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 programming.

2 Answer Set Programming

Given an alphabet \( \mathcal{P} \), a (normal) logic program is a finite set of rules of the form \( p_0 \leftarrow p_1, \ldots, p_m, \neg p_{m+1}, \ldots, \neg p_n \), 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 \( \neg p \). For a rule \( r \), let \( \text{head}(r) = p_0 \) be the head of \( r \) and \( \text{body}(r) = \{p_1, \ldots, p_m, \neg p_{m+1}, \ldots, \neg p_n\} \) be the body of \( r \). Furthermore, we let \( \text{body}^+(r) = \{p_1, \ldots, p_m\} \) and \( \text{body}^-(r) = \{p_{m+1}, \ldots, p_n\} \). The set of atoms occurring in a logic program \( \Pi \) is given by \( \text{atom}(\Pi) \). The set of bodies in \( \Pi \) is \( \text{body}(\Pi) = \{\text{body}(r) \mid r \in \Pi\} \). For regrouping rule bodies sharing the same head \( p \in \text{atom}(\Pi) \), define \( \text{body}(p) = \{\text{body}(r) \mid r \in \Pi, \text{head}(r) = p\} \). A program \( \Pi \) is called positive if \( \text{body}^-(r) = \emptyset \) for all \( r \in \Pi \). \( \text{Can}(\Pi) \) denotes the smallest set of atoms closed under positive program \( \Pi \).

The reduct, \( \Pi_X \), of \( \Pi \) relative to a set \( X \) of atoms is defined by \( \Pi_X = \{\text{head}(r) \leftarrow \text{body}^+(r) \mid r \in \Pi, \text{body}^-(r) \cap X = \emptyset\} \). A set \( X \) of atoms is an answer set of a logic program \( \Pi \) if \( \text{Can}(\Pi) \subseteq X \). As an example, consider Program \( \Pi_1 = \{a \leftarrow c; c \leftarrow \neg b, \neg d; d \leftarrow a, \neg c\} \); it has two answer sets \( \{a, c\} \) and \( \{a, d\} \).

An assignment is a partial mapping of objects in a program \( \Pi \) into \( \{T, F\} \), indicating whether a member of the domain of \( A \), \( \text{dom}(A) \), is true or false, respectively. In order to capture the whole spectrum of ASP-solving techniques, we fix \( \text{dom}(A) \) to \( \text{atom}(\Pi) \cup \text{body}(\Pi) \). We define \( \tilde{A} = \{v \in \text{dom}(A) \mid A(v) = T\} \) and \( A^F = \{v \in \text{dom}(A) \mid A(v) = F\} \). We also denote an assignment \( A \) by a set of signed objects: \( \{T_v \mid v \in A^T\} \cup \{F_v \mid v \in A^F\} \). For instance with \( \Pi_1 \), the assignment mapping \( a \) to \( T \) and \( b \) to \( F \) is represented by \( \{Ta, Tb\} \); \( c \) and \( d \) remain undefined. Following up this notation, we call an assignment empty if it leaves all objects undefined.

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

\[
(\text{body}^+(r) \cap A^F) \cup (\text{body}^-(r) \cap A^T) \neq \emptyset
\]

or

\[
\text{body}^+(r) \cap U \neq \emptyset.
\]

We define the greatest unfounded set of \( \Pi \) wrt \( A \), denoted \( \text{GUS}(\Pi, A) \), as the union of all unfounded sets of \( \Pi \) 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 \( \Pi \) as \( \text{EB}(L) = \{\text{body}(r) \mid r \in \Pi, \text{head}(r) \in L, \text{body}^+(r) \cap L = \emptyset\} \). We denote the set of all loops in \( \Pi \) by \( \text{loop}(\Pi) \). 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 \( T \) or \( F \), indicating an assumed truth value for the proposition. A tableau for a logic program \( \Pi \) and an initial assignment \( A \) is a binary tree such that the root node of the tree consists of the rules in \( \Pi \) and all members of \( A \). The other nodes in the tree are entries of the form \( Tv \) or \( Fv \), where \( v \in \text{dom}(A) \), generated by extending a tableau using the rules in Figure 1 in the following standard

