics coincide (a model is perfect iff it is stable) [15]. The evidential transformation, on the other hand, may generate a disjunctive program with many more minimal models than those that are stable (possibly an exponential number of extra models). Hence, we intend to reduce the number of models to be checked by computing the perfect models of a stratified program instead of the minimal models of a negation-free program.

To illustrate our approach, let us analyze the structure of the predicate dependency graph of a normal logic program. A program is stratifiable if there is no recursion through negation, which means that there are no cycles in the predicate dependency graph of the program that involve dependencies. We will refer to these cycles as *negative cycles*. The cycles of the dependency graph can be determined by the strongly connected components of the graph. If one of them involves a negative

dependency, the program is not stratifiable. If we had a normal logic program that was stratified, then there would be no need for the evidential transformation. In this case, it suffices to find the perfect models of the program. The dependency graph for a stratifiable normal program has no negative cycles. If we now had a normal logic program that had a dependency graph with only one strongly connected component and such that it contains a negative cycle, then the evidential transformation has to be applied to every predicate. Now, we may have a predicate dependency graph such that some strongly connected components involve negative dependencies and others do not. In this case, we would like to apply the evidential transformation only where needed.

A stratification partitions the predicate symbols of a program using the following two rules:

1. If p and q belong to the same strongly connected component, then they belong to the same partition.
2. If predicate p depends negatively on predicate q and they do not belong to the same strongly connected component, then q belongs to a lower partition than p.

It is trivial to show that when the program is stratified, any partition that obeys these two rules produces a stratification of the program, since no strongly connected component involves negative dependencies. For non-stratifiable normal programs where some components involve negative dependencies, these rules generate what we call a *semi-stratification* of the program. Our objective is to find the stable models by the perfect models approach for those partitions (semi-stratum) that do not involve negative dependencies, and by the evidential transformation for those that do.

DEFINITION 3.6

Let P be a normal logic program. A *semi-stratification* $\{S_1, \ldots, S_r\}$ of P is a partition of the set of predicate symbols defined in P 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.

As for *stratification*, this partition induces a semi-stratification of the clauses of the program $\{P_1, \ldots, P_r\}$. It is also trivial to prove that any normal logic program has a *semi-stratification*. We only need to notice that any negative cycle in the graph of the program resides in a particular strongly connected component, and therefore all the predicate symbols involved in the cycle belong to the same semi-stratum. The *stratified evidential transformation* takes a semi-stratified logic program and produces a disjunctive stratified logic program and sets of integrity constraints for each strata. Formally, we define the transformation as follows.

DEFINITION 3.7

Let P be a normal logic program with semi-stratification $\{P_1, \ldots, P_r\}$. The *stratified evidential transformation of* P defines a stratified disjunctive logic program $P^{\mathcal{C}}$ with stratification $\{P_1^{\mathcal{C}}, \ldots, P_r^{\mathcal{C}}\}$ and a set of integrity constraints $IC_P = IC_{P_1} \cup \cdots \cup IC_{P_r}$ such that:

1. For each clause $A \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, D_k, not\ E_1, \ldots, not\ E_m$ of P_i, the clause $A \vee \mathcal{C}D_1 \vee \cdots \vee \mathcal{C}D_k \leftarrow B_1, \ldots, B_n, not\ E_1, \ldots, not\ E_m$ belongs to $P_i^{\mathcal{C}}$, where the predicate symbols of the D_l, $1 \leq l \leq k$, are defined in stratum i and the predicate symbols of the E_j, $1 \leq j \leq m$, are defined in the strata strictly below i.
2. For each predicate symbol p defined in P_i, the clause $\mathcal{C}p(x) \leftarrow p(x)$ belongs to $P_i^{\mathcal{C}}$.
3. For each predicate symbol defined in P_i, IC_{P_i} contains an integrity constraint of the form $p(x) \Leftarrow \mathcal{C}p(x)$.

Nothing else belongs either to $P_i^{\mathcal{C}}$ or IC_{P_i}.

In this new transformation, we substitute by evidences only those occurrences of negated literals that cannot be dealt with by the use of stratification techniques. We only modify normal semi-strata so that the recursions through negation are changed into an evidence. In this way, we obtain a stratified disjunctive logic program $P^{\mathcal{C}}$. Moreover, $\{P_1^{\mathcal{C}}, \ldots, P_r^{\mathcal{C}}\}$ constitutes a stratification of $P^{\mathcal{C}}$.

THEOREM 3.8

Let P be a normal logic program. M is a stable model of P iff $(M \cup \mathcal{C}M)$ is a stable model of $P^{\mathcal{C}}$.

Proof

Let C be a ground instance of a clause in P $(C = A \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, not\ D_k, not\ E_1, \ldots, E_m)$. We should look at the transformed programs P^M and $(P^{\mathcal{C}})^M$. Therefore, we denote the transformed versions of the clause C as follows:

- $C^M = (A \leftarrow B_1, \ldots, B_n)$,
- $C_{\mathcal{C}}^M = (A \vee \mathcal{C}D_1 \vee \cdots \vee \mathcal{C}D_k \leftarrow B_1, \ldots, B_n)$,

($\Rightarrow$) Let $N = M \cup \mathcal{C}M$. First we prove that N is a model of $(P^{\mathcal{C}})^N$. Then we prove that it is minimal.

