2008

Abstract Answer Set Solvers

Yuliya Lierler
University of Nebraska at Omaha, ylierler@unomaha.edu

Abstract Answer Set Solvers

Yuliya Lierler
University of Texas at Austin
yuliya@cs.utexas.edu

Abstract

Nieuwenhuis, Oliveras, and Tinelli showed how to describe enhancements of the Davis-Putnam-Logemann-Loveland algorithm using transition systems, instead of pseudocode. We design a similar framework for three algorithms that generate answer sets for logic programs: SMODELS, ASP-SAT with Backtracking, and a newly designed and implemented algorithm SUP. This approach to describing answer set solvers makes it easier to prove their correctness, to compare them, and to design new systems.

1 Introduction

Most state-of-the-art Satisfiability (SAT) solvers are based on variations of the Davis-Putnam-Logemann-Loveland (DPLL) procedure [1]. Usually enhancements of DPLL are described fairly informally with the use of pseudocode. It is often difficult to understand the precise meaning of these modifications and to prove their properties on the basis of such informal descriptions. In [2], the authors proposed an alternative approach to describing DPLL and its enhancements (for instance, backjumping and learning). They describe each variant of DPLL by means of a transition system that can be viewed as an abstract framework underlying DPLL computation. The authors further extend the framework to the algorithms commonly used in Satisfiability Modulo Background Theories.

The abstract framework introduced in [2] describes what "states of computation" are, and which transitions between states are allowed. In this way, it defines a directed graph such that every execution of the DPLL procedure corresponds to a path in this graph. Some edges may correspond to unit propagation steps, some to branching, some to backtracking. This allows the authors to model a DPLL algorithm by a mathematically simple and elegant object, graph, rather than a collection of pseudocode statements. Such an abstract way of presenting DPLL simplifies the analysis of its correctness and facilitates formal reasoning about its properties. Instead of reasoning about pseudocode constructs, we can reason about properties of a graph. For instance, by proving that the graph corresponding to a version of DPLL is acyclic we demonstrate that the algorithm always terminates. On the other hand, by checking that every terminal state corresponds to a solution we establish the correctness of the algorithm.

The graph introduced in [2] is actually an imperfect representation of DPLL in the sense that some paths in the graph do not correspond to any execution of DPLL (for example, paths in which branching is used even though unit propagation is applicable). But this level of detail is irrelevant when we talk about correctness. Furthermore, it makes our correctness theorems more general. These theorems cover not only executions of the pseudo-code, but also some computations that are prohibited by its details.

In this paper we take the abstract framework for describing DPLL-like procedures for SAT solvers as a starting point and design a similar framework for three algorithms that generate answer sets for logic programs. The first one is the SMODELS algorithm [3], implemented in one of the major answer set solvers[1]. The other algorithm is called SUP and can be seen as a simplification of SMODELS algorithm.[2] We implemented this algorithm in the new, previously unpublished system SUP[3]. The last algorithm that we describe is ASP-SAT with Backtracking[4] [4]. It computes models of the completion of the given program using DPLL and tests them until an answer set is found.

We start by reviewing the abstract framework for DPLL developed in [2] in a form convenient for our purposes. We demonstrate how this framework can be modified to describe an algorithm for computing supported models of a logic program, and then extend it to the SMODELS algorithm for computing answer sets. We show that for a large class of programs, called tight, the graph representing SMODELS is closely related to the graph representing the application of DPLL to the completion of the program. As a step towards extending these ideas to ASP-SAT with Backtracking, we analyze a modification of the original DPLL graph that includes testing the models found by DPLL. We then show how a special case of this construction corresponds to ASP-SAT with Backtracking.

We hope that the analysis of algorithms for computing answer sets in terms of transition systems described in this paper will contribute to clarifying computational principles of answer set programming and to the development of new systems.

2 Review: Abstract DPLL

For a set σ of atoms, a *state* relative to σ is either a distinguished state *FailState* or a list M of literals over σ such that M contains no repetitions, and each literal in M has an *annotation*, a bit that marks it as a *decision* literal or not. For instance, the states relative to a singleton set $\{a\}$ of atoms are

$$\begin{array}{l} \mathit{FailState},\ \emptyset,\ a,\ \neg a,\ a^d,\ \neg a^d, a\ \neg a,\ a^d\ \neg a, \\ a\ \neg a^d,\ a^d\ \neg a^d, \neg a\ a,\ \neg a^d\ a,\ \neg a\ a^d,\ \neg a^d\ a^d, \end{array}$$

[1] SMODELS: `http://www.tcs.hut.fi/Software/smodels` .

[2] The idea of simplifying the SMODELS algorithm in this manner was suggested to us by Mirosław Truszczyński (August 2, 2007).

[3] SUP: `http://www.cs.utexas.edu/users/tag/sup`. In fact, SUP implements a more sophisticated form of the algorithm that is enhanced with learning.

[4] A more sophisticated form of this algorithm, ASP-SAT with Learning, is implemented in system CMODELS: `http://www.cs.utexas.edu/users/tag/cmodels` .

where by $\emptyset$ we denote the empty list. The concatenation of two such lists is denoted by juxtaposition. Frequently, we consider M as a set of literals, ignoring both the annotations and the order between its elements. We write l^d to emphasize that l is a decision literal. A literal l is *unassigned by* M if neither l nor $\overline{l}$ belongs to M.

If C is a disjunction (conjunction) of literals then by $\overline{C}$ we understand the conjunction (disjunction) of the complements of the literals occurring in C. We will sometimes identify C with the set of its elements.

For any CNF formula F (a set of clauses), we will define its *DPLL graph* DP_F. The set of nodes of DP_F consists of the states relative to the set of atoms occurring in F. We use the terms "state" and "node" interchangeably. If a state is consistent and complete then it represents a truth assignment for F.

The set of edges of DP_F is described by a set of "transition rules". Each transition rule has the form $M \implies M'$ followed by a condition, so that

- M and M' are symbolic expressions for nodes of DP_F, and
- if the condition is satisfied there is an edge between node M and M' in the graph.

There are four transition rules that characterize the edges of DP_F:

Unit Propagate: $M \implies M\, l$ if $C \vee l \in F$ and $\overline{C} \subseteq M$

Decide: $M \implies M\, l^d$ if l is unassigned by M

Fail: $M \implies \mathit{FailState}$ if $\begin{cases} M \text{ is inconsistent, and} \\ M \text{ contains no decision literals} \end{cases}$

Backtrack: $P\, l^d\, Q \implies P\, \overline{l}$ if $\begin{cases} P\, l^d\, Q \text{ is inconsistent, and} \\ Q \text{ contains no decision literals} \end{cases}$

Note that an edge in the graph may be justified by several transition rules.

