

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, Ψ_1, Ψ_2, \dots , 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 $O(lnk)$, where l is the length of the knowledge base and n the number of atoms in T ; 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.

2 Preliminary Definitions

2.1 syntax

We assume that all rules are in the form

$$A_1 \wedge \dots \wedge A_n \longrightarrow B_1 \vee \dots \vee B_m \quad (1)$$

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

$$A_1 \wedge \dots \wedge A_n \longrightarrow \text{false},$$

and is called *an integrity constraint* (name borrowed from database terminology). When $n = 0$ (1) becomes

$$\text{true} \longrightarrow B_1 \vee \dots \vee B_m,$$

or simply:

$$B_1 \vee \dots \vee B_m,$$

and is called a fact. When both m, n are 0, (1) is equal to **false**. The rule (1) is *Horn* whenever $m \leq 1$.

It is easy to see that our language covers all CNF propositional formulas. Hence, for any propositional formula there is a logically equivalent formula in our language.

2.2 The dependency graph

We will divide all the atoms in the knowledge base to equivalence sets as follows:

- If P and Q are in the head of the same rule, then P and Q are in the same set.
- If P and Q are both in the body of the same integrity constraint, they are in the same set.
- If P and Q are both unconstrained, they are in the same set. An atom P is *unconstrained* iff it appears in no integrity constraint and in no head of any rule in the theory.

It's easy to see that all the equivalence sets make up a partition of all the atoms in the knowledge base.

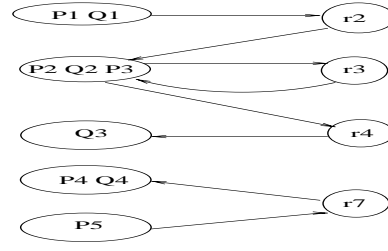


Figure 1: The dependency graph of T_0

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

$$\begin{aligned} r_1 : & P_1 \vee Q_1 \\ r_2 : & P_1 \longrightarrow P_2 \vee Q_2 \\ r_3 : & P_2 \longrightarrow P_3 \vee Q_2 \\ r_4 : & P_3 \longrightarrow Q_3 \\ r_5 : & P_2 \wedge Q_2 \longrightarrow \text{false} \\ r_6 : & P_4 \vee Q_4 \\ r_7 : & P_5 \longrightarrow P_4. \end{aligned}$$

Note that P_5 is the only unconstrained atom. The equivalence sets we get are:

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

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

Nodes: there are two types of nodes:

1. each equivalence set is a node, called *ES-node*.
2. each rule having nonempty head and nonempty body is a node, called *R-node*.

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

Example 2.2 The dependency graph of the knowledge base of Example 2.1 is shown in figure (1).

With each ES-node s in the dependency graph, we associate a subset of rules from T , called T_s . T_s is all the integrity constraints over the atoms in s and all the rules in which the atoms from s appear in the head. For example, T_{s_1} is r_1 and T_{s_2} is r_2, r_3 , and r_5 .

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

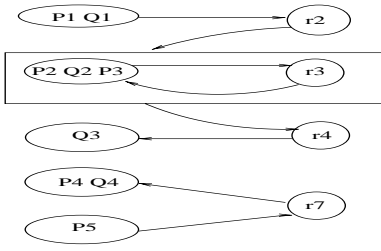


Figure 2: The super dependency graph of T_0

to one of the nodes in v in the dependency graph of T . Note that G_T is an acyclic graph.

The strongly connected components 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 linear time (Tarjan 1972).

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

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

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

If $I(P) = J(P)$ for every $P \in A \cap B$, we say that I and J are *consistent*.

A *partial* interpretation is a truth assignment over a subset of the atoms. Hence, a partial interpretation can be represented as a consistent set of literals: positive literals represent the atoms that are true, negative literals the atoms that are false, and the rest are unknown. A model for a knowledge base (set of rules) in propositional logic is a truth assignment 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 $m' \subset m$. A knowledge base will be called *Horn* iff all its rules are Horn. A Horn knowledge base has a unique minimal model (if it has a model at all) that can be found in linear time (Dowling & Gallier 1984).

3 The algorithm

Algorithm ALL-MINIMAL (AAM) in Figure 3 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 can use any algorithm for computing minimal models as subroutine.

