Abstract Disjunctive Answer Set Solvers

Remi Brochenin\(^1\) and Yuliya Lierler\(^2\) and Marco Maratea\(^3\)

Abstract. A fundamental task in answer set programming is to compute answer sets of logic programs. Answer set solvers are the programs that perform this task. The problem of deciding whether a disjunctive program has an answer set is \(\Sigma^P_2\)-complete. The high complexity of reasoning within disjunctive logic programming is responsible for few solvers capable of dealing with such programs, namely DLV, GNT, CMODELS and CLASP. We show that transition systems introduced by Nieuwenhuis, Oliveras, and Tinelli can model and analyze satisfiability solvers can be adapted for disjunctive answer set solvers. In particular, we present transition systems for CMODELS (without backjumping and learning), GNT and DLV (without backjumping). The unifying perspective of transition systems on satisfiability and non-disjunctive answer set solvers proved to be an effective tool for analyzing, comparing, proving correctness of each underlying search algorithm as well as bootstrapping new algorithms. Given this, we believe that this work will bring clarity and inspire new ideas in design of more disjunctive answer set solvers.

1 Introduction

Answer set programming (ASP) is a declarative programming paradigm oriented towards difficult combinatorial search problems \([20, 21]\). ASP has been applied to many areas of science and technology, from the design of a decision support system for the Space Shuttle \([24]\) to graph-theoretic problems arising in zoology and linguistics \([1]\). A fundamental task in ASP is to compute answer sets of logic programs. Answer set solvers are the programs that perform this task. There were sixteen answer set solvers participating in the Fourth Answer Set Programming Competition in 2013\(^4\).

Gelfond and Lifschitz introduced logic programs with disjunctive rules \([8]\). The problem of deciding whether a disjunctive program has an answer set is \(\Sigma^P_2\)-complete \([3]\). The high complexity of reasoning within disjunctive logic programming stems from two sources: (i) there is an exponential number of possible candidate models, and (ii) the hardness of checking whether a candidate model is an answer set of a propositional disjunctive logic program is NP-complete. Only four answer set systems allow programs with disjunctive rules: DLV \([13]\), GNT \([10]\), CMODELS \([14]\) and CLASP \([6]\).

Recently, several formal approaches have been used to describe and compare search procedures implemented in answer set solvers. These approaches range from a pseudo-code representation of the procedures \([9]\), to tableau calculi \([7]\), to abstract frameworks via transition systems \([17, 18]\). The last method proved to be particularly suited for the goal. It originates from the work by Nieuwenhuis et al. \([23]\), where authors proposed to use transition systems to describe the DPLL (Davis-Putnam-Logemann-Loveland) procedure \([2]\). They introduced an abstract framework – a DPLL graph – that captures what “states of computation” are, and what transitions between states are allowed. Every execution of the DPLL procedure corresponds to a path in the DPLL graph. Lierler and Truszczynski \([17, 18]\) adapted this approach to describing answer set solvers for non-disjunctive programs including CMODELS, CMODELS, and CLASP. Such an abstract way of presenting algorithms simplifies the analysis of their correctness and facilitates formal reasoning about their properties, by relating algorithms in precise mathematical terms.

In this paper we present transition systems that account for disjunctive answer set solvers implementing plain backtracking. We define abstract frameworks for CMODELS (without backjumping and learning), GNT and DLV (without backjumping). We also identify a close relationship between answer set solvers DLV and CMODELS by means of properties of the related graphs. We believe that this work will bring better understanding of the main design features of current disjunctive answer set solvers as well as inspire new algorithms.

The paper is structured as follows. Sec. 2 introduces needed preliminaries. Sec. 3, 4 and 5 show the abstract frameworks of CMODELS, GNT and DLV, respectively. The paper ends in Sec. 6 by discussing related works and with final remarks.

2 Preliminaries

Formulas, Logic Programs, and Program’s Completion

Atoms are Boolean variables over \(\{\text{true}, \text{false}\}\). The symbols \(\bot\) and \(\top\) are the false and the true constant, respectively. The letter \(l\) denotes a literal, that is an atom \(a\) or its negation \(\neg a\), and \(\top\) is the complement of \(l\), i.e., literal \(a\) for \(\neg a\) and literal \(\neg a\) for \(a\). Propositional formulas are logical expressions defined over atoms and symbols \(\bot, \top\) that take value in the set \(\{\text{true}, \text{false}\}\). A finite disjunction of literals, is a clause. We identify an empty clause with the clause \(\bot\). A CNF formula is a conjunction (alternatively, a set) of clauses. A conjunction (disjunction) of literals will sometimes be seen as a set, containing each of its literals. Given a conjunction (disjunction) \(B\) of literals, by \(\overline{B}\) we denote the disjunction (conjunction) of the complements of the elements of \(B\). For example, \(\overline{a \lor \neg b}\) denotes \(\neg a \land \neg b\), while \(a \land \neg b\) denotes \(\neg a \lor b\). A (truth) assignment to a set \(X\) of atoms is a function from \(X\) to \(\{\text{false}, \text{true}\}\). A satisfying assignment or a model for a formula \(F\) is an assignment \(M\) such that \(F\) evaluates to true under \(M\). If \(F\) evaluates to false under \(M\), we say that \(M\) contradicts \(F\). If \(F\) has no model we say that \(F\) is unsatisfiable. We often identify a consistent set \(L\) of literals (i.e., a set that does not contain complementary literals, for example, \(a\) and \(\neg a\)) with an assignment as follows: if \(a \in L\) then \(a\) maps to true, while if \(\neg a \in L\) then \(a\) maps to false. We also identify a set \(X\) of atoms over \(At(\Pi)\) with an assignment as follows: if \(a \in X\) then \(a\) maps to true, while if \(a \in At(\Pi) \setminus X\) then \(a\) maps to false.

---

1 University of Genova, Italy, email: remi.brochenin@unige.it
2 University of Nebraska at Omaha, email: ylierler@unomaha.edu
3 University of Genova, Italy, email: marco@dbiris.unige.it
4 https://www.mat.unical.it/aspcomp2013/Participants
A (propositional) disjunctive logic program is a finite set of disjunctive rules of the form
\[
a_1 \lor \ldots \lor a_i \leftarrow a_{i+1}, \ldots, a_j, \text{not } a_{j+1}, \ldots, \text{not } a_k, \\
\quad \text{not not } a_{k+1}, \ldots, \text{not not } a_n,
\]
where \(a_1, \ldots, a_n\) are atoms. The left hand side expression of a rule is called the head. We call rule (1) non-disjunctive if its head contains not more than one atom. A program is non-disjunctive if it consists of non-disjunctive rules. The letter \(B\) often denotes the body
\[
a_{i+1}, \ldots, a_j, \text{not } a_{j+1}, \ldots, \text{not } a_k, \text{not not } a_{k+1}, \ldots, \text{not not } a_n
\]
of a rule (1). We often identify (2) with the conjunction
\[
a_{i+1} \land \ldots \land a_j \land \lnot a_{j+1} \land \ldots \land \lnot a_k \land \lnot a_{k+1} \land \ldots \land \lnot a_n.
\]
We identify the rule (1) with the clause
\[
a_1 \lor \ldots \lor a_i \lor \lnot a_{i+1} \lor \ldots \lor \lnot a_j \lor a_{j+1} \lor \ldots \lor a_k \lor \lnot a_{k+1} \lor \ldots \lor \lnot a_n.
\]
This allows us to sometimes view a program \(\Pi\) as a CNF formula.

It is important to note the presence of doubly negated atoms in the bodies of rules. This version of logic programs is a special case of programs with nested expressions introduced by Lifschitz et al.\cite{[19]}. A choice rule\cite{[22]} construct \((a) \leftarrow B\), originally employed in the \textsc{lparsel} and \textsc{gringo}\cite{[5]} languages, can be seen as an abbreviation for a rule \(a \leftarrow B, \text{ not not } a\). In this work we adopt this abbreviation. We sometimes write (1)\ as
\[
A \leftarrow D, F
\]
where \(A\) is  \(a_1 \lor \ldots \lor a_i, D = a_{i+1}, \ldots, a_j,\) and \(F\) is
\[
\text{not not } a_{j+1}, \ldots, \text{not not } a_k, \ldots, \text{not not } a_n,
\]
The reduct \(\Pi^X\) of a disjunctive program \(\Pi\) w.r.t. a set \(X\) of atoms is obtained from \(\Pi\) by deleting each rule (4) such that \(X \not\subseteq F\) and replacing each remaining rule (4) with \(A \leftarrow D\). A set \(X\) of atoms is an answer set of \(\Pi\) if \(X\) is minimal among the sets of atoms that satisfy \(\Pi^X\). For any consistent and complete set \(M\) of literals, if \(M^+\) is an answer set for a program \(\Pi\), then \(M\) is a model of \(\Pi\). Moreover, in this case \(M\) is a supported model of \(\Pi\), in the sense that for every atom \(a \in M\), \(M \models B\) for some rule \(a \leftarrow B\) in \(\Pi\).

