Tableaux Calculi for Answer Set Programming
— *Extended Abstract* —

Martin Gebser and Torsten Schaub

Institut für Informatik, Universität Potsdam, Postfach 90 03 27, D–14439 Potsdam

Abstract. We introduce a family of calculi for Answer Set Programming (ASP) based on tableaux methods. Our approach furnishes declarative and fine-grained instrumentalities for characterizing operations as well as strategies of ASP-solvers. First, the granulation is detailed enough to capture the variety of propagation and choice operations in algorithms used for ASP; this also includes SAT-based approaches. Second, it is general enough to encompass the various strategies pursued by existing ASP-solvers, like *assat*, *cmodels*, *dlv*, *nomore++*, *smodels*, etc. This provides us with a uniform framework for comparing existing solvers. Third, the approach is flexible enough to integrate new inference patterns, so to study their relation to existing ones. As a result, we obtain a new approach to computing unfounded sets by means of loops. Furthermore, it allows us to define a backward inference for unfounded set computation that appears to be the first of its kind. Finally, our approach allows us to investigate the proof complexity of ASP-solvers, depending on choice operations. In particular, we show that exponentially different best-case computations can be obtained for different ASP-solvers.

1 Introduction

Answer Set Programming (ASP;[1]) is an appealing tool for knowledge representation and reasoning. Its attractiveness is supported by the availability of efficient off-the-shelf ASP-solvers that allow for computing answer sets of logic programs. In contrast to the related area of satisfiability checking (SAT), ASP lacks a formal framework for describing inferences conducted by ASP-solvers, such as the resolution proof theory underlying SAT-solvers [2].

To this end, we introduce a family of tableaux calculi [3] for ASP: A branch in a tableau corresponds to a successful or unsuccessful computation of an answer set. An entire tableau represents a traversal of the search space. Our approach furnishes declarative and fine-grained instrumentalities for characterizing operations as well as strategies of ASP-solvers. First, the granulation is detailed enough to capture the variety of propagation and choice operations in algorithms used for ASP; this also includes SAT-based approaches. Second, it is general enough to encompass the various strategies pursued by existing ASP-solvers, like *assat*, *cmodels*, *dlv*, *nomore++*, *smodels*, etc [4–8]. This provides us with a uniform framework for comparing existing solvers. Third, the approach is flexible enough to integrate new inference patterns, so to study their relation to existing ones. As a result, we obtain a new approach to computing unfounded sets by means of loops. Furthermore, it allows us to define a backward inference for unfounded set computation that appears to be the first of its kind. Finally, our approach allows us to investigate the proof complexity of ASP-solvers, depending on choice operations. In particular, we show that exponentially different best-case computations can be obtained for different ASP-solvers.

Related work. Our work is inspired by the one of Jarvisalo, Junttila, and Niemelä, who use tableaux methods in [9, 10] for investigating Boolean circuit satisfiability checking in the context of symbolic model checking. Although their target is different from ours, both approaches have many aspects in common. First, both use tableaux methods for characterizing DPLL-type techniques.[1] Second, using cut rules for characterizing DPLL-type split operations is the key idea for analyzing the proof complexity of different inference strategies. General investigations in propositional proof complexity, in particular, the one of satisfiability checking (SAT) can be found in [13, 14]. From the perspective of tableaux systems, DPLL is very similar to the propositional version of the KE tableaux calculus; both are closely related to weak

[1] *Davis-Putnam-Logemann-Loveland* (DPLL) techniques are introduced in [11, 12].

connection tableaux with atomic cut (as pointed out in [15]). Tableaux-based characterizations of logic programming are elaborated upon in [16]. Pearce, Guzmán, and Valverde provide in [17] a tableaux calculus for equilibrium logic based on its 5-valued semantics. Other tableaux approaches to nonmonotonic logics are summarized in [18]. Bonatti describes in [19] a resolution method for skeptical answer set programming. Operator-based characterizations of propagation and choice operations in ASP can be found in [7, 20, 21].

2 Answer Set Programming

Given an alphabet $\mathcal{P}$, a (normal) *logic program* is a finite set of rules of the form $p_0 \leftarrow p_1, \dots, p_m, \mathit{not}\ p_{m+1}, \dots, \mathit{not}\ p_n$, where $n \geq m \geq 0$ and each $p_i \in \mathcal{P}$ $(0 \leq i \leq n)$ is an *atom*. A *literal* is an atom p or its negation $\mathit{not}\ p$. For a rule r, let $\mathit{head}(r) = p_0$ be the *head* of r and $\mathit{body}(r) = \{p_1, \dots, p_m, \mathit{not}\ p_{m+1}, \dots, \mathit{not}\ p_n\}$ be the *body* of r. Furthermore, we let $\mathit{body}^+(r) = \{p_1, \dots, p_m\}$ and $\mathit{body}^-(r) = \{p_{m+1}, \dots, p_n\}$. The set of atoms occurring in a logic program Π is given by $\mathit{atom}(\Pi)$. The set of bodies in Π is $\mathit{body}(\Pi) = \{\mathit{body}(r) \mid r \in \Pi\}$. For regrouping rule bodies sharing the same head $p \in \mathit{atom}(\Pi)$, define $\mathit{body}(p) = \{\mathit{body}(r) \mid r \in \Pi, \mathit{head}(r) = p\}$.[2] A program Π is called *positive* if $\mathit{body}^-(r) = \emptyset$ for all $r \in \Pi$. $\mathit{Cn}(\Pi)$ denotes the smallest set of atoms closed under positive program Π. The *reduct*, Π^X, of Π relative to a set X of atoms is defined by $\Pi^X = \{\mathit{head}(r) \leftarrow \mathit{body}^+(r) \mid r \in \Pi,\ \mathit{body}^-(r) \cap X = \emptyset\}$. A set X of atoms is an *answer set* of a logic program Π if $\mathit{Cn}(\Pi^X) = X$. As an example, consider Program $\Pi_1 = \{a \leftarrow;\ c \leftarrow \mathit{not}\ b, \mathit{not}\ d;\ d \leftarrow a, \mathit{not}\ c\}$; it has two answer sets $\{a, c\}$ and $\{a, d\}$.

