Tableau Calculi for Logic Programs under Answer Set Semantics

Martin Gebser and Torsten Schaub[1]
Institut für Informatik
Universität Potsdam
August-Bebel-Str. 89, D-14482 Potsdam
{gebser,torsten}@cs.uni-potsdam.de

We introduce formal proof systems based on tableau methods for analyzing computations in Answer Set Programming (ASP). Our approach furnishes fine-grained instruments for characterizing operations as well as strategies of ASP solvers. The granularity is detailed enough to capture a variety of propagation and choice methods of algorithms used for ASP solving, also incorporating SAT-based and conflict-driven learning approaches to some extent. This provides us with a uniform setting for identifying and comparing fundamental properties of ASP solving approaches. In particular, we investigate their proof complexities and show that the run-times of best-case computations can vary exponentially between different existing ASP solvers. Apart from providing a framework for comparing ASP solving approaches, our characterizations also contribute to their understanding by pinning down the constitutive atomic operations. Furthermore, our framework is flexible enough to integrate new inference patterns, and so to study their relation to existing ones. To this end, we generalize our approach and provide an extensible basis aiming at a modular incorporation of additional language constructs. This is exemplified by augmenting our basic tableau methods with cardinality constraints and disjunctions.

Categories and Subject Descriptors: D.1.6 [**Logic Programming**]:

General Terms: Theory

Additional Key Words and Phrases: Answer Set Programming, Tableau calculi, Proof complexity

1. INTRODUCTION

Answer Set Programming (ASP; [Baral 2003]) is an appealing tool for knowledge representation and reasoning. Its attractiveness is supported by the availability of efficient off-the-shelf solvers that allow for computing answer sets of logic programs. However, in contrast to the neighboring area of Satisfiability (SAT; [Biere et al. 2009]) checking, where solving approaches can be analyzed by means of resolution proof theory (cf. [Beame and Pitassi 1998; Beame et al. 2004; Pipatsrisawat and Darwiche 2011]), ASP lacks formal frameworks for describing inferences con-

[1] Affiliated with the School of Computing Science at Simon Fraser University, Burnaby, Canada, and the Institute for Integrated and Intelligent Systems at Griffith University, Brisbane, Australia.

This paper combines and extends [Gebser and Schaub 2006b; 2007]; the former received the best paper award at ICLP'06.

ducted by ASP solvers. This has led to great heterogeneity in the description of algorithms for ASP solving, ranging over operational [Faber 2002; Simons et al. 2002; Anger et al. 2005; Calimeri et al. 2006; Konczak et al. 2006] and procedural [Lin and Zhao 2004; Ward and Schlipf 2004; Giunchiglia et al. 2006; Lin et al. 2006; Gebser et al. 2007; Giunchiglia et al. 2008] characterizations, which complicates identifying fundamental properties of algorithms (such as soundness and completeness) as well as formal comparisons between them.

Our work is motivated by the desire to converge the various heterogeneous characterizations of current ASP solvers' inferences by developing common proof-theoretic foundations. Examples for this are the *Davis-Putnam-Logemann-Loveland* (DPLL; [Davis and Putnam 1960; Davis et al. 1962]) procedure and *Conflict-Driven Clause Learning* (CDCL; [Marques-Silva and Sakallah 1999; Moskewicz et al. 2001; Eén and Sörensson 2004]) for SAT, which are both based on resolution proof theory. As in the context of SAT, the proof-theoretic perspective allows us to identify general, rather than solver-specific, properties and to study inferences by their admissibility, rather than from an implementation point of view. To this end, we introduce a family of tableau calculi (cf. [D'Agostino et al. 1999]) for ASP, viewing answer set computations as derivations in an inference system: a branch in a tableau corresponds to a successful or unsuccessful attempt to compute an answer set; an entire tableau represents a traversal of the search space.

Our approach furnishes fine-grained instruments for characterizing propagation operations as well as search strategies of ASP solvers. In particular, relating the approaches of ASP solvers to appropriate tableau calculi, in the sense that computations of a solver correspond to tableaux of an associated calculus, allows us to analyze the (best-case) complexities of ASP solving strategies. In fact, we investigate the proof complexities of different approaches, depending on choice operations. It turns out that exponentially different run-times of best-case computations can be obtained for existing ASP solvers. Furthermore, our proof-theoretic frameworks allow us to describe and study new inference patterns, going beyond implemented systems. As a result, we obtain a loop-oriented approach to unfounded set handling, which is not restricted to SAT-based solvers, and we also identify backward propagation operations for unfounded sets.

While our basic tableau framework mainly aims at characterizing inference patterns of solvers on the common language fragment of (propositional) normal programs, in practice, ASP solvers additionally support composite language constructs, such as *dlv*'s aggregates [Faber et al. 2008] or *smodels*' cardinality and weight constraints [Simons et al. 2002]. To address these extensions, too, we generalize our basic approach towards a flexible framework amenable to additional language constructs. For simplicity, we begin with characterizing inferences in a generic core fragment in which rule heads and bodies consist of atomic literals. We then gradually extend this setting by focusing on particular *aggregates*, understood as expressions over collections of literals. To this end, we first view conjunctions in rule bodies as simple Boolean aggregates. Embedding them into our generic framework, we characterize a class of logic programs that is as expressive as normal programs (already dealt with in our basic approach). We further augment our generic framework with cardinality constraints as well as disjunctions in rule heads. The integration of these popular language constructs gives an idea of how to adapt our proof-theoretic approach to richer settings.

The remainder of this paper is organized as follows. After providing the necessary background on (normal) logic programs, Section 3 introduces our basic tableau framework for them. In Section 4, we apply our methodology to characterize well-known logic programming operators and existing ASP solvers, investigating both traditional (DPLL-style) approaches as well as those based on translation to SAT or natively supporting conflict-driven learning. Section 5 shifts the focus to logic programs over extended languages, for which we first provide a generic core frame-

work that is afterwards extended to conjunctions, cardinality constraints, and disjunctions. In Section 6, we analyze the relative power of several practically relevant tableau calculi in terms of (best-case) proof complexity. Section 7 and 8 conclude the paper by surveying related work and discussing the achieved results, respectively. Finally, proofs are provided in the appendix.

2. NORMAL LOGIC PROGRAMS

Given an alphabet $\mathcal{P}$, a *normal (logic) program* Π is a finite set of rules of the form

$$p_0 \leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n \tag{1}$$

where $0 \leq m \leq n$ and each $p_i \in \mathcal{P}$ is an *atom* for $0 \leq i \leq n$. A *literal* is an atom p or its (default) negation $not\ p$. For a rule r as in (1), let $head(r) = p_0$ be the *head* of r and $body(r) = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$ be the *body* of r. We sometimes write a rule r as $(head(r) \leftarrow body(r))$, where $body(r)$ represents the conjunction of its contained literals. For a set B of literals, let $B^+ = B \cap \mathcal{P}$ and $B^- = \{p \in \mathcal{P} \mid not\ p \in B\}$; accordingly, for $body(r) = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$, we get $body(r)^+ = \{p_1, \ldots, p_m\}$ and $body(r)^- = \{p_{m+1}, \ldots, p_n\}$. We denote the set of atoms occurring in a program Π by $atom(\Pi)$, and the set of bodies occurring in Π is given by $body(\Pi) = \{body(r) \mid r \in \Pi\}$. For regrouping rule bodies with the same head p, let $body(p) = \{body(r) \mid r \in \Pi, head(r) = p\}$.[2] Given a program Π such that $body(r)^- = \emptyset$ for all $r \in \Pi$, a set X of atoms is closed under Π if, for every $r \in \Pi$, $head(r) \in X$ or $body(r)^+ \not\subseteq X$; $Cn(\Pi)$ denotes the (unique) smallest set of atoms closed under Π. The *reduct* of a normal program Π relative to a set X of atoms is defined by $\Pi^X = \{head(r) \leftarrow body(r)^+ \mid r \in \Pi,\ body(r)^- \cap X = \emptyset\}$. A set X of atoms is an *answer set* of a normal program Π if $Cn(\Pi^X) = X$. As an example, consider the program

$$\Pi_1 = \{a \leftarrow;\ c \leftarrow not\ b, not\ d;\ d \leftarrow a, not\ c\}$$

and its two answer sets $\{a, c\}$ and $\{a, d\}$.

A (partial) *assignment* $\mathbf{A}$ over a *domain*, $dom(\mathbf{A})$, is a set of *entries* of the form $\boldsymbol{T}v$ or $\boldsymbol{F}v$, indicating whether some $v \in dom(\mathbf{A})$ is true or false, respectively. The complement of an entry ℓ is denoted by $\overline{\ell}$, that is, $\overline{\boldsymbol{T}v} = \boldsymbol{F}v$ and $\overline{\boldsymbol{F}v} = \boldsymbol{T}v$. Letting $\mathbf{A}^{\boldsymbol{T}} = \{v \in dom(\mathbf{A}) \mid \boldsymbol{T}v \in \mathbf{A}\}$ and $\mathbf{A}^{\boldsymbol{F}} = \{v \in dom(\mathbf{A}) \mid \boldsymbol{F}v \in \mathbf{A}\}$, we say that $\mathbf{A}$ is *contradictory* if $\mathbf{A}^{\boldsymbol{T}} \cap \mathbf{A}^{\boldsymbol{F}} \neq \emptyset$; otherwise, $\mathbf{A}$ is *non-contradictory*. Furthermore, $\mathbf{A}$ is *total* if it is non-contradictory and $\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}} = dom(\mathbf{A})$. For a normal program Π, we fix $dom(\mathbf{A})$ to $atom(\Pi) \cup body(\Pi)$ in the sequel. For instance, with Π_1, the (partial) assignment $\{\boldsymbol{T}\emptyset, \boldsymbol{F}b\}$ maps the body $\emptyset$ of rule $(a \leftarrow)$ to true and the atom b to false, and the assignment $\{\boldsymbol{T}\emptyset, \boldsymbol{F}b, \boldsymbol{T}a, \boldsymbol{T}c, \boldsymbol{F}\{a, not\ c\}, \boldsymbol{F}d, \boldsymbol{T}\{not\ b, not\ d\}\}$ is total.

For a normal program Π and a set $U \subseteq atom(\Pi)$, we define the *external bodies* of U for Π by $EB_\Pi(U) = \{body(r) \mid r \in \Pi, head(r) \in U, body(r)^+ \cap U = \emptyset\}$. We call U an *unfounded set* of Π wrt an assignment $\mathbf{A}$ if $EB_\Pi(U) \subseteq \mathbf{A}^{\boldsymbol{F}}$.[3] Since the union of two unfounded sets remains unfounded, the union of all unfounded sets of Π wrt $\mathbf{A}$, denoted by $\mathcal{U}_\Pi(\mathbf{A})$, is the *greatest unfounded set* of Π wrt $\mathbf{A}$. Semantically, an unfounded set identifies atoms that lack external support (cf. [Lee 2005]) and thus ought to be false. The (positive) *dependency graph* of Π is $(atom(\Pi), \{(head(r), p) \mid r \in \Pi, p \in body(r)^+\})$. A non-empty $U \subseteq atom(\Pi)$ is a *loop* [Lin and Zhao 2004] of Π if, for every pair $p, q \in U$ (including $p = q$), there is a path of non-zero length from p to q in the dependency graph of Π such that all vertices in the path belong to U.

[2] For convenience, we overload function *body* to refer to sets of bodies associated with a program or an atom, respectively.
[3] Original unfounded sets [Van Gelder et al. 1991] rely on $EB_\Pi(U) \subseteq \{B \in body(\Pi) \mid (B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\}$.

We denote the set of all loops of Π by $loop(\Pi)$. The main interest of loops is that they can be unfounded even if each contained atom is (circularly) supported by some rule with non-false body.

3. TABLEAUX FOR NORMAL LOGIC PROGRAMS

We describe calculi consisting of tableau rules for the construction of answer sets of logic programs. A *tableau* for a logic program Π and an initial assignment $\mathbf{A}$ is a binary tree with the rules of Π and the entries of $\mathbf{A}$ at its root.[4] Further entries can be generated by applying tableau rules in the following standard way [D'Agostino et al. 1999]: given a tableau rule and a branch in a tableau such that the prerequisites of the rule hold in the branch, the tableau can be extended by appending entries to the end of the branch as specified by the rule. Note that every *branch* corresponds to a pair $(\Pi, \mathbf{A})$; we draw on this relationship for identifying branches in the sequel. For some $v \in dom(\mathbf{A})$, we say that $\boldsymbol{T}v$ or $\boldsymbol{F}v$ can be deduced by a set $\mathcal{T}$ of tableau rules in a branch $(\Pi, \mathbf{A})$ if the entry can be generated by applying some rule in $\mathcal{T}$ other than ***Cut*** (see below). We let $D_{\mathcal{T}}(\Pi, \mathbf{A})$ denote the set of entries deducible by $\mathcal{T}$ in $(\Pi, \mathbf{A})$; $D^*_{\mathcal{T}}(\Pi, \mathbf{A})$ represents the set of entries in a smallest branch that extends $(\Pi, \mathbf{A})$ and is closed under $\mathcal{T}$, that is, $D_{\mathcal{T}}(\Pi, D^*_{\mathcal{T}}(\Pi, \mathbf{A})) \subseteq D^*_{\mathcal{T}}(\Pi, \mathbf{A})$. A branch $(\Pi, \mathbf{A})$ is *contradictory* if $\mathbf{A}$ is contradictory, and *non-contradictory* otherwise; $(\Pi, \mathbf{A})$ is *complete* (wrt a tableau calculus $\mathcal{T}$) if it is contradictory or if $\mathbf{A}$ is total and $D_{\mathcal{T}}(\Pi, \mathbf{A}) \subseteq \mathbf{A}$. A tableau is *complete* if all of its branches are complete. A complete tableau for a logic program and the empty assignment such that all branches are contradictory is called a *refutation* for the program (meaning that the program has no answer set).

Our tableau rules for normal programs Π are shown in Figure 1. For convenience, they make use of two conjugation functions, $\boldsymbol{t}$ and $\boldsymbol{f}$. For a literal l, define:

$$\boldsymbol{t}l = \begin{cases} \boldsymbol{T}l & \text{if } l \in dom(\mathbf{A}) \\ \boldsymbol{F}v & \text{if } l = not\ v \text{ for } v \in dom(\mathbf{A}) \end{cases} \qquad \boldsymbol{f}l = \begin{cases} \boldsymbol{F}l & \text{if } l \in dom(\mathbf{A}) \\ \boldsymbol{T}v & \text{if } l = not\ v \text{ for } v \in dom(\mathbf{A}) \end{cases}$$

In view of this, the *FTB* rule in *(a)* expresses that truth of a rule body can be deduced if the body's literals hold in a branch. Conversely, if the body is already assigned to false and all but one literal hold, the remaining literal must necessarily be false; this contrapositive argument is formalized by the *BFB* rule in *(b)*. Likewise, the tableau rules *FTA* and *FFB* in *(c)* and *(e)* capture straightforward conditions under which an atom must be assigned to true or a body to false, respectively. Their contrapositives are given by *BFA* and *BTB* in *(d)* and *(f)*. The remaining tableau rules in *(g)–(k)* are subject to provisos. For an application of *FFA* in *(g)*, deducing an unsupported atom p to be false, $(\S)$ stipulates that $B_1, \ldots, B_m$ comprise all bodies of rules with head p. Its contrapositive, the *BTA* rule in *(h)*, is also guided by $(\S)$. The outer structure of *WFN*$[\Omega]$ and *WFJ*$[\Omega]$ in *(i)* and *(j)*, aiming at unfounded sets, is similar to *FFA* and *BTA*, yet their proviso $(\dagger[\Omega])$ requires a concerned atom p to belong to some set $U \in \Omega$ such that $B_1, \ldots, B_m$ comprise all external bodies of U for Π. We below investigate two alternative options for Ω: $\Omega = 2^{atom(\Pi)}$ and $\Omega = loop(\Pi)$. Finally, $(\sharp[\Gamma])$ guides applications of the ***Cut***$[\Gamma]$ rule in *(k)* by restricting cut objects v to members of Γ. For a normal program Π, we below consider $\Gamma = atom(\Pi)$, $\Gamma = body(\Pi)$, and $\Gamma = atom(\Pi) \cup body(\Pi)$.[5] Note that a ***Cut*** application adds entries $\boldsymbol{T}v$ and $\boldsymbol{F}v$ as the left and the right child to the end of a branch, thus reflecting non-determinism in assigning v. With every other tableau rule, its consequent is appended to a branch, i.e., applications are deterministic.

The deterministic tableau rules in Figure 1 preserve answer sets; this also applies to ***Cut*** when considering both resulting branches. Different tableau calculi, viz. particular rule sets, yield char-

[4] We do not mark the immanent validity of rules in Π by $\boldsymbol{T}$, as rules are not assigned by $\mathbf{A}$.

[5] The *Cut* rule may, in principle, introduce more general structures, which would necessitate further decomposition rules.

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n}{\boldsymbol{T}\{l_1, \ldots, l_n\}}$$

(a) Forward True Body (FTB)

$$\frac{\boldsymbol{F}\{l_1, \ldots, l_{i-1}, l_i, l_{i+1}, \ldots, l_n\} \quad \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_{i-1}, \boldsymbol{t}l_{i+1}, \ldots, \boldsymbol{t}l_n}{\boldsymbol{f}l_i}$$

(b) Backward False Body (BFB)

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \boldsymbol{T}\{l_1, \ldots, l_n\}}{\boldsymbol{T}p}$$

(c) Forward True Atom (FTA)

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \boldsymbol{F}p}{\boldsymbol{F}\{l_1, \ldots, l_n\}}$$

(d) Backward False Atom (BFA)

$$\frac{p \leftarrow l_1, \ldots, l_i, \ldots, l_n \quad \boldsymbol{f}l_i}{\boldsymbol{F}\{l_1, \ldots, l_i, \ldots, l_n\}}$$

(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)

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

(h) Backward True Atom (BTA)

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

(i) Well-Founded Negation (WFN[Ω])

$$\frac{\boldsymbol{T}p \quad \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_{i-1}, \boldsymbol{F}B_{i+1}, \ldots, \boldsymbol{F}B_m}{\boldsymbol{T}B_i}\,(\dagger[\Omega])$$

(j) Well-Founded Justification (WFJ[Ω])

$$\frac{}{\boldsymbol{T}v \;\mid\; \boldsymbol{F}v}\,(\sharp[\Gamma])$$

(k) Cut (Cut[Γ])

$$\begin{array}{rl} (\S) & : \; p \in \mathit{atom}(\Pi),\;\; \mathit{body}(p) \subseteq \{B_1, \ldots, B_m\} \subseteq \mathit{body}(\Pi) \\ (\dagger[\Omega]) & : \; p \in U, U \in \Omega, EB_\Pi(U) \subseteq \{B_1, \ldots, B_m\} \subseteq \mathit{body}(\Pi) \\ (\sharp[\Gamma]) & : \; v \in \Gamma \end{array}$$

Fig. 1. Tableau rules for normal programs.

acteristic correspondences. For instance, $D^*_{\{\mathit{FTB},\mathit{FTA},\mathit{FFB},\mathit{FFA}\}}(\Pi, \mathbf{A})$ corresponds to Fitting's operator [Fitting 2002], and $D^*_{\{\mathit{FTB},\mathit{FTA},\mathit{FFB},\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A})$ amounts to the well-founded operator [Van Gelder et al. 1991]. Similarly, we show below that $D^*_{\{(a)\text{–}(h)\}}(\Pi, \mathbf{A})$ coincides with unit propagation on a program's completion [Clark 1978; Apt et al. 1987], and $D^*_{\{(a)\text{–}(h),\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A})$ captures *smodels*' propagation [Simons et al. 2002], that is, well-founded operator enhanced by backward propagation. Furthermore, the following tableau calculi are of particular interest:

$a \leftarrow$
$c \leftarrow not\ b, not\ d$
$d \leftarrow a, not\ c$

$\boldsymbol{T}\emptyset$ (*FTB*)
$\boldsymbol{T}a$ (*FTA*)
$\boldsymbol{F}b$ (*FFA*)

$\boldsymbol{T}c$		$\boldsymbol{F}c$		(*Cut*$[atom(\Pi_1)]$)
$\boldsymbol{T}\{not\ b, not\ d\}$	(*BTA*)	$\boldsymbol{F}\{not\ b, not\ d\}$	(*BFA*)	
$\boldsymbol{F}d$	(*BTB*)	$\boldsymbol{T}d$	(*BFB*)	
$\boldsymbol{F}\{a, not\ c\}$	(*FFB*)	$\boldsymbol{T}\{a, not\ c\}$	(*FTB*)	

Fig. 2. Complete tableau of $\mathcal{T}_{smodels}$ for Π_1 and the empty assignment.

$$\begin{aligned}
\mathcal{T}_{comp} &= \{(a)\text{–}(h), Cut[atom(\Pi) \cup body(\Pi)]\} \\
\mathcal{T}_{smodels} &= \{(a)\text{–}(h), WFN[2^{atom(\Pi)}], Cut[atom(\Pi)]\} \\
\mathcal{T}_{nomore} &= \{(a)\text{–}(h), WFN[2^{atom(\Pi)}], Cut[body(\Pi)]\} \\
\mathcal{T}_{nomore++} &= \{(a)\text{–}(h), WFN[2^{atom(\Pi)}], Cut[atom(\Pi) \cup body(\Pi)]\}
\end{aligned}$$

An exemplary complete tableau of $\mathcal{T}_{smodels}$ is given in Figure 2, where rule applications are indicated by rule names, e.g., (*Cut*$[atom(\Pi_1)]$). Both branches in Figure 2 are non-contradictory. They comprise Π_1 along with total assignments: the left branch represents answer set $\{a, c\}$, while the right one gives answer set $\{a, d\}$.

