ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Artificial Intelligence 169 (2005) 1–22

Artificial Intelligence

www.elsevier.com/locate/artint

An incremental algorithm for generating all minimal models

Rachel Ben-Eliyahu – Zohary [1]

Software Engineering Department, Jerusalem College of Engineering (JCE), Jerusalem 91035, Israel

Received 1 October 2002; received in revised form 23 March 2005; accepted 22 June 2005

Available online 11 August 2005

Abstract

The task of generating minimal models of a knowledge base is at the computational heart of diagnosis systems like truth maintenance systems, and of nonmonotonic systems like autoepistemic logic, default logic, and disjunctive logic programs. Unfortunately, it is NP-hard. In this paper we present a hierarchy of classes of knowledge bases, $\Psi_1, \Psi_2, \ldots$, with the following properties: first, Ψ_1 is the class of all Horn knowledge bases; second, if a knowledge base T is in Ψ_k, then T has at most k minimal models, and all of them may be found in time $O(lk^2)$, where l is the length of the knowledge base; third, for an arbitrary knowledge base T, we can find the minimum k such that T belongs to Ψ_k in time polynomial in the size of T; and, last, where $\mathcal{K}$ is the class of all knowledge bases, it is the case that $\bigcup_{i=1}^{\infty} \Psi_i = \mathcal{K}$, that is, every knowledge base belongs to some class in the hierarchy. The algorithm is incremental, that is, it is capable of generating one model at a time.

Keywords: Minimal models; Nonmonotonic reasoning; Diagnosis; Logic programming; Knowledge representation; Propositional statisfiability; Datalog

E-mail address: rbz@jce.ac.il (R. Ben-Eliyahu – Zohary).

[1] Part of this work was done while the author was a visiting scholar in the division of engineering and applied sciences, Harvard university, Cambridge, Massachusetts. A paper discussing a preliminary version of this work has appeared in AAAI-2000 under the name "A demand-driven algorithm for generating minimal models".

0004-3702/$ – see front matter
doi:10.1016/j.artint.2005.06.003

1. Introduction

Computing minimal models is an essential task in many reasoning systems in Artificial Intelligence, including circumscription [23,25,26], default logic [28], and minimal diagnosis [12], and in answering queries posed on logic programs and deductive databases [27]. 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 help reaching a substantial speed up in inference in implemented systems.

Let us take a closer look at the task of computing the stable models of a knowledge base expressed in the language of disjunctive logic programs. One of the most successful semantics for logic programs is *stable model semantics* [5,16,17], which associates with any logic program a (possibly empty) set of models called *stable models*. Intuitively, each stable model represents a set of coherent conclusions one might deduce from the logic program. It turns out that the task of computing grounded interpretations for a set of TMS justifications corresponds exactly to the task of computing the stable models of the logic programs represented by the set of TMS justifications, and that algorithms for computing stable models may be used in computing expansions of autoepistemic programs and extensions of Reiter's default theories [15,18].

Each stable model of a knowledge base is a minimal model. Moreover, if the knowledge base is stratified, that is, if there are no circular dependencies between the facts that involve negation, the computation of the stable model is carried by dividing the knowledge into layers (strata) and computing the set of minimal models in each strata.

The algorithm presented in this paper can be used for computing all minimal models, but it can stop once only part of the models have been generated. That is, there is no need to compare all the models of the knowledge base with each other in order to find out which of them is minimal. This feature can be used, for example, as follows:

In entailment—a fact follows from the knowledge base iff it is true in all minimal models. We can check the minimal models one at a time and refute a fact before seeing all of them.

In diagnosis—each minimal model is an indication of a possible set of faulty components. We can check the components suggested by some minimal model while the next minimal model is being generated.

The task of reasoning with minimal models has received a formal analysis in several studies [3,8–10,14,22]. Unfortunately, the results of the above work on the complexities of reasoning with minimal models are discouraging. It turns out that even when the knowledge base is positive, that is, when the knowledge base has no integrity constraints, finding one minimal model is $P^{NP[O(\log n)]}$-hard [9] (and positive theories always have a minimal model!),[2] and checking whether a model is minimal for some knowledge base is co-NP-complete [8].

[2] 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 [10].

In this paper we present a new algorithm for computing minimal models. Using this algorithm, we can show a hierarchy of classes of knowledge bases, $\Psi_1, \Psi_2, \ldots$, with the following properties: first, Ψ_1 is the class of all Horn knowledge bases; second, if a knowledge base T is in Ψ_k, then T has at most k minimal models, and all of them may be found in time $O(lk^2)$, where l is the length of the knowledge base; third, for an arbitrary knowledge base T, we can find the minimum k such that T belongs to Ψ_k in time polynomial in the size of T; and, last, where $\mathcal{K}$ is the class of all knowledge bases, it is the case that $\bigcup_{i=1}^{\infty} \Psi_i = \mathcal{K}$, that is, every knowledge base belongs to some class in the hierarchy. The algorithm that we present is demand-driven, that is, it is capable of generating one model at a time. We show how the algorithm can be generalized to allow efficient computation of minimal Herbrand models for the subclass of all function-free first-order knowledge bases.

The paper is organized as follows. In the next section, we define some basic terminology. In Section 3 we present the algorithm for minimal model generation, called AM. Algorithm AM works from the bottom up on the superstructure of the dependency graph of the knowledge base and may use any known procedure for computing minimal models as a subroutine. Section 4 explains how the algorithm AM can be generalized to handle knowledge bases over a function-free first-order language. Finally, in Sections 5 and 6, we discuss related work and make concluding remarks.

2. Preliminary definitions

2.1. Syntax

We define a knowledge base to be a set of rules which are implications of the form

$$A_1 \wedge \cdots \wedge A_n \longrightarrow B_1 \vee \cdots \vee B_m \tag{1}$$

where all the A's and B's are positive atoms and $m, n \geqslant 0$. The B's are called the *head* of the rule, the A's—the *body*. When $n = 0$ the rule (1) becomes

$$\textbf{true} \longrightarrow B_1 \vee \cdots \vee B_m,$$

or simply:

$$B_1 \vee \cdots \vee B_m, \tag{2}$$

and is called a *fact*. Rule (1) and fact (2) are said to be *about* $B_1, \ldots, B_m$. When $m = 0$ the rule (1) becomes

$$A_1 \wedge \cdots \wedge A_n \longrightarrow \textbf{false}, \tag{3}$$

and is called *an integrity constraint* (name borrowed from database terminology). A rule of the form (3) is said to be *about* $A_1, \ldots, A_n$.

When both m, n are 0, the rule (1) is equal to **false**. The rule (1) is *Horn* whenever $m \leqslant 1$.

The *size*, or *length* of a knowledge base T is the total number of symbols that it takes to write all the rules in it. It is well known that for any propositional formula there is a logically equivalent CNF formula. It is easy to verify that any CNF propositional formula has an equivalent formula in our language. Hence for any propositional formula there is a logically equivalent formula in our language.

2.2. The dependency graph

Let T be a knowledge base. We will define a binary relation $\equiv$ over the atoms in T to be the minimal relation having the following properties. Let P, Q and R be atoms in the knowledge base.

1. $P \equiv P$.
2. If at least one of the following conditions hold, $P \equiv Q$:
   (a) P and Q are in the head of the same rule;
   (b) P and Q are both in the body of the same integrity constraint;
   (c) P and Q are both unconstrained. An atom P is *unconstrained* iff it appears only in bodies of rules which are not integrity constraints.
3. If $P \equiv Q$ and $Q \equiv R$ then $P \equiv R$.

Clearly, $\equiv$ is an equivalence relation and hence the equivalence sets make up a partition of all the atoms in the knowledge base.

Example 2.1 (*Running example*). Consider the following knowledge base T_0:

r_1: $P_1 \vee Q_1$
r_2: $P_1 \longrightarrow P_2 \vee Q_2$
r_3: $P_2 \longrightarrow P_3 \vee Q_2$
r_4: $P_3 \longrightarrow Q_3$
r_5: $P_2 \wedge Q_2 \longrightarrow$ **false**
r_6: $P_4 \vee Q_4$
r_7: $P_5 \longrightarrow P_4$.