An *assignment* is a partial mapping of objects in a program Π into $\{\boldsymbol{T}, \boldsymbol{F}\}$, indicating whether a member of the *domain* of A, $\mathit{dom}(A)$, is true or false, respectively. In order to capture the whole spectrum of ASP-solving techniques, we fix $\mathit{dom}(A)$ to $\mathit{atom}(\Pi) \cup \mathit{body}(\Pi)$. We define $A^{\boldsymbol{T}} = \{v \in \mathit{dom}(A) \mid A(v) = \boldsymbol{T}\}$ and $A^{\boldsymbol{F}} = \{v \in \mathit{dom}(A) \mid A(v) = \boldsymbol{F}\}$. We also denote an assignment A by a set of signed objects: $\{\boldsymbol{T}v \mid v \in A^{\boldsymbol{T}}\} \cup \{\boldsymbol{F}v \mid v \in A^{\boldsymbol{F}}\}$. For instance with Π_1, the assignment mapping a to $\boldsymbol{T}$ and b to $\boldsymbol{F}$ is represented by $\{\boldsymbol{T}a, \boldsymbol{F}b\}$; c and d remain undefined. Following up this notation, we call an assignment *empty* if it leaves all objects undefined.

We define a set $U \subseteq \mathit{atom}(\Pi)$ as an *unfounded set* [22] of a program Π wrt a partial assignment A, if, for every rule $r \in \Pi$ such that $\mathit{head}(r) \in U$, either

$$(\mathit{body}^+(r) \cap A^{\boldsymbol{F}}) \cup (\mathit{body}^-(r) \cap A^{\boldsymbol{T}}) \neq \emptyset \tag{1}$$

$$\text{or } \mathit{body}^+(r) \cap U \neq \emptyset. \tag{2}$$

We define the *greatest unfounded set* of Π wrt A, denoted $\mathit{GUS}(\Pi, A)$, as the union of all unfounded sets of Π wrt A. *Loops* are sets of atoms that circularly depend upon each other in a program's positive atom dependency graph [4]. In analogy to external support [23] of loops, we define the *external bodies* of a loop L in Π as $\mathit{EB}(L) = \{\mathit{body}(r) \mid r \in \Pi, \mathit{head}(r) \in L, \mathit{body}^+(r) \cap L = \emptyset\}$. We denote the set of all loops in Π by $\mathit{loop}(\Pi)$. In Sections 4 and 6, we take a closer look on solvers and tableaux rules working on greatest unfounded sets or loops, respectively.

3 Tableaux calculi

We describe calculi for constructing answer sets from logic programs. Computations are characterized by binary trees called *tableaux* [3]. The nodes of the trees are (mainly) *signed propositions*, that is, propositions preceded by either $\boldsymbol{T}$ or $\boldsymbol{F}$, indicating an assumed truth value for the proposition. A *tableau* for a logic program Π and an initial assignment A is a binary tree such that the root node of the tree consists of the rules in Π and all members of A.[3] The other nodes in the tree are *entries* of the form $\boldsymbol{T}v$ or $\boldsymbol{F}v$, where $v \in \mathit{dom}(A)$,[4] generated by extending a tableau using the rules in Figure 1 in the following standard

[2] For convenience, we overload function *body* for denoting sets of associated bodies.

[3] Without set notation of A: $\boldsymbol{T}v$ (or $\boldsymbol{F}v$) for each $v \in \mathit{dom}(A)$ such that $A(v) = \boldsymbol{T}$ (or $A(v) = \boldsymbol{F}$).

[4] The *Cut* rule may, in principle, introduce more general entries; this would however necessitate additional decomposition rules, leading to an extended calculus.

$$\frac{\begin{array}{c} p \leftarrow l_1, \ldots, l_n \\ \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n \end{array}}{\boldsymbol{T}\{l_1, \ldots, l_n\}}$$

(a) Forward True Body (FTB)

$$\frac{\begin{array}{c} \boldsymbol{F}\{l_1, \ldots, l_i, \ldots, l_n\} \\ \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_{i-1}, \boldsymbol{t}l_{i+1}, \ldots, \boldsymbol{t}l_n \end{array}}{\boldsymbol{f}l_i}$$

(b) Backward False Body (BFB)

$$\frac{\begin{array}{c} p \leftarrow l_1, \ldots, l_n \\ \boldsymbol{T}\{l_1, \ldots, l_n\} \end{array}}{\boldsymbol{T}p}$$

(c) Forward True Atom (FTA)

$$\frac{\begin{array}{c} p \leftarrow l_1, \ldots, l_n \\ \boldsymbol{F}p \end{array}}{\boldsymbol{F}\{l_1, \ldots, l_n\}}$$

(d) Backward False Atom (BFA)

$$\frac{\begin{array}{c} p \leftarrow l_1, \ldots, l_i, \ldots, l_n \\ \boldsymbol{f}l_i \end{array}}{\boldsymbol{F}\{l_1, \ldots, l_i, \ldots, l_n\}}$$

(e) Forward False Body (FFB)

$$\frac{\boldsymbol{T}\{l_1, \ldots, l_i, \ldots, l_n\}}{\boldsymbol{t}l_i}$$

(f) Backward True Body (BTB)

$$\frac{\boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_m}{\boldsymbol{F}p}\,(\S)$$

(g) Forward False Atom (FFA)

$$\frac{\begin{array}{c} \boldsymbol{T}p \\ \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_{i-1}, \boldsymbol{F}B_{i+1}, \ldots, \boldsymbol{F}B_m \end{array}}{\boldsymbol{T}B_i}\,(\S)$$

(h) Backward True Atom (BTA)

$$\frac{\boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_m}{\boldsymbol{F}p}\,(\dagger)$$

(i) Well-Founded Negation (WFN)

$$\frac{\begin{array}{c} \boldsymbol{T}p \\ \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_{i-1}, \boldsymbol{F}B_{i+1}, \ldots, \boldsymbol{F}B_m \end{array}}{\boldsymbol{T}B_i}\,(\dagger)$$

(j) Well-Founded Justification (WFJ)

$$\frac{\boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_m}{\boldsymbol{F}p}\,(\ddagger)$$

