TLP **16** (4): 465–497, 2016. © Cambridge University Press 2015
doi:10.1017/S1471068415000411 First published online 17 December 2015

Disjunctive answer set solvers via templates

REMI BROCHENIN and MARCO MARATEA
DIBRIS, University of Genova, Viale F. Causa 15, 16145, Genova, Italy
(*e-mail:* `remi.brochenin@unige.it, marco@dibris.unige.it`)

YULIYA LIERLER
Department of Computer Science, University of Nebraska at Omaha, 6001 Dodge Street, Omaha, NE 68182
(*e-mail:* `ylierler@unomaha.edu`)

submitted 9 April 2015; accepted 2 August 2015

Abstract

Answer set programming is a declarative programming paradigm oriented towards difficult combinatorial search problems. A fundamental task in answer set programming is to compute stable models, i.e., solutions of logic programs. Answer set solvers are the programs that perform this task. The problem of deciding whether a disjunctive program has a stable model is Σ_2^P-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, CLASP and WASP. In this paper, we show that transition systems introduced by Nieuwenhuis, Oliveras, and Tinelli to model and analyze satisfiability solvers can be adapted for disjunctive answer set solvers. Transition systems give a unifying perspective and bring clarity in the description and comparison of solvers. They can be effectively used for analyzing, comparing and proving correctness of search algorithms as well as inspiring new ideas in the design of disjunctive answer set solvers. In this light, we introduce a general template, which accounts for major techniques implemented in disjunctive solvers. We then illustrate how this general template captures solvers DLV, GNT, and CMODELS. We also show how this framework provides a convenient tool for designing new solving algorithms by means of combinations of techniques employed in different solvers.

KEYWORDS: answer set programming, abstract solvers

1 Introduction

Answer set programming (ASP) (Gelfond and Lifschitz 1988; Gelfond and Lifschitz 1991; Eiter *et al.* 1997; Marek and Truszczyński 1999; Niemelä 1999; Baral 2003) is a declarative programming paradigm oriented towards difficult combinatorial search problems. The idea of ASP is to represent a given problem with a logic program, whose answer sets correspond to solutions of the problem (see e.g., Lifschitz 1999). ASP has been applied to solve problems in various areas of science and technology including graph-theoretic problems arising in zoology and linguistics (Brooks *et al.* 2007), team-building problems in container terminal (Ricca *et al.* 2012), and product configuration tasks (Soininen and Niemelä 1999). A fundamental task in ASP is to compute stable models of logic programs. Answer set solvers are the programs that perform this task. There were 16 answer set solvers participating in the recent Fifth Answer Set Programming Competition[1].

Gelfond and Lifschitz introduced logic programs with disjunctive rules (Gelfond and Lifschitz 1991). The problem of deciding whether a disjunctive program has a stable model is Σ_2^P-complete (Eiter and Gottlob 1993). The problem of deciding whether a non-disjunctive program has a stable model is NP-complete. The high complexity of reasoning within disjunctive logic programming stems from two sources: first, there is a potentially exponential number of candidate models, and, second, the hardness of checking whether a candidate model is a stable model of a propositional disjunctive logic program is co-NP-complete. Only five answer set systems can solve disjunctive programs: DLV (Leone *et al.* 2006), GNT (Janhunen *et al.* 2006), CMODELS (Lierler 2005), CLASP (Gebser *et al.* 2013), and WASP (Alviano *et al.* 2013).

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 (Giunchiglia and Maratea 2005; Giunchiglia *et al.* 2008), to tableau calculi (Gebser and Schaub 2006; Gebser and Schaub 2013), to abstract frameworks via transition systems (Lierler 2008; Lierler 2011; Lierler and Truszczynski 2011). The latter method originates from the work by Nieuwenhuis *et al.* (2006), where authors propose to use transition systems to describe the Davis–Putnam–Logemann–Loveland (DPLL) procedure (Davis *et al.* 1962). Nieuwenhuis et al. introduce an abstract framework called 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. Some edges may correspond to unit propagation steps, some to branching, some to backtracking.

Such an abstract way of presenting algorithms simplifies their analysis. This approach has been adapted (Lierler 2011; Lierler and Truszczynski 2011) to describe answer set solvers for *non-disjunctive* programs including SMODELS, CMODELS, and CLASP. This type of graphs has been used to relate algorithms in precise mathematical terms. Indeed, once we represent algorithms via graphs, comparing the graphs translates into studying the relationships of underlying algorithms. More generally, the unifying perspective of transition systems brings clarity in the description and comparison of solvers. Practically, such graph representations may serve as an effective tool for analyzing, comparing, proving correctness of, and reasoning formally about the underlying search algorithms. It may also inspire new ideas in the design of solvers.

In this paper, we present transition systems that suit multiple *disjunctive* answer set solvers. We define a general framework, a *graph template*, which accounts for major techniques implemented in disjunctive answer set solvers excluding backjumping and learning. We study formal properties of this template and we use the template to describe GNT, CMODELS, and DLV implementing plain backtracking. We then show

[1] `https://www.mat.unical.it/aspcomp2014/FrontPage#Participant_Teams`

how a graph template facilitates a design of new solving algorithms by means of combinations of techniques employed in different solvers. For instance, we present a new abstract solver that can be seen as a hybrid between CMODELS and GNT. We also present how different solvers may be compared by means of transition systems. In particular, we illustrate a close relationship between answer set solvers DLV and CMODELS through the related graphs. The fact that proposed framework does not account for backjumping and learning is one of the reasons that prevents us from capturing such advanced disjunctive answer set solvers as CLASP and WASP. It is a direction of future work to investigate how the proposed framework can be adjusted to accommodate these solvers in full generality.

The paper is structured as follows. Section 2 introduces required preliminaries. Section 3 presents a first abstract solver related to CMODELS. Section 4 defines our general template that accounts for techniques implemented in disjunctive solvers, and Section 5 uses this template to define abstract frameworks for disjunctive solvers. Proofs are presented in Section 6. Section 7 discusses related work and concludes with the final remarks.

The current paper builds on the content presented by Brochenin *et al.* (2014). It enhances the earlier work by introducing notions of a graph template, “propagator conditions”, and “approximating pairs” that allow to more uniformly account for major techniques implemented in disjunctive answer set solvers. Complete proofs of the formal results are also provided.

2 Preliminaries

2.1 Formulas, logic programs, and program's completion

Formulas. Atoms are Boolean variables over $\{true, false\}$. The symbols $\bot$ and $\top$ are the *false* and the *true* constants, respectively. The letter l denotes a literal, that is an atom a or its negation $\neg a$, and $\bar{l}$ 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 $\bot$, $\top$ in usual way. A finite disjunction of literals is a *clause*. We identify an empty clause with the symbol $\bot$. A conjunction (resp. a disjunction) of literals will sometimes be seen as a set, containing each of its literals. Since a clause is identified with a set of its literals, there are no repetition of literals in a clause. A *CNF formula* is a finite conjunction (alternatively, a set) of clauses. Since a CNF formula is identified with a set of clauses, there are no repetition of clauses in a CNF formula.

For a conjunction (resp. a disjunction) D of literals, by $\overline{D}$ we denote the disjunction (resp. the conjunction) of the complements of the elements of D. For example, $\overline{a \vee \neg b}$ denotes $\neg a \wedge b$, while $\overline{a \wedge \neg b}$ denotes $\neg a \vee b$. For a set L of literals, by $L^{\vee}$ we denote the disjunction of its elements and $L^{\wedge}$ the conjunction of its elements; by $atoms(L)$, we denote the set of atoms occurring in L. For a set N of sets of literals by $atoms(N)$, we denote the set of atoms occurring in the elements of N. For example, $atoms(\{a, \neg b\}) = \{a, b\}$ and $atoms(\{\{a\}, \{\neg b\}\}) = \{a, b\}$. For a set L of literals, by L^{+} we denote atoms that occur positively in L. For instance, $\{a, \neg b\}^{+} = \{a\}$. For a set

X of atoms and a set L of literals, by $L_{|X}$ we denote the maximal subset of L over X. For example, $\{a, \neg b, c\}_{|\{a,b\}} = \{a, \neg b\}$.

A *(truth) assignment* to a set X of atoms is a function from X to $\{false, true\}$. An assignment *satisfies* a formula F if F evaluates to *true* under this assignment. We call an assignment that satisfies formula F a *satisfying assignment* or a *(classical) model* for F. If F evaluates to *false* under an assignment, we say that this assignment *contradicts* F. If F has no model, we say that F is *unsatisfiable*. For sets X and Y of atoms such that $X \subseteq Y$, we identify X with an assignment to Y as follows: if $a \in X$ then a maps to *true*, while if $a \in Y \setminus X$ then a maps to *false*. We also identify a consistent set L of literals (i.e., a set that does not contain both a literal and its complement) with an assignment to $atoms(L)$ as follows: if $a \in L$ then a maps to *true*, while if $\neg a \in L$ then a maps to *false*. The set M is a complete set of literals over the set of atoms X if $atoms(M) = X$; hence, a consistent and complete set of literals over X represents an assignment to X.

Logic Programs. A *head* is a (possibly empty) disjunction of atoms. A *body* is an expression of the form

$$a_1, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_k \tag{1}$$

where $a_1, \ldots, a_k$ are atoms, and *not* is the negation-as-failure operator. We identify body (1) with the following conjunction of literals

$$a_1 \wedge \ldots \wedge a_j \wedge \neg a_{j+1} \wedge \ldots \wedge \neg a_k.$$

Expressions $a_1, \ldots, a_j$ and $not\ a_{j+1}, \ldots, not\ a_k$ are called *positive* and *negative* parts of the body, respectively. Recall that we sometimes view a conjunction of literals as a set containing all of its literals. Thus, given body B we may write an expression $b \in B$, which means that atom b occurs in the positive part of the body. Similarly, an expression $\neg b \in B$ means that the atom b (or, in other words, expression *not* b) occurs in the negative part of the body.

A *disjunctive rule* is an expression of the form $A \leftarrow B$, where A is a head and B is a body. If A is empty, we drop it from the expression. A *disjunctive logic program* is a finite set of *disjunctive rules*. We call a rule *non-disjunctive* if its head contains no more than one atom. A program is *non-disjunctive* if it consists of non-disjunctive rules. By $atoms(\Pi)$, we denote the set of atoms occurring in a logic program Π. If we understand $A \leftarrow B$ as a classical logic implication, we can see any rule $A \leftarrow B$ as logically equivalent to clause $A \vee \overline{B}$ (if A is an empty clause, then we view the rule as the clause $\overline{B}$). This allows us to view a program Π as a CNF formula when useful. Conversely, we identify CNF formulas with logic programs: syntactically, every clause C in a given formula is seen as a rule $\leftarrow C$. For instance $a_1 \vee \neg a_2$ is seen as a rule $\leftarrow not\ a_1, a_2$.

The presented definition of a logic program accounts for propositional programs only. Indeed, all modern disjunctive answer set solvers consider propositional programs only. In practice, answer set programmers devise programs with variables. Software systems called grounders (Syrjänen 2001; Perri *et al.* 2007) are used to take a logic program with variables as its input and produce a propositional program as its output so that the resulting propositional program has the same answer sets as the input program.

Reduct and Supporting Rules. In the following definition, we write rules in the form $A \leftarrow B_1, B_2$ where B_1 denotes the positive part of the body, whereas B_2 denotes the negative part of the body. The *reduct* Π^X of a disjunctive program Π with respect to a set X of atoms is obtained from Π by deleting each rule $A \leftarrow B_1, B_2$ such that $X \cap atoms(B_2) \neq \emptyset$ and replacing each remaining rule $A \leftarrow B_1, B_2$ with $A \leftarrow B_1$. A set X of atoms is an *answer set* of a program Π if X is minimal among the sets of atoms that satisfy Π^X.

For a program Π, an atom a, and a set L of literals, we call any rule $A \vee a \leftarrow B$ in Π a *supporting* rule for a with respect to L when $L \cap (\overline{B} \cup A) = \emptyset$.

A consistent and complete set L of literals over $atoms(\Pi)$ is

(1) a *classical model* of Π if L satisfies every rule in Π;
(2) a *supported model* of Π if L is a classical model of Π and for every atom $a \in L^+$ there is a supporting rule for a with respect to L;
(3) a *stable model* of program Π if L^+ is an answer set of Π.

Completion. The *completion* $comp(\Pi)$ of a program Π is the formula that consists of Π and the formulas

$$\{\neg a \vee \bigvee_{A \vee a \leftarrow B \in \Pi} (B \wedge \overline{A}) \mid a \in atoms(\Pi)\}. \tag{2}$$

This formula has the property that any stable model of Π is a classical model of $comp(\Pi)$. The converse does not hold in general.

For a program Π and a consistent set L of literals over $atoms(\Pi)$, a set X of atoms over $atoms(\Pi)$ is said to be *unfounded* (Leone *et al.* 1997) on L with respect to the program Π when for each atom $a \in X$ and each rule $A \leftarrow B \in \Pi$ such that $a \in A$, either of the following conditions hold:

(1) $L \cap \overline{B} \neq \emptyset$,
(2) $X \cap B \neq \emptyset$, or
(3) $(A \setminus X) \cap L \neq \emptyset$.

We restate Theorem 4.6 from Leone *et al.* (1997) that relates the notions of unfounded set and stable model.

Theorem 1
For a program Π and a consistent and complete set L of literals over $atoms(\Pi)$, L is a stable model of Π if and only if L is a classical model of Π and no non-empty subset of L^+ is an unfounded set on L with respect to Π.

This theorem is crucial for understanding key computational ideas behind modern answer set solvers.

2.2 *Abstract* DPLL

The DPLL algorithm from Davis *et al.* (1962) is a well-known method that exhaustively explores sets of literals to generate classical models of a propositional formula.

$$\begin{gathered}
\emptyset,\ \bot,\ a,\ \neg a,\ a^\Delta,\ \neg a^\Delta,\ a\ \bot,\ \bot\ a,\ a^\Delta\ \bot,\ \bot\ a^\Delta,\ \neg a\ \bot,\ \bot\ \neg a,\ \neg a^\Delta\ \bot,\ \bot\ \neg a^\Delta,\\
a\ \neg a,\ a^\Delta\ \neg a,\ a\ \neg a^\Delta,\ a^\Delta\ \neg a^\Delta,\ \bot\ a\ \neg a,\ \bot\ a^\Delta\ \neg a,\ \bot\ a\ \neg a^\Delta,\ \bot\ a^\Delta\ \neg a^\Delta,\\
a\ \neg a\ \bot,\ a^\Delta\ \neg a\ \bot,\ a\ \neg a^\Delta\ \bot,\ a^\Delta\ \neg a^\Delta\ \bot,\ a\ \bot\ \neg a,\ a^\Delta\ \bot\ \neg a,\ a\ \bot\ \neg a^\Delta,\ a^\Delta\ \bot\ \neg a^\Delta,\\
\neg a\ a,\ \neg a^\Delta\ a,\ \neg a\ a^\Delta,\ \neg a^\Delta\ a^\Delta,\ \bot\ \neg a\ a,\ \bot\ \neg a^\Delta\ a,\ \bot\ \neg a\ a^\Delta,\ \bot\ \neg a^\Delta\ a^\Delta,\\
\neg a\ a\ \bot,\ \neg a^\Delta\ a\ \bot,\ \neg a\ a^\Delta\ \bot,\ \neg a^\Delta\ a^\Delta\ \bot,\ \neg a\ \bot\ a,\ \neg a^\Delta\ \bot\ a,\ \neg a\ \bot\ a^\Delta,\ \neg a^\Delta\ \bot\ a^\Delta.
\end{gathered}$$

Fig. 1. Records relative to $\{a\}$.