This graph can be used for deciding the satisfiability of a formula F simply by constructing an arbitrary path leading from node $\emptyset$ until a terminal node M is reached. The following proposition shows that this process always terminates, that F is unsatisfiable if M is *FailState*, and that M is a model of F otherwise.

Proposition 1. *For any CNF formula F,*

(a) graph DP_F is finite and acyclic,

(b) any terminal state of DP_F other than FailState is a model of F,

(c) FailState is reachable from $\emptyset$ in DP_F if and only if F is unsatisfiable.

For instance, let F be the set consisting of the clauses

$$a \vee b$$
$$\neg a \vee c.$$

Here is a path in DP_F with every edge annotated by the name of a transition rule that justifies the presence of this edge in the graph:

$$\begin{array}{ll} \emptyset & \Longrightarrow\ (\mathit{Decide}) \\ a^d & \Longrightarrow\ (\mathit{Unit\ Propagate}) \\ a^d\ c & \Longrightarrow\ (\mathit{Decide}) \\ a^d\ c\ b^d & \end{array} \tag{1}$$

Since the state $a^d\ c\ b^d$ is terminal, Proposition 1(b) asserts that $\{a,c,b\}$ is a model of F. Here is another path in DP_F from $\emptyset$ to the same terminal node:

$$\begin{array}{ll} \emptyset & \Longrightarrow\ (\mathit{Decide}) \\ a^d & \Longrightarrow\ (\mathit{Decide}) \\ a^d\ \neg c^d & \Longrightarrow\ (\mathit{Unit\ Propagate}) \\ a^d\ \neg c^d\ c & \Longrightarrow\ (\mathit{Backtrack}) \\ a^d\ c & \Longrightarrow\ (\mathit{Decide}) \\ a^d\ c\ b^d & \end{array} \tag{2}$$

Path (1) corresponds to an execution of DPLL; path (2) does not, because it uses *Decide* instead of *Unit Propagate*.

Note that the graph DP_F is a modification of the *classical DPLL* graph defined in [2, Section 2.3]. It is different in three ways. First, the description of the classical DPLL graph involves a "PureLiteral" transition rule, which we have dropped. Second, its states are pairs $M \parallel F$ for all CNF formulas F. For our purposes, it is not necessary to include F. Third, in the definition of that graph, each M is required to be consistent. In case of the DPLL, due to the simple structure of a clause, it is possible to characterize the applicability of *Backtrack* in a simple manner: when some of the clauses become inconsistent with the current partial assignment, *Backtrack* is applicable. In ASP, it is not easy to describe the applicability of *Backtrack* if only consistent states are taken into account. We introduced inconsistent states in the graph DP_F to facilitate our work on extending this graph to model the SMODELS algorithm.

3 Background: Logic Programs

A *(propositional) logic program* is a finite set of rules of the form

$$a_0 \leftarrow a_1, \ldots, a_m, \mathit{not}\ a_{m+1}, \ldots, \mathit{not}\ a_n, \tag{3}$$

where each a_i is an atom. By $\mathit{Bodies}(\Pi, a)$ we denote the (multi-)set of the bodies of all rules of Π with head a. We will identify the body of (3) with the conjunction of literals

$$a_1 \wedge \ldots \wedge a_m \wedge \neg a_{m+1} \wedge \ldots \neg a_n.$$

and (3) with the implication

$$a_1 \wedge \ldots \wedge a_m \wedge \neg a_{m+1} \wedge \ldots \neg a_n \rightarrow a_0.$$

For any set M of literals, by M^+ we denote the set of positive literals from M. We assume that the reader is familiar with the definition of an answer set (stable model) of a logic program [5]. For any consistent and complete set M of literals (*assignment*), if M^+ is an answer set for a program Π, then M is a model of Π. Moreover, in this case M is a *supported* model of Π, in the sense that for every atom $a \in M$, $M \models B$ for some $B \in Bodies(\Pi, a)$.

4 Generating Supported Models

In the next section we will define, for an arbitrary program Π, a graph SM_Π representing the application of the SMODELS algorithm to Π; the terminal nodes of SM_Π are answer sets of Π. As a step in this direction, we describe here a simpler graph ATLEAST_Π. The terminal nodes of ATLEAST_Π are supported models of Π.

The set of nodes of ATLEAST_Π consists of the states relative to the set of atoms occurring in Π. The edges of the graph ATLEAST_Π are described by the transition rules *Decide*, *Fail*, *Backtrack* introduced above in the definition of DP_F and the additional transition rules[5]:

Unit Propagate LP: $M \implies M\, a$ if $a \leftarrow B \in \Pi$ and $B \subseteq M$

All Rules Cancelled: $M \implies M\, \neg a$ if $\overline{B} \cap M \neq \emptyset$ for all $B \in Bodies(\Pi, a)$,

Backchain True: $M \implies M\, l$ if $\begin{cases} a \leftarrow B \in \Pi, \\ a \in M, \\ \overline{B'} \cap M \neq \emptyset \text{ for all } B' \in Bodies(\Pi, a) \setminus B, \\ l \in B \end{cases}$

Backchain False: $M \implies M\, \overline{l}$ if $\begin{cases} a \leftarrow l, B \in \Pi, \\ \neg a \in M, \text{ and} \\ B \subseteq M \end{cases}$

Note that each of the rules *Unit Propagate LP* and *Backchain False* is similar to *Unit Propagate*: the former corresponds to *Unit Propagate* on $C \vee l$ where l is the head of the rule, and the latter corresponds to *Unit Propagate* on $C \vee l$ where $\overline{l}$ is an element of the body of the rule.

This graph can be used for deciding whether program Π has a supported model by constructing a path from $\emptyset$ to a terminal node:

Proposition 2. *For any program* Π*,*

(a) graph ATLEAST_Π *is finite and acyclic,*

(b) any terminal state of ATLEAST_Π *other than FailState is a supported model of* Π*,*

(c) FailState is reachable from $\emptyset$ *in* ATLEAST_Π *if and only if* Π *has no supported models.*

[5] The names of some of these rules follow [6].