The tableau calculi that include *WFN*$[2^{atom(\Pi)}]$ to deal with unfounded sets are sound and complete, which can be formalized as follows.

THEOREM 3.1. *Let Π be a normal program.*
Then, we have that the following holds for tableau calculi $\mathcal{T}_{smodels}$, $\mathcal{T}_{nomore}$, and $\mathcal{T}_{nomore++}$:

(*1*) *Every incomplete tableau for Π and $\emptyset$ can be extended to a complete tableau for Π and $\emptyset$.*
(*2*) *Program Π has an answer set X iff every complete tableau for Π and $\emptyset$ has a unique non-contradictory branch $(\Pi, \mathbf{A})$ such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$.*
(*3*) *Program Π has no answer set iff every complete tableau for Π and $\emptyset$ is a refutation.*

In particular, note that each of *Cut*$[atom(\Pi)]$, *Cut*$[body(\Pi)]$, and *Cut*$[atom(\Pi) \cup body(\Pi)]$ is sufficient to complete tableaux for Π and $\emptyset$. However, we show in Section 6 that different proof complexities are obtained wrt such *Cut* variants. Moreover, as each of $\mathcal{T}_{smodels}$, $\mathcal{T}_{nomore}$, and $\mathcal{T}_{nomore++}$ admits a (unique) non-contradictory complete branch $(\Pi, \mathbf{A})$ in some tableau iff $(\Pi, \mathbf{A})$ belongs to every complete tableau for Π and $\emptyset$, Theorem 3.1 remains valid when replacing "every" by "some" in the second and the third item of its statement.

4. CHARACTERIZING EXISTING ASP SOLVERS

In this section, we discuss the relationships between the tableau rules in Figure 1 and existing ASP solving approaches. As it turns out, our tableau rules are well-suited for describing the main principles of a variety of ASP solvers. We however start in Section 4.1 by showing correspondences with familiar logic programming operators: Fitting's operator [Fitting 2002] and the well-founded operator [Van Gelder et al. 1991]. Section 4.2 then covers traditional approaches to answer set computation for normal programs, including *smodels* [Simons et al. 2002], *dlv* [Leone et al. 2006], *nomore* [Konczak et al. 2006], and *nomore++* [Anger et al. 2005]. Finally, we sketch in Section 4.3 how tableau rules relate to propagation principles of SAT-based solvers *assat* [Lin and Zhao 2004], *cmodels* [Giunchiglia et al. 2006], and *sag* [Lin et al. 2006] as well as to the

approaches of native conflict-driven learning ASP solvers *smodels*$_{cc}$ [Ward and Schlipf 2004] and *clasp* [Gebser et al. 2007].

4.1 Fitting's Operator and Well-Founded Operator

Given a normal program Π and an assignment $\mathbf{A}$, the two operators in question can be defined in terms of the following sets of atoms:

$$\begin{array}{rcl}\mathbb{T}_\Pi(\mathbf{A}) & = & \{head(r) \mid r \in \Pi, body(r)^+ \subseteq \mathbf{A}^{\boldsymbol{T}}, body(r)^- \subseteq \mathbf{A}^{\boldsymbol{F}}\} \\ \mathbb{N}_\Pi(\mathbf{A}) & = & atom(\Pi) \setminus \{head(r) \mid r \in \Pi, (body(r)^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (body(r)^- \cap \mathbf{A}^{\boldsymbol{T}}) = \emptyset\} \\ \mathbb{U}_\Pi(\mathbf{A}) & = & \bigcup_{U \subseteq atom(\Pi), EB_\Pi(U) \subseteq \{body(r) \mid r \in \Pi, (body(r)^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (body(r)^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\}} U\end{array}$$

$\mathbb{T}_\Pi(\mathbf{A})$ contains the head atoms of rules whose bodies hold wrt $\mathbf{A}$. In contrast, if an atom of $\mathbb{N}_\Pi(\mathbf{A})$ occurs in the head of any rule, then the body of the rule is falsified by $\mathbf{A}$. Given that neither $\mathbb{T}_\Pi(\mathbf{A})$ nor $\mathbb{N}_\Pi(\mathbf{A})$ make use of any entry over $body(\Pi)$, they can be viewed as functions mapping partial interpretations over atoms. In fact, for $\mathbf{A}' = \{\boldsymbol{T}p \mid p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)\} \cup \{\boldsymbol{F}p \mid p \in \mathbf{A}^{\boldsymbol{F}} \cap atom(\Pi)\}$, we have that $\mathbb{T}_\Pi(\mathbf{A}) = \mathbb{T}_\Pi(\mathbf{A}')$ and $\mathbb{N}_\Pi(\mathbf{A}) = \mathbb{N}_\Pi(\mathbf{A}')$. Similarly, it holds that $\mathbb{U}_\Pi(\mathbf{A}) = \mathbb{U}_\Pi(\mathbf{A}')$, where the idea is to reflect the greatest unfounded set, $\mathcal{U}_\Pi(\mathbf{A})$, without referring to entries over $body(\Pi)$. Thus, $\mathbb{U}_\Pi(\mathbf{A})$ collects all $U \subseteq atom(\Pi)$ such that every external body of U for Π is falsified by $\mathbf{A}$, i.e., $EB_\Pi(U) \subseteq \{body(r) \mid r \in \Pi, (body(r)^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (body(r)^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\}$. Given that, for every $p \in \mathbb{N}_\Pi(\mathbf{A})$, we have $EB_\Pi(\{p\}) \subseteq body(p) \subseteq \{body(r) \mid r \in \Pi, (body(r)^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (body(r)^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\}$, it is always the case that $\mathbb{N}_\Pi(\mathbf{A}) \subseteq \mathbb{U}_\Pi(\mathbf{A})$, while the converse does not hold in general.

With the sets $\mathbb{T}_\Pi(\mathbf{A})$, $\mathbb{N}_\Pi(\mathbf{A})$, and $\mathbb{U}_\Pi(\mathbf{A})$ at hand, we can now make the partial interpretations obtained by Fitting's operator and the well-founded operator precise. The following assignment amounts to the result of an application of Fitting's operator: $\{\boldsymbol{T}p \mid p \in \mathbb{T}_\Pi(\mathbf{A})\} \cup \{\boldsymbol{F}p \mid p \in \mathbb{N}_\Pi(\mathbf{A})\}$. The result of an application of the well-founded operator is the assignment $\{\boldsymbol{T}p \mid p \in \mathbb{T}_\Pi(\mathbf{A})\} \cup \{\boldsymbol{F}p \mid p \in \mathbb{U}_\Pi(\mathbf{A})\}$. While both operators use $\mathbb{T}_\Pi(\mathbf{A})$ to infer true atoms, false atoms are obtained via either $\mathbb{N}_\Pi(\mathbf{A})$ or $\mathbb{U}_\Pi(\mathbf{A})$. Since $\mathbb{N}_\Pi(\mathbf{A}) \subseteq \mathbb{U}_\Pi(\mathbf{A})$, the result of Fitting's operator is always subsumed by the one of the well-founded operator.

For illustration, consider the following normal program:

$$\Pi_2 \; = \; \{a \leftarrow not\ b;\ b \leftarrow not\ a;\ c \leftarrow b, not\ a;\ c \leftarrow d;\ d \leftarrow b, not\ a;\ d \leftarrow c;\ e \leftarrow c;\ e \leftarrow d\}$$

For $\mathbf{A} = \{\boldsymbol{T}a, \boldsymbol{F}b\}$, we get $\mathbb{T}_{\Pi_2}(\mathbf{A}) = \{a\}$, $\mathbb{N}_{\Pi_2}(\mathbf{A}) = \{b\}$, and $\mathbb{U}_{\Pi_2}(\mathbf{A}) = \{b, c, d, e\}$. That is, $\{\boldsymbol{T}a, \boldsymbol{F}b\}$ is the result of applying Fitting's operator to $\mathbf{A}$, while $\{\boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{F}c, \boldsymbol{F}d, \boldsymbol{F}e\}$ is obtained with the well-founded operator.

For linking the operators to tableau rules, the following result characterizes the sets of inferred atoms in terms of *FTB* plus *FTA*, and by *FFB* along with either *FFA* or *WFN*$[2^{atom(\Pi)}]$, respectively, where *FFA* is subsumed by *WFN*$[2^{atom(\Pi)}]$.

PROPOSITION 4.1. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment.*
Then, we have that

(*1*) $\mathbb{T}_\Pi(\mathbf{A}) = \big(D_{\{FTA\}}(\Pi, D_{\{FTB\}}(\Pi, \mathbf{A}))\big)^{\boldsymbol{T}}$;

(*2*) $\mathbb{N}_\Pi(\mathbf{A}) = \big(D_{\{FFA\}}(\Pi, D_{\{FFB\}}(\Pi, \mathbf{A}))\big)^{\boldsymbol{F}}$;

(*3*) $\mathbb{U}_\Pi(\mathbf{A}) = \big(D_{\{WFN[2^{atom(\Pi)}]\}}(\Pi, D_{\{FFB\}}(\Pi, \mathbf{A}))\big)^{\boldsymbol{F}}$.

On the right-hand sides, the application of either *FTB* or *FFB* serves as an intermediate step to propagate the truth of all or the falsity of some body literal to the body as such.

The valuations of bodies are then exploited by *FTA*, *FFA*, and *WFN*$[2^{atom(\Pi)}]$, which in turn deduce atoms assigned to true and false, respectively, in an application of Fitting's operator or the well-founded operator. For instance, given $\mathbf{A} = \{\boldsymbol{T}a, \boldsymbol{F}b\}$, $D_{\{FTB\}}(\Pi_2, \mathbf{A}) = \{\boldsymbol{T}\{not\ b\}\}$ gives rise to $D_{\{FTA\}}(\Pi_2, D_{\{FTB\}}(\Pi_2, \mathbf{A})) = \{\boldsymbol{T}a\}$. Furthermore, we have that $D_{\{FFB\}}(\Pi_2, \mathbf{A}) = \{\boldsymbol{F}\{not\ a\}, \boldsymbol{F}\{b, not\ a\}\}$, $D_{\{FFA\}}(\Pi_2, D_{\{FFB\}}(\Pi_2, \mathbf{A})) = \{\boldsymbol{F}b\}$, and $D_{\{WFN[2^{atom(\Pi_2)}]\}}(\Pi_2, D_{\{FFB\}}(\Pi_2, \mathbf{A})) = \{\boldsymbol{F}b, \boldsymbol{F}c, \boldsymbol{F}d, \boldsymbol{F}e\}$. As stated in Proposition 4.1, these outcomes correspond to $\mathbb{T}_{\Pi_2}(\mathbf{A}) = \{a\}$, $\mathbb{N}_{\Pi_2}(\mathbf{A}) = \{b\}$, and $\mathbb{U}_{\Pi_2}(\mathbf{A}) = \{b, c, d, e\}$.

Although we omit further details, note that the correspondences established in Proposition 4.1 carry forward to iterated applications and fixpoints of Fitting's operator and the well-founded operator, respectively. In particular, the entries over $atom(\Pi)$ in $D^*_{\{FTB,FTA,FFB,FFA\}}(\Pi, \emptyset)$ and $D^*_{\{FTB,FTA,FFB,WFN[2^{atom(\Pi)}]\}}(\Pi, \emptyset)$ yield the least fixpoint of Fitting's operator and the well-founded operator, respectively; additional entries over $body(\Pi)$ make valuations of bodies explicit.

4.2 Traditional ASP Solvers

We start by investigating the relationship between *smodels* [Simons et al. 2002] (and *dlv* [Leone et al. 2006]) on the one hand and our tableau rules on the other hand. After that, we extend these considerations to the rule-based approach of *nomore* [Konczak et al. 2006] as well as to the hybrid assignments dealt with by *nomore++* [Anger et al. 2005].

The atom-based approach of *smodels* (logically) works on assignments over atoms, and its propagation (cf. [Simons et al. 2002; Ward and Schlipf 2004]) can be specified in terms of the tableau rules shown in Figure 3.[6] While *FI* expresses that the head of a rule whose body literals hold must be true, its contrapositive, *CFH*, describes that a body literal of a rule must not hold if the head is false and all other body literals hold already. Moreover, an unsupported atom p must be false, and *ARC* reflects this by checking for the presence of some body literal that does not hold in every rule with head p. Conversely, *CTH* expresses that the body literals of a rule $(p' \leftarrow l'_1, \ldots, l'_i, \ldots, l'_n)$ must hold if an atom p is true and any other rule with head p contains some body literal that does not hold.[7] Finally, *AM* formalizes that any atom p belonging to some unfounded set U, in view of some false body literal in every element of $EB_\Pi(U)$, must be false. Note that Figure 3 does not show a contrapositive of *AM*, as *smodels*' propagation does not include such reasoning; it could still be defined analogously to *CTH*, yet requiring the conditions $p \in U$, $U \subseteq atom(\Pi)$, and $\{r \in \Pi \mid head(r) \in U, body(r)^+ \cap U = \emptyset, body(r) \cap \{l_1, \ldots, l_m\} = \emptyset\} \subseteq \{p' \leftarrow l'_1, \ldots, l'_i, \ldots, l'_n\}$ in the proviso (thus checking that every element of $EB_\Pi(U) \setminus \{\{l'_1, \ldots, l'_i, \ldots, l'_n\}\}$ contains some body literal that does not hold), while there is a true atom p in U. Augmenting the tableau rules in Figure 3 with *Cut*$[atom(\Pi_1)]$, we can generate the following tableau for Π_1 and the empty assignment:

$$
\begin{array}{lll}
a \leftarrow & & \\
c \leftarrow not\ b, not\ d & & \\
d \leftarrow a, not\ c & & \\
\quad \boldsymbol{T}a & & (\textit{FI}) \\
\quad \boldsymbol{F}b & & (\textit{ARC}) \\
\boldsymbol{T}c & \boldsymbol{F}c & (Cut[atom(\Pi_1)]) \\
\boldsymbol{F}d\ (\textit{CTH}) & \boldsymbol{T}d\ (\textit{CFH}) &
\end{array}
$$

[6] The names of tableau rules in Figure 3 are aligned to the ones used for *smodels*' propagation rules in [Ward and Schlipf 2004], and the tableau rules reflect *smodels*' propagation rules by respective prerequisites and consequents.

[7] If $p' = p$, then $(p' \leftarrow l'_1, \ldots, l'_i, \ldots, l'_n)$ is the only remaining rule to support p; otherwise, the true atom p is unsupported, and arbitrary body literals may be deduced in the face of a contradiction.

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n}{\boldsymbol{T}p}$$

Forward Inference (FI)

$$\frac{\boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_m}{\boldsymbol{F}p}\,(\lhd)$$

All Rules Canceled (ARC)

$$\frac{p' \leftarrow l'_1, \ldots, l'_i, \ldots, l'_n \quad \boldsymbol{T}p, \boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_m}{\boldsymbol{t}l'_i}\,(\rhd)$$

Contraposition for True Heads (CTH)

$$\frac{p \leftarrow l_1, \ldots, l_{i-1}, l_i, l_{i+1}, \ldots, l_n \quad \boldsymbol{F}p, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_{i-1}, \boldsymbol{t}l_{i+1}, \ldots, \boldsymbol{t}l_n}{\boldsymbol{f}l_i}$$

Contraposition for False Heads (CFH)

$$\frac{\boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_m}{\boldsymbol{F}p}\,(\diamond)$$

At Most (AM)

$(\lhd)$: $p \in atom(\Pi), \{r \in \Pi \mid head(r) = p, body(r) \cap \{l_1, \ldots, l_m\} = \emptyset\} = \emptyset$
$(\rhd)$: $p \in atom(\Pi), \{r \in \Pi \mid head(r) = p, body(r) \cap \{l_1, \ldots, l_m\} = \emptyset\} \subseteq \{p' \leftarrow l'_1, \ldots, l'_i, \ldots, l'_n\}$
$(\diamond)$: $p \in U, U \subseteq atom(\Pi), EB_\Pi(U) \subseteq \{body(r) \mid r \in \Pi, body(r) \cap \{l_1, \ldots, l_m\} \neq \emptyset\}$

Fig. 3. Deterministic tableau rules for traditional (atom-based) ASP solvers.

This tableau is similar to the one of $\mathcal{T}_{smodels}$ shown in Figure 2, yet it omits entries for rule bodies. Note that $\boldsymbol{F}d$ and $\boldsymbol{T}d$ can likewise be generated by applying *ARC* (or *AM*) and *FI* (instead of *CTH* and *CFH*), and corresponding alternative deductions of entries (generated in different order) in the left and the right branch of the tableau in Figure 2 rely on *FFA* (or *WFN*$[2^{atom(\Pi)}]$) and *FTA* (instead of *BTB* and *BFB*).

The next result characterizes *smodels*' propagation in terms of the tableau rules in Figure 1.

PROPOSITION 4.2. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment. Then, we have that*

(*1*) $D_{\{FI\}}(\Pi, \mathbf{A}) = D_{\{FTA\}}(\Pi, D_{\{FTB\}}(\Pi, \mathbf{A}))$;
(*2*) $D_{\{ARC\}}(\Pi, \mathbf{A}) = D_{\{FFA\}}(\Pi, D_{\{FFB\}}(\Pi, \mathbf{A}))$;
(*3*) $D_{\{CTH\}}(\Pi, \mathbf{A}) = D_{\{BTB\}}(\Pi, D_{\{BTA\}}(\Pi, D_{\{FFB\}}(\Pi, \mathbf{A}) \cup \{\boldsymbol{T}p \mid p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)\}))$;
(*4*) $D_{\{CFH\}}(\Pi, \mathbf{A}) = D_{\{BFB\}}(\Pi, D_{\{BFA\}}(\Pi, \mathbf{A}) \cup \{\boldsymbol{T}p \mid p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)\} \cup \{\boldsymbol{F}p \mid p \in \mathbf{A}^{\boldsymbol{F}} \cap atom(\Pi)\})$;
(*5*) $D_{\{AM\}}(\Pi, \mathbf{A}) = D_{\{WFN[2^{atom(\Pi)}]\}}(\Pi, D_{\{FFB\}}(\Pi, \mathbf{A}))$.