(k) Forward Loop (FL)

$$\frac{\begin{array}{c} \boldsymbol{T}p \\ \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_{i-1}, \boldsymbol{F}B_{i+1}, \ldots, \boldsymbol{F}B_m \end{array}}{\boldsymbol{T}B_i}\,(\ddagger)$$

(l) Backward Loop (BL)

$$\frac{}{\boldsymbol{T}\phi \quad | \quad \boldsymbol{F}\phi}\,(\sharp[X])$$

(m) Cut (Cut$[X]$)

$(\S) : \mathit{body}(p) = \{B_1, \ldots, B_m\}$ $\quad (\dagger) : p \in \mathit{GUS}(\{r \in \Pi \mid \mathit{body}(r) \notin \{B_1, \ldots, B_m\}\}, \emptyset)$

$(\ddagger) : p \in L, L \in \mathit{loop}(\Pi),$ $EB(L) = \{B_1, \ldots, B_m\}$ $\quad (\sharp[X]) : \phi \in X$

Fig. 1. Tableaux rules for answer set programming.

way [3]: Given a tableaux rule and a branch in the tableau such that the prerequisites of the rule hold in the branch, the tableau can be extended by adding new entries to the end of the branch as specified by the rule. If the rule is the Cut rule in *(m)*, then entries $\boldsymbol{T}\phi$ and $\boldsymbol{F}\phi$ are added as the left and the right child to the end of the branch. For the other rules, the consequent of the rule is added to the end of the branch. The application of rules makes use of two conjugation functions, $\boldsymbol{t}$ and $\boldsymbol{f}$. For a literal l, define:

$$\boldsymbol{t}l = \begin{cases} \boldsymbol{T}l & \text{if } l \in \mathcal{P} \\ \boldsymbol{F}p & \text{if } l = not\ p \text{ for a } p \in \mathcal{P} \end{cases} \qquad \boldsymbol{f}l = \begin{cases} \boldsymbol{T}p & \text{if } l = not\ p \text{ for a } p \in \mathcal{P} \\ \boldsymbol{F}l & \text{if } l \in \mathcal{P} \end{cases}$$

Some rule applications are subject to provisos. (§) stipulates that $B_1, \ldots, B_m$ constitute all bodies of rules with head p; (†) requires that p belongs to the greatest unfounded set induced by all rules whose body is not among $B_1, \ldots, B_m$. (‡) makes sure that p belongs to a loop, all of whose external bodies are $B_1, \ldots, B_m$. Finally, $(\sharp[X])$ guides the application of the Cut rule in *(m)*, by restricting cut formulas to members of X. Different tableaux calculi are obtained from different rule sets. When needed this is made precise by enumerating the tableaux rules. Of particular interest are the following tableaux calculi:

$$\mathcal{T}_{comp} = \{\textit{(a)-(h)}, \textit{Cut}[atom(\Pi) \cup body(\Pi)]\}, \tag{3}$$

$$\mathcal{T}_{smodels} = \{\textit{(a)-(i)}, \textit{Cut}[atom(\Pi)]\}, \tag{4}$$

$$\mathcal{T}_{noMoRe} = \{\textit{(a)-(i)}, \textit{Cut}[body(\Pi)]\}, \tag{5}$$

$$\mathcal{T}_{nomore^{++}} = \{\textit{(a)-(i)}, \textit{Cut}[atom(\Pi) \cup body(\Pi)]\}. \tag{6}$$

An exemplary tableau of $\mathcal{T}_{smodels}$ is given in Figure 2. We indicate rule applications by either letters or

$$\begin{array}{cccc}
 & a \leftarrow & & \\
 & c \leftarrow not\ b, not\ d & & \\
 & d \leftarrow a, not\ c & & \\
 & \boldsymbol{T}\emptyset & & (a) \\
 & \boldsymbol{T}a & & (c) \\
 & \boldsymbol{F}b & & (g) \\
\boldsymbol{T}c & & \boldsymbol{F}c & (Cut[atom(\Pi)]) \\
\boldsymbol{T}\{not\ b, not\ d\} & (h) & \boldsymbol{F}\{not\ b, not\ d\} & (d) \\
\boldsymbol{F}d & (f) & \boldsymbol{T}d & (b) \\
\boldsymbol{F}\{a, not\ c\} & (e) & \boldsymbol{T}\{a, not\ c\} & (a)
\end{array}$$

Fig. 2. Tableau for Π_1 and the empty assignment.

rule names, like (a) or $(Cut[atom(\Pi)])$. Both branches comprise Π_1 along with a complete assignment for $atom(\Pi_1) \cup body(\Pi_1)$; the left one represents answer set $\{a, c\}$, the right one gives answer set $\{a, d\}$.

A branch in a tableau is *contradictory*, if it contains both $\boldsymbol{T}v$ and $\boldsymbol{F}v$ entries for some $v \in dom(A)$. A branch is *complete*, if it is contradictory, or if the branch contains either the entry $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for each $v \in dom(A)$ and is closed under all rules in a given calculus, except for the Cut rule in *(m)*. For instance, both branches in Figure 2 are complete and non-contradictory.

For each $v \in dom(A)$, we say that entry $\boldsymbol{T}v$ (or $\boldsymbol{F}v$) can be deduced by a set $\mathcal{R}$ of tableaux rules in a branch, if the entry $\boldsymbol{T}v$ (or $\boldsymbol{F}v$) can be generated from nodes in the branch by applying rules in $\mathcal{R}$ only. Note that every branch corresponds to a pair (Π, A) consisting of a program Π and an assignment A, and vice versa;[5] we draw on this relationship for identifying branches in the sequel. Accordingly, we let $T_{\mathcal{R}}(\Pi, A)$ denote the set of all entries deducible by rule set $\mathcal{R}$ in branch (Π, A). Moreover, $C_{\mathcal{R}}(\Pi, A)$ represents the set of all entries in the smallest branch extending (Π, A) and being closed under $\mathcal{R}$. When dealing with tableaux calculi, like $\mathcal{T}$, we slightly abuse notation and write $T_{\mathcal{T}}(\Pi, A)$ (or $C_{\mathcal{T}}(\Pi, A)$) instead of $T_{\mathcal{T}\setminus\{(m)\}}(\Pi, A)$ (or $C_{\mathcal{T}\setminus\{(m)\}}(\Pi, A)$), thus ignoring *Cut*. We mention that $C_{\{(a),(c),(e),(g)\}}(\Pi, A)$ corresponds to Fitting's operator [24]. Similarly, we detail in the subsequent sections that $C_{\{(a)\text{-}(h)\}}(\Pi, A)$ coincides with