For instance, let Π be the program

$$\begin{array}{l} a \leftarrow \mathit{not}\ b \\ b \leftarrow \mathit{not}\ a \\ c \leftarrow a \\ d \leftarrow d. \end{array} \tag{4}$$

Here is a path in ATLEAST_Π:

$$\begin{array}{ll} \emptyset & \Longrightarrow\ (\mathit{Decide}) \\ a^d & \Longrightarrow\ (\mathit{Unit\ Propagate\ LP}) \\ a^d\ c & \Longrightarrow\ (\mathit{All\ Rules\ Cancelled}) \\ a^d\ c\ \neg b & \Longrightarrow\ (\mathit{Decide}) \\ a^d\ c\ \neg b\ d^d & \end{array} \tag{5}$$

Since the state $a^d\ c\ \neg b\ d^d$ is terminal, Proposition 2(b) asserts that $\{a, c, \neg b, d\}$ is a supported model of program Π.

The assertion of Proposition 2 will remain true if we drop the transition rules *Backchain True* and *Backchain False* from the definition of ATLEAST_Π.

The transition rules defining ATLEAST_Π are closely related to procedure *Atleast* [3, Sections 4.1], which is one of the core procedures of the SMODELS algorithm.

5 Smodels

Recall that a set U of atoms occurring in a program Π is said to be *unfounded* [7] on a consistent set M of literals w.r.t. Π if for every $a \in U$ and every $B \in \mathit{Bodies}(\Pi, a)$, $M \models \neg B$ or $U \cap B^+ \neq \emptyset$.

We now describe the graph SM_Π that represents the application of the SMODELS algorithm to program Π. SM_Π is a graph whose nodes are the same as the nodes of the graph ATLEAST_Π. The edges of SM_Π are described by the transition rules of ATLEAST_Π and the additional transition rule:

$$\mathit{Unfounded}: \quad M \implies M\ \neg a \ \text{ if } \begin{cases} M \text{ is consistent, and} \\ a \in U \text{ for a set } U \text{ unfounded on } M \text{ w.r.t. } \Pi \end{cases}$$

This transition rule of SM_Π is closely related to procedure *Atmost* [3, Sections 4.2], which together with the procedure *Atleast* forms the core of the SMODELS algorithm.

The graph SM_Π can be used for deciding whether program Π has an answer set by constructing a path from $\emptyset$ to a terminal node:

Proposition 3. *For any program* Π,

(a) graph SM_Π *is finite and acyclic,*

(b) for any terminal state M of SM_Π *other than FailState,* M^+ *is an answer set of* Π*,*

(c) FailState is reachable from $\emptyset$ *in* SM_Π *if and only if* Π *has no answer sets.*

To illustrate the difference between SM_Π and ATLEAST_Π, assume again that Π is program (4). Path (5) in the graph ATLEAST_Π is also a path in SM_Π. But state $a^d\ c\ \neg b\ d^d$, which is terminal in ATLEAST_Π, is not terminal in SM_Π. This is not surprising, since the set $\{a,c,d\}$ of atoms that belongs to this state is not an answer set of Π. To get to a state that is terminal in SM_Π, we need two more steps:

$$\begin{array}{ll} \vdots & \\ a^d\ c\ \neg b\ d^d & \Longrightarrow\ (\textit{Unfounded},\ U=\{d\}) \\ a^d\ c\ \neg b\ d^d\ \neg d & \Longrightarrow\ (\textit{Backtrack}) \\ a^d\ c\ \neg b\ \neg d. & \end{array} \tag{6}$$

Proposition 3(b) asserts that $\{a,c\}$ is an answer set of Π.

The assertion of Proposition 3 will remain true if we drop the transition rules *All Rules Cancelled*, *Backchain True*, and *Backchain False* from the definition of SM_Π.

6 Sup

In this section we show how to extend the graph ATLEAST_Π by the modification of transition rule *Unfounded* so that terminal nodes of the resulting graph correspond to answer sets of Π.

The graph SUP_Π is the subgraph of SM_Π such that its nodes are the same as the nodes of the graph SM_Π and its edges are described by the transition rules of ATLEAST_Π and the following modification of the rule *Unfounded* of SM_Π:

$$\textit{Unfounded SUP}: \quad M \Longrightarrow M\ \neg a \ \text{ if } \begin{cases} \text{no literal is unassigned by } M, \\ M \text{ is consistent, and} \\ a \in U \text{ for a set } U \text{ unfounded on } M \text{ w.r.t. } \Pi \end{cases}$$

This graph can be used for deciding whether a program Π has an answer set by constructing a path from $\emptyset$: Proposition 3 remains correct after replacing graph SM_Π with SUP_Π.

The only difference between SUP_Π and SM_Π is due to the additional restriction in *Unfounded SUP*: it is applicable only to the states that assign all atoms in Π. To illustrate the difference between SUP_Π and SM_Π, assume that Π is program (4). Path (6) in SM_Π is also a path in SUP_Π. On the other hand the path

$$\begin{array}{ll} \emptyset & \Longrightarrow\ (\textit{Unfounded},\ U=\{d\}) \\ \neg d & \end{array}$$

of SM_Π does not belong to SUP_Π

We can view the graph SUP_Π as a description of a particular strategy for traversing SM_Π, i.e., an edge corresponding to an application of *Unfounded* to a state in SM_Π is considered only if a transition rule *Decide* is not applicable in this state. Note that system SMODELS implements the opposite strategy, i.e., an edge corresponding to an application of *Decide* is considered only if *Unfounded* is not applicable. Nevertheless, the strategy described by SUP_Π may be reasonable for many problems. For instance,

it is easy to see that transition rule *Unfounded* is redundant for tight programs. Furthermore, the analogous strategy has been successfully used in SAT-based answer set solvers ASSAT[6] [8] and CMODELS (see Footnote 4) [4]. These systems first compute the completion of a program and then test each model of the completion whether it is an answer set (this can be done by testing whether it contains unfounded sets). In fact, the work on ASSAT and CMODELS inspired the development of system SUP. Unlike ASSAT and CMODELS, SUP does not compute the completion of a program but performs its inference directly on the the program by means of transition rules of the graph SUP_Π.

We have implemented system SUP (see Footnote 3), whose underlying algorithm is modelled by the graph SUP_Π. In the implementation, we used

- the interface of SAT-solver MINISAT[7] (v1.12b) that supports non-clausal constraints [9] in order to implement inferences described by *Unit Propagate LP*, *All Rules Cancelled*, *Backchain True*, *Backchain False*, *Decide*, and *Fail*,
- parts of the CMODELS code that support transition rule *Unfounded SUP*.

Note that system SUP also implements conflict-driven backjumping and learning. Preliminary results available at SUP web site (see Footnote 3) comparing SUP with other answer set solvers are promising.

The implementation of SUP proofs that the abstract framework for answer set solvers introduced in this work may suggest new designs for solvers.

