ELSEVIER

Artificial Intelligence 96 (1997) 421–449

Artificial Intelligence

Reasoning with minimal models: efficient algorithms and applications [1]

Rachel Ben-Eliyahu-Zohary [a,*], Luigi Palopoli [b,2]

[a] *Mathematics and Computer Science Department, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel*

[b] *DEIS, Universitá della Calabria, 87030 Rende (CS), Italy*

Received September 1996

Abstract

Reasoning with minimal models is at the heart of many knowledge-representation systems. Yet it turns out that this task is formidable, even when very simple theories are considered. In this paper, we introduce the *elimination algorithm*, which performs, in linear time, minimal model finding and minimal model checking for a significant subclass of positive CNF theories which we call positive head-cycle-free (HCF) theories. We also prove that the task of minimal entailment is easier for positive HCF theories than it is for the class of all positive CNF theories. Finally, we show how variations of the elimination algorithm can be applied to allow queries posed on disjunctive deductive databases and disjunctive default theories to be answered in an efficient way.

Keywords: Minimal models; Disjunctive databases; Disjunctive default logic; Disjunctive logic programs; Stable model semantics; Linear time algorithms

1. Introduction

Computing minimal models is an essential task in many reasoning systems in artificial intelligence, including circumscription [29–31], default logic [39], and minimal

* Corresponding author. Email: rachel@cs.bgu.ac.il.

[1] Part of this work was done while the first author was visiting the Cognitive Systems Laboratory, Computer Science Department, University of California, Los Angeles, CA, USA, and the second author was a visiting scholar at the Computer Science Department, University of California, Los Angeles, CA, USA. This is an extended and revised version of a paper presented at the Fourth International Conference on Principles of Knowledge Representation and Reasoning, Bonn, Germany, 1994.

[2] Email: luigi@si.deis.unical.it.

0004-3702/97/$17.00
PII S0004-3702(97)00060-X

diagnosis [12], and in answering queries posed on logic programs (under stable model semantics [4,21]) and deductive databases (under the generalized closed-world assumption [32]). In such reasoning systems, the goal is to produce plausible inferences or plausible explanations, not to compute minimal models. Nonetheless, efficient algorithms for computing minimal models can substantially speed up inference in these systems.

Surprisingly, and perhaps due to its inherent difficulty, reasoning with minimal models has received a formal analysis only recently [6,9–11,18,26,34]. Given a propositional CNF theory T and a literal L in T, the following tasks (and others) have been considered: [3]

- *Model finding.* Find a minimal model for T.
- *Model checking.* Check whether a given interpretation is a minimal model for T.
- *Minimal entailment.* Is L *true* in all the minimal models of T?
- *Minimal membership.* Is L *true* in at least one minimal model of T?

Unfortunately, the results of the formal work on the complexities of reasoning with minimal models are discouraging. It turns out that even when the theory is positive, that is, when the theory has no clause in which all the literals are negative, these questions are very hard to answer: model finding is $P^{NP[O(\log n)]}$-hard [10] (and positive theories always have a minimal model!) [4], model checking is co-NP-complete [9], minimal entailment is Π_2^p-complete, and minimal membership is Σ_2^p-complete [18].

In this paper, we exploit a basic property that turns out to characterize a subclass of all CNF theories for which the above and related problems can be solved efficiently. The property is *head-cycle-freeness* [7]. The idea is simple. A clause [5] is viewed as having a direction—from the negative to the positive literals—and this direction is made explicit in the way clauses are represented in logic programs. We then associate a dependency graph with each theory: each atom and each clause is a node in the graph; there is an arc directed from an atom A to a clause δ if and only if A appears negative in δ, and there is an arc directed from a clause δ to an atom A if and only if A appears positive in δ. A CNF theory will be called *head-cycle-free* (HCF) if and only if in its dependency graph there is no directed cycle that goes through two different atoms that appear positive in the same clause. Head-cycle-freeness can be checked in time linear in the size of the theory.

We show that for positive HCF theories, the above problems are more manageable than they are in the general case: model finding and model checking can be done in linear time in the size of the theory, minimal entailment is co-NP-complete [20], and minimal membership is NP-complete. The complexity results for propositional theories are summarized in Fig. 1. Entries without a reference number indicate results presented in this paper.

[3] See the next section for formal definitions of *minimal model, interpretation,* and *literal.*

[4] We recall that $P^{NP[O(\log n)]}$ is the class of decision problems that are solved by polynomial-time bounded deterministic Turing machines making at most a logarithmic number of calls to an oracle in NP. For a precise characterization of the complexity of model finding, given in terms of complexity classes of functions, see [11].

[5] In this section, a clause is a disjunction of literals. In the following sections we use a different syntax.

Language	Model checking	Model finding	Minimal entailment	Minimal membership
CNF	co-NP-complete [9]	NPMV//OptP[O(log n)]-complete [11]	Π_2^p-complete [18]	Σ_2^p-complete [18]
positive HCF	O(n)	O(n)	co-NP-complete [20]	NP-complete

Fig. 1. Complexity of computational tasks with minimal models.

Our algorithms can be generalized to allow efficient computation of minimal Herbrand models for a significant subclass of positive first-order CNF theories. In addition, we apply our results on CNF theories to answering queries on deductive databases that use disjunctive rules. Specifically, we provide a polynomial-time data complexity algorithm [44] that computes a stable model for a stratified HCF disjunctive deductive database[6]. This algorithm also can be used to answer queries posed on disjunctive default theories.

HCF theories form a relevant fragment of CNF theories in that seemingly they can represent meaningful knowledge about the world [16,19,27]. The relevance of HCF theories is formally confirmed in [27], where their expressive power is precisely stated.

The rest of this paper is organized as follows. In Section 2, we present the elimination algorithm which performs minimal model finding for propositional positive HCF theories, and consider the other tasks of minimal model checking, minimal membership, and minimal entailment for this class. In Section 3, we generalize the elimination algorithm for the class of function-free first-order positive HCF theories. In Section 4, we demonstrate some applications of the elimination algorithm in knowledge-representation systems. Related work is discussed in Section 5, and conclusions are presented in Section 6.

2. The elimination algorithm for positive HCF theories

In this section, we introduce the *elimination algorithm* (EA), which can be used to perform minimal model finding on a propositional positive HCF theory in linear time. We will also establish the complexity of minimal model checking, minimal entailment, and minimal membership for propositional positive HCF theories.

We define a theory T to be a set of clauses of the form

$$A_1 \wedge A_2 \wedge \cdots \wedge A_m \rightarrow C_1 \vee C_2 \vee \cdots \vee C_n, \tag{1}$$

where $n, m \geqslant 0$ and all the A's and the C's are atoms.[7] The expression to the left of $\rightarrow$ is called the *body* of the clause, while the expression to the right of $\rightarrow$ is called the *head* of the clause. We assume that all of the Cs are distinct. A theory T is called *positive* if, for every clause, $n > 0$. In this section, we deal with positive theories, unless we state otherwise.

[6] *Stable models* and *stratified disjunctive deductive databases* will be defined in the following sections.

[7] Note that the syntax of (1) is a bit unusual for a clause; usually, the equivalent notation $\neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_m \vee C_1 \vee C_2 \vee \cdots \vee C_n$ is used.

A set of atoms *satisfies the body of a clause* if and only if all the atoms in the body of the clause belong to this set. A set of atoms *violates a clause* if and only if the set satisfies the body of the clause but none of the atoms in the head of the clause belongs to the set. A set of atoms X is a *model* of a theory T if none of its clauses is violated by X. A model X of a theory T is *minimal* if there is no $Y \subset X$ that is also a model of T. An *interpretation* for a theory T is an assignment of truth values to the atoms in T. Interpretations and models will be represented by the set of atoms being assigned the value **true**. A *literal* is an atom ("positive" literal) or a negated atom ("negative" literal).

With every theory T we associate a directed graph G_T, called the *dependency graph*[8] of T, in which (a) each atom A and each clause δ in T is a node and (b) there is an arc from A to δ if and only if A is in the body of δ and an arc from δ to A if and only if A is in the head of δ.

As mentioned before, model finding for positive theories is $\mathrm{P}^{\mathrm{NP}[\mathrm{O}(\log n)]}$-hard, model checking is co-NP-complete, model entailment is Π_2^p-complete, and minimal membership is Σ_2^p-complete. From the work of [7] and the results presented here it follows that these problems are easier for the class of positive HCF theories. A theory T is HCF if and only if there is no clause in T such that for some C_i and C_j, $i \neq j$, G_T contains a directed cycle involving C_i and C_j. So, for example, the theory $A \rightarrow B$, $B \rightarrow A$, $A \vee B$ is *not* HCF, while the theory $A \rightarrow B$, $B \rightarrow A$, $A \vee C$ *is* HCF.

Proposition 1. *Head-cycle-freeness of a propositional theory T of size n can be checked in time* $\mathrm{O}(n)$.[9]

Proof. The algorithm for checking head-cycle-freeness consists of three steps:

1. Construct G_T, the dependency graph of the theory.
2. Identify the strongly connected components. This can be done in time linear in the size of G_T [43].
3. For every clause, check whether there are two atoms in its head that belong to the same component. This can be done by assigning to each atom the number of its component and then checking whether the same number appears twice in some head.

Since each step is done in linear time, the whole algorithm can be done in linear time. □

2.1. Model finding

Clearly, just any model for a positive theory can be found very easily—take, for example, the set of all atoms in the theory. What is difficult is finding a *minimal* model for the theory. Roughly speaking, the idea behind the EA, shown in Fig. 2, is as follows: we pick a model of the theory and then *eliminate* from this model all the atoms that

[8] In [7] a different definition of dependency graph is given, but the two definitions are equivalent when defining HCF theories. The dependency graph as defined here has the advantage that it can be constructed in linear time.

[9] Throughout the paper, the size of a theory is the number of symbols (characters) it contains.

EA(T)

Input: A positive HCF theory T.
Output: A minimal model for T.