Most satisfiability and non-disjunctive answer set solvers are based on variations of the DPLL procedure that is a classical backtrack search-based algorithm. We now review the abstract transition system for DPLL proposed by Nieuwenhuis *et al.* (2006), which is an alternative to common pseudo-code descriptions of backtrack search-based algorithms. For our purposes, it is convenient to state DPLL as the procedure applied to a logic program in order to find its classical models.

For a set X of atoms, a *record* relative to X is a string L composed of literals over X or the symbol $\bot$ so that there are no repetitions, and some literals l may be annotated as l^Δ. The annotated literals are called *decision* literals. Figure 1 presents the set of all records relative to the singleton set $\{a\}$. We say that a record L is *inconsistent* if it contains both a literal l and its complement $\bar{l}$, or if it contains $\bot$, and *consistent* otherwise. For instance, only five records in Figure 1, namely $\emptyset$, a, $\neg a$, a^Δ and $\neg a^\Delta$, are consistent. We will sometime view a record as the set containing all its elements disregarding their annotations. For example, a record $b^\Delta\ \neg a$ is identified with the set $\{\neg a, b\}$. A *basic state* relative to X is either

(1) a record relative to X,
(2) $Ok(L)$ where L is a record relative to X, or
(3) the distinguished state *Failstate*.

Each program Π determines its DPLL *graph* DP_Π. The set of nodes of DP_Π consists of the basic states relative to $atoms(\Pi)$. A node in the graph is *terminal* if no edge originates from it. The state $\emptyset$ is called *initial*. The edges of the graph DP_Π are specified by the transition rules presented in Figure 2.

Intuitively, every state of the DPLLgraph represents some hypothetical state of the DPLL computation whereas a path in the graph is a description of a process of search for a classical model of a given program. The rule *Unit* asserts that we can add a literal that is a logical consequence of our previous decisions and the given program. The rule *Decide* asserts that we make an arbitrary decision to add a literal or, in other words, to assign a value to an atom. Since this decision is arbitrary, we are allowed to backtrack at a later point. The rule *Backtrack* asserts that the present state of computation is inconsistent but can be fixed: at some point in the past, we added a decision literal whose value we can now reverse. The rule *Conclude* asserts that the current state of computation has failed and cannot be fixed. The rule *Success* asserts that the current state of computation corresponds to a successful outcome.

We say that a graph G *checks* a set N of sets of literals when all the following conditions hold:

(1) G is finite and acyclic;
(2) Any terminal state in G is either *Failstate* or of the form $Ok(L)$;

$$
\begin{array}{lll}
\mathit{Conclude}: & L \Longrightarrow \mathit{Failstate} & \text{if } \begin{cases} L \text{ is inconsistent and} \\ L \text{ contains no decision literals} \end{cases} \\[2ex]
\mathit{Backtrack}: & Ll^{\Delta}L' \Longrightarrow L\bar{l} & \text{if } \begin{cases} Ll^{\Delta}L' \text{ is inconsistent and} \\ L' \text{ contains no decision literals} \end{cases} \\[2ex]
\mathit{Unit}: & L \Longrightarrow Ll & \text{if } \begin{cases} l \text{ does not occur in } L \text{ and} \\ \text{a rule in } \Pi \text{ is equivalent to } C \vee l \text{ and} \\ \text{all the literals of } \overline{C} \text{ occur in } L \end{cases} \\[2ex]
\mathit{Decide}: & L \Longrightarrow Ll^{\Delta} & \text{if } \begin{cases} L \text{ is consistent and} \\ \text{neither } l \text{ nor } \bar{l} \text{ occur in } L \end{cases} \\[2ex]
\mathit{Success}: & L \Longrightarrow Ok(L) & \text{if no other rule applies}
\end{array}
$$

Fig. 2. Transitions of the graph DP_{Π}.

$$
\begin{array}{lcl|lcl}
 & & & \text{Initial state}: & & \emptyset \\
\text{Initial state}: & & \emptyset & \mathit{Decide} & \Longrightarrow & a^{\Delta} \\
\mathit{Decide} & \Longrightarrow & a^{\Delta} & \mathit{Decide} & \Longrightarrow & a^{\Delta}\ \neg c^{\Delta} \\
\mathit{Unit} & \Longrightarrow & a^{\Delta}\ c & \mathit{Unit} & \Longrightarrow & a^{\Delta}\ \neg c^{\Delta}\ c \\
\mathit{Decide} & \Longrightarrow & a^{\Delta}\ c\ b^{\Delta} & \mathit{Backtrack} & \Longrightarrow & a^{\Delta}\ c \\
\mathit{Success} & \Longrightarrow & Ok(a^{\Delta}\ c\ b^{\Delta}) & \mathit{Decide} & \Longrightarrow & a^{\Delta}\ c\ b^{\Delta} \\
 & & & \mathit{Success} & \Longrightarrow & Ok(a^{\Delta}\ c\ b^{\Delta})
\end{array}
$$

Fig. 3. Examples of paths in $DP_{\{\leftarrow not\ a,\ not\ b;\ \ \leftarrow a,\ not\ c\}}$.

(3) If a state $Ok(L)$ is reachable from the initial state in G, then $L_{|atoms(N)} \in N$;
(4) *Failstate* is reachable from the initial state in G if and only if N is empty.

Proposition 1
For any program Π, the graph DP_{Π} checks the classical models of Π.

Thus, to decide the satisfiability of a program Π it is enough to find a path leading from node $\emptyset$ to a terminal node. If it is *Failstate*, then Π has no classical models. Otherwise, Π has classical models. For instance, let Π_1 be

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

Figure 3 presents two paths in DP_{Π_1} from the node $\emptyset$ to the node $Ok(a^{\Delta}\ c\ b^{\Delta})$. Every edge is annotated on the left by the name of the transition rule that gives rise to this edge in DP_{Π_1}. The node $Ok(a^{\Delta}\ c\ b^{\Delta})$ is terminal. Thus, Proposition 1 asserts that Π_1 is satisfiable and $\{a, c, b\}$ is a classical model of Π_1.

A path in the graph DP_{Π} is a description of a process of search for a classical model of a program Π. 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_{Π} by describing a strategy for choosing a path. A strategy can be based on assigning priorities to transition rules of DP_{Π} so that a solver never applies a rule in a node if a rule with higher priority is applicable to the same node. The DPLL procedure is captured by the priorities ordered as we stated rules in Figure 2. For instance, transition rule *Conclude* has the highest priority. In Figure 3, the path on the left complies with the DPLL priorities. Thus, it corresponds to an execution of the DPLL procedure. The path on the right does not: it uses *Decide* when *Unit* is applicable. The proof of Proposition 1 follows the lines of the proof of Theorem 2.13 in Nieuwenhuis *et al.* (2006)[2].

Abstract Answer Set Solver for Non-disjunctive Programs. Lierler (2011) illustrated that extending DP_Π by a transition rule

$$\textit{Unfounded}: \qquad L \Longrightarrow L\ \neg a \text{ if } \begin{cases} \neg a \text{ does not occur in } L \text{ and} \\ L \text{ is consistent and} \\ \text{there is a set } X \text{ of atoms containing } a \text{ such that} \\ X \text{ is unfounded on } L \text{ w.r.t. } \Pi \end{cases}$$

captures a backtrack-search procedure for finding answer sets of non-disjunctive programs. Many answer set solvers for such programs can be seen as extensions of this procedure (Lierler and Truszczynski 2011).

3 A two-layer abstract solver

The problem of deciding whether a disjunctive program has a stable model is Σ_2^P-complete (Eiter and Gottlob 1993). This translates into the following: (i) there is an exponential number of possible candidate models, and (ii) the problem of deciding whether a candidate model is an answer set of a disjunctive logic program is co-NP-complete. The latter condition differentiates algorithms of answer set solvers for disjunctive programs from the procedures for non-disjunctive programs. Indeed, the problem of deciding whether a candidate model is an answer set of a non-disjunctive program is tractable.

A common architecture of a disjunctive answer set solver is composed of two layers corresponding to the two above conditions: a generate layer and a test layer, each typically based on DPLL-like procedures. In particular,

- the generate layer is used to obtain a set of candidates that are potentially stable models;
- the test layer is used to verify whether a candidate (produced by the generate layer) is a stable model of the given program.

We now proceed to present a graph $DP^2_{g,t}(\Pi)$ that captures such two-layer architecture. It is based on instances of the DPLL procedure for both its generating task and its testing task. We then illustrate how the $DP^2_{g,t}(\Pi)$ transition system can be used to capture the disjunctive answer set solver CMODELS in its basic form.

[2] This work defines a different DPLL graph, avoiding the reference to the transition rule *Success*. The presence of this rule in this presentation is important for the generalizations of the DPLL graph we introduce in the sequel.

3.1 A two-layer abstract solver via DPLL

We start by extending the notion of a basic state to accommodate for generate and test layers. We call symbols $\mathscr{L}$ and $\mathscr{R}$ *labels*. A *state* relative to sets X and X' of atoms is either

(1) a pair $(L, R)_s$, where L and R are records relative to X and X', respectively, and s is a label (either symbol $\mathscr{L}$ or $\mathscr{R}$),
(2) $Ok(L)$, where L is a record relative to X, or
(3) the distinguished state *Failstate*.

We say that a set M of literals *covers* a program Π if $atoms(\Pi) \subseteq atoms(M)$. We say that a function g from a program to another program is a *generating (program)* function if for any program Π, $atoms(\Pi) \subseteq atoms(g(\Pi))$. We call a function from a program Π and a consistent set M of literals covering Π to a non-disjunctive program Π' a *witness (program)* function. Intuitively, a program Π' resulting from a witness function is a *witness (program)* with respect to Π and M. For a program Π and a witness function t, by $atoms(t, \Pi, X)$ we denote the union of $atoms(t(\Pi, L))$ for all possible consistent and complete sets L of literals over X.

We are now ready to define a graph $DP^2_{g,t}(\Pi)$ for a generating function g, a witness function t and a program Π. The set of nodes of $DP^2_{g,t}(\Pi)$ consists of the states relative to sets $atoms(g(\Pi))$ and $atoms(t, \Pi, atoms(g(\Pi)))$. The state $(\emptyset, \emptyset)_{\mathscr{L}}$ is called *initial*. The edges of the graph $DP^2_{g,t}(\Pi)$ are specified by the transition rules presented in Figure 4. The graph $DP^2_{g,t}(\Pi)$ can be used for deciding whether a program $g(\Pi)$ has a classical model M such that the witness $t(\Pi, M)$ is unsatisfiable.

Proposition 2
For any generating function g, any witness function t and any program Π, the graph $DP^2_{g,t}(\Pi)$ checks the classical models M of $g(\Pi)$ such that $t(\Pi, M)$ is unsatisfiable.

Informal Account of the Two-Layer Abstract Solver. Each of the rules of the graph $DP^2_{g,t}(\Pi)$ is placed into one of the three groups *Left*, *Right*, and *Crossing*. The left-rules of $DP^2_{g,t}(\Pi)$ capture the generate layer that applies the DPLL procedure to the program $g(\Pi)$ produced by the generating function. The right-rules of $DP^2_{g,t}(\Pi)$ capture the test layer that applies the DPLL procedure to the computed witness program. The label $\mathscr{L}$ (resp. $\mathscr{R}$) suggests that currently the computation is within the generate (resp. test) layer. The left-hand-side L (resp. right-hand-side R) of the state $(L, R)_{\mathscr{L}}$ records the computation state due to the generate (resp. test) layer. The crossing-rules form a bridge between the two layers.

It turns out that the left-rules no longer apply to a state of the form $(L, R)_{\mathscr{L}}$ only when L is a classical model of $g(\Pi)$. Thus, when a classical model L of $g(\Pi)$ is found, then the $Cross_{\mathscr{L}\mathscr{R}}$ is used and a witness program with respect to L is computed. If no classical model is found for the witness program, then $Conclude_{\mathscr{R}}$ rule applies, which brings us to a terminal state $Ok(L)$, suggesting that L represents a solution to a given search problem. It turns out that no right-rules apply in a state of the form $(L, R)_{\mathscr{R}}$ only when R is a classical model for the witness program. Thus, the set L of literals is not such that $t(\Pi, M)$ is unsatisfiable and the DPLL procedure of the generate layer, embodied by the left-rules, proceeds with the search,

Left-rules

$$\begin{array}{llll}
Conclude_{\mathcal{L}} & (L,\emptyset)_{\mathcal{L}} & \Longrightarrow Failstate & \text{if}\begin{cases} L \text{ is inconsistent and}\\ L \text{ contains no decision literal}\end{cases}\\
Backtrack_{\mathcal{L}} & (Ll^{\Delta}L',\emptyset)_{\mathcal{L}} & \Longrightarrow (L\bar{l},\emptyset)_{\mathcal{L}} & \text{if}\begin{cases} Ll^{\Delta}L' \text{ is inconsistent and}\\ L' \text{ contains no decision literal}\end{cases}\\
Unit_{\mathcal{L}} & (L,\emptyset)_{\mathcal{L}} & \Longrightarrow (Ll,\emptyset)_{\mathcal{L}} & \text{if}\begin{cases} l \text{ is a literal over } atoms(g(\Pi)) \text{ and}\\ l \text{ does not occur in } L \text{ and}\\ \text{a rule in } g(\Pi) \text{ is equivalent to } C \vee l \text{ and}\\ \text{all the literals of } \overline{C} \text{ occur in } L\end{cases}\\
Decide_{\mathcal{L}} & (L,\emptyset)_{\mathcal{L}} & \Longrightarrow (Ll^{\Delta},\emptyset)_{\mathcal{L}} & \text{if}\begin{cases} L \text{ is consistent and}\\ l \text{ is a literal over } atoms(g(\Pi)) \text{ and}\\ \text{neither } l \text{ nor } \bar{l} \text{ occur in } L\end{cases}
\end{array}$$

Crossing-rule $\mathcal{LR}$

$$\begin{array}{llll}
Cross_{\mathcal{LR}} & (L,\emptyset)_{\mathcal{L}} & \Longrightarrow (L,\emptyset)_{\mathcal{R}} & \text{if}\{ \text{ no left-rule applies}
\end{array}$$