The completion \(\text{Comp}(\Pi)\) of a program \(\Pi\) is a formula
\[
\text{Comp}(\Pi) = \Pi \cup \{ a \lor \bigvee_{C \vdash a \in \text{At}(\Pi)} B \land \lnot C \}\, \text{where by At(\Pi) we denote the set of atoms occurring in \(\Pi\). This formula has the property that any answer set of \(\Pi\) is a model of Comp(\Pi). The converse does not hold in general.}
\]

\textbf{Abstract DPLL.} The Davis-Putnam-Logemann-Loveland (DPLL) procedure\cite{[2]} is a well-known method that exhaustively explores assignments to generate models of a propositional formula. Most modern satisfiability and answer set solvers are based on variations of the DPLL procedure. We now review the abstract transition system for DPLL proposed by Nieuwenhuis et al.\cite{[23]}. This framework provides an alternative to common pseudo-code descriptions of backtrack search based algorithms.

For a set \(X\) of atoms, a record relative to \(X\) is a string \(L\) composed of literals over \(X\) or symbol \(\bot\) without repetitions where some literals are annotated by \(\Delta\). The annotated literals are called decision literals. We say that a record \(L\) is inconsistent if it contains both a literal \(l\) and its complement \(\lnot l\), or if it contains \(\bot\). We will sometime identify a record with the set containing all its elements disregarding its annotations. For example, we will identify a record \(\Delta \bot \lnot a\) with the set \(\{\bot, a, \lnot a\}\) of literals.

A state relative to \(X\) is either the distinguished state \text{Failstate}, a record relative to \(X\), or \text{Ok}\((L)\) where \(L\) is a record relative to \(X\).

For instance, states relative to a singleton set \(\{a\}\) include
\[
\text{Failstate, } \emptyset, \bot, a, \bot, a, \lnot a, a^\Delta, \lnot a^\Delta, a \lnot a, a^\Delta \lnot a, \lnot a^\Delta a, \lnot a a^\Delta, \text{Ok}(a).
\]

Each CNF formula \(F\) determines its \textsc{dpll} graph \(\text{DPLL}_{F}\). The set of nodes of \(\text{DPLL}_{F}\) consists of the states relative to the set of atoms occurring in \(F\). The edges of the graph \(\text{DPLL}_{F}\) are specified by the transition rules:\footnote{\url{http://www.tcs.hut.fi/Software/sm0models/}}\footnote{\url{http://potassco.sourceforge.net/}}
\[
\begin{align*}
\text{UnitPropagate}: \quad & L \Rightarrow L_i & \text{if } C \lor \bot \text{ is a clause in } F \text{ and all the literals of } C \text{ occur in } L \\
\text{Decide}: \quad & L \Rightarrow L_{\Delta a} \quad \text{if } \{ a \text{ or not } a \text{ occur in } L \} \text{ and } \text{not } l \text{ or } \lnot l \text{ occur in } L \\
\text{Conclude}: \quad & L \Rightarrow \text{Failstate} & \text{if } \{ a \text{ or not } a \text{ occur in } L \} \text{ and } L \text{ contains no decision literals} \\
\text{Backtrack}: \quad & L_{\Delta a} \Rightarrow L_i & \text{if } L_{a} \text{ is a record relative to } X \text{ and } L \text{ contains no decision literals} \\
\text{OK}: \quad & L \Rightarrow \text{Ok}(L) & \text{if no other rule applies}
\end{align*}
\]
A node (state) in the graph is terminal if no edge originates in it. The following theorem gathers key properties of the graph \(\text{DPLL}_{F}\).

\textbf{Theorem 1 (Proposition 1 in \cite{[17]}) For any CNF formula } \(F\),
\begin{enumerate}
\item graph \(\text{DPLL}_{F}\) is finite and acyclic,
\item any terminal state reachable from \(\emptyset\) in \(\text{DPLL}_{F}\) other than \text{Failstate} is \text{Ok}(L), with \(L\) being a model of \(F\),
\item \text{Failstate} is reachable from \(\emptyset\) in \(\text{DPLL}_{F}\) if and only if \(F\) is unsatisfiable.
\end{enumerate}

Thus, to decide the satisfiability of a CNF formula \(F\) it is enough to find a path leading from node \(\emptyset\) to a terminal node. If it is a \text{Failstate}, \(F\) is unsatisfiable. Otherwise, \(F\) is satisfiable. For instance, let \(F = \{a \lor b, \lnot a \lor c\}\). Below we show a path in \(\text{DPLL}_{F}\) with every edge annotated by the name of the transition rule that gives rise to this edge in the graph (\text{UP} abbreviates \text{UnitPropagate})
\[
\emptyset \xrightarrow{\text{Decide}} a^\Delta \xrightarrow{\text{UP}} a^\Delta c \xrightarrow{\text{Decide}} a^\Delta c \lnot a \xrightarrow{\text{OK}} \text{Ok}(a^\Delta c \lnot a^\Delta).\quad (5)
\]
The state \(\text{Ok}(a^\Delta c \lnot a^\Delta)\) is terminal. Thus, Theorem 1 asserts that \(F\) is satisfiable and \(\{a, c, b\}\) is a model of \(F\). Here is another path to the same terminal state
\[
\emptyset \xrightarrow{\text{Decide}} a^\Delta \xrightarrow{\text{Decide}} a^\Delta \lnot c \xrightarrow{\text{UP}} a^\Delta \lnot c \xrightarrow{\text{Decide}} a^\Delta c \lnot a \xrightarrow{\text{OK}} \text{Ok}(a^\Delta c \lnot a^\Delta).\quad (6)
\]
A path in the graph \(\text{DPLL}_{F}\) is a description of a process of search for a model of a CNF formula \(F\). The process is captured via applications of transition rules. Therefore, we can characterize the algorithm
of a solver that utilizes the transition rules of \( DP_F \) by describing a strategy for choosing a path in this graph. A strategy can be based on assigning priorities to transition rules of \( DP_F \) so that a solver never applies a rule in a state if a rule with higher priority is applicable to the same state. The \( DPLL \) procedure is captured by the following priorities:

\[
\text{Conclude, Backtrack} \gg \text{UnitPropagate} \gg \text{Decide.}
\]

Path (5) complies with the \( DPLL \) priorities. Thus it corresponds to an execution of \( DPLL \). Path (6) does not: it uses \( Decide \) when \( \text{UnitPropagate} \) is applicable.

**Disjunctive Answer Set Solver: Discussion**

The problem of deciding whether a disjunctive program has an answer set is \( \Sigma_2^P \)-complete [3]. This is because: (i) there is an exponential number of possible candidate models, and (ii) the hardness of checking whether a candidate model is an answer set of a disjunctive program is \( \Sigma_2^P \)-complete. The latter condition differentiates disjunctive answer set solving procedures from answer set solvers for non-disjunctive programs. Informally, a disjunctive (answer set) solver requires two “layers” of computation – two solving engines: one that generates candidate models, and another that tests candidate models. Existing disjunctive solvers differ in underlying technology for each of the solving engines. System \( CMODELS \) uses instances of SAT solvers for each of the tasks. System \( GNT \) uses instances of non-disjunctive answer set solver \( SMODELS \). System \( DLV \) uses the \( SMODELS \)-like procedure to generate candidate models, and instances of SAT solvers to test candidate models. These substantial differences obscure the thorough analysis and understanding of similarities and differences between the existing disjunctive solvers. To elevate this difficulty, we generalize the graph-based framework for capturing \( DPLL \)-like procedures to the case of disjunctive answer set solving.

### 3 Abstract CMODELS

We start by introducing a graph \( DP_{F,f} \) based on two instances of \( DPLL \) graph. We then describe how it can be used to capture the \( CMODELS \) procedure for disjunctive programs.

**Abstract Solver via DPLL.** We call a function \( f : M \rightarrow F \) from a set \( M \) of literals to a CNF formula \( F \) a witness-formula function. Intuitively, a CNF formula resulting from a witness function is a witness (formula) with respect to \( M \). Informally, a witness formula is what is tested by a solver after generating a candidate model so as to know whether this candidate is good.

