Decomposing Minimal Models

Rachel Ben-Eliyahu-Zohary
Azrieli College of Engineering,
Jerusalem, Israel
rbz@jce.ac.il

Fabrizio Angiulli,
DIMES, University of Calabria,
Rende, Italy
f.angiulli@dimes.unical.it

Fabio Fassetti and **Luigi Palopoli**
DIMES, University of Calabria,
Rende, Italy
{fassetti, palopoli}@dimes.unical.it

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. It is, therefore, crucial to be able to break this task into several subtasks that can be solved separately and in parallel. We show that minimal models of positive propositional theories can be decomposed based on the structure of the dependency graph of the theories. This observation can be useful for many applications involving computation with minimal models. As an example of such benefits, we introduce new algorithms for minimal model finding and checking that are based on model decomposition. The algorithms' temporal worst-case complexity is exponential in the size s of the largest connected component of the dependency graph, but their actual cost depends on the size of the largest source actually encountered, which can be far smaller than s, and on the class of theories to which sources belong. Indeed, if all sources reduce to an HCF or HEF theory, the algorithms are polynomial in the size of the theory.

1 Introduction

The tasks of minimal model finding and checking are central in Artificial Intelligence (AI). These computational tasks are at the heart of several knowledge representation systems, including circumscription [28; 29; 27], default logic [30], minimal diagnosis [11; 32], planning [21], multi-agents coordination [20], and logic programs under stable model semantics [17; 6; 13].

Reasoning with minimal models has been the subject of several studies in the AI community [8; 7; 24; 14; 9; 3; 25; 4; 22; 18; 1]. Given a theory T, the *Minimal Model Finding* task consists of computing a minimal model of T, whereas the *Minimal Model Checking* task is concerned with the problem of checking whether a given set of atoms is indeed a minimal model of T. Both tasks have been proven to be intractable even if only positive theories are considered [8; 7]. Therefore, we deem it relevant and interesting to single out classes of theories for which these problems can be solved efficiently [1; 5; 2]. In particular, a recent work [1] shows how it is possible to construct minimal models of positive theories by an incomplete algorithm, called IGEA, that always converges in polynomial time by either declaring success or failure, while it is guaranteed to end successfully at least on the class of HEF theories [16], which forms a significant strict superclass of HCF theories [3].

This work looks for methods to decompose a theory into disjoint subsets of clauses, such that the formidable task of minimal model computation is split between subsets of the original theory. We do so by investigating the relationship between a propositional theory and its super-dependency graph. We show that a minimal model of a theory can be generated by first computing, separately and in parallel, the minimal models of the theories corresponding to sources of the graph and then by computing the minimal models of the rest of the theory, after propagating the assignment to variables by the minimal models computed at the sources. Regarding the opposite direction, we show that given a minimal model, if its projection on a source is a minimal model of the theory corresponding to the source, then the rest of the model is a minimal model of the theory updated by the content of the minimal model computed at the sources.

To demonstrate the merits of theory decomposition, we present two new algorithms- one for minimal model generation and one for minimal model checking. The basic idea of the model generation algorithm is to compute the minimal models bottom to up while traversing the graph source following source. Intuitively, the algorithm starts with an empty model and iteratively adds to it "necessary" atoms. When a source in the graph is encountered during the computation, first the algorithm calls an external procedure like, for example, IGEA, to compute a minimal model of the sub-theory induced by that source. In many cases, this external computation will successfully terminate in polynomial time. Clearly enough, any algorithm possibly proposed in the future might be plugged into the algorithmic schema to ameliorate its performance. The model checking algorithm works in a way opposite to the model finding algorithm. It starts with a model, and it decomposes the model and the theory until both become empty, which means the model is, indeed, a minimal model of the given theory.

Noteworthy, almost all the studies mentioned above indicate that the source of intractability in minimal model finding stems from the presence of head-loops in the dependency graphs of the theories. In fact, in HCF theories no such a loop occurs, whereas in HEF theories only specific kinds of loops are allowed. Starting from this, the work reported in this manuscript presents an algorithm that finds a minimal model of any positive theory in time exponential in the size of the largest head-loop that induces a sub-theory on which the incomplete algorithm of [1] fails. In particular, when run on HEF theories, our algorithm is guaranteed to find a minimal model in polynomial time.

Note that our decomposition strategy has three related advantages: (i) even if our algorithm resorts to an exponential-time complete procedure, the procedure will be executed on just one loop and not on the whole theory, (ii) even if a theory is initially neither HEF nor HCF, while considering loops from bottom to up it may hold that a sub-theory induced by a specific loop is either HEF or HCF; this is due to the fact that the forward propagation of values of resolved atoms towards forward components may decrease their complexity, and (iii) models of theories associated with sources of the graph can be computed in parallel and then combined with the rest of the theory.