Note that P_5 is the only unconstrained atom. The equivalence sets with respect to the relation $\equiv$ are:

s_1: $\{P_1, Q_1\}$
s_2: $\{P_2, Q_2, P_3\}$
s_3: $\{Q_3\}$
s_4: $\{P_4, Q_4\}$
s_5: $\{P_5\}$.

Given a knowledge base T, the *dependency graph* of T is a directed graph built as follows:

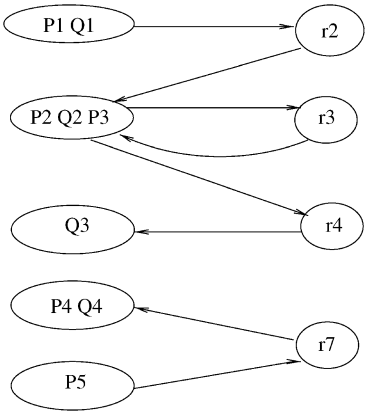

Fig. 1. The dependency graph of T_0.

Nodes: there are two types of nodes:
1. Each equivalence set with respect to the relation $\equiv$ is a node, called *ES-node*.
2. Each rule having both a non-empty head and a non-empty body is a node, called *R-node*. Note that, by definition, an integrity constraint is considered a rule with an empty head and hence will *not* be represented as an R-node in the graph.

Edges: Edges exist only between ES-nodes and R-nodes. Let s be an ES-node and r an R-node. There is an edge directed from s to r iff an atom from ES-node s appears in the body of r, and there is an edge directed from r to s iff there is an atom in s that appears in the head of r.

Example 2.2. The dependency graph of the knowledge base T_0 of Example 2.1 is shown in Fig. 1.

The *strongly connected components* (SCC) of a directed graph G make up a partition of its set of nodes such that, for each subset S in the partition and for each $x, y \in S$, there are directed paths from x to y and from y to x in G. The strongly connected components are identifiable in $O(V + E)$ time, where V is the number of nodes and E the number of edges in the graph [30].

The *super dependency graph* of a knowledge base T, denoted G_T, is the superstructure of the dependency graph of T. That is, G_T is a directed graph built by making each strongly connected component (SCC) in the dependency graph of T into a node in G_T. An arc exists from an SCC s to an SCC v iff there is an arc from one of the nodes in s to one of the nodes in v in the dependency graph of T.

Note that G_T is an acyclic graph; for if it has a directed cycle, all the SCCs on it should have been one component.

Example 2.3. The super dependency graph of T_0 is shown in Fig. 2. The nodes in the square are grouped into a single node.

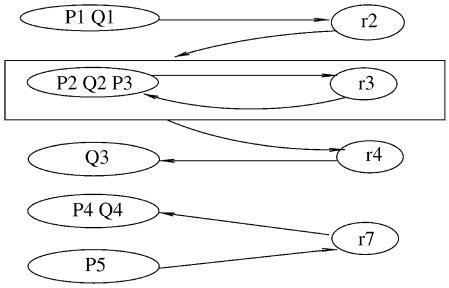

Fig. 2. The super dependency graph of T_0.

Recall that a *source* of a directed graph is a node with no incoming edges, while a *sink* is a node with no outgoing edges. Given a directed graph G and a node s in G, the *subgraph rooted by* s, is the subgraph of G having all and only nodes t such that there is a path directed from t to s in G (this includes s itself). The *children* of s in G are all nodes t such that there is an arc directed from t to s in G.

Example 2.4. Consider the graph illustrated in Fig. 2. Node $\{P_1, Q_1\}$ is a source in the graph; $\{P_5\}$ is a sink. The subgraph rooted by $\{r_2\}$ consists of the node $\{r_2\}$, the node $\{P_1, Q_1\}$, and the arc between them. $\{r_7\}$ has only one child—$\{P_5\}$.

2.3. Minimal models

Sometimes we will treat a truth assignment (in other words, an interpretation) in propositional logic as a set of atoms—the set of all atoms assigned **true** by the interpretation. Given two interpretations, I and J, over sets of atoms A and B, respectively, the interpretation $I + J$ is defined as follows, where P is an arbitrary atom:

$$(I + J)(P) = \begin{cases} I(P) & \text{if } P \in A \setminus B, \\ J(P) & \text{if } P \in B \setminus A, \\ I(P) & \text{if } P \in A \cap B \text{ and } I(P) = J(P), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

It is easy to verify that the operator $+$ as defined above is commutative and associative. If $I(P) = J(P)$ for every $P \in A \cap B$, we say that I and J are *consistent*. A set of interpretations is consistent if every pair of interpretations in the set is consistent.

A *partial* interpretation over a set of atoms A is a truth assignment over a set of atoms B where $B \subseteq A$. A *literal* is an atom (e.g., P) or a negated atom (e.g., $\neg P$). A partial interpretation can be represented as a set of literals: positive literals represent the atoms that are true, negative literals—the atoms that are false, and the rest are unknown. Note that if A is a set of literals which is a partial interpretation then A must be consistent. That is, A cannot contain two literals which are the negation of each other (e.g., both P and $\neg P$). A model of a knowledge base in propositional logic is an interpretation that satisfies all the rules. A model m is *minimal* among a set of models M iff there is no model $m' \in M$ such that $m' \subset m$. We will use the term *minimal model* of some knowledge base T to denote a model which is minimal among all the models of T. A knowledge base will be called *Horn*

iff all its rules are Horn. A consistent Horn knowledge base has a unique minimal model that can be found in linear time [11].

3. The algorithm

Algorithm ALL-MINIMAL (AM) is shown in Fig. 3. Algorithm AM exploits the structure of the knowledge base as it is reflected in its super dependency graph. It computes all minimal models while traversing the super dependency graph from the bottom up, and may use any algorithm for computing minimal models as a subroutine.

Let T be a knowledge base. With each node s in G_T (the super dependency graph of T), we associate the sets T_s, A_s, M_s, and $\widehat{T}_s$. T_s is the subset of T containing all the rules about the atoms in s. In other words, if s contains no ES-nodes then T_s is the empty set. Otherwise, T_s is the set of all the rules which are about the atoms from all the ES-nodes in s. Note that the R-nodes in the set s have no role in the procedure for constructing T_s.

Formally:

$$T_s = \{r \mid r \text{ is about a set of atoms } B \text{ and } B \subseteq s\}.$$

For example, $T_{\{P_1,Q_1\}}$ is r_1 and $T_{\{P_2,Q_2,P_3,r_3\}}$ is r_2, r_3, and r_5.

A_s is the set of all atoms in the nodes in the subgraph of G_T rooted by s, and M_s is the set of minimal models associated with the subset of the knowledge base T which contains only rules about atoms in A_s. We define $\widehat{T}_s$ to be the knowledge base obtained from T_s by deleting each occurrence of an atom that does not belong to s from the body of every rule. For example, if $T_s = \{b \longrightarrow a, a \wedge d \longrightarrow c, a\}$ and $s = \{a, c\}$, then $\widehat{T}_s = \{a, a \longrightarrow c\}$. The running example is next.

ALL-MINIMAL(T)
Input: A knowledge base T.
Output: The set of all minimal models of T.

1. Construct G_T;
2. Traverse G_T from the bottom up. For each node s, do:
   (a) $M_s := \emptyset$;
   (b) Let $s_1, \ldots, s_j$ be the children of s;
   (c) If $j = 0$, then $M_{c(s)} := \{\emptyset\}$;
   else $M_{c(s)} := \text{Combine}(\{M_{s_1}, \ldots, M_{s_j}\})$;
   (d) For each $m \in M_{c(s)}$, do:
   (i) $T_{s_m} := \text{Transform}(T_s, m)$;
   (ii) M := ALL-MINIMAL-SUBROUTINE(T_{s_m});
   (iii) If $M \neq \emptyset$,
   then $M_s := M_s \cup \text{Combine}(\{\{m\}, M\})$;
3. Output Combine($\{M_{s_1}, \ldots, M_{s_k}\}$),
   where $s_1, \ldots, s_k$ are the sinks of G_T.