As in Proposition 4.1, *FTB* and *FFB* are used on the right-hand sides to reflect truth or falsity of bodies wrt their contained literals by corresponding entries. Also observe that the characterizations of *FI*, *ARC*, and *AM* are similar to those of $\mathbb{T}_\Pi(\mathbf{A})$, $\mathbb{N}_\Pi(\mathbf{A})$, and $\mathbb{U}_\Pi(\mathbf{A})$ in Proposition 4.1. In addition, backward propagation via *CTH* and *CFH* is captured by means of *BTA* plus *BTB* and *BFA* plus *BFB*, respectively, deducing first entries for bodies that need to be either true or false and then corresponding entries for their contained literals.

In view of the fact that all tableau rules used on the right-hand sides in Proposition 4.2 are contained in tableau calculus $\mathcal{T}_{smodels}$ (along with the observation that their application conditions are monotonic wrt assignments), we immediately conclude the following.

COROLLARY 4.3. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment. Then, we have that* $D^*_{\{FI,ARC,CTH,CFH,AM\}}(\Pi, \mathbf{A}) \subseteq D^*_{\mathcal{T}_{smodels}}(\Pi, \mathbf{A})$.

To illustrate that the converse of Corollary 4.3 does not hold in general, even if we limit the attention to entries over atoms, consider the following normal program:

$$\Pi_3 = \{a \leftarrow not\ b;\ b \leftarrow not\ a;\ c \leftarrow b, not\ a;\ d \leftarrow b, not\ a\}$$

Given $\mathbf{A} = \{\boldsymbol{F}c\}$, we obtain $D^*_{\{FI,ARC,CTH,CFH,AM\}}(\Pi_3, \mathbf{A}) = \{\boldsymbol{F}c\}$, while $D^*_{\mathcal{T}_{smodels}}(\Pi_3, \mathbf{A}) = \{\boldsymbol{F}c, \boldsymbol{F}\{b, not\ a\}, \boldsymbol{F}d\}$. Note that the falsity of atom c necessitates the falsity of $body(c \leftarrow b, not\ a)$ because c would be derived otherwise. However, since there is more than one literal in $\{b, not\ a\}$, it is not immediately clear which body literal ought to be false. Hence, an "atom-only" approach like the one of *smodels* gets stuck. In contrast, when we assign the body $\{b, not\ a\}$ to false, we see that $(d \leftarrow b, not\ a)$ is inapplicable as well, which enables $\mathcal{T}_{smodels}$ to deduce $\boldsymbol{F}d$.

We note that, given the similarities between *smodels* and *dlv* on normal programs [Giunchiglia et al. 2008], the latter is also characterized by the tableau rules in Figure 3 (along with $Cut[atom(\Pi)]$). Interestingly, both *smodels* and *dlv* use dedicated data structures in their implementations for indicating (in)applicability of program rules: in *smodels*, a rule with a false body is marked as "inactive" [Simons 2000]; similarly, *dlv* determines truth values of bodies from specific counters [Faber 2002]. It is thus justified to describe the proceeding of atom-based solvers by assignments and tableau rules that incorporate both atoms and bodies.[8] Hence, for comparing atom-based to other ASP solving approaches, we in the following refer to $\mathcal{T}_{smodels}$, rather than a calculus built from the tableau rules in Figure 3. As shown above, $\mathcal{T}_{smodels}$ slightly overestimates the propagation capacities of atom-based solvers, so that lower bounds for its efficiency, considered in Section 6, also apply to the real solvers, viz. *smodels* and *dlv*.

A similar analysis as in Proposition 4.2 is also possible for the rule-based approach of *nomore*, amounting to assignments over rule bodies, and for the hybrid approach of *nomore++*. By assigning truth values to atoms as well as bodies, the latter works on the same basis as its associated tableau calculus $\mathcal{T}_{nomore++}$. In fact, although we omit the details, it is not hard to verify that the propagation operators $\mathcal{P}$, $\mathcal{B}$, and $\mathcal{U}$ (cf. [Anger et al. 2005]) implemented by *nomore++* match the deterministic tableau rules in $\mathcal{T}_{nomore++}$. Furthermore, the choice operator $\mathcal{C}$ of *nomore++* coincides with $Cut[atom(\Pi) \cup body(\Pi)]$. On the other hand, when we consider *nomore* augmented with backward propagation [Linke et al. 2002], we obtain that its propagation is subsumed by $\mathcal{T}_{nomore}$. As with $\mathcal{T}_{smodels}$ and *smodels*, we have that some entries deducible by $\mathcal{T}_{nomore}$ are not inferred by *nomore*. The following program illustrates that assignments over rule bodies only may be less

[8] In [Giunchiglia and Maratea 2005], additional variables for bodies, one for each rule in a program, are explicitly introduced for comparing *smodels* to DPLL, and the characterization of *smodels*' propagation rules in terms of unit propagation provided in [Gebser and Schaub 2006a] likewise shows that rule bodies serve as inherent structural variables.

informative than assignments that also include atoms:

$$\Pi_4 \;=\; \{a \leftarrow b;\; b \leftarrow c;\; b \leftarrow \mathit{not}\ c;\; c \leftarrow a, \mathit{not}\ b\}$$

Given $\mathbf{A} = \{\boldsymbol{T}\{b\}\}$, since two rules, $(b \leftarrow c)$ and $(b \leftarrow \mathit{not}\ c)$, have b as their head, *nomore* cannot determine a unique program rule to derive b from, so that its propagation cannot infer anything. In contrast, $D^*_{\mathcal{T}_{\mathit{nomore}}}(\Pi_4, \mathbf{A}) = \{\boldsymbol{T}\{b\}, \boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}\{a, \mathit{not}\ b\}, \boldsymbol{F}c, \boldsymbol{F}\{c\}, \boldsymbol{T}\{\mathit{not}\ c\}\}$. Here, the fact that the atom b must be true yields that the body $\{a, \mathit{not}\ b\}$ is necessarily false, from which $\mathcal{T}_{\mathit{nomore}}$ then deduces the remaining entries.

4.3 SAT-Based and Conflict-Driven Learning ASP Solvers

Lin and Zhao [Lin and Zhao 2004] showed that the answer sets of a normal program Π coincide with the models of the completion of Π satisfying all loop formulas of Π. By introducing auxiliary variables for rule bodies, as also used in [Babovich and Lifschitz 2003] to avoid an exponential blow-up due to clausification, the *completion*, $\mathit{Comp}(\Pi)$, and *loop formulas*, $\mathit{LF}(\Pi)$, of a program Π can be defined as follows:

$$\begin{aligned}
\mathit{Comp}(\Pi) \;&=\; \{p_B \leftrightarrow (\textstyle\bigwedge_{p \in B^+} p \wedge \bigwedge_{q \in B^-} \neg q) \mid B \in \mathit{body}(\Pi)\} \\
&\cup\; \{p \leftrightarrow (\textstyle\bigvee_{B \in \mathit{body}(p)} p_B) \mid p \in \mathit{atom}(\Pi)\} \\
\mathit{LF}(\Pi) \;&=\; \{(\textstyle\bigvee_{p \in U} p) \rightarrow (\bigvee_{B \in EB_\Pi(U)} p_B) \mid U \in \mathit{loop}(\Pi)\}
\end{aligned}$$

Given that a program Π may yield exponentially many (non-redundant) loop formulas [Lifschitz and Razborov 2006], SAT-based ASP solvers *assat* [Lin and Zhao 2004], *cmodels* [Giunchiglia et al. 2006], and *sag* [Lin et al. 2006] do not a priori construct $\mathit{LF}(\Pi)$. Rather, they use some SAT solver to generate a model of $\mathit{Comp}(\Pi)$ and, afterwards, check whether the model contains a loop U of Π whose loop formula is violated. If so, U is unfounded wrt the model at hand, and adding the loop formula for U eliminates the model as an answer set candidate.[9]

Regarding the generation of answer set candidates, the following analog of Theorem 3.1 applies to $\mathcal{T}_{\mathit{comp}}$ and models of $\mathit{Comp}(\Pi)$.

THEOREM 4.4. *Let Π be a normal program.*
Then, we have that the following holds for tableau calculus $\mathcal{T}_{\mathit{comp}}$:

(*1*) *Every incomplete tableau for Π and $\emptyset$ can be extended to a complete tableau for Π and $\emptyset$.*
(*2*) *$\mathit{Comp}(\Pi)$ has a model X iff every complete tableau for Π and $\emptyset$ has a unique non-contradictory branch $(\Pi, \mathbf{A})$ such that $(\mathbf{A}^{\boldsymbol{T}} \cap \mathit{atom}(\Pi)) \cup \{p_B \mid B \in \mathbf{A}^{\boldsymbol{T}} \cap \mathit{body}(\Pi)\} = X$.*
(*3*) *$\mathit{Comp}(\Pi)$ has no model iff every complete tableau for Π and $\emptyset$ is a refutation.*

Theorem 4.4 shows that $\mathcal{T}_{\mathit{comp}}$ captures exactly the models of $\mathit{Comp}(\Pi)$ for a normal program Π.[10] Since $\mathcal{T}_{\mathit{comp}}$ admits a non-contradictory complete branch $(\Pi, \mathbf{A})$ in some tableau iff $(\Pi, \mathbf{A})$ belongs to every complete tableau for Π and $\emptyset$, Theorem 4.4 (like Theorem 3.1) remains valid when replacing "every" by "some" in the second and the third item of its statement.

By Theorem 3.1, adding *WFN*$[2^{\mathit{atom}(\Pi)}]$ to $\mathcal{T}_{\mathit{comp}}$ leads to a tableau calculus characterizing answer sets. However, SAT-based ASP solvers check for unfounded loops rather than arbitrary unfounded sets, while atom-wise unfounded set handling is accomplished via *FFA* (enclosed in $\mathit{Comp}(\Pi)$). Given that *FFA* is subsumed by *WFN*$[2^{\mathit{atom}(\Pi)}]$, but not by *WFN*$[\mathit{loop}(\Pi)]$, the next

[9] While *assat* and *cmodels* apply sophisticated unfounded set checks only wrt total answer set candidates, *sag* can perform them also wrt partial assignments generated by a SAT solver.
[10] The models of the completion of a logic program Π are also called the "supported models" of Π [Apt et al. 1987].

result, which shows that *WFN*$[2^{atom(\Pi)}]$ and *WFN*$[loop(\Pi)]$ plus *FFA* yield the same deductive closure (in combination with *FFB*, deducing the falsity of bodies from body literals that do not hold), tells us that limiting *WFN* to loops abolishes overlaps between tableau rules.

PROPOSITION 4.5. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment.*
Then, we have that $D^*_{\{FFB,WFN[2^{atom(\Pi)}]\}}(\Pi,\mathbf{A}) = D^*_{\{FFB,FFA,WFN[loop(\Pi)]\}}(\Pi,\mathbf{A})$.

In view of Proposition 4.5, by adding the tableau rule *WFN*$[loop(\Pi)]$ to $\mathcal{T}_{comp}$, we characterize models of $Comp(\Pi) \cup LF(\Pi)$, which coincide with the answer sets of Π.

THEOREM 4.6. *Let* Π *be a normal program.*
Then, we have that the following holds for tableau calculus $\mathcal{T}_{comp} \cup \{WFN[loop(\Pi)]\}$*:*

(*1*) *Every incomplete tableau for* Π *and* $\emptyset$ *can be extended to a complete tableau for* Π *and* $\emptyset$*.*

(*2*) *Program* Π *has an answer set* X *iff every complete tableau for* Π *and* $\emptyset$ *has a unique non-contradictory branch* $(\Pi,\mathbf{A})$ *such that* $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$*.*

(*3*) *Program* Π *has no answer set iff every complete tableau for* Π *and* $\emptyset$ *is a refutation.*

As with Theorem 3.1 and 4.4, we have that Theorem 4.6 remains valid when replacing "every" by "some" in the second and the third item of its statement.

For illustration, consider the following normal program:

$$\Pi_5 = \{a \leftarrow not\ b;\ b \leftarrow not\ a;\ c \leftarrow a;\ c \leftarrow d;\ d \leftarrow c, not\ a;\ e \leftarrow c;\ e \leftarrow d\}$$

We have that $loop(\Pi_5) = \{\{c,d\}\}$ and $EB_{\Pi_5}(\{c,d\}) = EB_{\Pi_5}(\{c,d,e\}) = \{\{a\}\}$. For $\mathbf{A} = \{\boldsymbol{F}\{a\}\}$, we thus get $D_{\{WFN[2^{atom(\Pi_5)}]\}}(\Pi_5,\mathbf{A}) = \{\boldsymbol{F}c, \boldsymbol{F}d, \boldsymbol{F}e\}$, while $D_{\{WFN[loop(\Pi_5)]\}}(\Pi_5,\mathbf{A}) = \{\boldsymbol{F}c, \boldsymbol{F}d\}$ does not include $\boldsymbol{F}e$. As $body(e) = \{\{c\},\{d\}\} \subseteq \big(D_{\{FFB\}}(\Pi_5,\{\boldsymbol{F}c,\boldsymbol{F}d\})\big)^{\boldsymbol{F}}$, $\boldsymbol{F}e \in D^*_{\{FFB,FFA,WFN[loop(\Pi_5)]\}}(\Pi_5,\mathbf{A})$ nonetheless holds. Hence, the combination of *FFB*, *FFA*, and *WFN*$[loop(\Pi_5)]$ allows us to deduce the same entries as obtained with *FFB* and *WFN*$[2^{atom(\Pi_5)}]$.

To see the relationship between the tableau rules in Figure 1 and the clausal representation of $Comp(\Pi) \cup LF(\Pi)$, given a tableau rule with prerequisites $\ell_1,\ldots,\ell_n$ and ℓ as its consequence, we define a clause δ by

$$\delta = \{p_v \mid \boldsymbol{T}v \in \{\overline{\ell}_1,\ldots,\overline{\ell}_n,\ell\}\} \cup \{\neg p_v \mid \boldsymbol{F}v \in \{\overline{\ell}_1,\ldots,\overline{\ell}_n,\ell\}\}$$

where we let $p_v = v$ if v is an atom in $\mathcal{P}$. For Π_5 as above and the tableau rules in $\mathcal{T}_{comp}$, the following theory Δ constitutes the union of (unsubsumed) clauses given by tableau rules (other than *Cut*$[atom(\Pi_5) \cup body(\Pi_5)]$):

$$\Delta = \{\ \{p_{\{not\ b\}}, b\}, \{p_{\{not\ a\}}, a\}, \{p_{\{a\}}, \neg a\}, \{p_{\{d\}}, \neg d\}, \{p_{\{c\}}, \neg c\}, \{p_{\{c,not\ a\}}, \neg c, a\}\ \} \quad (2)$$

$$\cup \left\{ \begin{array}{l} \{\neg p_{\{not\ b\}}, \neg b\}, \{\neg p_{\{not\ a\}}, \neg a\}, \{\neg p_{\{a\}}, a\}, \{\neg p_{\{d\}}, d\}, \{\neg p_{\{c\}}, c\}, \\ \{\neg p_{\{c,not\ a\}}, c\}, \{\neg p_{\{c,not\ a\}}, \neg a\} \end{array} \right\} \quad (3)$$

$$\cup \left\{ \begin{array}{l} \{a, \neg p_{\{not\ b\}}\}, \{b, \neg p_{\{not\ a\}}\}, \{c, \neg p_{\{a\}}\}, \{c, \neg p_{\{d\}}\}, \{d, \neg p_{\{c,not\ a\}}\}, \\ \{e, \neg p_{\{c\}}\}, \{e, \neg p_{\{d\}}\} \end{array} \right\} \quad (4)$$

$$\cup\ \{\ \{\neg a, p_{\{not\ b\}}\}, \{\neg b, p_{\{not\ a\}}\}, \{\neg c, p_{\{a\}}, p_{\{d\}}\}, \{\neg d, p_{\{c,not\ a\}}\}, \{\neg e, p_{\{c\}}, p_{\{d\}}\}\ \} \quad (5)$$

The clauses in (2) are obtained from tableau rule *FTB*, and likewise from its contrapositive *BFB*. Similarly, the clauses in (3), (4), and (5) result from *FFB* or *BTB*, *FTA* or *BFA*, and *FFA* or *BTA*, respectively. Note that the clauses in Δ rephrase the equivalences contained in $Comp(\Pi_5)$, so

that $\Delta \equiv Comp(\Pi_5)$. In fact, the (deterministic) tableau rules in $\mathcal{T}_{comp}$ express conditions for unit propagation, as performed in SAT solvers used by *assat*, *cmodels*, and *sag*. When we extract clauses from tableau rule *WFN*$[loop(\Pi_5)]$, or likewise from *WFJ*$[loop(\Pi_5)]$, we get the theory Λ:

$$\Lambda \ = \ \big\{ \ \{\neg c, p_{\{a\}}\}, \{\neg d, p_{\{a\}}\} \ \big\}$$

Once clauses from $\Lambda \equiv LF(\Pi_5)$ have been added to $Comp(\Pi_5)$ (to eliminate some invalid answer set candidate), they can be used for unit propagation in a SAT solver, just like the clauses in Δ.

In native conflict-driven learning ASP solvers, such as *smodels*$_{cc}$ [Ward and Schlipf 2004] and *clasp* [Gebser et al. 2007], the clauses obtained from tableau rules contribute reasons for conflicts. For *clasp*, such clauses correspond to the sets Δ_Π and Λ_Π of nogoods (cf. [Gebser et al. 2007]), and the (implicit) constraints propagated by *smodels*$_{cc}$, extracting reasons relative to *smodels*' propagation rules, are closely related to clauses obtained from tableau rules (cf. [Giunchiglia and Maratea 2005; Gebser and Schaub 2006a; Giunchiglia et al. 2008]). This indicates that the tableau rules in Figure 1 are well-suited to characterize conditions for propagation as applied in both SAT-based and native conflict-driven learning ASP solvers. Of course, our tableaux reflect neither preprocessing techniques, such as the ones described in [Babovich and Lifschitz 2003; Gebser et al. 2008], nor backjumping and learning schemes of conflict-driven learning SAT and ASP solvers (see, e.g., [Marques-Silva and Sakallah 1999; Zhang et al. 2001; Eén and Sörensson 2004; Biere et al. 2009]). We note that the calculus based on state transition graphs presented in [Lierler 2011] captures backjumping and conflict-driven learning as performed by *smodels*$_{cc}$; in that work, the clauses attributed to *smodels*$_{cc}$ avoid auxiliary variables for rule bodies in favor of (non-erasable) duplicate literals.

Finally, let us comment on some particularities of unfounded set handling. On the one hand, Proposition 4.5 shows that tableau rule *WFN*$[2^{atom(\Pi)}]$ can be replaced by more restrictive rule *WFN*$[loop(\Pi)]$ without sacrificing deducible entries. In fact, SAT-based ASP solvers concentrate the consideration of positive recursion on loops, and native ASP solvers like *clasp*, *dlv*, and *smodels* exploit strongly connected components of programs' dependency graphs to achieve a similar effect. However, no existing ASP solver incorporates a contrapositive of *WFN*, that is, *WFJ*$[2^{atom(\Pi)}]$ or *WFJ*$[loop(\Pi)]$, in its propagation (unless loop formulas have been recorded). An approach to extend unfounded set handling in this direction has been presented in [Chen et al. 2008; 2009]. Unfortunately, it amounts to failed-literal detection (cf. [Freeman 1995; Simons et al. 2002]), whose high (polynomial) computational cost makes its unrestricted application prohibitive in practice. It is still interesting that a result similar to Proposition 4.5 cannot be obtained for *WFJ*$[2^{atom(\Pi)}]$ and *WFJ*$[loop(\Pi)]$. The fact that the latter tableau rule is strictly weaker than the former can be observed by considering Π_5 along with $\mathbf{A} = \{\boldsymbol{T}e\}$. Since $EB_{\Pi_5}(\{c, d, e\}) = \{\{a\}\}$, we obtain that $\boldsymbol{T}\{a\} \in D_{\{\textit{WFJ}[2^{atom(\Pi_5)}]\}}(\Pi_5, \mathbf{A})$. On the other hand, since $loop(\Pi_5) = \{\{c, d\}\}$ and $body(e) = \{\{c\}, \{d\}\}$, neither *WFJ*$[loop(\Pi_5)]$ nor any of the tableau rules *(a)–(i)* in Figure 1 (in particular, *BTA*) is applicable in the branch $(\Pi_5, \mathbf{A})$, that is, $D_{\{\textit{(a)–(i)},\textit{WFJ}[loop(\Pi_5)]\}}(\Pi_5, \mathbf{A}) = \emptyset$. Regarding the impact of not at all applying *WFJ* or restricting the sets of atoms to which it can be applied to loops, in Section 6, we show that *WFJ* can be simulated by means of *Cut* and *WFN*, using exactly the idea of failed-literal detection.