2 Preliminaries

We focus on propositional theories. We will refer to a theory as a set of clauses of the form

$$a_1 \wedge a_2 \wedge ... \wedge a_m \supset c_1 \vee c_2 \vee ... \vee c_n \tag{1}$$

where all the a's and the c's are atoms[1]. We assume that all the c's are different. The expression to the left of $\supset$ is called the *body* of the clause, while the expression to the right of $\supset$ is called the *head* of the clause. We will sometimes denote a clause by $B \supset H$, where B is the set of atoms in the body of the clause and H the set of atoms in its head. A clause is disjunctive if $n > 1$. A theory is called *positive* if, for every clause, $n > 0$. From now on, when we refer to a theory it is a positive theory.

Let X be a set of atoms. X *satisfies the body of a clause* if and only if all the atoms in the body of the clause belong to X. X *violates a clause* if and only if X satisfies the body of the clause, but none of the atoms in the head of the clause belongs to X. X is a *model* of a theory if none of its clauses is violated by X. A model X of a theory T is *minimal* if there is no $Y \subset X$, which is also a model of T. Note that positive theories always have at least one minimal model.

With every theory T we associate a directed graph, called the *dependency graph* of T, in which (a) each atom and each clause in T is a node, and (b) there is an arc directed from a node a to a clause δ if and only if a is in the body of δ. There is an arc directed from δ to a if a is in the head of δ[2].

A *super-dependency graph SG* is an acyclic graph built from a dependency graph G as follows: for each strongly connected component c in G, there is a node in SG, and for each arc in G from a node in a strongly connected component c_1 to a node in a strongly connected component c_2 there is an arc in SG from the node associated with c_1 to the node associated with c_2. A theory T is Head-Cycle-Free (HCF) if there are no two atoms in the head of some clause in T that belong to the same component in the super-dependency graph of T [3].

[1]Note that the syntax of (1) is a bit unusual for a clause; usually, the equivalent notation $\neg a_1 \vee \neg a_2 \vee ... \vee \neg a_m \vee c_1 \vee c_2 \vee ... \vee c_n$ is employed.

[2]Clause nodes in the dependency graph are mandatory to achieve a graph which is linear in the size of the theory.

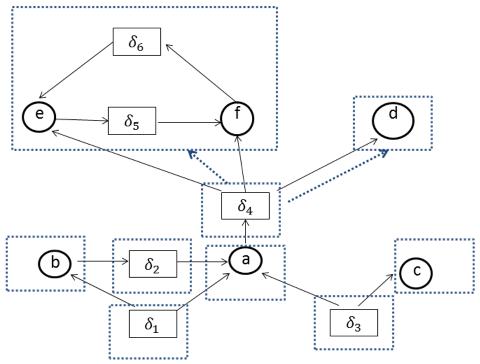

Figure 1: The [super]dependency graph of the theory T.

A *source* in a directed graph is a node with no incoming edges. By abuse of terminology, we will sometimes use the term "source" as the set of atoms in the source. *A source in a propositional theory* will serve as a shorthand for "a source in the super dependency graph of the theory." A source is called *empty* if the set of atoms in it is empty. Given a source S of a theory T, T_S denotes the set of clauses in T that uses only atoms from S.

Our algorithms use function Reduce(T, X, Y) which resembels many reasoning methods in knowledge representation, like, for example, unit propagation in DPLL and other constraint satisfaction algorithms[10; 12]. Reduce returns the theory obtained from T where all atoms in X are set to true and all atoms in Y are set to false. More specifically, `Reduce` returns the theory obtained by first removing all clauses that contain atoms in X in the head and atoms in Y in the body, and second removing all remaining atoms in $X \cup Y$ from T. So, for example, Reduce$(\{a \wedge b \supset c \vee d, c \supset d, a \supset d\},\{a\},\{c\})$ returns the theory $\{b \supset d, \supset d\}$.

Example 2.1 (Running Example) *Suppose we are given the following theory T*

$$\begin{array}{llllll} \delta_1: & a \vee b & \delta_2: & b \supset a & \delta_3: & a \vee c \\ \delta_4: & a \supset d \vee e \vee f & \delta_5: & e \supset f & \delta_6: & f \supset e \end{array}$$

In Figure 1 the dependency graph of T is illustrated in **solid** *lines. The nodes of the SG are marked with* **dotted** *lines. The arcs of the SG converge with the arcs of the dependency graph except the arcs going out of node δ_4, which are marked with dotted lines.*

3 Modular properties of minimal models

In this section we show that it is possible to compute a minimal model of a theory T by computing a minimal model of

T_S for each source S of T, and then propagating the values assigned to atoms in the source to the rest of the theory. We also prove that in some theories, some of the minimal models can be decomposed to minimal models of the sources and minimal models of the rest of the theory.