Fig. 3. Algorithm ALL-MINIMAL (AM).

Example 3.1 (*Running example cont.*). Suppose s is the node including P_2, P_3, Q_2, and r_3. Then:

T_s is the set of the following rules:

r_2: $P_1 \longrightarrow P_2 \vee Q_2$

r_3: $P_2 \longrightarrow P_3 \vee Q_2$

r_5: $P_2 \wedge Q_2 \longrightarrow \textbf{false}$.

A_s is the set of atoms $\{P_1, Q_1, P_2, Q_2, P_3\}$.

Computing M_s: First, we have to find out what are all the rules about atoms in A_s. These are:

r_1: $P_1 \vee Q_1$

r_2: $P_1 \longrightarrow P_2 \vee Q_2$

r_3: $P_2 \longrightarrow P_3 \vee Q_2$

r_5: $P_2 \wedge Q_2 \longrightarrow \textbf{false}$.

M_s is the set of minimal models of the above knowledge base. That is,

$$M_s = \big\{\{P_1, P_2, P_3\}, \{P_1, Q_2\}, \{Q_1\}\big\}.$$

$\widehat{T}_s$ is the set of the following rules:

$P_2 \vee Q_2$

r_3: $P_2 \longrightarrow P_3 \vee Q_2$

r_5: $P_2 \wedge Q_2 \longrightarrow \textbf{false}$.

For a simpler case, suppose s is the node containing r_7. Then:
T_s is the empty set, because s contains no atoms.
A_s is a set containing only one atom—P_5.
M_s is the empty set because there are no rules about P_5 in T.
$\widehat{T}_s$ is also the empty set because T_s is the empty set.

The need to consider all the above sets of atoms and rules will be clarified in the sequel, when we present algorithm AM and analyze its complexity.

Algorithm AM works with partial interpretations, which are sets of literals. In the following paragraph, we will simply use the term "interpretations" instead of partial interpretations. Initially, M_s is the empty set for every s. The algorithm traverses G_T from the bottom up (that is, starting from the sources). When at a node s, it first combines all the partial models computed at the children nodes into a single set of interpretations $M_{c(s)}$. If s is a source, then $M_{c(s)}$ is set to $\{\emptyset\}$.[3] Next, for each interpretation m in $M_{c(s)}$, AM converts T_s to a knowledge base T_{s_m} using some transformations that depend on the atoms

[3] Note the difference between $\{\emptyset\}$, which is a set of one partial interpretation—the interpretation that does not assign any truth value, and $\emptyset$, which is a set that contains no interpretations.

Combine($\mathcal{I}$)
Input: A set of sets of partial interpretations $\mathcal{I} = \{I_1, \ldots, I_n\}$.
Output: The set of partial interpretations $\{i_1 + \cdots + i_n \mid \text{for } 1 \leqslant j \leqslant n, i_j \in I_j \text{ and } \{i_1, \ldots, i_n\} \text{ is a consistent set of interpretations}\}$.

1. If $\mathcal{I}$ has a single element $\{E\}$, then return E;
2. $I := \emptyset$;
3. Let $I' \in \mathcal{I}$;
4. $D :=$ Combine($\mathcal{I} \setminus \{I'\}$);
5. For each d in D, do:
   (a) For each m in I', do:
       If m and d are consistent,
       then $I := I \cup \{m + d\}$;
   (b) EndFor;
6. EndFor;
7. Return I.

Fig. 4. Procedure Combine.

in m (this is done using procedure Transform in Fig. 5, which will be described shortly); then, algorithm AM finds all the minimal models of T_{s_m} and combines them with m. The set M_s is obtained by repeating this operation for each m in $M_{c(s)}$. The procedure ALL-MINIMAL-SUBROUTINE called by AM may be any procedure that generates all minimal models.

AM uses the procedure Combine (Fig. 4), which receives as input a set of sets of interpretations $\{I_1, \ldots, I_n\}$ and returns all the consistent combinations of assignments from each set. Formally, procedure Combine returns the set $\{i_1 + \cdots + i_n \mid \text{for } 1 \leqslant j \leqslant n, i_j \in I_j \text{ and } \{i_1, \ldots, i_n\} \text{ is a consistent set of interpretations}\}$. For example, if $\mathcal{M} = \{\{\{P, \neg Q, R\}, \{\neg P, \neg Q, \neg R\}\}, \{\{P, S, U\}, \{\neg P, \neg S, U\}\}\}$ then Combine($\mathcal{M}$) = $\{\{P, \neg Q, R, S, U\}, \{\neg P, \neg Q, \neg R, \neg S, U\}\}$. If one of the sets of interpretations which Combine gets as input is the empty set, Combine will output an empty set of interpretations.

Procedure Transform (Fig. 5) gets as input a knowledge base T and a interpretation i, and it changes T to reflect the truth assignment that i represents, in the following way:

- For each positive literal P in i, each occurrence of P is deleted from the body of each rule in T.
- For each negative literal $\neg P$ in i, if P occurs in a body of some rule in T, that rules is deleted.

For example, if T is $\{P_1 \longrightarrow P_2 \vee Q_2,\ P_2 \longrightarrow P_3 \vee Q_2,\ P_2 \wedge Q_2 \longrightarrow \textbf{false}\}$ and i is $\{\neg P_1, P_2\}$, Transform(T, i) returns $\{P_3 \vee Q_2,\ Q_2 \longrightarrow \textbf{false}\}$. Here is another example using the running example.

Example 3.2. Suppose s is the node including $P_2, P_3,\ Q_2$, and r_3. Then, as discussed in Example 3.1, T_s is the following knowledge base:

Transform(T, m)
Input: A knowledge base T and an interpretation m;
Output: A transform of T where all the atoms having a truth value in m are deleted.

1. For each positive literal P in m,
   For each rule r in T
   Delete each occurrence of P from the body of r;
2. For each negative literal $\neg P$ in m,
   For each rule r in T
   If P occurs in the body of r, delete r from T.

Fig. 5. Procedure Transform.

r_2: $P_1 \longrightarrow P_2 \vee Q_2$

r_3: $P_2 \longrightarrow P_3 \vee Q_2$

r_5: $P_2 \wedge Q_2 \longrightarrow$ **false**.

Assume $i_1 = \{P_1, \neg Q_1\}$ and $i_2 = \{\neg P_1, Q_1\}$. Then:
Transform(T_s, i_1) *is*

r_2: $P_2 \vee Q_2$

r_3: $P_2 \longrightarrow P_3 \vee Q_2$

r_5: $P_2 \wedge Q_2 \longrightarrow$ **false**.

Transform(T_s, i_2) *is*

r_3: $P_2 \longrightarrow P_3 \vee Q_2$

r_5: $P_2 \wedge Q_2 \longrightarrow$ **false**.

We next show that Algorithm AM is correct.

Theorem 3.3. *Algorithm AM is correct, that is, m is a minimal model of a knowledge base T iff m is generated by AM when applied to T.*

Proof. Assume without loss of generality that G_T has a single sink s (to get a single sink, we can add to the program the rule $P \longleftarrow P_1, \ldots, P_k$, where atoms $P_1, \ldots, P_k$ are all the atoms that appear in the sinks and P is a new atom. Note that any sink will have at least one S-node). Let $s_0, s_1, \ldots, s_n$ be the ordering of the nodes of the super dependency graph by which the algorithm is executed. Denote by K_{s_i} the portion of the knowledge base composed of rules that only use atoms from A_{s_i}. We will show by induction on i that AM, when at node s_i, generates all and only the minimal models of K_{s_i}.

Case $i = 0$. S_0 is a course and hence has no children. Therefore $M_{c(s_0)}$ is set to $\{\emptyset\}$ at step 2(c). The loop in step 2(d) is executed once with $m = \emptyset$. Transform returns $\widehat{T}_{s_0} = T_{s_0}$ and ALL-MINIMAL subroutine returns the set of minimal models of T_{s_0}. Since m is the empty set, Combine returns this set as well. Note that for a source s_i, $T_{s_i} = K_{s_i}$.