Let $C = (A \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, not\ D_k, not\ E_1, \ldots, E_m)$ be a ground instance of a clause in P.

Assume $\nexists j\ D_j \in M$ and $\nexists l\ E_l \in M$. Since $M \models C^M$, then $A \in M$ or $\exists i\ B_i \notin M$. Hence, $N \models C_{\mathcal{C}}^N$.

Assume $\exists l\ E_l \in M$, then $C^M \notin P^M$ and $C_{\mathcal{C}}^N \notin (P^{\mathcal{C}})^N$.

Assume $\nexists l\ E_l \in M$ and $\exists j\ D_j \in M$, then $C^M \notin P^M$ but $\mathcal{C}D_j \in \mathcal{C}M$, hence $N \models C_{\mathcal{C}}^N$.

Consequently, $\forall C^N_{\mathcal{C}} \in (P^{\mathcal{C}})^N\ N \models C^N_{\mathcal{C}}$.
$N \models \mathcal{C}A \leftarrow A,\ \forall A \in HB_P$.
Therefore, $(M \cup \mathcal{C}M \models P^{\mathcal{C}}$.

Minimality: Assume $\exists N' \subset N$ such that $N' \models (P^{\mathcal{C}})^N$ and let $N' = M' \cup \mathcal{C}M''$. We prove that in this case M' is a model of P^M, contradicting the assumption that M is a minimal model of P^M.
Since $M' \cup \mathcal{C}M'' \models \mathcal{C}A \leftarrow A,\ \forall A \in HB_P$, then $M' \subseteq M'' \subseteq M$ and $M' \subset M$.
Let $C = (A \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, not\ D_k, not\ E_1, \ldots, not\ E_m)$ be a ground instance of a clause in P and $N' \models C^N_{\mathcal{C}}$.
Assume $\nexists j\ D_j \in M$ and $\nexists l\ E_l \in M$, then $A \in M'$ or $\exists i\ B_i \notin M'$. Hence, $M' \models C^M$.
Assume $\exists j\ D_j \in M$ or $\exists l\ E_l \in M$, then $C^M \notin P^M$.
Hence, $M' \models P^M$ since $\forall C^M \in P^M\ M' \models C^M$. $(\Rightarrow\Leftarrow)$

Therefore $(M \cup \mathcal{C}M)$ is a stable model of $P^{\mathcal{C}}$.

$(\Leftarrow)$ As before, we prove that M is a model of P^M and then that it is minimal.
Let $C = (A \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, not\ D_k, not\ E_1, \ldots, not\ E_m)$ be a ground instance of a clause in P.
Assume $\nexists l\ E_l \in M$, then $N \models C^N_{\mathcal{C}}$.
  Assume $\nexists j\ D_j \in M$, then $A \in M$ or $\exists i\ B_i \notin M$. Hence, $M \models C^M$.
  Assume $\exists j\ D_j \in M$, then $C^M \notin P^M$.
Assume $\exists l\ E_l \in M$, then $C^N_{\mathcal{C}} \notin (P^{\mathcal{C}})^N$ and $C^M \notin P^M$.
Therefore, $\forall C^M \in P^M\ M \models C^M$, hence, $M \models P^M$.

Minimality: Assume $\exists M' \subset M$ such that $M' \models P^M$. We prove that in this case $M' \cup \mathcal{C}$ is a model of $(P^{\mathcal{C}})^N$, contradicting the assumption that $M \cup \mathcal{C}M$ is a minimal model of $(P^{\mathcal{C}})^N$.
Let $C = (A \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, not\ D_k, not\ E_1, \ldots, not\ E_m)$ be a ground instance of a clause in P.
Assume $\nexists j\ D_j \in M$ and $\nexists l\ E_l \in M$. Since $C^M \in P^M$ and $M' \models C^M$, then $A \in M'$ or $\exists i\ B_i \notin M'$, therefore $M' \models C^N_{\mathcal{C}}$.
Assume $\exists j\ D_j \in M$ or $\exists l\ E_l \in M$, then $C^M \notin P^M$.
  Assume $\exists l\ E_l \in M$, then $C^N_{\mathcal{C}} \notin (P^{\mathcal{C}})^N$.
  Assume $\nexists l\ E_l \in M$, and then $\exists j\ \mathcal{C}D_j \in \mathcal{C}M$. Hence $(M' \cup \mathcal{C}M) \models C^N_{\mathcal{C}}$.
Hence, $M' \cup \mathcal{C}M \models C^N_{\mathcal{C}}$.
$(M' \cup \mathcal{C}M) \models \mathcal{C}A \leftarrow A,\ \forall A \in HB_P$, since $M' \subset M$.
Hence $(M' \cup \mathcal{C}M) \models (P^{\mathcal{C}})^N$. $(\Rightarrow\Leftarrow)$

Therefore, M is a stable model of P. □

Since $P^{\mathcal{C}}$ is a stratified program, we know that stable and perfect models coincide. Moreover, from lemma 3.2 we know that models of the form $(M \cup \mathcal{C}M)$ are the only models that satisfy the constraints defined by IC_P.

COROLLARY 3.9

Let P be a normal logic program. M is a stable model of P iff

$$(M \cup \mathcal{C}M) \in \mathcal{M}_{P^{\mathcal{C}}}^{IC_P},$$

where $\mathcal{M}_{P^{\mathcal{C}}}$ denotes the set of perfect models of the stratified program $P^{\mathcal{C}}$.