Theorem 3.1 (Theory decomposition) *Let T be a theory, let G be the SG of T. For any source S in G, let X be a minimal model of T_S. Moreover, let $T' = Reduce(T, X, S-X)$. Then, for any minimal model M' of T', $M' \cup X$ is a minimal model of T.*

Proof: The proof has two steps. We prove that (1)- $(M' \cup X)$ is a model of T and (2) - that it is minimal.

1. Assume that $(M' \cup X)$ is not a model of T. Then, there is a rule $\delta : B \supset H$ in T whose body B is fully contained in $(M' \cup X)$, and the head H has empty intersection with $(M' \cup X)$. Note that δ is not in T_S. Otherwise it would not be violated by $(M' \cup X)$, since X is a model of T_S, and no atom in T_S is in M'.

Since B is fully contained in $(M' \cup X)$, B can always be written as $(B_{M'} \cup B_X)$, where $B_{M'} = (B \cap M')$, $B_X = (B \cap X)$, and $B_{M'} \cap B_X = \emptyset$. Analogously, since H has empty intersection with $(M' \cup X)$, it can always be written as $H' \cup H_{S-X}$, where $H_{S-X} = (H \cap (S-X))$, and H' is the set of all the other atoms occurring in H.

After executing procedure `Reduce()`, T' will contain the rule $\delta' : B_{M'} \supset H'$. But, since H has an empty intersection with $(M' \cup X)$, H' has an empty intersection with M', thus δ' is violated by M', and then M' is not a model of T' which contradicts the hypothesis.

2. Assume that $(M' \cup X)$ is not a minimal model of T, then there is a nonempty set of atoms A, such that $(M' \cup X) - A$ is a model of T. In particular let A_X denote the atoms of A belonging to X and $A_{M'}$ the atoms of A belonging to M'. For A to be non-empty, $A_{M'}$ or A_X has to be non-empty. We prove that in both cases there is a contradiction.

[$A_X \neq \emptyset$] Since X is a minimal model of T_S, $(X - A_X)$ is not a model of T_S. Then, in T_S there is a clause $\delta_S : B \supset H$, such that B is fully contained in $(X - A_X)$ and no atom of H is in $(X - A_X)$. Since δ_S is in T_S, by definition of T_S no atom of H is outside S, and then no atom of H is in M'. Thus, δ_S is a clause of T_S (and then of T) whose body is contained in $X - A_X$ (and then in $M' \cup X$) and any atom in the head of δ is neither in M' nor in X. Thus, δ_S is violated by $(M' \cup X) - A$. Since $\delta_S \in T$, $(M' \cup X) - A$ is not a model of T, a contradiction.

[$A_{M'} \neq \emptyset$] Since M' is a minimal model of T', $(M' - A_{M'})$ is not a model of T'. Then, there is in T' a clause $\delta' : B \supset H$, such that B is fully contained in $(M' - A_{M'})$ and no atom of H is in $(M' - A_{M'})$.

By the way `Reduce` works, there must be in T the clause $\delta : (B \cup B_X) \supset H \cup H_{S-X}$ with B_X a possibly empty subset of X and H_{S-X} a possibly empty subset of $S - X$. This clause has the body fully contained in $(M' - A_{M'}) \cup X$ and then also in $(M' \cup X) - A)$ and no atom of its head is in $(M' \cup X) - A$. Thus, δ is violated and $(M' \cup X) - A$ is not a model of T, a contradiction. □

Theorem 3.2 (Minimal model decomposition) *Let T be a positive theory, let G be the SG of T, and let M be a minimal model of T. Moreover, assume there is a source S in G such that $X = M \cap S$ is a minimal model of T_S, and let $T' = Reduce(T, X, S-X)$. Then $M - X$ is a minimal model of T'.*

Proof: We first show that $M' = M - X$ is a model of T'. Let $B \supset H \in T'$ and assume $B \subseteq M'$. By the way `Reduce` works, there must be a possibly empty set D such that $D \subseteq X$, $(B \cup D) \supset H \in T$, and $H \cap X = \emptyset$. Since $B \subseteq M'$ and $D \subseteq X$, $B \cup D \subseteq M$, and since M must satisfy the clause $(B \cup D) \supset H$, $H \subseteq M$. Since $H \cap X = \emptyset$, $H \subseteq M - X$. Hence $B \supset H$ is satisfied by M'. Assume conversely that $M - X$ is not a minimal model of T'. Then there must be a nonempty subset of atoms W, such that $M - X - W$ is a model of T'. Note that $W \cap X = \emptyset$ and hence $X \subseteq M - W$. We show that $M - W$ is a minimal model of T, a contradiction to M being minimal. Let $B \supset H \in T$ and assume $B \subseteq M - W$. We have to show that $H \cap (M - W) \neq \emptyset$. H can be written as $H' \cup H_X \cup H_{S-X}$, where $H_X = H \cap X$, $H_{S-X} = H \cap (S-X)$, and $H' = H - S$. B can be written as $B' \cup B_X \cup B_{S-X}$, where $B_X = B \cap X$, $B_{S-X} = B \cap (S-X)$, and $B' = B - S$. Since $B \subseteq M - W$, it must be that $B_{S-X} = \emptyset$. In case $H_X \neq \emptyset$, clearly $H \cap (M - W) \neq \emptyset$ because $X \subseteq M - W$. So assume $B \supset H$ is actually of the form $(B' \cup B_X) \supset (H' \cup H_{S-X})$. Hence the clause $B' \supset H'$ must belong to T'. Since $B' \subseteq M - W$ and $B' \cap S = \emptyset$, it must be that $B' \subseteq M - X - W$. Since $M - X - W$ is a model of T', it must be that $H' \cap (M - X - W) \neq \emptyset$. So clearly $H' \cap (M - W) \neq \emptyset$. Since $H' \subseteq H$, $H \cap (M - W) \neq \emptyset$. □