Let T be a knowledge base. With each node s in G_T (the super dependency graph of T), we associate T_s , A_s , M_s , and \hat{T}_s . T_s is the subset of T containing all the rules about the atoms in s (as explained in section 2), A_s is the set of all atoms 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 . The definition of \hat{T}_s is more involved: we define \hat{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 $T_s = \{b \rightarrow a, a \wedge d \rightarrow c, a\}$ and $s = \{a, c\}$, then $\hat{T}_s = \{a, a \rightarrow c\}$. While visiting a node s during the execution of AAM, we have to compute at step 1.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 \hat{T}_s , because the truth value of atoms out of s is already known at this stage of the computation. Thus, if \hat{T}_s is a Horn knowledge base, we can find the minimal model of \hat{T}_s , and hence of T_s , in polynomial time. If \hat{T}_s is not Horn, then we can find all minimal models of \hat{T}_s , and hence of T_s , in time $O(2^{2^n})$ where n is the number of atoms used in \hat{T}_s . Note that in many cases we can use algorithms with better performance as subroutines.

Initially, M_s is empty 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 submodels of the children of s into a single set of models $M_{c(s)}$. If s is a source, then $M_{c(s)}$ is set to $\{\emptyset\}^2$. Next, for each model m in $M_{c(s)}$, AAM converts T_s to a knowledge base $T_{s,m}$ using some transformations that depend on the atoms in m ; then, it 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 $M_{c(s)}$. AAM uses the procedure CartesProd (Figure 4), which receives as input several sets of models and returns the consistent portion of their Cartesian product. If one of the sets of models which CartesProd gets as input is the empty set, CartesProd will output an empty set of models. The procedure Convert gets as input a knowledge base T and a model m , and performs the following: for each positive literal P in m , each occurrence of P is deleted from the body of each rule in T . The procedure ALL-MINIMAL-SUBROUTINE called by AAM may be any procedure that generates all minimal models.

Theorem 3.1 *Algorithm AAM is correct, that is, m is a minimal model of a knowledge base T iff m is generated by AAM when applied to T .*

²Note the difference between $\{\emptyset\}$, which is a set of one model - the model that assigns **false** to all the atoms, and \emptyset , which is a set that contains no models.

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

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

Figure 3: Algorithm ALL-MINIMAL (AAM)

CartesProd(\mathcal{M})
Input: A set of sets of models \mathcal{M} .
Output: A set of models which is the consistent portion of the Cartesian product of the sets in \mathcal{M} .

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

Figure 4: Procedure CartesProd

Proof: (sketch) Let s_0, s_1, \dots, s_n be the ordering of the nodes of the super dependency graph by which the algorithm is executed. We can show by induction on i that AAM, when at node s_i , generates all and only the minimal models of the portion of the knowledge base composed of rules that only use atoms from A_{s_i} . \square

We will now analyze the complexity of AAM. With each knowledge base T , we associate a number t_T as follows. Associate a number v_s with every node in G_T . If \hat{T}_s is a Horn knowledge base, then v_s is 1; else, v_s is (2^{2^n}) , where n is the number of atoms that appear in \hat{T}_s . 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, \dots, s_j in G_T , then $t_s = v_s * t_{s_1} * \dots * t_{s_j}$. Define t_T to be $t_{s_1} * \dots * t_{s_k}$, where s_1, \dots, s_k are all the sink nodes in G_T .

Definition 3.2 A knowledge base T belongs to Ψ_j if $t_T = j$.

Theorem 3.3 If a knowledge base belongs to Ψ_j for some j , then it has at most j minimal models that can be computed in time $O(\ln j)$, where l is the length of T and n is the number of atoms used in T .

The proof is omitted due to space constraints.

Note that all Horn theories belong to Ψ_1 , and the more that any knowledge base looks Horn, the more efficient algorithm AAM will be.

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

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

Example 3.5 Consider knowledge base T_2 :

$$\begin{array}{rcl}
& & a \vee c \\
a & \longrightarrow & b \\
a & \longrightarrow & d \\
b \wedge d & \longrightarrow & \text{false}
\end{array}$$

During the run of algorithm AAM, 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 there are no integrity constraints in the nodes that follows, each minimal model generated at some node will be a part of a minimal model of the whole knowledge base.

Despite the deficiency illustrated in Example 3.5, algorithm AAM does have desirable features. First, AAM 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, in using the AAM 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.

Third, the AAM 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.³ 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 AAM 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 submodels that do not comply with the nogood constraints and erase these submodels from M_s 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 AAM 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 (FAAM).

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

$$A_1 \wedge A_2 \wedge \dots \wedge A_n \longrightarrow B_1 \vee B_2 \vee \dots \vee B_m \quad (2)$$

where all A s and B s are atoms in a *first-order* language with no function symbols. The definitions of head, body, facts, and integrity constraints are analogous to the propositional case. In the expression $p(X_1, \dots, X_k)$, 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 predicate 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 constants 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 constants 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.

³In our terminology nogoods are simply integrity constraints, and can be added directly to the knowledge base.