[5] Given a branch (Π, A) in a tableau for Π and initial assignment A_0, we have $A_0 \subseteq A$.

unit propagation on a program's completion [25], $C_{\{(a),(c),(e),(g),(i)\}}(\Pi, A)$ amounts to propagation via well-founded semantics [22], and $C_{\{(a)\text{-}(i)\}}(\Pi, A)$ matches *smodels*' propagation, that is, well-founded semantics enhanced by backward propagation. In fact, all deterministic rules in Figure 1 are answer set preserving; this also applies to the Cut rule when considering both resulting branches.

A tableau is *complete* if all its branches are complete. A complete tableau is *closed* if all its branches are contradictory. A closed tableau for a program and the empty assignment is called a *refutation* for the program; it means that the program has no answer set, as exemplary shown next for *smodels*-type tableaux.

Theorem 1. *Let Π be a logic program and let $\emptyset$ denote the empty assignment. Then, the following holds for Tableaux Calculus $\mathcal{T}_{smodels}$:*

1. *Program Π has no answer set iff any complete tableau for Π and $\emptyset$ is closed.*
2. *Program Π has an answer set X iff some tableau for Π and $\emptyset$ has a complete and non-contradictory branch (Π, A) such that $X = A^{\boldsymbol{T}} \cap atom(\Pi)$.*

The same results are obtained for other tableaux calculi, like $\mathcal{T}_{noMoRe}$ and $\mathcal{T}_{nomore^{++}}$. Notably, all of them are sound and complete for ASP (as detailed in the full paper).

4 Characterizing existing ASP-solvers

In this section, we discuss the relation between our tableaux rules in Figure 1 and well-known ASP-solvers. As it turns out, our tableaux rules are well-suited for describing a wide variety of ASP-solvers. In particular, we cover all of the leading approaches to computing answer sets of normal logic programs. We start with SAT-based solvers, *assat* and *cmodels*, then go on with literal-based solvers, *smodels* and *dlv*, and end with *hybrid* solvers, working on literals as well as bodies.

SAT-based solvers. The basic idea of SAT-based solvers is to use some SAT-solver as model generator and to afterwards apply an unfounded set check to the generated model(s). In [4], it is shown that the answer sets of a normal logic program Π coincide with models of the propositional logic translation $Comp(\Pi) \cup LF(\Pi)$, where

$$\begin{aligned} Comp(\Pi) &= \{p \equiv (\textstyle\bigvee_{k=1\ldots m} \bigwedge_{l\in B_k} l) \mid p \in atom(\Pi), body(p) = \{B_1, \ldots, B_m\}\}, \\ LF(\Pi) &= \{\neg(\textstyle\bigvee_{k=1\ldots m} \bigwedge_{l\in B_k} l) \rightarrow \bigwedge_{p\in L} \neg p \mid L \in loop(\Pi), EB(L) = \{B_1, \ldots, B_m\}\}.^6 \end{aligned}$$

This translation constitutes the backbone of the SAT-based ASP-solvers *assat* [4] and *cmodels* [5]. Since $LF(\Pi)$ requires exponential space in the worst case [26], both *assat* and *cmodels* add loop formulas from $LF(\Pi)$ incrementally to $Comp(\Pi)$, whenever some model of $Comp(\Pi)$ not representing an answer set has been computed by the underlying SAT-solver. We first describe tableaux capturing the proceedings of the underlying SAT-solver and then go on with unfounded set checks.

Models of $Comp(\Pi)$ correspond to tableaux as follows.

Theorem 2. *Let Π be a logic program. Then, M is a model of $Comp(\Pi)$ iff there is a complete and non-contradictory branch (Π, A) in some tableau of $\mathcal{T}_{comp}$ such that $M = A^{\boldsymbol{T}} \cap atom(\Pi)$.*

Tableaux Rules *(a)-(h)* correspond to unit propagation on a program's completion. Note that *assat* and *cmodels* introduce propositional variables for bodies in order to obtain a polynomially-sized set of clauses equivalent to a program's completion [27]. Due to the fact that atoms and bodies are represented as propositional variables, allowing both of them as branching variables in $\mathcal{T}_{comp}$ (via $Cut[atom(\Pi) \cup body(\Pi)]$; cf. (3)) makes sense.

After a model of $Comp(\Pi)$ has been computed by the underlying SAT-solver, *assat* and *cmodels* apply an unfounded set check for deciding whether the computed model is an answer set.[7] If it fails, unfounded loops whose atoms are true (so-called *terminating loops* [4]) are determined and their loop formulas are added to the completion in order to eliminate the computed model of $Comp(\Pi)$. *assat*'s and *cmodels*' unfounded set check can be captured by Rules *FL* and *FFB* (*(k)* and *(e)* in Figure 1) as follows.

[6] Note that a default negated literal $not\ p$ is translated as $\neg p$.

[7] Note that every answer set of Π is a model of $Comp(\Pi)$, but not vice versa [28, 29].

Theorem 3. *Let Π be a logic program, let M be a model of $Comp(\Pi)$, and let $A = \{\boldsymbol{T}p \mid p \in M\} \cup \{\boldsymbol{F}p \mid p \in atom(\Pi) \setminus M\}$. Then, M is an answer set of Π iff $(T_{\{FL\}}(\Pi, T_{\{FFB\}}(\Pi, A)))^{\boldsymbol{F}} \cap M = \emptyset$.*

The drawback of SAT-based solvers is that, for deciding unsatisfiability, exponentially many loop formulas must be added to a program's completion in the worst case [26]. In view of Theorem 3, this means that, in order to close a tableau, exponentially many branches have to be completed by unfounded set checks.[8]

Literal-based solvers. We now describe the relation between *smodels* [8] and *dlv* [6, 20] on the one side and our tableaux rules on the other side. We first concentrate on characterizing *smodels* and then sketch how our characterization applies to *dlv* on normal logic programs.