7 Tight Programs

We now recall the definitions of the positive dependency graph and a tight program. The *positive dependency graph* of a program Π is the directed graph G such that

- the nodes of G are the atoms occurring in Π, and
- G contains the edges from a_0 to a_i ($1 \leq i \leq m$) for each rule (3) in Π.

A program is *tight* if its positive dependency graph is acyclic. For instance, program (4) is not tight since its positive dependency graph has a cycle due to the rule $d \leftarrow d$. On the other hand, the program constructed from (4) by removing this rule is tight.

Recall that for any program Π and any assignment M, if M^+ is an answer set of Π then M is a supported model of Π. For the case of tight programs, the converse holds also: M^+ is an answer set for Π if and only if M is a supported model of Π [10].

It is also well known that the supported models of a program can be characterized as models of its completion in the sense of [11]. It turns out that for tight programs the graph SM_Π is "almost identical" to the graph DP_F, where F is the (clausified) completion of Π. To make this claim precise, we need the following terminology.

We say that an edge $M \implies M'$ in the graph SM_Π is *singular* if

- the only transition rule justifying this edge is *Unfounded*, and

[6] ASSAT: `http://assat.cs.ust.hk/`.
[7] MINISAT: `http://minisat.se/`.

- some edge $M \Longrightarrow M''$ can be justified by a transition rule other than *Unfounded*.

For instance, let Π be the program

$$\begin{array}{l} a \leftarrow b \\ b \leftarrow c. \end{array}$$

The edge

$$\begin{array}{ll} a^d\ b^d\ \neg c^d & \Longrightarrow (\mathit{Unfounded},\ U = \{a,b\}) \\ a^d\ b^d\ \neg c^d\ \neg a & \end{array}$$

in the graph SM_Π is singular, because the edge

$$\begin{array}{ll} a^d\ b^d\ \neg c^d & \Longrightarrow (\mathit{All\ Rules\ Cancelled}) \\ a^d\ b^d\ \neg c^d\ \neg b & \end{array}$$

belongs to SM_Π also.

From the point of view of actual execution of the SMODELS algorithm, singular edges of the graph SM_Π are inessential: SMODELS never follows a singular edge. By SM^-_Π we denote the graph obtained from SM_Π by removing all singular edges.

Recall that for any program Π, its completion consists of Π and the formulas that can be written as

$$\neg a \vee \bigvee_{B \in \mathit{Bodies}(\Pi,a)} B \tag{7}$$

for every atom a in Π. *CNF-Comp*(Π) is the completion converted to CNF using straightforward equivalent transformations. In other words, *CNF-Comp*(Π) consists of clauses of two kinds:

1. the rules $a \leftarrow B$ of the program written as clauses

$$a \vee \overline{B}, \tag{8}$$

2. formulas (7) converted to CNF using the distributivity of disjunction over conjunction[8].

Proposition 4. *For any tight program* Π*, the graph* SM^-_Π *is equal to each of the graphs* $\mathrm{ATLEAST}_\Pi$ *and* $\mathrm{DP}_{\mathit{CNF\text{-}Comp}(\Pi)}$.

For instance, let Π be the program

$$\begin{array}{l} a \leftarrow b,\ \mathit{not}\ c \\ b. \end{array}$$

This program is tight. Its completion is

$$(a \leftrightarrow b \wedge \neg c) \wedge b \wedge \neg c,$$

[8] It is essential that repetitions are not removed in the process of clausification. For instance, *CNF-Comp*$(a \leftarrow \mathit{not}\ a)$ is the formula $(a \vee a) \wedge (\neg a \vee \neg a)$.

and *CNF-Comp*(Π) is

$$(a \vee \neg b \vee c) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg c) \wedge b \wedge \neg c.$$

Proposition 4 asserts that, for this formula F, SM^{-}_{Π} coincides with DP_F and with $\mathrm{ATLEAST}_{\Pi}$.

From Proposition 4, it follows that applying the SMODELS algorithm to a tight program essentially amounts to applying DPLL to its completion. A similar relationship, in terms of pseudocode representations of SMODELS and DPLL, is established in [12].

8 Generate and Test

In this section, we present a modification of the graph DP_F that includes testing the models of F found by DPLL. Let F be a CNF formula, and let X be a set of models of F. The terminal nodes of the graph $\mathrm{GT}_{F,X}$ defined below are models of F that belong to X.

The nodes of the graph $\mathrm{GT}_{F,X}$ are the same as the nodes of the graph DP_F. The edges of $\mathrm{GT}_{F,X}$ are described by the transition rules of DP_F and the additional transition rules:

$$\textit{Fail GT}: \qquad M \Longrightarrow \textit{FailState} \text{ if } \begin{cases} \text{no literal is unassigned by } M, \\ M \notin X, \\ M \text{ contains no decision literals} \end{cases}$$

$$\textit{Backtrack GT}: \quad P\ l^d\ Q \Longrightarrow P\ \overline{l} \text{ if } \begin{cases} \text{no literal is unassigned by } P\ l^d\ Q, \\ P\ l^d\ Q \notin X, \\ Q \text{ contains no decision literals.} \end{cases}$$

It is easy to see that the graph DP_F is a subgraph of $\mathrm{GT}_{F,X}$. Furthermore, when the set X coincides with the set of all models of F the graphs are identical. This graph can be used for deciding whether a formula F has a model that belongs to X by constructing a path from $\emptyset$ to a terminal node:

Proposition 5. *For any CNF formula F and any set X of models of F,*

(a) graph $\mathrm{GT}_{F,X}$ is finite and acyclic,

(b) any terminal state of $\mathrm{GT}_{F,X}$ other than FailState belongs to X,

(c) FailState is reachable from $\emptyset$ in $\mathrm{GT}_{F,X}$ if and only if X is empty.

Note that to verify the applicability of the new transition rules *Fail GT* and *Backtrack GT* we need a procedure for testing whether a set of literals belongs to X, but there is no need to have the elements of X explicitly listed.

ASP-SAT with Backtracking [4] is a procedure that computes models of the completion of the given program using DPLL, and tests them until an answer set is found. The application of the ASP-SAT with Backtracking algorithm to a program Π can be viewed as constructing a path from $\emptyset$ to a terminal node in the graph $\mathrm{GT}_{F,X}$, where

- F is the completion of Π converted to conjunctive normal form, and
- X is the set of all assignments corresponding to answer sets of Π.