Hence, it is possible to compute the stable models of a normal program P by computing, using an iterative fixpoint operator, the perfect models of $P^{\mathcal{C}}$ and then eliminating those models that violate the integrity constraints in IC_P.

4. Fixpoint characterization of perfect models

In this section, we review the main results of Fernández and Minker [6] on devising a fixpoint operator to compute the perfect models of stratified disjunctive logic programs. Since disjunctive programs can have more than one minimal model, their fixpoint operator maps *sets of minimal interpretations* to *sets of minimal interpretations.*

DEFINITION 4.1 [6]

A *set of interpretations* $\mathcal{S}$ is called a *set of minimal interpretations* iff $\forall I \in \mathcal{S}$ $\nexists J \in \mathcal{S}, J \subset I$.

Using *sets of minimal interpretations*, Fernández and Minker define the partial order, $\sqsubseteq$.

DEFINITION 4.2 [6]

Let $\mathcal{J}$ and $\mathcal{S}$ be sets of minimal interpretations, $\mathcal{J} \sqsubseteq \mathcal{S}$ iff $\forall I \in \mathcal{S}, \exists J \in \mathcal{J}, J \subseteq I$.

This partial order leads to a very natural model semantics; if we move upward, we monotonically increase the set of positive clauses that are modeled by the sets of minimal interpretations.

THEOREM 4.3 [6]

Let $\mathcal{J}$ and $\mathcal{S}$ be sets of minimal interpretations. Then

$$\mathcal{J} \sqsubseteq \mathcal{S} \Leftrightarrow \forall C \text{ such that } C \text{ is a positive clause, } \mathcal{J} \models C \Rightarrow \mathcal{S} \models C,$$

where $\mathcal{S} \models C$ iff $\forall I \in \mathcal{S}, I \models C$.

Fernández and Minker define an operator T_P^M whose least fixpoint coincides with the minimal models of P, when P is a disjunctive logic program (negation free). Using this operator as a starting point, they define an iterative operator $T^M_{\langle P_1,\ldots,P_r\rangle}$ whose result is the set of perfect models for a stratified disjunctive program P.

DEFINITION 4.4 [6]

Let P be a normal disjunctive logic program and let $\mathcal{S}$ be a set of minimal interpretations.

$$T_P^M(\mathcal{S}) = \min\left(\bigcup_{I \in \mathcal{S}} models_I(state_P(I))\right),$$

where

$$state_P(I) = \{(A_1 \vee \cdots \vee A_k) : (A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, not\ D_m) \text{ is a ground instance of a clause in } P \text{ and } \forall i, B_i \in I \text{ and } \exists l, D_l \in I\},$$

$$models_I(S) = \{M \subseteq HB_P \mid M \text{ is a model of } (S \cup I)\},$$

$$min(\mathcal{S}) = \{I \in \mathcal{S} \mid \nexists I' \in \mathcal{S}, I' \subset I\},$$

where A_i, B_j and D_l are atoms.

$T_P^M(\mathcal{S})$ takes the set S_I of heads of all clauses in which the bodies are true in I for each $I \in \mathcal{S}$ and computes the models of $S_I \cup I$. Then from the union of all these models, it selects the minimal models. The operator T_P^M is monotonic when P is negation free. The ordinal powers of T_P^M are defined as follows.

DEFINITION 4.5 [6]

Let P be a normal disjunctive logic program.

$$T_P^M \uparrow 0(\mathcal{S}) = \mathcal{S},$$

$$T_P^M \uparrow \alpha(\mathcal{S}) = T_P^M(T_P^M \uparrow (\alpha - 1)(\mathcal{S})) \quad \text{if } \alpha \text{ is a successor ordinal,}$$

$$T_P^M \uparrow \alpha(\mathcal{S}) = lub\{T_P^M \uparrow \beta(\mathcal{S}) : \beta < \alpha\} \quad \text{if } \alpha \text{ is a limit ordinal.}$$

The operator $T_P^M \uparrow \alpha(\{\emptyset\})$ is denoted by $T_P^M \uparrow \alpha$ and if $T_P^M \uparrow \alpha(\mathcal{S}) = T_P^M \uparrow (\alpha + 1)(\mathcal{S})$, then α is called a *fixpoint ordinal* for $T_P^M(\mathcal{S})$.

For disjunctive logic programs (negation free), the fixpoint operator characterizes the set of minimal models of the program.

THEOREM 4.6 [6]

Let P be a disjunctive logic program and let $\mathcal{M}_P$ be the set of minimal models of P; then

$$\mathcal{M}_P = T_P^M \uparrow \alpha$$

for α a fixpoint ordinal.

The following example from [6] illustrates how the minimal models are computed using theorem 4.6.

EXAMPLE 4

Let $P = \{a \vee b;\ a \leftarrow b;\ c \leftarrow b\}$, then

1: $T_P^M(\{\emptyset\}) = \{\{a\}, \{b\}\}$ the minimal models of $a \vee b$,

2: $T_P^M(\{\{a\}, \{b\}\}) = \min(\{\{a\}, \{a, b, c\}\}) = \{\{a\}\}$ notice that $\{\{a\}, \{b\}\} \sqsubseteq \{\{a\}\}$,

3: $T_P^M(\{\{a\}\}) = \{\{a\}\}$ $\{\{a\}\}$ is a fixpoint.