Given that only literals are explicitly represented in *smodels*' assignments, whereas truth and falsity of bodies is determined implicitly, one might consider rewriting tableaux rules to work on literals only, thereby, restricting the domain of assignments to atoms. For instance, Rule *(g)* in Figure 1 would then turn into:

$$\frac{\boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_m}{\boldsymbol{F}p} \; (\{r \in \Pi \mid head(r) = p, body(r) \cap \{l_1, \ldots, l_m\} = \emptyset\} = \emptyset)$$

Observe that, in such a reformulation, we again refer to bodies by determining their values in the proviso associated with an inference rule. So reformulating tableaux rules to work on literals only complicates provisos and is not beneficial for practical considerations. In [30], additional variables for logic programs' rules are even explicitly introduced for comparing *smodels* with DPLL [11, 12], working on clauses.[9]

Propagation in *smodels* is accomplished by two functions, called $atleast$ and $atmost$ [8].[10] Roughly, the former computes deterministic consequences by applying forward and backward propagation (*(a)-(h)* in Figure 1), the latter falsifies greatest unfounded sets (WFN; *(i)* in Figure 1).

The following result captures propagation via $atleast$.

Theorem 4. *Let Π be a logic program and let A be an assignment such that $A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}} \subseteq atom(\Pi)$. Let $A_S = atleast(\Pi, A)$ and $A_T = C_{\mathcal{T}_{comp}}(\Pi, A)$. If $A_S^{\boldsymbol{T}} \cap A_S^{\boldsymbol{F}} \neq \emptyset$, then $A_T^{\boldsymbol{T}} \cap A_T^{\boldsymbol{F}} \neq \emptyset$; otherwise, we have $A_S \subseteq A_T$.*

Observe that function $atleast$ derives consequences of unit propagation on a program's completion ($C_{\mathcal{T}_{comp}}$). If $atleast$ derives a contradiction, so does $\mathcal{T}_{comp}$. Otherwise, $\mathcal{T}_{comp}$ derives at least as much as $atleast$. In addition, bodies' values are inferred, which might lead to inferring atoms' values not inferred by $atleast$, because of redundant representation of rules' bodies in *smodels*. (The same body can occur in several rules.) So $\mathcal{T}_{comp}$ does not fully comply with $atleast$ but approximates its behavior close enough for our purposes.

Propagation via $atmost$ (which returns the set of still potentially derivable atoms) is captured by WFN applied to bodies containing some false literal.

Theorem 5. *Let Π be a logic program and let A be an assignment such that $A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}} \subseteq atom(\Pi)$. We have $(T_{\{WFN\}}(\Pi, T_{\{FFB\}}(\Pi, A)))^{\boldsymbol{F}} \cup A^{\boldsymbol{F}} = atom(\Pi) \setminus atmost(\Pi, A)$.*

Note that *smodels* adds literals $\{not\ p \mid p \in atom(\Pi) \setminus atmost(\Pi, A)\}$ to an assignment A. If this leads to a contradiction, so does $T_{\{WFN\}}(\Pi, T_{\{FFB\}}(\Pi, A))$.

We have seen that *smodels*' propagation functions, $atleast$ and $atmost$, can be described by Tableaux Rules *(a)-(i)*. By adding Rule $Cut[atom(\Pi)]$, we thus get Tableaux Calculus $\mathcal{T}_{smodels}$ (cf. (4)). Note that *smodels*' lookahead can also be described by means of $Cut[atom(\Pi)]$: If lookahead yields a decision on

[8] Note that fully describing the interplay of model generation and unfounded set checks requires additional tableaux rules for dealing with explicitly represented loop formulas. Such loop formulas are added to a tableau in reaction to a failed unfounded set check.

[9] In fact, representing a program's completion without introducing additional variables for either rules or bodies does not properly reflect *smodels*' propagation. For instance, if $a \leftarrow b, c$ and $a \leftarrow b, d$ are the only rules with head atom a, *smodels* does not infer literal b from literal a. In contrast, clause $\neg a \vee b$ allows inferring literal b by unit propagation.

[10] Here, *atleast* and *atmost* are taken as defined on sets of signed propositions instead of literals, as in [8].

some atom p, we can extend a respective branch by the *Cut* rule applied to p, thereby, obtaining a tableau reflecting inferences achieved by lookahead (see full paper for details).

After having discussed *smodels*, we briefly turn to *dlv*: The inference rules applied by *dlv* [20] boil down to those of *smodels* on normal logic programs, so Tableaux Calculus $\mathcal{T}_{smodels}$ captures *dlv* as well (see full paper for details).[11]

Hybrid solvers. Finally, we discuss similarities and differences between literal-based ASP-solvers, *smodels* and *dlv*, and *hybrid* solvers, working on bodies in addition to atoms. Let us first mention that SAT-based solvers, *assat* and *cmodels*, are in a sense hybrid, since the CNF representation of a program's completion contains variables for bodies. Thus, underlying SAT-solvers can branch on both atoms and bodies (via $Cut[atom(\Pi) \cup body(\Pi)]$ in $\mathcal{T}_{comp}$). The only genuine ASP-solver we know of explicitly assigning values to atoms and bodies is *nomore++* [7].[12]

In [7], inference rules applied by *nomore++* are described in terms of operators: $\mathcal{P}$ for forward propagation, $\mathcal{B}$ for backward propagation, $\mathcal{U}$ for falsifying greatest unfounded sets, and $\mathcal{L}$ for lookahead. Similar to our tableaux rules, these operators apply to bodies in addition to atoms. As detailed in the full paper, we thus obtain direct correspondence between Rules *(a), (c), (e), (g)* and $\mathcal{P}$, Rules *(b), (d), (f), (h)* and $\mathcal{B}$, and Rule *(i)* and $\mathcal{U}$. Similarly to *smodels*' lookahead, inferences achieved by $\mathcal{L}$ can be described by means of $Cut[atom(\Pi) \cup body(\Pi)]$. So by replacing $Cut[atom(\Pi)]$ with $Cut[atom(\Pi) \cup body(\Pi)]$, we obtain Tableaux Calculus $\mathcal{T}_{nomore++}$ (cf. (6)) from $\mathcal{T}_{smodels}$.[13] In the next section, we show that this subtle difference, also observed on SAT-based solvers, has a great impact on proof complexity.