9 Related Work

Simons [3] described the SMODELS algorithm by means of a pseudocode and demonstrated its correctness. Gebser and Schaub [13] provided a deductive system for describing inferences involved in computing answer sets by tableaux methods. The abstract framework presented in this paper can be viewed as a deductive system also, but it is a very different system. For instance, we describe backtracking by an inference rule, and the Gebser-Schaub system doesn't. Accordingly, the derivations considered in this paper describe search process, and derivations in the Gebser-Schaub system don't. Also, the abstract framework discussed here doesn't have any inference rule similar to Cut; this is why its derivations are paths, rather than trees.

10 Proofs

Lemma 1. *For any CNF formula F and a path from $\emptyset$ to a state $l_1 \ \dots \ l_n$ in DP_F, every model X of F satisfies l_i if it satisfies all decision literals l_j^d with $j \leq i$.*

Proof. By induction on the length of a path. Since the property trivially holds in the initial state $\emptyset$, we only need to prove that all transition rules of DP_F preserve it.

Consider an edge $M \Longrightarrow M'$ where M is a sequence $l_1 \ \dots \ l_k$ such that every model X of F satisfies l_i if it satisfies all decision literals l_j^d with $j \leq i$.

Unit Propagate: M' is $M \ l_{k+1}$. Take any model X of F such that X satisfies all decision literals l_j^d with $j \leq k+1$. By the inductive hypothesis, $X \models M$. From the definition of *Unit Propagate*, for some clause $C \vee l_{k+1} \in F$, $\overline{C} \subseteq M$. Consequently, $M \models \neg C$. It follows that $X \models l_{k+1}$.

Decide: M' is $M \ l_{k+1}^d$. Obvious.

Fail: Obvious.

Backtrack: M has the form $P \ l_i^d \ Q$ where Q contains no decision literals. M' is $P \ \overline{l_i}$. Take any model X of F such that X satisfies all decision literals l_j^d with $j \leq i$. We need to show that $X \models \overline{l_i}$. By contradiction. Assume that $X \models l_i$. Since Q does not contain decision literals, X satisfies all decision literals in $P \ l_i^d \ Q$. By the inductive hypothesis, it follows that X satisfies $P \ l_i^d \ Q$, that is, M. This is impossible because M is inconsistent.

Proposition 1. *For any CNF formula F,*

(a) graph DP_F is finite and acyclic,

(b) any terminal state of DP_F other than FailState is a model of F,

(c) FailState is reachable from $\emptyset$ in DP_F if and only if F is unsatisfiable.

Proof. (a) The finiteness of DP_F is obvious. For any list N of literals by $|N|$ we denote the length of N. Any state M, other than *FailState*, has the form $M_0 \ l_1 \ M_1 \dots l_p \ M_p$, where $l_1 \dots l_p$ are all desicion literals of M; we define $\alpha(M)$ as the sequence of non-negative integers $|M_0|, |M_1|, \dots, |M_p|$, and $\alpha(\mathit{FailState}) = \infty$. For any states M and M' of DP_F, we understand $\alpha(M) < \alpha(M')$ as the lexicographical order. By the definition

of the transition rules defining the edges of DP_F, if there is an edge from a state M to M' in DP_F, then $\alpha(M) < \alpha(M')$. It follows that if a state M' is reachable from M then $\alpha(M) < \alpha(M')$. Consequently, the graph is acyclic.
(b) Consider any terminal state M other than *FailState*. From the fact that *Decide* is not applicable, we derive that M assigns all literals. Similarly, since neither *Backtrack* nor *Fail* is applicable, M is consistent. Consequently, M is an assignment. Consider any clause $C \vee l$ in F. It follows that if $\overline{C} \not\subseteq M$ then $C \cap M \neq \emptyset$. Since *Unit Propagate* is not applicable, it follows that if $\overline{C} \subseteq M$ then $l \in M$. We derive that $M \models C \vee l$. Hence, M is a model of F.
(c) Left-to-right: Since *FailState* is reachable from $\emptyset$, there is an inconsistent state M without decision literals such that there exists a path from $\emptyset$ to M. By Lemma 1, any model of F satisfies M. Since M is inconsistent we conclude that F has no models.

Right-to-left: From (a) it follows that there is a path from $\emptyset$ to some terminal state. By (b), this state cannot be different from *FailState*, because F is unsatisfiable.

Lemma 2. *For any program* Π *and a path from* $\emptyset$ *to a state* $l_1 \ \ldots \ l_n$ *in* ATLEAST_Π*, every supported model* X *for* Π *satisfies* l_i *if it satisfies all decision literals* l_j^d *with* $j \leq i$*.*

Proof. By induction on the length of the path. Similar to the proof of Lemma 1. We will show that the property in question is preserved by the four new rules.

Unit Propagate LP: M' is $M\ a$. Take any model X of Π such that X satisfies all decision literals l_j^d with $j \leq k$. From the inductive hypothesis it follows that $X \models M$. By the definition of *Unit Propagate LP*, $B \subseteq M$ for some rule $a \leftarrow B$. Consequently, $M \models B$. Since X is a model of Π we derive that $X \models a$.

All Rules Cancelled: M' is $M\ \neg a$, such that $\overline{B} \cap M \neq \emptyset$ for every $B \in \mathit{Bodies}(\Pi, a)$. Consequently, $M \models \neg B$ for every $B \in \mathit{Bodies}(\Pi, a)$. Take any model X of Π such that X satisfies all decision literals l_j^d with $j \leq k$. We need to show that $X \models \neg a$. By contradiction. Assume that $X \models a$. By the inductive hypothesis, $X \models M$. Therefore, $X \models \neg B$ for every $B \in \mathit{Bodies}(\Pi, a)$. We derive that X is not a supported model of Π.

Backchain True: M' is $M\ l$. Take any supported model X of Π such that X satisfies all decision literals l_j^d with $j \leq k$. We need to show that $X \models l$. By contradiction. Assume $X \models \overline{l}$. Consider the rule $a \leftarrow B$ corresponding to this application of *Backchain True*. Since $l \in B$, $X \models \neg B$. By the definition of *Backchain True*, $\overline{B'} \cap M \neq \emptyset$ for every B' in $\mathit{Bodies}(\Pi, a) \setminus B$. Consequently, $M \models \neg B'$ for every B' in $\mathit{Bodies}(\Pi, a) \setminus B$. By the inductive hypothesis, $X \models M$. It follows that $X \models \neg B'$ for every B' in $\mathit{Bodies}(\Pi, a) \setminus B$. Hence X is not supported by Π.