---

1 For convenience, we overload function \( \text{body} \) for denoting sets of associated bodies.
2 Without set notation of \( A: Tv \) (or \( Fv \)) for each \( v \in \text{dom}(A) \) such that \( A(v) = T \) (or \( A(v) = F \)).
3 The \textit{Cut} rule may, in principle, introduce more general entries; this would however necessitate additional decomposition rules, leading to an extended calculus.
Tableaux Calculi for Answer Set Programming

\( p \leftarrow l_1, \ldots, l_n \)
\( T[l_1, \ldots, l_n] \)
(a) Forward True Body (FTB)

\( p \leftarrow l_1, \ldots, l_n \)
\( F[l_1, \ldots, l_n] \)
(b) Backward False Body (BFB)

\( p \leftarrow l_1, \ldots, l_n \)
\( T[l_1, \ldots, l_n] \)
(c) Forward True Atom (FTA)

\( p \leftarrow l_1, \ldots, l_i, \ldots, l_n \)
\( F[l_1, \ldots, l_i, \ldots, l_n] \)
(d) Backward False Atom (BFA)

\( F_B_1, \ldots, F_B_m \)
\( F_L \)
(g) Forward False Atom (FFA)

\( F_B_1, \ldots, F_B_i, \ldots, F_B_m \)
\( F_L \)
(h) Backward True Atom (BTA)

\( F_B_1, \ldots, F_B_i, \ldots, F_B_m \)
\( T_B_i \)
(i) Well-Founded Negation (WFN)

\( F_B_1, \ldots, F_B_i, \ldots, F_B_m \)
\( T_B_i \)
(j) Well-Founded Justification (WFJ)

\( F_B_1, \ldots, F_B_i, \ldots, F_B_m \)
\( F_L \)
(k) Forward Loop (FL)

\( T \phi \quad F \phi \) (\( \phi \in X \))
(l) Backward Loop (BL)

\( (\$) \) : \( \text{body}(p) = \{ B_1, \ldots, B_m \} \)
\( (\dagger) \) : \( p \in \text{GUS}\{ \{ r \in II \mid \text{body}(r) \not\in \{ B_1, \ldots, B_m \}, \emptyset \} \}
\( (\ddagger) \) : \( p \in L, L \in \text{loop}(II) \),
\( E_B(L) = \{ B_1, \ldots, B_m \} \)
\( (\natural) \) : \( \phi \in X \)

\( \text{Fig. 1. Tableaux rules for answer set programming.} \)
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 \(T\phi\) and \(F\phi\) 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, \(t\) and \(f\). For a literal \(l\), define:

\[
\begin{align*}
  tl &= \begin{cases} 
  Tl & \text{if } l \in P \\
  Fl & \text{if } l = \not p \text{ for } p \in P 
  \end{cases} \\
  fl &= \begin{cases} 
  Tp & \text{if } l = \not p \text{ for } p \in P \\
  Fp & \text{if } l \in P 
  \end{cases}
\end{align*}
\]

Some rule applications are subject to provisos. (\(\xi\)) stipulates that \(B_1, \ldots, B_m\) constitute all bodies of rules with head \(p\); (\(\xi\)) requires that \(p\) belongs to the greatest unfounded set induced by all rules whose body is not among \(B_1, \ldots, B_m\). Finally, (\(\xi[X]\)) 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:

\[
\begin{align*}
  T_{comp} &= \{(a)-(h), Cut[atom(II) \cup body(II)]\}, \\
  T_{smodels} &= \{(a)-(i), Cut[atom(II)]\}, \\
  T_{compRe} &= \{(a)-(i), Cut[body(II)]\}, \\
  T_{nomore++} &= \{(a)-(i), Cut[atom(II) \cup body(II)]\}.
\end{align*}
\]