4 Minimal model finding

We now show how the graph-based decompositions presented in the previous section can be exploited for minimal model finding. We first introduce algorithm `ModuMin`, which can be used to perform model finding.

Algorithm `ModuMin` uses the function $head$. Given a clause δ, $head$ returns the set of all atoms belonging to the head of δ. The algorithm works on the super-dependency graph of the theory, from bottom to up. It starts with the empty set as a minimal model and adds to it atoms only when proved to be necessary to build a model.

Theorem 4.1 *Algorithm* `ModuMin` *is correct: it outputs a minimal model of the input theory.*

The following example demonstrates how `ModuMin` works.

Example. Suppose that the theory T of Example 2.1 is given as input to `ModuMin`. At Step 1 of `ModuMin`, $M := \emptyset$. The condition in the *If* statement at Step 3 is **false** and we jump to the *Else* section in Step 6. The graph G shown in Figure 1 is built, and in Step 8 the two sources containing δ_1 and δ_3, respectively, are removed from the graph because they are empty. At Step 9, we have to choose a source in G. We can choose either b or c.

1. If we choose b**:** S is set to $\{b\}$ and in Step 10, T_S is the empty set, and so in Step 11, X is empty. In Step 12, M is still empty, and by calling `Reduce`$(T, \emptyset, \{b\})$, T becomes:

$\delta_1 : \quad a \qquad\qquad\qquad\qquad \delta_3 : \quad a \vee c$
$\delta_4 : \quad a \supset d \vee e \vee f \qquad \delta_5 : \quad e \supset f \qquad \delta_6 : \quad f \supset e$

Algorithm 1: Algorithm ModuMin

```
Input: A positive theory $T$
Output: A minimal model for $T$
1  $M := \emptyset$ ;
2  while $T \neq \emptyset$ do
3      if There is a clause $\delta$ in $T$ violated by $M$ such that
         $|\text{head}(\delta)| = 1$ then
4          let $X := \text{head}(\delta)$; $M := M \cup X$ ;
5          $T :=$ Reduce$(T, X, \emptyset)$ ;
6      else
7          let $G$ be the super-dependency graph of $T$;
8          Iteratively delete from $G$ all the empty sources ;
9          let $S$ be the set of atoms in a source of $G$ ;
10         let $T_S$ be the subset of $T$ containing all the clauses
             from $T$ having only atoms from $S$;
11         let $X$ be a minimal model of $T_S$;
12         $M := M \cup X$ ;
13         $T := T - T_S$; $T :=$ Reduce$(T, X, S - X)$;
14 return $M$
```

Now we go again to the *While* condition in Step 2. Since T is not empty, we check the *If* condition in Step 3. In Step 4 we set $X = \{a\}$ and $M = \{a\}$ and after running Reduce$(T, \{a\}, \emptyset)$, T becomes the following theory:

$\delta_4: \quad d \vee e \vee f \qquad \delta_5: \quad e \supset f \qquad \delta_6: \quad f \supset e$

Now we go again to the *While* condition in Step 2. Since T is not empty, we check the *If* condition in Step 3. The condition is **false**, and we jump to the *Else* section in Step 6. The graph G of T is built. In Step 8 the source containing δ_4 is removed from the graph because it is empty. At Step 9 we have to choose a source in G.

We can choose between two sources : $\{d\}$, and $\{e, f\}$.

1.1 If we choose $\{d\}$**:** In this case T_S is empty and so is X. In Step 12 M is still $\{a\}$. After we run Reduce$(T, \emptyset, \{d\})$, T becomes:

$\delta_4: \quad e \vee f \qquad \delta_5: \quad e \supset f \qquad \delta_6: \quad f \supset e$

Now we are left with only one source, $\{e, f\}$. T_S is T. T_S has only one minimal model which is $\{e, f\}$. So M is set to $\{a, e, f\}$. Since now T becomes empty, the algorithm terminates returning $\{a, e, f\}$ as a minimal model of the input theory.