Right-rules

$$\begin{array}{llll}
Conclude_{\mathcal{R}} & (L,R)_{\mathcal{R}} & \Longrightarrow Ok(L) & \text{if}\begin{cases} R \text{ is inconsistent and}\\ R \text{ contains no decision literal}\end{cases}\\
Backtrack_{\mathcal{R}} & (L,Rl^{\Delta}R')_{\mathcal{R}} & \Longrightarrow (L,R\bar{l})_{\mathcal{R}} & \text{if}\begin{cases} Rl^{\Delta}R' \text{ is inconsistent and}\\ R' \text{ contains no decision literal}\end{cases}\\
Unit_{\mathcal{R}} & (L,R)_{\mathcal{R}} & \Longrightarrow (L,Rl)_{\mathcal{R}} & \text{if}\begin{cases} l \text{ is a literal over } atoms(t(\Pi,L)) \text{ and}\\ l \text{ does not occur in } R \text{ and}\\ \text{a rule in } t(\Pi,L) \text{ is equivalent to } C \vee l \text{ and}\\ \text{all the literals of } \overline{C} \text{ occur in } L\end{cases}\\
Decide_{\mathcal{R}} & (L,R)_{\mathcal{R}} & \Longrightarrow (L,Rl^{\Delta})_{\mathcal{R}} & \text{if}\begin{cases} R \text{ is consistent and}\\ l \text{ is a literal over } atoms(t(\Pi,L)) \text{ and}\\ \text{neither } l \text{ nor } \bar{l} \text{ occur in } R\end{cases}
\end{array}$$

Crossing-rules $\mathcal{RL}$

$$\begin{array}{llll}
Conclude_{\mathcal{RL}} & (L,R)_{\mathcal{R}} & \Longrightarrow Failstate & \text{if}\begin{cases} \text{no right-rule applies and}\\ L \text{ contains no decision literal}\end{cases}\\
Backtrack_{\mathcal{RL}} & (Ll^{\Delta}L',R)_{\mathcal{R}} & \Longrightarrow (L\bar{l},\emptyset)_{\mathcal{L}} & \text{if}\begin{cases} \text{no right-rule applies and}\\ L' \text{ contains no decision literal}\end{cases}
\end{array}$$

Fig. 4. The transition rules of the graph $DP^2_{g,t}(\Pi)$.

after backtracking through *Backtrack*$_{\mathscr{RL}}$. In the case when *Backtrack*$_{\mathscr{RL}}$ cannot be applied, it follows that no other candidate can be found by the generate layer, so the transition *Conclude*$_{\mathscr{RL}}$ leading to *Failstate* is the only available one from such a state.

3.2 Abstract basic CMODELS

We now relate the graph $DP^2_{g,t}(\Pi)$ to the procedure DP-ASSAT-PROC from Lierler (2005). This procedure forms the basis of the answer set solver CMODELS. Yet, it does not account for backjumping and learning techniques, implemented in CMODELS.

Given a disjunctive program Π, the answer set solver CMODELS starts its computation by computing a CNF formula $g^C(\Pi)$ that corresponds to the clausified program completion of Π. The DPLL procedure is then applied to $g^C(\Pi)$. The test layer of the CMODELS computation relies on the programs produced by a witness program function called t^C that intuitively tests minimality of found models of completion.

To be complete in our presentation, we now review the details of g^C and t^C functions (Lierler 2010). To construct $g^C(\Pi)$, CMODELS introduces an auxiliary atom α_B for every body B occurring in Π. The atom α_B is an explicit definition for B, it is true if and only if B is true. Also every disjunctive rule gives rise to as many auxiliary variables as there are atoms in the head of the rule: for a disjunctive rule $A \leftarrow B$ and every atom $a \in A$, an auxiliary atom $\alpha_{a,B}$ is equivalent to a conjunction $B \wedge \overline{A'}$, where A' is $(A \setminus \{a\})^\vee$. Formulas (3) and (4) present the definitions of g^C and t^C for a program Π. The first four lines of the definition of the CNF formula $g^C(\Pi)$ concern clausification of the introduced explicit definitions, namely α_B and $\alpha_{a,B}$. The last two lines encode clausified completion with the use of α_B and $\alpha_{a,B}$.

$$
\begin{aligned}
g^C(\Pi) = \; & \{\alpha_B \vee \overline{B} \mid B \in Bodies(\Pi)\} \\
& \{\neg\alpha_B \vee a \mid B \in Bodies(\Pi), a \in B\} \\
& \{\alpha_{a,B} \vee \neg\alpha_B \vee A \mid A \vee a \leftarrow B \in \Pi\} \\
& \{\neg\alpha_{a,B} \vee b \mid A \vee a \leftarrow B \in \Pi, b \in \overline{A} \cup \{\alpha_B\}\} \\
& \{\neg\alpha_B \vee A \mid A \leftarrow B \in \Pi\} \\
& \{\neg a \bigvee_{a \leftarrow B \in \Pi} \alpha_B \bigvee_{A \vee a \leftarrow B \in \Pi} \alpha_{a,B}\}
\end{aligned}
\tag{3}
$$

$$
\begin{aligned}
t^C(\Pi, M) = \; & \{\overline{M^+_{|atoms(\Pi)}}^{\vee}\} \cup \\
& \{\neg a \mid \neg a \in M_{|atoms(\Pi)}\} \cup \\
& \{\overline{B} \vee A \mid A \leftarrow B \in \Pi^{M^+}, B \subseteq M\}.
\end{aligned}
\tag{4}
$$

Intuitively, CMODELS uses the program $g^C(\Pi)$ as an approximation of Π during the generate-layer computation. Indeed, any stable model of Π is also a classical model of $g^C(\Pi)$. The converse does not always hold. Thus, classical models of $g^C(\Pi)$ must be checked. For a classical model M of $g^C(\Pi)$, a program produced by $t^C(\Pi, M)$ has no classical models iff M is a stable model of Π. In fact, any model N of $t^C(\Pi, M)$ is such that it satisfies the reduct Π^{M^+}, while $N^+ \subset M^+_{|atoms(\Pi)}$. In such case, $M^+_{|atoms(\Pi)}$ is not an answer set of Π by definition and, consequently, M is not a stable model of Π.

By DP^2_Π, we denote the graph $DP^2_{g^C,t^C}(\Pi)$. Proposition 3 below illustrates that the graph DP^2_Π can be used for deciding whether a given program Π has a stable model, similarly as the graph DP_Π can be used for deciding whether Π has a classical model.

Proposition 3

For any program Π, the graph DP^2_Π checks the stable models of Π.

The graph DP^2_Π captures the search procedure of DP-ASSAT-PROC of CMODELS. The DP-ASSAT-PROC algorithm follows the priorities on its transition rules as they are ordered in Figure 4. We often use this convention when describing other procedures in the sequel.

4 Graph templates

The differences in design choices of disjunctive answer set solvers obscure the understanding of their similarities. In Brochenin *et al.* (2014), transition systems exemplified by the graph DP^2_Π were used to capture several disjunctive solvers, namely, CMODELS, GNT, and DLV implementing backtracking. The transitions systems made the similarities that these solvers share explicit. For example, all solvers are based on a two-layer approach in the spirit of the DP-ASSAT-PROC algorithm. In this work, we make an additional move towards a unifying framework for capturing two-layer methods. We introduce a graph template that we then use to encompass disjunctive solvers CMODELS, GNT, and DLV.

4.1 A single-layer graph template

In the next section, we will define a graph template suitable for capturing two-layer computation of disjunctive answer set solvers. As a step in this direction, we describe here a simpler graph template that can be used to capture the DPLL procedure by encapsulating the DPLL graph. We also show that this template can encapsulate a graph capturing the computation underlying the algorithm of answer set solver SMODELS for non-disjunctive programs.

Template. A function from a program Π and a set of literals over $atoms(\Pi)$ to a set of literals over $atoms(\Pi)$ is called a *propagator condition* or, shortly, *p-condition*. Figure 5 presents four p-conditions, namely, $UnitPropagate$, $AllRulesCancelled$, $BackchainTrue$, and $Unfounded$. For a set $\mathscr{P}$ of p-conditions, a program Π and a set M of literals, by $\mathscr{P}(\Pi, M)$ we denote the set of literals $\bigcup_{p \in \mathscr{P}} p(\Pi, M)$. Intuitively, if each image through a p-condition is a set of possible outcomes, this set represents the union of the possible outcomes through $\mathscr{P}$.

Definition 1
Given a program Π and a set $\mathscr{P}$ of p-conditions, a DPLL *graph template* $DPT_{\mathscr{P},\Pi}$ is a graph of which nodes are the basic states relative to $atoms(\Pi)$ and edges are specified by the transition rules *Conclude*, *Backtrack*, *Decide*, *Success* presented in Figure 2 and the transition rule

$$Propagate\; L \Longrightarrow Ll \quad \text{if} \quad l \in \mathscr{P}(\Pi, L). \qquad (5)$$

For instance, the instantiation $DPT_{\{UnitPropagate\},\Pi}$ of the DPLL graph template results in the DPLL graph DP_Π. Indeed, by definition these graphs share the same nodes as well as their rules *Conclude*, *Backtrack*, *Decide*, and *Success* coincide. Then, one can see that $l \in UnitPropagate(\Pi, L)$ if and only if the transition rule *Unit* in DP_Π is applicable in L and supports the transition to a state Ll, which shows that the *Unit* rule and the *Propagate* rule coincide when $\mathscr{P} = \{UnitPropagate\}$.

$$l \in \mathit{UnitPropagate}(\Pi, L) \quad \text{iff} \begin{cases} l \text{ does not occur in } L \text{ and} \\ \text{a rule in } \Pi \text{ that is equivalent to } C \vee l \text{ and} \\ \text{all the literals of } \overline{C} \text{ occur in } L \end{cases}$$

$$\neg a \in \mathit{AllRulesCancelled}(\Pi, L) \quad \text{iff} \begin{cases} \neg a \text{ does not occur in } L \text{ and} \\ \text{there is no rule in } \Pi \text{ supporting } a \text{ with respect to } L \end{cases}$$

$$l \in \mathit{BackchainTrue}(\Pi, L) \quad \text{iff} \begin{cases} l \text{ does not occur in } L \text{ and} \\ \text{there is a rule } A \vee a \leftarrow B \text{ in } \Pi \\ \text{so that (i) } a \in L, \text{and (ii) either } \bar{l} \in A \text{ or } l \in B \text{ and,} \\ \text{(iii) no other rule in } \Pi \text{ is supporting } a \text{ with respect to } L \end{cases}$$

$$\neg a \in \mathit{Unfounded}(\Pi, L) \quad \text{iff} \begin{cases} \neg a \text{ does not occur in } L \text{ and} \\ L \text{ is consistent and} \\ \text{there is a set } X \text{ of atoms containing } a \text{ such that} \\ X \text{ is unfounded on } L \text{ with respect to } \Pi \end{cases}$$

Fig. 5. Propagator conditions.

Instantiation. We call *types* the elements of the set $T = \{cla, sup, sta\}$. In the following, by *cla*-model, *sup*-model, and *sta*-model, we denote classical, supported, and stable models, respectively. We also use letter w to denote a variable over set T of types. We say that a set $\mathscr{P}$ of p-conditions is *w-sound* if for any program Π, for any set M of literals, and for any w-model M_1 of Π such that $M \subseteq M_1$, it also holds that $\mathscr{P}(\Pi, M) \subseteq M_1$. Note that any *cla*-sound set of p-conditions is *sup*-sound, and any *sup*-sound set of p-conditions is *sta*-sound. We say that a set $\mathscr{P}$ of p-conditions is *w-complete* when for any program Π and any consistent and complete set M of literals over $atoms(\Pi)$, set M is a w-model of Π if and only if $\mathscr{P}(\Pi, M) = \emptyset$. For a type w, we say that a set $\mathscr{P}$ of p-conditions is *w-enforcing* if $\mathscr{P}$ is both w-sound and w-complete.

Next theorem summarizes properties of several sets of p-conditions:

$$\begin{aligned} up &= \{UnitPropagate\} \\ sd &= \{UnitPropagate, AllRulesCancelled, BackchainTrue\} \\ sm &= \{UnitPropagate, AllRulesCancelled, BackchainTrue, Unfounded\} \end{aligned}$$

Theorem 2
The following statements hold:

(1) the set *up* is *cla*-enforcing;
(2) all the subsets of *sd* that contain $\{UnitPropagate, AllRulesCancelled\}$ are *sup*-enforcing; and
(3) all the subsets of *sm* that contain $\{UnitPropagate, Unfounded\}$ are *sta*–enforcing.

We are now ready to state the main result of this section.

Theorem 3
For any program Π, any type w, and any w-enforcing set of p-conditions $\mathscr{P}$, the graph $DPT_{\mathscr{P},\Pi}$ checks the w-models of Π.

$$\begin{array}{llll} Propagate_{\mathcal{L}} & (L,\emptyset)_{\mathcal{L}} & \Longrightarrow (Ll,\emptyset)_{\mathcal{L}} & \text{if } l \in \mathcal{P}_{\mathcal{L}}(g(\Pi), L) \\ Propagate_{\mathcal{R}} & (L,R)_{\mathcal{R}} & \Longrightarrow (L,Rl)_{\mathcal{R}} & \text{if } l \in \mathcal{P}_{\mathcal{R}}(t(\Pi, L), R) \end{array}$$

Fig. 6. Transition rules of the graph template $STT^{\mathscr{P}_{\mathscr{L}},g}_{\mathscr{P}_{\mathscr{R}},t}(\Pi)$.

Theorems 2 and 3 give rise to families of valid solvers for deciding where classical, supported, or stable models exist for a program. For instance, for a non-disjunctive program Π, the graph $DPT_{sm,\Pi}$ coincides with the graph $\textsc{sm}_{\Pi}$ (Lierler 2011) that captures computation of answer set solver SMODELS (Simons *et al.* 2002). The graph $DPT_{sd,\Pi}$ coincides with the graph $\textsc{atleast}_{\Pi}$ (Lierler 2011) that provides a procedure for deciding whether a non-disjunctive program has supported models. For a disjunctive program Π, the same single-layer graph $DPT_{sm,\Pi}$ forms a procedure for deciding whether Π has a stable model. Note, however, that generally the problem of deciding whether $l \in Unfounded(\Pi, L)$ is NP-complete for the case when Π is disjunctive.

4.2 A two-layer graph template

We extend here the approach of Section 4.1 to capture two-layer methodology of disjunctive solvers.

Definition 2
Given a program Π, sets $\mathscr{P}_{\mathscr{L}}$ and $\mathscr{P}_{\mathscr{R}}$ of p-conditions, a generating function g, and a witness function t, a *two-layer template graph* $STT^{\mathscr{P}_{\mathscr{L}},g}_{\mathscr{P}_{\mathscr{R}},t}(\Pi)$ is a graph defined as follows:

- the set of nodes is, as in the previous two-layer graphs, the set of states relative to $atoms(g(\Pi))$ and $atoms(t, \Pi, atoms(g(\Pi)))$; and
- the transition rules are the rules presented in Figure 4 except the rules $Unit_{\mathscr{L}}$ and $Unit_{\mathscr{R}}$, that are replaced by the rules $Propagate_{\mathscr{L}}$ and $Propagate_{\mathscr{R}}$ presented in Figure 6.