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

1. Traverse G_T from the bottom up. For each node s , do:
 - (a) $M_s := \emptyset$;
 - (b) Let s_1, \dots, s_j be the children of s ;
 - (c) $M_{c(s)} := \text{CartesProd}(\{M_{s_1}, \dots, M_{s_j}\})$;
 - (d) For each $m \in M_{c(s)}$ do
 $M_s := M_s \cup \text{all-minimal}(T_s \cup \{\text{true} \longrightarrow P \mid P \in m\})$
2. Output $\text{CartesProd}(\{M_{s_1}, \dots, M_{s_k}\})$, where s_1, \dots, s_k are the sinks of G_T .

Figure 5: Algorithm FIRST-ALL-MINIMAL (FAAM)

A *model* for a knowledge base is a subset M of the knowledge base's Herbrand base having the following two properties:

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 FAAM, 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 P 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 FAAM traverses G_T from the bottom up. When at a node s , the algorithm first combines all the submodels of 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 FAAM can be any procedure that computes all the minimal models of a first-order knowledge base. Because procedure ALL-MINIMAL 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 of a fraction.

Theorem 4.1 *Algorithm FAAM 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 FAAM to T .*

Proof: As the proof of Theorem 3.1. □

Note that the more that a knowledge base appears Horn, the more efficient algorithm FAAM becomes.

5 Related Work

During the last few years there have been several studies regarding the problem of minimal model computation. Ben-Eliyahu and Dechter (Ben-Eliyahu & Dechter 1996) 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 (Ben-Eliyahu & Dechter 1996) we can show a set of theories for which our algorithm performs better. A more detailed comparison is omitted here because of space constraints.

Ben-Eliyahu and Palopoli (Ben-Eliyahu & Palopoli 1997) 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 (Ben-Eliyahu 1996) for finding stable models of logic programs has some 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.

Special cases of this task 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 (de Kleer & Williams 1987; de Kleer, Mackworth, & Reiter 1992) are highly complex in the worst case. To find a minimal diagnosis, they first compute all 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. 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., (Bell *et al.* 1994)). One limitation of this approach is that it does not address the issue of worst-case and average-case complexities.

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. This algorithm calls for a distributed implementation, an issue we leave for future work.

References

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

Ben-Eliyahu, R., and Dechter, R. 1996. On computing minimal models. *Annals of Mathematics and Artificial Intelligence* 18:3–27. A short version in AAAI-93: Proceedings of the 11th national conference on artificial intelligence.

Ben-Eliyahu, R., and Palopoli, L. 1997. Reasoning with minimal models: Efficient algorithms and applications. *Artificial Intelligence* 96:421–449. A short version in KR-94.

Ben-Eliyahu, R. 1996. A hierarchy of tractable subsets for computing stable models. *Journal of Artificial Intelligence Research* 5:27–52.

Cadoli, M. 1991. The complexity of model checking for circumscriptive formulae. Technical Report RAP.15.91, Università di Roma “La Sapienza”, Dipartimento di Informatica e Sistemistica. To appear in *Information Processing Letters*.

Cadoli, M. 1992. On the complexity of model finding for nonmonotonic propositional logics. In Marchetti Spaccamela, A.; Mentrasti, P.; and Venturini Zilli, M., eds., *Proceedings of the 4th Italian conference on theoretical computer science*, 125–139. World Scientific Publishing Co.

Chen, Z., and Toda, S. 1993. The complexity of selecting maximal solutions. In *Proc. 8th IEEE Int. Conf. on Structures in Complexity Theory*, 313–325.

de Kleer, J., and Williams, B. 1987. Diagnosis multiple faults. *Artificial Intelligence* 32:97–130.

de Kleer, J.; Mackworth, A.; and Reiter, R. 1992. Characterizing diagnosis and systems. *Artificial Intelligence* 56:197–222.

Dowling, W. F., and Gallier, J. H. 1984. Linear time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming* 3:267–284.

Eiter, T., and Gottlob, G. 1993. Propositional circumscription and extended closed-world reasoning are Π_2^p -complete. *Theoretical Computer Science* 114:231–245.

Elkan, C. 1990. A rational reconstruction of nonmonotonic truth maintenance systems. *Artificial Intelligence* 43:219–234.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R. A., and Bowen, K. A., eds., *Logic Programming: Proceedings of the 5th international conference*, 1070–1080. MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385.

Kolaitis, P. G., and Papadimitriou, C. H. 1990. Some computational aspects of circumscription. *J. ACM* 37:1–14.

McCarthy, J. 1980. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence* 13:27–39.

Minker, J. 1982. On indefinite databases and the closed world assumption. In *Proceedings of the 6th conference on automated deduction, Lecture Notes in Computer Science Vol. 138*, 292–308. Springer-Verlag.

Tarjan, R. 1972. Depth-first search and linear graph algorithms. *SIAM Journal on Computing* 1:146–160.