For stratified disjunctive logic programs, Fernández and Minker define an iterative version of the T_P^M operator that is able to use the structure of the program in the same way as the *iter* operator of Apt, Blair and Walker for stratified definite logic programs [1].

DEFINITION 4.7 [6]

Let P be a stratified disjunctive logic program and let $\{P_1, \ldots, P_r\}$ be a stratification of P. Then

$$T^M_{\langle P_1\rangle} = T^M_{P_1} \uparrow \alpha \qquad \text{for } \alpha \text{ a fixpoint ordinal,}$$

$$T^M_{\langle P_1,\ldots,P_n,P_{n+1}\rangle} = T^M_{P_{n+1}} \uparrow \alpha(T^M_{\langle P_1,\ldots,P_n\rangle}) \quad \text{for } \alpha \text{ a fixpoint ordinal and } n > 0.$$

For each stratum $P_{i+1}$of the program, the operator $T^M_{\langle P_1,\ldots,P_i,P_{i+1}\rangle}$ computes for each model $M \in T^M_{\langle P_1,\ldots,P_i\rangle}$ the minimal models of $(\cup_{j=1}^{i+1} P_j)^M$. From the union of all the resulting minimal models, the operator selects the minimal ones. When applied to all the strata of the program, the resulting models are exactly the perfect models of P, as stated in the following theorem by Fernández and Minker.

THEOREM 4.8 [6]

Let P be a stratified disjunctive logic program and let $\{P_1, \ldots, P_r\}$ be a stratification of P. Then

$$\mathcal{M}_P = T^M_{\langle P_1,\ldots,P_r\rangle},$$

where $\mathcal{M}_P$ is the set of perfect models of P.

The fixpoint operator of Fernández and Minker provides us with a tool to compute iteratively the stable models of normal programs. Since the result of applying the *stratified evidential transformation* to a normal program P is a stratified disjunctive logic program $P^{\mathcal{E}}$, then the stable models of P can be computed using the iterative fixpoint and selecting those models that are consistent with the integrity constraint IC_P.

COROLLARY 4.9

Let P be a normal logic program and let $\{P_1, \ldots, P_r\}$ be a semi-stratification of P. Then M is a stable model of P iff

$$(M \cup \mathcal{C}M) \in \left\{ N \in T^{M}_{\langle P_1^{\mathcal{C}}, \ldots, P_r^{\mathcal{C}} \rangle} : N \models IC_P \right\},$$

where $\{P_1^{\mathcal{C}}, \ldots, P_r^{\mathcal{C}}\}$ conform a stratification of $P^{\mathcal{C}}$.

5. Computing the stable models of normal deductive databases

A *normal deductive database* (*NDDB*), *DB*, is a function-free logic program whose clauses are range restricted and safe. By range restricted we mean that any variable that occurs in the head of the clause also occurs in its body. By safe we mean that any variable occurring in a negative literal in a clause body also occurs in a positive literal in the body of the clause. NDDBs defined in this way have a finite Herbrand base. Similarly, a *disjunctive stratified database* (DSDB for short) is a function-free stratified disjunctive logic program whose clauses are range restricted and safe.

To compute the stable models of an NDDB, we can apply the *stratified evidential transformation* defined in the previous section and in this way create a new disjunctive stratified database (DSDB)[3], $DB^{\mathcal{C}}$, and a set of integrity constraints, IC_{DB}. In the rest of this section, we present algorithms to compute the perfect models of disjunctive deductive databases and its restriction by a set of integrity constraints. These algorithms, when applied to the transformed program, will allow the computation of the stable models of an NDDB.

Fernández and Minker have presented algorithms for computing, in a bottom-up fashion, the model semantics of disjunctive databases – hierarchic [5], recursive [4], and stratified [6]. Their approach is based on the use of a new data structure called a *model tree* to represent the minimal models being computed. Here, we present how these algorithms can be extended to compute the stable models of *normal deductive databases*.

5.1. COMPUTING MINIMAL MODELS OF A DSDB

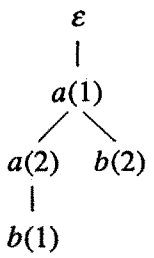
In this section, we review the results of Fernández and Minker on computing the perfect models of a DSDB. Roughly, a *model tree* for a set of minimal interpretations $\mathcal{I}$ is a tree structure representing all the interpretations in $\mathcal{I}$ such that each node of the tree is labeled by an atom that occurs in $\mathcal{I}$, and each branch of the tree (the

[3] Notice that the safeness condition imposed over the NDDB guarantees that the resulting DSDB will be range restricted and safe.

atoms in the path from the root to a leaf node) represents a different interpretation of $\mathcal{S}$. The special symbol ε labels the root of the tree and it represents no atom.

EXAMPLE 5

A model tree for the minimal models of example 1 would be as follows:



Fernández and Minker [5,6] developed a simple and general algorithm for the evaluation of clauses in these tree structures. The computation of the minimal models of a DSDB with recursive definitions can be accomplished by applying algorithm 1 to each rule in the database, and repeating this process until no modifications are performed on the tree. The fact that the Herbrand base for databases is finite guarantees the termination of the iterative process.