Description of the Template. We call the state $(\emptyset,\emptyset)_{\mathscr{L}}$ *initial.* Note how the rules $Propagate_{\mathscr{L}}$ and $Propagate_{\mathscr{R}}$ in $STT^{\mathscr{P}_{\mathscr{L}},g}_{\mathscr{P}_{\mathscr{R}},t}(\Pi)$ refer to the parameters $\mathscr{P}_{\mathscr{L}}$, $\mathscr{P}_{\mathscr{R}}$, g, and t of the graph template. Varying these parameters will allow us to specify transition systems that capture different disjunctive answer set solvers. Intuitively, the parameters $\mathscr{P}_{\mathscr{L}}$ and $\mathscr{P}_{\mathscr{R}}$ are sets of p-conditions defining a propagation rule on generate and test side of computation, respectively.

The instantiation $STT^{up,g^C}_{up,t^C}(\Pi)$ of the two-layer graph template results in DP^2_{Π}. Indeed, the graphs share the same nodes. Also their rules $Conclude_{\mathscr{L}}$, $Conclude_{\mathscr{R}}$, $Decide_{\mathscr{L}}$, $Decide_{\mathscr{R}}$, $Backtrack_{\mathscr{L}}$, $Backtrack_{\mathscr{R}}$, and $Conclude_{\mathscr{R}\mathscr{L}}$ coincide. It is easy to see that a literal l is in $up(g^C(\Pi), L)$ if and only if the transition rule $Unit_{\mathscr{L}}$ in DP^2_{Π} is applicable in $(L,\emptyset)_{\mathscr{L}}$ and supports the transition to a state $(Ll,\emptyset)_{\mathscr{L}}$. Thus, the transition rule $Propagate_{\mathscr{L}}$ supports the transition from $(L,\emptyset)_{\mathscr{L}}$ to $(Ll,\emptyset)_{\mathscr{L}}$ if and only if the transition rule $Unit_{\mathscr{L}}$ supports the same transition. A similar statement holds for the case of $Propagate_{\mathscr{R}}$ and $Unit_{\mathscr{R}}$.

Recall that in Section 3.2 we showed that CMODELS implementing backtracking can be defined using the graph DP^2_Π. The fact that instantiation $STT^{up,g^C}_{up,t^C}(\Pi)$ coincides with DP^2_Π illustrates that the introduced template is sufficient for capturing existing solvers. Next section demonstrates that the proposed template is suitable for capturing GNT and DLV.

Instantiation: Approximating and Ensuring Pairs. In the definition of the two-layer template graph $STT^{\mathscr{P}_\mathscr{L},g}_{\mathscr{P}_\mathscr{R},t}(\Pi)$, we pose no restrictions on its four key parameters: sets $\mathscr{P}_\mathscr{L}$, $\mathscr{P}_\mathscr{R}$ of p-conditions, and generating and witness functions g, t. In practice, when this template is utilized to model, characterize, and elicit disjunctive solvers these four parameters exhibit specific properties. We now introduce terminology that allows us to specify essential properties of these parameters that will translate into correctness of solvers captured by properly instantiated template. On the one hand, we introduce the conditions on generating and witness functions under which we call these functions "approximating" and "ensuring", respectively. On the other hand, we couple these conditions with restrictions on sets of p-conditions so that we can speak of (i) approximating-pair $(\mathscr{P}_g, g)$ for a set $\mathscr{P}_g$ of p-conditions and a generating function g, and (ii) ensuring-pair $(\mathscr{P}_t, t)$ for a set $\mathscr{P}_t$ of p-conditions and a witness function t. For such pairs, the template instantiation $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$ results in a graph that checks stable models of Π. As a result, when we characterize such solvers as GNT and DLV by means of the two-layer template we focus on (i) specifying their generating and witness function as well as their sets of p-conditions, and (ii) illustrating that they form proper approximating and ensuring pairs. This also brings us to the realization that an inception of a novel solver can be triggered by a creation of a novel approximation and ensuring pairs or their combinations. We now make these ideas precise.

For types w and w_1, we say that a generating function g is w_1-approximating with respect to type w if for any program Π:

(1) for any stable model L of Π there is a w_1-model L_1 of $g(\Pi)$ such that $L = L_{1|atoms(\Pi)}$; and
(2) for any w_1-model M of $g(\Pi)$, $M_{|atoms(\Pi)}$ is a w-model of Π.

Consider the generating function $cnfcomp(\Pi)$ that returns a CNF formula, which stands for the completion $comp(\Pi)$ converted to CNF using straightforward equivalent transformations. In other words, $cnfcomp(\Pi)$ consists of clauses of two kinds

(1) the rules $A \leftarrow B$ of the program written as clauses $A \vee \overline{B}$, and
(2) formulas of $cnfcomp(\Pi)$ from (2) converted to CNF using the distributivity of disjunction over conjunction.[3]

The function $cnfcomp$ is cla-approximating with respect to sup. Indeed,

(1) any stable model of a program Π is also a cla-model of $cnfcomp(\Pi)$, and
(2) any cla-model of $cnfcomp(\Pi)$ is a sup-model of Π.

[3] It is essential that repetitions are not removed in the process of clausification. For instance, $cnfcomp(a \leftarrow not\ a) = (a \vee a) \wedge (\neg a \vee \neg a)$.

Since any supported model is also a classical model, the *cnfcomp* function is also *cla*-approximating with respect to *cla*. Note that when a generating function g is w_1-approximating with respect to w, then enumerating all w_1-models of $g(\Pi)$ results in enumerating some w-models of Π modulo a restriction to $atoms(\Pi)$.

For types w and w_1, and a witness function t, we say that t is w_1-*ensuring* with respect to w when for any set M of literals covering Π such that $M_{|atoms(\Pi)}$ is w-model of Π, $M_{|atoms(\Pi)}$ is a stable model of Π if and only if $t(\Pi, M)$ results in a program that has no w_1-model.

For instance, the witness function t^C is *cla*-ensuring with respect to *cla*. Since any *sup*-model is also a *cla*-model, the function t^C is also *cla*-ensuring with respect to *sup*. It is easy to see that when a witness function t is w_1-ensuring with respect to w, then given any w-model L of a program Π we may use the function t to test that L is also a stable model of Π. Indeed, an application of t resulting in a program that has no w_1-models translates into the statement that L is a stable model of Π.

These newly defined concepts of approximating and ensuring functions provide the following characterization for the set of stable models of a program Π.

Proposition 4
For any types w, w_1 and w_2, generating function g that is w_1-approximating with respect to w, witness function t that is w_2-ensuring with respect to w, and program Π, the set of all stable models of Π is

$$\{L_{|atoms(\Pi)} \mid L \text{ is a } w_1\text{-model of } g(\Pi) \text{ and } t(\Pi, L) \text{ has no } w_2\text{-models}\}.$$

We now introduce the notion of ensuring and approximating pairs that permit an operational use of generating and witness functions, by matching them with a relevant set of propagators. We call a pair $(\mathscr{P}, g)$ of a set of p-conditions and a generating function an *approximating-pair* with respect to w if for some type w_1, the set $\mathscr{P}$ is w_1-enforcing and the function g is w_1-approximating with respect to w. For example, the pair $(up, cnfcomp)$ is an approximating-pair with respect to *sup* as well as to *cla*. The (up, g^C) is also an approximating-pair with respect to *sup* as well as to *cla*.

We call a pair $(\mathscr{P}, t)$ of a set of p-conditions and a witness function an *ensuring-pair* with respect to w if for some type w_1, the set $\mathscr{P}$ is w_1-enforcing and the function t is w_1-ensuring with respect to w. For example, the pair (up, t^C) is an ensuring-pair with respect to any defined type.

We are now ready to state the main result of this section.

Theorem 4
For any program Π, any type w, any $(\mathscr{P}_g, g)$ approximating-pair with respect to w, and any $(\mathscr{P}_t, t)$ ensuring-pair with respect to w, the graph $STT^{\mathscr{P}_g, g}_{\mathscr{P}_t, t}(\Pi)$ checks the stable models of Π.

Theorem 4 illustrates how the template $STT^{\mathscr{P}_g, g}_{\mathscr{P}_t, t}(\Pi)$ can serve as a framework for defining transitions systems that result in correct algorithms for deciding whether a program Π has a stable model. The facts that (up, g^C) is an approximating-pair with respect to cla and that (up, t^C) is an ensuring-pair with respect to cla, together with Theorem 4, subsume the result of Proposition 3.

We now state propositions that capture interesting properties about states of the graph $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$. The former proposition concerns states with the label $\mathscr{L}$, the latter concerns states with the label $\mathscr{R}$.

Proposition 5
For any type w, generating function g, witness function t, w-enforcing set of p-conditions $\mathscr{P}_g$, set of p-conditions $\mathscr{P}_t$, and program Π, if no left-rule is applicable in some state $(l_1.\cdots.l_{k_1}, r_1.\cdots.r_{k_2})_{\mathscr{L}}$ in $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$ reachable from the initial state, then $l_1.\cdots.l_{k_1}$ is a w-model of $g(\Pi)$.

Proposition 6
For any types w_1 and w_2, generating function g witness function t, w_1-enforcing set of p-conditions $\mathscr{P}_g$, w_2-enforcing set of p-conditions $\mathscr{P}_t$, program Π, and a state $(l_1.\cdots.l_{k_1}, r_1.\cdots.r_{k_2})_{\mathscr{R}}$ in $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$ reachable from the initial state, the following conditions hold:

(a) $t(\Pi, l_1.\cdots.l_{k_1})$ is defined,
(b) $r_1.\cdots.r_{k_2}$ is a set of literals over $t(\Pi, L)$,
(c) $l_1.\cdots.l_{k_1}$ is a w_1-model of $g(\Pi)$, and
(d) If no right-rule is applicable to $(l_1.\cdots.l_{k_1}, r_1.\cdots.r_{k_2})_{\mathscr{R}}$ then $r_1.\cdots.r_{k_2}$ is a w_2-model of $t(\Pi, l_1.\cdots.l_{k_1})$.

5 Applications of the template

Section 3.2 illustrates how CMODELS implementing backtracking can be defined using the graph DP^2_Π, while the previous section states that the instantiation $STT^{up,g^C}_{up,t^C}(\Pi)$ of the two-layer graph template results in DP^2_Π. Thus, this template is suitable for capturing computations of CMODELS. In this section, we show how the template also captures the solvers GNT and DLV without backjumping. Then, we discuss how the framework facilitates the design of new abstract solvers and their comparison, by means of inspecting the structures of the related graphs.

Abstract GNT. We now show how the procedure underlying disjunctive solver GNT can be captured by the two-layer template. Unlike solver CMODELS that uses the DPLL procedure for generating and testing, system GNT uses the SMODELS procedure for respective tasks. Recall that the SMODELS procedure finds stable models for non-disjunctive logic programs, while the DPLL procedure finds classical models. The graph SM_Π (Section 4.1) captures the computation underlying SMODELS just as the graph DP_Π captures the computation underlying DPLL. It forms a basis for devising the transition system suitable to describe GNT. The graph describing the general structure of GNT is obtained from the graph template $STT^{sm,g}_{sm,t}(\Pi)$ that rely on the set sm of p-contitions.[4]

[4] The graph template $STT^{sm,g}_{sm,t}(\Pi)$ corresponds to the graph $SM^2_{g(\Pi),t}$ defined in Brochenin *et al.* (2014).

Janhunen *et al.* (2006) define the generating function g^G and the witness function t^G used in GNT. We present these definitions in (6) and (7).[5] For a disjunctive program Π, by Π_N we denote the set of non-disjunctive rules of Π, by Π_D we denote the set of disjunctive rules $\Pi \setminus \Pi_N$. For each atom a in $atoms(\Pi)$, let a^r and a^s be new atoms.

$$
\begin{aligned}
g^G(\Pi) = {} & \{a \leftarrow B, not\ a^r \mid A \vee a \leftarrow B \in \Pi_D\} \cup \\
& \{a^r \leftarrow not\ a \mid A \vee a \leftarrow B \in \Pi_D\} \cup \\
& \{\leftarrow \overline{A}, B \mid A \leftarrow B \in \Pi_D\} \cup \\
& \Pi_N \cup \\
& \{a^s \leftarrow \overline{A \setminus \{a\}}, B \mid A \vee a \leftarrow B \in \Pi_D\} \cup \\
& \{\leftarrow a, not\ a^s \mid a \vee A \leftarrow B \in \Pi_D\}
\end{aligned} \tag{6}
$$

$$
\begin{aligned}
t^G(\Pi, M) = {} & \{a \leftarrow B, not\ a^r \mid A \vee a \leftarrow B \in \Pi_D^M, a \in M, B \subseteq M\} \cup \\
& \{a^r \leftarrow not\ a \mid A \vee a \leftarrow B \in \Pi\} \cup \\
& \{\leftarrow \overline{A}, B \mid A \leftarrow B \in \Pi_D^M, B \subseteq M\} \cup \\
& \{a \leftarrow B \mid a \leftarrow B \in \Pi_N^M, a \in M, B \subseteq M\} \cup \\
& \{\leftarrow M_{|atoms(\Pi)}\}
\end{aligned} \tag{7}
$$

By SM_Π^2, we denote the graph $STT_{sm,t^G}^{sm,g^G}(\Pi)$. The graph SM_Π^2 captures the GNT procedure by Janhunen *et al.* (2006) in a similar way as the graph DP_Π^2 captures the CMODELS procedure of DP-ASSAT-PROC in Section 3.2. Figure 7 presents an example of a path in a graph $SM^2_{\{a\leftarrow c; b\leftarrow c; c\leftarrow a,b; a\vee b\leftarrow\}}$. From the formal results by Janhunen *et al.* (2006), it immediately follows that g^G is *sta*-approximating with respect to *cla* and t^G is *sta*-ensuring with respect to *cla*. The pair (sm, g^G) is an approximating-pair with respect to *cla*, while (sm, t^G) is an ensuring-pair with respect to *cla*. The following result immediately follows from Theorem 4.[6]

Corollary 1
For any Π, the graph SM_Π^2 checks the stable models of Π.

Abstract DLV *Without Backjumping*. This section introduces graphs that capture the answer set solver DLV without backjumping. The generate layer, i.e., the left-rule layer, is reminiscent to the SMODELS algorithm except it does not use *Unfounded*. The test layer applies the DPLL procedure to a witness formula.