An (extended) state relative to sets \( X \) and \( X' \) of atoms is a pair \((L,R)\) or distinguished states \( \text{Failstate} \) or \( \text{Ok}(L) \), where \( L \) and \( R \) are records relative to \( X \) and \( X' \), respectively. We often drop the word extended before state, when it is clear from a context. A state \((\emptyset,\emptyset)\) is called initial. For a formula \( F \), by \( \text{At}(F) \) we denote the set of atoms occurring in \( F \). For a formula \( F \) and a witness function \( f \), by \( \text{At}(F,f) \) we denote the union of \( \text{At}(f(L)) \) for all possible consistent records \( L \) over \( \text{At}(F) \). It is not necessarily equal to \( \text{At}(F) \) as \( f \) may, for instance, introduce additional variables.

We now define a graph \( DP_{F,f} \) for a CNF formula \( F \) and a witness function \( f \). The set of nodes of \( DP_{F,f} \) consists of the states relative to \( \text{At}(F) \) and \( \text{At}(F,f) \). The edges of the graph \( DP_{F,f} \) are specified by the transition rules presented in Figure 1. We use the following abbreviations in stating these rules. Expression \( \text{up}(L,l,F) \) holds when the condition of the transition rule \( \text{UnitPropagate} \) of the graph \( DP_F \) holds, i.e., when

\[
C \lor l \text{ is a clause in } F \text{ and all the literals of } C \text{ occur in } L
\]

Similarly, \( \text{de}(L,l,F), \text{fa}(L) \), and \( \text{ba}(L,l,L') \) hold when the conditions of \( \text{Decide}, \text{Conclude} \), and \( \text{Backtrack} \) of \( DP_F \) hold, respectively.

A graph \( DP_{F,f} \) can be used for deciding whether a CNF formula \( F \) has a model \( M \) such that witness formula defined by \( f \) with respect to \( M \) is unsatisfiable.

**Theorem 2**

For any CNF formula \( F \) and a witness function \( f \):

1. graph \( DP_{F,f} \) is finite and acyclic.
2. any terminal state of \( DP_{F,f} \) reachable from the initial state and other than \( \text{Failstate} \) is \( \text{Ok}(L) \), with \( L \) being a model of \( F \) such that \( f(L) \) is unsatisfiable.
3. \( \text{Failstate} \) is reachable from the initial state if and only if \( F \) has no model such that its witness is unsatisfiable.

This graph can be used to capture two layers of computation – generate and test – by combining two \( DPLL \) procedures as follows. The generate layer applies the \( DPLL \) procedure to a given formula \( F \) (see left-rules). It turns out that left-rules no longer apply to a state \((L,R)\) only when \( L \) is a model for \( F \). Thus, when a model \( L \) for \( F \) is found, then a witness formula with respect to \( L \) is built. The test layer applies the \( DPLL \) procedure to the witness formula (see right-rules). If no model is found for the witness formula, then \( \text{Conclude} \) rule applies bringing us to a terminal state \( \text{Ok}(L) \) suggesting that \( L \) represents a solution to a given search problem. It turns out that no left-rules and no right-rules apply in a state \((L,R)\) only when \( R \) is a model for the witness formula. Thus, the set \( L \) of literals is not a solution and the \( DPLL \) procedure of the generate layer proceeds with the search (see crossing-rules).

**CMODELS via the Abstract Solver.** We now relate the graph \( DP_{F,f} \) to the \( CMODELS \) procedure, \( DP\)-ASSAT-PROC, described by Lierler [14]. We start by introducing some required notation.

For a set \( M \) of literals, by \( M^+ \) we denote atoms that occur positively in \( M \). For example, \( \{a,b\} = \{b\} \). For set \( \sigma \) of atoms and set \( M \) of literals, by \( M|_\sigma \) we denote the maximal subset of \( M \) over \( \sigma \). For example, \( \{a,\neg b, c\}|_{\{a,\neg b\}} = \{a, \neg b\} \). We say that a set \( M \) of
literals covers a set $\sigma$ of atoms if for each atom $a$ in $\sigma$ either $a$ or $\neg a$ is in $M$. For example, set $\{a\}$ of literals covers set $\{a, b\}$ of atoms while $\{\neg a\}$ does not cover $\{a, b\}$. Given a program $\Pi$ and a consistent set $M$ of literals that covers $\Pi$, a witness function $f_{\text{min}}$ maps $M$ into a formula composed of the clause $M^+$, one clause $\neg a$ for each literal $\neg a \in M$, and the clauses of $\Pi^M +$. Recall that we identify a program with a CNF formula.

Given a disjunctive program $\Pi$, the answer set solver CMODELS starts its computation by converting program’s completion $\text{Comp}(\Pi)$ into a CNF formula that we call $E\text{Dcomp}(\Pi)$. Lierler (Section 13.2, [16]) describes the details of the transformation. The graph $DP^2_{E\text{Dcomp}(\Pi), f_{\text{min}}}$ captures the search procedure of DP-AssAT-PROC of CMODELS. The DP-AssAT-PROC algorithm follows the priorities on its transition rules listed below:

- $\text{Backtrack}_{\Pi}, \text{Conclude}_{\Pi} >> \text{UnitPropagate}_{\Pi} >> \text{Decide}_{\Pi}$
- $\text{Backtrack}_{\Pi}, \text{Conclude}_{\Pi} >> \text{UnitPropagate} >> \text{Decide}_{\Pi}$
- $\text{Backtrack}_{\Pi}, \text{Conclude}_{\Pi}$

A proof of correctness and termination of the DP-AssAT-PROC procedure results from Theorem 2 and two conditions on formula $E\text{Dcomp}(\Pi)$ and function $f_{\text{min}}$: (i) for any answer set $X$ of $\Pi$ there is a model $M$ of $E\text{Dcomp}(\Pi)$ such that $X = M^\text{At}(\Pi)$; and (ii) for any consistent set $M$ of literals covering $\Pi$, $M^\text{At}(\Pi)$ is an answer set of $\Pi$ if and only if $f_{\text{min}}(M)$ results in an unsatisfiable formula.

We now capture the graph $DP^2_{E\text{Dcomp}(\Pi), f_{\text{min}}}$, general properties which guarantee that a similar solving strategy that uses the DPLL procedure for generate and test layers results in a correct answer set solver. We say that a propositional formula $F$ $DP$-approximates a program $\Pi$ if for any answer set $X$ of $\Pi$ there is a model $M$ of $F$ such that $X = M^\text{At}(\Pi)$. For instance, completion of $\Pi$ $DP$-approximates $\Pi$. We say that a witness-formula function $f$ $DP$-ensures a program $\Pi$ if for any consistent set $M$ of literals that covers $\Pi$, $M^\text{At}(\Pi)$ is an answer set of $\Pi$ if and only if $f(M)$ results in an unsatisfiable formula. For example, the witness-formula function $f_{\text{min}}$ $DP$-ensures $\Pi$. It turns out that for any program $\Pi$, given any formula $F$ that $DP$-approximates $\Pi$ and any witness function $f$ that $DP$-ensures $\Pi$, the graph $DP^2_{\Pi, f}$ captures a correct algorithm for establishing whether $\Pi$ has answer sets.

**Theorem 3** For a disjunctive program $\Pi$, a CNF formula $F$ that $DP$-approximates $\Pi$, and a witness-formula function $f$ that $DP$-ensures $\Pi$,

1. $\text{graph } DP^2_{\Pi, f}$ is finite and acyclic,
2. any terminal state of $DP^2_{\Pi, f}$ reachable from the initial state and other than $\text{Failstate}$ is $\text{Ok}(L)$, with $L^\text{At}(\Pi)$ being an answer set of $\Pi$.
3. $\text{Failstate}$ is reachable from the initial state if and only if $\Pi$ has no answer sets.