Case $i > 0$. First, we will show that every minimal model is generated. Assume by contradiction that m' is a minimal model of K_{s_i} not generated by AM.

Claim 1. *Suppose m' is a minimal model of K_{s_i}, and let c be a child of s_i. Let m_1 be the interpretation resulting from projecting m' on A_c. Then m_1 is a minimal model of K_c, the portion of the knowledge base composed of rules that only use atoms from A_c.*

Proof of Claim 1. Assume by contradiction that m_1 is not a minimal model of K_c. Then there must be another model of K_c, m_2, such that $m_2 \subset m_1$. We claim that $m'' = m_2 \cup (m' - m_1)$ is a model of K_{s_i}. Clearly, all rules in K_c are satisfied by m''. Rules in K_{s_i} having no atoms from A_c are also satisfied. All other rules from $K_{s_i} - K_c$ having atoms from A_c must have these atoms in the body. Since $m_2 \subset m_1$, m'' must satisfy these rules as well. Since $m'' \subset m'$, m' is not minimal—a contradiction. □

According to the above claim, for each child c of s_i, the interpretation resulting from projecting m' on A_c is a minimal model of K_c, the portion of the knowledge base composed of rules that only use atoms from A_c. By the induction hypothesis, all these models are generated at previous steps of the algorithm, and hence the interpretation m_c, resulting from projecting m' on $\bigcup_{c \text{ is a child of } s_i} A_c$, must belong to $M_{c(s_i)}$ as defined in step 2(c) of the algorithm. It is easy to verify that if this is the case, m' must be generated by AM, a contradiction to our initial assumption.

Next we will show that every model generated by AM is minimal. Let m be a model generated by AM at node s_i, we will show that m is a minimal model of K_{s_i}. Assume by contradiction that m is not minimal. Then there must be a model m' of K_{s_i} such that $m' \subset m$. Let E be the set of all atoms P such that $m(P) = \textbf{true}$ and $m'(P) = \textbf{false}$. By the induction hypothesis, for every child c of s_i, the projection of m on A_c is a minimal model of K_c. By Claim 1, the projection of m' on A_c is a model of K_c. So it must be the case that $E \cap A_c$ is empty. By the way step 2(d) of AM is executed, E cannot be a subset of $A_{s_i} - \bigcup_{c \text{ is a child of } s_i} A_c$, a contradiction. □

Example 3.4 (*Running example cont.*). Suppose algorithm AM is executed with the knowledge base T_0 of Example 2.1 as input. The super dependency graph of this knowledge base is in Fig. 2. Suppose the order of nodes visited is $\{P_1, Q_1\}$, $\{r_2\}$, $\{r_3, P_2, Q_2, P_3\}$, $\{r_4\}$, $\{Q_3\}$, $\{P_5\}$, $\{r_7\}$, $\{P_4, Q_4\}$.

- Since node $\{P_1, Q_1\}$ is a source, we set $M_{c(\{P_1,Q_1\})} = \{\emptyset\}$. $T_{\{P_1,Q_1\}}$ is $\{P_1 \vee Q_1\}$, and procedure Transform with input $T_{\{P_1,Q_1\}}$ and $\emptyset$ returns $T_{\{P_1,Q_1\}}$. The ALL-MINIMAL-SUBROUTINE returns the two minimal models of $T_{\{P_1,Q_1\}}$ which are $\{P_1, \neg Q_1\}$ and $\{\neg P_1, Q_1\}$, and $M_{\{P_1,Q_1\}}$ is set to $\{\{P_1, \neg Q_1\}, \{\neg P_1, Q_1\}\}$.
- Since $\{P_1, Q_1\}$ is the only child of $\{r_2\}$ and $T_{\{r_2\}}$ is empty (by definition), $M_{\{r_2\}}$ is set to $M_{\{P_1,Q_1\}}$ which is $\{\{P_1, \neg Q_1\}, \{\neg P_1, Q_1\}\}$.
- When visiting node $s = \{r_3, P_2, Q_2, P_3\}$, which has also one child ($\{r_2\}$), we set at step 2(c) $M_{c(s)}$ to be $M_{\{P_1,Q_1\}}$ which is $\{\{P_1, \neg Q_1\}, \{\neg P_1, Q_1\}\}$. As explained in Example 3.1, T_s is r_2, r_3 and r_5, or in other words, the set $\{P_1 \longrightarrow P_2 \vee Q_2$, $P_2 \longrightarrow P_3 \vee Q_2$, $P_2 \wedge Q_2 \longrightarrow \textbf{false}\}$. As shown in Example 3.2, at step 2(d), when

we first set m to be $\{P_1, \neg Q_1\}$, T_{s_m} computed by Transform is $\{P_2 \vee Q_2,\ P_2 \longrightarrow P_3 \vee Q_2,\ P_2 \wedge Q_2 \longrightarrow \textbf{false}\}$. The two minimal models of T_{s_m} are $\{P_2, \neg Q_2, P_3\}$ and $\{\neg P_2, Q_2, \neg P_3\}$, and M_s is set to be $\{\{P_1, \neg Q_1, P_2, \neg Q_2, P_3\}, \{P_1, \neg Q_1, \neg P_2, Q_2, \neg P_3\}\}$.

Next we set m to be $\{\neg P_1, Q_1\}$. As explained in Example 3.2, Transform computes T_{s_m} to be $\{P_2 \longrightarrow P_3 \vee Q_2,\ P_2 \wedge Q_2 \longrightarrow \textbf{false}\}$, and this knowledge base has exactly one minimal model—$\{\neg P_2, \neg Q_2, \neg P_3\}$. Combine combines this model with $m = \{\neg P_1, Q_1\}$ and we add $\{\neg P_1, Q_1, \neg P_2, \neg Q_2, \neg P_3\}$ to M_s. So at the end of this visit $M_{\{r_3, P_2, Q_2, P_3\}}$ is set to $\{\{P_1, \neg Q_1, P_2, \neg Q_2, P_3\}, \{P_1, \neg Q_1, \neg P_2, Q_2, \neg P_3\}, \{\neg P_1, Q_1, \neg P_2, \neg Q_2, \neg P_3\}\}$.

- Next we visit node $\{r_4\}$. Since $\{r_3, P_2, Q_2, P_3\}$ is the only child of $\{r_4\}$ and $T_{\{r_4\}}$ is empty, $M_{\{r_4\}}$ is set to be $M_{\{r_3, P_2, Q_2, P_3\}}$.
- Next we visit $\{Q_3\}$. Since this node has one child, $\{r_4\}$, we get $M_{c(\{Q_3\})} = M_{\{r_4\}}$. The knowledge base $T_{\{Q_3\}}$ is $\{P_3 \longrightarrow Q_3\}$. For simplicity, let us number the models in $M_{c(\{Q_3\})}$:
  1. $\{P_1, \neg Q_1, P_2, \neg Q_2, P_3\}$,
  2. $\{P_1, \neg Q_1, \neg P_2, Q_2, \neg P_3\}$,
  3. $\{\neg P_1, Q_1, \neg P_2, \neg Q_2, \neg P_3\}$.

  Let us now look at the loop in step (d). When m is model 1, Transform returns $\{Q_3\}$ the minimal model of which is $\{Q_3\}$, and when m is model 2 or 3, Transform returns the empty set, the minimal model of which is $\{\neg Q_3\}$. Therefore, at the end of the loop in step (d) $M_{\{Q_3\}}$ is the set of the following three models:
  1. $\{P_1, \neg Q_1, P_2, \neg Q_2, P_3, Q_3\}$,
  2. $\{P_1, \neg Q_1, \neg P_2, Q_2, \neg P_3, \neg Q_3\}$,
  3. $\{\neg P_1, Q_1, \neg P_2, \neg Q_2, \neg P_3, \neg Q_3\}$.
- We now visit node $\{P_5\}$. The node $\{P_5\}$ is a source and $T_{\{P_5\}}$ is empty and so we get that $M_{\{P_5\}}$ is $\{\{\neg P_5\}\}$.
- Since $\{P_5\}$ is the only child of $\{r_7\}$ and $T_{\{r_7\}}$ is empty, $M_{\{r_7\}}$ is set to $M_{\{P_5\}}$ which is $\{\{\neg P_5\}\}$.
- We leave it for the reader to convince herself that after visiting node $\{P_4, Q_4\}$, $M_{\{P_4, Q_4\}}$ is set to $\{\{\neg P_5, \neg P_4, Q_4\}, \{\neg P_5, P_4, \neg Q_4\}\}$.
- At the last step of the algorithm, step 3, we output all the consistent combinations of the models generated at the sinks. In this example, we will output all the six combinations of the three models in $M_{\{Q_3\}}$ and the two models in $M_{\{P_4, Q_4\}}$, because all these combinations are consistent.