The graph templates $STT_{up,t}^{sd,g}(\Pi)$ describe the general structure of DLV. The generating function g^D is the identity function as in (8), and the witness function t^D follows in (9).

$$g^D(\Pi) = \Pi \tag{8}$$

$$
\begin{aligned}
t^D(\Pi, M) = {} & \{(B \cap M^+)^\vee \vee \overline{A'}^\vee \mid A \leftarrow B \in \Pi^{M^+}, B \subseteq M, A' = A \cap M^+\} \cup \\
& \{(M_{|atoms(\Pi)})^\vee\}
\end{aligned} \tag{9}
$$

[5] The presented functions g^G and t^G capture the essence of functions *Gen* and *Test* defined by Janhunen et al., but they are not identical. Our language of disjunctive programs includes rules with empty heads. This allows us a more concise description.

[6] Corollary 1 corresponds to Theorem 5 in Brochenin *et al.* (2014).

$$
\begin{array}{ll}
g^G(\Pi) = & a \leftarrow c \\
& b \leftarrow c \\
& c \leftarrow a, b \\
& a \leftarrow \mathit{not}\ a^r \\
& b \leftarrow \mathit{not}\ b^r \\
& a^r \leftarrow \mathit{not}\ a \\
& b^r \leftarrow \mathit{not}\ b \\
& \leftarrow \mathit{not}\ a, \mathit{not}\ b \\
& a^s \leftarrow c \\
& a^s \leftarrow \mathit{not}\ b \\
& b^s \leftarrow c \\
& b^s \leftarrow \mathit{not}\ a \\
& \leftarrow a, \mathit{not}\ a^s \\
& \leftarrow b, \mathit{not}\ b^s
\end{array}
$$

$$
\begin{array}{ll}
t^G(\Pi, L) = & a \leftarrow \mathit{not}\ a^r \\
& a^r \leftarrow \mathit{not}\ a \\
& \leftarrow \mathit{not}\ a, \mathit{not}\ b \\
& \leftarrow a, \mathit{not}\ b, \mathit{not}\ c
\end{array}
$$

$$
\begin{array}{ll}
\text{Initial state}: & (\emptyset, \emptyset)_{\mathcal{L}} \\
\mathit{Decide}_{\mathcal{L}}: & ((\neg a^r)^{\Delta}, \emptyset)_{\mathcal{L}} \\
\mathit{Propagate}^2_{\mathcal{L}}: & ((\neg a^r)^{\Delta}\ a\ a^s, \emptyset)_{\mathcal{L}} \\
& \quad (\text{with } \mathit{UnitPropagate}) \\
\mathit{Decide}_{\mathcal{L}}: & ((\neg a^r)^{\Delta}\ a\ a^s\ \neg b^{\Delta}, \emptyset)_{\mathcal{L}} \\
\mathit{Propagate}_{\mathcal{L}}: & ((\neg a^r)^{\Delta}\ a\ a^s\ \neg b^{\Delta}\ b^r, \emptyset)_{\mathcal{L}} \\
& \quad (\text{with } \mathit{UnitPropagate}) \\
\mathit{Propagate}_{\mathcal{L}}: & ((\neg a^r)^{\Delta}\ a\ a^s\ \neg b^{\Delta}\ b^r\ \neg c, \emptyset)_{\mathcal{L}} \\
& \quad (\text{with } \mathit{Unfounded}) \\
\mathit{Decide}_{\mathcal{L}}: & ((\neg a^r)^{\Delta}\ a\ a^s\ \neg b^{\Delta}\ b^r\ \neg c\ b^s, \emptyset)_{\mathcal{L}} \\
\mathit{Cross}_{\mathcal{L}\mathcal{R}}: & ((\neg a^r)^{\Delta}\ a\ a^s\ \neg b^{\Delta}\ b^r\ \neg c, \emptyset)_{\mathcal{R}}
\end{array}
$$

Let $L = (\neg a^r)^{\Delta}\ a\ a^s\ \neg b^{\Delta}\ b^r\ \neg c$

$$
\begin{array}{ll}
\text{Current state}: & (L, \emptyset)_{\mathcal{R}} \\
\mathit{Decide}_{\mathcal{R}}: & (L, \neg a^{\Delta})_{\mathcal{R}} \\
\mathit{Propagate}_{\mathcal{R}}: & (L, \neg a^{\Delta}\ b)_{\mathcal{R}} \\
& \quad (\text{with } \mathit{UnitPropagate}) \\
\mathit{Propagate}_{\mathcal{R}}: & (L, \neg a^{\Delta}\ b\ \neg b)_{\mathcal{R}} \\
& \quad (\text{with } \mathit{AllRulesCancelled}) \\
\mathit{Backtrack}_{\mathcal{R}}: & (L, a)_{\mathcal{R}} \\
\mathit{Propagate}_{\mathcal{R}}: & (L, a\ \neg a^r)_{\mathcal{R}} \\
& \quad (\text{with } \mathit{BackchainTrue}) \\
\mathit{Propagate}^2_{\mathcal{R}}: & (L, a\ \neg a^r\ \neg b\ \neg c)_{\mathcal{R}} \\
& \quad (\text{with } \mathit{AllRulesCancelled}) \\
\mathit{Propagate}_{\mathcal{R}}: & (L, a\ \neg a^r\ \neg b\ \neg c\ c)_{\mathcal{R}} \\
& \quad (\text{with } \mathit{UnitPropagate}) \\
\mathit{Conclude}_{\mathcal{R}}: & \mathit{Ok}(L)
\end{array}
$$

Fig. 7. Example of path through the graph $SM^2_{\{a\leftarrow c; b\leftarrow c; c\leftarrow a,b; a\vee b\leftarrow\}}$.

Following the results from Faber (2002) and Koch *et al.* (2003), the generating function g^D is *sup*-approximating with respect to *cla* while the witness function t^D is *cla*-ensuring with respect to *cla*. The pair (sd, g^D) is an approximating-pair with respect to *cla*, while (up, t^D) is an ensuring-pair with respect to *cla*. The result below immediately follows from Theorem 4.[7]

Corollary 2
For any Π, the graph $STT^{sd,g^D}_{up,t^D}(\Pi)$ checks the stable models of Π.

This corollary is an alternative proof of correctness for the DLV algorithm previously stated by Faber (2002) and Koch *et al.* (2003) in terms of pseudo-code. Figure 8 presents an example of a path through one of the graph describing abstract DLV.

Designing New Graphs and Comparing Graphs. The two-layer graph template can be conveniently used to define new abstract solvers. For instance, one may choose to combine (up, g^C) with (sm, t^G) to obtain a solver captured by the graph template $STT^{up,g^C}_{sm,t^G}(\Pi)$. Theorem 4 provides a proof of correctness for the procedure summarized by this family of graphs. More generally, to obtain a new solver one can combine any approximating-pair on the left side of the graphs with any ensuring-pair on the right side with respect to the same type. For instance, for any pair $(\mathscr{P}, t)$

[7] Corollary 2 corresponds to Theorem 6 in Brochenin *et al.* (2014).

$$\begin{array}{ll}
g^D(\Pi) = & a \leftarrow c \\
 & b \leftarrow c \\
 & c \leftarrow a, b \\
 & a \vee b \leftarrow \\
 & \\
t^D(\Pi, L) = & \neg a \vee c \\
 & \neg b \vee c \\
 & \neg c \vee a \vee b \\
 & \neg a \vee \neg b \\
 & c \vee a \vee b \\
 & \\
t^D(\Pi, L') = & \neg a \\
 & \neg b \\
 & \neg c \vee a \\
 & \neg a \vee \neg b \\
 & a
\end{array}$$

$$\begin{array}{ll}
\text{Initial state}: & (\emptyset, \emptyset)_{\mathcal{L}} \\
\mathit{Decide}_{\mathcal{L}}: & (c^{\Delta}, \emptyset)_{\mathcal{L}} \\
\mathit{Propagate}^2_{\mathcal{L}}: & (c^{\Delta}\ a\ b, \emptyset)_{\mathcal{L}} \\
 & \quad (\text{with } \mathit{UnitPropagate}) \\
\mathit{Cross}_{\mathcal{L}\mathcal{R}}: & (c^{\Delta}\ a\ b, \emptyset)_{\mathcal{R}}
\end{array}$$

Let $L = c^{\Delta}\ a\ b$

$$\begin{array}{ll}
\text{Current state}: & (L, \emptyset)_{\mathcal{R}} \\
\mathit{Decide}_{\mathcal{R}}: & (L, c^{\Delta})_{\mathcal{R}} \\
\mathit{Propagate}^2_{\mathcal{R}}: & (L, c^{\Delta}\ a\ b)_{\mathcal{R}} \\
\mathit{Backtrack}_{\mathcal{R}\mathcal{L}}: & (\neg c, \emptyset)_{\mathcal{L}} \\
\mathit{Decide}_{\mathcal{L}}: & (\neg c\ a^{\Delta}, \emptyset)_{\mathcal{L}} \\
\mathit{Propagate}_{\mathcal{L}}: & (\neg c\ a^{\Delta}\ \neg b, \emptyset)_{\mathcal{L}} \\
 & \quad (\text{with } \mathit{UnitPropagate}) \\
\mathit{Cross}_{\mathcal{L}\mathcal{R}}: & (\neg c\ a^{\Delta}\ \neg b, \emptyset)_{\mathcal{R}}
\end{array}$$

Let $L' = \neg c\ a^{\Delta}\ \neg b$

$$\begin{array}{ll}
\text{Current state}: & (L', \emptyset)_{\mathcal{R}} \\
\mathit{Propagate}^2_{\mathcal{R}}: & (L', \neg a\ a)_{\mathcal{R}} \\
\mathit{Conclude}_{\mathcal{R}}: & \mathit{Ok}(L')
\end{array}$$

Fig. 8. Example of path through the graph $STT^{sd,g^D}_{up,t^D}(\{a \leftarrow c; b \leftarrow c; c \leftarrow a, b; a \vee b \leftarrow\})$.

that is ensuring with respect to *cla*, the family of graphs $STT^{up,cnfcomp}_{\mathscr{P},t}(\Pi)$ captures a correct procedure for a disjunctive answer set solver.

In the following, we illustrate how abstract solvers can serve also as a convenient tool for comparing search procedures from an abstract point of view, by means of comparison to the related graphs. In this respect, we now state the result that illustrates a strong relation between CMODELS and DLV. Indeed, their generate layer:

Theorem 5
For any $(\mathscr{P}, t)$ ensuring-pair with respect to *cla*, and any program Π, the graphs $STT^{up,cnfcomp}_{\mathscr{P},t}(\Pi)$ and $STT^{sd,g^D}_{\mathscr{P},t}(\Pi)$ are identical graphs.

6 Proofs

6.1 Proof of Theorem 2

We start by stating several lemmas that will be instrumental in constructing arguments for Theorem 2. Recall that $up = \{\mathit{UnitPropagate}\}$.

Lemma 1
The set *up* is *cla*-complete.

In other words, for any program Π and any complete and consistent set M of literals over $atoms(\Pi)$, the set M is a *cla*-model of Π iff $\mathit{UnitPropagate}(\Pi, M) = \emptyset$.

Proof
Left-to-right: Let M be a *cla*-model of Π. Our proof is by contradiction. Assume that $\mathit{UnitPropagate}(\Pi, M) \neq \emptyset$. Take any literal l from this set. Then, the literal l is

such that it does not belong to M. Also, there is a rule in Π that is equivalent to a clause $C \vee l$ so that all the literals of $\overline{C}$ occur in M. Since M is a *cla*-model of Π, we conclude that $l \in M$. We derive a contradiction.

Right-to-left: Let $UnitPropagate(\Pi, M) = \emptyset$. By contradiction. Assume that M is not a *cla*-model of Π. Then there is a rule in Π that is equivalent to a clause $C \vee l$ so that all the literals of $\overline{C}$ as well as $\overline{l}$ occur in M (indeed, M is a complete set of literals over $atoms(\Pi)$ that does not satisfy some rule in Π). Since M is consistent, $l \notin M$. It follows that $l \in UnitPropagate(\Pi, M)$. We derive a contradiction. □

Lemma 2

For any program Π, any atom a, and any sets M and M' of literals such that $M \subseteq M'$, if a rule in Π is not a supporting rule for a with respect to M then this rule is also not a supporting rule for a with respect to M'.

Proof

By contradiction. Assume that there is a rule $A \vee a \leftarrow B$ in Π such that it is not a supporting rule for a with respect to M but it is a supporting rule for a with respect to M'. It follows that $M \cap (\overline{B} \cup A) \neq \emptyset$, while $M' \cap (\overline{B} \cup A) = \emptyset$. This contradicts the fact that $M \subseteq M'$. □

We now generalize Lemma 4 from Lierler (2008) to the case of disjunctive programs.

Lemma 3

For any unfounded set U on a consistent set L of literals w.r.t. a program Π and any consistent and complete set M of literals over $atoms(\Pi)$, if $L \subseteq M$ and $M \cap U \neq \emptyset$, then M is not a stable model of Π.

Proof

By contradiction. Assume that M is a stable model of Π. Then, M is a classic model of Π also. By Theorem 1, M is such that there is no non-empty subset of M^+ such that it is an unfounded set on M w.r.t. Π. Since $M \cap U \neq \emptyset$, it follows that $M \cap U$ is not an unfounded set on M w.r.t. Π. It follows that for some rule $a \vee A \leftarrow B \in \Pi$ such that $a \in M \cap U$ all of the following conditions hold

(1) $M \cap \overline{B} = \emptyset$,
(2) $M \cap U \cap B = \emptyset$, and
(3) $(A \setminus (M \cap U)) \cap M = \emptyset$.

Since $M \cap \overline{B} = \emptyset$ and $L \subseteq M$ it follows that $L \cap \overline{B} = \emptyset$. Since $M \cap \overline{B} = \emptyset$ and the fact that M is consistent and complete set of literals over $atoms(\Pi)$, $B^+ \subseteq M$. Consequently, $U \cap B^+ = M \cap U \cap B^+ = \emptyset$. Since $L \subseteq M$ and $(A \setminus (M \cap U)) \cap M = \emptyset$, it follows that $(A \setminus U) \cap L = \emptyset$. Consequently, the set U is not an unfounded set on L. □

We are now ready to introduce the proof of Theorem 2.

Proof of Theorem 2

Statement 1. We have to show that the set *up* is *cla*-enforcing. Lemma 1 states that the set *up* is *cla*-complete. Thus, we only ought to illustrate that *up* is *cla*-sound. Let Π be any program, M be any set of literals, M' be any *cla*-model of Π such that $M \subseteq M'$. We have to show that $up(\Pi, M) \subseteq M'$. Let l be any literal in $up(\Pi, M)$. We now show that $l \in M'$. The p-condition *UnitPropagate* is the only member of the set *up*. Thus, $up(\Pi, M) = UnitPropagate(\Pi, M)$. It follows that $l \in UnitPropagate(\Pi, M)$. By the conditions of *UnitPropagate* definition, there is a rule in Π that is equivalent to a clause $C \vee l$ so that all the literals of $\overline{C}$ occur in M. Since $M \subseteq M'$, it follows that all the literals of $\overline{C}$ occur in M'. From the fact that M' is *cla*-model of Π, it follows that $M' \models C \vee l$. Consequently, $l \in M'$.