5. GENERIC TABLEAUX FOR COMPOSITE LANGUAGE CONSTRUCTS

In what follows, we generalize our approach and develop an extensible tableau framework for logic programs incorporating composite language constructs, such as *dlv*'s aggregates [Faber et al. 2008] or *smodels*' cardinality and weight constraints [Simons et al. 2002]. To this end, we take a more abstract perspective than before and view, e.g., conjunctions as (simple) Boolean aggregates,

which like atoms can be preceded by not. However, we also show below that the dedicated tableau framework introduced in Section 3 and the generic approach developed in the sequel coincide on the common language fragment of normal programs.

The basic idea of our generic approach is to specify core tableau rules that, for one, aim at establishing model conditions by propagating truth and falsity of entries along program rules. For instance, consider the rule $(0\{a,d\}1 \leftarrow 1\{b, not\ e\}2)$, including two cardinality constraints.[11] If $1\{b, not\ e\}2$ holds wrt an assignment, we know that $0\{a,d\}1$ must hold as well; conversely, $1\{b, not\ e\}2$ must not hold if it is known that $0\{a,d\}1$ does not hold. In our generic framework, such inferences are reflected by deducing $\boldsymbol{T}0\{a,d\}1$ from $\boldsymbol{T}1\{b, not\ e\}2$ or, conversely, $\boldsymbol{F}1\{b, not\ e\}2$ from $\boldsymbol{F}0\{a,d\}1$. Notice that we associate truth values with cardinality constraints, and that matching them to truth values of the cardinality constraints' literals requires specific tableau rules. Hence, when we below augment our framework with composite language constructs, we also introduce such construct-specific tableau rules.

Regardless of concrete program syntax, an answer set must be a minimal model of the reduct relative to itself, as in the case of normal programs (cf. Section 2). Our generic tableau rules reflect this minimality requirement by investigating external supports of sets of atoms. To this end, for a rule $(\alpha \leftarrow \beta)$ and an assignment $\mathbf{A}$, we make use of two predicates, $\overleftarrow{sup}_{\mathbf{A}}(\alpha, S)$ and $\overrightarrow{sup}_{\mathbf{A}}(\beta, S')$, to check whether a set S of atoms can be supported via α and whether β can hold independently of atoms in S'. Since our tableau rules consider subsets S' of S, viz. $S' = \emptyset$ or $S' = S$, the two predicates allow us to test whether $(\alpha \leftarrow \beta)$ provides an external support for S wrt $\mathbf{A}$. Of particular interest are the cases where there is no external support for S, as it tells us that all atoms of S must be false, or where there is exactly one external support $(\alpha \leftarrow \beta)$ for S. In the latter case, if S contains some true atom, we know that β must hold, and additional entries might be required too, for which we provide two sets, $min_{\mathbf{A}}(\alpha, S)$ and $max_{\mathbf{A}}(\beta, S')$. For instance, if at least one of the entries $\boldsymbol{T}a$ and $\boldsymbol{T}b$ belongs to a given assignment $\mathbf{A}$ and the rule $(0\{a,d\}1 \leftarrow 1\{b, not\ e\}2)$ is the single external support for the set $\{a,b\}$, we get $min_{\mathbf{A}}(0\{a,d\}1, \{a,b\}) = \{\boldsymbol{F}d\}$ and $max_{\mathbf{A}}(1\{b, not\ e\}2, \{a,b\}) = \{\boldsymbol{F}e\}$. In fact, if d was true, the upper bound 1 of $0\{a,d\}1$ would be reached, so that no further atom can be supported. Likewise, if the literal $not\ e$ in $1\{b, not\ e\}2$ was false, the lower bound 1 of $1\{b, not\ e\}2$ could only be achieved by making b true, so that the support is not external to $\{a,b\}$. Since the definitions of $\overleftarrow{sup}$, $\overrightarrow{sup}$, min, and max are specific to a language construct at hand, we below develop them gradually upon integrating composite language constructs into our generic framework. To be more precise, after in Section 5.1 generalizing the definitions from Section 2, we devise generic tableau rules in Section 5.2. These are augmented with tableau rules for conjunctions, cardinality constraints, and disjunctions in Section 5.3, 5.4, and 5.5, respectively.

5.1 Answer Sets for Propositional Theories

Among several proposals defining answer sets for logic programs accommodating particular language extensions (e.g., [Simons et al. 2002; Ferraris and Lifschitz 2005; Faber 2005; Ferraris 2005; Liu and Truszczyński 2006; Faber et al. 2011]), we rely on the one by Ferraris [Ferraris 2005], as it is general enough to deal with arbitrary (ground) aggregates and has a firm basis in the logic of here-and-there [Pearce 1996; Lifschitz et al. 2001]. To achieve generality, this semantics

[11] A cardinality constraint like $1\{b, not\ e\}2$ resembles a linear inequality, viz. $1 \leq b + (1 - e) \leq 2$, over Boolean variables. By assigning each of the variables b and e to either 1 (true) or 0 (false), the inequality evaluates to true or false, respectively. For instance, $1 \leq 1 + (1 - 1) \leq 2$ holds with $b = e = 1$, while $1 \leq 0 + (1 - 1) \leq 2$, obtained with $b = 0$ and $e = 1$, does not hold.

applies to propositional theories and identifies aggregates with propositional formulas.

Formally, a propositional theory is a finite set of propositional formulas, constructed from atoms in an alphabet $\mathcal{P}$ and the connectives $\bot$, $\wedge$, $\vee$, and $\rightarrow$. Any other connective is considered as an abbreviation, in particular, $\neg\phi$ stands for $(\phi \rightarrow \bot)$. An interpretation, represented by the set X of its entailed atoms, is a model of a propositional theory Φ if $X \models \phi$ for all $\phi \in \Phi$, where $\models$ is the standard satisfaction relation of propositional logic. The *reduct*, denoted by Φ^X, of Φ wrt X is a propositional theory, (recursively) defined as follows:

$$\Phi^X = \{\phi^X \mid \phi \in \Phi\}$$

$$\phi^X = \begin{cases} \bot & \text{if } X \not\models \phi \\ \phi & \text{if } \phi \in X \\ \phi_1^X \circ \phi_2^X & \text{if } X \models \phi \text{ and } \phi = (\phi_1 \circ \phi_2) \text{ for } \circ \in \{\wedge, \vee, \rightarrow\} \end{cases}$$

Intuitively, all (maximal) subformulas of Φ that are false in X are replaced by $\bot$ in Φ^X, while other subformulas of Φ stay intact. Hence, any model X of Φ is a model of Φ^X as well. Also note that all occurrences of negation, i.e., subformulas of the form $(\phi \rightarrow \bot)$, are replaced by constants in Φ^X, since either ϕ or $(\phi \rightarrow \bot)$ is false in X. An interpretation X is an *answer set* of a propositional theory Φ if X is a minimal model of Φ^X. In fact, an answer set X of Φ is the unique least model of Φ^X because all atoms occurring in Φ^X belong to X, but a least model of Φ^X is, in general, not guaranteed to exist.

As an example, consider the propositional theory

$$\Phi = \{a \vee b\}$$

along with its reducts $\Phi^\emptyset = \{\bot\}$, $\Phi^{\{a\}} = \{a \vee \bot\}$, $\Phi^{\{b\}} = \{\bot \vee b\}$, and $\Phi^{\{a,b\}} = \{a \vee b\}$. While $\emptyset$ is not a model of $\Phi^\emptyset$, the proper subsets $\{a\}$ and $\{b\}$ of $\{a, b\}$ are models of $\Phi^{\{a,b\}}$. As $\{a\}$ and $\{b\}$ are minimal models of $\Phi^{\{a\}}$ and $\Phi^{\{b\}}$, respectively, they are answer sets of Φ. Furthermore, observe that a is the only atom occurring in $\Phi^{\{a\}}$, and the same applies to b and $\Phi^{\{b\}}$.

In a general setting, we understand a *(propositional) logic program* Π over an alphabet $\mathcal{P}$ as a finite set of rules of the form $(\alpha \leftarrow \beta)$ where α and β are *literals*, that is, expressions over $\mathcal{P}$ possibly preceded by not. As before, $atom(\Pi)$ denotes the set of atoms occurring in Π. In the following, we refine heads α and bodies β for obtaining particular classes of logic programs. The semantics of a logic program is given by the answer sets of an associated propositional theory, obtained via a translation τ described below. However, the proof-theoretic characterizations we provide apply directly to logic programs, without translating them to propositional theories.

5.2 Generic Tableau Rules

We begin with a simple class of *unary programs* where rules $(\alpha \leftarrow \beta)$ are restricted to *atomic literals*, that is, each of α and β is equal to either p or $not\ p$ for an atom $p \in \mathcal{P}$.[12] The semantics of a unary program Π is given by the answer sets of a propositional theory, $\tau[\Pi]$, (recursively) defined as follows:

$$\tau[\Pi] = \{\tau[\beta] \rightarrow \tau[\alpha] \mid (\alpha \leftarrow \beta) \in \Pi\} \tag{6}$$

$$\tau[\pi] = \begin{cases} \neg\tau[v] & \text{if } \pi = not\ v \\ \pi & \text{if } \pi \in \mathcal{P} \end{cases} \tag{7}$$

[12] Our notion of a unary program is different from the one considered in [Janhunen 2006]. The latter allows for one positive and arbitrarily many negative body literals, but not for negative head literals.

For illustration, consider the following unary program:

$$\Pi_6 \;=\; \{a \leftarrow \mathit{not}\ b;\ \mathit{not}\ a \leftarrow c;\ b \leftarrow c;\ c \leftarrow b\}$$

The associated propositional theory is as follows:

$$\tau[\Pi_6] \;=\; \{\neg b \rightarrow a;\ c \rightarrow \neg a;\ c \rightarrow b;\ b \rightarrow c\}$$

The sets $\{a\}$ and $\{b,c\}$ are models of $\tau[\Pi_6]$, and their respective reducts are:

$$\begin{aligned}(\tau[\Pi_6])^{\{a\}} &= \{\neg\bot \rightarrow a;\ \bot \rightarrow \bot\}\\ (\tau[\Pi_6])^{\{b,c\}} &= \{\bot \rightarrow \bot;\ c \rightarrow \neg\bot;\ c \rightarrow b;\ b \rightarrow c\}\end{aligned}$$

Clearly, $\{a\}$ is the least model of $(\tau[\Pi_6])^{\{a\}}$, so that $\{a\}$ is an answer set of Π_6. The (unique) minimal model of $(\tau[\Pi_6])^{\{b,c\}}$ is $\emptyset$. Hence, $\{b,c\}$ is not the least model of $(\tau[\Pi_6])^{\{b,c\}}$ and thus not an answer set of Π_6.

While the semantics is based on translation to propositional theories, our tableau framework deals with logic programs as such. The global design, however, follows the two semantic requirements for answer sets: modelhood wrt a program and (non-circular) support wrt the reduct. In order to establish the latter, for a program Π, two sets $S \subseteq \mathit{atom}(\Pi), S' \subseteq \mathit{atom}(\Pi)$, and an assignment $\mathbf{A}$, we define:

$$\mathit{sup}_{\mathbf{A}}(\Pi, S, S') \;=\; \{(\alpha \leftarrow \beta) \in \Pi \mid \boldsymbol{f}\beta \notin \mathbf{A},\ \overleftarrow{\mathit{sup}}_{\mathbf{A}}(\alpha, S),\ \overrightarrow{\mathit{sup}}_{\mathbf{A}}(\beta, S')\} \tag{8}$$

The purpose of $\mathit{sup}_{\mathbf{A}}(\Pi, S, S')$ is to determine all rules of Π that can, wrt $\mathbf{A}$, provide a support for the atoms in S that is external to S'. Of particular interest are the cases where $\mathit{sup}_{\mathbf{A}}(\Pi, S, S')$ is empty or a singleton $\{\alpha \leftarrow \beta\}$. In the first case, the atoms in S cannot be supported and are prone to be false, while the second case tells us that $(\alpha \leftarrow \beta)$ is the unique support for S external to S'. Since we below consider only situations where $S' \subseteq S$, viz. $S' = \emptyset$ and $S' = S$, $\mathit{sup}_{\mathbf{A}}(\Pi, S, S') = \{\alpha \leftarrow \beta\}$ indicates that β must hold to (non-circularly) support S.

Further investigating the definition of $\mathit{sup}_{\mathbf{A}}(\Pi, S, S')$ in (8), we note that a rule $(\alpha \leftarrow \beta)$ such that $\boldsymbol{f}\beta \in \mathbf{A}$ cannot provide any support wrt $\mathbf{A}$. Otherwise, we check via $\overleftarrow{\mathit{sup}}_{\mathbf{A}}(\alpha, S)$ that α can support S, and via $\overrightarrow{\mathit{sup}}_{\mathbf{A}}(\beta, S')$ that β does not (positively) rely on S'. For the simple class of unary programs, these concepts are defined as follows:

$$\overleftarrow{\mathit{sup}}_{\mathbf{A}}(p, S) \quad \text{if } p \in S \tag{9}$$

$$\overrightarrow{\mathit{sup}}_{\mathbf{A}}(p, S') \quad \text{if } p \in \mathcal{P} \setminus S' \tag{10}$$

$$\overrightarrow{\mathit{sup}}_{\mathbf{A}}(\mathit{not}\ v, S') \quad \text{for every expression } v \tag{11}$$

The universal validity of (11) is because only *positive* dependencies are taken into account. Also note that a rule $(\alpha \leftarrow \beta)$ such that $\alpha = \mathit{not}\ v$ cannot support any set S of atoms. (We further illustrate the above concepts below Theorem 5.1.)

The tableau rules constituting our primal generic calculus are shown in Figure 4. Among them, $I{\uparrow}$ and $I{\downarrow}$ provide *rule-based* inferences, such as modus ponens and modus tollens. The tableau rules $N{\uparrow}$ and $N{\downarrow}$ amount to *negation* and *support* for atoms, building on similar principles as completion (of normal programs). Note that the derivability of an atom p and thus the applicability of tableau rules $N{\uparrow}$ and $N{\downarrow}$, respectively, is determined by $\mathit{sup}_{\mathbf{A}}(\Pi, \{p\}, \emptyset)$. In the general case, rule $N{\downarrow}$ makes use of two further concepts, $\mathit{min}_{\mathbf{A}}(\alpha, S)$ and $\mathit{max}_{\mathbf{A}}(\beta, S')$, used to determine entries that must necessarily be added to $\mathbf{A}$ in order to support some atom in S via $(\alpha \leftarrow \beta)$ without positively relying on S'. However, as these concepts play no role in the setting of unary

$$\frac{\alpha \leftarrow \beta \quad \boldsymbol{t}\beta}{\boldsymbol{t}\alpha} \qquad \frac{\alpha \leftarrow \beta \quad \boldsymbol{f}\alpha}{\boldsymbol{f}\beta}$$

(a) Implication ($I\uparrow$) *(b) Contraposition ($I\downarrow$)*

$$\frac{\Pi, \mathbf{A}}{\boldsymbol{F}p}\ (p \in atom(\Pi), sup_{\mathbf{A}}(\Pi, \{p\}, \emptyset) = \emptyset)$$

(c) Negation ($N\uparrow$)

$$\frac{\Pi, \mathbf{A}}{\boldsymbol{t}\beta, min_{\mathbf{A}}(\alpha, \{p\}), max_{\mathbf{A}}(\beta, \emptyset)}\ (p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi), sup_{\mathbf{A}}(\Pi, \{p\}, \emptyset) = \{\alpha \leftarrow \beta\})$$

(d) Support ($N\downarrow$)

$$\frac{\Pi, \mathbf{A}}{\boldsymbol{F}p}\ (S \subseteq atom(\Pi), p \in S, sup_{\mathbf{A}}(\Pi, S, S) = \emptyset)$$

(e) Unfounded Set ($U\uparrow$)

$$\frac{\Pi, \mathbf{A}}{\boldsymbol{t}\beta, min_{\mathbf{A}}(\alpha, S), max_{\mathbf{A}}(\beta, S)}\ (S \subseteq atom(\Pi), \mathbf{A}^{\boldsymbol{T}} \cap S \neq \emptyset, sup_{\mathbf{A}}(\Pi, S, S) = \{\alpha \leftarrow \beta\})$$

(f) Well-Founded Set ($U\downarrow$)

$$\frac{}{\boldsymbol{T}v \mid \boldsymbol{F}v}\ (v \in \Gamma)$$

(g) Cut (Cut$[\Gamma]$)

Fig. 4. Tableau rules for rules *(a),(b)*; atoms *(c),(d)*; sets of atoms *(e),(f)*; and cutting *(g)*.

programs, they are defined to be empty for atomic literals:

$$min_{\mathbf{A}}(p, S) = \emptyset \quad \text{for } p \in \mathcal{P} \tag{12}$$
$$max_{\mathbf{A}}(p, S') = \emptyset \quad \text{for } p \in \mathcal{P} \tag{13}$$
$$max_{\mathbf{A}}(not\ v, S') = \emptyset \quad \text{for every expression } v \tag{14}$$

Furthermore, the tableau rules $U\uparrow$ and $U\downarrow$ take care of (non-empty) "unfounded sets," either by identifying atoms that cannot be non-circularly supported ($U\uparrow$) or by preventing true atoms from becoming unfounded ($U\downarrow$). The applicability of $U\uparrow$ and $U\downarrow$ is determined by $sup_{\mathbf{A}}(\Pi, S, S)$ for a set S of atoms. Since $sup_{\mathbf{A}}(\Pi, S, S) \subseteq sup_{\mathbf{A}}(\Pi, S, S')$ for every $S' \subseteq S$ (cf. (10)) and, in particular, for $S' = \emptyset$, $U\uparrow$ and $U\downarrow$ subsume $N\uparrow$ and $N\downarrow$, which rely on the weaker concept $sup_{\mathbf{A}}(\Pi, \{p\}, \emptyset)$. We nonetheless include $N\uparrow$ and $N\downarrow$ because their applicability is easy to determine, and thus they have counterparts in virtually all ASP solvers. Also note that we do not parameterize $U\uparrow$ and $U\downarrow$ by Ω, which has been included in corresponding tableau rules *WFN*$[\Omega]$ and *WFJ*$[\Omega]$ in Figure 1 to reflect a possible restriction to loops. Albeit loops can also be identified in propositional theories [Ferraris et al. 2006], the generalization is not straightforward and

omitted here for brevity. Finally, the *Cut*$[\Gamma]$ rule, allowing for case analyses on the expressions in Γ, is identical to its counterpart in Figure 1.

For a unary program Π, we fix the domain of assignments $\mathbf{A}$ as well as the cut objects used by *Cut*$[\Gamma]$ to $dom(\mathbf{A}) = \Gamma = atom(\Pi)$. Similar to Theorem 3.1 applying to normal programs, we can now characterize the answer sets of unary programs in terms of generic tableaux.