ALGORITHM 1 [6]

Let $A_1 \vee \cdots \vee A_k \leftarrow B_1, \ldots, B_n, not\ D_1, \ldots, not\ D_m$ be a DSDB clause with $k \geq 1$ and $n, m \geq 0$ and $\mathcal{T}_{\mathcal{S}}$ be a model tree for a set of minimal interpretations $\mathcal{S}$.

1. Let $\mathcal{D}_{\mathcal{S}} = (\bigcup_{I \in \mathcal{S}} I)$ (i.e. the set of atoms occurring in $\mathcal{T}_{\mathcal{S}}$).
2. Compute $J = \{\theta \mid \theta$ is ground and $\forall i, B_i\theta \in \mathcal{D}_{\mathcal{S}}\}$.[4)]
3. For each $\theta \in J$
   For each branch b of $\mathcal{T}_{\mathcal{S}}$
   If $\forall i, B_i\theta$ occurs in b and $\exists j, A_j\theta$ occurs in b and $\exists l, D_l\theta$ occurs in b, then
   Add to $\mathcal{T}_{\mathcal{S}}$ new leaf nodes $A_1\theta, \ldots, A_k\theta$ as children of the leaf node of b.
4. Eliminate any non-minimal branch.

Algorithm 1 uses an auxiliary data structure $\mathcal{D}_{\mathcal{S}}$ to keep track of those atoms already in the model tree. If we consider $\mathcal{D}_{\mathcal{S}}$ as the dictionary of data stored in $\mathcal{T}_{\mathcal{S}}$, then the computation of the set J is equivalent to the computation, in a definite

database, of the complex *join* operation[4] represented by $\leftarrow B_1, \ldots, B_n$. J covers the substitutions that can trigger the inclusion of an atom $A_j\theta$ in a model. Hence, the number of elements in the set J is an upper limit to the set of substitutions that need to be checked in order to determine what atoms must be added to the interpretations. J reduces the number of grounded rules to be inspected by the algorithm in step 3.

For disjunctive stratified databases, we compute the perfect models of the DSDB by iterating algorithm 1 as before, on each *stratum* of the databases as in algorithm 2.

ALGORITHM 2 [6]

Let $\{DB_1, \ldots, DB_r\}$ be a stratification for a DSDB, DB.

1. Let $T \leftarrow \varepsilon$.
2. For $i = 1$ to r do
   Let $DB_i = \{C_1, \ldots, C_m\}$
   Repeat
   For $j = 1$ to m do $T \leftarrow T_{C_j}^M(T)$ using algorithm 1.
   Until no modification is performed on T.
3. Return T.

Fernández and Minker show that algorithm 2 is correct [6]; the resulting tree represents the set of perfect models of the database.

5.2. APPLYING INTEGRITY CONSTRAINTS

In this section, we present algorithms for the application of integrity constraints and show how they can be used to extend algorithm 2 for the computation of the stable models of *normal deductive databases*.

A minimal model M of a DSDB violates an integrity constraint of the form $A_1 \vee \cdots \vee A_k \Leftarrow B_1, \ldots, B_n$ iff M is not a model of the constraint. That is, $\forall i\, B_i \in M$ and $\nexists j\, A_j \in M$. Algorithm 3 filters those models that violate an integrity constraint by removing the corresponding branch in the model tree.

ALGORITHM 3

Let $\mathcal{T}_\$$ be a model tree for a set of minimal interpretations $\$$ and $A_1 \vee \cdots \vee A_k \Leftarrow B_1, \ldots, B_n$ be an integrity constraint with $k + n > 0$.

1. Let $D_\$ = (\bigcup_{I \in \$} I)$ (i.e. the set of atoms occurring in $\mathcal{T}_\$$).
2. Compute $J = \{\theta \mid \theta \text{ is a ground and } \forall i, B_i\theta \in D_\$\}$.

[4] The substitution in J only involves variables that occur in the B_i's. If no variables occur in the B_i's, then J contains only the identity substitution.

3. For each $\theta \in J$
   For each branch b of $\mathcal{T}_{\mathcal{S}}$
   If $\forall i, B_i\theta$ occurs in b and $\exists j, A_j\theta$ occurs in b then
   Remove branch b from the tree.

The stable models of a normal deductive database DB can be computed by conducting the following steps.

1. Use the *stratified evidential transformation* to create the DSDB $DB^{\mathcal{C}}$ and the set of integrity constraints IC_{DB} from the original database DB.
2. Use algorithm 2 to compute a model tree $\mathcal{T}_{DB^{\mathcal{C}}}$ representing the set of perfect models of $DB^{\mathcal{C}}$.
3. Apply algorithm 3 to $\mathcal{T}_{DB^{\mathcal{C}}}$ with every integrity constraint in IC_{DB} to eliminate any model violating an integrity constraint.

The resulting tree represents the set of stable models of the original deductive database DB. If the tree is empty (all branches have been removed), then DB has no stable model and is inconsistent with respect to the stable model semantics.

Some improvements can be made to this strategy. Notice that any violation of an integrity constraint (i.e. $p(x) \Leftarrow \mathcal{C}p(x)$) can be checked after all the rules defining the predicates $p(x)$ and $\mathcal{C}p(x)$ have been evaluated (after the stratum containing these predicates has been processed). Since by construction of $P^{\mathcal{C}}$ these predicates belong to the same stratum, we can alternate the computation of the perfect models of each stratum with the verification of the constraints. In this way, we eliminate unstable models as soon as possible. Moreover, if during the verification of a stratum all models are eliminated, we can stop the computation since the program is inconsistent with respect to the stable model semantics.