You see that although the language of knowledge base T_0 uses nine atoms, the largest knowledge base for which we had to call subroutine ALL-MINIMAL (at node $\{r_3, P_2, Q_2, P_3\}$) was using only three atoms.

We will now analyze the complexity of AM. We need to consider the set $\widehat{T}_s$ because while visiting a node s during the execution of AM, we have to compute at step 2(d) all minimal models of some knowledge base T_s. The estimated time required to find all minimal models of T_s is shorter than or equal to the time required to find all minimal models of $\widehat{T}_s$, because the truth value of atoms out of s is already known at this stage of

the computation. Thus, if $\widehat{T}_s$ is a Horn knowledge base, we can find the minimal model of $\widehat{T}_s$ in polynomial time. If $\widehat{T}_s$ happens to be a single fact having n atoms, then it has at most n minimal models, and we can find all its minimal models in time $O(n^2)$. If $\widehat{T}_s$ is not Horn and not a single fact, then we can find all minimal models of $\widehat{T}_s$, in time $O(2^{2n}l)$ where n is the number of distinct atoms used in $\widehat{T}_s$ and l is the length of $\widehat{T}_s$ ($l \geqslant n$, and it takes $O(2^n * l)$ time to find all the models and then $O(2^{2n} * n)$ time to compare them to each other in order to select the minimal ones). Note that in many cases we can use algorithms with better performance as subroutines.

Based on the above analysis, to every knowledge base T, we assign a number t_T as follows. First, associate a number v_s with every node s in G_T. Assume there are d distinct atoms in $\widehat{T}_s$. If $\widehat{T}_s$ is a Horn knowledge base, then v_s is 1; else, if $\widehat{T}_s$ is a single fact, then $v_s = d$, otherwise, v_s is (2^d). Note that if s contains no ES-nodes, then T_s is by definition the empty set and therefore $\widehat{T}_s$ is Horn. So for states s having no ES-nodes, $v_s = 1$.

Now associate another number t_s with every node s. If s is a leaf node, then $t_s = v_s$. If s has children $s_1, \ldots, s_j$ in G_T, then $t_s = v_s * \max(t_{s_1} * \cdots * t_{s_j}, t_{s_1} + t_{s_j})$. Define t_T to be $t_{s_1} * \cdots * t_{s_k}$, where $s_1, \ldots, s_k$ are all the sink nodes in G_T.

Definition 3.5. A knowledge base T belongs to Ψ_j if $t_T = j$.

Theorem 3.6. *If a knowledge base T belongs to Ψ_j for some j, then it has at most j minimal models and all of them can be computed in time $O(lj^2)$, where l is the length of T.*

Proof. (*By induction on j.*) The dependency graph and the super dependency graph are both built in time linear in the size of the knowledge base. So we may only consider the time it takes to compute all minimal models with the super dependency graph given.

Case $j = 1$. $T \in \Psi_1$ means that for every node s in G_T, $\widehat{T}_s$ is a Horn knowledge base. In other words, T is Horn and therefore has exactly one minimal model. It is known that a minimal model of a Horn knowledge base can be computed in time $O(l)$ [11,19].

Case $j > 1$. By induction on h, the number of nodes in the super dependency graph of T.

Case $h = 1$: Let s be the single node in G_T. Thus, $T = T_s = \widehat{T}_s$ and $j = v_s$. If T is a single fact, then $j = d$, where d is the number of distinct atoms in T. Obviously, a single fact has exactly d minimal models (all possible models where exactly one of the atoms in the fact is true), and these models can be computed in time $O(d^2)$. If T is not a single fact, then v_s is (2^d) where there are d distinct atoms in T.

Clearly, all minimal models of T can be found in time $O(lv_s^2)$, and T has at most v_s models, and hence at most v_s minimal models.

Case $h > 1$: First, assume that G_T has a single sink s. Let $c_1, \ldots, c_k$ be the children of s. For each child c_i, K_{c_i}, the part of the knowledge base which corresponds to the subgraph rooted by c_i, must belong to Ψ_{t_i} for some t_i. By the induction hypothesis, for each child node c_i, all minimal models of K_{c_i} can be computed in time $O(lt_i^2)$, and K_{c_i} has at most t_i minimal models.

By definition, $j = v_s * \max(t_1 * \cdots * t_k, t_1 + \cdots + t_k)$. We will show that T has at most j minimal models and that using the AM algorithm, all minimal models of T can be computed in time $O(lj^2)$.

By the induction hypothesis, the total time taken to compute all minimal models in the children nodes is $o(lt_1^2 + lt_2^2 + \cdots + lt_k^2) = O(l(t_1^2 + \cdots + t_k^2)) \leqslant O(lj^2)$. Now let us observe what happens when AM is visiting node s. First, the combination of all the models computed at the children nodes is taken. By the induction hypothesis, this can be executed in time $O(l * t_1 * \cdots * t_k) \leqslant O(lj)$, and yields at most $t_1 * \cdots * t_k$ interpretations in $M_{c(s)}$. For every $m \in M_{c(s)}$, we call Transform ($O(l)$)—so this step is $O(l * t_1 * \cdots * t_k) \leqslant O(lj)$ and compute all the minimal models of T_{s_m} (total of $O(lv_s^2 * t_1 * \cdots * t_k) \leqslant O(lj^2)$ steps). We then merge all the minimal models of T_{s_m} with m using Combine ($O(lv_s)$ for each m, and total of $O(l * v_s * t_1 * \cdots * t_k) \leqslant O(lj)$ time for all the models in $M_{c(s)}$). Since T_{s_m} has at most v_s minimal models, this last step yields at most $v_s * t_1 * \cdots * t_k = v_s * t_1 * \cdots * t_k \leqslant j$ minimal models for T. Thus, the overall computation of all the minimal models of T yields at most j minimal models and takes a finite number of steps, each of which is $O(lj^2)$ time, and hence it is $O(lj^2)$ time.

If G_T has multiple sinks, then we can artificially add to the knowledge base the rule $P \longleftarrow s_1, \ldots, s_k$, where $s_1, \ldots, s_k$ are all atoms used in theories in all the sinks and P is a new atom. All minimal models of the revised knowledge base, the value given to P ignored, are minimal models of the original knowledge base. Since for the new state $\{P\}$, $\widehat{T}_{\{P\}}$ is a Horn knowledge base, using the induction hypothesis we realize that T has at most $(t_{s_1} * \cdots * t_{s_j}$ models, where $s_1, \ldots, s_k$ are all the sink nodes in G_T, and all these models can be computed in time $O((t_{s_1} * \cdots * t_{s_j})^2)$. □

Note that all Horn theories belong to Ψ_1, and the largest portion of Horn rules that any knowledge base has, the more likely it is that algorithm AM will be efficient.