**4 Abstract GNT**

We illustrated how the graph $DP^2_{\Pi, f}$ captures the basic CMODELS procedure. This section describes a respective graph for the procedure underlying disjunctive solver GNT. Recall that unlike solver CMODELS that uses the DPLL procedure for generating and testing, system GNT uses the SMODELS procedure – an algorithm for finding answer sets of non-disjunctive logic programs – for respective tasks. Lierler [17] introduced the graph $G_{\Pi}$, that captures the computation underlying the SMODELS algorithm just as the graph $DP^2_{\Pi}$ captures the computation underlying DPLL. The graph $G_{\Pi}$ forms a basis for devising the transition system suitable to describe GNT.

$$ac(L, a, A) \quad \text{if} \quad \begin{cases} \text{for each rule } a \leftarrow B \text{ of } A \\ B \text{ is contradicted by } L \end{cases}$$

$$bt(L, l, A) \quad \text{if} \quad \begin{cases} \text{there is a rule } a \leftarrow l, B \text{ of } A \text{ such that } \\ a \text{ is a literal of } L \text{ and } \\ B^* \text{ is contradicted by } L \end{cases}$$

$$uf(L, a, A) \quad \text{if} \quad \begin{cases} \text{there is a set } L \text{ containing } a \text{ such that } \\ M \text{ is unfounded on } L \text{ w.r.t. } A \end{cases}$$

**Figure 2.** The properties for rules of the graph $SM^2_{\Lambda, p}$.

**Left-rules:**
- $\text{AllRulesCancelled}_{\Pi} (L, R) \quad \Rightarrow \quad (L, R)\_a \quad \text{if } ac(L, a, A)$
- $\text{BackchainTrue}_{\Pi} (L, R) \quad \Rightarrow \quad (L, R) \quad \text{if } bt(L, l, A)$
- $\text{Unfounded}_{\Pi} (L, R) \quad \Rightarrow \quad (L, R)\_a \quad \text{if } uf(L, a, A)$

**Right-rules,** applicable when no left-rule applies:
- $\text{AllRulesCancelled}_{\Pi} (L, R) \quad \Rightarrow \quad (L, R)\_a \quad \text{if } ac(L, a, A)$
- $\text{BackchainTrue}_{\Pi} (L, R) \quad \Rightarrow \quad (L, R) \quad \text{if } bt(L, l, A)$
- $\text{Unfounded}_{\Pi} (L, R) \quad \Rightarrow \quad (L, R)\_a \quad \text{if } uf(L, a, A)$

**Figure 3.** Transition rules of the graph $SM^2_{\Lambda, p}$.

**Abstract Solver via SMODELS.** We abuse some terminology, by calling a function $p : M \rightarrow \Lambda$ from a set of literals to a non-disjunctive program $\Lambda$ a witness-(program) function. Intuitively, a program resulting from a witness function is a witness (program) with respect to $M$. For a program $\Lambda$ and a witness function $p$, by $At(\Lambda, p)$ we denote the union of $At(p(L))$ for all possible consistent records $L$ over $At(\Lambda)$.

We now define a graph $SM^2_{\Lambda, p}$ for a non-disjunctive program $\Lambda$ and a witness function $p$. The set of nodes of $SM^2_{\Lambda, p}$ consists of the states relative to $At(\Lambda)$ and $At(\Lambda, p)$. The edges of the graph $SM^2_{\Lambda, p}$ are specified by the transition rules of the $DP^2_{\Pi, f}$ graph extended with the transition rules presented in Figure 3 and based on the properties listed in Figure 2. We refer the reader to [12] for the definition of “unfounded” sets.

A graph $SM^2_{\Lambda, p}$ can be used for deciding whether a non-disjunctive program $\Lambda$ has an answer set $X$ such that witness program defined by $p(X)$ has no answer sets.

**Theorem 4** For any non-disjunctive program $\Lambda$ and a witness function $p$,

1. $\text{graph } SM^2_{\Lambda, p}$ is finite and acyclic,
2. any terminal state of $SM^2_{\Lambda, p}$ reachable from the initial state and other than $\text{Failstate}$ is $\text{Ok}(L)$, with $L^\text{At}(\Lambda)$ being an answer set of $\Lambda$ such that $p(L)$ has no answer set.
3. $\text{Failstate}$ is reachable from the initial state if and only if there is no set $L$ of literals such that $L^+$ is an answer set of $\Lambda$ and $p(L)$ has no answer set.

Similarly to the graph $DP^2_{\Pi, f}$, the graph $SM^2_{\Lambda, p}$ has two layers. It combines two SMODELS procedures in place of DPLL procedures.

**GNT via the Abstract Solver.** Let us illustrate how GNT is described by this graph. We need some additional notations for that. For a disjunctive program $\Pi$, by $\Pi_N$ we denote the set of non-disjunctive rules of $\Pi$, by $\Pi_D$ we denote $\Pi \setminus \Pi_N$. For each atom $a$ in $At(\Pi)$ let
$a^*$ be a new atom. For a set $X$ of atoms by $X^+$ we denote a set \{ $a^*$ $| a \in X$\} of atoms. The non-disjunctive program $Gen(II)\text{ }$defined by Janhunen et al.\textsuperscript{10} consists of the rules below
\[
\begin{align*}
\{\{a\} & \leftarrow B \mid a, A \leftarrow B \in \Pi_D\} \cup \\
\{\lnot A, B & \mid A \leftarrow B \in \Pi_D\} \cup \\
\Pi_F & \cup \\
\{a^* & \leftarrow \{a\}, B \mid A \leftarrow B \in \Pi_D; a \in A; a' \leftarrow B' \in \Pi_D\} \cup \\
\{a & \not\in a^* \mid a \leftarrow B \in \Pi_D\}
\end{align*}
\]

Janhunen et al.\textsuperscript{10} defined a witness-program that they call $Test$. The graph $SM_{Gen(II),Test}^2$ captures the GNT procedure in a similar way as $DP_{F,Gen(II),f_{min}}^2$ captures the CMODELS procedure of DP-ASSAT-PROC. The precedence order
\[
Backtrack_L, Conclude_L >> UnitPropagate_L, AllRulesCancelled_R, \\
BackchainTrue_L >> Unfounded_L >> Decide_L >> \\
Backtrack_R, Conclude_R >> UnitPropagate_R, AllRulesCancelled_R, \\
BackchainTrue_R >> Unfounded_R >> Decide_R >> Backtrack_R, Conclude_R
\]

on the rules of the graph $SM_{Gen(II),Test}^2$ describes GNT.\textsuperscript{9}

We say that a non-disjunctive program $\Lambda$ $SM$-approximates a program $\Pi$ (resp. $SM$-approximates) if for any answer set $X$ of $\Pi$ there is a consistent and complete set $M^+$ of literals such that $X = M^+_{At(\Pi)}$. The program $Gen(II)$ both SM-approximates II and $SM^+$-approximates II. We say that a witness-program function $p$ $SM$-ensures a program $\Pi$ if for any consistent set $M^+$ of literals that covers $At(\Pi)$, $M^+_{At(\Pi)}$ is an answer set of $\Pi$ if and only if $p(M)$ results in a program that has no answer sets. The function $Test$ $SM$-ensures II. We also define the graph $SM^+ \times SM^+_\Lambda$ as the graph $SM^2_{\Lambda,p}$ but the rule $Unfounded$. It turns out that for any program $\Pi$, given a witness-program function $p$ that SM-ensures II and a nondisjunctive program $\Lambda$ that SM-approximates II (resp. $SM^+$-approximates II), the graph $SM^2_{\Lambda,p}$ (resp. $SM^+ \times SM^+_\Lambda$) captures a correct algorithm for establishing whether II has answer sets.

\textbf{Theorem 5} For a disjunctive program II, a non-disjunctive program $\Lambda$, that $SM$-approximates II (resp. $SM^+$-approximates II), and a witness-program function $p$ that $SM$-ensures II,
\begin{enumerate}
\item graph $SM^2_{\Lambda,p}$ (resp. $SM^+ \times SM^+_\Lambda$) is finite and acyclic,
\item any terminal state of $SM^2_{\Lambda,p}$ (resp. $SM^+ \times SM^+_\Lambda$) reachable from the initial state and other than Failstate is Ok(L), with $L_{At(\Pi)}$ being an answer set of II,
\item Failstate is reachable from the initial state if and only if $II$ has no answer sets.
\end{enumerate}

Gelfond and Lifschitz\textsuperscript{8} defined a mapping from a disjunctive program II to a non-disjunctive program $II_{\alpha}$, the shifted variant of II, by replacing each rule (1) in II by $i$ new rules:
\[
a_m & \leftarrow B, not a_1, \ldots, not a_{m-1}, not a_{m+1}, \ldots, not a_i
\]
where $1 < m \leq i$, $B$ stands for the body (2) of the rule (1). Program $II_{\alpha}$ $SM$-approximates II. Theorem 5 ensures the graph $SM^\alpha \times SM^\alpha_{II_{\alpha},Test}$ captures a correct procedure for establishing whether a program $\Pi$ has answer sets.