1. M :=a model of T; $M' := \emptyset$.
2. Let Δ be the set of all clauses δ in T violated by M' such that $|head(\delta, M)| = 1$.
   If $\Delta = \emptyset$, go to Step 3.
   Else, let $X := \bigcap_{\delta \in \Delta} head(\delta, M)$; $M' := M' \cap X$; $M := M - X$;
   repeat Step 2.
3. Let Δ be the set of all clauses δ in T violated by M' such that for each $\delta \in \Delta$, $|head(\delta, M)| \geqslant 2$.
   If $\Delta = \emptyset$, return M'.
   Else, let $H := \bigcap_{\delta \in \Delta} head(\delta, M)$ and let X be a source of H in G_T; let $M := M - X$; go to Step 2.

Fig. 2. The elimination algorithm for positive HCF theories.

we know will not be part of one of the minimal models that are subsets of this model (hence, the name of the algorithm).

Given a directed graph G and a set Y of nodes in G, a set X of nodes in G will be called a *source of* Y if and only if (a) $X \cap Y$ is not empty, (b) all the nodes in X are in the same strongly connected component in G,[10] and (c) for each node A in $Y - X$, there is no directed path in G from A to any of the nodes in X. Intuitively, if X is a source of Y in a dependency graph of some theory, then none of the atoms in $Y - X$ can be used to derive any of the atoms in X. At the generic step of the execution of the EA, we have a "current" (non-minimal) model M of the input theory. Then we eliminate subsets of atoms from M in a way that prevents elimination of atoms that turn out to be part of a minimal model. To this end, if S is the set of atoms that are candidates for elimination, we delete a *source* of S. At Step 3 of the EA, we delete a source of the set of all atoms in the heads of clauses violated by M'. Since each such clause has at least two atoms in its head, and since the theory is HCF, it is always the case that each clause in this set has in its head atoms that belong to at least two different strongly connected components in the dependency graph of the theory. Therefore, we can always find a nonempty source in Step 3 of the EA.

Given a set X, $|X|$ denotes the cardinality of X. The EA uses the function $head()$, which is defined as follows: given a clause δ and a set of atoms M, $head(\delta, M)$ is the set of all atoms in M that belong to the head of δ.

The proof of the following theorem is quite involved and therefore appears in the Appendix.

Theorem 2 (The EA is correct). *Given a positive HCF theory T as input, the EA generates a minimal model of T.*

The choice of the model to start with (Step 1) can be looked at as a nondeterministic step. Indeed, for different initial models, the EA may output different minimal models.

[10] A strongly connected component C of a directed graph G is a maximal subgraph of G such that for each pair of nodes v_1 and v_2 in C, C contains both a directed path from v_1 to v_2 and a directed path from v_2 to v_1.

In fact, the EA is able to generate *any* of the minimal models of the input theory. Before proving this result, it is useful to state the following lemma.

Lemma 3. *In any execution of the EA, if M is initialized to some minimal model of the input positive HCF theory T, the EA will output this minimal model.*

Proof. Suppose we begin executing EA(T) with M set to some minimal model $\widehat{M}$. By Theorem A.1, every atom in $\widehat{M}$ has a proof with respect to T and $\widehat{M}$. Let k be the maximum number of clauses used in any minimal-length proof of an atom from $\widehat{M}$. It is easy to verify that EA(T) will run as follows: Step 2 will be executed at most k times in a row, adding to M' all the atoms having a proof of length at most i by the i-th iteration, and then M' will be returned at Step 3. Hence, $M' = \widehat{M}$ will be generated by the EA. □

Proposition 4 (Nondeterministic completeness). *If M is a minimal model of a positive HCF theory T, then there is an execution of the* EA *that outputs M.*

Proof. Suppose M is a minimal model of T. There is an execution of EA(T) that picks the model M in Step 1. By Lemma 3, if the EA starts with a minimal model M, it will output this model. Hence, the result follows. □

The following example demonstrates how the EA works.

Example 5. Suppose we have the theory

1. $a \vee b$
2. $b \rightarrow a$
3. $a \vee c$

and suppose we start the EA with $M = \{a, b, c\}$. At Step 1 of the EA, $M' = \emptyset$. At Step 2, we get that $\Delta = \emptyset$, because the clauses violated by M' are the first and third clauses but both atoms in their heads belong to M. Since Δ is empty, we go to Step 3, and in Step 3, we get $\Delta := \{a \vee b, a \vee c\}$. Since $\{b\}$ is a source of $\{a, b, c\}$ (note that $\{c\}$ is also a source; we will describe shortly what happens when we choose $\{c\}$), we delete $\{b\}$ from M and are left with $M = \{a, c\}$, and we go to Step 2. In Step 2, we now get $\Delta = \{a \vee b\}$, so we add $\{a\}$ to M' and delete $\{a\}$ from M, which leaves us with $M' = \{a\}$ and $M = \{c\}$. We then repeat Step 2, but this time we get $\Delta = \emptyset$ (because none of the clauses is violated by M'), and so we go to Step 3. In Step 3 we also have $\Delta = \emptyset$, so the EA in this case returns $\{a\}$. Indeed, $\{a\}$ is a minimal model for the theory.

Let us now follow the option of choosing $\{c\}$ instead of $\{b\}$ as the source of $\{a, b, c\}$ the first time Step 3 is executed. We delete $\{c\}$ from M and are left with $M = \{a, b\}$, and we go to Step 2. In Step 2, we now get $\Delta = \{a \vee c\}$, so we add $\{a\}$ to M' and delete $\{a\}$ from M, which leaves us with $M' = \{a\}$ and $M = \{b\}$. We then repeat Step 2, but this time we get $\Delta = \emptyset$ (because none of the clauses is violated by M'), and so we go

to Step 3. In Step 3, we also have $\Delta = \emptyset$, so the EA in this case too returns $\{a\}$. Indeed, $\{a\}$ is the only minimal model for the theory.

The theory of Example 5 has only one minimal model. In the next example, the theory has several.

Example 6. Suppose we have the theory

1. $a \vee b$
2. $c \vee b$
3. $a \vee c$

and suppose we start the EA with $M = \{a, b, c\}$. At Step 1 of the EA, $M' = \emptyset$. At Step 2, we get that $\Delta = \emptyset$, because although all the clauses are violated by M', all the atoms in their heads belong to M. Since Δ is empty, we go to Step 3, and in Step 3 we get $\Delta := \{\text{all the clauses}\}$. Since $\{a\}$, $\{b\}$, and $\{c\}$ are each a source of $\{a, b, c\}$, we can delete from M any one of them. Suppose we delete $\{b\}$. We are left with $M = \{a, c\}$, and we go to Step 2. In Step 2, we now get $\Delta = \{a \vee b, b \vee c\}$, so we add $\{a, c\}$ to M' and delete $\{a, c\}$ from M, which leaves us with $M' = \{a, c\}$ and $M = \emptyset$. We then repeat Step 2, but this time we get $\Delta = \emptyset$, so we go to Step 3. In Step 3, we also have $\Delta = \emptyset$, so the EA in this case returns $\{a, c\}$. Indeed, $\{a, c\}$ is a minimal model for the theory. It is easy to see that the EA would return $\{a, b\}$ or $\{b, c\}$ had we selected, respectively, $\{c\}$ or $\{a\}$ as a source.

We next prove that the EA's time complexity is linear.

Theorem 7 (Complexity). *For positive HCF theories, the EA runs in time* $O(n)$, *where n is the size of the input theory.*

Proof. We describe briefly how the EA can be implemented in linear time. Following [33], we assume that each propositional variable is assigned a unique number so that a data structure with constant access time can be used for storing the data related to each variable.

We use the following data structures:

- *SCC array*: An array indexed by the strongly connected components (SCCs). Each entry in this array consists of the set of atoms that belongs to the SCC represented by the entry.
- *AtomH array*: An array that stores for each atom a list of clauses in which the atom appears in the head.
- *AtomB array*: An array that stores for each atom a list of clauses in which the atom appears in the body.
- *Rules array*: An array that stores for each clause
  (1) the clause itself as a linked list (so that deletions of atoms from the clause's head and body will be done in constant time once we have a pointer to the atom we want to delete), and

(2) two counters:
Head—number of atoms in the head, and
Body—number of atoms in the body.

- *Unsat2clauses*: The set of clauses having an empty body and a head of size $\geqslant 2$.
- *Unsat1clauses*: The set of clauses having an empty body and a head of size 1.
- *MinSCC*: The lowest SCC not visited yet.

All these data structures can be initialized in linear time.

At the end of Step 2 of the EA, we set *Unsat1clauses* to $\emptyset$. Then, for each atom A in X (X as in Step 2) we do the following:

> Using the array *AtomB*, delete A from the body of each clause in which A appears, and update the counter *Body* for this clause (namely, do *Body* := *Body*-1). If *Body* becomes 0, then add the clause to either *Unsat2clauses* or *Unsat1clauses*, depending on the number of atoms in the head.

Step 3 is done as follows. In *MinSCC* we have the minimum SCC in which there may be an atom that appears in the head of some clause in *Unsat2clauses*. For each atom A that belongs to *MinSCC*, we do the following:

> Delete A from the head of each clause, using the appropriate pointer in *AtomH*. Move clauses from *Unsat2clauses* to *Unsat1clauses* if necessary.

At the end of Step 3, we increase the counter *MinSCC* by one. Note that any clause added to *Unsat2clauses* in the future cannot have an atom in its head that belongs to an SCC that is lower than *MinSCC*, because we have removed such atoms from the heads. Also, any clause added to *Unsat2clauses* or *Unsat1clauses* at a later step could not originally have had an atom in its head that belongs to the current *MinSCC* because of how the dependency graph is built.

In this way, each clause is visited at most k times, where k is the number of atoms in the clause. Each time a clause is visited, we spend an amount of time which is a constant. Hence, the algorithm is linear. □

2.2. Other computational tasks

We will now consider other computational tasks needed for reasoning with minimal models.