Given a knowledge base T, it is easy to find the minimum j such that T belongs to Ψ_j. This follows because building G_T and finding t_s for every node in G_T are polynomial-time tasks. Hence,

Theorem 3.7. *Given a knowledge base T, we can find the minimum j such that T belongs to Ψ_j in polynomial time.*

Example 3.8 (*Running example cont.*). $v_{\{P_1,Q_1\}} = 2$, since $T_{\{P_1,Q_1\}}$ is the single fact $P_1 \vee Q_1$. $v_{\{P_2,Q_2,P_3\}}$ is 2^3, since $\widehat{T}_{\{P_2,Q_2,P_3\}}$ has there distinct atoms. $v_{\{P_4,Q_4\}} = 2$, since $T_{\{P_4,Q_4\}}$ is the single fact $P_4 \vee Q_4$. $v_{\{P_5\}} = 1$ because $T_{\{P_5\}}$ is the empty set and hence it is Horn. $v_{\{Q_3\}} = 1$ because $\widehat{T}_{\{Q_3\}}$ is Horn. All other nodes s in G_{T_0} are R-nodes and hence for each of them $v_s = 1$. Thus $T \in \Psi_j$ where $j = t_{\{Q_3\}} * t_{\{P_4,Q_4\}} = 1 * 2^3 * 2 * 2 = 2^5$. Note that there are nine distinct atoms in T_0, and thus algorithm AM improves on the trivial algorithm which has a worst case of producing $O(2^d)$ models in time $O(2^{2d}l)$ where l is the length of the knowledge base and d the number of distinct atoms in the knowledge base.

Note that some interpretations generated at some nodes of the super dependency graph during the run of AM may be later deleted, since they cannot be completed to a minimal model of the whole knowledge base:

Example 3.9. Consider knowledge base T_2:

$$\begin{aligned} a \vee c \\ a \longrightarrow b \\ a \longrightarrow d \\ b \wedge d \longrightarrow \textbf{false}. \end{aligned}$$

During the run of algorithm AM, $M_{\{ac\}}$ (the set of models computed at the node $\{a, c\}$) is set to $\{\{a\}, \{c\}\}$. However, only $\{c\}$ is a minimal model of T_2.

Nevertheless, we can show that if the knowledge base has no integrity constraints, each minimal model generated at some node will be a part of a minimal model of the whole knowledge base.

Theorem 3.10. *Assume T is a knowledge base having no integrity constraints, and let s be any node in G_T, the super dependency graph of T. Then each model in M_s, computed when algorithm AM is visiting node s, can be completed to a minimal model of T.*

Proof. Let m_1 be a minimal model of K_s, the part of the knowledge base which corresponds to the subgraph rooted by s, computed when algorithm AM is visiting node s, and let T_{K_s} be the knowledge base obtained when we take $T - K_s$ and instantiate all atoms belonging to A_s according to the truth value they have in m_1. Since the atoms in A_s may appear only in bodies of rules from T_{K_s} and since T has no integrity constraints, it must be the case that T_{K_s} has no integrity constraints and hence must be consistent (every knowledge base with no integrity constraints is consistent—the interpretation that assigns **true** to all the atoms in the knowledge base is a model). Let m_2 be a model of T_{K_s}. Clearly, $m = m_1 + m_2$ is a model of T. If m is minimal, then we are done. Else, there must be a model m' of T_{K_s} such that $m' \subset m$. Let E be the set of all atoms P such that $m(P) = \textbf{true}$ and $m'(P) = \textbf{false}$. Since the projection of m on the atoms of A_s must be a model of K_s and since m_1 is a minimal model of K_s, it must be the case that $E \cap A_s$ is empty. Hence m_1 can be completed into a minimal model of T. □

Theorem 3.10 shows that if the knowledge base has no integrity constraints, then when we use the AM algorithm, we do not always have to compute all minimal models up to the root node. If the query is about an atom that is somewhere in the middle of the graph, it is enough to compute only the interpretations of the subgraph rooted by the node that includes this atom. The following example will illustrate this.

Example 3.11 (*Running example cont.*). Supposed we are given knowledge base T_0 with rule r_5, which is the only integrity constraint excluded. The reader can check that the dependency graph will be still the same as shown in Fig. 1. Now, suppose that we ask the

query "Is P_3 true in all minimal models of the knowledge base T_0?". It is enough to run algorithm AM only for the nodes $\{P_1, Q_1\}$, r_2, and $\{r_3, P_2, Q_2, P_3\}$. The computation of nodes $\{P_1, Q_1\}$ and r_2 is exactly as shown in Example 3.4. As explained in Example 3.4, when visiting node $s = \{r_3, P_2, Q_2, P_3\}$, which has also one child ($\{r_2\}$), we set $M_{c(s)}$ to be $M_{\{P_1,Q_1\}}$ which is $\{\{P_1, \neg Q_1\}, \{\neg P_1, Q_1\}\}$. But unlike Example 3.4, since r_5 is missing, T_s is composed only of r_2 and r_3, or in other words, the set $\{P_1 \longrightarrow P_2 \vee Q_2,\ P_2 \longrightarrow P_3 \vee Q_2\}$. At step 2(d), when we first set m to be $\{\{P_1, \neg Q_1\}, T_{s_m}\}$ computed by Transform is $\{P_2 \vee Q_2,\ P_2 \longrightarrow P_3 \vee Q_2\}$. The two minimal models of T_{s_m} are $\{P_2, \neg Q_2, P_3\}$ and $\{\neg P_2, Q_2, \neg P_3\}$ and M_s is set to be $\{\{P_1, \neg Q_1, P_2, \neg Q_2, P_3\}, \{P_1, \neg Q_1, \neg P_2, Q_2, \neg P_3\}\}$.

Next we set m to be $\{\neg P_1, Q_1\}$. As explained in Example 3.2, Transform computes T_{s_m} to be $\{P_2 \longrightarrow P_3 \vee Q_2\}$, and this knowledge base has exactly one minimal model—$\{\neg P_2, \neg Q_2, \neg P_3\}$. Combine combines this model with $m = \{\neg P_1, Q_1\}$ and we add $\{\neg P_1, Q_1, \neg P_2, \neg Q_2, \neg P_3\}$ to M_s. So at the end of this visit $M_{\{r_3,P_2,Q_2,P_3\}}$ is set to $\{\{P_1, \neg Q_1, P_2, \neg Q_2, P_3\}, \{P_1, \neg Q_1, \neg P_2, Q_2, \neg P_3\}, \{\neg P_1, Q_1, \neg P_2, \neg Q_2, \neg P_3\}\}$.

By Theorem 3.10, all these models are part of minimal models of the knowledge base T_0. Hence, without having to continue the computation for all the nodes in the tree we can answer that P_3 is not true in every minimal model.

In addition to the beneficial feature discussed above, Algorithm AM has other desirable features. First, AM enables us to compute minimal models in a modular fashion. We can use G_T as a structure in which to store the minimal models. Once the knowledge base is changed, we need to resume computation only at the nodes affected by the change.

Second, the AM algorithm is useful in computing the labeling of a TMS subject to nogoods. A set of nodes of a TMS can be declared *nogood*, which means that all acceptable labeling should assign **false** to at least one node in the nogood set.[4] In minimal models terminology, this means that when handling nogoods, we look for minimal models in which at least one atom from a nogood is **false**. A straightforward approach would be to first compute all the minimal models and then choose only the ones that comply with the nogood constraints. But since the AM algorithm is modular and works from the bottom up, in many cases it can prevent the generation of unwanted minimal models at an early stage. During the computation, we can exclude the partial interpretations that do not comply with the nogood constraints and erase these partial interpretations. We can do this once we are at a node s in the super dependency graph such that A_s includes all the members of a certain nogood.

4. Computing minimal models of first-order knowledge bases

In this section, we show how we can generalize algorithm AM so that it can find all minimal models of a knowledge base over a first-order language with no function symbols. The new algorithm will be called FIRST-ALL-MINIMAL (FAM).

[4] In our terminology nogoods are simply integrity constraints, and can be added directly to the knowledge base.

We will now refer to a knowledge base as a set of rules of the form

$$A_1 \wedge A_2 \wedge \cdots \wedge A_n \longrightarrow B_1 \vee B_2 \vee \cdots \vee B_m \tag{4}$$

where all As and Bs are atoms in a *first-order* language $\mathcal{L}$ with no function symbols. The definitions of head, body, facts, and integrity constraints are analogous to the propositional case. In the expression $p(X_1, \ldots, X_k)$, p is called a *predicate name*.

As in the propositional case, every knowledge base T is associated with a directed graph called the *dependency graph* of T, in which we have predicates names instead of atoms. The super dependency graph, G_T, is defined in an analogous manner.