\textbf{5 Abstract DLV and More}

We illustrated how procedures behind CMODELS and GNT are captured by the graphs $DP^{2}_{F,L}$ and $SM^{2}_{\Lambda,p}$ respectively. We now introduce a graph that captures answer set solver DLV.

We define a graph $SM^\prime \times DP^\prime_{L}$ for a program II and a witness-formula function $f$ that $DP$-ensures II:
\begin{enumerate}
\item graph $SM^\prime \times DP^\prime_{L}$ is finite and acyclic,
\item any terminal state of $SM^\prime \times DP^\prime_{L}$ reachable from the initial state and other than Failstate is Ok(L), with $L_{At(\Pi)}$ being an answer set of II,
\item Failstate is reachable from the initial state if and only if $II$ has no answer set.
\end{enumerate}

The graph $SM^\prime \times DP^\prime_{L}$ has two layers. The generate layer, i.e., the left-rule layer, is reminiscent to the SMODELS algorithm without $Unfounded$. The test layer applies the DPLL procedure to the witness formula. We refer the reader to [11] for the details of the specific witness function $f$ employed in DLV.

It differs from $f_{min}$ used in CMODELS. The graph $SM^\prime \times DP^\prime_{L}$, along with the precedence order (7) trivially extended to the rules of $SM^\prime \times DP^\prime_{L}$ describes DLV, as in [4] and [11].

It turns out that systems DLV and CMODELS share a lot in common: the transition systems that capture DLV and CMODELS fully coincide in their left-rules.

\textbf{Theorem 7} For a disjunctive program II, the edge-induced subgraph of $SM^\prime \times DP^\prime_{L}$ w.r.t. left-edges is equal to the edge-induced subgraph of $DP^{2}_{Gen(II),f_{Comp(II)},f}$ w.r.t. left-edges.

Additionally, the precedence orders on their left-rules coincide. The proof of this fact illustrates that $UnitPropagate_L \text{ and } dAllRulesCancelled_L \text{ and } BackchainTrue_L$ is applicable in the same state in $SM^\prime \times DP^\prime_{L}$. The last result is remarkable as it illustrates close relation between solving technology for different propositional formalisms.

\textbf{Alternative Solvers} We now illustrate how transition systems introduced earlier may inspire the design of new solving procedures. We start by defining a graph that is a “symbiosis” of graphs $DP^{2}_{F,L}$ and $SM^{2}_{\Lambda,p}$.\textsuperscript{10}
A graph $DP \times SM_{F,p}$ for a CNF formula $F$ and a witness-program function $p$ is defined as follows. The set of nodes of $DP \times SM_{F,p}$ consists of the states relative to $At(F)$ and $At(F, p)$. The edges of the graph $DP \times SM_{F,p}$ are specified by (i) the Left-rules and Crossing-rules of the $DP_{F,p}$ graph, and (ii) the Right-rules of $SM_{F,p}$. This graph allows us to define a new procedure for deciding whether disjunctive answer set program has an answer set.

One can use this framework to define a theorem in the spirit of Theorem 6, in order to prove the correctness of, for instance, a procedure based on the graph $DP \times SM_{E:Decomp(T),Test}$.

6 Related Work and Conclusions

Lierler [15] introduced and compared the transition systems for the answer set solvers SMODELS and CMODELS for non-disjunctive programs. We extend that work as we design and compare transition systems for ASP procedures for disjunctive programs. Lierler [17] considered another extension of her earlier work by introducing transition rules that capture backjumping and learning techniques common in design of modern solvers. It is a direction of future work to extend the transition systems presented in this paper to capture backjumping and learning. This extension will allow us to model answer set solver CLASP for disjunctive programs as well as CMODELS that implements these features.

The approach based on transition systems for describing and comparing ASP procedures is one of the three main alternatives studied in the literature. The other methods include pseudo-code presentation of algorithms [9] and tableau calculi [7]. Giunchiglia et al. [9] presented pseudo-code descriptions of CMODELS (without backjumping and learning), SMODELS and DLV (without backjumping) restricted to non-disjunctive programs. They note the relation between solvers CMODELS and DLV on tight non-disjunctive programs. Gebser et al. [7] considered formal proof systems based on tableau methods for characterizing the operations and the strategies of ASP procedures for disjunctive programs. These proof systems also allow cardinality constraints in the language of logic programs, yet they do not capture backjumping and learning.

In this work we focused on developing graph-based representation for disjunctive answer set solvers GMT, DLV, and CMODELS implementing plain backtracking to allow simpler analysis and comparison of these systems. Similar effort for the case of non-disjunctive solvers resulted in design of a novel answer set solver SUP [17]. We believe that this work is a stepping stone towards clear, comprehensive articulation of main design features of current disjunctive answer set solvers that will inspire new solving algorithms. Sections 4 and 5 hint at some of the possibilities.

REFERENCES


A Proofs

In the appendix, we write $L^{\prec}$ so as to explicitly denote the assignment that corresponds to a string of literals. Also, we identify any empty clause with the clause \{\}, so that we can assume that in the studied CNF formulas no clause is empty.

The following lemma is used in proof of Theorem 2.

Lemma 1 Let $F$ be a CNF formula and $f$ be a witness-formula function. Let $(l_1, \ldots, l_{k_1}, l'_1, \ldots, l'_{k_2})$ be a state of the graph $SM_{x,y}$, reachable from the initial state. Then:

(a) any model of $f(l_1, \ldots, l_{k_1})$ satisfies $l_i'$ if it satisfies all decision literals $\{l'_j\}^{\prec}$ with $j \leq i$.
(b) any model of $F$ such that its witness is unsatisfiable satisfies $l_i$, if it satisfies all decision literals $\{l'_j\}^{\prec}$ with $j \leq i$.

Proof We prove the lemma by induction on the states of the graph. It obviously holds for $(\emptyset, \emptyset)$. Let us assume it holds for state $S = (l_1, \ldots, l_{k_1}, l'_1, \ldots, l'_{k_2})$. Let us prove that it holds for each successor of this state.

The rules $\text{Conclude}_L$, $\text{Conclude}_R$ and $\text{Conclude}_R$ are not of concern as the successors through these rules are not of the ruled type.

Case $\text{Decide}_L$ and $\text{Decide}_R$ obvious.

Case $\text{UnitPropagate}_L$: Assume $\text{UnitPropagate}_L$ is applied to $S$. By the rule’s definition $S$ has the form $(l_1, \ldots, l_{k_1}, \emptyset)$. Let us assume it holds for state $S = (l_1, \ldots, l_{k_1}, \emptyset, l'_1, \ldots, l'_{k_2})$. Let us prove that it holds for each successor of this state.

Case $\text{Decide}_L$ and $\text{Decide}_R$: obvious.

Case $\text{UnitPropagate}_R$: Assume $\text{UnitPropagate}_R$ is applied to $S$. By the rule’s definition $S$ has the form $(l_1, \ldots, l_{k_1}, \emptyset, l'_1, \ldots, l'_{k_2})$. Let us assume it holds for state $S = (l_1, \ldots, l_{k_1}, \emptyset, l'_1, \ldots, l'_{k_2})$. Let us prove that it holds for each successor of this state.

Claim (a) is obvious. Claim (b): by the conditions of $\text{UnitPropagate}_R$ there is a clause $C \lor \emptyset$ of $F$ such that all the literals of $C$ occur in $l_1, \ldots, l_{k_1}$ while $l$ does not occur in $l_1, \ldots, l_{k_1}$. Let $M$ be a model of $F$ of which witness is unsatisfiable, satisfying all the decision literals of $l_1, \ldots, l_{k_1}$. Then this model satisfies $C \lor l$. Also $M$ satisfies all the literals of $l_1, \ldots, l_{k_1}$ by the induction hypothesis.

Since all the literals of $C$ occur in $l_1, \ldots, l_{k_1}$ this model satisfies all the literals of $C$. So, since $M$ satisfies $C \lor l$ this model satisfies $l$.

Case $\text{UnitPropagate}_R$: The proof is similar to the case $\text{UnitPropagate}_L$. Claim (b) is proved the same way as claim (a) for $\text{UnitPropagate}_L$. Claim (a) trivially holds due to the inductive hypothesis.

Case $\text{Backtrack}_L$: Assume $\text{Backtrack}_L$ is applied to $S$. By the rule’s definition

- there is an index $i$ such that
  \[ l_1, \ldots, l_{k_1} = l_1, \ldots, l_{i-1}, l^2_{i+1}, \ldots, l_{k_1}, \]
  where none of the literals $l_{i+1}, \ldots, l_{k_1}$ is a decision literal,
- a successor of $S$ has the form $(l_1, \ldots, l_{i-1}, l_i, \emptyset)$
- no right-rule and no left-rule can apply to $S$.