5 Proof complexity

We have seen that genuine ASP-solvers largely coincide on their inference rules and differ primarily in the usage of the Cut rule. We analyze in this section the relative efficiency of tableaux calculi with different Cut rules. Thereby, we take $\mathcal{T}_{smodels}$, $\mathcal{T}_{noMoRe}$, and $\mathcal{T}_{nomore++}$ into account, all using Rules *(a)-(i)* in Figure 1 but applying the Cut rule either to $atom(\Pi)$, $body(\Pi)$, or both of them (cf. (4), (5), and (6)).

For comparing different tableaux calculi, we use the notion of *proof complexity* [13, 14, 9]. That is, we measure the complexity of unsatisfiable logic programs, i.e. programs without answer sets, in terms of minimal refutations. Thereby, the size of a tableau is determined in the standard way as the number of nodes in it. A tableaux calculus $\mathcal{T}$ is not *polynomially simulated* [13, 14, 9] by another tableaux calculus $\mathcal{T}'$ if there is an infinite (witnessing) family $\{\Pi^n\}$ of unsatisfiable logic programs such that minimal refutations of $\mathcal{T}'$ for Π are asymptotically exponential in the size of minimal refutations of $\mathcal{T}$ for Π. A tableaux calculus $\mathcal{T}$ is *exponentially stronger* than a tableaux calculus $\mathcal{T}'$ if $\mathcal{T}$ polynomially simulates $\mathcal{T}'$, but not vice versa. Two tableaux calculi are *efficiency-incomparable* if neither one polynomially simulates the other. Note that proof complexity says nothing about how difficult it is to find a minimal refutation. Rather, it provides a lower bound on the run-time of proof-finding algorithms, independent from heuristic influences.

In what follows, we provide families of unsatisfiable logic programs witnessing that neither $\mathcal{T}_{smodels}$ polynomially simulates $\mathcal{T}_{noMoRe}$ nor vice versa. This means that, on certain instances, restricting the Cut rule to either only atoms or bodies leads to exponentially longer minimal run-times of either literal- or rule-based solvers in comparison to their counterparts, no matter which heuristic is applied. Due to space restrictions, we cannot provide proofs in this extended abstract. (Proofs are available at [32].) We however mention that family $\{\Pi^n_a \cup \Pi^n_c\}$ witnesses Lemma 1 and $\{\Pi^n_b \cup \Pi^n_c\}$ witnesses Lemma 2 (see Figure 3).

Lemma 1. *There is an infinite family $\{\Pi^n\}$ of logic programs such that*

1. *the size of minimal refutations of $\mathcal{T}_{noMoRe}$ is linear in n and*
2. *the size of minimal refutations of $\mathcal{T}_{smodels}$ is exponential in n.*

[11] Note that neither *smodels*' cardinality and weight constraints nor *dlv*'s aggregates are dealt with by rules of $\mathcal{T}_{smodels}$. For handling them, additional tableaux rules would be required.

[12] Complementing literal-based solvers, the *noMoRe* system [31] is rule-based (cf. $\mathcal{T}_{noMoRe}$ in (5)).

[13] In [7], also the restriction of the Cut rule to "supported bodies" is discussed. We however refer to the unrestricted Cut rule, $Cut[atom(\Pi) \cup body(\Pi)]$, here.

$$\Pi_a^n = \left\{ \begin{array}{l} x \leftarrow not\ x \\ x \leftarrow a_1, b_1 \\ \quad\vdots \\ x \leftarrow a_n, b_n \end{array} \right\} \quad \Pi_b^n = \left\{ \begin{array}{ll} \multicolumn{2}{c}{x \leftarrow c_1, \dots, c_n, not\ x} \\ c_1 \leftarrow a_1 & c_1 \leftarrow b_1 \\ \quad\vdots & \quad\vdots \\ c_n \leftarrow a_n & c_n \leftarrow b_n \end{array} \right\} \quad \Pi_c^n = \left\{ \begin{array}{c} a_1 \leftarrow not\ b_1 \\ b_1 \leftarrow not\ a_1 \\ \vdots \\ a_n \leftarrow not\ b_n \\ b_n \leftarrow not\ a_n \end{array} \right\}$$

Fig. 3. Families of programs $\{\Pi_a^n\}$, $\{\Pi_b^n\}$, and $\{\Pi_c^n\}$.

Lemma 2. *There is an infinite family $\{\Pi^n\}$ of logic programs such that*

1. *the size of minimal refutations of $\mathcal{T}_{smodels}$ is linear in n and*
2. *the size of minimal refutations of $\mathcal{T}_{noMoRe}$ is exponential in n.*

The next result follows immediately from Lemmas 1 and 2.

Theorem 6. *$\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are efficiency-incomparable.*

Given that any refutations of $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are as well refutations of $\mathcal{T}_{nomore^{++}}$, we have that $\mathcal{T}_{nomore^{++}}$ polynomially simulates both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$. So the following is an immediate consequence of Theorem 6.

Corollary 1. *$\mathcal{T}_{nomore^{++}}$ is exponentially stronger than both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$.*

The major implication of Corollary 1 is that, on certain logic programs, a priori restricting the Cut rule to either atoms or bodies leads to an exponentially greater search space to be traversed inevitably than with unrestricted Cut. Note that the phenomenon of exponentially worse proof complexity in comparison to $\mathcal{T}_{nomore^{++}}$ does not, depending on the instance, apply to one of $\mathcal{T}_{smodels}$ or $\mathcal{T}_{noMoRe}$ alone. Rather, combining families $\{\Pi_a^n\}$, $\{\Pi_b^n\}$, and $\{\Pi_c^n\}$ leads to a new family such that both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are exponentially worse than $\mathcal{T}_{nomore^{++}}$. So the unrestricted Cut rule is the only way to have at least the chance of finding a short refutation.