Backchain False: M' is $M\ \overline{l}$. Take any model X of Π such that X satisfies all decision literals l_j^d with $j \leq k$. We need to show that $X \models \overline{l}$. By contradiction. Assume that $X \models l$. By the definition of *Backchain False* there exists a rule $a \leftarrow l, B$ in Π such that $\neg a \in M$ and $B \subseteq M$. Consequently, $M \models \neg a$ and $M \models B$. By the inductive hypothesis, $X \models M$. It follows that $X \models \neg a$ and $X \models B$. Since $X \models l$, X does not satisfy the rule $a \leftarrow l, B$, so that it is not a model of Π.

Proposition 2. *For any program* Π*,*

(a) graph ATLEAST_Π *is finite and acyclic,*

(b) any terminal state of ATLEAST_Π *other than FailState is a supported model of* Π*,*

(c) *FailState is reachable from* $\emptyset$ *in* ATLEAST_Π *if and only if* Π *has no supported models.*

Proof. Parts (a) and (c) are proved as in the proof of Proposition 1, using Lemma 2.
(b) Let M be a terminal state. It follows that none of the rules are applicable. From the fact that *Decide* is not applicable, we derive that M assigns all literals. Since neither *Backtrack* nor *Fail* is applicable, M is consistent. Since *Unit Propagate LP* is not applicable, it follows that for every rule $a \leftarrow B \in \Pi$, if $B \subseteq M$ then $a \in M$. Consequently, if $M \models B$ then $M \models a$. We derive that M is a model of Π. We now show that M is a supported model of Π. By contradiction. Suppose that M is not a supported model. Then, there is an atom $a \in M$ such that $M \not\models B$ for every $B \in \mathit{Bodies}(\Pi, a)$. Since M is consistent, $\overline{B} \cap M \neq \emptyset$ for every $B \in \mathit{Bodies}(\Pi, a)$. Consequently, *All Rules Cancelled* is applicable. This contradicts the assumption that M is terminal.

We say that a model X of a program Π is *unfounded-free* if no non-empty subset of X is an unfounded set on X w.r.t. Π.

Lemma 3 (Theorem 4.6 [14])**.** *For any model* X *of a program* Π, X^+ *is an answer set for* Π *if and only if* X *is unfounded-free.*

Lemma 4. *For any unfounded set* U *on a consistent set* Y *of literals w.r.t. a program* Π*, and any assignment* X*, if* $X \models Y$ *and* $X \cap U \neq \emptyset$*, then* X^+ *is not an answer set for* Π*.*

Proof. Assume that X^+ is an answer set for Π. Then X is a model of Π. By Lemma 3, it follows that X^+ is unfounded-free. Since $X \cap U \neq \emptyset$ it follows that $X \cap U$ is not unfounded on X. This means that for some rule $a \leftarrow B$ in Π such that $a \in X \cap U$, $X \not\models \neg B$ and $X \cap U \cap B^+ = \emptyset$. Since $X \models Y$, it follows that $Y \not\models \neg B$. Since X satisfies B, $B^+ \subseteq X$ and consequently $U \cap B^+ = X \cap U \cap B^+ = \emptyset$. It follows that set U is not an unfounded set on Y.

Lemma 5. *For any program* Π *and a path from* $\emptyset$ *to a state* $l_1 \ \ldots \ l_n$ *in* SM_Π*, and any assignment* X*, if* X^+ *is an answer set for* Π *then* X *satisfies* l_i *if it satisfies all decision literals* l_j^d *with* $j \leq i$*.*

Proof. By induction on the length of a path. Recall that for any assignment X, if X^+ is an answer set for Π, then X is a supported model of Π, and that the transition system SM_Π extends ATLEAST_Π by the transition rule *Unfounded*. Given our proof of Lemma 2, we only need to demonstrate that application of *Unfounded* preserves the property.

Consider a transition $M \Longrightarrow^{\mathit{Unfounded}} M'$, where M is a sequence $l_1 \ \ldots \ l_k$. M' is $M \ \neg a$, such that $a \in U$, where U is an unfounded set on M w.r.t Π. Take any assignment X such that X^+ is an answer set for Π and X satisfies all decision literals l_j^d with $j \leq k$. By the inductive hypothesis, $X \models M$. Then $X \models \neg a$. Indeed, otherwise a would be a common element of X and U, and $X \cap U$ would be non-empty, which contradicts Lemma 4 with M as Y.

Since the graph SUP_Π is a subgraph of SM_Π, Lemma 5 immediately holds for SUP_Π.

Proposition 3. *For any program* Π*,*

(a) *graph* SM_Π *[*SUP_Π*] is finite and acyclic.*

(b) *for any terminal state* M *of* SM_Π *[*SUP_Π*] other than FailState,* M^+ *is an answer set of* Π.

(c) *FailState is reachable from* $\emptyset$ *in* SM_Π *[*SUP_Π*] if and only if* Π *has no answer sets.*

Proof. Parts (a) and (c) are proved as in the proof of Proposition 1, using Lemma 5.
(b) As in the proof of Proposition 2(b) we derive that M is a model of Π. Assume that M^+ is not an answer set. Then, by Lemma 3, there is a non-empty unfounded set U on M w.r.t. Π such that $U \subseteq M$. It follows that *Unfounded* [*Unfounded SUP*] is applicable (with an arbitrary $a \in U$). This contradicts the assumption that M is terminal.

Lemma 6. *For any program* Π*, the graphs* $\mathrm{ATLEAST}_\Pi$ *and* $\mathrm{DP}_{\mathit{CNF\text{-}Comp}(\Pi)}$ *are equal.*

Proof. It is easy to see that the states of the graphs $\mathrm{ATLEAST}_\Pi$ and $\mathrm{DP}_{\mathit{CNF\text{-}Comp}(\Pi)}$ coincide. We will now show that the edges of $\mathrm{ATLEAST}_\Pi$ and $\mathrm{DP}_{\mathit{CNF\text{-}Comp}(\Pi)}$ coincide also.

It is clear that there is an edge $M \implies M'$ in $\mathrm{ATLEAST}_\Pi$ justified by the rule *Decide* if and only if there is an edge $M \implies M'$ in $\mathrm{DP}_{\mathit{CNF\text{-}Comp}(\Pi)}$ justified by *Decide*. The same holds for the transition rules *Fail* and *Backtrack*.

We will now show that if there is an edge from a state M to a state M' in the graph $\mathrm{DP}_{\mathit{CNF\text{-}Comp}(\Pi)}$ justified by the transition rule *Unit Propagate* then there is an edge from M to M' in $\mathrm{ATLEAST}_\Pi$. Consider a clause $C \vee l \in \mathit{CNF\text{-}Comp}(\Pi)$ such that $M \models \neg C$. We will consider two cases, depending on whether $C \vee l$ comes from (8) or from the CNF of (7).