An exemplary tableau of \(T_{smodels}\) is given in Figure 2. We indicate rule applications by either letters or

\[
\begin{align*}
  &a \leftarrow \not b, \not d \\
  &d \leftarrow a, \not c \\
  &T\emptyset \quad (a) \\
  &Ta \quad (c) \\
  & Tb \quad (g) \\
  &T\{not b, not d\} \quad (h) \\
  &T\{not b, not d\} \quad (d) \\
  &F\{not b, not d\} \quad (f) \\
  &Fd \quad (b) \\
  &F\{a, not c\} \quad (e) \\
  &T\{a, not c\} \quad (a)
\end{align*}
\]

Fig. 2. Tableau for \(II_1\) and the empty assignment.

rule names, like \((a)\) or \((Cut[atom(II)])\). Both branches comprise \(II_1\) along with a complete assignment for \(atom(II_1) \cup body(II_1)\); 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 \(Tv\) and \(Fv\) entries for some \(v \in \text{dom}(A)\). A branch is complete, if it is contradictory, or if the branch contains either the entry \(Tv\) or \(Fv\) for each \(v \in \text{dom}(A)\) 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 \(v \in \text{dom}(A)\), we say that entry \(Tv\) (or \(Fv\)) can be deduced by a set \(R\) of tableaux rules in a branch, if the entry \(Tv\) (or \(Fv\)) can be generated from nodes in the branch by applying rules in \(R\) only. Note that every branch corresponds to a pair \((II, A)\) consisting of a program \(II\) and an assignment \(A\), and vice versa;\(^5\) we draw on this relationship for identifying branches in the sequel. Accordingly, we let \(T_R(II, A)\) denote the set of all entries deducible by rule set \(R\) in branch \((II, A)\). Moreover, \(C_R(II, A)\) represents the set of all entries in the smallest branch extending \((II, A)\) and being closed under \(R\). When dealing with tableaux calculi, like \(T\), we slightly abuse notation and write \(T_T(II, A)\) (or \(C_T(II, A)\)) instead of \(T^{(m)}(II, A)\) (or \(C^{(m)}(II, A)\)), thus ignoring \(Cut\). We mention that \(C^{(a,c)(c,d)(g)}(II, A)\) corresponds to Fitting’s operator [24]. Similarly, we detail in the subsequent sections that \(C^{(a,b)(h)}(II, A)\) coincides with

\(^5\) Given a branch \((II, A)\) in a tableau for \(II\) and initial assignment \(A_0\), we have \(A_0 \subseteq A\).
unit propagation on a program’s completion [25], \( C_{(a\cup c)\cup l(l)}(II, A) \) amounts to propagation via well-founded semantics [22], and \( C_{(a\cup c)}(II, A) \) 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 \( II \) be a logic program and let \( \emptyset \) denote the empty assignment. Then, the following holds for Tableaux Calculus \( \mathcal{T}_{\text{smodels}} \):

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

The same results are obtained for other tableaux calculi, like \( \mathcal{T}_{\text{noMoRe}} \) and \( \mathcal{T}_{\text{nomore++}} \). 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 tableau 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 \( II \) coincide with models of the propositional logic translation \( \text{Comp}(II) \cup \text{LF}(II) \), where

\[
\text{Comp}(II) = \{ p \equiv (\bigvee_{k=1}^{m} \wedge_{L \in B_k} l) \mid p \in \text{atom}(II), \text{body}(p) = \{B_1, \ldots, B_m\} \},
\]

\[
\text{LF}(II) = \{ \neg(\bigwedge_{k=1}^{m} \wedge_{L \in B_k} l) \rightarrow \bigwedge_{p \in L} \neg p \mid L \in \text{loop}(II), \text{EB}(L) = \{B_1, \ldots, B_m\} \}.^{6}
\]

This translation constitutes the backbone of the SAT-based ASP-solvers assat [4] and cmodels [5]. Since \( \text{LF}(II) \) requires exponential space in the worst case [26], both assat and cmodels add loop formulas from \( \text{LF}(II) \) incrementally to \( \text{Comp}(II) \), whenever some model of \( \text{Comp}(II) \) 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 \( \text{Comp}(II) \) correspond to tableaux as follows.