6 Unfounded sets

In the previous sections, we have analyzed inference rules and complexity of existing approaches to ASP-solving. We have seen that all approaches apply inference rules reflecting program completion (*(a)-(h)* in Figure 1). Inference mechanisms of SAT-based and genuine ASP-solvers differ only in the treatment of unfounded sets: The former apply unfounded set checks to total assignments only, whereas the latter incorporate unfounded set falsification (WFN; *(i)* in Figure 1) as an integral part of their inference structure. However, Rule WFN, as it is currently applied by genuine ASP-solvers, has several flaws:

1. It deals with greatest unfounded sets whose computation can be exhaustive.
2. It is asymmetrically applied, i.e. solvers apply no backward counterpart.
3. It is partly redundant, that is, it overlaps with completion-based Rule FFA (*(g)* in Figure 1) also falsifying atoms by forward propagation.

In what follows, we thus propose and discuss alternative approaches to unfounded set treatment, motivated by SAT-based solvers and results in [4]. Before we start, let us briefly introduce some vocabulary. Given two sets of tableaux rules, $\mathcal{R}$ and $\mathcal{R}'$, we say that $\mathcal{R}$ is *weaker* than $\mathcal{R}'$ if, for any branch (Π, A), we have $C_{\mathcal{R}}(\Pi, A) \subseteq C_{\mathcal{R}'}(\Pi, A)$. We say that $\mathcal{R}$ is *strictly weaker* than $\mathcal{R}'$ if $\mathcal{R}$ is weaker than $\mathcal{R}'$, but not vice versa. If $\mathcal{R}$ is weaker than $\mathcal{R}'$ and vice versa, then $\mathcal{R}$ and $\mathcal{R}'$ are *equally effective*. Finally, $\mathcal{R}$ and $\mathcal{R}'$ are *orthogonal* if they are not equally effective and neither one is weaker than the other.

We start with analyzing the relation between WFN and other rules falsifying atoms and bodies by forward propagation. Taking up *3.* above, we have the following result.

Proposition 1. *Set of rules $\{FFB, FFA\}$ is strictly weaker than $\{FFB, WFN\}$.*[14]

[14] We include FFB in both sets for falsifying bodies that positively depend on falsified atoms.

From Proposition 1, we have that, in presence of WFN, Rule FFA is actually not needed. However, all genuine ASP-solvers apply FFA as a sort of "local negation" and separately WFN as "global negation". Certainly, applying FFA is reasonable as applicability is easy to determine atom-wise. But with FFA at hand, Proposition 1 tells us that greatest unfounded sets are too unfocused for describing the sort of unfounded sets for which a dedicated inference rule is intrinsically necessary.

A characterization of WFN's effects, not built upon greatest unfounded sets, is obtained by putting results in [4] into the context of partial assignments.

Theorem 7. *Sets of rules* $\{FFB, WFN\}$ *and* $\{FFB, FFA, FL\}$ *are equally effective.*

By Theorem 7, one may safely substitute WFN by FFA and FL (*(k)* in Figure 1), falsifying unfounded loops, without forfeiting atoms that must be false due to the lack of (non-circular) support. SAT-based approaches, based on loop formulas, provide an explanation why concentrating on cyclic structures, namely loops, is sufficient: When falsity of unfounded atoms does not follow from a program's completion or FFA, respectively, then there is a loop all of whose external bodies are false. Such a loop (called *terminating loop* in [4]) is a (possibly strict) subset of a greatest unfounded set, so in reply to *1.* above, loop-oriented computations are less exhaustive.

Splitting up falsification of unfounded atoms into FFA for single atoms (aiming at the unfounded set condition in (1)) and FL for loops (aiming mainly at the unfounded set condition in (2)) implies that neither rule is weaker than the other.

Proposition 2. *Sets of rules* $\{FFB, FFA\}$ *and* $\{FFB, FL\}$ *are orthogonal.*

Having considered forward propagation for unfounded sets, we come to backward propagation, that is, WFJ and BL (*(j)* and *(l)* in Figure 1). Though no genuine ASP-solver currently applies such rules (as mentioned in *2.* above), they are answer set preserving.

Proposition 3. *Let* Π *be a logic program and let* A *be an assignment. Let* $B \in body(\Pi)$ *such that* $\boldsymbol{T}B \in T_{\{WFJ\}}(\Pi, A)$ *(or* $\boldsymbol{T}B \in T_{\{BL\}}(\Pi, A)$*).*

Then, we have that $(\Pi, A \cup T_{\{WFN\}}(\Pi, A \cup \{\boldsymbol{F}B\}))$ *(or* $(\Pi, A \cup T_{\{FL\}}(\Pi, A \cup \{\boldsymbol{F}B\}))$*) is contradictory.*

Both, WFJ and BL, merely make sure that falsifying some body does not lead to a conflict by applying their forward counterparts, WFN and FL. So any answer set agreeing with the current assignment also agrees with the result of applying either WFJ or BL.

A particularity of backward propagating true atoms' supports is that global Rule WFJ is stronger than the other both: BTA (*(h)* in Figure 1) applying to single atoms and BL applying to loops. However, WFJ is as unfocused as its forward counterpart WFN is, so we argue below that using BL (in combination with BTA) nonetheless makes sense.

Proposition 4. *Set of rules* $\{BTB, BTA, BL\}$ *is strictly weaker than* $\{BTB, WFJ\}$.

We conclude this section with discussing different alternatives to treat unfounded sets. First of all, let us mention that each of the proposed rules, namely WFJ, FL, and BL, is as complex to compute (i.e. linear[15]) as WFN. However, only the latter is currently applied by genuine ASP-solvers. Protecting true atoms from becoming unfounded (backward propagation) is as well a reasonable way of exploiting (potential) unfounded sets as falsifying unfounded atoms (forward propagation) is. The integration of respective inference rules in ASP-solvers would break asymmetry in unfounded set treatment, similar to Rules *(a)*, *(c)*, *(e)*, and *(g)*, each of which has a backward counterpart. As we have already mentioned, falsifying greatest unfounded sets is unfocused and partly overlaps with simpler Rule FFA. Thus, we would recommend Rules FL and BL for implementation. These rules have the advantage that they focus on loops, which is the class of unfounded sets that cannot be eliminated by program completion and must thus be handled separately. Also the concept of loop formulas, known from SAT-based solvers, puts application of FL and BL on a solid declarative footing, thereby, narrowing the gap between ASP- and SAT-solving.