Model checking for a positive HCF theory can also be done in time linear in the size of the theory. This is due to the fact that if we start executing the EA with M initialized to some minimal model, this will be the model that it outputs. Hence,

Theorem 8 (Model checking). *The EA solves model checking for the class of positive HCF theories in time* $O(n)$, *where n is the size of the theory.*

Proof. By Lemma 3, when at Step 1 of the EA, M is initialized to be some minimal model of T, the EA will output M. Therefore, the algorithm for checking whether a given interpretation $\widehat{M}$ is a minimal model for a theory T consists of two steps. The first step checks whether $\widehat{M}$ is a model for T (can be done in linear time). The second step

runs EA(T) with M initialized to $\widehat{M}$. By Theorem 7, the second step takes linear time in the size of T. Hence, the result follows. □

Likewise, minimal entailment and minimal membership are easier for positive HCF theories than they are in general.

Theorem 9 (Minimal entailment, minimal membership). *Minimal entailment for the class of positive HCF theories is* co-NP-*complete. Minimal membership for the class of positive HCF theories is* NP-*complete.*

Proof. That minimal entailment is in co-NP and minimal membership is in NP are results already implied by the work of Ben-Eliyahu and Dechter [7]. We begin by showing the NP-hardness of minimal membership, and we do so by showing a reduction from the problem POSITIVE-ONE-IN-THREE 3SAT, which is known to be NP-complete [41]:

Instance: Sets $S_1, \ldots, S_k$ having three elements X_i^1, X_i^2, X_i^3 each, $1 \leqslant i \leqslant k$.
Question: Return *YES* when there is a set of elements that contains exactly one element from each set.

Consider the positive HCF theory T where for each set S_i we have the set of clauses

$$X_i^1 \vee X_i^2 \vee X_i^3$$
$$X_i^1 \wedge X_i^2 \rightarrow NO$$
$$X_i^1 \wedge X_i^3 \rightarrow NO$$
$$X_i^2 \wedge X_i^3 \rightarrow NO$$

and we also have the clause $YES \vee NO$. *YES* is in some minimal model of T if and only if there is a set of elements that contains exactly one element from each set. Hence, minimal membership is NP-hard.

To show the co-NP-hardness of minimal entailment, we just note that a minimal model of the theory T above contains *YES* if and only if it does not contain *NO*. Thus, minimal entailment is co-NP-hard. □

Before closing this section, we would like to address an important issue raised by Dechter [14]. Instead of representing a theory as a set of clauses of the form (1), we could have represented a theory as a set of clauses of the form

$$A_1 \wedge A_2 \wedge \cdots \wedge A_m \wedge \neg C_1 \wedge \neg C_2 \wedge \cdots \wedge \neg C_{n-1} \rightarrow C_n, \tag{2}$$

where all the As and the Cs are atoms. We could then identify the class of *stratified* theories in a way that is parallel to the way stratified deductive databases and logic programs are defined.[11] It is well known that if a logic program is stratified, its intended model, called the *perfect model*, is one of its minimal models and can be found in linear time [33]. Therefore, it is quite immediate that a minimal model for stratified theories can be found in linear time. Recent research has explored relating CNF theories (or,

[11] For a formal definition of stratification, see the next section.

analogously, negation-free disjunctive logic programs) to the set of stratified theories obtainable from them through shifting. In particular, Schaerf [42] advocates a semantics for disjunctive deductive databases that is based on considering both the stratified and the unstratified normal logic programs that can be obtained from the given database by applying shifting in all possible ways. Following Dix et al. [16], we can define a CNF theory T to be a *causal theory* if there exists at least one stratified theory that can be obtained from T through shifting. Then, for any CNF theory T, we can define the set of *causal models* of T to be the set of perfect models of the stratified theories obtainable from T through shifting. But what is the relation between positive HCF theories, causal theories, and stratified theories? We note that, apparently, there is no obvious transformation (namely, shifting atoms from head to body or vice versa) from HCF theories to stratified theories or from stratified theories to HCF theories, as the following example illustrates.

Example 10. On the one hand, consider the positive HCF theory T:

$$a \vee b, \quad a \rightarrow c, \quad b \vee c$$

Theory T' is obtained by shifting all but one of the atoms from the head of the clause to the body:

$$\neg b \rightarrow a, \quad a \rightarrow c, \quad \neg c \rightarrow b$$

Theory T' is not stratified, however. On the other hand, consider the stratified theory T:

$$b \rightarrow a, \quad a \rightarrow b, \quad \neg a \wedge \neg b \rightarrow c$$

Theory T' is obtained by shifting negative body literals to the heads of clauses:

$$b \rightarrow a, \quad a \rightarrow b, \quad a \vee b \vee c$$

Theory T' is positive but not HCF.

However, it is easy to see that a simple variant of the EA can be used to find for each positive HCF theory T, a stratified theory T_s that is obtained by shifting all but one of the atoms from the head to the body and such that the perfect model of T_s is also a minimal model of T. Thus, any positive HCF theory is causal. Moreover, it is clear that the set of causal models of T is strictly contained in the set of minimal models of T. That such a containment is strict is shown by the next example.

Example 11. Consider the positive HCF theory T:

$$a \vee c, \quad b \wedge c \rightarrow a$$

This theory has two minimal models, namely, $\{a\}$ and $\{c\}$. The only stratified theory obtainable from T by shifting operations is

$$\neg c \rightarrow a, \quad b \wedge c \rightarrow a$$

whose perfect model is $\{a\}$. Thus, the minimal model $\{c\}$ is not a causal model for T.

By a similar line of reasoning it is clearly possible to compute one minimal model of a positive HCF theory by transforming it into a stratified theory and then computing its perfect model (in linear time). When compared to the EA, this procedure has two disadvantages: it is not complete, since it cannot generate some models although they are minimal (the EA is complete; see Proposition 4); and in many cases it will not be as efficient as the EA. For example, if the theory has an empty model, the EA will discover this fact at an early stage, whereas the procedure always transforms the theory into a stratified theory before computing any model.

We would also like to note that the class of Horn theories, for which a unique minimal model can be found in linear time [17,25], intersects the class of HCF theories, although these classes are distinct. Consider, for example, the theories $T_1 = \{a\}$, $T_2 = \{\neg a\}$, and $T_3 = \{a \vee b\}$. T_1 is both Horn and positive HCF, T_2 is Horn but not positive HCF, and T_3 is positive HCF but not Horn.

3. The elimination algorithm for nonground positive HCF theories

In this section, we generalize the EA so that it can be used to efficiently perform (Herbrand) model finding on a *nonground* positive HCF theory.

We could approach this problem by computing the ground instantiation of the input theory and then applying the EA algorithm (where distinct ground instances of atoms are handled as distinct propositional letters). This method is not convenient in many cases, since computing the ground instantiation of the input theory could be expensive. Therefore, in this section, we present a variation of the EA that does not require the input theory to be grounded first in order to compute one of its minimal models. This variant of the algorithm has the further advantage that it does not require us to construct an arbitrary model before finding a minimal one.

We will now refer to a nonground theory as a set of clauses of the form

$$\forall(X_1,\ldots,X_n)\, A_1 \wedge A_2 \wedge \cdots \wedge A_m \rightarrow C_1 \vee C_2 \vee \cdots \vee C_n \tag{3}$$

where all the As and the Cs are atoms in a *first-order* language with no function symbols, and $X_1,\ldots,X_n$ are all the variables that appear in the clause. We will often write (3) simply as

$$A_1 \wedge A_2 \wedge \cdots \wedge A_m \rightarrow C_1 \vee C_2 \vee \cdots \vee C_n \tag{4}$$

keeping in mind that all the variables are universally quantified. The definitions of head, body, and positive theories for nonground theories are the same as they were for propositional theories. Throughout this section, we assume that theories are positive. In the expression $p(X_1,\ldots,X_n)$, p is called a *predicate name*. The set of all constants appearing in a given theory T is called the *Herbrand universe* of T. If no constants occur in T, then an arbitrary constant is assumed to be contained in its Herbrand universe. The set of all atoms constructed using predicate names from T and constants from its Herbrand universe is called the *Herbrand base* of T. A *ground instance* of a clause δ of T is formed by consistently substituting constants from the Herbrand universe of T

$EA_{\mathcal{F}}(T)$

Input: A nonground positive HCF theory T.
Output: A minimal Herbrand model for T.

1. $M := \emptyset$; $M' := \emptyset$.
2. Let Δ be the set of all ground instances δ of rules in T violated by M such that $|atom\text{-}head\text{-}out(\delta, S)M'| = 1$.
   If $\Delta = \emptyset$, go to Step 3.
   Else, let $X := \bigcap_{\delta \in \Delta} atom\text{-}head\text{-}out(\delta, S)M'$;
   $M := M \cap X$;
   repeat Step 2.
3. Let Δ be the set of all ground instances δ of rules in T violated by M such that $|atom\text{-}head\text{-}out(\delta, S)M'| \geqslant 2$.
   If $\Delta = \emptyset$, return M.
   Else, let $H := \bigcap_{\delta \in \Delta} name\text{-}head\text{-}out(\delta, S)M'$, and let N be a source of all the predicate names in H in G_T; let $M' := M' \cap N$; go to Step 2.

Fig. 3. The elimination algorithm for nonground positive HCF theories.

for variables in δ. The *grounded version* of T, denoted $gr(T)$, is the set of all ground instances of all the clauses of T. A *Herbrand model* of T is a subset of the Herbrand base that satisfies $gr(T)$. A Herbrand model M of T is minimal if it does not properly contain any Herbrand model of T.

As we did in the propositional case, with every theory T we associate a directed graph G_T, called the *predicate-name dependency graph* of T, in which (a) each predicate name and each clause in T is a node and (b) there is an arc directed from a node p to a clause δ if and only if p is a predicate name in the body of δ, and there is an arc directed from δ to p if and only if p is a predicate name in the head of δ.