Claim (a) trivially holds.

Claim (b): From the fact that $\text{Backtrack}_L$ is not applicable to $S$, it follows that $l'_1, \ldots, l'_{k_2}$ contains no decision literal. By the induction hypothesis any model of $f(l_1, \ldots, l_{k_1})$ satisfies all the literals of $l'_1, \ldots, l'_{k_2}$. As $\text{Conclude}_R$ could not be applied, and since no left-rule applied, $l'_1, \ldots, l'_{k_2}$ is consistent; so $(l'_1, \ldots, l'_{k_2})^{\prec}$ is well defined. Also, since $\text{Decide}_R$ could not be applied, all the atoms of the signature of $f(l_1, \ldots, l_{k_1})$ occur in $l'_1, \ldots, l'_{k_2}$. As a consequence the model $(l'_1, \ldots, l'_{k_2})^{\prec}$ is an assignment of all the atoms of the signature of $f(l_1, \ldots, l_{k_1})$. Consider any clause of $f(l_1, \ldots, l_{k_1})$. It has the form $C \lor l$. Then since $\text{UnitPropagate}_R$ could not be applied one literal of $C$ occurs in $l'_1, \ldots, l'_{k_2}$ or it occurs in $l'_1, \ldots, l'_{k_2}$. In both cases, one literal of $C \lor l$ occurs in $l'_1, \ldots, l'_{k_2}$. Consequently, $(l'_1, \ldots, l'_{k_2})^{\prec}$ satisfies this clause. It follows that $(l'_1, \ldots, l'_{k_2})^{\prec}$ is a model of $f(l_1, \ldots, l_{k_1})$.

Let $M$ be a model of $F$ of which witness is unsatisfiable, satisfying all the decision literals of $l_1, \ldots, l_{k_1}$. Then this model satisfies all the literals of $l_1, \ldots, l_{k_1}$ by the induction hypothesis. Since the witness of $l_1, \ldots, l_{k_1}$ is satisfiable, $M$ cannot satisfy all of its literals. So either this model does not satisfy $l_i$ or it does not satisfy a literal among $l_{i+1}, \ldots, l_{k_1}$. In the second case, by the induction hypothesis, $M$ does not satisfy one of the decision literals of $l_{i+1}, \ldots, l_{k_1}$. This literal cannot be among $l_{i+1}, \ldots, l_{k_1}$ since $M$ satisfies all these literals, and it cannot be among $l_1, \ldots, l_{i-1}$ as these are not decision literals. So it can only be $l_i$. Consequently, $M$ must satisfy $l_i$. We derive that claim (b) holds for $l_i$.

Theorem 2 For any CNF formula $F$ and a witness function $f$:

1. graph $\text{DP}_{F,f}$ is finite and acyclic.
2. any terminal state of $\text{DP}_{F,f}$ reachable from the initial state and other than $\text{Failstate}$ is $\text{Ok}(L)$, with $L$ being a model of $F$ such that $f(L)$ is unsatisfiable.
3. $\text{Failstate}$ is reachable from the initial state if and only if $F$ has no model such that its witness is unsatisfiable.

Proof Claim 1. Consider any state $(L, R)$ of the graph $\text{DP}_{F,f}$. The set of atoms over which $L$ is defined is bounded by the size $|F|$ of a formula. So, there is only a finite number of possible strings $L$ in the states $(L, R)$. Similar argument holds for $R$. Strings $L$ and $R$ allow no repetitions. Thus the set of states is finite in the graph $\text{DP}_{F,f}$.

For any string $L$ of literals, by $|L|$ we denote the length of this string. Any string of literals $L$ can be written $L_1 l_1^2 L_2 l_2^2 L_3 l_3^2 L_4$, where $(l_1^2 l_2^2 l_3^2 l_4^2)_{1\leq i\leq k}$ contains all the decision literals of $L$. Let us call $\alpha(L)$ the sequence $|L_1|, |L_2|, |L_3|, |L_4|$. We then write $L < L'$ iff $\alpha(L) < \alpha(L')$ where $<\text{lex}$ is the lexicographic order.

Then, for any states $(L, R)$ and $(L', R')$, if there is a transition from $(L, R)$ and $(L', R')$ then: either $L < L'$, or $L = L'$ and $R < R'$. This can be checked simply for each of the rules. As a consequence, by induction, for any states $(L, R)$ and $(L', R')$, if the state $(L', R')$ is reachable from $(L, R)$ then: either $L < L'$, or $L = L'$ and $R < R'$. It follows that $\text{DP}_{F,f}$ is acyclic.
Claim 2. We first illustrate that any terminal state other than \textit{Failstate} is of the form Ok(L) for some \textit{L}. By contradiction. Assume there is a terminal state of the form (L, R). Since Conclude_{L,R} does not apply unless there is no left-rule and no right-rule applies, \textit{L} contains at least one decision literal. This contradicts the fact that the rule Backtrack_{L,R} is not applicable.

Now, let Ok(L) be a terminal state reachable from the initial state. As it is different from the initial state there is a transition leading to it. This transition can only be Conclude_{L}. Let us call (L, R) a state from which a transition Conclude_{R} leads to Ok(L). By the definition of Conclude_{R}, no left-rule is applicable to state (L, R).

We now illustrate that \textit{L} is a model of \textit{F}. We first show that \textit{L} is consistent. By contradiction. Assume that \textit{L} is inconsistent. Since Conclude_{L} could not be applied, \textit{L} contains a decision literal. Also, since Backtrack_{L} could not be applied, \textit{L} contains no decision literal. We derive a contradiction.

Also, since Decide_{L} could not be applied, all the atoms of the signature of \textit{F} occur in \textit{L}. Since \textit{L} is consistent they occur only once. So \textit{L}^a is well defined and is an assignment of all the atoms of the signature of \textit{F}.

Let C ∨ l be any clause of \textit{F}. Then since UnitPropagate_{L} could not be applied either one literal of \textit{C} occurs in \textit{L} or \textit{l} occurs in \textit{L}. In both cases, one literal of \textit{C} ∨ \textit{l} occurs in \textit{L}. Consequently, \textit{L}^a satisfies this clause. It follows that \textit{L}^a is a model of \textit{F}.

We now illustrate that \textit{f}(\textit{L}) is unsatisfiable. By the definition of Conclude_{R}, \textit{R} contains no decision literal. So any model of \textit{f}(\textit{L}) satisfies all the decision literals of \textit{R} as there is none, by Lemma 1 any model of \textit{f}(\textit{L}) satisfies all the literals of \textit{R}. By the definition of Conclude_{R}, \textit{R} is inconsistent. There is no assignment that satisfies inconsistent \textit{R}. Thus, \textit{f}(\textit{L}) is unsatisfiable.

Claim 3. Right-to-left: From claim 1, it follows that there is a path from the initial state to some terminal state. From claim 2, it follows that this state cannot be different from \textit{Failstate}.

Left-to-right: Consider the case that \textit{Failstate} is reachable from the initial state. We now illustrate that \textit{F} has no model such that its witness is unsatisfiable. Since \textit{Failstate} can be reached from the initial state, either Conclude_{L} or Conclude_{L,R} has been applied to a state (L, R). In any case, \textit{L} does not contain any decision literal. By the Lemma 1, any model of \textit{F} such that its witness is unsatisfiable satisfies all the literals of \textit{L}.

Case 1. Conclude_{L} has been applied. Then \textit{L} is inconsistent. There is no assignment that satisfies inconsistent \textit{R}. It follows that there is no model of \textit{F} such that its witness is unsatisfiable.

Case 2. Conclude_{L,R} has been applied. By the definition of the graph, no right-rule and no left-rule is applicable to (L, R). Then no right-rule is applicable to (L, R).

We first illustrate that \textit{R}^a is a model of \textit{f}(\textit{L}). As Backtrack_{R} is not applicable to (L, R), \textit{R} contains no decision literal. By Lemma 1 any model of \textit{f}(\textit{L}) satisfies all the literals of \textit{R}. As Conclude_{R} is inapplicable to (L, R) as well as any left-rule, \textit{R} is consistent; so \textit{R}^a is well defined. Since Decide_{L} could not be applied, all the atoms of the signature of \textit{f}(\textit{L}) occur in \textit{R}. Consequently, the model \textit{R}^a is an assignment of all the atoms of the signature of \textit{f}(\textit{L}). Let C ∨ \textit{l} be any clause of \textit{f}(\textit{L}). Then since UnitPropagate_{R} is not applicable to (L, R), either one literal of \textit{C} occurs in \textit{R} or \textit{l} occurs in \textit{R}. In both cases, one literal of \textit{C} ∨ \textit{l} occurs in \textit{R}. So \textit{R}^a is a model of this clause. Consequently, \textit{R}^a is a model of \textit{f}(\textit{L}).