Statement 2. We have to show that the subsets of *sd* containing $\{UnitPropagate, AllRulesCancelled\}$ are *sup*-enforcing. We first illustrate this property for the set $\{UnitPropagate, AllRulesCancelled\}$. We call this set *ua*. We start by showing that the set *ua* is *sup*-sound. Let Π be any program, M be any set of literals, M' be any *sup*-model of Π such that $M \subseteq M'$. We have to illustrate that the set $ua(\Pi, M)$ is a subset of M'. Consider any literal l in the set $ua(\Pi, M)$. We now show that l is also in M'.

Case 1. $l \in UnitPropagate(\Pi, M)$. Since M' is a *sup*-model, M' is also a *cla*-model. The rest of the argument follows the lines of proof in *Statement 1*, which illustrates that *up* is *cla*-sound.

Case 2. $l \in AllRulesCancelled(\Pi, M)$. l has the form $\neg a$. By the conditions of *AllRulesCancelled* definition, it follows that there is no rule in Π supporting a with respect to M. By Lemma 2, we derive that there is no rule in Π supporting a with respect to M'. From the fact that M' is *sup*-model of Π, it follows that $\neg a \in M'$. (Indeed, a may not be a member of M', while M' is a complete set of literals over $atoms(\Pi)$.)

Second, we show that the set *ua* is *sup*-complete. Let Π be any program, M be any complete and consistent set of literals over $atoms(\Pi)$. We now show that M is *sup*-model of Π iff $ua(\Pi, M) = \emptyset$.

Left-to-right: Let M be a *sup*-model of Π. By contradiction. Assume that the set $ua(\Pi, M)$ is not empty. Then there is a literal l in this set.

Case 1. $l \in UnitPropagate(\Pi, M)$. Since M is also *cla*-model of Π, by Lemma 1 we derive a contradiction.

Case 2. $l \in AllRulesCancelled(\Pi, M)$. l has the form $\neg a$. By the conditions of *AllRulesCancelled* definition, it follows that (i) literal $\neg a$ is such that it does not belong to M, and (ii) there is no supporting rule in Π for a with respect to M. Since M is a *sup*-model of Π, we conclude from (ii) that $\neg a \in M$. This contradicts (i).

Right-to-left: Assume $ua(\Pi, M) = \emptyset$. By contradiction. Assume that M is not a *sup*-model of Π. Then either M is not a *cla*-model of Π or there is an atom $a \in M^+$ such that there is no supporting rule in Π for a with respect to M. In the former case, when M is not a *cla*-model of Π, by Lemma 1 we derive a contradiction. In the latter case, it follows that $\neg a \in AllRulesCancelled(\Pi, M)$ by the conditions of the *AllRulesCancelled* definition. We derive a contradiction.

We now show that the set *sd* is *sup*-enforcing. We start by claiming that the set *sd* is *sup*-sound. Let Π be any program, M be any set of literals, M' be any *sup*-model of Π such that $M \subseteq M'$. We have to illustrate that the set $sd(\Pi, M)$ is a subset of M'. Consider any literal l in the set $sd(\Pi, M)$. We show that l is also in M'. Given a proof that *ua* is *sup*-sound, it is only left to be proved that for any literal l that is in $BackchainTrue(\Pi, M)$, it also holds that $l \in M'$. Consider literal $l \in BackchainTrue(\Pi, M)$. By the definition of $BackchainTrue$, it follows that there is a rule $r = A \vee a \leftarrow B$ in Π so that (i) $a \in M$, and (ii) either $\overline{l} \in A$ or $l \in B$ and, (iii) no other rule in Π is supporting a with respect to M. By Lemma 2 and (iii), we derive that every rule other than r is such that it is not a supporting rule for a with respect to M'. By (i) and the fact that $M \subseteq M'$, $a \in M'$. Since M' is a *sup*-model of Π, it follows that $M \cap (\overline{B} \cup A) = \emptyset$. By the fact that M' is a consistent and complete set of literals over $atoms(\Pi)$, we conclude that $B \cup \overline{A} \subseteq M$. By (ii), $l \in M'$.

Statement 3. We have to show that the subsets of *sm* containing $\{UnitPropagate, Unfounded\}$ are *sta*-enforcing. We only illustrate this for the set $\{UnitPropagate, Unfounded\}$. We call this set *uu*. The proof for other sets (i) relies on the fact that any *sta*-model is also a *cla* and *sup*-model and (ii) follows the ideas presented in the proof of Statement 2.

We start by showing that the set *uu* is *sta*-sound. Let Π be any program, M be any set of literals, M' be any *sta*-model of Π such that $M \subseteq M'$. We have to illustrate that the set $uu(\Pi, M)$ is a subset of M'. Consider any literal l in the set $uu(\Pi, M)$. We now show that l is also in M'.

Case 1. $l \in UnitPropagate(\Pi, M)$. Since M' is a *sta*-model, M' is also a *cla*-model. The rest of the argument follows the lines of proof in *Statement 1*, which illustrates that *up* is *cla*-sound.

Case 2. $l \in Unfounded(\Pi, M)$. Literal l has the form $\neg a$. By the conditions of *Unfounded* definition, it follows that there is a set X containing a such that X is unfounded with respect to Π. By Lemma 3 and the fact that M' is *sta*-model of Π it follows that $\neg a \in M'$. (Indeed, consider a simple argument by contradiction.)

Second, we show that the set *uu* is *sta*-complete. Let Π be any program, M be any complete and consistent set of literals over $atoms(\Pi)$. We now show that M is *sta*-model of Π iff $uu(\Pi, M) = \emptyset$.

Left-to-right: Let M be a *sta*-model of Π. By contradiction. Assume that the set $uu(\Pi, M)$ is not empty. Then there is a literal l in this set.

Case 1. $l \in UnitPropagate(\Pi, M)$. Since M is also *cla*-model of Π, by Lemma 1 we derive a contradiction.

Case 2. $l \in Unfounded(\Pi, M)$. Literal l has the form $\neg a$. By the conditions of *Unfounded* definition, it follows that (i) literal $\neg a$ is such that it does not belong to M, and (ii) there is a set X containing a such that X is unfounded with respect to Π. Since M is a *sta*-model of Π, we conclude from (ii) that $\neg a \in M$. This contradicts (i).

Right-to-left: Assume that $uu(\Pi, M) = \emptyset$. By contradiction. Assume that M is not a *sta*-model of Π. By Theorem 1, either M is not a *cla*-model of Π or there is a non-empty subset of L^+ that is an unfounded set on L with respect to Π. In the former case, when M is not a *cla*-model of Π, by Lemma 1 we derive a contradiction. In the latter case, it follows that there is some atom a in an existing unfounded set so that $\neg a \in \mathit{Unfounded}(\Pi, M)$ by the conditions of the *Unfounded* definition. We derive a contradiction. □

6.2 Proofs of Theorems 3, 4, Propositions 2, 4, 5, 6

We start by the proof of Theorem 4. We skip the proof of Theorem 3 as it relies on the same proof techniques that proof of Theorem 4 exhibits. The proof of Theorem 4 relies on auxiliary lemmas as well as proofs of Propositions 4–6 that follow. We conclude this section with the proof of Proposition 2.

Lemma 4
Let g be a generating function and t be a witness function.

Let $\mathscr{P}_g$ and $\mathscr{P}_t$ be sets of p-conditions.

Then for any Π, the graph $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$ is finite and acyclic.

Proof
Consider the states of the graph $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$. The string L of states of the form $(L,R)_s$ or of the type $Ok(L)$ is built over a set of atoms which is bounded by the size of Π. Also, L does not allow repetitions. Thus, there is a finite number of possible strings L in the states $(L,R)_s$ or $Ok(L)$. It immediately follows that there is a finite number of states $Ok(L)$ in $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$.

Consider the right side of a state of the form $(L,R)_s$. Since $t(\Pi, L)$ has a finite number of atoms and there is a finite number of possible L, the set of atoms over which R is built is finite. Consequently, there is a finite number of possible R. We conclude that there is a finite number of possible states $(L,R)_s$. Thus, the set of states is finite in $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$.

For any string L of literals, by $|L|$ we denote the length of this string. Any string L of literals can be written $L_0 l_1^\Delta L_1 \ldots l_k^\Delta L_k$, where $(l_i^\Delta)_{1\leqslant i\leqslant k}$ contains all the decision literals of L. Let us call $v(L)$ the sequence $|L_0|, |L_1| \ldots |L_k|$. We then write $L \leqslant L'$ iff $v(L) \leqslant_{lex} v(L')$ where $\leqslant_{lex}$ is the lexicographic order. Since the length of the sequence $v(L)$ is bounded by the finite number of possible decision literals, this is a well-founded order. Finally, we say that $(L,R)_s \leqslant (L',R')_{s'}$ iff $(L,R,s) \leqslant_{lex} (L',R',s')$ where $\leqslant_{lex}$ is the lexicographic order and $\mathscr{L} < \mathscr{R}$. This is clearly well founded as it is the lexicographic composition of well-founded orders.

If there is a transition from $(L,R)_s$ to $(L',R')_{s'}$, then $(L,R)_s \leqslant (L',R')_{s'}$ and $(L,R)_s \neq (L',R')_{s'}$. This can be checked simply for each of the rules. Since the order is well founded, there is no infinite path in the graph. Consequently, the graph is acyclic. □

Proof of Proposition 5
We first show that $l_1.\cdots.l_{k_1}$ is consistent. By contradiction. Assume that $l_1.\cdots.l_{k_1}$ is inconsistent. Then since $\mathit{Conclude}_{\mathscr{L}}$ is not applicable $l_1.\cdots.l_{k_1}$ contains at least one decision literal. We now define i as $l_1.\cdots.l_{k_1} = l_1.\cdots.l_{i-1}.l_i^\Delta.l_{i+1}.\cdots.l_{k_1}$ where l_i^Δ is the rightmost decision literal. Since $\mathit{Backtrack}_{\mathscr{L}}$ is not applicable $l_1.\cdots.l_{k_1}$ contains no decision literal. We derive a contradiction.

Since $Decide_{\mathscr{L}}$ is not applicable and $l_1.\cdots.l_{k_1}$ is consistent, $l_1.\cdots.l_{k_1}$ assigns all the atoms of $atoms(g(\Pi))$. As a consequence, $l_1.\cdots.l_{k_1}$ is a consistent and complete set of literals that covers $atoms(g(\Pi))$. Finally, $Propagate_{\mathscr{L}}$ is not applicable. So $\mathscr{P}_g(\Pi, l_1.\cdots.l_{k_1})$ is the empty set. Since $\mathscr{P}_g$ is w-enforcing and hence w-complete, $l_1.\cdots.l_{k_1}$ is a w-model of $g(\Pi)$. □

Proof of Proposition 6

Statements $(a-c)$ We prove these statements by induction on the length of a path in the graph $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$ from the initial state. Since the statements trivially hold in the initial state of the graph, we only have to prove that all transition rules of $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$ preserve the properties.

Statement (c) trivially holds for all transitions but crossing-rules $\mathscr{RL}$. Statements (a) and (b) trivially hold for transitions due to left-rules, crossing-rules $\mathscr{RL}$, $Conclude_{\mathscr{R}}$, $Conclude_{\mathscr{RL}}$.

Consider an edge due to one of the right-rules or crossing-rules $\mathscr{LR}$ from state $S = (l^0_1.\cdots.l^0_{k^0_1}, r^0_1.\cdots.r^0_{k^0_2})_{s^0}$ to state $S' = (l_1.\cdots.l_{k_1}, r_1.\cdots.r_{k_2})_{\mathscr{R}}$ so that the statements (a) and (b) hold on S (an inductive hypothesis). For the right-rules (excluding $Conclude_{\mathscr{R}}$), the left side of the state remains unchanged. Thus, by induction hypothesis (a) immediately follows. Similarly, it is easy to see from the conditions of these rules that they also preserve property (b). We now illustrate that the $Cross_{\mathscr{LR}}$ preserves $(a-c)$.

Case $Cross_{\mathscr{LR}}$*:* It follows that (i) $s_0 = \mathscr{L}$, (ii) $l_1.\cdots.l_{k_1} = l^0_1.\cdots.l^0_{k^0_1}$, (iii) no left-rule applies to S, (iv) $r_1.\cdots.r_{k_1} = r^0_1.\cdots.r^0_{k^0_1} = \emptyset$. By Proposition 5, (i), and (iii), we conclude that $l^0_1.\cdots.l^0_{k^0_1}$ is a w_1-model of $g(\Pi)$. By (ii), it follows that $l_1.\cdots.l_{k_1}$ is also a w_1-model of $g(\Pi)$. Thus, (c) holds. From the definition of $g(\Pi)$, it follows that the set $\{l_1.\cdots.l_{k_1}\}$ of literals covers Π. It follows that $t(\Pi, l_1.\cdots.l_{k_1})$ is defined. Thus, (a) holds. From (iv), (b) trivially follows as the right side of the state is empty.

Statement (d) We first show that $r_1.\cdots.r_{k_2}$ is consistent. By contradiction. Assume that $r_1.\cdots.r_{k_2}$ is inconsistent. Then since $Conclude_{\mathscr{R}}$ is not applicable, $r_1.\cdots.r_{k_2}$ contains at least one decision literal. We now define i as $r_1.\cdots.r_{k_2} = r_1.\cdots.r_{i-1}.r^{\Delta}_i.r_{i+1}.\cdots.r_{k_2}$ where r^{Δ}_i is the rightmost decision literal. Since the rule $Backtrack_{\mathscr{R}}$ is not applicable $r_1.\cdots.r_{k_2}$ contains no decision literal. We derive a contradiction.

Since $Decide_{\mathscr{R}}$ is not applicable and $r_1.\cdots.r_{k_2}$ is consistent, by (b) $r_1.\cdots.r_{k_2}$ assigns all the atoms of $atoms(t(\Pi, l_1.\cdots.l_{k_1}))$. Thus, $r_1.\cdots.r_{k_2}$ is a consistent and complete set of literals over $atoms(t(\Pi, l_1.\cdots.l_{k_1}))$. Finally, $Propagate_{\mathscr{R}}$ is not applicable. So $\mathscr{P}_t(t(\Pi, l_1.\cdots.l_{k_1}), r_1.\cdots.r_{k_2})$ is the empty set. Since $\mathscr{P}_t$ is w_2-enforcing and hence w_2-complete, $r_1.\cdots.r_{k_2}$ is a w_2-model of $t(\Pi, l_1.\cdots.l_{k_1})$. □

Lemma 5

Let w_1 and w_2 be some types in $\{cla, sup, sta\}$.

Let g be a generating function and t be a witness function.

Let $\mathscr{P}_g$ be a w_1-enforcing set of p-conditions and $\mathscr{P}_t$ be a w_2-enforcing set of p-conditions.

Let Π be a program.

Let $(l_1.\cdots.l_{k_1}, r_1.\cdots.r_{k_2})_s$ be a state of $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$ reachable from the initial state.