A theory T is HCF if and only if there is no clause of the form (3) in T such that for some C_i and C_j, $i \neq j$, G_T contains a directed cycle involving the predicate name of C_i and the predicate name of C_j. Specifically, in HCF theories two atoms with the same predicate name cannot be in the head of the same clause.

Proposition 12. *The head-cycle-freeness of a nonground theory T of size n can be checked in time $O(n)$.*

Proof. The algorithm for checking head-cycle-freeness here is similar to the one for a propositional theory (see Proposition 1). □

In Fig. 3, we present a variation of the EA, called $EA_{\mathcal{F}}$ ($\mathcal{F}$ for "first-order"), that computes a minimal Herbrand model for a positive nonground HCF theory. The $EA_{\mathcal{F}}$ uses the functions *atom-head-out*( ) and *name-head-out*( ). Given a rule δ and a set of predicate names S, $atom\text{-}head\text{-}out(\delta, S)S$ will return the set of *atoms* that appear in the head of δ but whose predicate names do not belong to S, and $name\text{-}head\text{-}out(\delta, S)S$ will return the set of *predicate names* that appear in the head of δ but do not belong to S. Thus, for example, if $\delta = c(X) \rightarrow a(X) \vee b(X)$ and $S = \{a\}$, then $atom\text{-}head\text{-}out(\delta, S)S = \{b(X)\}$ and $name\text{-}head\text{-}out(\delta, S)S = \{b\}$. The algorithm also employs a set M' of predicate

names. After a predicate name is added to M', no ground atom with this predicate name will be added to the output minimal model.

The proof of the following theorem appears in the Appendix.

Theorem 13 (The $EA_{\mathcal{F}}$ is correct). *The $EA_{\mathcal{F}}$ generates a minimal Herbrand model of the input theory.*

The following example shows how the $EA_{\mathcal{F}}$ works.

Example 14. Suppose we have the theory

1. $a(s) \vee b(s)$
2. $b(Y) \rightarrow a(Y)$
3. $a(s) \vee c(s)$
4. $a(Y) \rightarrow d(Y)$

At Step 1 of the $EA_{\mathcal{F}}$, $M = \emptyset$ and $M' = \emptyset$. At Step 2, we get that $\Delta = \emptyset$, because although clauses 1 and 3 are violated by M, each has two predicate names in its head that do not belong to M'. Since Δ is empty, we go to Step 3, and in Step 3, we get $\Delta = \{a(s) \vee b(s), a(s) \vee c(s)\}$. As M' is still empty, we get $H = \{a, b, c\}$. Since $\{b\}$ is a source of $\{a, b, c\}$ (note that $\{c\}$ is also a source), we set M' equal to $\{b\}$ and go to Step 2. In Step 2, we now get $\Delta = \{a(s) \vee b(s)\}$. Therefore X, and, consequently, M are both set equal to $\{a(s)\}$, and Step 2 is repeated. Since now $M = \{a(s)\}$, the clause violated by M for which $|\textit{atom-head-out}(\delta, S)M'| = 1$ is $a(s) \rightarrow d(s)$ (which we get by instantiating Y to s in clause 4). So we add $d(s)$ to M and get $M = \{a(s), d(s)\}$. Since there are no more instances of clauses violated by M, the algorithm stops and returns M. Indeed, $\{a(s), d(s)\}$ is a minimal Herbrand model for the theory.

The complexity of the $EA_{\mathcal{F}}$ for nonground positive HCF theories can be analyzed using the principles by which the *data complexity* of a query language over a relational database under some fixed semantics is defined [44].

Any nonground theory can be divided into two disjoint sets [38]: One set comprises the *intentional* component, which represents the reasoning component of the theory, and the other set the *extensional* component, which represents the collection of facts in the theory. For our purposes, the extensional part of a given nonground positive HCF theory T, denoted T_E, is the set of all the clauses that have an empty body and grounded atoms only in the head, and the intentional part of T, denoted T_I, is simply $T - T_E$. For instance, in the theory presented in Example 14, clauses 1 and 3 form the extensional component, and clauses 2 and 4 form the intentional component. If we analyze how the complexity of $EA_{\mathcal{F}}$ changes when we fix T_I and vary T_E, we discover the following.

Theorem 15. *Using the algorithm $EA_{\mathcal{F}}$, a minimal model of a nonground positive HCF theory $T_I \cap T_E$ can be found in time polynomial in the size of T_E.*

Proof. Let n be the size of T_E, m be the number of distinct predicate names occurring in T, and k be the maximum arity of any predicate in T. Since by assumption T_I is fixed, it is immediate that the maximum size of any model resulting from running $\mathrm{EA}_{\mathcal{F}}$ on T is $O(n^k m)$. Note, moreover, that the maximum size for M' is $O(m)$. We have:

(1) Step 1 of the $\mathrm{EA}_{\mathcal{F}}$ takes $O(1)$ time.
(2) Step 2 (construction of Δ and X, and update of M) takes $O(n^k m)$.
(3) Step 3 (construction of Δ and N, and update of M') takes $O(n^k m)$.

The statement then follows, because:

(1) Each time Step 2 is executed, $\mathrm{EA}_{\mathcal{F}}$ either adds (at least) one new atom to M or jumps to Step 3.
(2) Each time Step 3 is executed, $\mathrm{EA}_{\mathcal{F}}$ either adds (at least) one new predicate name to M' or stops and returns M.
(3) The size of m is $O(n)$.

As we have pointed out, using $\mathrm{EA}_{\mathcal{F}}$ instead of EA has two advantages: $\mathrm{EA}_{\mathcal{F}}$ requires neither preliminary grounding of the input theory nor construction of a starting model of the input theory. However, the $\mathrm{EA}_{\mathcal{F}}$ presents the drawback that it is not nondeterministically complete, that is, there are some minimal models that cannot be generated by this algorithm. Consider again, for instance, the theory $T = \{a \vee c,\ b \wedge c \rightarrow a\}$ of Example 11. Clearly, both $\{a\}$ and $\{c\}$ are minimal models of T, but since $\{c\}$ is a source of $\{a\}$ in G_T, the model $\{c\}$ will never be generated by the $\mathrm{EA}_{\mathcal{F}}$. Nondeterministic completeness can be achieved by grounding the theory and then using the EA algorithm.

4. Applications of the elimination algorithm

4.1. Disjunctive deductive databases

A variation of the EA can be used to construct a stable model of a disjunctive deductive database. We define a *disjunctive deductive database* (DDB) as a finite set of rules of the form

$$C_1 \mid C_2 \mid \ldots \mid C_n \leftarrow A_1, \ldots, A_m, not\, B_1, \ldots, not\, B_k \tag{5}$$

where all the As, the Bs, and the Cs are atoms over a first-order language with no function symbols. Without loss of generality, we assume that all of the Cs are distinct. By using the symbol "|" instead of the classical symbol "$\vee$", we mean to emphasize that here disjunction is used in a slightly different manner than it is in classical logic. We call the Bs *negative predicates*, the As *positive predicates*. A DDB is a *positive* DDB if and only if, for each rule, $k = 0$. The *predicate-name dependency graph* of a DDB *DB*, denoted G_{DB}, is defined as for theories by simply ignoring the B_is in form (5). A *head-cycle-free* DDBs (HDDB) is also defined in analogy to HCF theories.

Proposition 16. *The head-cycle-freeness of a DDB of size n can be checked in time* $O(n)$.

Proof. The algorithm for checking head-cycle-freeness here is similar to the one for a propositional theory (see Proposition 1). □

Following [36], we define a *stratified* DDB (SDDB) to be a DDB where it is possible to partition the set S of predicate names into subsets $\{S_0, \ldots, S_r\}$, called *strata*, such that for each rule δ of the form (5),

(1) all the predicate names of the Cs (that appear in the head of δ) have the same stratum index c,
(2) the strata indexes associated with the predicate names of the As are smaller than or equal to c, and
(3) the strata indexes associated with the predicate names of the Bs are strictly smaller than c.

So, each SDDB DB is associated with at least one partition of its predicate names into strata. For every stratification $\{S_0, \ldots, S_r\}$ of DB's predicate names ($r \geqslant 1$), we can partition the rules of DB into corresponding subsets $\{DB_1, \ldots, DB_r\}$ such that DB_i contains the rules that have in their heads predicates that are in the stratum S_i. (We assume without loss of generality that S_0 contains the predicates not appearing in the head of any rule.)

Several different semantics have been proposed for DDBs [3, 15, 21, 36, 37]. Notably, all these semantics use the set of the minimal models of a *positive* DDB to define the intended meaning of the DDB. The same holds for SDDBs: all the semantics that handle SDDBs agree on identifying the *stable* models of a SDDB as the intended meaning of the SDDB.

Definition 17 (Stable model [21]). Suppose DB is a variable-free DDB. If DB has no occurrence of "*not*", then the set of all its *stable models* is the set of all its minimal models.

If "*not*" occurs in DB, then its stable models are defined as follows. For any subset S of the atoms in DB, define DB^S to be the DDB obtained from DB by deleting

(1) all formulas of the form "*not* B" where $B \notin S$ from the body of each rule and
(2) each rule that has in its body a formula "*not* B" for some $B \in S$.

If S is one of the minimal models of DB^S (DB^S has no "*not*"), then we say that S is a *stable model* of DB. To apply the definition to a DDB with variables, we first have to replace each rule with its grounded instances.

A DDB may have no, one, or several stable models. We claim, as do others [1, 23, 40], that sometimes a problem can be solved by computing only one arbitrary stable model of a DDB. Two examples follow.

Example 18. Consider the well-known graph-theoretic problem of minimal set covering. This problem has applications in many domains, including diagnosis [35]. The problem is as follows: given a bipartite digraph $G = (V_1, V_2, E)$, a solution to the minimal set-covering problem for G is a minimal subset S of V_1 such that for each node $x \in V_2$ there exists a node $s \in S$ such that the edge (s, x) is in E. Consider the HDDB DB that includes, for each node $x \in V_2$, the rules

$EA_{\mathcal{S}}(DB)$

Input: A propositional SHDDB DB.
Output: A stable model for DB.