Since Conclude_{L} is not applicable to (L, R), and since \textit{L} contains no decision literal, \textit{L} is consistent and \textit{L}^a is well defined. Also, since Decide_{L} could not be applied, all the atoms of the signature of \textit{F} occur in \textit{L}. As a consequence the model \textit{L}^a is an assignment of all the atoms of \textit{F}. By Lemma 1 and the fact that \textit{L} contains no decision literal, any model of \textit{F} such that its witness is unsatisfiable satisfies each literal of \textit{L}. But \textit{f}(\textit{L}) is satisfiable, since one of its model is \textit{R}^a; so \textit{L}^a is not a suitable candidate. So there is no model of \textit{F} such that its witness is unsatisfiable.

Theorem 3 follows immediately from Theorem 2 and the definitions of DP-approximating and DP-ensuring.

Corollary 2 from (Sacca and Zaniolo 1990) states that: For any model \textit{M} of a program \textit{II}, \textit{M}^+ is an answer set for \textit{II} if and only if \textit{M} is unfounded-free. This is an important property that following proofs rely on.

The following lemma is used in proof of theorem 4.

Lemma 2 Let \textit{\Lambda} be a program and \textit{p} be a witness-program function. Let (l_1, \ldots, l_k, \textit{l}_k') be a state of the graph reachable from the initial state in the graph \textit{SM}^2_{\textit{p},\textit{p}}. Let \textit{M} be a consistent and complete set of literals over atoms occurring in states in \textit{SM}^2_{\textit{p},\textit{p}}.

(a) If \textit{M}^+ is an answer set of \textit{p}(l_1, \ldots, l_k), then \textit{M} satisfies \textit{l}_k' if \textit{M} satisfies all decision literals \textit{l}^a_k with \textit{j} ≤ \textit{i}.

(b) If \textit{M}^+ is an answer set of \textit{\Lambda} and \textit{p}(\textit{M}) has no answer set. Then \textit{M} satisfies \textit{l}_k' if \textit{M} satisfies all decision literals \textit{l}^a_k with \textit{j} ≤ \textit{i}.

Proof The proof is similar to the proof of Lemma 1. To prove properties of cases due to AllRulesCancelled_{\textit{L}}, AllRulesCancelled_{\textit{R}}, BackchainTrue_{\textit{L}}, BackchainTrue_{\textit{R}}, Unfounded_{\textit{L}}, and Unfounded_{\textit{R}}, it will rely on the arguments made in proofs of Lemma 2 and Lemma 5 in [17] for the transition rules AllRules Cancelled, Backchain True, and Unfounded of the graph \textit{SM}_\textit{\Lambda}. Corollary 2 from (Sacca and Zaniolo 1990) is important in stating these results.

Theorem 4 For any non-disjunctive program \textit{\Lambda} and a witness function \textit{p}:

1. graph \textit{SM}^2_{\textit{p},\textit{p}} is finite and acyclic.
2. any terminal state of \textit{SM}^2_{\textit{p},\textit{p}} reachable from the initial state and other than \textit{Failstate} is Ok(L), with \textit{L}^+ being an answer set of \textit{\Lambda} such that \textit{p}(\textit{L}) has no answer set.
3. \textit{Failstate} is reachable from the initial state if and only if there is no set \textit{L} of literals such that \textit{L}^+ is an answer set of \textit{\Lambda} and \textit{p}(\textit{L}) has no answer set.

Proof Proof of Claim 1 follows the lines of the proof of claim 1 in Theorem 2.

Claim 2. The proof of claim 2 of Theorem 2 shows us that any terminal state other than \textit{Failstate} is Ok(L) for some \textit{L}. It also shows that for any terminal state Ok(L) reached from a state (L, \textit{R}), the assignment \textit{L}^a is a model of \textit{\Lambda}. Also, by applying Lemma 2 instead of Lemma 1, we know that \textit{p}(\textit{L}) has no answer sets. Remains to prove that \textit{L}^+ is an answer set of \textit{\Lambda}.

Since AllRulesCancelled_{\textit{L}} can not be applied and \textit{L}^a is a model of \textit{\Lambda}, we conclude that \textit{L}^a is a supported model of \textit{\Lambda}. Since Unfounded_{\textit{L}} can not be applied, we conclude that \textit{L}^+ is unfounded-free. Since \textit{L}^a is also a model of \textit{\Lambda}, \textit{L}^+ an answer set of \textit{\Lambda} by Corollary 2 from Sacca and Zaniolo 1990.

Claim 3. Right-to-left is proved the same straightforward way as in the proof of Theorem 2. For left-to-right, the case of Conclude_{\textit{L}} is also handled the same way as Theorem 2, using Lemma 2 instead of Lemma 1. Remains the case of Conclude_{\textit{L},\textit{R}}. Corollary 2 from
The following lemma is essential in a proof of Theorem 4’ that we state immediately after.

**Lemma 3** Let $\Lambda$ be a program and $p$ be a witness-program function. Let $(l_1, \ldots, l_{k_1}, l'_1, \ldots, l'_{k_2})$ be a state of the graph reachable from the initial state in the graph $SM' \times SM_{\Lambda, p}$. Let $M$ be a consistent and complete set of literals over atoms occurring in states in $SM' \times SM_{\Lambda, p}$.

- If $M^+$ is an answer set of $p(l_1, \ldots, l_{k_1})$. Then $M$ satisfies $l_i$ if $M$ satisfies all decision literals $(l'_j)^+_{\Lambda}$ with $j \leq i$.
- If $M^+$ is a supported model of $\Lambda$ and $p(M)$ has no answer set. Then $M$ satisfies $l_i$ if $M$ satisfies all decision literals $l'_j^+$ with $j \leq i$.

**Theorem 4’** For any non-disjunctive program $\Lambda$ and a witness function $\nu$:

1. graph $SM' \times SM_{\Lambda, p}$ is finite and acyclic,
2. any terminal state of $SM' \times SM_{\Lambda, p}$ reachable from the initial state and other than Failstate is Ok($L$), with $L$ being a supported model of $\Lambda$ such that $p(L)$ has no answer set.
3. Failstate is reachable from the initial state if and only if there is no set $L$ of literals such that $L$ is a supported model of $\Lambda$ and $p(L)$ has no answer set.

Proofs of Lemma 3 and Theorem 5 are in style of similar claims in earlier lemmas and theorems. The essence of the proofs lies in the results that Lierler [17] established earlier. She introduced a graph ATLEAS$\nu$ whose key property is such that its terminal states corresponded to supported models of program $\Lambda$. The graph ATLEAS$\nu$ differs from the graph $SM_\Lambda$ by the lack of the transitions due to the rules Unfounded. This is precisely the difference between the graphs $SM' \times SM_{\Lambda, p}$ and $SM^2_{\Lambda, p}$.

Theorem 5 is a clear corollary of Theorems 4 and Theorems 4’.

The following lemma is used in proof of Theorem 6.

**Lemma 4** Let $\Pi$ be a program and $f$ be a witness-formula function. Let $(l_1, \ldots, l_{k_1}, l'_1, \ldots, l'_{k_2})$ be a state of the graph reachable from the initial state in $SM \times DP_{\Pi, f}$. Then:

(a) Let $M$ be a model of $f(l_1, \ldots, l_{k_1})$. Then $M$ satisfies $l'_i$ if $M$ satisfies all decision literals $(l'_j)^+_{\Pi, f}$ with $j \leq i$.

(b) Let $M$ be a consistent and complete set of literals over $\Pi$, such that $M^+$ is an answer set of $\Pi$. Then $M$ satisfies $l_i$ if $M$ satisfies all decision literals $l'_j^+$ with $j \leq i$.

**Proof** Mostly, the proof is similar to that of Lemma 1. We prove the lemma by induction on the states of the graph. It obviously holds for $(0, 0)$. Let us assume it holds for state $S = (l_1, \ldots, l_{k_1}, l'_1, \ldots, l'_{k_2})$. Let us prove that it holds for each successor of this state.