Case 1: $C \vee l$ is $a \vee \overline{B}$ corresponding to a rule $a \leftarrow B$.

Case 1.1: l is a. Then there is an edge from M to M' in $\mathrm{ATLEAST}_\Pi$ justified by the transition rule *Unit Propagate LP*.

Case 1.2: l is an element of $\overline{B}$. Then B has the form $\overline{l}, D$ and C is $a \vee \overline{D}$. From $\overline{C} \subseteq M$, we derive that $D \subseteq M$ and $\neg a \in M$. There is an edge from M to M' in the graph $\mathrm{ATLEAST}_\Pi$ justified by the following instance of *Backchain False*

$$M \implies M\, l \text{ if } \begin{cases} a \leftarrow \overline{l}, D \in \Pi, \\ \neg a \in M, \text{ and} \\ D \subseteq M \end{cases}$$

Case 2: $C \vee l$ has the form $\neg a \vee D$, where D is one of the clauses of the CNF of

$$\bigvee_{B \in \mathit{Bodies}(\Pi, a)} B.$$

Then D has the form

$$\bigvee_{B \in \mathit{Bodies}(\Pi, a)} f(B)$$

where f is a function that maps every $B \in \mathit{Bodies}(\Pi, a)$ to an element of B.

Case 2.1: l is $\neg a$. Then C is D, so that $\overline{D} \subseteq M$. Consequently, $\overline{B} \cap M \neq \emptyset$ for every $B \in \mathit{Bodies}(\Pi, a)$. There is an edge from M to M' in $\mathrm{ATLEAST}_\Pi$ justified by *All Rules Cancelled*.

Case 2.2: l is an element of D. From the construction of D, it follows that $l = f(B) \in B$ for some rule $a \leftarrow B$. Then C is

$$\neg a \vee \bigvee_{B' \in Bodies(\Pi,a) \setminus B} f(B').$$

From $\overline{C} \subseteq M$ we derive that $a \in M$ and that $\overline{f}(B') \in M$ for every $B' \in Bodies(\Pi,a) \setminus B$. Since $f(B')$ is a conjunctive term of B', it follows that $\overline{B'} \cap M \neq \emptyset$. Then there is an edge from M to M' in ATLEAST_Π justified by *Backchain True*.

We will now show that if there is an edge from a state M to a state M' in the graph ATLEAST_Π justified by one of the transition rules *Unit Propagate LP*, *All Rules Cancelled*, *Backchain True*, and *Backchain False* then there is an edge from M to M' in $\text{DP}_{CNF\text{-}Comp(\Pi)}$.

Case 1: The edge is justified by *Unit Propagate LP*. Then there is a rule $a \leftarrow B \in \Pi$ where $B \subseteq M$, and M' is $M\ a$. By the construction of $CNF\text{-}Comp(\Pi)$, $a \vee \overline{B} \in CNF\text{-}Comp(\Pi)$. There is an edge from M to M' in $\text{DP}_{CNF\text{-}Comp(\Pi)}$ justified by the following instance of *Unit Propagate*:

$$M \implies M\ a \text{ if } \begin{cases} \overline{B} \vee a \in CNF\text{-}Comp(\Pi) \text{ and} \\ B \subseteq M. \end{cases}$$

Case 2: The edge is justified by *All Rules Cancelled*. By the definition of *All Rules Cancelled*, there is an atom a such that for all $B \in Bodies(\Pi,a)$, $\overline{B} \cap M \neq \emptyset$; and M' is $M\ \neg a$. Consequently, M contains the complement of some literal in B. Denote that literal by $f(B)$, so that $\overline{f(B)} \in M$. From the construction of $CNF\text{-}Comp(\Pi)$,

$$\neg a \vee \bigvee_{B \in Bodies(\Pi,a)} f(B)$$

belongs to $CNF\text{-}Comp(\Pi)$. By the choice of f,

$$\overline{\bigvee_{B \in Bodies(\Pi,a)} f(B)} \subseteq M.$$

There is an edge from M to M' in $\text{DP}_{CNF\text{-}Comp(\Pi)}$ justified by the following instance of *Unit Propagate*:

$$M \implies M\ \neg a \text{ if } \begin{cases} \bigvee_{B \in Bodies(\Pi,a)} f(B) \vee \neg a \in CNF\text{-}Comp(\Pi), \\ \overline{\bigvee_{B \in Bodies(\Pi,a)} f(B)} \subseteq M. \end{cases}$$

Case 3: The edge is justified by *Backchain True*. By the definition of *Backchain True*, there is a rule $a \leftarrow B \in \Pi$ and a literal l such that $a \in M$; for all $B' \in Bodies(\Pi,a) \setminus B$, $\overline{B'} \cap M \neq \emptyset$; $l \in B$; and M' is $M\ l$. Let $f(B')$ be an element of B' such that $\overline{f(B')} \in M$. From the construction of $CNF\text{-}Comp(\Pi)$,

$$\neg a \vee l \vee \bigvee_{B' \in Bodies(\Pi,a) \setminus B} f(B')$$

belongs to *CNF-Comp*(Π). By the choice of f,

$$\overline{\bigvee_{B' \in Bodies(\Pi,a)\setminus B} f(B')} \subseteq M.$$

There is an edge from M to M' in $\mathrm{DP}_{\mathit{CNF\text{-}Comp}(\Pi)}$ justified by the following instance of *Unit Propagate*:

$$M \implies M\, l \text{ if } \begin{cases} \neg a \vee l \vee \bigvee_{B' \in Bodies(\Pi,a)\setminus B} f(B') \in \mathit{CNF\text{-}Comp}(\Pi), \\ \overline{(\neg a \vee \bigvee_{B' \in Bodies(\Pi,a)\setminus B} f(B'))} \subseteq M. \end{cases}$$

Case 4: The edge is justified by *Backchain False*. By the definition of *Backchain False*, there is a rule $a \leftarrow l, B \in \Pi$ such that $\neg a \in M$, $B \subseteq M$, and M' is $M\,\overline{l}$. By the construction of *CNF-Comp*(Π), $a \vee \overline{B} \vee \overline{l} \in \mathit{CNF\text{-}Comp}(\Pi)$. There is an edge from M to M' in $\mathrm{DP}_{\mathit{CNF\text{-}Comp}(\Pi)}$ justified by the following instance of *Unit Propagate*:

$$M \implies M\,\overline{l} \text{ if } \begin{cases} a \vee \overline{B} \vee \overline{l} \in \mathit{CNF\text{-}Comp}(\Pi) \text{ and} \\ \overline{a \vee \overline{B}} \subseteq M. \end{cases}$$

Lemma 7. *For any tight program Π and any non-empty unfounded set U on Π w.r.t. a consistent set X of literals there is an atom a such that $a \in U$ and for every $B \in Bodies(\Pi, a)$, $\overline{B} \cap X \neq \emptyset$.*

Proof. By contradiction. Assume that, for every $a \in U$ there exists $B \in Bodies(\Pi, a)$ such that $\overline{B} \cap X = \emptyset$. Consequently, $X \not\models \neg B$. By the definition of an unfounded set it follows that for every atom $a \in U$ there is $B \in Bodies(B, a)$ such that $U \cap B^+ \neq \emptyset$. Consequently the subgraph of the positive dependency graph of Π induced by U has no terminal nodes. Then, the program Π is not tight.