1. Partition DB into strata $DB_1, \ldots, DB_r$.
2. $M := \emptyset$.
3. For i:=1 to r, do:
   (a) Eliminate from DB_i each clause having in its body a negative literal "*not* D" with $D \in M$;
   (b) Eliminate all the negative literals from the remaining clauses in DB_i;
   (c) $M := \mathrm{EA}(DB_i \cup \{P \leftarrow \mid P \in M\})$.

Fig. 4. The elimination algorithm for SHDDBs.

$$s_1 \mid s_2 \mid \ldots \mid s_n \leftarrow x$$
$$x \leftarrow$$

where $s_1, \ldots, s_n$ are all the nodes belonging to V_1 such that $(s_1, x), \ldots, (s_n, x)$ are edges in E. Each stable model of DB encodes a solution to the minimal set covering problem. Therefore, we can pick an arbitrary stable model of DB to find a set covering for the graph G.

Example 19. Assume we have a pictorial database and want to construct a tool that when given the type of an object in a scene (say tree or river) suggests a default coloring for the object, under the constraint that objects close to one another in a scene should be distinguishable and, as such, should have different colors. Consider the following HDDB:

$$\begin{aligned}
blue(X) \mid green(X) \mid softgreen(X) &\leftarrow tree(X)\\
green(X) \mid brown(X) &\leftarrow mountain(X)\\
blue(X) &\leftarrow river(X)\\
green(X) \mid softgreen(X) &\leftarrow tree(X), river(Y),\\
&\qquad pclose(X,Y)\\
softgreen(X) &\leftarrow tree(X), mountain(Y),\\
&\qquad green(Y), pclose(X,Y)
\end{aligned}$$

Assume that the classification of the objects to be painted is stored in the database encoding of a particular scene, along with facts like $pclose(a, b)$, meaning that objects a and b will be painted close to one another in the scene. Each stable model of the HDDB encodes a feasible coloring of the objects in the scene.

In Fig. 4 we show algorithm $EA_{\mathcal{S}}$, a variation of the EA which computes one arbitrary stable model of a propositional stratified head-cycle-free DDB (SHDDB). The basic idea is to partition the SHDDB according to its stratification and then call the EA on each subset in the order implied by the stratification.

Note that when the algorithm is run on a DDB with variables, the grounded version of the DDB must be used.

Proofs of the following two theorems are given in the Appendix.

Theorem 20 (The $EA_{\mathcal{S}}$ is correct). *The $EA_{\mathcal{S}}$ generates a stable model of the input SHDDB.*

Theorem 21 (Nondeterministic completeness). *If M is a stable model of a SHDDB DB, then there is an execution of the $EA_{\mathcal{S}}$ that outputs M.*

The $EA_{\mathcal{S}}$ can also be used for computing the (unique) stable model of a stratified (nondisjunctive) deductive database.

Corollary 22. *Let DB be a stratified (nondisjunctive) deductive database. Then $EA_{\mathcal{S}}(DB)$ coincides with the (unique) stable model for DB.*

Theorem 23 (Complexity). *The $EA_{\mathcal{S}}$ for SHDDBs runs in time $O(n)$, where n is the size of the grounded version of the input DDB.*

Proof. Assume that the input DDB has k strata. It follows from Theorem 7 that the $EA_{\mathcal{S}}$ runs in time $O(\sum_{i=1}^{k} n_i)$, where n_i denotes the size of the ith stratum. Since $\sum_{i=1}^{k} n_i = n$, we get linear time complexity. □

We illustrate how the $EA_{\mathcal{S}}$ works with the next example.

Example 24. Suppose we have the following SHDDB DB:

1. $a \mid b \leftarrow$
2. $c \leftarrow not\ a$
3. $d \mid e \leftarrow c$

Note that, for this DDB, $S_0 = \emptyset$. Assume that we adopt a stratification such that $S_0 = \emptyset$, $S_1 = \{a, b\}$, $S_2 = \{c, d, e\}$. At Step 1 of the $EA_{\mathcal{S}}$, we compute the stratification of the rules. In this case, DB_1 consists of the first rule, and DB_2 consists of the other two rules. After setting $M = \emptyset$ (Step 2), we start the *for* loop at Step 3. Steps 3a and 3b do not modify the rule in DB_1. So, we apply the EA to DB_1. If we assume that $EA(DB_1) = \{b\}$, then $\{b\}$ is the value assigned to M, and we consider stratum DB_2. Step 3a does not modify DB_2. At Step 3b, the literal "*not a*" is eliminated from the body of rule number 2. Therefore, at Step 3c, the EA is applied to the DDB:

2′. $c \leftarrow$
3. $d \mid e \leftarrow c$
4. $b \leftarrow$

This last application may yield, for instance, the set $\{b, c, d\}$, which is then assigned to M and returned as the result of $EA_{\mathcal{S}}(DB)$. It is easily seen that the returned set $\{b, c, d\}$ is indeed a stable model for DB.

4.2. Disjunctive default logic

Disjunctive default logic is a generalization of Reiter's default logic introduced by Gelfond et al. [22] in order to overcome some of the difficulties that Reiter's default logic has when dealing with disjunctive information. In this section, we will focus on *propositional* disjunctive default logic. Gelfond et al. define a *disjunctive default theory* Δ as a set of disjunctive defaults. A *disjunctive default* is an expression of the form

$$\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma_1 | \ldots | \gamma_m} \tag{6}$$

where $\alpha, \beta_1, \ldots, \beta_n$, and $\gamma_1, \ldots, \gamma_m$ are formulas in some propositional language. Gelfond et al. define an *extension* for a disjunctive default theory Δ to be one of the minimal deductively closed set of sentences E' satisfying the condition[12] that, for any disjunctive default from Δ, if $\alpha \in E'$ and $\neg\beta_1, \ldots, \neg\beta_n \notin E$, then for some $1 \leqslant i \leqslant m$, $\gamma_i \in E'$.

Let us now consider the subset of disjunctive default theories that we call *disjunctive default programs* (DDPs). A DDP is a set of defaults of the form

$$\frac{A_1 \wedge \cdots \wedge A_m : \neg B_1, \ldots, \neg B_k}{C_1 | \ldots | C_n} \tag{7}$$

in which each of the As, the Bs, and the Cs is an atom and $n > 0$. Each such DDP Δ can be associated with a DDB DB_Δ by replacing each default of the form (7) with the rule

$$C_1 | \ldots | C_n \leftarrow A_1, \ldots, A_m, not\, B_1, \ldots, not\, B_k$$

The following theorem implies that all of the techniques and complexity results established with respect to DDBs also apply to DDPs.

Theorem 25 (Gelfond et al. [22]). *Let Δ be a DDP. The logical closure of E is an extension of Δ if and only if E is a stable model of DB_Δ.*

So, in particular, we can conclude that for the class of DDPs, computing an extension is $P^{NP[O(\log n)]}$-hard, checking whether a set of atoms is an extension is co-NP-hard, and deciding whether an atom belongs to some extension is Σ_2^p-hard.

Let us call a DDP *completely ordered* if and only if its corresponding DDB is stratified and HCF. Then, using the results in previous sections, we can identify subclasses of DDPs that are more manageable than the class of DDPs in general.

Theorem 26. *Let Δ be a completely ordered DDP, and let n be its size.*[13] *An extension for Δ can be found in time $O(n)$.*

Proof. Follows from Theorems 25 and 23. □

[12] Note the appearance of E in the condition.

[13] We measure the size of a DDP by the number of symbols it contains.

These results can be extended to deal with first-order disjunctive default theories, by using the same principles that were used to generalize the EA for propositional HCF theories to first-order HCF theories.

5. Related work

The class of HDDBs was introduced by Ben-Eliyahu and Dechter [7,8]. They have shown that queries on propositional HDDBs can be answered by solving the satisfiability problem from classical logic. Cadoli [9,10] has described a partitioning of the set of all propositional theories into classes for which model finding and model checking are either tractable or NP-hard. His partition invlolves considering the set of logical relations that corresponds to the theory, and it is not clear whether the tractable classes can be identified effectively. Ben-Eliyahu and Dechter [6] have developed other efficient algorithms for finding minimal models of propositional CNF theories. Their algorithms are efficient either when the theory is almost Horn, that is, when there are few non-Horn clauses and the set of all positive literals in any non-Horn clause is small, or when the theory can be represented as an acyclic network of low-arity relations.

Tasks related to minimal model computation have been studied also in the diagnosis literature [12,13] and, more recently, the logic programming literature [4,5]. Many of the algorithms used in diagnosis systems [12,13] are highly complex in the worst case. To find a minimal diagnosis, they first compute all the prime implicates of a theory and then find a minimal cover of the prime implicates. The first task is output exponential, while the second is NP-hard. Therefore, in the diagnosis literature, researchers often compromise completeness by using heuristic approaches. The work in the logic programming literature has focused on using efficient optimization techniques, such as linear programming, for computing minimal models (e.g., [4]). One limitation of this approach is that it does not address the issue of worst-case and average-case complexities.

Eiter, Leone, and Saccá [19] have studied the expressive power of various subsets of disjunctive databases. In particular, they have shown that SHDDBs are able to express all the queries in NP under brave semantics and in co-NP under cautious semantics. To compute answers to queries under brave semantics and cautious semantics, we check minimal membership and minimal entailment, respectively. Hence, the results reported in [19] confirm the relevance of HCF theories.

6. Conclusion

The task of computing minimal models is of interest to researchers in artificial intelligence and logic programming. Whether we are looking at circumscription, default logic, diagnosis, or commonsense reasoning in general, computing minimal models has been crucial to speeding up the reasoning process.

In this paper, we have introduced the algorithm EA, which performs, in linear time, minimal model finding and minimal model checking for a significant subclass of CNF

theories, namely, positive HCF theories. We have also shown that minimal entailment is easier for the class of positive HCF theories than it is in the general case. Our complexity results are included in Fig. 1.