Then,

(a) any w_2-model of $t(\Pi, l_1.\cdots.l_{k_1})$ satisfies r_i if it satisfies all decision literals $(r_j)^\Delta$ with $j \leqslant i$.

(b) Any w_1-model L of $g(\Pi)$ such that $t(\Pi, L)$ has no w_2-model satisfies l_i if it satisfies all decision literals l_j^Δ with $j \leqslant i$.

Proof

We prove statements (a) and (b) by induction on the length of a path in the graph $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$ from the initial state. Since the statements trivially hold in the initial state of the graph, we only have to prove that all transition rules of $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$ preserve the properties.

Consider an edge from the state $S = (l_1^0.\cdots.l^0_{k_1^0}, r_1^0.\cdots.r^0_{k_2^0})_{s^0}$ to the state $S' = (l_1.\cdots.l_{k_1}, r_1.\cdots.r_{k_2})_s$ so that the statements (a) and (b) hold on S (an inductive hypothesis).

The statements (a) and (b) trivially hold for the case of transitions due to $Conclude_{\mathscr{L}}$, $Conclude_{\mathscr{R}}$, $Conclude_{\mathscr{R}\mathscr{L}}$.

For the case of transition rules $Cross_{\mathscr{L}\mathscr{R}}$, $Backtrack_{\mathscr{R}}$, $Decide_{\mathscr{R}}$, $Propagate_{\mathscr{R}}$ it holds that $l_1.\cdots.l_{k_1} = l_1^0.\cdots.l^0_{k_1^0}$. So by the induction hypothesis, (b) trivially holds on $(l_1.\cdots.l_{k_1}, r_1.\cdots.r_{k_2})_s$. For these rules, we are left to show that (a) holds on $(l_1.\cdots.l_{k_1}, r_1.\cdots.r_{k_2})_s$. Note that for the case of $Backtrack_{\mathscr{R}}$, $Decide_{\mathscr{R}}$, $Propagate_{\mathscr{R}}$, by Proposition 6 (a) it follows that $t(\Pi, l_1.\cdots.l_{k_1})$ is defined.

Case $Cross_{\mathscr{L}\mathscr{R}}$: It follows that $r_1.\cdots.r_{k_1} = r_1^0.\cdots.r^0_{k_1^0} = \emptyset$. Consequently, (a) holds as right side of the state is empty.

Case $Backtrack_{\mathscr{R}}$. In this case, there is an index i such that $r_1^0.\cdots.r^0_{k_2^0} = r_1^0.\cdots.r^0_{i-1}.(r_i^0)^\Delta.r^0_{i+1}.\cdots.r^0_{k_2^0}$ and $r_1.\cdots.r_{k_2-1} = r_1^0.\cdots.r^0_{i-1}$. Also, by the conditions of $Backtrack_{\mathscr{R}}$, the string of literals $r_1^0.\cdots.r^0_{k_2^0}$ is inconsistent. Let M be a w_2-model of $t(\Pi, l_1.\cdots.l_{k_1})$. Let r_j be a literal of $r_1.\cdots.r_{k_1}$. Assume M satisfies all decision literals $(r_{j'})^\Delta$ with $j' \leqslant j$. By the induction hypothesis, if $j \neq k_2$ then M satisfies r_j. It remains to prove that this is also true when $j = k_2$. Assume M satisfies all the decision literals of $r_1.\cdots.r_{k_2}$. They include all the decision literals of $r_1.\cdots.r_{k_2-1}$. Then, M satisfies all the literals of $r_1.\cdots.r_{k_2-1}$ by the induction hypothesis. We now show that M also satisfies r_{k_2}.

None of the literals $r^0_{i+1}\cdots r^0_{k_2^0}$ is a decision literal. Additionally, $r_1.\cdots.r_{k_2} = r_1^0.\cdots.r^0_{i-1}.\overline{r_i^0}$. Since M satisfies all the literals of $r_1.\cdots.r_{k_2-1}$, it satisfies all the literals of $r_1^0.\cdots.r^0_{i-1}$. Since $r_1^0.\cdots.r^0_{k_2^0}$ is inconsistent, M cannot satisfy all of its literals, so M does not satisfy at least one literal of $r_i^0\cdots r^0_{k_2^0}$. By the contraposition of the induction hypothesis (a), and since none of the literals $r^0_{i+1}\cdots r^0_{k_2^0}$ is a decision literal, one of the literals not satisfied by M has to be r_i^0. So M must satisfy $\overline{r_i^0}$, that is r_{k_2}.

Case $Decide_{\mathscr{R}}$. Obvious.

Case $Propagate_{\mathscr{R}}$. Let M be a w_2-model of $t(\Pi, l_1.\cdots.l_{k_1})$. Assume M satisfies all the decision literals of $r_1.\cdots.r_{k_2}$. Since for any propagator condition r_{k_2} is not a decision literal, they are the decision literals of $r_1.\cdots.r_{k_2-1}$. So M satisfies all the literals of $r_1.\cdots.r_{k_2-1}$ by the induction hypothesis. In other words, $\{r_1.\cdots.r_{k_2-1}\} \subseteq M$. Proving that M satisfies r_{k_2} will complete the proof. We are given that $\mathscr{P}_t$ is w_2-enforcing and hence w_2-sound. By definition of w_2-soundness and the fact that $\{r_1.\cdots.r_{k_2-1}\} \subseteq M$, it follows that $\mathscr{P}_t(\Pi, \{r_1.\cdots.r_{k_2-1}\}) \subseteq M$. Since $r_{k_2} \in \mathscr{P}_t(\Pi, \{r_1.\cdots.r_{k_2-1}\})$, also $r_{k_2} \in M$. In other words, M satisfies r_{k_2}.

We are left to illustrate that transition rules $Backtrack_{\mathscr{R}\mathscr{L}}$, $Backtrack_{\mathscr{L}}$, $Decide_{\mathscr{L}}$, $Propagate_{\mathscr{L}}$ preserve properties (a) and (b). Since all of these rules are such that the right side of the resulting state is $\emptyset$, clearly (a) is preserved. We will only illustrate that $Backtrack_{\mathscr{R}\mathscr{L}}$ preserves (b) as the remaining cases for (b) are similar to the arguments constructed above for the respective right-rules and property (a).

Case $Backtrack_{\mathscr{R}\mathscr{L}}$. There is an index i such that $l_1.\cdots.l_{k_1-1} = r^0_1.\cdots.r^0_{i-1}$ and $l^0_1.\cdots.l^0_{k^0_1} = l^0_1.\cdots.l^0_{i-1}.(l^0_i)^\Delta.l^0_{i+1}.\cdots.l^0_{k^0_1}$. Let M be a w_1-model of $g(\Pi)$ such that $t(\Pi, M)$ has no w_2-model. Assume that M satisfies all the decision literals of $l_1.\cdots.l_{k_1}$. Since l_{k_1} is not a decision literal, they are the decision literals of $l_1.\cdots.l_{k_1-1}$. So M satisfies all the literals of $l_1.\cdots.l_{k_1-1}$ by the induction hypothesis. Showing that M satisfies l_{k_1} will complete the proof.

Since the transition is justified by $Backtrack_{\mathscr{R}\mathscr{L}}$, by Proposition 6 (c), $l^0_1.\cdots.l^0_{k^0_1}$ is a w_1-model of $g(\Pi)$. By Proposition 6 (d) and the fact that no right-rule applies, $r^0_1.\cdots.r^0_{k^0_2}$ is a w_2-model of $t(\Pi, l_1.\cdots.l_{k_1})$. So $t(\Pi, l_1.\cdots.l_{k_1})$ has a w_2-model, hence M does not satisfy all the literals of $l^0_1.\cdots.l^0_{k^0_1}$. Consequently, since M satisfies all the literals of $l_1.\cdots.l_{k_1-1}$, at least one literal from $(l_i{}^0)^\Delta.l^0_{i+1}.\cdots.l^0_{k^0_1}$ is not satisfied by M, which by the contraposition of the induction hypothesis (b) proves that $l^0_i = \overline{l_{k_1}}$ is not satisfied by M. This means that M satisfies l_{k_1}. □

Lemma 6
Let w_1 and w_2 be some types in $\{cla, sup, sta\}$.

Let $\mathscr{P}_g$ be a w_1-enforcing set of p-conditions. Let g be a generating function.

Let $\mathscr{P}_t$ be a w_2-enforcing set of p-conditions. Let t be a witness function.

Let Π be a program. Then,

(1) any terminal state of $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$ reachable from the initial state and other than *Failstate* is $Ok(L)$, with L being a w_1-model of $g(\Pi)$ such that $t(\Pi, L)$ has no w_2-model,

(2) *Failstate* is reachable from the initial state iff $g(\Pi)$ has no w_1-model L such that $t(\Pi, L)$ has no w_2-model.

Proof
We first illustrate that any terminal state is either *Failstate* or of the form $Ok(L)$ for some L. By contradiction. Assume there is a terminal state of the form $(L, R)_s$. Case 1. $s = \mathscr{L}$. Then either a left-rule or $Cross_{\mathscr{L}\mathscr{R}}$ applies, so $(L, R)_s$ is not terminal. We derive a contradiction. Case 2. $s = \mathscr{R}$. Since $Conclude_{\mathscr{R}\mathscr{L}}$ does not apply while no right-rule applies and no left-rule applies, L contains at least one decision literal. Since $Backtrack_{\mathscr{R}\mathscr{L}}$ is not applicable, L contains no decision literal. We derive a contradiction.

Statement 1. 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 $Conclude_{\mathscr{R}}$. Let us call $(L, R)_s$ a state from which a transition $Conclude_{\mathscr{R}}$ leads to $Ok(L)$. By the definition of $Conclude_{\mathscr{R}}$, we know that: $s = \mathscr{R}$, that R is inconsistent and that R contains no decision literal. By Lemma 5 item (c), the consistent set of literals obtained from L is a w_1-model of $g(\Pi)$.

By Lemma 5 item (a), and as R contains no decision literal, any w_2-model of $t(\Pi, L)$ satisfies all the literals of R. Since R is inconsistent, any w_2-model of $t(\Pi, L)$ is inconsistent. So $t(\Pi, L)$ has no w_2-model.

We have just proved that L is a w_1-model of $g(\Pi)$ such that $t(\Pi, L)$ has no w_2-model.

Statement 2. Assume *Failstate* is not reachable from the initial state. Then, since the graph is acyclic, there is a terminal state different from *Failstate*. By Claim 1, this state is $Ok(L)$, and L is a w_1-model of $g(\Pi)$ such that $t(\Pi, L)$ has no w_2-model.

Assume *Failstate* is 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 $Conclude_{\mathscr{L}}$ or $Conclude_{\mathscr{R}\mathscr{L}}$. Let us call $(L, R)_s$ a state from which a transition leads to *Failstate*. In either of these cases, L does not contain any decision literal; so by Lemma 5, any w_1-model M of $g(\Pi)$ such that $t(\Pi, M)$ has no w_2-model satisfies all the literals of L. In other words, L is the only possible candidate for a w_1-model of $g(\Pi)$ such that $t(\Pi, L)$ has no w_2-model.

Case $Conclude_{\mathscr{L}}$. It follows that L is inconsistent. Consequently, it is not a w_1-model M of $g(\Pi)$ such that $t(\Pi, L)$ has no w_2-model.

Case $Conclude_{\mathscr{R}\mathscr{L}}$. By Proposition 5 (d), the set of literals R is a w_2-model of $t(\Pi, L)$. Thus, L is not a w_1-model of $g(\Pi)$ such that $t(\Pi, L)$ has no w_2-model. □

Proof of Proposition 4

We first illustrate that any set M of literals that is a w_1-model of $g(\Pi)$ such that $t(\Pi, M)$ has no w_2-model is such that $M_{|atoms(\Pi)}$ is a stable model of Π. Indeed, by the definition of w_1-approximating functions w.r.t. w, $M_{|atoms(\Pi)}$ is a w-model of Π. Also, by the definition of w_2-ensuring functions w.r.t. w, $M_{|atoms(\Pi)}$ is a stable model of Π.

Second, consider any stable model L of Π. By the definitions of w_1-approximating and w_2-ensuring functions w.r.t. w, it follows there is M' such that $M'_{|atoms(\Pi)} = L$ and M' is a w_1-model of $g(\Pi)$ such that $t(\Pi, M')$ has no w_2-model. □

Proof of Theorem 4

Let w_1 denote a type such that $\mathscr{P}_g$ is w_1-enforcing and the function g is w_1-approximating w.r.t. w. Let w_2 denote a type such that $\mathscr{P}_t$ is w_2-enforcing and function t is w_2-ensuring w.r.t. w. We now proceed to prove the four conditions of the definition of 'checks' one by one.

(1) By Lemma 4, the graph $STT^{\mathscr{P}_g, g}_{\mathscr{P}_t, t}(\Pi)$ is acyclic and finite.
(2) By Lemma 6 item 1, any terminal state is either *Failstate* or $Ok(L)$.
(3) By Lemma 6 item 1, any terminal state of $STT^{\mathscr{P}_g, g}_{\mathscr{P}_t, t}(\Pi)$ reachable from the initial state and other than *Failstate* is $Ok(L)$, with L being a w_1-model of $g(\Pi)$ such that $t(\Pi, L)$ has no w_2-model. By Proposition 4, $L_{|atoms(\Pi)}$ is a stable model of Π.

(4) By Lemma 6 item 2, *Failstate* is reachable from the initial state iff $g(\Pi)$ has no w_1-model L such that $t(\Pi, L)$ has no w_2-model. By Proposition 4, Π has no stable models. □

Proof of Proposition 2

Recall how we argued DP^2_{Π} is $STT^{up,g^C}_{up,t^C}(\Pi)$. Similarly, $DP^2_{g,t}(\Pi)$ is $STT^{up,g}_{up,t}(\Pi)$. By Theorem 2, *up* is *cla*-enforcing.

(1) By Lemma 4, $DP^2_{g,t}(\Pi)$ is finite and acyclic.
(2) By Lemma 6 item 1, any terminal state is either *Failstate* or $Ok(L)$.
(3) By Lemma 6 item 1, any terminal state of $STT^{\mathscr{P}_g,g}_{\mathscr{P}_t,t}(\Pi)$ reachable from the initial state and other than *Failstate* is $Ok(L)$, with L being a *cla*-model of $g(\Pi)$ such that $t(\Pi, L)$ has no *cla*-model.
(4) By Lemma 6 item 2, *Failstate* is reachable from the initial state iff $g(\Pi)$ has no *cla*-model L such that $t(\Pi, L)$ has no *cla*-model. □

6.3 *Proof of Theorem 5*

First, we prove an auxiliary lemma that will help handling CNF conversions of DNF formulas.

For a DNF formula F, we define $CNF(F)$ as the conversion of F to CNF using straightforward equivalent transformations: the distributivity of disjunction over conjunction.

Lemma 7

Let F be a DNF formula. Let l be a literal of F. Let M be a set of literals.