THEOREM 5.1. *Let Π be a unary program.*

Then, we have that the following holds for the tableau calculus consisting of the tableau rules (a)–(g):

(*1*) *Every incomplete tableau for Π and $\emptyset$ can be extended to a complete tableau for Π and $\emptyset$.*

(*2*) *Program Π has an answer set X iff every complete tableau for Π and $\emptyset$ has a unique non-contradictory branch $(\Pi, \mathbf{A})$ such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$.*

(*3*) *Program Π has no answer set iff every complete tableau for Π and $\emptyset$ is a refutation.*

As with the tableau rules in Figure 1, we have that the generic calculus including the tableau rules *(a)–(g)* admits a (unique) non-contradictory complete branch $(\Pi, \mathbf{A})$ in some tableau iff $(\Pi, \mathbf{A})$ belongs to every complete tableau for Π and $\emptyset$. Hence, Theorem 5.1 as well as its generalizations to further language constructs provided in the sequel remain valid when replacing "every" by "some" in their second and their third item.[13] Yet before turning to extensions, let us illustrate the generic tableau rules in Figure 4 on a couple of examples.

For instance, consider a tableau with $\{a \leftarrow not\ a\}$ at its root. A cut on a yields two branches, one with $\boldsymbol{T}a$ and another one with $\boldsymbol{F}a$. The first branch can be closed by deducing $\boldsymbol{F}a$ via $N{\uparrow}$. To see this, observe that $sup_{\{\boldsymbol{T}a\}}(\{a \leftarrow not\ a\}, \{a\}, \emptyset) = \emptyset$ because $\boldsymbol{f}not\ a = \boldsymbol{T}a \in \{\boldsymbol{T}a\}$. That is, a must be false since all rules from which it could be derived are inapplicable. The second branch can be closed by deducing $\boldsymbol{T}a$ via $I{\uparrow}$, given that $\boldsymbol{t}not\ a = \boldsymbol{F}a \in \{\boldsymbol{F}a\}$. Hence, all branches of the tableau are contradictory, indicating that the unary program $\{a \leftarrow not\ a\}$ has no answer set.

For another example, consider the following unary program:

$$\Pi_7 \ = \ \{a \leftarrow not\ b;\ b \leftarrow not\ a;\ c \leftarrow not\ a\}$$

Cutting on c results in branches with $\boldsymbol{T}c$ and $\boldsymbol{F}c$, respectively. The first one can be extended by $\boldsymbol{t}not\ a = \boldsymbol{F}a$ via $N{\downarrow}$. Indeed, $sup_{\{\boldsymbol{T}c\}}(\Pi_7, \{c\}, \emptyset) = \{c \leftarrow not\ a\}$ tells us that $(c \leftarrow not\ a)$ is the only rule that allows for deriving c, which necessitates a to be false. To be more precise, we have that $\boldsymbol{f}not\ a = \boldsymbol{T}a \notin \{\boldsymbol{T}c\}$, and both $\overleftarrow{sup}_{\{\boldsymbol{T}c\}}(c, \{c\})$ and $\overrightarrow{sup}_{\{\boldsymbol{T}c\}}(not\ a, \emptyset)$ are satisfied. This shows that the proviso of $N{\downarrow}$ is established, so that we can deduce $\boldsymbol{t}not\ a = \boldsymbol{F}a$. Given $\boldsymbol{F}a$, we can further apply $I{\uparrow}$ or $I{\downarrow}$ to deduce $\boldsymbol{T}b$ and to so obtain a non-contradictory complete branch. The second branch with $\boldsymbol{F}c$ can be extended by $\boldsymbol{f}not\ a = \boldsymbol{T}a$ via $I{\downarrow}$. Given $\boldsymbol{T}a$, we deduce $\boldsymbol{F}b$, by $N{\uparrow}$ or $N{\downarrow}$, to obtain a second non-contradictory complete branch. The two complete branches, consisting of Π_7 along with $\{\boldsymbol{T}c, \boldsymbol{F}a, \boldsymbol{T}b\}$ and $\{\boldsymbol{F}c, \boldsymbol{T}a, \boldsymbol{F}b\}$, respectively, tell us that $\{b, c\}$ and $\{a\}$ are the two answer sets of Π_7.

Finally, consider the following unary program:

$$\Pi_8 \ = \ \{a \leftarrow b;\ b \leftarrow a;\ b \leftarrow c;\ c \leftarrow not\ d;\ d \leftarrow not\ c\}$$

Let us further investigate two non-contradictory complete branches generated as follows:

[13] The uniqueness of branches stated in the second item of Theorem 5.1 is trivial for unary programs. It becomes more interesting below when composite language constructs are assigned in addition to atoms.

Π_8	
$\boldsymbol{T}d$	(*Cut*$[atom(\Pi_8)]$)
$\boldsymbol{F}c$	$(N\uparrow, N\downarrow, U\uparrow, U\downarrow)$
$\boldsymbol{F}a$	$(U\uparrow)$
$\boldsymbol{F}b$	$(I\downarrow, N\uparrow, U\uparrow)$

Π_8	
$\boldsymbol{T}a$	(*Cut*$[atom(\Pi_8)]$)
$\boldsymbol{T}b$	$(I\uparrow, N\downarrow, U\downarrow)$
$\boldsymbol{T}c$	$(U\downarrow)$
$\boldsymbol{F}d$	$(N\uparrow, N\downarrow, U\uparrow, U\downarrow)$

We have chosen these branches for illustrating the application of the unfounded set rule ($U\uparrow$) and the well-founded set rule ($U\downarrow$), respectively. (Along branches, we indicate in parentheses all possible rule applications leading to the same result.) We first inspect the deduction of $\boldsymbol{F}a$ by $U\uparrow$ in the left branch. Taking the set $\{a,b\}$ (and its element a) makes us check whether $sup_{\{\boldsymbol{T}d,\boldsymbol{F}c\}}(\Pi_8,\{a,b\},\{a,b\})$ is empty. To this end, we have to investigate all rules that allow for deriving an atom in $\{a,b\}$ (as stipulated via $\overleftarrow{sup}_{\{\boldsymbol{T}d,\boldsymbol{F}c\}}(\alpha,\{a,b\})$). In view of $\boldsymbol{f}c = \boldsymbol{F}c \in \{\boldsymbol{T}d,\boldsymbol{F}c\}$, we have $(b \leftarrow c) \notin sup_{\{\boldsymbol{T}d,\boldsymbol{F}c\}}(\Pi_8,\{a,b\},\{a,b\})$, so that only $(a \leftarrow b)$ and $(b \leftarrow a)$ require further consideration. Because of $b \in \{a,b\}$ and $a \in \{a,b\}$, respectively, neither of these rules $(\alpha \leftarrow \beta)$ satisfies $\overrightarrow{sup}_{\{\boldsymbol{T}d,\boldsymbol{F}c\}}(\beta,\{a,b\})$, which leaves us with $sup_{\{\boldsymbol{T}d,\boldsymbol{F}c\}}(\Pi_8,\{a,b\},\{a,b\}) = \emptyset$. After $\boldsymbol{F}a$ has been deduced, it follows that $(b \leftarrow a) \notin sup_{\{\boldsymbol{T}d,\boldsymbol{F}c,\boldsymbol{F}a\}}(\Pi_8,\{b\},S')$ and $(b \leftarrow c) \notin sup_{\{\boldsymbol{T}d,\boldsymbol{F}c,\boldsymbol{F}a\}}(\Pi_8,\{b\},S')$ for $S' \subseteq \{b\}$. Hence, we can deduce $\boldsymbol{F}b$ by $N\uparrow$ or $U\uparrow$, or alternatively by means of $I\downarrow$ in view of rule $(a \leftarrow b)$ and $\boldsymbol{F}a$. On the other hand, the well-founded set inference of $\boldsymbol{T}c$ in the right branch requires a set of atoms, some of whose elements is true, such that only one rule can non-circularly support the set. Taking again $\{a,b\}$, since $\overrightarrow{sup}_{\{\boldsymbol{T}a,\boldsymbol{T}b\}}(b,\{a,b\})$ and $\overrightarrow{sup}_{\{\boldsymbol{T}a,\boldsymbol{T}b\}}(a,\{a,b\})$ do not hold, we get $sup_{\{\boldsymbol{T}a,\boldsymbol{T}b\}}(\Pi_8,\{a,b\},\{a,b\}) = \{b \leftarrow c\}$. The membership of $(b \leftarrow c)$ is justified by $\boldsymbol{f}c = \boldsymbol{F}c \notin \{\boldsymbol{T}a,\boldsymbol{T}b\}$ along with the fact that $\overleftarrow{sup}_{\{\boldsymbol{T}a,\boldsymbol{T}b\}}(b,\{a,b\})$ and $\overrightarrow{sup}_{\{\boldsymbol{T}a,\boldsymbol{T}b\}}(c,\{a,b\})$ hold. Since $(b \leftarrow c)$ is the only rule that can non-circularly support $\{a,b\}$, the presence of $\boldsymbol{T}a$ (or $\boldsymbol{T}b$) in the assignment necessitates $\boldsymbol{t}c = \boldsymbol{T}c$, as it is deduced by means of $U\downarrow$. Finally, $\boldsymbol{T}c$ allows us to further deduce $\boldsymbol{F}d$ by $N\downarrow$ or $U\downarrow$ in view of $(c \leftarrow not\ d)$, or alternatively by $N\uparrow$ or $U\uparrow$ in view of $(d \leftarrow not\ c)$.

5.3 Conjunctive Bodies

Having settled our constitutive generic framework, we now allow rule bodies to contain conjunctions. While rule bodies are often considered to be conjunctions (as in Section 3), we here take a slightly different perspective in viewing conjunctions as (simple) Boolean aggregates, which like atoms can be preceded by not. This gives us some first insights into the treatment of more sophisticated aggregates, such as cardinality constraints to be dealt with afterwards.

A *conjunction* over an alphabet $\mathcal{P}$ is an expression of the form $\{l_1,\ldots,l_n\}$, where l_i is an atomic literal for $1 \leq i \leq n$. We denote by $conj(\mathcal{P})$ the set of all conjunctions that can be constructed from atoms in $\mathcal{P}$. A rule $(\alpha \leftarrow \beta)$ such that α is an atomic literal and β is an atomic literal or a possibly negated conjunction of atomic literals is a *conjunctive rule*. A logic program is a *conjunctive program* if it consists of conjunctive rules. For defining the semantics of conjunctive programs, we add the following case to translation $\tau[\pi]$ in (7):

$$\tau[\pi] \;=\; \textstyle\bigwedge_{l\in\pi}\tau[l] \qquad \text{if } \pi \in conj(\mathcal{P})$$

For accommodating conjunctions within the generic tableau rules in Figure 4, we further extend the previous concepts in (9)–(14) in a straightforward way:

$$\begin{aligned} \overrightarrow{sup}_{\mathbf{A}}(\{l_1,\ldots,l_n\},S') \qquad & \text{if } \overrightarrow{sup}_{\mathbf{A}}(l,S') \text{ holds for every } l \in \{l_1,\ldots,l_n\} \\ max_{\mathbf{A}}(\{l_1,\ldots,l_n\},S') \;&=\; \textstyle\bigcup_{l\in\{l_1,\ldots,l_n\}} max_{\mathbf{A}}(l,S') \end{aligned}$$

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

(h) True Conjunction ($TC\uparrow$)

$$\frac{\boldsymbol{F}\{l_1,\ldots,l_{i-1},l_i,l_{i+1},\ldots,l_n\}\quad \boldsymbol{t}l_1,\ldots,\boldsymbol{t}l_{i-1},\boldsymbol{t}l_{i+1},\ldots,\boldsymbol{t}l_n}{\boldsymbol{f}l_i}$$

(i) Falsify Conjunction ($TC\downarrow$)

$$\frac{\boldsymbol{f}l_i}{\boldsymbol{F}\{l_1,\ldots,l_i,\ldots,l_n\}}$$

(j) False Conjunction ($FC\uparrow$)

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

(k) Justify Conjunction ($FC\downarrow$)

Fig. 5. Tableau rules for conjunctions.

Note that $max_{\mathbf{A}}(\{l_1,\ldots,l_n\},S')$ is still empty since $max_{\mathbf{A}}(l,S')=\emptyset$ for every atomic literal $l\in\{l_1,\ldots,l_n\}$. Thus, it has no effect yet, but this changes when adding cardinality constraints.

For a conjunctive program Π, we fix the domain of assignments $\mathbf{A}$ as well as the cut objects used by *Cut*$[\Gamma]$ to $dom(\mathbf{A})=\Gamma=atom(\Pi)\cup conj(\Pi)$, where $conj(\Pi)$ is the set of conjunctions occurring in Π. The additional tableau rules for handling conjunctions are shown in Figure 5. Their purpose is to ensure that $\boldsymbol{T}\{l_1,\ldots,l_n\}\in\mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}}\cap\mathcal{P}\models(\tau[l_1]\wedge\cdots\wedge\tau[l_n])$ holds for total assignments $\mathbf{A}$. By augmenting our generic calculus with the tableau rules in Figure 5, Theorem 5.1 extends to conjunctive programs.

THEOREM 5.2. *Let Π be a conjunctive program.*
Then, we have that Statement 1, 2, and 3 given in Theorem 5.1 hold for the tableau calculus consisting of the tableau rules (a)–(k).

It is interesting to note that conjunctive programs extend normal programs by admitting negative literals in heads of rules and rule bodies to be (default) negated conjunctions. This implies that normal programs are conjunctive and can thus be treated using the tableau rules *(a)–(k)*.[14] The naturally arising question is how generic tableau rules relate to the ones in Figure 1, which were specialized to normal programs. To answer it, Table I shows the inherent correspondences between both kinds of tableau rules. For instance, the tableau rule *FTA*, allowing for making derivable atoms true, achieves the same effect as the generic implication rule $I\uparrow$. In fact, we obtain the following correspondence result for normal programs.

PROPOSITION 5.3. *Let Π be a normal program, $\mathbf{A}$ an assignment, and F,G any pair of a basic tableau rule F and a generic tableau rule G belonging to the same line in Table I.*
Then, we have that

(*1*) $D_{\{F\}}(\Pi,\mathbf{A})=D_{\{G\}}(\Pi,\mathbf{A})$ *if* $F\notin\{$*BTA*, *WFJ*$[2^{atom(\Pi)}]\}$;
(*2*) $D_{\{\textit{BTA}\}}(\Pi,\mathbf{A})\supseteq D_{\{N\downarrow\}}(\Pi,\mathbf{A})$ *and, if* $D_{\{\textit{BTA}\}}(\Pi,\mathbf{A})\neq D_{\{N\downarrow\}}(\Pi,\mathbf{A})$, *then* $\mathbf{A}\cup D_{\{N\uparrow\}}(\Pi,\mathbf{A})$ *is contradictory;*
(*3*) $D_{\{\textit{WFJ}[2^{atom(\Pi)}]\}}(\Pi,\mathbf{A})\supseteq D_{\{U\downarrow\}}(\Pi,\mathbf{A})$ *and, if* $\boldsymbol{T}B\in D_{\{\textit{WFJ}[2^{atom(\Pi)}]\}}(\Pi,\mathbf{A})\setminus D_{\{U\downarrow\}}(\Pi,\mathbf{A})$, *then* $\mathbf{A}\cup D_{\{U\uparrow\}}(\Pi,\mathbf{A}\cup\{\boldsymbol{F}B\})$ *is contradictory.*

This shows that similar entries are deducible by either kind of tableau rules. A technical difference, though, is that, if *BTA* is applicable because of some $p\in\mathbf{A}^{\boldsymbol{T}}\cap atom(\Pi)$ such that

[14] Answer sets of $\tau[\Pi]$ match answer sets (as introduced in Section 2) of a normal program Π (cf. [Lifschitz 2008]).

Basic Tableau Rule			Generic Tableau Rule		
(c)	Forward True Atom	*FTA*	*(a)*	Implication	$I\uparrow$
(d)	Backward False Atom	*BFA*	*(b)*	Contraposition	$I\downarrow$
(g)	Forward False Atom	*FFA*	*(c)*	Negation	$N\uparrow$
(h)	Backward True Atom	*BTA*	*(d)*	Support	$N\downarrow$
(i)	Well-Founded Negation	*WFN*$[2^{atom(\Pi)}]$	*(e)*	Unfounded Set	$U\uparrow$
(j)	Well-Founded Justification	*WFJ*$[2^{atom(\Pi)}]$	*(f)*	Well-Founded Set	$U\downarrow$
(a)	Forward True Body	*FTB*	*(h)*	True Conjunction	$TC\uparrow$
(b)	Backward False Body	*BFB*	*(i)*	Falsify Conjunction	$TC\downarrow$
(e)	Forward False Body	*FFB*	*(j)*	False Conjunction	$FC\uparrow$
(f)	Backward True Body	*BTB*	*(k)*	Justify Conjunction	$FC\downarrow$

Table I. Correspondences between basic and generic tableau rules (for normal programs).

$body(p) \subseteq \mathbf{A}^{\boldsymbol{F}}$, then $N\downarrow$ is not applicable to p since $sup_{\mathbf{A}}(\Pi, \{p\}, \emptyset) = \emptyset$. In such a case, a contradictory assignment is obtained by applying $N\uparrow$. Furthermore, if $\boldsymbol{T}B$ is deduced by *WFJ*$[2^{atom(\Pi)}]$ in view of some $S \subseteq atom(\Pi)$ such that $\mathbf{A}^{\boldsymbol{T}} \cap S \neq \emptyset$ and $EB_\Pi(S) \setminus \mathbf{A}^{\boldsymbol{F}} \subseteq \{B\}$, there may be none or multiple rules $(p \leftarrow B)$ in Π for which $p \in S$, so that $|sup_{\mathbf{A}}(\Pi, S, S)| = 1$ is not guaranteed. In such a case, applying $U\uparrow$ wrt $\mathbf{A} \cup \{\boldsymbol{F}B\}$ yields a contradiction. That is, an entry $\boldsymbol{T}B$ deducible by *WFJ*$[2^{atom(\Pi)}]$ can also be obtained with the generic calculus by means of cutting (on B or its body literals) and closing branches with $\boldsymbol{F}B$ via $U\uparrow$, which yields $\boldsymbol{T}B$ in the single remaining branch. Hence, differences between *BTA* and $N\downarrow$ as well as *WFJ*$[2^{atom(\Pi)}]$ and $U\downarrow$ are merely technical, but not fundamental, discrepancies.

We further define the *generic image* of a basic calculus $\mathcal{T}$ as the generic calculus $\mathcal{T}'$ containing the *Cut* rules of $\mathcal{T}$ and the generic tableau rules associated with basic tableau rules in $\mathcal{T}$ according to Table I. Then, we obtain the following from Proposition 5.3.

PROPOSITION 5.4. *Let Π be a normal program, $\mathbf{A}$ an assignment, $\mathcal{T}$ a tableau calculus containing any subset of the tableau rules in Figure 1 for $\Omega = 2^{atom(\Pi)}$, and $\mathcal{T}'$ the generic image of $\mathcal{T}$.*

If FFA $\in \mathcal{T}$ or BTA $\notin \mathcal{T}$ and if WFJ$[\Omega] \in \mathcal{T}$ implies that $\{$FTB, FFB, WFN$[\Omega]$, Cut$[\Gamma]\} \subseteq \mathcal{T}$ for $\Gamma \subseteq atom(\Pi) \cup body(\Pi)$ such that $atom(\Pi) \subseteq \Gamma$ or $body(\Pi) \subseteq \Gamma$, then we have that the following holds:

(*1*) *For every complete tableau of $\mathcal{T}$ for Π and $\mathbf{A}$ with n branches, there is a complete tableau of $\mathcal{T}'$ for Π and $\mathbf{A}$ with the same non-contradictory branches and at most $(\max\{|atom(\Pi)|, |body(\Pi)|\} + 1) * n$ branches overall.*

(*2*) *Every (complete) tableau of $\mathcal{T}'$ for Π and $\mathbf{A}$ is a (complete) tableau of $\mathcal{T}$ for Π and $\mathbf{A}$.*

While every tableau of the generic image $\mathcal{T}'$ is likewise a tableau of $\mathcal{T}$, in view of Proposition 5.3, deductions by *BTA* or *WFJ*$[\Omega]$ may not be obtainable with $N\downarrow$ or $U\downarrow$, respectively. Such deductions can still be simulated by means of other generic tableau rules, possibly introducing a polynomial number of additional branches, for which the approximation given in the first item of Proposition 5.4 provides an upper limit.