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

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 \( \mathcal{T}_{\text{comp}} \) (via \( \text{Cut}('\text{atom}(II) \cup \text{body}(II)) \)); cf. (3) makes sense.

After a model of \( \text{Comp}(II) \) 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 \( \text{Comp}(II) \). 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 \( \neg p \) is translated as \( \neg p \).

\(^7\) Note that every answer set of \( II \) is a model of \( \text{Comp}(II) \), but not vice versa [28, 29].
Theorem 3. Let \( \Pi \) be a logic program, let \( M \) be a model of \( \text{Comp}(\Pi) \), and let \( A = \{ Tp \mid p \in M \} \cup \{ Fp \mid p \in \text{atom}(\Pi) \setminus M \} \). Then, \( M \) is an answer set of \( \Pi \) iff \( (T_{\text{FL}}(\Pi, T_{\text{FFB}}(\Pi, A)))^F \cap M = \emptyset \).

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 \textit{smodels} [8] and \textit{dlv} [6, 20] on the one side and our tableaux rules on the other side. We first concentrate on characterizing \textit{smodels} and then sketch how our characterization applies to \textit{dlv} on normal logic programs.

Given that only literals are explicitly represented in \textit{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:

\[
\frac{f_1, \ldots, f_m}{Fp} \quad (\{ r \in \Pi \mid \text{head}(r) = p, \text{body}(r) \cap \{l_1, \ldots, l_m\} = \emptyset \} = \emptyset)
\]

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 how our characterization applies to \textit{dlv} on normal logic programs.

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

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

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

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

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

We have seen that \textit{smodels’} propagation functions, \textit{atleast} and \textit{atmost}, can be described by Tableaux Rules (a)-(i). By adding Rule \textit{Cut}(\text{atom}(\Pi)), we thus get Tableaux Calculus \( T_{\text{smodels}} \) (cf. (4)). Note that \textit{smodels’} lookahead can also be described by means of \textit{Cut}(\text{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 rules or bodies does not properly reflect \textit{smodels’} propagation. For instance, if \( a \leftarrow b, c \) and \( a \leftarrow b, d \) are the only rules with head atom \( a \), \textit{smodels} does not infer literal \( b \) from literal \( a \). In contrast, clause \( \neg a \lor b \) allows inferring literal \( b \) by unit propagation.

\(^{10}\) Here, \textit{atleast} and \textit{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 \( T_{\text{smodels}} \) 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(\( I \)) \( \cup \) body(\( I \))] in \( T_{\text{comp}} \)). 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(\( I \)) \( \cup \) body(\( I \))]. So by replacing Cut[atom(\( I \))] with Cut[atom(\( I \)) \( \cup \) body(\( I \))], we obtain Tableaux Calculus \( T_{\text{nomore++}} \) (cf. (6)) from \( T_{\text{smodels}} \).\(^{13}\) 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 \( T_{\text{smodels}} \), \( T_{\text{noMoRe}} \), and \( T_{\text{nomore++}} \) into account, all using Rules (a)-(i) in Figure 1 but applying the Cut rule either to atom(\( I \)), body(\( I \)), 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 \( T \) is not polynomially simulated [13, 14, 9] by another tableaux calculus \( T' \) if there is an infinite (witnessing) family \( \{ P_n \} \) of unsatisfiable logic programs such that minimal refutations of \( T' \) for \( I \) are asymptotically exponential in the size of minimal refutations of \( T \) for \( I \). A tableaux calculus \( T \) is exponentially stronger than a tableaux calculus \( T' \) if \( T \) polynomially simulates \( T' \), 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 \( T_{\text{smodels}} \) nor \( T_{\text{noMoRe}} \) 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 \( \{ \Pi^a_n \cup \Pi^b_n \} \) witnesses Lemma 1 and \( \{ \Pi^a_n \cup \Pi^c_n \} \) witnesses Lemma 2 (see Figure 3).

Lemma 1. There is an infinite family \( \{ \Pi^a_n \} \) of logic programs such that

1. the size of minimal refutations of \( T_{\text{noMoRe}} \) is linear in \( n \) and
2. the size of minimal refutations of \( T_{\text{smodels}} \) is exponential in \( n \).