A knowledge base will be called *safe* iff each of its rules is safe. A rule is *safe* iff all the variables appearing in the head of the rule also appear in the body of the rule. In this section, we assume that knowledge bases are safe. The *Herbrand base* of a knowledge base is the set of all atoms constructed using predicate names and constant symbols[5] from the knowledge base. The set of *ground instances of a rule* is the set of rules obtained by consistently substituting variables from the rule with constant symbols that appear in the knowledge base in all possible ways. The *ground instance of a knowledge base* is the union of all ground instances of its rules. Note that the ground instance of a first-order knowledge base can be viewed as a propositional knowledge base.

A (Herbrand) *model* for a knowledge base is a subset M of the knowledge base's Herbrand base having

1. For every rule with non-empty head in the grounded knowledge base, if all the atoms that appear in the body of the rule belong to M then at least one of the atoms in the head of the rule belongs to M.
2. For every integrity constraint, not all the atoms in the body appear in M.

A minimal model for a first-order knowledge base T is a Herbrand model of T, which is also a minimal model of the grounded version of T.

We now present FAM, an algorithm that computes all minimal models of a first-order knowledge base. Let T be a first-order knowledge base. As in the propositional case, with each node s in G_T (the super dependency graph of T), we associate T_s, A_s, and M_s. T_s is the subset of T containing all the rules about predicates whose names are in s. A_s is the set of all predicate names that appear in the subgraph of G_T rooted by s. M_s are the minimal models associated with the sub-knowledge base of T that contains only rules about predicates whose names are in A_s. Initially, M_s is empty for every s. Algorithm FAM traverses G_T from the bottom up. When at a node s, the algorithm first combines all partial models computed by the children of s into a single set of models, $M_{c(s)}$. Then, for each model m in $M_{c(s)}$, it calls a procedure that finds all the minimal models of T_s union the set of all the clauses **true** $\longrightarrow P$ such that $P \in m$. The procedure ALL-MINIMAL called by FAM can be any procedure that computes all the minimal models of a first-order knowledge base, such as one of the procedures suggested by [7]. Because procedure ALL-MINIMAL

[5] We remind the reader that the knowledge base is over a first-order language who has, by definition, a predefined set of constant symbols.

FIRST-ALL-MINIMAL(T)
Input: A first-order knowledge base T.
Output: All the minimal models of T.

1. Construct G_T;
2. Traverse G_T from the bottom up. For each node s, do:
   (a) $M_s := \emptyset$;
   (b) Let $s_1, \ldots, s_j$ be the children of s;
   (c) $M_{c(s)} :=$ Combine($\{M_{s_1}, \ldots, M_{s_j}\}$);
   (d) For each $m \in M_{c(s)}$ do

   $$M_s := M_s \cup \text{all-minimal}(T_s \cup \{\textbf{true} \longrightarrow P \mid P \in m\});$$

3. Output Combine($\{M_{s_1}, \ldots, M_{s_k}\}$),
   where $s_1, \ldots, s_k$ are the sinks of G_T.

Fig. 6. Algorithm FIRST-ALL-MINIMAL (FAM).

computes minimal models for only parts of the knowledge base, it may take advantage of some fractions of the knowledge base being Horn or having any other property that simplifies computation of the minimal models.

Theorem 4.1. *Algorithm FAM is correct, that is, m is a minimal model of a knowledge base T iff m is one of the models in the output when applying FAM to T.*

Proof. As the proof of Theorem 3.3. □

Example 4.2. Consider knowledge base T_3:

r_1: $bird(X) \longrightarrow fly(x) \vee abnormal(X)$

r_2: $bird(X) \longrightarrow female(X) \vee male(X)$

r_3: $peacock(X) \longrightarrow bird(X)$

r_4: $peacock(X) \longrightarrow abnormal(X)$

r_5: $pheasant(X) \longrightarrow bird(X)$

r_6: $peacock(peaki)$.

The super dependency graph of T_3, G_{T_3}, is shown in Fig. 7. Observe that when at node *bird*, for example, in step 2(d) the algorithm looks for all minimal models of the knowledge base $T' = T_{bird} \cup \{\longleftarrow peacock(peaki)\}$, where $T_{bird} = \{peacock(X) \longrightarrow bird(X), pheasant(X) \longrightarrow bird(X)\}$. T' is a Horn knowledge base that has a unique minimal model that can be found efficiently. Hence, algorithm FAM saves us from having to ground all the rules of the knowledge base before starting to calculate the models, and it can take advantage of parts of the knowledge base being Horn.

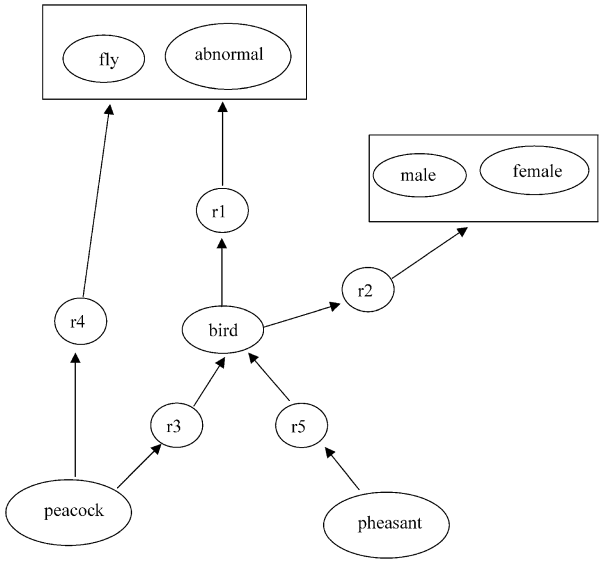

Fig. 7. The super dependency graph of T_3.

5. Related work

During the last few years there have been several studies regarding the problem of minimal model computation. Ben-Eliyahu and Dechter [3] have presented several algorithms for computing minimal models, all of them different from the one presented here. One limitation of the algorithms presented there is that they produce a *superset* of all minimal models while every model produced using our algorithm is minimal. In addition, for each of the algorithms presented by [3] we can show a set of theories for which our algorithm performs better. For example, one of the heuristics employed in [3] is to convert a knowledge base into a Horn knowledge base by instantiation of some of the atoms. This heuristics works well when the knowledge base is close to being Horn, or in other words, when only few atoms should be instantiated. Consider the following knowledge base:

$$\begin{aligned} A \vee C \\ A &\longrightarrow B \\ C &\longrightarrow D \\ B \wedge D &\longrightarrow X_1 \vee \cdots \vee X_n \end{aligned}$$

and assume A, B, C or D are not one of $X_1, \ldots, X_n$. By the heuristics of [3], all combinations of instantiating the variables $X_2, \ldots, X_n$ should be tried when computing all the models. This approach will work well only if n is very small. On the other hand, if algorithm AM is used for this knowledge base, it will work in linear time on such theories, no matter what n is. This is because when node $\{X_1, \ldots, X_n\}$ of the dependency graph is reached during the computation of the AM algorithm, it is clear that there is no minimal model in which both B and D are true. Hence the rule $B \wedge D \longrightarrow X_1 \vee \cdots \vee X_n$ is not

considered at all during the computation. Ben-Eliyahu and Palopoli [4] have presented a polynomial algorithm for finding a minimal model, but it works only for a subclass of all CNF theories and it finds only one minimal model.

The algorithm of Ben-Eliyahu [1] for finding stable models of logic programs has several common ideas with the one presented here. However, it finds only stable models and it does not work for rules with more than one atom in the head.

Bry and Yahya [7] describe an approach for generating the minimal Herbrand models of sets of first-order clauses. Their approach builds upon positive unit hyperresolution (PUHR) tableaux. Two minimal model generation procedures are described by [7]. The first one expands PUHR tableaux depth-first relying on a complement splitting expansion rule and on a form of backtracking involving constraints. The second minimal model generation procedure performs a breadth-first, constrained expansion of PUHR (complement) tableaux. Like the algorithm presented here, both procedures described in [7] are optimal in the sense that each minimal model is constructed only once, and the construction of nonminimal models is interrupted as soon as possible. The advantage of the algorithm presented here is that for the propositional case, an upper-bound on the time complexity and the number of models generated can be assessed ahead of time in time polynomial in the size of the theory.