To illustrate the correspondence between basic and generic tableau rules, reconsider the normal program Π_1 from Section 2. The tableau of $\mathcal{T}_{smodels}$ for Π_1 and the empty assignment shown in Figure 2 can also be generated by the generic image of $\mathcal{T}_{smodels}$, consisting of the generic tableau rules *(a)*–*(e)*, *(h)*–*(k)*, and *Cut*$[atom(\Pi)]$. (Note that, like $\mathcal{T}_{smodels}$, its generic image restricts cut objects to atoms, and it does not include $U\downarrow$ because $\mathcal{T}_{smodels}$ does not contain its associated basic tableau rule *WFJ*$[2^{atom(\Pi)}]$.) The generic tableau resembling the one in Figure 2 is shown in

$$
\begin{array}{llll}
 & a \leftarrow & & \\
 & c \leftarrow \mathit{not}\ b, \mathit{not}\ d & & \\
 & d \leftarrow a, \mathit{not}\ c & & \\
 & \boldsymbol{T}\emptyset & & (TC{\uparrow}) \\
 & \boldsymbol{T}a & & (I{\uparrow}) \\
 & \boldsymbol{F}b & & (N{\uparrow}) \\
\boldsymbol{T}c & & \boldsymbol{F}c & (\mathit{Cut}[\mathit{atom}(\Pi_1)]) \\
\boldsymbol{T}\{\mathit{not}\ b, \mathit{not}\ d\} & (N{\downarrow}) & \boldsymbol{F}\{\mathit{not}\ b, \mathit{not}\ d\} & (I{\downarrow}) \\
\boldsymbol{F}d & (FC{\downarrow}) & \boldsymbol{T}d & (TC{\downarrow}) \\
\boldsymbol{F}\{a, \mathit{not}\ c\} & (FC{\uparrow}) & \boldsymbol{T}\{a, \mathit{not}\ c\} & (TC{\uparrow})
\end{array}
$$

Fig. 6. Complete tableau of the generic image of $\mathcal{T}_{\mathit{smodels}}$ for Π_1 and the empty assignment.

Figure 6.[15] It is obtained by replacing the references to applied tableau rules adequately based on the mapping in Table I. Converse replacements of generic by basic tableau rules are also possible, provided that a logic program at hand is normal.

5.4 Cardinality Constraints

We further extend our generic tableau framework to logic programs including cardinality constraints [Simons et al. 2002]. The expressiveness of such programs, in terms of compact modeling, goes well beyond the one of normal (and conjunctive) programs. Hence, the integration of cardinality constraints sheds light on how to extend our generic framework to accommodate (sophisticated) aggregates.

A *cardinality constraint* over an alphabet $\mathcal{P}$ is an expression of the form $j\{l_1, \ldots, l_n\}k$, where l_i is an atomic literal for $1 \leq i \leq n$ and j, k are integers such that $0 \leq j \leq k \leq n$. We denote by $\mathit{card}(\mathcal{P})$ the set of all cardinality constraints that can be constructed from atoms in $\mathcal{P}$. For $v \in \mathcal{P} \cup \mathit{card}(\mathcal{P})$, we say that v and $\mathit{not}\ v$ are *cardinality literals*. A rule $(\alpha \leftarrow \beta)$ such that α is a cardinality literal and β is a cardinality literal or a possibly negated conjunction of cardinality literals is a *cardinality rule*. A logic program is a *cardinality program* if it consists of cardinality rules.

For cardinality constraints in heads of rules, we adopt the approach of [Simons et al. 2002; Ferraris and Lifschitz 2005; Liu and Truszczyński 2006] and interpret them as "choice constructs," meaning that atoms are not minimized within such cardinality constraints. To reflect the "lack of minimization," cardinality constraints in heads of rules necessitate an extended translation of cardinality programs to propositional theories.[16] Hence, we let $\mathit{atom}(j\{l_1, \ldots, l_n\}k) = \{l_1, \ldots, l_n\} \cap \mathcal{P}$ and replace the definition of $\tau[\Pi]$ in (6) by the following translation:

$$
\begin{array}{rcl}
\tau[\Pi] & = & \{\tau[\beta] \rightarrow \tau[\alpha] \mid (\alpha \leftarrow \beta) \in \Pi, \alpha \notin \mathit{card}(\mathcal{P})\} \\
 & \cup & \{\tau[\beta] \rightarrow \big(\tau[\alpha] \wedge \bigwedge_{p \in \mathit{atom}(\alpha)}(p \vee \neg p)\big) \mid (\alpha \leftarrow \beta) \in \Pi, \alpha \in \mathit{card}(\mathcal{P})\}
\end{array} \tag{15}
$$

Note that conjuncts $(p \vee \neg p)$ are tautological and thus neutral as regards the (classical) models of $\tau[\Pi]$. Given an interpretation X, they however justify the truth of all $p \in \mathit{atom}(\alpha) \cap X$ in $(\tau[\Pi])^X$, in which $\neg p$ is replaced by $\bot$. We further add another case to translation $\tau[\pi]$ in (7), which arises from the general aggregate semantics in [Ferraris 2005]:

[15] In Figure 6 and in the sequel, we skip set notation for conjunctions within bodies of rules, like $\emptyset$, $\{\mathit{not}\ b, \mathit{not}\ d\}$, and $\{a, \mathit{not}\ c\}$ in the bodies of rules in Π_1.

[16] Interpreting aggregates in heads of rules as "choice constructs" avoids an increase of computational complexity by one level in the polynomial time hierarchy. If derivable atoms were to be minimized, it would be straightforward to embed disjunctive programs (considered in Section 5.5) into cardinality programs.

$$\tau[\pi] \;=\; \textstyle\bigwedge_{L\subseteq\{l_1,\ldots,l_n\},|L|<j \text{ or } k<|L|}((\bigwedge_{l\in L}\tau[l]) \rightarrow (\bigvee_{l\in\{l_1,\ldots,l_n\}\setminus L}\tau[l]))$$
$$\text{if } \pi = j\{l_1,\ldots,l_n\}k \in \mathit{card}(\mathcal{P})$$

Since $\tau[j\{l_1,\ldots,l_n\}k]$ inspects individual subsets of $\{l_1,\ldots,l_n\}$, its size is, in general, exponential in n. Albeit the use of auxiliary atoms admits a polynomial translation of cardinality constraints to normal rules as, e.g., described in [Simons et al. 2002], compilation approaches incur a significant blow-up in space. Our proof-theoretic characterizations given below apply directly to cardinality constraints and thus avoid any such blow-up.

For a cardinality program Π, we fix the domain of assignments $\mathbf{A}$ as well as the cut objects used by $\mathit{Cut}[\Gamma]$ to $\mathit{dom}(\mathbf{A}) = \Gamma = \mathit{atom}(\Pi) \cup \mathit{conj}(\Pi) \cup \mathit{card}(\Pi)$, where $\mathit{card}(\Pi)$ is the set of cardinality constraints occurring in Π. Figure 7 shows the tableau rules augmenting those in Figure 4 and 5 to additionally handle cardinality constraints. These rules match truth values of atoms occurring in a cardinality constraint to the one assigned to the constraint as a whole, in order to ensure that $\boldsymbol{T}j\{l_1,\ldots,l_n\}k \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap \mathcal{P} \models \tau[j\{l_1,\ldots,l_n\}k]$ holds for total assignments $\mathbf{A}$.

As an example, consider the cardinality constraint $\gamma = 2\{a, b, c, \mathit{not}\ d, \mathit{not}\ e\}3$ including $n = 5$ literals, lower bound $j = 2$, and upper bound $k = 3$. For an assignment $\mathbf{A}$, tableau rule $TLU{\uparrow}$ allows for deducing $\boldsymbol{T}\gamma$ if at least $j = 2$ literals l of γ hold (i.e., $\boldsymbol{t}l \in \mathbf{A}$) and at least $n - k = 2$ literals l of γ are false (i.e., $\boldsymbol{f}l \in \mathbf{A}$). This applies, for instance, to the assignment $\mathbf{A}_1 = \{\boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{T}d, \boldsymbol{F}e\}$; hence, $\boldsymbol{T}\gamma$ can be deduced by $TLU{\uparrow}$. Indeed, the lower and the upper bound of γ are respected in every non-contradictory assignment that extends $\mathbf{A}_1$, no matter whether $\boldsymbol{T}c$ or $\boldsymbol{F}c$ is additionally included. Tableau rules $TL_U{\downarrow}$ and $T_LU{\downarrow}$ are the contrapositives of $TLU{\uparrow}$, ensuring that either the lower or the upper bound of γ is violated if $\boldsymbol{F}\gamma$ belongs to an assignment. For instance, $\boldsymbol{F}c$ and $\boldsymbol{T}e$ can be deduced by $TL_U{\downarrow}$ wrt the assignment $\mathbf{A}_2 = \{\boldsymbol{F}\gamma, \boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{T}d\}$. Observe that the upper bound $k = 3$ cannot be violated in non-contradictory extensions of $\mathbf{A}_2$, since $n - k = 2$ literals of γ are already false wrt $\mathbf{A}_2$ containing $\boldsymbol{F}b$ and $\boldsymbol{T}d$. On the other hand, $T_LU{\downarrow}$ allows for deducing $\boldsymbol{T}c$ and $\boldsymbol{F}e$ wrt $\{\boldsymbol{F}\gamma, \boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{F}d\}$ in order to violate the upper bound $k = 3$; the lower bound $j = 2$ cannot be violated because of $\boldsymbol{T}a$ and $\boldsymbol{F}d$. The remaining four tableau rules in Figure 7 allow for deducing $\boldsymbol{F}\gamma$ if its lower ($FL{\uparrow}$) or upper ($FU{\uparrow}$) bound is violated, or for making sure that the lower ($FL{\downarrow}$) and the upper ($FU{\downarrow}$) bound are respected if $\boldsymbol{T}\gamma$ belongs to an assignment. For instance, $\boldsymbol{F}\gamma$ can be deduced by $FL{\uparrow}$ wrt the assignment $\{\boldsymbol{F}b, \boldsymbol{F}c, \boldsymbol{T}d, \boldsymbol{T}e\}$, and by $FU{\uparrow}$ wrt $\{\boldsymbol{T}b, \boldsymbol{T}c, \boldsymbol{F}d, \boldsymbol{F}e\}$. Conversely, $FL{\downarrow}$ allows for deducing $\boldsymbol{T}a$ and $\boldsymbol{F}e$ wrt $\{\boldsymbol{T}\gamma, \boldsymbol{F}b, \boldsymbol{F}c, \boldsymbol{T}d\}$, and $FU{\downarrow}$ allows for deducing $\boldsymbol{F}a$ and $\boldsymbol{T}e$ wrt $\{\boldsymbol{T}\gamma, \boldsymbol{T}b, \boldsymbol{T}c, \boldsymbol{F}d\}$.

To integrate cardinality constraints into the generic setting of the tableau rules in Figure 4, we also need to extend the concepts in (9)–(14):

$$\overleftarrow{\mathit{sup}}_{\mathbf{A}}(j\{l_1,\ldots,l_n\}k, S) \quad \text{if } \{l_1,\ldots,l_n\} \cap S \neq \emptyset \text{ and } |\{l \in \{l_1,\ldots,l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\}| < k$$

$$\overrightarrow{\mathit{sup}}_{\mathbf{A}}(j\{l_1,\ldots,l_n\}k, S') \quad \text{if } |\{l \in \{l_1,\ldots,l_n\} \setminus S' \mid \boldsymbol{f}l \notin \mathbf{A}\}| \geq j$$

$$\mathit{min}_{\mathbf{A}}(j\{l_1,\ldots,l_n\}k, S) \;=\; \begin{cases} \{\boldsymbol{f}l \mid l \in \{l_1,\ldots,l_n\} \setminus S,\ \boldsymbol{t}l \notin \mathbf{A}\} \\ \quad \text{if } |\{l \in \{l_1,\ldots,l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\}| = k-1 \\ \emptyset \text{ if } |\{l \in \{l_1,\ldots,l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\}| \neq k-1 \end{cases}$$

$$\mathit{max}_{\mathbf{A}}(j\{l_1,\ldots,l_n\}k, S') \;=\; \begin{cases} \{\boldsymbol{t}l \mid l \in \{l_1,\ldots,l_n\} \setminus S',\ \boldsymbol{f}l \notin \mathbf{A}\} \\ \quad \text{if } |\{l \in \{l_1,\ldots,l_n\} \setminus S' \mid \boldsymbol{f}l \notin \mathbf{A}\}| = j \\ \emptyset \text{ if } |\{l \in \{l_1,\ldots,l_n\} \setminus S' \mid \boldsymbol{f}l \notin \mathbf{A}\}| \neq j \end{cases}$$

$$\frac{\boldsymbol{t}l_1,\ldots,\boldsymbol{t}l_j,\boldsymbol{f}l_{k+1},\ldots,\boldsymbol{f}l_n}{\boldsymbol{T}j\{l_1,\ldots,l_j,\ldots,l_{k+1},\ldots,l_n\}k}$$

(l) True Bounds ($TLU\uparrow$)

$$\frac{\boldsymbol{F}j\{l_1,\ldots,l_{j-1},l_j,\ldots,l_k,l_{k+1},\ldots,l_n\}k \quad \boldsymbol{t}l_1,\ldots,\boldsymbol{t}l_{j-1},\boldsymbol{f}l_{k+1},\ldots,\boldsymbol{f}l_n}{\boldsymbol{f}l_j,\ldots,\boldsymbol{f}l_k}$$

(m) Falsify Lower Bound ($TL_U\downarrow$)

$$\frac{\boldsymbol{F}j\{l_1,\ldots,l_j,l_{j+1},\ldots,l_{k+1},l_{k+2},\ldots,l_n\}k \quad \boldsymbol{t}l_1,\ldots,\boldsymbol{t}l_j,\boldsymbol{f}l_{k+2},\ldots,\boldsymbol{f}l_n}{\boldsymbol{t}l_{j+1},\ldots,\boldsymbol{t}l_{k+1}}$$

(n) Falsify Upper Bound ($T_LU\downarrow$)

$$\frac{\boldsymbol{f}l_j,\ldots,\boldsymbol{f}l_n}{\boldsymbol{F}j\{l_1,\ldots,l_j,\ldots,l_n\}k}$$

(o) False Lower Bound ($FL\uparrow$)

$$\frac{\boldsymbol{T}j\{l_1,\ldots,l_j,l_{j+1},\ldots,l_n\}k \quad \boldsymbol{f}l_{j+1},\ldots,\boldsymbol{f}l_n}{\boldsymbol{t}l_1,\ldots,\boldsymbol{t}l_j}$$

(p) Justify Lower Bound ($FL\downarrow$)

$$\frac{\boldsymbol{t}l_1,\ldots,\boldsymbol{t}l_{k+1}}{\boldsymbol{F}j\{l_1,\ldots,l_{k+1},\ldots,l_n\}k}$$

(q) False Upper Bound ($FU\uparrow$)

$$\frac{\boldsymbol{T}j\{l_1,\ldots,l_k,l_{k+1},\ldots,l_n\}k \quad \boldsymbol{t}l_1,\ldots,\boldsymbol{t}l_k}{\boldsymbol{f}l_{k+1},\ldots,\boldsymbol{f}l_n}$$

(r) Justify Upper Bound ($FU\downarrow$)

Fig. 7. Tableau rules for cardinality constraints.

Recall that $\overleftarrow{sup}_{\mathbf{A}}(\alpha,S)$ is used to determine whether a rule with head α can provide support for the atoms in S. If $\alpha = j\{l_1,\ldots,l_n\}k$, then some atom of S must belong to $\{l_1,\ldots,l_n\}$. Furthermore, if $\{l_1,\ldots,l_n\}\setminus S$ already contains k (or more) literals that hold wrt $\mathbf{A}$, then the addition of $\boldsymbol{T}p$ to $\mathbf{A}$ for $p\in\{l_1,\ldots,l_n\}\cap S$ would violate the upper bound k, so that the corresponding rule $(\alpha\leftarrow\beta)$ cannot support S. This also explains the false literals in $min_{\mathbf{A}}(j\{l_1,\ldots,l_n\}k,S)$ that can be deduced if $k-1$ literals of $\{l_1,\ldots,l_n\}\setminus S$ hold already. In addition, $\overrightarrow{sup}_{\mathbf{A}}(\beta,S')$ is used to verify whether a support via β is external to S'. If, for a rule $(\alpha\leftarrow\beta)$, either $\beta = j\{l_1,\ldots,l_n\}k$ or β is a conjunction such that $j\{l_1,\ldots,l_n\}k\in\beta$, then there must be enough non-false literals wrt $\mathbf{A}$ in $\{l_1,\ldots,l_n\}\setminus S'$ to achieve the lower bound j. If the number of such literals is exactly j, then all of them must hold for providing a support that is external to S'. This is expressed by $max_{\mathbf{A}}(j\{l_1,\ldots,l_n\}k,S')$.

For illustration, consider the following cardinality program:

$$\Pi_9 = \left\{\begin{array}{lll} r_1: & 0\{c,d,e\}3 \leftarrow & \\ r_2: & 1\{a,b\}2 \leftarrow c,d & r_4: \ 1\{b,d\}2 \leftarrow 1\{a,c\}2 \\ r_3: & 0\{a,d\}1 \leftarrow 1\{b,not\ e\}2 & r_5: \ 1\{a,d\}2 \leftarrow b \end{array}\right\}$$

Let $\mathbf{A} = \{\boldsymbol{T}a,\boldsymbol{F}c,\boldsymbol{F}\{c,d\}\}$, and note that tableau rule $U\downarrow$ (or $N\downarrow$) does not apply to the set $\{a\}$ since $sup_{\mathbf{A}}(\Pi_9,\{a\},\{a\}) = \{r_3,r_5\}$. We further consider the set $\{a,b\}$. Given that $\{c,d,e\}\cap\{a,b\} = \emptyset$ and $\boldsymbol{F}\{c,d\}\in\mathbf{A}$, we have that $r_1\notin sup_{\mathbf{A}}(\Pi_9,\{a,b\},\{a,b\})$ and $r_2\notin sup_{\mathbf{A}}(\Pi_9,\{a,b\},\{a,b\})$. Regarding the body of r_5, $b\in\{a,b\}$ is the reason for $\overrightarrow{sup}_{\mathbf{A}}(b,\{a,b\})$ not to hold. For the body of r_4, we have that $\{a,c\}\setminus\{a,b\} = \{c\}$ and $\boldsymbol{f}c = \boldsymbol{F}c\in\mathbf{A}$, so that there are no non-false literals in $\{a,c\}\setminus\{a,b\}$. That is, the lower bound 1 of $1\{a,c\}2$ cannot be achieved independently of $\{a,b\}$, and thus $\overrightarrow{sup}_{\mathbf{A}}(1\{a,c\}2,\{a,b\})$ does not

hold. We have now established that only r_3 is potentially contained in $sup_{\mathbf{A}}(\Pi_9, \{a,b\}, \{a,b\})$. As $not\ e$ in $1\{b, not\ e\}2$ is a non-false literal not belonging to $\{a,b\}$, $\overrightarrow{sup}_{\mathbf{A}}(1\{b, not\ e\}2, \{a,b\})$ holds. In addition, $\overleftarrow{sup}_{\mathbf{A}}(0\{a,d\}1, \{a,b\})$ holds because $\{a,d\} \cap \{a,b\} \neq \emptyset$ and $\boldsymbol{t}d = \boldsymbol{T}d \notin \mathbf{A}$. This shows that $sup_{\mathbf{A}}(\Pi_9, \{a,b\}, \{a,b\}) = \{r_3\}$. As entries deducible by $U{\downarrow}$, we obtain $\boldsymbol{t}1\{b, not\ e\}2 = \boldsymbol{T}1\{b, not\ e\}2$, $min_{\mathbf{A}}(0\{a,d\}1, \{a,b\}) = \{\boldsymbol{f}d\} = \{\boldsymbol{F}d\}$, and $max_{\mathbf{A}}(1\{b, not\ e\}2, \{a,b\}) = \{\boldsymbol{t}not\ e\} = \{\boldsymbol{F}e\}$. In fact, if $\boldsymbol{T}d$ or $\boldsymbol{T}e$ had been contained in $\mathbf{A}$, we would have obtained $sup_{\mathbf{A}}(\Pi_9, \{a,b\}, \{a,b\}) = \emptyset$, so that deducing $\boldsymbol{F}a$ by $U{\uparrow}$ would have led to a contradiction. Also note that the only answer set of Π_9 compatible with both $\boldsymbol{T}a$ and $\boldsymbol{F}c$, $\{a,b\}$, does not include d and e.