\(^{11}\) Note that neither smodels’ cardinality and weight constraints nor dlv’s aggregates are dealt with by rules of \( T_{\text{smodels}} \). For handling them, additional tableaux rules would be required.

\(^{12}\) Complementing literal-based solvers, the noMoRe system [31] is rule-based (cf. \( T_{\text{noMoRe}} \) 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(\( I \)) \( \cup \) body(\( I \))], here.
Lemma 2. There is an infinite family \( \{ \Pi^n \} \) of logic programs such that

1. the size of minimal refutations of \( T_{\text{smolds}} \) is linear in \( n \) and
2. the size of minimal refutations of \( T_{\text{noMoRe}} \) is exponential in \( n \).

The next result follows immediately from Lemmas 1 and 2.

Theorem 6. \( T_{\text{smolds}} \) and \( T_{\text{noMoRe}} \) are efficiency-incomparable.

Given that any refutations of \( T_{\text{smolds}} \) and \( T_{\text{noMoRe}} \) are as well refutations of \( T_{\text{nomore}++} \), we have that \( T_{\text{nomore}++} \) polynomially simulates both \( T_{\text{smolds}} \) and \( T_{\text{noMoRe}} \). So the following is an immediate consequence of Theorem 6.

Corollary 1. \( T_{\text{nomore}++} \) is exponentially stronger than both \( T_{\text{smolds}} \) and \( T_{\text{noMoRe}} \).

The major implication of Corollary 1 is that, on certain logic programs, a priori restricting the \textit{Cut} rule to either atoms or bodies leads to an exponentially greater search space to be traversed inevitably than with unrestricted \textit{Cut}. Note that the phenomenon of exponentially worse proof complexity in comparison to \( T_{\text{nomore}++} \) does not, depending on the instance, apply to one of \( T_{\text{smolds}} \) or \( T_{\text{noMoRe}} \) alone. Rather, combining families \( \{ \Pi^a_n \} \), \( \{ \Pi^b_n \} \), and \( \{ \Pi^n_c \} \) leads to a new family such that both \( T_{\text{smolds}} \) and \( T_{\text{noMoRe}} \) are exponentially worse than \( T_{\text{nomore}++} \). So the unrestricted \textit{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 overlays 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 \textit{weaker} than \( \mathcal{R}' \) if, for any branch \( (\Pi, A) \), we have \( C_{\mathcal{R}}(\Pi, A) \subseteq C_{\mathcal{R}'}(\Pi, A) \). We say that \( \mathcal{R} \) is \textit{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 \textit{equally effective}. Finally, \( \mathcal{R} \) and \( \mathcal{R}' \) are \textit{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 \( \{ \text{FFB}, \text{FFA} \} \) is strictly weaker than \( \{ \text{FFB}, \text{WFN} \} \).\(^{14}\)

\(^{14}\) We include FFB in both sets for falsifying bodies that positively depend on falsified atoms.

Figure 3. Families of programs \( \{ \Pi^a_n \}, \{ \Pi^b_n \}, \) and \( \{ \Pi^n_c \} \).
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 I. 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 \( \Pi \) be a logic program and let \( A \) be an assignment. Let \( B \in \text{body}(\Pi) \) such that \( T_B \in T_{\{WFJ\}}(\Pi, A) \) (or \( T_B \in T_{\{BL\}}(\Pi, A) \)).

Then, we have that \( (\Pi, A \cup T_{\{WFN\}}(\Pi, A \cup \{FB\})) \) (or \( (\Pi, A \cup T_{\{FL\}}(\Pi, A \cup \{FB\})) \)) 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: $BTA$ ((b) 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 formulas, 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

26. Lifschitz, V., Razborov, A.: Why are there so many loop formulas? ACM Transactions on Computational Logic (To appear.)
27. Lierler, Y. (Personal communication)
32. (http://www.cs.uni-potsdam.de/~gebser/kr06proofs.pdf)