1.2 When we choose $\{e, f\}$**:** In this case $T_S = \{e \supset f, f \supset e\}$, and the only minimal model of T_S is the empty set. So in Step 12 nothing is added to M. After running Reduce$(T, \emptyset, \{e, f\})$, T becomes a theory with only one clause, d. We then go to Step 3. In Step 4 M becomes $\{a, d\}$. Since now T becomes empty, the algorithm terminates returning $\{a, d\}$ as a minimal model of the input theory.

2. If we choose c**:** S is set to $\{c\}$. In Step 10 T_S is the empty set, so in Step 11 X is empty. In Step 12 M is still empty. By calling Reduce$(T, \emptyset, \{c\})$, T becomes:

$\delta_1: \quad a \vee b \qquad \delta_2: \quad b \supset a \qquad \delta_3: \quad a$
$\delta_4: \quad a \supset d \vee e \vee f \qquad \delta_5: \quad e \supset f \qquad \delta_6: \quad f \supset e$

Now we go to Step 2. Since T is not empty, we go to Step 3. The condition of the *if* statement is **true**, and we set $X = \{a\}$ and $M = \{a\}$. After running Reduce$(T, \{a\}, \emptyset)$, T becomes the following theory:

$\delta_4: \quad d \vee e \vee f \qquad \delta_5: \quad e \supset f \qquad \delta_6: \quad f \supset e$

Algorithm 2: Algorithm CheckMin

```
Input: A positive theory $T$ and a model $M$ of $T$
Output: true or false
1  Let $G$ be the super-dependency graph of $T$;
2  Recursively delete from $G$ all the empty sources;
3  while There is a source $S$ in $G$ such that $M \cap S$ is a minimal
     model of $T_S$ do
4      $X := M \cap S$; $M := M - X$;
         $T :=$ Reduce$(T, X, (S - X))$;
         $G :=$ the super dependency graph of $T$;
5      Recursively delete from $G$ all the empty sources;
6  if $M = \emptyset$ then
7      return true
8  else
9      return false
```

It is easy to see that taking steps as in previous cases, there are two minimal models that the algorithm might return: $\{a, d\}$ and $\{a, e, f\}$.

□

As far as the complexity of ModuMin is concerned, initially the dependency graph associated with the whole theory is considered. This graph and the related super-dependency graph can be built in linear time with respect to the size of the theory. At each iteration of the algorithm one connected component S is taken into account. At the end of each iteration the atoms in S are deleted from the theory. Thus, the number of iterations is at most linear with the theory size. As for the cost of a single iteration, it depends on the cost of computing a minimal model of the theory T_S induced by the source S considered. If, at each iteration, T_S is such that IGEA successfully outputs a minimal model, the cost of the whole algorithm is polynomial with respect to the size of the input theory. Conversely, if for one theory T_S IGEA fails, an exponential procedure should be adopted to find a minimal model of T_S and then the computational cost of the algorithm is exponential in the size of the largest connected component on which IGEA fails.

Summarizing, let n be the size of a theory T, s the size of the largest connected component, and k be the number of connected components in the dependency graph of T. The cost of ModuMin is upper bounded by

$$t^{ub}_{\texttt{ModuMin}}(n) = O(n + k \cdot 2^s).$$

5 Minimal model checking

In this section we show how the ideas of algorithm ModuMin can be adopted to solving the minimal model checking problem. The minimal model checking problem is defined as follows: *Given a theory T and a model M, check whether M is a minimal model of T.*

Algorithm CheckMin in Figure 2 can be used to check whether a model M of a theory T is a minimal model. It works through the super dependency graph of T, and it recursively deletes from M sets of atoms that are minimal models of the sources of T. T is reduced after each such deletion, to reflect the minimal models found for the sources. This process goes on until T shrinks to the empty set. When this happens, we check if M has shrunk to be the empty set as well. If this is the case, we conclude that M is indeed a minimal model of T.