In view of the generalizations of $\overleftarrow{sup}$, $\overrightarrow{sup}$, min, and max provided above, Theorem 5.1 extends to cardinality programs.

THEOREM 5.5. *Let* Π *be a cardinality program.*
Then, we have that Statement 1, 2, and 3 given in Theorem 5.1 hold for the tableau calculus consisting of the tableau rules (a)–(r).

On the example of cardinality constraints, we have demonstrated the full granularity of our generic tableau rules in Figure 4, making use of the generalized definitions of $\overleftarrow{sup}$, $\overrightarrow{sup}$, min, and max. Importantly, we have extended the domain of assignments (and cut objects) to include cardinality constraints. As a matter of fact, this admits the addition of the tableau rules in Figure 7, matching the truth value of a cardinality constraint to those of its constituents, without any modification of tableau rules dealing with other language constructs, like the ones for conjunctions in Figure 5. The same methodology could be applied to logic programs further including *smodels*' weight constraints [Simons et al. 2002] or *dlv*'s aggregates [Faber et al. 2008]. Notably, the approach of *clasp* to incorporate extended language constructs into its algorithmic framework [Gebser et al. 2009] is closely related: *clasp* extends assignments as well as unit propagation to composite language constructs and keeps track of reasons for inferences, which can be extracted from tableau rules by using the scheme described in Section 4.3.

5.5 Disjunctive Heads

The extension of normal programs' syntax and semantics by allowing heads of rules to be (proper) disjunctions of atoms is one of the earliest generalizations of answer set semantics [Gelfond and Lifschitz 1991]. Due to admitting the (unrestricted) use of disjunction, the inherent computational complexity of important reasoning tasks increases from the first to the second level of the polynomial time hierarchy [Eiter and Gottlob 1995; Leone et al. 2006]. As we show in the following, it is nonetheless possible to extend our generic tableau framework by allowing (proper) disjunctions in heads of rules, without imposing any restrictions on computational complexity.

A *disjunction* over an alphabet $\mathcal{P}$ is an expression of the form $\{l_1; \ldots; l_n\}$, where l_i is an atomic literal for $1 \leq i \leq n$. We denote by $disj(\mathcal{P})$ the set of all disjunctions that can be constructed from atoms in $\mathcal{P}$. For $v \in \mathcal{P} \cup card(\mathcal{P}) \cup disj(\mathcal{P})$, v and $not\ v$ are *disjunctive literals*. A rule $(\alpha \leftarrow \beta)$ such that α is a disjunctive literal and β is a cardinality literal or a possibly negated conjunction of cardinality literals is a *disjunctive rule*. A logic program is a *disjunctive program* if it consists of disjunctive rules.

In contrast to cardinality constraints serving as "choice constructs," the common semantics for disjunctions relies on the minimization of derivable atoms. Hence, we adhere to the definition of $\tau[\Pi]$ in (15) and just add another case to $\tau[\pi]$ in (7):

$$\tau[\pi] \;=\; \bigvee_{l \in \{l_1, \ldots, l_n\}} \tau[l] \qquad \text{if } \pi = \{l_1; \ldots; l_n\} \in disj(\mathcal{P})$$

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

(s) True Disjunction ($TD\uparrow$)

$$\frac{\boldsymbol{F}\{l_1;\ldots;l_n\}}{\boldsymbol{f}l_1,\ldots,\boldsymbol{f}l_n}$$

(t) Falsify Disjunction ($TD\downarrow$)

$$\frac{\boldsymbol{f}l_1,\ldots,\boldsymbol{f}l_n}{\boldsymbol{F}\{l_1;\ldots;l_n\}}$$

(u) False Disjunction ($FD\uparrow$)

$$\frac{\boldsymbol{T}\{l_1;\ldots;l_{i-1};l_i;l_{i+1};\ldots;l_n\}\quad \boldsymbol{f}l_1,\ldots,\boldsymbol{f}l_{i-1},\boldsymbol{f}l_{i+1},\ldots,\boldsymbol{f}l_n}{\boldsymbol{t}l_i}$$

(v) Justify Disjunction ($FD\downarrow$)

Fig. 8. Tableau rules for disjunctions.

We further extend the concepts in (9)–(14) to disjunctive heads:

$$\begin{aligned}\overleftarrow{sup}_{\mathbf{A}}(\{l_1;\ldots;l_n\},S) \qquad & \text{if } \{l_1,\ldots,l_n\}\cap S\neq\emptyset \text{ and}\\ & \{l\in\{l_1,\ldots,l_n\}\setminus S\mid \boldsymbol{t}l\in\mathbf{A}\}=\emptyset\\ min_{\mathbf{A}}(\{l_1;\ldots;l_n\},S) \;=\; & \{\boldsymbol{f}l\mid l\in\{l_1,\ldots,l_n\}\setminus S\}\end{aligned}$$

Observe that the notion of support, $\overleftarrow{sup}_{\mathbf{A}}(\{l_1;\ldots;l_n\},S)$, is closely related to, yet simpler than, the corresponding concept for cardinality constraints, given that disjunctions do not possess an upper bound k. Rather, support requires all literals of $\{l_1,\ldots,l_n\}\setminus S$ to be false; this condition can be established by means of $min_{\mathbf{A}}(\{l_1;\ldots;l_n\},S)$, used by generic tableau rules $N\downarrow$ and $U\downarrow$.

For a disjunctive program Π, we fix the domain of assignments $\mathbf{A}$ as well as the cut objects used by $Cut[\Gamma]$ to $dom(\mathbf{A})=\Gamma=atom(\Pi)\cup conj(\Pi)\cup card(\Pi)\cup disj(\Pi)$, where $disj(\Pi)$ is the set of disjunctions occurring in Π. The additional tableau rules for handling disjunctions are shown in Figure 8. Their purpose is to ensure that $\boldsymbol{T}\{l_1;\ldots;l_n\}\in\mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}}\cap\mathcal{P}\models(\tau[l_1]\vee\cdots\vee\tau[l_n])$ holds for total assignments $\mathbf{A}$. The tableau calculus for disjunctive programs is obtained by adding the rules in Figure 8 to the ones in Figure 4, 5, and 7. Then, Theorem 5.1 extends to disjunctive programs.

THEOREM 5.6. *Let Π be a disjunctive program.*
Then, we have that Statement 1, 2, and 3 given in Theorem 5.1 hold for the tableau calculus consisting of the tableau rules (a)–(v).

As with conjunctive programs that are slightly more general than normal ones, we have a parallel relationship between our and the traditional concept of a disjunctive program [Gelfond and Lifschitz 1991], given that we admit (default) negation in front of and within a disjunction.[17] However, as the equivalences discussed in [Lifschitz et al. 1999] show, rules with negative formulas in the head can be rewritten such that occurrences of negation are limited to rule bodies. In our setting, the generic tableau rules in Figure 4 tolerate negative literals in heads of rules, so that they can be admitted without difficulties. In view of this, the class of disjunctive programs we consider here is expressive enough to represent any nested program [Lifschitz et al. 1999] (in which heads and bodies of rules are allowed to be formulas).

As regards computational complexity of reasoning tasks, e.g., answer set existence, which can increase due to disjunctions in heads of rules, it does not impose any extra difficulties in specifying

[17] Unlike [Gelfond and Lifschitz 1991], we do not consider classical negation, but it could be handled easily by compilation.

tableau rules. Rather, elevated complexity manifests itself in the hardness of verifying the application conditions of tableau rules, namely, the ones of $U\uparrow$ and $U\downarrow$ dealing with unfounded sets. While such conditions can be checked in polynomial time for cardinality programs (cf. [Simons et al. 2002]), determining a (non-empty) unfounded set is NP-complete for disjunctive programs (cf. [Leone et al. 1997]). However, it is interesting to note that the condition $sup_{\mathbf{A}}(\Pi, S, S) = \emptyset$, checked in the proviso of $U\uparrow$, along with Theorem 5.6 contribute a definition of unfounded sets for disjunctive programs including cardinality constraints. Restricting $\mathbf{A}$ to entries over atoms only also yields a counterpart of interpretation-based unfounded sets [Van Gelder et al. 1991; Leone et al. 1997] for such disjunctive programs. (It merely requires replacing the support condition $\boldsymbol{f}\beta \notin \mathbf{A}$ in (8) by an evaluation of β wrt assigned atoms.) To our knowledge, there is no direct unfounded set definition (not relying on compilation to basic language constructs) for the class of disjunctive programs considered here,[18] yet soundness and completeness results (like Theorem 5.6) in the presence of $U\uparrow$ inherently contribute unfounded set definitions for extended classes of logic programs.

6. PROOF COMPLEXITY

In Section 4, we have seen that native ASP solvers largely coincide on their propagation rules and differ primarily in the usage of *Cut*. In this section, we analyze the relative efficiency of tableau calculi wrt different *Cut* rules. We start by taking $\mathcal{T}_{smodels}$, $\mathcal{T}_{nomore}$, and $\mathcal{T}_{nomore++}$ (defined in Section 3) into account, all using the deterministic tableau rules *(a)–(i)* in Figure 1 but applying *Cut* to either $atom(\Pi)$, $body(\Pi)$, or both of them. These three calculi are of particular interest as they closely characterize strategies of existing ASP solvers, viz. *smodels* (and *dlv*), *nomore*, and *nomore++*. After in Section 6.1 dealing with calculi aiming at normal programs, in Section 6.2, we extend our analysis to generic calculi and the impact of *Cut* wrt extended language constructs.

For comparing tableau calculi, we use the concept of *proof complexity* [Cook and Reckhow 1979] and evaluate the relative efficiency of calculi on unsatisfiable logic programs (having no answer set) in terms of *minimal* refutations. The size of a tableau is determined in the standard way by the number of its nodes (program rules and entries).[19] A tableau calculus $\mathcal{T}$ is *not polynomially simulated* [Beame and Pitassi 1998; Järvisalo et al. 2005] by another calculus $\mathcal{T}'$ if there is an infinite (witnessing) family $\{\Pi^n\}$ of unsatisfiable logic programs such that the asymptotic size of minimal refutations of $\mathcal{T}'$ for its members is exponentially greater than with $\mathcal{T}$. A tableau calculus $\mathcal{T}$ is *exponentially stronger* than a tableau calculus $\mathcal{T}'$ if $\mathcal{T}'$ is polynomially simulated by $\mathcal{T}$ (for refutations of $\mathcal{T}'$, there are refutations of $\mathcal{T}$ of up to a polynomial same asymptotic size), but not vice versa. Two tableau calculi are *efficiency-incomparable* if neither one is polynomially simulated by the other.

6.1 Tableaux for Normal Logic Programs

In what follows, we provide infinite families of unsatisfiable normal programs witnessing that neither $\mathcal{T}_{nomore}$ is polynomially simulated by $\mathcal{T}_{smodels}$, nor vice versa. This means that, on certain

[18] In [Faber 2005; Faber et al. 2011], occurrences of aggregates are limited to rule bodies. Furthermore, the semantics proposed there is not based on the logic of here-and-there, since (default) negated aggregates like $not\ 0\{a\}0$ are treated differently. Hence, important properties that can be verified in the logic of here-and-there (e.g., strong equivalence [Lifschitz et al. 2001]) do not carry forward to the semantics proposed in [Faber 2005; Faber et al. 2011].

[19] The determining factor for the asymptotic size of minimal refutations is the number of required *Cut* applications, that is, the number of branches that need to be investigated, because the depth of each branch is bounded by the input size. Also note that proof complexity says nothing about the difficulty of finding minimal refutations; rather, it provides a lower bound on the efficiency of proof-finding procedures, independent of heuristic influences.

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

Fig. 9. Families $\{\Pi_a^n\}$, $\{\Pi_b^n\}$, and $\{\Pi_c^n\}$ of normal programs.

normal programs, restricting *Cut* to only either atoms or bodies leads to exponentially greater (optimal) search space traversals of atom- or rule-based ASP solvers in comparison to their counterparts. The following results state the existence of witnessing families.

PROPOSITION 6.1. *There is an infinite family $\{\Pi^n\}$ of normal programs such that*

(*1*) *the size of minimal refutations of $\mathcal{T}_{nomore}$ for Π^n is asymptotically linear in n;*
(*2*) *the size of minimal refutations of $\mathcal{T}_{smodels}$ for Π^n is asymptotically exponential in n.*

PROPOSITION 6.2. *There is an infinite family $\{\Pi^n\}$ of normal programs such that*

(*1*) *the size of minimal refutations of $\mathcal{T}_{smodels}$ for Π^n is asymptotically linear in n;*
(*2*) *the size of minimal refutations of $\mathcal{T}_{nomore}$ for Π^n is asymptotically exponential in n.*

Family $\{\Pi_a^n \cup \Pi_c^n\}$ witnesses Proposition 6.1, and $\{\Pi_b^n \cup \Pi_c^n\}$ witnesses Proposition 6.2 (see Figure 9). The reason why $\mathcal{T}_{smodels}$ does not admit compact refutations for $\Pi_a^n \cup \Pi_c^n$ is that its proofs must exhaustively investigate symmetric alternatives obtained by cutting on atoms a_i or b_i. In fact, minimal refutations of $\mathcal{T}_{smodels}$ for $\Pi_a^n \cup \Pi_c^n$ are of the shape sketched in Figure 10. While an initial cut on x yields an immediate contradiction in the branch with $\boldsymbol{F}x$, branches with $\boldsymbol{T}x$ can only be completed after adding $n-1$ entries of the form $\boldsymbol{F}\{a_i, b_i\}$ for $1 \leq i \leq n$. However, *Cut*$[atom(\Pi_a^n \cup \Pi_c^n)]$ does not allow for introducing such entries, so that they must be generated indirectly, extending branches obtained by cutting on atoms a_i or b_i. But cascaded applications of *Cut*$[atom(\Pi_a^n \cup \Pi_c^n)]$ yield a subtableau with 2^{n-1} branches below $\boldsymbol{T}x$ (and $\boldsymbol{F}\{not\ x\}$), whose leaves are indicated at the bottom of Figure 10. Given that exponentially many branches are required, the size of minimal refutations of $\mathcal{T}_{smodels}$ for $\Pi_a^n \cup \Pi_c^n$ is asymptotically exponential in n. Unlike this, the use of *Cut*$[body(\Pi_a^n \cup \Pi_c^n)]$ admits linear refutations for $\Pi_a^n \cup \Pi_c^n$ with $\mathcal{T}_{nomore}$, like the one sketched in Figure 11. In such a refutation, cuts on $\{not\ x\}$ or $\{a_i, b_i\}$ for $1 \leq i \leq n$ yield immediate contradictions in branches with $\boldsymbol{T}\{not\ x\}$ or $\boldsymbol{T}\{a_i, b_i\}$, respectively. In fact, only n applications of *Cut*$[body(\Pi_a^n \cup \Pi_c^n)]$ are required in total, so that there are linear refutations with $n+1$ branches overall.

The situation that $\mathcal{T}_{nomore}$ dominates $\mathcal{T}_{smodels}$ is reversed with programs of the family $\{\Pi_b^n \cup \Pi_c^n\}$, where Figure 12 sketches a minimal refutation of $\mathcal{T}_{nomore}$ for $\Pi_b^n \cup \Pi_c^n$. In this refutation, only the initial cut on $\{c_1, \ldots, c_n, not\ y\}$ yields an immediate contradiction in the branch with $\boldsymbol{T}\{c_1, \ldots, c_n, not\ y\}$. Then, cuts on rule bodies $\{not\ a_i\}$ (or $\{not\ b_i\}$) must be cascaded to deduce $\boldsymbol{T}c_i$ in each of the resulting branches. Only after $n-1$ entries of the form $\boldsymbol{T}c_i$ for $1 \leq i \leq n$ have been generated, the leaves indicated at the bottom of Figure 12 are obtained. In view of symmetry, the subtableau below $\boldsymbol{F}\{c_1, \ldots, c_n, not\ y\}$ (and $\boldsymbol{F}y$) necessarily includes 2^{n-1} branches, so that the size of minimal refutations of $\mathcal{T}_{nomore}$ for $\Pi_b^n \cup \Pi_c^n$ is asymptotically exponential in n. Unlike this, cuts on c_i, admitted with $\mathcal{T}_{smodels}$, yield immediate contradictions in branches with $\boldsymbol{F}c_i$, and the same applies to a branch with $\boldsymbol{T}y$. Thus, the refutation of $\mathcal{T}_{smodels}$ for $\Pi_b^n \cup \Pi_c^n$ sketched in Figure 13 involves only n applications of *Cut*$[atom(\Pi_b^n \cup \Pi_c^n)]$ in total, so that there are linear refutations with $n+1$ branches overall.

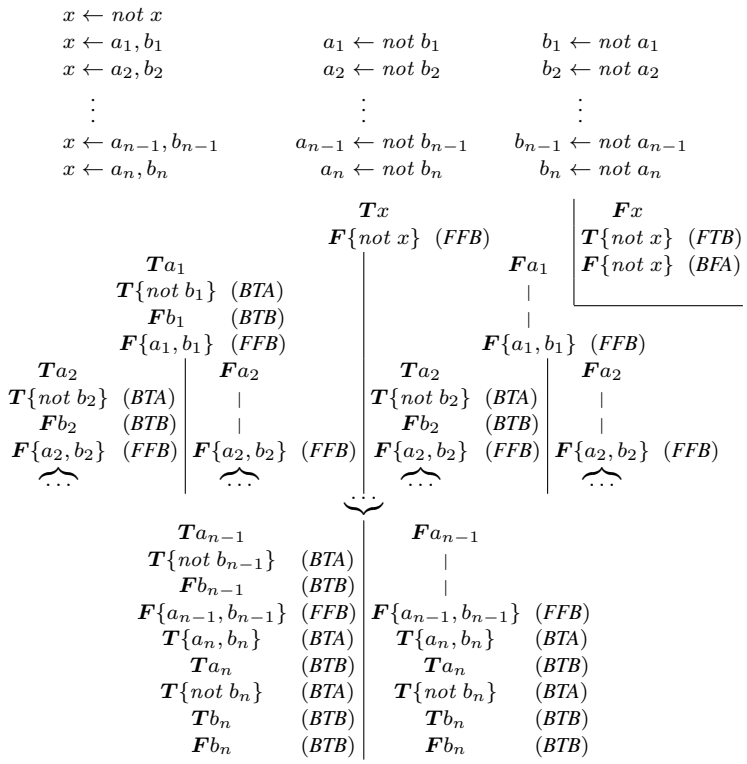

Fig. 10. A minimal refutation of $\mathcal{T}_{smodels}$ for $\Pi_a^n \cup \Pi_c^n$, using *Cut*$[atom(\Pi_a^n \cup \Pi_c^n)]$.

Notably, empirical evidence for divergent proof complexities of $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ has been given in [Anger et al. 2006], and [Gebser and Schaub 2006a] shows that conflict resolution as performed by *smodels*-based ASP solver *smodels*$_{cc}$ is unable to compensate for the exponential proof complexity of $\mathcal{T}_{smodels}$ on family $\{\Pi_a^n \cup \Pi_c^n\}$. In view of mutual exponential separations, the next result is immediately obtained from Proposition 6.1 and 6.2.