The two following statements are equivalent:

(1) there is a conjunctive clause D of F such that for every conjunctive clause $D' \in F$ different from D, D' is contradicted by M,
(2) there is a clause C of $CNF(F)$ such that $l \in C$ and M contradicts $C \setminus \{l\}$.

Proof

Formula F has the form $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{k} l_{ij}$ (when necessary the true constant $\top$ is added multiple times to ensure that the conjunctive clauses of F are of equal length). Also, $CNF(F) = \bigwedge_{(k_1 \dots k_n) \in \{1 \dots k\}^n} \bigvee_{i=1}^{n} l_{ik_i}$.

From Statement 1 to Statement 2: 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 M. Let l be a literal of D. Let D be $\bigwedge_{j=1}^{k} l_{i_0 j}$ for some i_0. As any other conjunctive clause is contradicted by M, and as these clauses are conjunctions, there is least one literal of each of these clauses that is contradicted by M. Let us call $r_1 \dots r_{i_0-1} r_{i_0+1} \dots r_n$ these literals. Then for each $i \in \{1, \dots, i_0 - 1, i_0 + 1, \dots, n\}$, there is $k_i^0 \in \{1, \dots k\}$ such that $l_{i,k_i^0} = r_i$. Also, there is some $k_{i_0}^0$ such that $l_{i,k_{i_0}^0} = l$. Then the clause $\bigvee_{i=1}^{n} l_{ik_i^0}$ of $CNF(F)$ contains l while each of the other literals it contains is contradicted by M.

From Statement 2 to Statement 1: Assume that for some clause of $CNF(F)$, all literals but one are known to be contradicted by M. Then let this clause be $\bigvee_{i=1}^{n} l_{ik_i}$ for some i and let $l_{i_0 k_{i_0}}$ be the literal that is not contradicted by M. Then l_{ik_i} is contradicted by M for any i other than i_0. So $\bigwedge_{j=1}^{k} l_{ij}$ is contradicted by M for any i other than i_0. So $D = \bigwedge_{j=1}^{k} l_{i_0 j}$ is a conjunctive clause of F such that for any other conjunctive clause D' of F, this clause is contradicted by M. $\square$

Proof of Theorem 5
We must prove that for any edge in the graph $STT^{sd,g^D}_{\mathscr{P}_t,t}(\Pi)$ there is an edge in $STT^{up,cnfcomp}_{\mathscr{P}_t,t}(\Pi)$ linking two identical vertexes, and for any edge in the graph $STT^{up,cnfcomp}_{\mathscr{P}_t,t}(\Pi)$ there is an edge in $STT^{sd,g^D}_{\mathscr{P}_t,t}(\Pi)$ linking two identical vertexes.

If the edge is justified by a right-rule, then this is obvious as these two graphs have the same witness function and the same set of conditions for the $Propagate_{\mathscr{R}}$ rule. If the edge is $Decide_{\mathscr{L}}$, $Conclude_{\mathscr{L}}$, $Backtrack_{\mathscr{L}}$, $Backtrack_{\mathscr{R}\mathscr{L}}$ or $Conclude_{\mathscr{R}\mathscr{L}}$, then obviously there is the same edge in the other graph, bearing the same name, as these edges do not depend on the generating program or set of conditions for the $Propagate_{\mathscr{R}}$ rule.

It remains to study the case of an edge justified by $Propagate_{\mathscr{L}}$ or $Cross_{\mathscr{L}\mathscr{R}}$. Assume we also have proved that $Propagate_{\mathscr{L}}$ rules are identical in both graphs. Then if an edge is justified by $Cross_{\mathscr{L}\mathscr{R}}$ in one of the graphs, which means that no left-rule applies in this graph, equivalently no left-rule applies in the other graph, and $Cross_{\mathscr{L}\mathscr{R}}$ also applies in that graph. We now show that $Propagate_{\mathscr{L}}$ rules are identical in both graphs, which will complete the proof.

Assume that an edge is justified by $Propagate_{\mathscr{L}}$ in one of the graphs, let us prove it also exists in the other graph.

A Transition in $STT^{sd,g^D}_{\mathscr{P}_t,t}(\Pi)$ is Justified by $Propagate_{\mathscr{L}}$ with UnitPropagate as Condition. Then also there is an edge in $STT^{up,cnfcomp}_{\mathscr{P}_t,t}(\Pi)$ with the same effect, and justified by $Propagate_{\mathscr{L}}$ with the *UnitPropagate* condition. Indeed, Π is part of $cnfcomp(\Pi)$.

A Transition in $STT^{sd,g^D}_{\mathscr{P}_t,t}(\Pi)$ is Justified by $Propagate_{\mathscr{L}}$ with AllRulesCancelled as Condition. Then the edge is turning $(L, \emptyset)_{\mathscr{L}}$ into $(L\neg a, \emptyset)_{\mathscr{L}}$, and each rule $A \vee a \leftarrow B \in \Pi$ is not a supporting rule for a w.r.t. L. In other words, for each rule $A \vee a \leftarrow B \in \Pi$ the following holds $L \cap (\overline{B} \cup A) \neq \emptyset$. Consequently, the conjunction $B \wedge \overline{A}$ is contradicted by L. As a consequence $\bigvee_{A \vee a \leftarrow B \in \Pi}(B \wedge \overline{A})$ is contradicted by L. From Lemma 7, the fact that the DNF formula $\neg a \vee \bigvee_{A \vee a \leftarrow B \in \Pi}(B \wedge \overline{A})$ belongs to $comp(\Pi)$, and the *cnfcomp* construction, it follows that there is a clause C in $cnfcomp(\Pi)$ such that $\neg a \in C$ and L contradicts $C \setminus \{\neg a\}$. So the rule $Propagate_{\mathscr{L}}$ with condition *UnitPropagate* of $STT^{up,cnfcomp}_{\mathscr{P}_t,t}(\Pi)$ can be applied to C to add $\neg a$, providing the edge we needed.

A Transition in $STT^{sd,g^D}_{\mathscr{P}_t,t}(\Pi)$ is Justified by $Propagate_{\mathscr{L}}$ with BackchainTrue as Condition. The proof of this case is similar to the proof of previous case.

A Transition in $STT^{up,cnfcomp}_{\mathscr{P}_t,t}(\Pi)$ is Justified by $Propagate_{\mathscr{L}}$ with the Condition UnitPropagate. Let us call F_0 the DNF formula $\neg a \vee \bigvee_{A \vee a \leftarrow B \in \Pi}(B \wedge \overline{A})$ of $comp(\Pi)$ for some atom a in Π.

Case 1: $Unit_{\mathscr{L}}$ is applied to a clause of Π in $cnfcomp(\Pi)$. Then $Propagate_{\mathscr{L}}$ with the condition *UnitPropagate* itself provides the desired edge in $STT^{sd,g^D}_{\mathscr{P}_t,t}(\Pi)$.

Case 2: $Unit_{\mathscr{L}}$ is applied to a clause obtained from F_0 by the cnfcomp conversion. Then by Lemma 7, the *cnfcomp* construction, and the $Unit_{\mathscr{L}}$ condition there is a conjunctive clause D of F_0 such that for every conjunctive clause D' in F_0 that is different from D the current L contradicts D'.

Case 2.1: This conjunctive clause is $\neg a$. Then L contradicts $\bigvee_{A \vee a \leftarrow B \in \Pi}(B \wedge \overline{A})$. It is easy to see that *AllRulesCancelled* provides the desired edge.

Case 2.2: This conjunctive clause is some $B \wedge \overline{A}$. Then L contradicts $\neg a$ so a belongs to L. Also L contradicts all of $\{B' \wedge \overline{A'} | A' \vee a \leftarrow B' \in \Pi \setminus \{A \vee a \leftarrow B\}\}$. As a consequence, *BackchainTrue* provides the desired edge. □

7 Conclusions, future, and related work

Transition systems for describing DPLL-based solving procedures have been introduced by Nieuwenhuis *et al.* (2006). Lierler (2008) introduced and compared the transition systems for the answer set solvers SMODELS and CMODELS for non-disjunctive programs. In this paper, we continue this direction of work by presenting a two-layer framework suitable to capture disjunctive answer set solvers. We argue that this framework allows simpler analysis and comparison of these systems. We first introduce a general template that includes the techniques implemented in such solvers, and then define specific solvers by instantiating appropriate techniques using this template. Formal results about the correctness of the abstract representations are given. 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. Section 5 hints at some of the possibilities. Indeed, to obtain a new solver one can combine any appropriately chosen approximating-pair and ensuring-pair.

Nieuwenhuis *et al.* (2006) considered another extension of the graphs by introducing transition rules that capture backjumping and learning techniques common in design of modern solvers, that later allowed Lierler (2011) to design, e.g., abstract CLASP. It is a direction of future work to extend the two-layer template graph to model such advances solving techniques. This extension will allow us to model disjunctive answer set solvers that rely heavily on backjumping and learning such as CLASP and WASP.

Related Work. The approach based on transition systems for describing and comparing ASP procedures is one of the three main alternatives studied in the ASP literature. Other methods include pseudo-code presentation of algorithms (Giunchiglia and Maratea 2005; Giunchiglia *et al.* 2008) and tableau calculi (Gebser and Schaub 2006; Gebser and Schaub 2013). Giunchiglia *et al.* (2008) presented pseudo-code descriptions of CMODELS without backjumping and learning, SMODELS and DLV without backjumping restricted to non-disjunctive programs. They study relationships to the solving algorithms by analyzing the correspondence about the search spaces they explore, focusing on tight programs: in particular, they note a tight relation between solvers CMODELS and DLV. Gebser and Schaub (2013) 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.

References

Alviano, M., Dodaro, C., Faber, W., Leone, N. and Ricca, F. 2013. WASP: A native ASP solver based on constraint learning. In *Proc. of the 12th International Conference of Logic Programming and Nonmonotonic Reasoning (LPNMR 2013)*, P. Cabalar and T. C. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer, Berlin, 54–66.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Brochenin, R., Lierler, Y. and Maratea, M. 2014. Abstract disjunctive answer set solvers. In *Proc. of the 21st European Conference on Artificial Intelligence (ECAI 2014)*. Frontiers in Artificial Intelligence and Applications, vol. 263. IOS, Amsterdam, 165–170.

Brooks, D. R., Erdem, E., Erdoğan, S. T., Minett, J. W. and Ringe, D. 2007. Inferring phylogenetic trees using answer set programming. *Journal of Automated Reasoning 39*, 471–511.

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

Eiter, T. and Gottlob, G. 1993. Complexity results for disjunctive logic programming and application to nonmonotonic logics. In *Proc. of the 1993 International Logic Programming Symposium (ILPS)*, D. Miller, Ed. MIT Press, 266–278.

Eiter, T., Gottlob, G. and Mannila, H. 1997. Disjunctive datalog. *ACM Transactions on Database Systems 22*(*3*), 364–418.

Faber, W. 2002. *Enhancing Efficiency and Expressiveness in Answer Set Programming Systems*. Ph.D. thesis, Vienna University of Technology.

Gebser, M., Kaufmann, B. and Schaub, T. 2013. Advanced conflict-driven disjunctive answer set solving. In *Proc. of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, F. Rossi, Ed. IJCAI/AAAI.

Gebser, M. and Schaub, T. 2006. Tableau calculi for answer set programming. In *Proc. of the 22nd International Conference on Logic Programming (ICLP 2006)*, S. Etalle and M. Truszczynski, Eds. Lecture Notes in Computer Science, vol. 4079. Springer, Berlin, 11–25.

Gebser, M. and Schaub, T. 2013. Tableau calculi for logic programs under answer set semantics. *ACM Transaction on Computational Logic 14*(*2*), 15.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP 1988)*, R. Kowalski and K. Bowen, Eds. MIT Press, Las Cruces, 1070–1080.

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

Giunchiglia, E., Leone, N. and Maratea, M. 2008. On the relation among answer set solvers. *Annals of Mathematics and Artificial Intelligence 53*(1–4), 169–204.

Giunchiglia, E. and Maratea, M. 2005. On the relation between answer set and SAT procedures (or, between smodels and cmodels). In *Proc. of the 21st International Conference on Logic Programming (ICLP 2005)*, M. Gabbrielli and G. Gupta, Eds. Lecture Notes in Computer Science, vol. 3668. Springer, Berlin, 37–51.

Janhunen, T., Niemelä, I., Seipel, D., Simons, P. and You, J.-H. 2006. Unfolding partiality and disjunctions in stable model semantics. *ACM Transactions on Computunational Logic 7*(*1*), 1–37.

Koch, C., Leone, N. and Pfeifer, G. 2003. Enhancing disjunctive logic programming systems by sat checkers. *Artificial Intelligence 151(1–2)*, 177–212.

Leone, N., Faber, W., Pfeifer, G., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7(3)*, 499–562.

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

Lierler, Y. 2005. Cmodels: SAT-based disjunctive answer set solver. In *Proc. of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2005)*, C. Baral, G. Greco, N. Leone, and G. Terracina, Eds. Lecture Notes in Computer Science, vol. 3662. Springer, Berlin, 447–452.

Lierler, Y. 2008. Abstract answer set solvers. In *Proc. of the 24th International Conference on Logic Programming (ICLP 2008)*, M. G. de la Banda and E. Pontelli, Eds. Lecture Notes in Computer Science, vol. 5366. Springer, Berlin, 377–391.

Lierler, Y. 2010. *SAT-Based Answer Set Programming*. Ph.D. thesis, University of Texas at Austin.

Lierler, Y. 2011. Abstract answer set solvers with backjumping and learning. *Theory and Practice of Logic Programming 11*, 135–169.

Lierler, Y. and Truszczynski, M. 2011. Transition systems for model generators – a unifying approach. *Theory and Practice of Logic Programming 11(4–5)*, 629–646.

Lifschitz, V. 1999. Answer Set Planning. In *Proc. of the 16th International Conference on Logic Programming (ICLP 1999)*, D. D. Schreye, Ed. The MIT Press, Las Cruces, 23–37.

Marek, V. and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: A 25-Year Perspective*. Springer-Verlag, Berlin, 375–398.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25*, 241–273.

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

Perri, S., Scarcello, F., Catalano, G. and Leone, N. 2007. Enhancing DLV instantiator by backjumping techniques. *Annals of Mathematics and Artificial Intelligence 51(2–4)*, 195–228.

Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S. and Leone, N. 2012. Team-building with answer set programming in the gioia-tauro seaport. *Theory and Practice of Logic Programming 12(3)*, 361–381.

Simons, P., Niemelä, I. and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138*, 181–234.

Soininen, T. and Niemelä, I. 1999. Developing a Declarative Rule Language for Applications in Product Configuration. In *Proc. of the 1st International Workshop on Practical Aspects of Declarative Languages (PADL 1999)*, G. Gupta, Ed. Lecture Notes in Computer Science, vol. 1551. Springer, Berlin, 305–319.

Syrjänen, T. 2001. Omega-restricted logic programs. In *Proc. of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2001)*, T. Eiter, W. Faber, and M. Truszczynski, Eds. Lecture Notes in Computer Science, vol. 2173. Springer, Berlin, 267–279.