As an example, suppose Algorithm CheckMin is given theory T from Example 2.1 and the model $M = \{a, d\}$. The algorithm considers the super-dependency graph G in Figure 1 bottom to up. First it removes the empty sources δ_1 and δ_3, and then it checks whether there is a source S, such that $S \cap M$ is a minimal model of T_S. The source $\{b\}$ is a good candidate because $T_{\{b\}}$ is empty, (there are no clauses in T written with the atom b only), $M \cap \{b\} = \emptyset$, and the empty set is a minimal model of the empty set of clauses. So following the commands inside the *While* loop, M does not change, T shrinks to be:

$$\begin{array}{llllll} \delta_1: & a & & & \delta_3: & a \lor c \\ \delta_4: & a \supset d \lor e \lor f & \delta_5: & e \supset f & \delta_6: & f \supset e \end{array}$$

and the source $\{b\}$ is removed from the graph. Then the source δ_2 is removed from G, because it is an empty source. We now have two sources: $\{a\}$ and $\{c\}$. $M \cap \{a\} = \{a\}$ and $\{a\}$ is indeed a minimal model of $T_{\{a\}}$ which is $\delta_1 : a$. So, following the commands inside the *While* loop, M shrinks to be $\{d\}$, and T shrinks to be: $\delta_4: \quad d \lor e \lor f \quad \delta_5: \quad e \supset f \quad \delta_6: \quad f \supset e$ and the sources $\{a\}$ and $\{c\}$ are removed from the graph. Next, we delete the source $\{\delta_4\}$ because it is an empty source. We are left with two sources: $\{d\}$ and $\{e, f\}$. The source $\{e, f\}$ is a good candidate because $T_{\{e,f\}}$ is $\{\delta_5, \delta_6\}$, $M \cap \{e, f\} = \emptyset$, and the empty set is a minimal model of the theory that consists of δ_5 and δ_6. Following the commands inside the *While* loop, M does not change, and T shrinks to be a theory that consists of the clause d. $\{d\}$ is the only source left in the graph and $M \cap \{d\} = \{d\}$ is the only minimal model of d. Following the commands inside the *While* loop, both M and T shrink to be the empty set, and the algorithm terminates returning **true**. The proof of the following theorem is straightforward given the correctness of the algorithm ModuMin. It is also clear that the time complexity of CheckMin is the same as the time complexity of ModuMin.

Theorem 5.1 *If algorithm* CheckMin *returns* **true** *when given a theory T and a model of T,M, then M is a minimal model of T.*

6 Completeness

In this section we discuss the benefits and the limitations of the algorithms presented.

An important question is, “Can algorithm ModuMin generate any minimal model of a given input theory T?” The answer is that while ModuMin is guaranteed to return a minimal model, for some theories there are minimal models that will never be generated by ModuMin. Consider the following example.

Example 6.1 *Let T' be the theory $\{c, c \supset b \lor a, a \supset d, d \supset c\}$. This theory has two minimal models: $\{c, b\}$ and $\{c, a, d\}$. However, in the graph of the theory the component $\{c, a, d\}$ precedes the component $\{b\}$, and therefore algorithm* ModuMin *will always pick the component $\{c, a, d\}$ before it picks the component $\{b\}$. Therefore the minimal model $\{c, a, d\}$ will never be generated by* ModuMin*. Moreover, if the algorithm* CheckMin *gets as input the theory T' and the minimal model $\{c, b\}$, it will return* **true**. *However, when given T' and the model $\{c, a, d\}$,* CheckMin *will return* **false**.

Clearly, there are theories for which ModuMin is complete. An example is theory T from Example 2.1. We have shown in Example 1 that all its minimal models can be generated. It would be useful to identify the class of theories for which ModuMin is complete. We will now define a subset of theories for which algorithms ModuMin and CheckMin are complete. Note that such a subset is orthogonal to the known class of HCF theories, because the theory T' above, for which the algorithms are not complete, is HCF, while the theory T from Example 2.1 is not HCF, and for T the algorithms are complete.

The question remains if we can find cases in which the algorithms will be complete. We provide a partial answer here, and leave the rest for further investigations.

We first define recursively a property called the *Modular property*.

Definition 6.2
1. *A minimal model M of a positive theory T has the Modular property with respect to T, if the SG of T has only one component.*
2. *A minimal model M of a positive theory T has the Modular property with respect to T, if there is a source S in T such that $X = M \cap S$ is a minimal model of T_S, and $M - X$, which is a minimal model of $T' =$ Reduce$(T, X, S - X)$ according to Theorem 3.2, has the Modular property with respect to T'.*

The following theorems hold:

Theorem 6.3 *Let T be the theory which is input into the algorithm* ModuMin. *If every minimal model of T has the modular property w.r.t. T, then* ModuMin *is complete for T.*

Theorem 6.4 *Assume the theory T and a minimal model M of T are given as input to the algorithm* CheckMin. *If M has the modular property w.r.t. T, then* CheckMin *will return* **true**.

Theorems 6.3 and Theorem 6.4 give us a useful analysis of the cases in which the algorithms presented in this manuscript are complete. They guide us to look for subclasses of theories with respect to which any minimal model has the modular property. One example is theories that have the *OSH Property*, defined next.

Definition 6.5 (one-source-head (OSH) Property) *A theory T has the* one-source-head (OSH) Property *if there is a source S in T such that for every atom $P \in S$, if P is in the head of some clause δ in T, then all the other atoms in the head of δ are also in S.*

Consider, for example, Theory T from Example 6.1. This theory does not have the OSH property. The SG of T has only two sources, and the clause $c \supset b \lor a$ has atoms from both components.