We have demonstrated how the EA can be modified so that it can be used to answer queries posed on disjunctive deductive databases and disjunctive default theories in an efficient way. We have presented algorithm EA_S, which computes a stable model of a propositional SHDDB in linear time, and identified a class of disjunctive default theories for which extensions can be found in linear time. Note that the class of SHDDBs is a strict superclass of the class of all stratified nondisjunctive deductive databases. This observation, when coupled with the results of Eiter, Leone, and Saccá [19] (see Section 5), supports the claim that our algorithms deal with quite expressive subclasses.

Levesque [28] has argued that deductions can sometimes be performed rapidly if a "vivid" form of the knowledge is used, where a vivid form of a theory is some data structure in which the information is stored in a way that enables fast answers to common queries. Thus, Halpern and Vardi [24], Papadimitriou [34], and others have fielded the appealing idea that a vivid form of a theory need only be a model of the theory. In this case, deduction can be replaced by model checking, which is often the much easier task. Since a theory might have an exponential number of models, only the models that "best" represent the theory, namely, the models that are the "closest" to the real world, should be selected. One approach, adopted in circumscription, for example, is to select the minimal models of a theory as its vivid form. We argue, as have others [1,23,40], that sometimes fast query answering can be done with only one arbitrary minimal model of a theory. The work presented here is a step toward efficient implementation of such ideas.

Appendix A. Proofs

A.1. Useful theorems and definitions

Theorems A.1 and A.2 are taken from [7]. These theorems will be used to prove the correctness of the EA.

Following [7], we define a *proof* of an atom to be a sequence of clauses which can be used to derive the atom from the theory. Formally, an atom A has a *proof* with respect to a set of atoms M and a theory T if and only if there is a sequence of clauses $\delta_1, \ldots, \delta_n$ from T such that

(1) for each clause δ_i, *one and only one* of the atoms that appear in its head belongs to M (this atom will be denoted $h_M(\delta_i)$),
(2) $A = h_M(\delta_n)$,
(3) the body of each δ_i is satisfied by M, and
(4) δ_1 has an empty body and, for each $i > 1$, each atom in the body of δ_i is equal to $h_M(\delta_j)$ for some $1 \leqslant j < i$.

Theorem A.1 (Ben-Eliyahu and Dechter [7]). *A set of atoms M is a minimal model of an HCF theory T if and only if*

(1) *M satisfies each clause in T, and*
(2) *for each atom A in M, there is a proof of A with respect to T and M.*

The results for positive HCF theories hold for propositional DDB as well. Say that an atom A has a *proof* with respect to a set of atoms M and a DDB DB if and only if there is a sequence of rules $\delta_1, \ldots, \delta_n$ from DB such that

(1) for each clause δ_i, *one and only one* of the atoms that appear in its head belongs to M (this atom will be denoted $h_M(\delta_i)$),
(2) $A = h_M(\delta_n)$,
(3) the body of each δ_i is satisfied by M, and
(4) δ_1 has an empty body and, for each $i > 1$, each atom in the body of δ_i is equal to $h_M(\delta_j)$ for some $1 \leqslant j < i$.

Theorem A.2 (Ben-Eliyahu and Dechter [7]). *A set of atoms M is a stable model of an HDDB DB if and only if*

(1) *M satisfies each rule in DB, and*
(2) *for each atom A in M, there is a proof of A with respect to DB and M.*

The following lemma is probably known. We include a proof for the sake of completeness.

Lemma A.3. *Let DB be a propositional SHDDB, and let $S_1, \ldots, S_r$ be a partition of its atoms into strata. The following hold:*

(1) *If M is a stable model of DB and, for each $n \leqslant r$, M_n is the restriction of M to the atoms belonging to $S_1 \cup \cdots \cup S_n$, then M_n is a stable model for $DB_1 \cup \cdots \cup DB_n$.*
(2) *If M' is a stable model of $DB_1 \cup \cdots \cup DB_n$ for some $n \leqslant r$, then every stable model of $DB_{n+1} \cup \cdots \cup DB_r \cup \{P \leftarrow \mid P \in M'\}$ is a stable model of DB.*

Proof. (1) By Theorem A.2, each atom in M has a proof with respect to M and DB, and every rule in DB is satisfied by M. Because DB is stratified, it immediately follows that each atom in M_n has a proof with respect to $DB_1 \cup \cdots \cup DB_n$ and M_n, and every rule in $DB_1 \cup \cdots \cup DB_n$ is satisfied by M_n. Hence, the lemma follows.

(2) The proof is by induction on $r - n$.

Case $r - n = 0$: trivial.

Case $r - n > 0$: By the induction hypothesis, if M' is a stable model of $DB_1 \cup \cdots \cup DB_n$, then every stable model of $DB_{n+1} \cup \cdots \cup DB_{r-1} \cup \{P \leftarrow \mid P \in M'\}$ is a stable model of $DB_1 \cup \cdots \cup DB_{r-1}$. Let DB^n be $DB_{n+1} \cup \cdots \cup DB_{r-1} \cup \{P \leftarrow \mid P \in M'\}$, and let DB^* be $DB_1 \cup \cdots \cup DB_{r-1}$. Let M be a stable model of $DB^n \cup DB_r$. Using Theorem A.2, we will show that M is a stable model of $DB^* \cup DB_r$. Let δ be a rule in $DB^* \cup DB_r$. If δ is in DB_r, clearly it is satisfied by M. Suppose that δ is in DB^*. Let M^* be the restriction of M to the atoms that appear in DB^*. By part (1) of this lemma, M^* is a stable model of DB^*, and so M^* satisfies δ. It cannot be the case that the body of a rule that belongs to a lower strata becomes satisfied by making atoms that belong to a higher strata true, and hence δ must be satisfied by M as well. Next, we show that every atom in M has a proof with respect to M and $DB^* \cup DB_r$. If an atom A belongs

Long-EA(T)

Input: A positive HCF theory T.
Output: A minimal model for T.

begin
1. $M :=$ a model of T; $M' := \emptyset$; $i := 0$; $j := 0$.
2. $Deleted(i) := \emptyset$; $iteration := (i, j)$.
   Let Δ be the set of all clauses δ in T violated by M' such that for each $\delta \in \Delta$, $|head(\delta, M)| = 1$.
   If $\Delta = \emptyset$, go to Step 3.
   Else, let $X := \bigcap_{\delta \in \Delta} head(\delta, M)$;
   $M' := M' \cap X$;
   $M := M - X$;
   For each $A \in X$, $stage(A) := (i, j)$;
   $j := j + 1$;
   repeat Step 2.
3. Let Δ be the set of all clauses in T violated by M' such that for each $\delta \in \Delta$, $|head(\delta, M)| \geqslant 2$.
   If $\Delta = \emptyset$, return M'.
   Else, let $H := \bigcap_{\delta \in \Delta} head(\delta, M)$, and let X be a source of H;
   let $M := M - X$; $Deleted(i) := Deleted(i) \cap X$;
   Let Δ' be the set of all clauses δ in T such that the body of δ is satisfied by M' and $|head(\delta, M)| = 1$.
   If $\Delta' = \emptyset$, repeat Step 3.
   Else, $j := 0$; $i := i + 1$; go to Step 2.
end.

Fig. A.1. The elimination algorithm: Proof version.

to M^*, the proof of A with respect to M^* and DB^* must be also a proof of A with respect to M and $DB^* \cup DB_r$. Suppose that A belongs to strata r. We show next that A has a proof with respect to M and $DB^* \cup DB_r$. We take the proof α that A has with respect to $DB^n \cup DB_r$ and M. α is composed from proofs of atoms that belong to M^* and from rules from DB_r. We have already shown that atoms that belong to M^* have a proof with respect to M and $DB^* \cup DB_r$. Hence, we can construct a proof of A with respect to M and $DB^* \cup DB_r$. □

A.2. Proofs of theorems

Theorem 2 (The EA is correct). *Given a positive HCF theory T as input, the EA generates a minimal model of T.*

The proof is done using algorithm Long-EA in Fig. A.1. Long-EA does exactly what the EA does, except that it uses some indexes that will be employed in the proof.

Lemma A.4. *Let T be a theory, and let $EA(T) = M$. For any atom A in M, if $stage(A) = (i, c)$, then:*

(1) *There is an atom B in M such that $stage(B) = (i, 0)$, and there is a path from B to A in G_T.*
(2) *For each atom B such that $stage(B) = (i, 0)$ and for each atom C such that $C \in Deleted(j)$ for some $0 \leqslant j < i$, there is no path from B to C.*

Proof. By induction on *stage*(A) (assuming a lexicographic order).

Case stage$(A) = (0, c)$. Condition 2 holds trivially. Condition 1 is proved by induction on c:

Case stage$(A) = (0, 0)$: Condition 1 clearly holds (take $B = A$).

Case stage$(A) = (0, k)$, *where* $k > 0$: Consider the execution of Step 2 in the time when $j = k$ (j as in Step 2 of the EA). It must be the case that there is a clause δ in Δ such that $head(\delta, M) = \{A\}$ and there is an atom D in the body of δ such that *stage*$(D) = (0, k-1)$ (otherwise A would belong to a lower stage). By the induction hypothesis, then, Condition 1 holds.

Case stage$(A) = (n, c)$, *for some* $n > 0$. Condition 2 holds: Assume by contradiction that it does not hold. In this case, there is an atom B such that *stage*$(B) = (n, 0)$, and there is an atom $C \in Deleted(j)$, $0 \leqslant j < n$, such that there is a path in G_T from B to C. Note that, since *stage*$(B) = (n, 0)$, it must be the case that B is in the head of some clause δ of the form

$$A_1 \wedge \cdots \wedge A_r \rightarrow B \vee B_1 \vee \cdots \vee B_s$$

such that for each A_t, $1 \leqslant t \leqslant r$, *stage*$(A_t) = (u, v)$ for some $u < n$. Let (w, z) be the highest stage of any of $A_1, \ldots, A_r$, and assume D is the one $A_1, \ldots, A_r$ having stage (w, z). Since *stage*$(B) = (n, 0)$ and $w < n$, it must be the case that for some $k \geqslant z$, the body of δ is satisfied by M' at Step 3 when *iteration* $= (w, k)$. Since there is a path from B to C in G_T, and since the X chosen in Step 3 is a source of H, it cannot be the case that C belongs to one of $Deleted(w), Deleted(w+1), \ldots, Deleted(n-1)$. Suppose that C belongs to $Deleted(t)$ for some $0 \leqslant t < w$. By the induction hypothesis on Condition 1, there must be an atom B' in M such that *stage*$(B') = (w, 0)$ and there is a path from B' to D. So there is a path from B' to C. But C is in $Deleted(t)$, a contradiction to the induction hypothesis on Condition 2.