Algorithm 4 uses this alternating approach for the computation of the consistent perfect models of $DB^{\mathcal{C}}$ with respect to IC_P.

ALGORITHM 4

Assume $\{DB_1, \ldots, DB_r\}$ is a stratification for a DSDB, DB and $IC_{DB_1}, \ldots, IC_{DB_r}$ are sets of integrity constraints for each stratum of DB.

1. Let $T \leftarrow \varepsilon$.
2. For $i = 1$ to r do
   Let $DB_i = \{C_1, \ldots, C_m\}$ and let $IC_{DB_i} = \{S_1, \ldots, S_t\}$
   Repeat
   For $j = 1$ to m do $T \leftarrow T_{C_j}^{M}(T)$ using algorithm 1.
   Until no modification is performed on T.
   For $j = 1$ to t do $T \leftarrow (T)^{S_j}$ using algorithm 3.
   If T is empty then return T.
3. Return T.

EXAMPLE 6

Let DB be an NDDB with the following semi-stratification:

$DB_1 = \{v(a); u(a); q(a);\}$

$DB_2 = \{r(X) \leftarrow t(X), not\ s(X); s(X) \leftarrow u(X), not\ r(X); t(X) \leftarrow v(X), not\ w(X);\}$

$DB_3 = \{p(X) \leftarrow q(X), not\ r(X); p(X) \leftarrow q(X), not\ s(X);\}$

The *stratified evidential transformation* produces the *disjunctive stratified databases*, *integrity constraints* and *model tree* for each semi-stratum that is shown in fig. 1.

The first semi-stratum produces a definite program and therefore a model tree representing one minimal model. The second semi-stratum produces a disjunctive stratum with three different minimal models. Out of these models, only two are consistent with the integrity constraints. Finally, in the third stratum, we use each one of the stable models of the previous stratum to generate the two stable models of $DB^{\mathcal{E}}$ which represent the stable models $\{v(a), u(a), q(a), t(a), r(a), p(a)\}$ and $\{v(a), u(a), q(a), t(a), s(a), p(a)\}$ in DB.

For programs without stable models, our approach to stable model semantics is useful in detecting the cause of inconsistencies. The following example shows what happens when a program does not have stable models.

EXAMPLE 7

Let $DB = \{w(X) \leftarrow m(X, Y), not\ w(Y); m(a, b); m(b, c); m(d, d)\}$; then

$\mathcal{E}$-transformation[5)]

$w(X) \vee \mathcal{E}w(Y) \leftarrow m(X,Y).$

$m(a,b).$

$m(b,c).$

$m(d,d).$

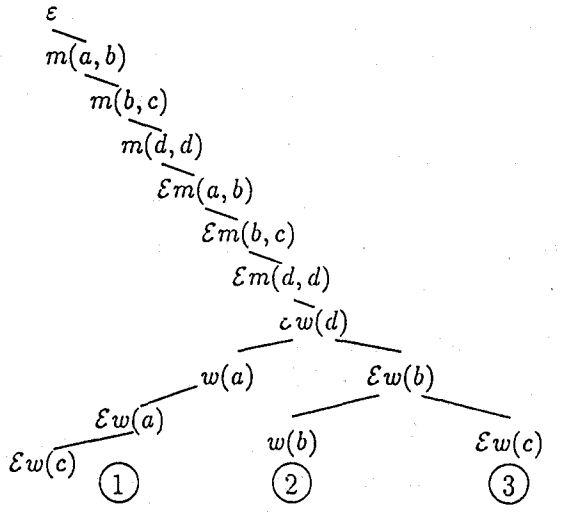
$\mathcal{E}w(X) \leftarrow w(X).$

$\mathcal{E}m(X,Y) \leftarrow m(X,Y).$

$w(X) \Leftarrow \mathcal{E}w(X).$

$m(X,Y) \Leftarrow \mathcal{E}m(X,Y).$

Model tree for $\mathcal{M}_{DB^{\mathcal{E}}}$



[5)] Notice that in this figure as well as in fig. 1, the symbol $\mathcal{E}$ is represented by the symbol ε.

Semi-stratum 1	Semi-stratum 2	Semi-stratum 3
$v(a).$	$r(X) \vee \mathcal{E}s(X) \leftarrow t(X).$	$p(X) \leftarrow q(X),\ not\ r(X).$
$u(a).$	$s(X) \vee \mathcal{E}r(X) \leftarrow u(X).$	$p(X) \leftarrow q(X),\ not\ s(X).$
$q(a).$	$t(X) \leftarrow v(X),\ not\ w(X).$	
$\mathcal{E}v(X) \leftarrow v(X).$	$\mathcal{E}r(X) \leftarrow r(X).$	$\mathcal{E}p(X) \leftarrow p(X).$
$\mathcal{E}u(X) \leftarrow u(X).$	$\mathcal{E}s(X) \leftarrow s(X).$	
$\mathcal{E}q(X) \leftarrow q(X).$	$\mathcal{E}t(X) \leftarrow t(X).$	
$\mathcal{E}w(X) \leftarrow w(X).$		
$v(X) \Leftarrow \mathcal{E}v(X).$	$r(X) \Leftarrow \mathcal{E}r(X).$	$p(X) \Leftarrow \mathcal{E}p(X).$
$u(X) \Leftarrow \mathcal{E}u(X).$	$s(X) \Leftarrow \mathcal{E}s(X).$	
$q(X) \Leftarrow \mathcal{E}q(X).$	$t(X) \Leftarrow \mathcal{E}t(X).$	
$w(X) \Leftarrow \mathcal{E}w(X).$		