First, let us notice that that if $M$ is an answer set of $\Pi$ then $f(M)$ has no model. So claim (b) to the lemma can be equivalently stated as follows. “Let $M$ be a model such that $M^+$ is an answer set of $\Pi$ and $f(M)$ has no answer set. Then $M$ satisfies $l_i$ if $M$ satisfies all decision literals $l'_j^+_{\Pi, f}$ with $j \leq i$.”

The rules Conclude$_L$, Conclude$_{ED}$ and Conclude$_{L, R}$ are not concerned as the successors through these rules are not of the studied type. Concerning the rules Decide$_L$ and Decide$_{R}$, the reasoning is obvious.

The following lemma is essential in a proof of Theorem 4’ that we state immediately after.

**Proof** Claim 1 is proved the same way as claim 1 of the theorem 2.

Claim 2. The proof of claim 2 of Theorem 2 shows us that any terminal state other than Failstate is Ok($L$) for some $L$. It also shows that for any terminal state Ok($L$) reached from a state $(L, R)$, the assignment $L^*$ is a model of $\Pi$. Also, by applying Lemma 4 instead of Lemma 1, we know that $f(L)$ has no model.

Thanks to the property we have made $f$ satisfy, $L^+$ is an answer set of $\Pi$. 

(Sacca and Zaniolo 1990) is essential in the following claims. Applying the same way the technique of Theorem 2, we obtain that $R$ is a model of $p(L)$, and that any answer set $M$ of $\Lambda$ such that $p(M)$ has no answer set is equal to $L^+$. Since $UnitPropagate_R$ is not applicable to $(L, R)$, $R^*$ is a model of $p(L)$. Since Unfoundedy is not applicable to $(L, R)$, we conclude that $R^*$ is unfounded-free. Since $R^*$ is also a model of $p(L)$, we conclude that $R^*$ is an an answer set of $p(L)$.

So $L^+$ is not such that $p(L)$ has no answer set. Since we have earlier proved that $L^+$ is the only candidate, $\Lambda$ has no answer set $M$ such that $p(M)$ has no answer set.

The proofs of Lemma 3 and Theorem 5 are in style of similar claims in earlier lemmas and theorems. The essence of the proofs lies in the results that Lierler [17] established earlier. She introduced a graph $ATLS^\nu$, whose key property is such that its terminal states corresponded to supported models of program $\Lambda$. The graph $ATLS^\nu$ differs from the graph $SM_\Lambda$ by the lack of the transitions due to the rules Unfounded. This is precisely the difference between the graphs $SM' \times SM_{\Lambda, p}$ and $SM^2_{\Lambda, p}$.
Claim 3: right-to-left is proved the same straightforward way as in the proof of Theorem 2. For left-to-right, the case of ConcludeL is also handled the same way as Theorem 2, using Lemma 4 instead of Lemma 1. Remains the case of ConcludeL,R. Applying the same way the technique of Theorem 2, we obtain that R is a model of f(L), and that any model M of Π such that f(M) has no model equal to L+.

Again thanks to the property we have made f satisfy, L+ is not an answer set of Π. As a consequence of what has been stated just above, Π does not have any answer set.

The following lemma is used in the proof of Theorem 7.

**Lemma 5** Let F be a DNF formula. Let l be a literal of F. The two following statements are equivalent:

- there is a conjunctive clause D of F such that for all D′ ≠ D ∈ F the conjunctive clause D′ is contradicted by L,
- there is a clause C of CNF(F) s.t. l ∈ C and L contradicts C \ {l}.

**Proof** Let F be a formula in DNF. We can assume that F = \bigvee_{i=1}^{m} l_{im_i}, if necessary adding the true constant ⊤ enough times to the shorter conjunctive clauses so as to have clauses of which lengths are equal. Also CNF(F) = \bigwedge_{(m_1,..,m_k)\in\{(1,..,k)\}^n} \bigvee_{i=1}^{m} l_{im_i}.

Assume that for some clause of CNF(F), only one literal is not contradicted by L. Then let this clause be \bigvee_{i=1}^{m} l_{im_i}, for some i and let l_{i0}, m_i be the literal that is not contradicted by L. Then l_{im_i} is contradicted by L for any i other than i_0. So \bigwedge_{j=1}^{k} l_{ij} is contradicted by L for any i other than i_0. So D = \bigwedge_{j=1}^{k} l_{ij} is a conjunctive clause of F such that for any other conjunctive clause D′ of F, this clause is contradicted by L.

Assume that there is a conjunctive clause D of F such that for any other conjunctive clause D′ of F, this clause is contradicted by L. Let f be a literal of D. Let D be \bigwedge_{j=1}^{k} l_{ij} for some i_0. As any other conjunctive clause is contradicted by L, and as these clauses are conjunctive, there is least one literal of each of these clauses that is contradicted by L. Let us call b_1, ..., b_{n-1}, b_n these literals. Then for each i \in \{1,..,n\} \{1,..,n\}, there is m_i ∈ \{1,..,k\} such that l_{im_i} = b_i. Also, there is some m_{i_0} such that l_{im_{i_0}} = l. Then the clause \bigvee_{j=1}^{m} l_{im_0} of CNF(F) contains l while each of the other literals it contains is contradicted by L.

**Theorem 7** For a disjunctive program Π, the edge-induced subgraph of SM+ × DΠf w.r.t. left-edges is equal to the edge-induced subgraph of DΠCNF-comp(Π),f w.r.t. left-edges.

**Proof** Left-to-right: We must prove that for any left-edge in SM+ × DΠf there is a left-edge in DΠCNF-comp(Π),f linking two identical vertices.

If the edge is DecideL, ConcludeL or BacktrackL then obviously there is the same edge in DΠCNF-comp(Π),f, bearing the same name, as these edges do not depend on the program or formula studied.

If the edge is UnitPropagateL then also there is an UnitPropagateL edge in DΠCNF-comp(Π),f with the same effect, applied to the Πf part of Comp(Π).

If the edge is dAllRulesCancelledL turning (L,∅) into (L,∅) then for each rule a ∨ X ← B ∈ Π the conjunction B is contradicted by L. As a consequence, for all of these rules B ∧ X is contradicted by L. As a consequence \bigvee_{X∧+\neg¬B∈Π}(B∧\neg X) is contradicted by L. Hence, as the formula ¬a ∨ \bigvee_{X∧+\neg¬B∈Π}(B∧\neg X) belongs to Comp(Π), by Lemma 5 there is a clause C in CNF(Comp(Π)) = CNF – Comp(Π) s.t. ¬a ∈ C and L contradicts C \ {¬a}. So the rule UnitPropagateL of DΠCNF-comp(Π),f can be applied to C to add ¬a, providing the edge we needed.

If the edge is dBackchainTrueL, turning (L,∅) into (L,∅), then there is a rule a ∨ X ← B ∈ Π with l ∈ B ∪ X and a ∈ L such that for each other rule a ∨ X ← B ∈ Π the conjunction B is contradicted by L. As a consequence of the above, L contradicts all of {¬a} ∪ {B’ ∧ X’|X’ ∨ a ← B’ ∈ Π \ {a ∨ X ← B}}. Since ¬a ∨ \bigvee_{X∧+¬B∈Π}(B∧\neg X) belongs to Comp(Π), and l ∈ B ∪ X, by Lemma 5 there is a clause C in CNF(Comp(Π)) = CNF – Comp(Π) s.t. l ∈ C and L contradicts C \ {l}. So the rule UnitPropagateL of DΠCNF-comp(Π),f can be applied to C to add l, providing the edge we needed.

Right-to-left: For DecideL, ConcludeL or BacktrackL, this is obvious.

For UnitPropagateL, there are three cases. Let us call F_0 the formula ¬a ∨ \bigvee_{X∧+¬B∈Π}(B∧\neg X).

**Case 1:** UnitPropagateL is applied to a clause of Πf. Then UnitPropagateL itself provides the desired edge in SM+ × DΠf.

**Case 2:** UnitPropagateL is applied to a clause obtained from F_0. Then by lemma 5, there is a conjunctive clause D of F_0 such that for all D′ ≠ D ∈ F_0 the current L contradicts D′.

**Case 2:** This conjunctive clause is ¬a. Then L contradicts \bigvee_{X∧+¬B∈Π}(B∧\neg X). So dAllRulesCancelledL provides the desired edge.

**Case 2:** This conjunctive clause is some B ∧ X. Then L contradicts ¬a so a belongs to L. Also L contradicts all of {B’ ∧ X’ ∨ a ← B’ ∈ Π \ {a ∨ X ← B}}. As a consequence dBackchainTrueL provides the desired edge.