Condition 1 is proved by induction on c:

Case stage$(A) = (n, 0)$: Condition 1 clearly holds (take $B = A$).

Case stage$(A) = (n, k)$, *for some* $k > 0$: In this case, there must be a clause in the program

$$A_1 \wedge \cdots \wedge A_r \rightarrow A \vee B_1 \vee \cdots \vee B_s$$

such that for some $1 \leqslant j \leqslant r$, *stage*$(A_j) = (n, k-1)$. Using the induction hypothesis, we observe that Condition 1 holds. □

Corollary A.5. *For each atom D such that stage$(D) = (i, j)$ for some i, j, and for each atom C such that $C \in Deleted(t)$ for some $0 \leqslant t < i$, there is no path from D to C.*

Proof. Suppose *stage*$(D) = (i, j)$. By Condition 1 of Lemma A.4 there is an atom B in $M = EA(T)$ such that *stage*$(B) = (i, 0)$ and there is a path from B to D in G_T. By Condition 2 of Lemma A.4, for each atom C such that $C \in Deleted(t)$ for some $0 \leqslant t < i$, there is no path from B to C. So there cannot be a path from D to C (otherwise we could get from B to C via D). □

Lemma A.6. *The following invariants hold throughout the execution of the* EA:

(1) *Every atom in M' has a proof with respect to M' and T.*

(2) *For each clause violated by M', there is an atom in its head that belongs to M.*

Proof. It is easy to observe that the first claim holds. We start the algorithm with $M' = \emptyset$, and whenever we add an atom A to M', it is the case that there is a clause δ in T such that all the atoms in the body of δ belong to M' and A is the only atom in the head of δ which belongs to $M \cap M'$. Atoms are added to M' only if they belong to M. Therefore, we conclude that for each atom in M', there is a proof with respect to T and M'.

The second claim holds when we finish Step 1 of the algorithm, because M' is empty and the initial M is a model of T. Hypothetically, this claim may become false after executing commands that add atoms to M' or delete atoms from M. The algorithm contains two such commands: "$M' := M' \cap X$" in Step 2 and "$M := M - X$" in Steps 2 and 3. We will show that if the claim holds just before we execute these commands, it also holds after we execute these commands. In Step 3, we delete from M only atoms that belong to a source of all the atoms in heads of violated clauses; hence, if the claim holds before doing this command, it holds after as well.

It is left to show that if the second claim holds before we execute the two consecutive commands "$M' := M' \cap X; M := M - X$" in Step 2, it holds after we execute these commands. Suppose we execute the commands when *iteration* is set to (i, j) for some i, j. Note that for each $A \in X$, we have $stage(A) = (i, j)$. Suppose the claim was valid just before executing the two commands. Further, assume by contradiction that after executing the commands, there is some clause δ in T which is violated by M' and no atom in this clause's head belongs to M. It must be the case that there is some atom A in the body of δ which belongs to X (otherwise the claim would not hold just before executing $M' := M' \cap X; M := M - X$). Since $iteration = (i, j)$, all atoms deleted so far from M which are not in M' must belong to $Deleted(t)$ for some $t < i$. By Corollary A.5, none of the atoms in the head of δ were deleted from (the initial) M so far. Hence, there must be an atom in the head of δ which belongs to M, a contradiction. □

Proof of Theorem 2. It is easy to see that the EA always terminates when the input is a propositional theory. Let M' be the model returned by EA when it is given the input theory T. Clearly, M' satisfies each clause in T. By Lemma A.6, each atom has a proof with respect to T and M'. By Theorem A.1, M' is a minimal model for T. □

Theorem 13 (The $EA_{\mathcal{F}}$ is correct). *The $EA_{\mathcal{F}}$ generates a minimal Herbrand model of the input theory.*

Before proving Theorem 13, we prove the following lemma.

Lemma A.7. *The following invariants hold throughout the execution of the algorithm* $EA_{\mathcal{F}}$:

(1) *Every atom in M has a proof with respect to M and $gr(T)$.*

(2) *For each ground instance of any rule violated by M, there is an atom in its head whose predicate name does not belong to M'.*

Proof. First we show that the first claim holds. We start the algorithm with $M = \emptyset$, and whenever we add an atom A to M, it is the case that there is an instance δ of a rule in T such that all the ground atoms in the body of δ belong to M and A is the only ground atom in the head of δ whose predicate name does not belong to M'. Ground atoms are added to M only if their predicate names do not belong to M'. Since no predicate name can be deleted from M', no other ground atom in the head of δ will be added to M. Consequently, we conclude that for each atom in M, there is a proof with respect to $gr(T)$ and M.

The second claim holds trivially when we finish Step 1 of the algorithm, because M' is empty. The claim could become false after executing commands that add atoms to M or predicate names to M'. $\mathrm{EA}_{\mathcal{F}}$ contains only two such commands: "$M := M \cap X$" in Step 2, and "$M' := M' \cap N$" in Step 3. We will show that if the claim holds just before we execute the command "$M := M \cap X$" in Step 2 and the command "$M' := M' \cap N$" in Step 3, it holds after we execute these commands.

Suppose we execute the command "$M := M \cap X$" in Step 2 and let Δ' be the set of all instances of rules in T that become violated after the execution of this command. Assume by contradiction that some instance $\delta \in \Delta'$ contains in its head only atoms with predicate names from M'. Let p be a predicate name of an atom from the head of δ added to M' sometime in Step 3, and let q be a predicate name of an atom from X in the body of δ. It follows that there is a path from q to p in the dependency graph of T. By induction on i, the number of times Step 2 was executed after p was added to M', we will show that there is no path from q to p, which is a contradiction to our assumption.

Basis, for $i = 0$. We execute Step 2 immediately after adding p to M' in Step 3. Let Δ be the set of all rules violated by M before executing Step 2. Clearly, q is in the head of a rule in Δ, and q was not added to M' in Step 3 (because $q \in X$), so q does not belong to the source we found in Step 3. However, p was added to M' during the last time that we executed Step 3, so p does belong to the source we found in Step 3. This means that there is no path from q to p in the dependency graph of T, by our definition of source.

Induction step. Assume that Step 2 was executed $k > 0$ times after we added p to M' in Step 3. Let Y be a set of predicate names of atoms added to M when executing Step 2 the $k - 1$ time after we added p to M' in Step 3. Y was added to M when executing Step 2 the $k - 1$ time, and the atom with predicate name q was added to M when executing Step 2 the k time. Therefore, there must exist some atom C whose predicate name is in Y and a rule containing an atom with predicate name q in the head and the atom C in the body. This means that there is a path from the predicate name of C to q. By the induction hypothesis, there is no path from the predicate name of C to p. Hence, there cannot be a path from q to p (otherwise we could reach p from the predicate name of C via q).

Suppose now that we execute the command $M' := M' \cap N$ in Step 3. Note that before executing this command, it is the case that the heads of all the instances in Δ, the set of

all instances violated by M, contain at least two ground atoms whose predicate names do not belong to M'. Now, it cannot be the case that there are two ground atoms from the head of the same instance such that their predicate names belong to N, because T is HCF and the predicate names of all the atoms in N are in the same strongly connected component in the dependency graph of T. Thus, it must be the case that after executing the command $M' := M' \cap N$ in Step 3 the second claim holds. □

Proof of Theorem 13. First we show that the $\mathrm{EA}_{\mathcal{F}}$ terminates. Suppose that the input is a nonground positive HCF theory T. Let $\mathcal{H}$ denote the Herbrand base of T. At the end of Step 2, either at least one atom from $\mathcal{H}$ is added to M or we jump to Step 3. At Step 3, the $\mathrm{EA}_{\mathcal{F}}$ either adds at least one predicate name to M' or stops. Since $M \subseteq \mathcal{H}$, $\mathcal{H}$ is finite, M' is finite, and the algorithm stops when there is no predicate name from T that is not in M', the $\mathrm{EA}_{\mathcal{F}}$ must terminate. It is left to show that when the algorithm terminates, M is a minimal model of T. Note that M is a minimal model of T if and only if it is a minimal model of $gr(T)$. By Theorem A.1, it is enough to show that when the $\mathrm{EA}_{\mathcal{F}}$ terminates,

(1) M is a model of $gr(T)$, and

(2) every atom in M has a proof with respect to M and $gr(T)$.

We proved in Lemma A.7 that (2) holds. Using Condition 2 of Lemma A.7, we can show that (1) holds as well. Hence, M is a minimal model of T. □

Theorem 20 (The $\mathrm{EA}_{\mathcal{S}}$ is correct). *The* $\mathrm{EA}_{\mathcal{S}}$ *generates a stable model of the input SHDDB.*

Proof. First note that since the EA terminates and since the number r of strata of any input SHDDB is finite by definition, the $\mathrm{EA}_{\mathcal{S}}$ terminates. By part (2) of Lemma A.3 and the correctness of the EA (used in Step 3c of $\mathrm{EA}_{\mathcal{S}}$), it is easy to show that the $\mathrm{EA}_{\mathcal{S}}$ indeed generates a stable model of the input SHDDB. □

Theorem 21. If M is a stable model of an SHDDB DB, then there is an execution of the $\mathrm{EA}_{\mathcal{S}}$ that outputs M.