Variations on the task of minimal model computation have been studied in the past in the diagnosis literature and the logic programming literature. For instance, 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 prime implicates of a knowledge base 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.

Some of the work in the logic programming literature has focused on using efficient optimization techniques, such as linear programming, for computing minimal models (e.g., [6]). The systems *dlv* [21,24] and *smodels* [20,29] compute stable models of disjunctive logic programs. If integrity constraints are allowed in the programs, then every knowledge base can be represented as a disjunctive logic program such that the set of all minimal models of the first coincide with the set of all stable models of the second. The system *dlv* takes advantage of the fact that minimal model checking for HCF theories [2] can be computed in linear time [4]. The system *smodels* translates the disjunctive program into a non-disjunctive one and then calls a stable model computing procedure that uses some constraints programming techniques. One limitation of the above approaches is that they do not provide a tool to assess ahead of time how complex will be the computation. An advantage of our approach compares to theirs is that our algorithm may be implemented as a parallel algorithm and thus take advantage of distributed computing systems. Our algorithm can call the algorithms of *dlv* or *smodels* as a subroutine.

6. Conclusions

We have presented a new algorithm for computing minimal models. Every model generated by this algorithm is minimal, and all minimal models are eventually generated. The

algorithm induces a hierarchy of tractable subsets for the problem of minimal model computation. The minimal models can be generated by the algorithm one at a time, a property which allows demand-driven computation.

Algorithm AM enables us to compute minimal models in a modular fashion. We can use the super-dependency graph of the knowledge base as a structure in which to store the minimal models. Once the knowledge base is changed, we need to resume computation only at the nodes affected by the change. By using the AM algorithm, we do not always have to compute all minimal models up to the root node. If we are queried about an atom that is somewhere in the middle of the graph, it is often enough to compute only the models of the subgraph rooted by the node that represents this atom.

A parallel implementation of the AM algorithm is possible. Computations related to each node in the super dependency graph could be executed in different machines. A machine that represents a specific node would have to wait to get messages containing models from machines that represent the node's children. Once a particular machine P gets at least one model from each child node, models for the knowledge base represented in P can be computed and sent to the machine that represents P's parent in the super dependency graph. The final results—the minimal models of the whole knowledge base – will be delivered by the machine that represents the root of the super dependency graph. Such parallel implementation could speed up the computation, since models of nodes that are not connected by a directed path in the super dependency graph can be executed in parallel. We leave details of the implementation and analysis of parallel version of the algorithm presented here for future research.

Acknowledgements

Thanks to Chen Avin and Luigi Palopoli for useful comments on earlier drafts of this paper, and to Barbara Grosz for inviting the author to spend a year at Harvard DEAS as a visiting scholar. The author is grateful to one of the referees for providing detailed and thoughtful comments that led to a significant improvement in the paper's presentation.

References

[1] R. Ben-Eliyahu, A hierarchy of tractable subsets for computing stable models, J. Artificial Intelligence Res. 5 (1996) 27–52.
[2] R. Ben-Eliyahu, R. Dechter, Propositional semantics for disjunctive logic programs, Ann. Math. Artificial Intelligence 12 (1994) 53–87. A short version appears in JICSLP-92: Proceedings of the 1992 Joint International Conference and Symposium on Logic Programming.
[3] R. Ben-Eliyahu, R. Dechter, On computing minimal models, Ann. Math. Artificial Intelligence 18 (1996) 3–27. A short version in AAAI-93: Proceedings of the 11th National Conference on Artificial Intelligence.
[4] R. Ben-Eliyahu, L. Palopoli, Reasoning with minimal models: Efficient algorithms and applications, Artificial Intelligence 96 (1997) 421–449. A short version in KR-94.
[5] N. Bidoit, C. Froidevaux, Minimalism subsumes default logic and circumscription in stratified logic programming, in: Proceedings of the IEEE Symposium on Logic in Computer Science, LICS-87, IEEE Computer Science Press, Los Alamitos, CA, 1987, pp. 89–97.
[6] C. Bell, A. Nerode, R.T. Ng, V.S. Subrahmanian, Mixed integer programming methods for computing nonmonotonic deductive databases, J. ACM 41 (6) (1994) 1178–1215.

[7] F. Bry, A. Yahya, Positive unit hyperresolution tableaux and their application to minimal model generation, J. Autom. Reason. 25 (1) (2000) 35–82.
[8] M. Cadoli, The complexity of model checking for circumscriptive formulae, Inform. Process. Lett. 44 (3) (1992) 113–118.
[9] M. Cadoli, On the complexity of model finding for nonmonotonic propositional logics, in: A. Marchetti Spaccamela, P. Mentrasti, M. Venturini Zilli (Eds.), Proceedings of the 4th Italian Conference on Theoretical Computer Science, World Scientific, Singapore, 1992, pp. 125–139.
[10] Z. Chen, S. Toda, The complexity of selecting maximal solutions, in: Proc. 8th IEEE Int. Conf. on Structures in Complexity Theory, 1993, pp. 313–325.
[11] W.F. Dowling, J.H. Gallier, Linear time algorithms for testing the satisfiability of propositional Horn formulae, J. Logic Programming 3 (1984) 267–284.
[12] J. de Kleer, A.K. Mackworth, R. Reiter, Characterizing diagnosis and systems, Artificial Intelligence 56 (1992) 197–222.
[13] J. de Kleer, B.C. Williams, Diagnosis multiple faults, Artificial Intelligence 32 (1987) 97–130.
[14] T. Eiter, G. Gottlob, Propositional circumscription and extended closed-world reasoning are Π_2^p-complete, Theoret. Comput. Sci. 114 (1993) 231–245.
[15] C. Elkan, A rational reconstruction of nonmonotonic truth maintenance systems, Artificial Intelligence 43 (1990) 219–234.
[16] K. Fine, The justification of negation as failure, Logic, Methodology and Philosophy of Science 8 (1989) 263–301.
[17] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R.A. Kowalski, K.A. Bowen (Eds.), Logic Programming: Proceedings of the 5th International Conference, MIT Press, Cambridge, MA, 1988, pp. 1070–1080.
[18] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gen. Comput. 9 (1991) 365–385.
[19] A. Itai, J.A. Makowsky, Unification as a complexity measure for logic programming, J. Logic Programming 4 (1987) 105–117.
[20] T. Janhunen, I. Niemelä, D. Seipel, P. Simons, J.-H. You, Unfolding partiality and disjunctions in stable model semantics, CoRR cs.AI/0303009, 2003.
[21] C. Koch, N. Leone, G. Pfeifer, Enhancing disjunctive logic programming systems by sat checkers, Artificial Intelligence 151 (1–2) (2003) 177–212.
[22] P.G. Kolaitis, C.H. Papadimitriou, Some computational aspects of circumscription, J. ACM 37 (1990) 1–14.
[23] V. Lifshitz, Computing circumscription, in: IJCAI-85: Proceedings of the International Joint Conference on AI, Los Angeles, CA, 1985, pp. 121–127.
[24] N. Leone, G. Pfeifer, W. Faber, F. Calimeri, T. Dell'Armi, T. Eiter, G. Gottlob, G. Ianni, G. Ielpa, C. Koch, S. Perri, A. Polleres, The dlv system, in: Proc. JELIA, 2002, pp. 537–540.
[25] J. McCarthy, Circumscription—a form of non-monotonic reasoning, Artificial Intelligence 13 (1980) 27–39.
[26] J. McCarthy, Application of circumscription to formalizing common-sense knowledge, Artificial Intelligence 28 (1986) 89–116.
[27] J. Minker, On indefinite databases and the closed world assumption, in: Proceedings of the 6th Conference on Automated Deduction, in: Lecture Notes in Computer Science, vol. 138, Springer, Berlin, 1982, pp. 292–308.
[28] R. Reiter, A logic for default reasoning, Artificial Intelligence 13 (1980) 81–132.
[29] P. Simons, I. Niemelä, T. Soininen, Extending and implementing the stable model semantics, Artificial Intelligence 138 (1–2) (2002) 181–234.
[30] R. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Computing 1 (1972) 146–160.