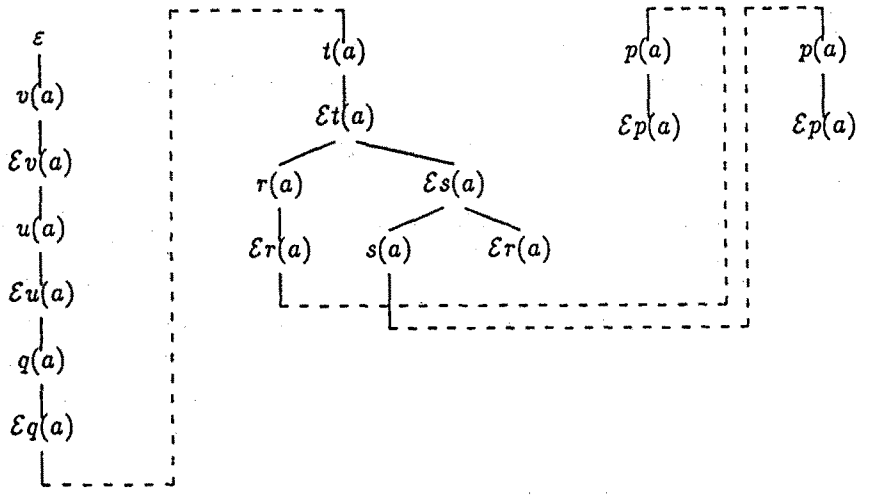


Fig. 1. Computing stable models in a semi-stratified database.

there is no stable model since no minimal model of $DB^{\mathcal{C}}$ satisfies the integrity constraints.

In example 7, the violated constraints are: $w(c) \Leftarrow \mathcal{C}w(c)$, in branches 1 and 3; $w(b) \Leftarrow \mathcal{C}w(b)$, in brach 3; and $w(d) \Leftarrow \mathcal{C}w(d)$ in all branches. It is clear that if we eliminate the violation of the third constraint, the theory would have stable

models. By tracking down why $\mathcal{C}w(d)$ is true, we can see that it comes from the fact that $m(d, d)$ is true. Removing that fact $m(d, d)$ from the program restores its consistency.

The integrity constraints can be used in this way to detect and correct inconsistencies in a normal logic program under the stable model semantics. How this can be achieved in the general case is a problem for future research.

6. Related work

In the four years since the development of stable models by Gelfond and Lifschitz [8], a great deal of work has been done on the declarative aspects of stable models, and the relationships between stable models and other non-monotonic logics [11,12,15]. In comparison, relatively few techniques have been developed for the computation and implementation of non-monotonic logic programming.

Cuadrado and Pimentel [3] show how to compute stable models of propositional logic programs. Their technique iteratively constructs a *set* of labeled trees for computing stable models. The node labels are sets of interpretations. Eventually, a tree is generated whose labels capture the stable models of P. However, to our knowledge, Cuadrado and Pimentel do not provide a proof that their procedure computes the set of all stable models of a given program. Our methods are sound and complete.

Fuentes [7] presents an algorithm for computing a stable model of a program. This work does *not* compute all stable models of a program. Thus, even though a program P may have n stable models where $n > 1$, his procedure would stop after one stable model is found. It is not clear how this technique extends to the computation of all stable models.

The LOPS project [2] is one of the first attempts to seriously study computation and implementation of non-monotonic logic programming. In ref. [2], a uniform framework for comparing, both theoretically and experimentally, three alternative strategies for computing non-monotonic logic programming is presented. The authors develop a prototype implementation of three techniques for stable model computation and compare and contrast these alternative strategies based both on theoretical arguments and on experimental results. The stratified evidential transformation described in this paper (definition 3.7) makes use of the dependency graphs of programs to prune the search space. We believe transformations of this kind can be used to enhance the computational methods described in [2].

Finally, concurrently with this effort, Warren [19] is working on a similar problem. His work attempts to modify OLD-resolution with tabulation and negation as failure for computing stable models. His procedure is a run-time computation procedure: given a query Q, it is possible to determine whether Q is true in some stable model of the program. Our method, on the contrary, computes all stable models of a program. Furthermore, our methods are query independent.

Given, as input, a logic program P and a set S of Herbrand interpretations, determining whether S is the set of all stable models of P is known to be NP-hard. Hence, all complete algorithms addressing this problem would be non-polynomial, unless $P = NP$. Hence, one way to compare alternative approaches is via implementation and experimentation.

7. Conclusions

The major result of this paper is the equivalence between stable models of normal logic programs and the minimal models of a transformed program that are consistent with a set of integrity constraints as defined by the transformations. The transformation to a negation-free disjunctive logic program allowed us to present a characterization of the stable model semantics for normal logic programs in terms of the EGCWA and integrity constraints. On the other hand, the transformation to a stratified disjunctive logic program allowed us to describe a procedure to compute the stable models of function-free programs. The algorithm is developed using the concept of model trees introduced by Fernández and Minker [5,6] in the context of disjunctive stratified databases and extends their previous algorithm to cover the presence of integrity constraints in the stratified program. All these results extend naturally to normal disjunctive logic programs.