Proof. Let $\{S_0, \ldots, S_r\}$ be a stratification of the predicate symbols in DB, and let $\{DB_1, \ldots, DB_r\}$ be the corresponding partition of DB's rules, $r \geqslant 1$. Let M be a stable model of DB. We want to show that M is generated by $\mathrm{EA}_{\mathcal{S}}$. We proceed by induction on r.

Basis, for $r = 1$. In this case, it is sufficient to show that M is a stable model for DB if and only if M is a minimal model of $DB^{\emptyset}$. Indeed, by definition, M is a stable model of DB if and only if M is a minimal model of DB^M. Because $r = 1$, the predicate name of each negative literal appearing in any rule of DB belongs to S_0. Therefore, $DB^M = DB^{\emptyset}$. The claim then follows from Theorem 4.

Induction, assume the algorithm is complete for r *strata.* Consider a SHDDB DB with $r + 1$ strata. Let M_r be the restriction of M to the predicate names belonging to $S_1 \cup \cdots \cup S_r$. By part (1) of Lemma A.3, M_r is a stable model for $DB_1 \cup \cdots \cup DB_r$. By induction, there exists an execution of the $\mathrm{EA}_{\mathcal{S}}$ with input $DB_1 \cup \cdots \cup DB_r$ that

generates M_r. It is then immediate that there exists an execution of the $EA_{\mathcal{S}}$ with input DB that generates M_r as the result associated to the first r strata of DB. By Theorem 4, it is therefore sufficient to show that if M is a stable model of DB, then M is a minimal model of $\widehat{M} = DB_{r+1}^{M_r} \cup \{A \leftarrow | \ A \in M_r\}$. Note that M is a stable model of DB if $M = M_r \cap M'$, for some M' which is a stable model of $DB_{r+1}^{M_r}$. Since $DB_{r+1}^{M_r}$ is positive, M' is a minimal model of $DB_{r+1}^{M_r}$. Hence, M must be a minimal model of $\widehat{M}$. □

Acknowledgments

We thank Victoria Zemlyanker for allowing us to include her version of algorithm $EA_{\mathcal{F}}$ in Section 3 of this paper. We also gratefully acknowledge points raised by the anonymous referees; their remarks helped us improve the quality of this paper. We thank Marco Cadoli, Rina Dechter, Thomas Eiter, and Georg (Giorgio) Gottlob for useful suggestions and discussions (some through e-mail). Thanks also to Michelle Bonnice for editing on parts of this paper.

The work of the first author was supported in part by grants NSF IRI-88-21444 and AFOSR 90-0136, and in part by an infrastructure grant for data-mining technology from the Israeli Ministry of Science. The second author is supported in part by the ECC under the EU-US project "DEUS EX MACHINA: Non-determinism for deductive databases" and by a MURST grant (40% share) under the project "Sistemi formali e strumenti per basi di dati evolute".

References

[1] S. Abiteboul, E. Simon and V. Vianu, Non-deterministic languages to express deterministic transformations, in: *Proceedings Ninth ACM Symposium on Principles of Database Systems*, Nashville, TN (1990) 218–229.

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

[3] C. Baral, J. Lobo and J. Minker, WF^3: a semantics for negation in normal disjunctive logic programs, in: Z.W. Ras and M. Zemankova, eds., *Methodologies for Intelligent Systems: Proceedings 6th International Symposium, ISMIS'91* (1991) 459–468.

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

[5] R. Ben-Eliyahu, A hierarchy of tractable subsets for computing stable models, *J. Artif. Intell. Res.* 5 (1996) 27–52.

[6] R. Ben-Eliyahu and R. Dechter, On computing minimal models, *Ann. Math. Artif. Intell.* 18 (1) (1996) 3–27; short version in: *Proceedings AAAI-93*, Washington, DC (1993) 2–8.

[7] R. Ben-Eliyahu and R. Dechter, Propositional semantics for disjunctive logic programs, *Ann. Math. Artif. Intell.* 12 (1994) 53–87; short version in: *Proceedings 1992 Joint International Conference and Symposium on Logic Programming (JICSLP-92)*, Washington, DC (1992).

[8] R. Ben-Eliyahu and R. Dechter, Default reasoning using classical logic, *Artificial Intelligence* 84 (1–2) (1996) 113–150.

[9] M. Cadoli, The complexity of model checking for circumscriptive formulae, *Inform. Process. Lett.* 44 (3) (1992) 113–118.

[10] M. Cadoli, On the complexity of model finding for nonmonotonic propositional logics, in: A. Marchetti Spaccamela, P. Mentrasti and M. Venturini Zilli, eds., *Proceedings 4th Italian Conference on Theoretical Computer Science* (1992) 125–139.

[11] Z. Chen and S. Toda, The complexity of selecting maximal solutions, *Inform. and Comput.* 119 (2) (1995) 313–325.

[12] J. de Kleer, A.K. Mackworth and R. Reiter, Characterizing diagnoses and systems, *Artificial Intelligence* 56 (2–3) (1992) 197–222.

[13] J. de Kleer and B.C. Williams, Diagnosing multiple faults, *Artificial Intelligence* 32 (1) (1987) 97–130.

[14] R. Dechter, Personal communication (October 1993).

[15] J. Dix, Classifying semantics of disjunctive logic programs, in: *Proceedings 1992 Joint International Conference and Symposium on Logic Programming (JICSLP-92)*, Washington, DC (1992) 798–812.

[16] J. Dix, G. Gottlob and V. Marek, Reducing disjunctive to non-disjunctive semantics by shift-operations, *Fund. Inform.* 28 (1–2) (1996) 87–100.

[17] W.F. Dowling and J.H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, *J. Logic Programming* 1 (3) (1984) 267–284.

[18] T. Eiter and G. Gottlob, Propositional circumscription and extended closed-world reasoning are Π_2^p-complete, *Theoret. Comput. Sci.* 114 (2) (1993) 231–245.

[19] T. Eiter, N. Leone and D. Saccá, The expressive power of partial models for disjunctive deductive databases, in: *Proceedings LID-96*, San Miniato, Italy (1996); also: *Theoret. Comput. Sci.*, to appear.

[20] T. Eiter, Personal communication (October 1993).

[21] M. Gelfond and V. Lifschitz, Classical negation in logic programs and disjunctive databases, *New Generation Comput.* 9 (3–4) (1991) 365–385.

[22] M. Gelfond, H. Przymusinska, V. Lifschitz and M. Truszczyński, Disjunctive defaults, in: *Proceedings 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR-91)*, Cambridge, MA (1991) 230–237.

[23] F. Giannotti, D. Pedreschi, D. Saccá and C. Zaniolo, Non-determinism in deductive databases, in: *Proceedings 2nd International Conference on Deductive and Object Oriented Databases* (1991).

[24] J. Halpern and M. Vardi, Model checking vs. theorem proving: A manifesto, in: *Proceedings International Conference on Principles of Knowledge Representation and Reasoning (KR-91)*, Cambridge, MA (1991) 325–334.

[25] A. Itai and J.A. Makowsky, Unification as a complexity measure for logic programming, *J. Logic Programming* 4 (2) (1987) 105–117.

[26] P.G. Kolaitis and C.H. Papadimitriou, Some computational aspects of circumscription, *J. ACM* 37 (1) (1990) 1–14.

[27] N. Leone, P. Rullo and F. Scarcello, Disjunctive stable models: unfounded sets, fixpoint semantics and computation, *Inform. and Comput.* 135 (2) (1997) 69–112.

[28] H. Levesque, Making believers out of computers, *Artificial Intelligence* 30 (1) (1986) 81–108.

[29] V. Lifschitz, Computing circumscription, in: *Proceedings IJCAI-85*, Los Angeles, CA (1985) 121–127.

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

[31] J. McCarthy, Applications of circumscription to formalizing common-sense knowledge, *Artificial Intelligence* 28 (1) (19860 89–116.

[32] J. Minker, On indefinite databases and the closed-world assumption, in: *Proceedings 6th Conference on Automated Deduction*, Lecture Notes in Computer Science, Vol. 138 (Springer, Berlin, 1982) 292–308.

[33] I. Niemela and J. Rintanen, On the impact of stratification on the complexity of non-monotonic reasoning, *J. Appl. Non-Classical Logics* 2 (1994) 141–179.

[34] C.H. Papadimitriou, On selecting a satisfying truth assignment, in: *Proceedings 32nd Annual ACM Symposium on Foundations of Computer Science* (1991) 163–169.

[35] Y. Peng and J. Reggia, Plausibility of diagnostic hypothesis: The nature of simplicity, *IEEE Trans. Systems Man Cybernet.* 17 (1987) 146–162.

[36] T. Przymusinski, On the declarative semantics of deductive databases and logic programs, in: J. Minker, ed., *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, Los Altos, CA, 1988) 193–216.

[37] T.C. Przymusinski, Stable semantics for disjunctive programs, *New Generation Comput.* 9 (1991) 401–424.
[38] R. Reiter, On closed-world databases, in: H. Gallaire and J. Minker, eds., *Logic and Data Bases* (Plenum Press, New York, 1978) 55–76.
[39] R. Reiter, A logic for default reasoning, *Artificial Intelligence* 13 (1–2) (1980) 81–132.
[40] D. Saccá and C. Zaniolo, Stable models and nondeterminism in logic programs with negation, in: *Proceedings Ninth ACM Symposium on Principles of Database Systems*, Nashville, TN (1990).
[41] T.J. Schaefer, The complexity of satisfiability problems, in: *Proceedings 10th Annual ACM Symposium on Theory of Computing* (1978) 216–226.
[42] M. Schaerf, Negation and minimality in disjunctive databases, *J. Logic Programming* 23 (1) (1995) 63–86.
[43] R. Tarjan, Depth-first search and linear graph algorithms, *SIAM J. Comput.* 1 (2) (1972) 146–160.
[44] M.Y. Vardi, The complexity of relational query languages, in: *Proceedings 14th Annual ACM Symposium on Theory of Computing (STOC-82)* (1982) 137–145.