Theories having the OSH property are useful for completeness:

Theorem 6.6 *If a theory T has the OSH property, then for every minimal model M of T there is a sourse S in T such that $X = M \cap S$ is a minimal model of T_S.*

Proof: Assume T has the OSH property. Then there is a source S such that for every $P \in S$, if P is in the head of some clause δ in T, then all other atoms in the head of δ are also in S. Let M be a minimal model of T. We show that

$X = M \cap S$ is a minimal model of T_S. Since M is a model of T, it is clear that X is a model of T_S. We show that X is minimal. Assume conversely that X is not minimal. Then there must be a noempty set of atoms $W \subseteq X \subseteq S$ such that $X - W$ is a model of T_S. We show that $M - W$ is a model of T, a contradiction of M being minimal. Let $(B \supset H) \in T$. If $H \cap S \neq \emptyset$. Since T has the OSH property, $H \subseteq S$, and since S is a source, it must be the case that $(B \supset H) \in T_S$, and since $X - W$ is a model of T_S and $X - W \subseteq M - W$, clearly $M - W$ satisfies $(B \supset H)$. So assume $H \cap S = \emptyset$, and assume $B \subseteq M - W$. It follows that $B \subseteq M$. Since M is a model of T, $M \cap H \neq \emptyset$. Since $H \cap S = \emptyset$ and $W \subseteq S$, it follows that $(M - W) \cap H \neq \emptyset$. So $M - W$ is a model of T, a contradiction. $\square$

Corollary 6.7 *Assume T has the OSH property, let M be a minimal model of T, let S be a source such that $X = M \cap S$ is a minimal model of T_S (note that by Theorem 6.6 there is such S), and let $T' = Reduce(T, X, S - X)$. If $M - X$ (which is a minimal model of T' according to Theorem 3.2) has the modular property w.r.t. T', then M can be generated by* `ModuMin`.

Corollary 6.8 *Assume T has the OSH property, let M be a minimal model of T, let S be a source such that $X = M \cap S$ is a minimal model of T_S (note that by Theorem 6.6 there is such S), and let $T' = Reduce(T, X, S - X)$. If $M - X$ (which is a minimal model of T' according to Theorem 3.2) has the modular property w.r.t. T', then* `CheckMin` *will return* **true** *when given T and M as input.*

The notion of OSH property has practical implications. If T and all the smaller and smaller theories generated by algorithm `CheckMin` while working on a the input theory T and a candidate minimal model M has the OSH property, then it can be certain that $CheckMin$ will return **true** if and only if M is a minimal model of T. Since the OSH property can be checked in linear time, we can easily check whether it holds for the theories generated during the execution of `CheckMin`.

7 Related Work

Many papers deal with complexity issues that rise due to the cycles in the dependency graphs of theories. There were also attempts to exploit parallelism to compute answer sets, but a different approach than here have been used [15]. In this section we discuss only the most relevant work that was not mentioned in previous sections.

The algorithms presented in this paper are based on an idea that appears in [26], where the authors show that in many cases a logic program can be divided into two parts. Our algorithm, using the superstructure of the dependency graph, exploits a specific method for splitting the program. The work of [19] is also about splitting a program into several modules to gain advantages in software development. The authors of that paper have also found that strongly connected components of the dependency graph provide a key criterion when it comes to confining program composition. Our work is different, as it focuses on computational issues and provides specific complexity results. Another difference is that the modules suggested in [19] overlap, while we split the program into disjoint sets of clauses.

In [31] the authors employ minimal model checking of strongly connected components while computing stable models of logic programs. However, the program is decomposed in a way that is different from what we present here and the paper deal with normal logic programs and not with disjunctive ones.

The dlv system described in [25; 23] also make use of program decomposition based on the strongly connected components of the dependency graph. However, they do not split the program to subprograms having disjoint sets of atoms. As a result, the upper bound for the algorithm complexity that we show here is not achieved.

The author of [2] presents a hierarchy of tractable subsets for computing stable models, which are minimal models. The idea is to exploit the structure of the theory as is reflected in its super-dependency graph, but a different algorithm is used. There are several main differences between the work of [2] and the current one. First, while we deal with disjunctive theories, that paper is about non-disjunctive ones. Second, the graph is built in a different manner. Third, the complexity estimate in [2] yields sometimes a higher complexity. Fourth, the decomposition used does not yield subtheories that are completely independent of each other. Atoms in theories that correspond to different strongly connected components may overlap.

In sum, while past algorithms for computing minimal model did make efforts to exploit the structure of the dependency graph of the theory, they did not manage to decompose the theory to totally independent sub-theories that can be computed in parallel as we do here. Hence past algorithms did not achieve the complexity analysis that we provide here, which shows that the complexity of model finding is exponential in the size of the largest strongly connected component of the dependency graph of the theory.