In our approach, the detection of inconsistency implies the lack of stable models. This inconsistency is introduced by the integrity constraints that are produced by the evidential transformation of the original normal logic program. For the class of normal deductive databases, we can identify the set of constraints that are possible causes of the inconsistency and through them, it may be possible to isolate the part of the original program that caused the non-existence of stable models. This information can be used either to correct the program or to compute answers that do not depend on the unstable part of the theory. Techniques by which this can be accomplished are a topic of future research.

Finally, we recognize that, for disjunctive programs, the two notions of integrity constraint satisfaction (consistency and entailment) differ. We believe that Kowalski's definition allows a powerful role for integrity constraints. Under the consistency satisfaction interpretation, constraints augment the expressive power of logic programming. Although we present algorithms to handle integrity constraints, these algorithms process information in a bottom-up fashion, which restricts their usage to a limited class of programs. Another direction of research is the study of top-down algorithms for processing integrity constraints in a larger class of logic programs.

Acknowledgements

We greatly appreciate the financial support of the Air Force Office of Scientific Research, provided under Grant No. AFOSR-91-0350, the Army Research Office under Grant No. DAAL-03-92-G-0225, and the National Science Foundation, provided

under Grant Nos. IRI-89-16059 and IRI-91-09755, that made this work possible. We are grateful to the referees for their useful comments.

References

[1] K.R. Apt, H.A. Blair and A. Walker, Towards a theory of declarative knowledge, in: *Foundations of Deductive Databases and Logic Programming*, ed. J. Minker (Morgan Kaufmann, Washington, D.C., 1988) pp. 89–148.

[2] 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.

[3] J. Cuadrado and S. Pimentel, A truth maintenance system based on stable models, in: *Proc. 1989 North American Conf. on Logic Programming* (1989) pp. 274–290.

[4] J.A. Fernández and J. Minker, Bottom-up evaluation of disjunctive deductive databases, Technical Report, Computer Science Department, University of Maryland (1991), in preparation.

[5] J.A. Fernández and J. Minker, Bottom-up evaluation of hierarchical disjunctive deductive databases, in: *Proc. 8th Int. Conf. on Logic Programming*, ed. K. Furukawa (MIT Press, 1991).

[6] J.A. Fernández and J. Minker, Computing perfect models of disjunctive stratified databases, in: *Proc. ILPS'91 Workshop on Disjunctive Logic Programs*, San Diego, CA (October 1991), ed. D. Loveland et al., pp. 110–117. An extended version has been submitted to Annals of Mathematical Artificial Intelligence.

[7] L.O. Fuentes, Applying uncertainty formalisms to well-defined problems, Master's Thesis, University of Texas at El Paso (1991).

[8] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming, in: *Proc. 5th Int. Conf. and Symp. on Logic Programming*, Seattle, Washington (August, 1988), ed. R.A. Kowalski and K.A. Bowen, pp. 1070–1080.

[9] R.A. Kowalski, Logic for data description, in: *Logic and Data Bases*, ed. H. Gallaire and J. Minker (Plenum Press, New York, 1978) pp. 77–102.

[10] J. McCarthy, Circumscription – a form of non-monotonic reasoning, Artificial Intelligence 13(1980) 27–39.

[11] W. Marek and V.S. Subrahmanian, The relationship between stable, supported, default and auto-epistemic semantics for general logic programs, Theor. Comp. Sci. 10(3)(1992)365–386.

[12] W. Marek and M. Truszczynski, Stable semantics for logic programs and default theories, in: *Proc. 1989 North American Conf. on Logic Programming*, ed. E. Lusk and R. Overbeek (MIT Press, 1989) pp. 243–256.

[13] T.C. Przymusinski, On the semantics of stratified deductive databases, in: *Proc. Workshop on Foundations of Deductive Databases and Logic Programming*, ed. J. Minker, Washington, D.C. (1986) pp. 433–443.

[14] T.C. Przymusinki, Extended stable semantics for normal and disjunctive programs, in: *Proc. 7th Int. Conf. on Logic Programming*, Jerusalem, 1990 (MIT Press), Extended Abstracts, pp. 459–477.

[15] T.C. Przymusinski, Stable semantics for disjunctive programs, New Generation Comput. J. 9(1991) 401–424. Extended Abstract appeared in [14].

[16] R. Reiter, Towards a logical reconstruction of relational database theory, in: *On Conceptual Modelling*, ed. M.L. Brodie et al. (Springer, New York, 1984) pp. 163–189.

[17] R. Reiter, What should a database know? in: *Proc. 7th Int. Conf. on Logic Programming*, Jerusalem (1990), Abstract of Invited Talk, p. 765.

[18] M.H. van Emden and R.A. Kowalski, The semantics of predicate logic as a programming language, J. ACM 23(1976)733–742.

[19] D.S. Warren, Using OLDT evaluation with meta-interpreters, in: *Logic in Databases, Knowledge, Representation and Reasoning*: A Symposium in Honor of Jack Minker's 65th Birthday for Contributions to Computer Science and Human Rights, College Park, Maryland (Nov. 1992).

[20] A. Yahya and L.J. Henschen, Deduction in non-Horn databases, J. Aut. Reasoning 1(1985)141–160.