[15] This is not to be confused with the (iterative) computation of a well-founded model, which is *quadratic*.

7 Discussion

A broad discussion is given in the full paper. Let us thus concentrate on two issues here:

The *Cut* rule is a powerful inference rule that has a major influence on proof complexity. However, it is well-known that an uncontrolled application of *Cut* is prone to inefficiency. The restriction of applying *Cut* to (sub)formula occurring in the input has already proven to be an effective way to "tame" the cut [3]. We followed this by investigating *Cut* applications to atoms and bodies occurring in a program.

The explicit integration of bodies into assignments has several benefits: First, it allows us to capture completion-based and hybrid systems in a closer fashion. Second, it allows us to reveal exponentially different proof complexities of ASP-solvers. Finally, even inferences in literal-based systems like *dlv* and *smodels* must take program rules into account, which is simulated through the corresponding bodies.

References

1. Baral, C.: Knowledge representation, reasoning and declarative problem solving with Answer sets. Cambridge University Press (2003)
2. Mitchell, D.: A SAT solver primer. Bulletin of the European Association for Theoretical Computer Science **85** (2005) 112–133
3. D'Agostino, M., Gabbay, D., Hähnle, R., Posegga, J., eds.: Handbook of Tableau Methods. Kluwer, Dordrecht (1999)
4. Lin, F., Zhao, Y.: Assat: computing answer sets of a logic program by sat solvers. Artificial Intelligence **157** (2004) 115–137
5. Lierler, Y., Maratea, M.: Cmodels-2: Sat-based answer sets solver enhanced to non-tight programs. In Lifschitz, V., Niemelä, I., eds.: Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04). Volume 2923 of Lecture Notes in Computer Science., Springer-Verlag (2004) 346–350
6. Leone, N., Faber, W., Pfeifer, G., Eiter, T., Gottlob, G., Koch, C., Mateis, C., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic (2005) To appear.
7. Anger, C., Gebser, M., Linke, T., Neumann, A., Schaub, T.: The `nomore++` approach to answer set solving. In Sutcliffe, G., Voronkov, A., eds.: Proceedings of the Twelfth International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'05). Volume 3835., Springer-Verlag (2005) 95–109
8. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138** (2002) 181–234
9. Jarvisalo, M., Junttila, T.A., Niemelä, I.: Unrestricted vs restricted cut in a tableau method for boolean circuits. In: AMAI. (2004)
10. Junttila, T.A., Niemelä, I.: Towards an efficient tableau method for boolean circuit satisfiability checking. In Lloyd, J.W., Dahl, V., Furbach, U., Kerber, M., Lau, K.K., Palamidessi, C., Pereira, L.M., Sagiv, Y., Stuckey, P.J., eds.: Computational Logic. Volume 1861 of Lecture Notes in Computer Science., Springer (2000) 553–567
11. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7** (1960) 201–215
12. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM **5** (1962) 394–397
13. Cook, S., Reckhow, R.: The relative efficiency of propositional proof systems. Journal of Symbolic Logic **44** (1979) 36–50
14. Beame, P., Pitassi, T.: Propositional proof complexity: Past, present, and future. Bulletin of the European Association for Theoretical Computer Science **65** (1998) 66–89
15. Hähnle, R.: Tableaux and related methods. In Robinson, J.A., Voronkov, A., eds.: Handbook of Automated Reasoning. Elsevier and MIT Press (2001) 100–178
16. Fitting, M.: Tableaux for logic programming. J. Autom. Reasoning **13** (1994) 175–188
17. Pearce, D., de Guzmán, I.P., Valverde, A.: A tableau calculus for equilibrium entailment. In Dyckhoff, R., ed.: TABLEAUX. Volume 1847 of Lecture Notes in Computer Science., Springer (2000) 352–367
18. Olivetti, N.: Tableaux for nonmonotonic logics. [3] 469–528
19. Bonatti, P.A.: Resolution for skeptical stable model semantics. J. Autom. Reasoning **27** (2001) 391–421
20. Faber, W.: Enhancing Efficiency and Expressiveness in Answer Set Programming Systems. Dissertation, Technische Universität Wien (2002)
21. Calimeri, F., Faber, W., Leone, N., Pfeifer, G.: Pruning operators for answer set programming systems. In Benferhat, S., Giunchiglia, E., eds.: Proceedings of the ninth International Workshop on Non-Monotonic Reasoning (NMR'04). (2002) 200–209

22. van Gelder, A., Ross, K., Schlipf, J.: The well-founded semantics for general logic programs. Journal of the ACM **38** (1991) 620–650
23. Lee, J.: A model-theoretic counterpart of loop formulas. In: Proceedings of the International Joint Conference on Artificial Intelligence. (2005)
24. Fitting, M.: Fixpoint semantics for logic programming: A survey. Theoretical Computer Science **278** (2002) 25–51
25. Clark, K.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and Data Bases. Plenum Press (1978) 293–322
26. Lifschitz, V., Razborov, A.: Why are there so many loop formulas? ACM Transactions on Computational Logic (To appear.)
27. Lierler, Y. (Personal communication)
28. Fages, F.: Consistency of clark's completion and the existence of stable models. Journal of Methods of Logic in Computer Science **1** (1994) 51–60
29. Erdem, E., Lifschitz, V.: Tight logic programs. Theory and Practice of Logic Programming **3** (2003) 499–518
30. Giunchiglia, E., Maratea, M.: On the relation between answer set and sat procedures (or, between cmodels and smodels). In Gabbrielli, M., Gupta, G., eds.: ICLP. Volume 3668 of Lecture Notes in Computer Science., Springer (2005) 37–51
31. Konczak, K., Linke, T., Schaub, T.: Graphs and colorings for answer set programming. Theory and Practice of Logic Programming (2005) To appear.
32. (http://www.cs.uni-potsdam.de/∼gebser/kr06proofs.pdf)