8 Conclusions

We have presented methods for decomposing minimal models of positive propositional theories based on the dependency graph of the theory. We have shown how those decomposing techniques can lead to efficient minimal model finding and checking for these theories.

It has long been realized that the source of complexity in computing minimal models of theories is the loops between atoms that lie in the heads of disjunctive clauses. Algorithm `ModuMin` presented in this paper enables us to compute minimal models in time complexity that is directly dependent on the size of the disjunctive head loops. `ModuMin` has other virtues as well. First, it is possible to achieve in linear time, before the computation, a non-trivial upper-bound for the time it would take to compute a minimal model of the theory. Second, since any atom that is added to the output model M is guaranteed to be part of a minimal model, we can answer some queries related to this atom before the whole model is computed. Third, while working bottom-up, we can employ AI search methods for picking the next source to compute. For example, assume each atom has a value, and we need to compute a minimal model such that the sum of values of atoms in the model is below some threshold. We can use *branch and bound* approach to do this.

References

[1] F. Angiulli, R. Ben-Eliyahu-Zohary, F. Fassetti, and L. Palopoli. On the tractability of minimal model computation for some cnf theories. *Artificial Intelligence*, 2014. doi: http://dx.doi.org/10.1016/j.artint.2014.02.003.

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

[3] R. Ben-Eliyahu and R. Dechter. On computing minimal models. *Annals of Mathematics and Artificial Intelligence*, 18:3–27, 1996.

[4] R. Ben-Eliyahu-Zohary. An incremental algorithm for generating all minimal models. *Artificial Intelligence*, 169(1):1–22, 2005.

[5] R. Ben-Eliyahu-Zohary and L. Palopoli. Reasoning with minimal models: Efficient algorithms and applications. *Artificial Intelligence*, 96(2):421–449, 1997.

[6] N. Bidoit and C. Froidevaux. Minimalism subsumes default logic and circumscription in stratified logic programming. In *Proceedings of the IEEE symposium on logic in computer science*, pages 89–97, June 1987.

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

[8] M. Cadoli. On the complexity of model finding for nonmonotonic propositional logics. In *Proceedings of the 4th Italian conference on theoretical computer science*, pages 125–139. World Scientific Publishing Co., October 1992.

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

[10] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

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

[12] R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

[13] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub. Conflict-driven disjunctive answer set solving. *KR*, 8:422–432, 2008.

[14] T. Eiter and G. Gottlob. Propositional circumscription and extended closed-world reasoning are iip2-complete. *Theor. Comput. Sci.*, 114(2):231–245, 1993.

[15] M. Gebser, B. Kaufmann, and T. Schaub. Advanced conflict-driven disjunctive answer set solving. In *IJCAI*, 2013.

[16] M. Gebser, J. Lee, and Y. Lierler. Elementary sets for logic programs. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, 2006.

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

[18] E. Giunchiglia and M. Maratea. Sat-based planning with minimal-#actions plans and "soft" goals. In *AI*IA*, pages 422–433, 2007.

[19] T. Janhunen, E. Oikarinen, H. Tompits, and S. Woltran. Modularity aspects of disjunctive stable models. *Journal of Artificial Intelligence Research*, pages 813–857, 2009.

[20] M. Kalech and G. A. Kaminka. On the design of coordination diagnosis algorithms for teams of situated agents. *Artificial Intelligence*, 171(8):491–513, 2007.

[21] H. A. Kautz, D. Mcallester, and B. Selman. Encoding Plans in Propositional Logic. In *Proceedings of the Fifth International Conference on the Principle of Knowledge Representation and Reasoning (KR'96)*, pages 374–384, 1996.

[22] L. M. Kirousis and P. G. Kolaitis. The complexity of minimal satisfiability problems. *Inf. Comput.*, 187(1):20–39, 2003.

[23] C. Koch, N. Leone, and G. Pfeifer. Enhancing disjunctive logic programming systems by sat checkers. *Artificial Intelligence*, 151(1):177–212, 2003.

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

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

[26] V. Lifschitz and H. Turner. Splitting a logic program. In *ICLP*, volume 94, pages 23–37, 1994.

[27] V. Lifshitz. Computing circumscription. In *IJCAI-85: Proceedings of the international joint conference on AI*, pages 121–127, 1985.

[28] J. McCarthy. Circumscription - a form of nonmonotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[29] J. McCarthy. Application of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[30] R. Reiter. A logic for default reasoning. *Art. Int.*, 13(1–2):81–132, 1980.

[31] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1):181–234, 2002.

[32] R. T. Stern, M. Kalech, A. Feldman, and G. M. Provan. Exploring the duality in conflict-directed model-based diagnosis. In *AAAI*, 2012.