Proposition 4. *For any tight program Π, the graph SM^-_Π is equal to each of the graphs $\mathrm{ATLEAST}_\Pi$ and $\mathrm{DP}_{\mathit{CNF\text{-}Comp}(\Pi)}$.*

Proof. In view of Lemma 6, it is sufficient to prove that SM^-_Π equals $\mathrm{ATLEAST}_\Pi$; or, in other words, that every edge of SM_Π justified by the rule *Unfounded* only is singular. Consider such an edge $M \implies M'$. We need to show that some transition rule other than *Unfounded* is applicable to M. By the definition of *Unfounded*, M is consistent and there exists a non-empty set U unfounded on M w.r.t. Π. By Lemma 7, it follows that there is an atom $a \in U$ such that for every $B \in Bodies(\Pi, a)$, $\overline{B} \cap M \neq \emptyset$. Therefore, the transition rule *All Rules Cancelled* is applicable to M.

Lemma 8. *For any CNF formula F and a set X of models of F, and a path from $\emptyset$ to a state $l_1 \ldots l_n$ in $\mathrm{GT}_{F,X}$, any model $Y \in X$ satisfies l_i if it satisfies all decision literals l^d_j with $j \leq i$.*

Proof. Similar to the proof of Lemma 1. There are two more rules to consider:

Fail GT: Obvious.

Backtrack GT: M has the form $P\ l_i^d\ Q$ where Q contains no decision literals, $M \notin X$. Then, M' is $P\ \overline{l}_i$. Take any model E of F in X such that E satisfies all decision literals l_j^d with $j \leq i$. We need to show that $E \models \overline{l}_i$. By contradiction. Assume $E \models l_i$. By the inductive hypothesis, and the fact that M' is $P\ l_i^d\ Q$ where Q contains no decision literals, it follows that $E \models M$. Since M has no unassigned literals, $E = M$. This contradicts the assumption that $M \notin X$.

Proposition 5. *For any CNF formula F and a set X of models of F,*

(a) graph $\mathrm{GT}_{F,X}$ is finite and acyclic,

(b) any terminal state of $\mathrm{GT}_{F,X}$ other than FailState belongs to X,

(c) FailState is reachable from $\emptyset$ in $\mathrm{GT}_{F,X}$ if and only if X is empty.

Proof. Part (a) and part (c) right-to-left are proved as in the proof of Proposition 1.
(b) Let M be any terminal state other than *FailState*. As in the proof of Proposition 1(b) it follows that M is a model of F. Neither *Fail GT* nor *Backtrack GT* is applicable. Then, M belongs to X.
(c) Left-to-right: Since *FailState* is reachable from $\emptyset$, there is a state M without decision literals such that it is reachable from $\emptyset$ and either transition rule *Fail* or *Fail GT* is applicable.

Case 1. *Fail* is applicable. Then, M is inconsistent. By Lemma 8, any model of F in X satisfies M. Since M is inconsistent we conclude that X is empty.

Case 2. *Fail GT* is applicable. Then, M assigns all literals and $M \notin X$. From Lemma 8, it follows that for any $Y \in X$, $Y = M$. Since $M \notin X$, we conclude that X is empty.

11 Conclusions

In this paper we showed how to model algorithms for computing answer sets of a program by means of simple mathematical objects, graphs. This approach simplifies the analysis of the correctness of algorithms and allows us to study the relationship between various algorithms using the structure of the corresponding graphs. For example, we used this method to establish that applying the SMODELS algorithm to a tight program essentially amounts to applying DPLL to its completion. It also suggests new designs for answer set solvers, as can be seen from our work on SUP. In the future we will investigate the generalization of this framework to backjumping and learning performed by the SMODELS$_{cc}$ algorithm [6], to SUP with Learning, and to ASP-SAT with Learning [4]. We also would like to generalize this approach to the algorithms used in disjunctive answer set solvers.

Acknowledgements

We are grateful to Marco Maratea for bringing to our attention the work by Nieuwenhuis et al. (2006), to Vladimir Lifschitz for the numerous discussions, to Martin Gebser and Michael Gelfond for valuable comments. The author was supported by the National Science Foundation under Grant IIS-0712113.

References

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

[2] Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). Journal of the ACM **53(6)** (2006) 937–977

[3] Simons, P.: Extending and Implementing the Stable Model Semantics. PhD thesis, Helsinki University of Technology (2000)

[4] Giunchiglia, E., Lierler, Y., Maratea, M.: Answer set programming based on propositional satisfiability. Journal of Automated Reasoning **36** (2006) 345–377

[5] Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R., Bowen, K., eds.: Proceedings of International Logic Programming Conference and Symposium, MIT Press (1988) 1070–1080

[6] Ward, J.: Answer Set Programming with Clause Learning. PhD thesis, The University of Cincinnati (2004)

[7] Van Gelder, A., Ross, K., Schlipf, J.: The well-founded semantics for general logic programs. Journal of ACM **38**(3) (1991) 620–650

[8] Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by SAT solvers.[9] Artificial Intelligence **157** (2004) 115–137

[9] Een, N., Sörensson, N.: An extensible sat-solver. In: SAT. (2003)

[10] Fages, F.: Consistency of Clark's completion and existence of stable models. Journal of Methods of Logic in Computer Science **1** (1994) 51–60

[11] Clark, K.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and Data Bases. Plenum Press, New York (1978) 293–322

[12] Giunchiglia, E., Maratea, M.: On the relation between answer set and SAT procedures (or, between smodels and cmodels). In: 21st International Conference on Logic Programming (ICLP'05). (2005) 37–51

[13] Gebser, M., Schaub, T.: Tableau calculi for answer set programming. In: 22d International Conference on Logic Programming (ICLP'06). (2006) 11–25

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

[9] Revised version: `http://www.cs.ust.hk/faculty/flin/papers/assat-aij-revised.pdf` .