COROLLARY 6.3. *Tableau calculi $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ are efficiency-incomparable.*

Given that refutations of $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ are refutations of $\mathcal{T}_{nomore++}$ as well, we have that $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ are both polynomially simulated by $\mathcal{T}_{nomore++}$. Hence, the following is an immediate consequence of Corollary 6.3.

COROLLARY 6.4. *Tableau calculus $\mathcal{T}_{nomore++}$ is exponentially stronger than both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$.*

The major implication of Corollary 6.4 is that, on certain normal programs, a priori restricting *Cut* to only either atoms or bodies necessitates exponentially greater search space traversals than unrestricted *Cut*. Note that the phenomenon of exponentially greater proof complexity in comparison to $\mathcal{T}_{nomore++}$ does not, depending on the program family, apply to one of $\mathcal{T}_{smodels}$ or $\mathcal{T}_{nomore}$ alone. Rather, the infinite family

$$\left\{ \begin{array}{l} (\Pi_a^n \setminus \{x \leftarrow not\ x\}) \cup (\Pi_b^n \setminus \{y \leftarrow c_1, \ldots, c_n, not\ y\}) \cup \\ \{y \leftarrow c_1, \ldots, c_n, not\ x, not\ y\} \cup \Pi_c^n \end{array} \right\}$$

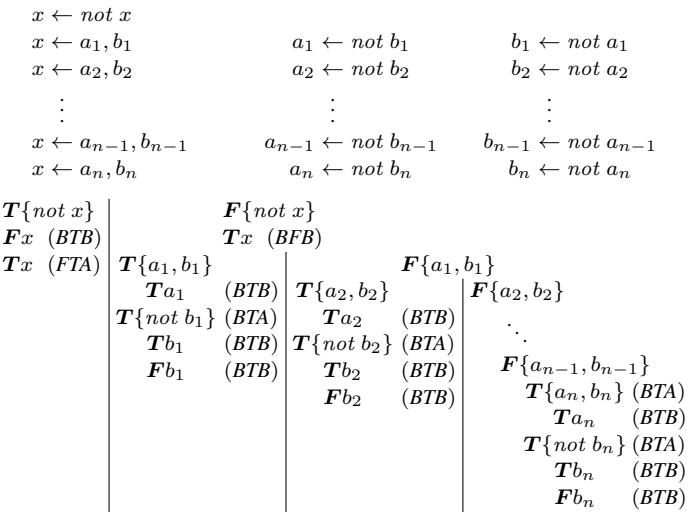

Fig. 11. A minimal refutation of $\mathcal{T}_{nomore}$ for $\Pi_a^n \cup \Pi_c^n$, using *Cut*$[body(\Pi_a^n \cup \Pi_c^n)]$.

is such that the asymptotic size of minimal refutations of both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ is exponential in n, while $\mathcal{T}_{nomore++}$ still admits refutations of linear size. Here, with $\mathcal{T}_{smodels}$, it is easy to prove that x needs to be true, while checking that this cannot be the case requires investigating symmetric alternatives by cutting on atoms a_i or b_i. On the other hand, with $\mathcal{T}_{nomore}$, it is easy to verify that x and y need to be false, but recognizing that $c_1, \ldots, c_n$ must be true, so that the rule $(y \leftarrow c_1, \ldots, c_n, not\ x, not\ y)$ is unsatisfied, requires exhaustive cutting on rule bodies $\{not\ a_i\}$ or $\{not\ b_i\}$. Unlike this, with $\mathcal{T}_{nomore++}$, it is easy to refute $c_1, \ldots, c_n$ to be false as well as x and y to be true, and the only remaining alternative yields $(y \leftarrow c_1, \ldots, c_n, not\ x, not\ y)$ as unsatisfied rule. Hence, $\mathcal{T}_{nomore++}$, but neither $\mathcal{T}_{smodels}$ nor $\mathcal{T}_{nomore}$, admits linear refutations for members of the above family.

Note that our proof complexity results are unimpaired by failed-literal detection [Freeman 1995] as, e.g., applied by *smodels*. In fact, failed-literal detection can be mimicked by means of *Cut*, so that proof complexity, already assuming an optimal heuristic, stays unaffected. In view of Corollary 4.3 and similarities between *smodels* and *dlv* on normal programs [Giunchiglia et al. 2008], the proof complexity of tableau calculus $\mathcal{T}_{smodels}$ indeed provides a lower bound on the efficiency of *smodels* and *dlv* (when applied to normal programs).

6.2 Generic Tableaux for Composite Language Constructs

After considering normal programs and tableau calculi for them, we now turn to the generic calculus (cf. Figure 4) and extensions thereof. In view of Proposition 5.4, the results in Section 6.1 (and the fact that minimal refutations of $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ for members of $\{\Pi_a^n \cup \Pi_c^n\}$ or $\{\Pi_b^n \cup \Pi_c^n\}$, respectively, are not significantly reduced when adding *WFJ*$[2^{atom(\Pi)}]$) allow us to immediately conclude that, for conjunctive programs, the generic calculi $\{$*(a)–(f)*, *(h)–(k)*, *Cut*$[atom(\Pi)]\}$ and $\{$*(a)–(f)*, *(h)–(k)*, *Cut*$[conj(\Pi)]\}$ are efficiency-incomparable, while $\{$*(a)–(f)*, *(h)–(k)*, *Cut*$[atom(\Pi) \cup conj(\Pi)]\}$ is exponentially stronger than both of them. We below extend the analysis of relative efficiency wrt different *Cut* rules to more general logic programs, addressing the question whether cutting on further language constructs, namely, cardinality constraints and disjunctions, leads to more powerful tableau calculi. Beforehand, note that cutting on atoms

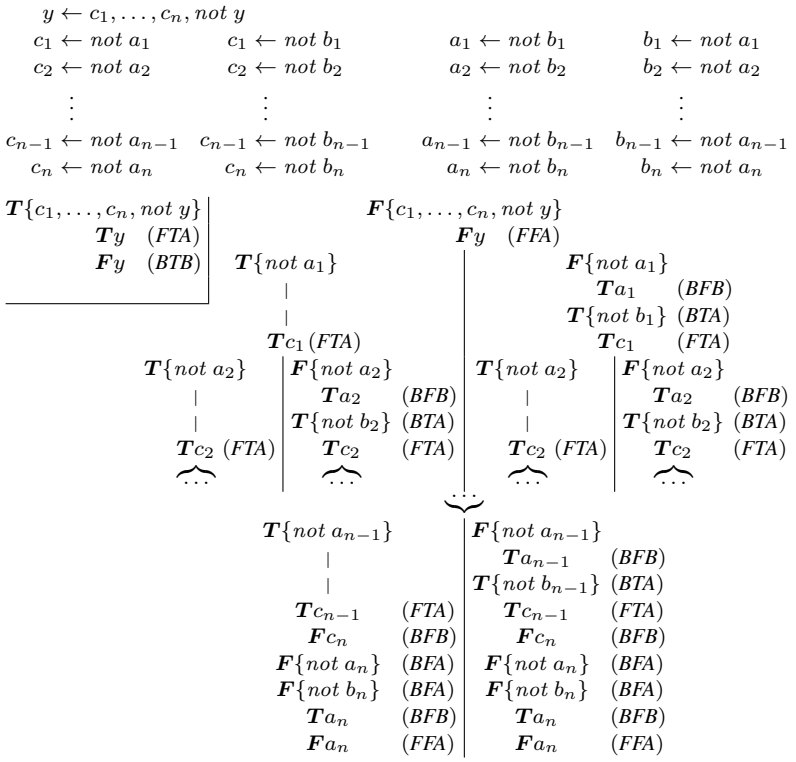

Fig. 12. A minimal refutation of $\mathcal{T}_{nomore}$ for $\Pi_b^n \cup \Pi_c^n$, using *Cut*$[body(\Pi_b^n \cup \Pi_c^n)]$.

is sufficient for obtaining complete calculi even in the presence of language extensions, given that the truth values of composite constructs can be deduced from atomic literals by (deterministic) tableau rules. For cardinality constraints and disjunctions, this is possible using the tableau rules in Figure 7 and 8, respectively.

For cardinality programs Π, we consider $\mathcal{T}_{card} = \{$*(a)–(f)*, *(h)–(r)*, *Cut*$[atom(\Pi) \cup conj(\Pi) \cup card(\Pi)]\}$ and $\mathcal{T}_{conj} = \{$*(a)–(f)*, *(h)–(r)*, *Cut*$[atom(\Pi) \cup conj(\Pi)]\}$. Both calculi contain all deterministic tableau rules dealing with cardinality programs; the difference is that cutting on cardinality constraints is allowed with $\mathcal{T}_{card}$, but not with $\mathcal{T}_{conj}$. As every tableau of $\mathcal{T}_{conj}$ is a tableau of $\mathcal{T}_{card}$ as well, it is clear that $\mathcal{T}_{conj}$ is polynomially simulated by $\mathcal{T}_{card}$. The following result states that the converse does not hold.

PROPOSITION 6.5. *Tableau calculus $\mathcal{T}_{card}$ is exponentially stronger than $\mathcal{T}_{conj}$.*

Proposition 6.5 is witnessed by the infinite family $\{\Pi_c^n \cup \Pi_d^n\}$ of unsatisfiable cardinality programs, where Π_c^n is shown in Figure 9 and Π_d^n is as follows:

$$\Pi_d^n \ = \ \{z \leftarrow 1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\}$$

For $\Pi_c^n \cup \Pi_d^n$, a branch containing $\boldsymbol{F}1\{a_i, b_i\}2$ is easy to refute because $\boldsymbol{F}a_i$ and $\boldsymbol{F}b_i$ can be deduced by tableau rule $TL_U{\downarrow}$ (cf. Figure 7), yielding an immediate contradiction since a_i and b_i cannot jointly be false (cf. Π_c^n in Figure 9). The unrestricted *Cut* rule of $\mathcal{T}_{card}$ can be used to exploit this by cutting on $1\{a_i, b_i\}2$ for $1 \leq i \leq n$, so that the resulting minimal refutations are of

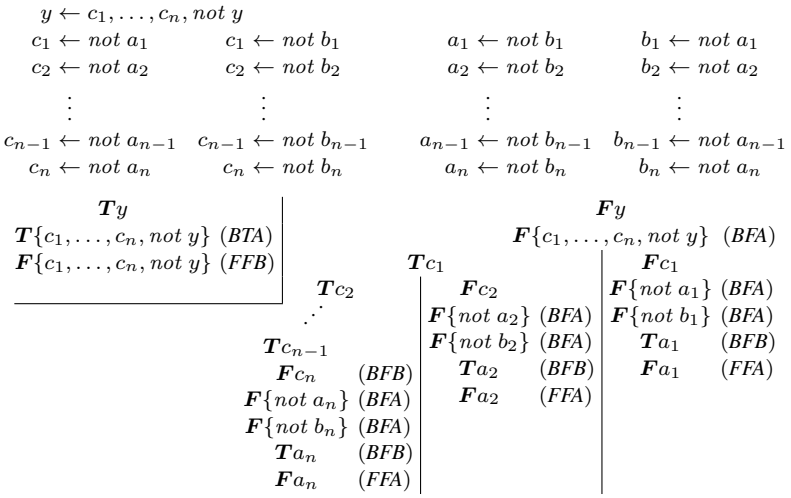

Fig. 13. A minimal refutation of $\mathcal{T}_{smodels}$ for $\Pi_b^n \cup \Pi_c^n$, using *Cut*$[atom(\Pi_b^n \cup \Pi_c^n)]$.

asymptotically linear size in n. In fact, when replacing cuts on y and c_i by cuts on z and $1\{a_i, b_i\}2$, respectively, minimal refutations of $\mathcal{T}_{card}$ for $\Pi_c^n \cup \Pi_d^n$ are of the shape sketched in Figure 13 (also assuming that applications of *BFA* to deduce $\boldsymbol{F}\{not\ a_i\}$ and $\boldsymbol{F}\{not\ b_i\}$ are replaced by $TL_U{\downarrow}$, deducing $\boldsymbol{F}a_i$ as well as $\boldsymbol{F}b_i$, and that $\boldsymbol{T}a_i$ is deduced by $I{\downarrow}$ instead of *BFB*). In contrast, with $\mathcal{T}_{conj}$, *Cut* must be applied to atoms a_i or b_i (or to bodies $\{not\ a_i\}$ or $\{not\ b_i\}$), while deducing $\boldsymbol{T}1\{a_i, b_i\}2$ in each of the resulting branches. Such refutations are of the same shape as the one sketched in Figure 12: an initial cut on $\{1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\}$ (or z) yields an immediate contradiction in the branch with $\boldsymbol{T}\{1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\}$ (or $\boldsymbol{T}z$) as well as a subtableau with 2^{n-1} branches below $\boldsymbol{F}\{1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\}$ (and $\boldsymbol{F}z$).

The practical consequence of Proposition 6.5 is that ASP solvers dealing with cardinality constraints can gain significant speed-ups by branching on them. Notably, the compilation of rules with cardinality constraints to so-called "basic constraint rules" [Simons et al. 2002], as done by grounders like *lparse* [Syrjänen] and *gringo* [Gebser et al. 2011], introduces auxiliary atoms abbreviating cardinality constraints. This allows ASP solvers to (implicitly) branch on cardinality constraints, even if case analyses are restricted to atoms as, e.g., in *smodels*. Unlike this, our tableau framework does not rely on any compilation and considers cardinality constraints as structural entities that can be used deliberately for branching.

Regarding disjunctive programs, we have that occurrences of disjunctions are limited to heads of rules (to avoid involved definitions of $\overrightarrow{sup}$ and max). If this restriction were dropped, program Π_d^n could be rewritten using disjunctions $\{a_i; b_i\}$ rather than $1\{a_i, b_i\}2$ for $1 \leq i \leq n$. This would yield the same exponential separation wrt *Cut* rules with and without disjunctions, respectively, as observed on cardinality constraints. However, with disjunctions $\{l_1; \ldots; l_n\}$ limited to heads of rules, the difficulty is that the information gained in the case of $\boldsymbol{T}\{l_1; \ldots; l_n\}$ is weak: it is exploited by tableau rule $FD{\downarrow}$ (cf. Figure 8) only if all but one of the literals $l_1, \ldots, l_n$ have already been assigned to false. In view of this, it is complicated (if at all possible) to come up with an infinite family of unsatisfiable disjunctive programs such that cutting on disjunctions is the source of an exponential separation. Hence, we leave the question open whether cutting on disjunctive heads admits exponentially smaller refutations than obtainable without it.

As noted in Section 4, existing ASP solvers, such as *smodels*, *nomore*, and *nomore++*, lack backward propagation for unfounded sets. Their associated tableau calculi reflect this by not including any variant of tableau rule *WFJ*$[\Omega]$ (cf. Figure 1) or $U\downarrow$ (cf. Figure 4), respectively. On the other hand, SAT-based and native conflict-driven learning ASP solvers, such as *assat*, *clasp*, *cmodels*, *sag*, and *smodels*$_{cc}$, are able to perform this kind of propagation relative to recorded loop formulas. This brings our attention to the question whether omitting some inferences deteriorates proof complexity. In what follows, we denote by $R\uparrow$ and $R\downarrow$ the forward and backward variant, respectively, of any of the (deterministic) tableau rules in Figure 4, 5, 7, and 8. For a tableau calculus $\mathcal{T}$, we say that $\mathcal{T}' \subseteq \mathcal{T}$ is an *approximation* of $\mathcal{T}$ if $\mathcal{T} \setminus \mathcal{T}' \subseteq \{R\downarrow \mid R\uparrow \in \mathcal{T}'\}$. (We assume that $TLU\uparrow \in \mathcal{T}'$ if $\{TL_U\downarrow, T_LU\downarrow\} \cap (\mathcal{T} \setminus \mathcal{T}') \neq \emptyset$, given that $TLU\uparrow$ has *two* backward counterparts.) That is, if $\mathcal{T}$ contains both $R\uparrow$ and $R\downarrow$, an approximation $\mathcal{T}'$ of $\mathcal{T}$ is allowed to drop $R\downarrow$. It is clear that every approximation $\mathcal{T}'$ of $\mathcal{T}$ is polynomially simulated by $\mathcal{T}$. Assuming *Cut* to be sufficiently powerful, the next result shows that the converse holds as well.

PROPOSITION 6.6. *Let* Π *be a disjunctive program,* $\mathcal{T}$ *a tableau calculus containing any subset of the tableau rules (a)–(v), and* $\mathcal{T}'$ *an approximation of* $\mathcal{T}$.

If Cut$[\Gamma] \in \mathcal{T}'$ *such that* $atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \subseteq \Gamma$*, then we have that* $\mathcal{T}$ *is polynomially simulated by* $\mathcal{T}'$.

In fact, an inference due to $R\downarrow$ can be mimicked by cutting on the consequent of $R\downarrow$. Then, one of the two resulting branches becomes contradictory when applying $R\uparrow$. But recall that proof complexity assumes an optimal heuristic, determining the "right" objects to cut on. As an optimal heuristic is inaccessible in practice, it is certainly advantageous to implement $R\downarrow$ within an ASP solver whenever it can be done efficiently.

7. RELATED WORK

Our work is inspired by the one of Järvisalo, Junttila, and Niemelä [Järvisalo et al. 2005], who use tableau methods for investigating Boolean circuit satisfiability checking. Although their target is different from ours, both approaches have aspects in common. First, both use tableau methods for characterizing DPLL-style search. Second, they analyze proof complexity wrt cut rules characterizing different concepts of case analyses.

As pointed out in [Hähnle 2001], DPLL is very similar to the propositional version of the KE tableau calculus; both are closely related to weak connection tableaux with atomic cut. Tableau-based characterizations of logic programming are elaborated upon in [Fitting 1994]. Pearce, de Guzmán, and Valverde [Pearce et al. 2000] provide a tableau calculus for automated theorem proving in equilibrium logic; its inference rules admit the decomposition of input formulas, while our calculi aim at the formation of (Boolean) assignments. Further tableau approaches to non-monotonic reasoning are summarized in [Olivetti 1999] and [Dix et al. 2001].

General investigations into propositional proof complexity [Cook and Reckhow 1979], in particular, the one of (UN)SAT, can be found in [Beame and Pitassi 1998]. Notably, recent results on CDCL [Beame et al. 2004; Pipatsrisawat and Darwiche 2011], the state-of-the-art complete algorithm for SAT solving, indicate its strong correlation to general resolution. Although DPLL amounts to a weaker form of resolution, called tree-like (cf. [Beame and Pitassi 1998]), Järvisalo and Oikarinen [Järvisalo and Oikarinen 2008] show that an extension of our basic tableau framework, admitting the addition of "redundant" rules to a logic program, is as powerful as extended resolution (under standard translations between ASP and SAT, viz. completion and the reduction in [Niemelä 1999]). The complexity considerations in [Giunchiglia et al. 2008] also build on the proximity of traditional ASP solving methods to DPLL.

Regarding inference systems for ASP, Bonatti [Bonatti 2001] describes a resolution method for skeptical reasoning. Unlike our approach, it is based on query-oriented top-down evaluation, as also performed in SLDNF resolution (cf. [Lloyd 1987]). Similarly, the proof schemes investigated by Marek and Remmel in [Marek and Remmel 2008] are closely related to SLDNF resolution. Operator-based characterizations of propagation and choice techniques of ASP solvers can be found in [Faber 2002; Simons et al. 2002; Anger et al. 2005; Calimeri et al. 2006; Konczak et al. 2006]. They are more coarse-grained than our tableau rules, which aim at characterizing fundamental inference steps. Although SAT-based and native conflict-driven learning ASP solvers are usually described in terms of algorithms [Lin and Zhao 2004; Ward and Schlipf 2004; Giunchiglia et al. 2006; Lin et al. 2006; Gebser et al. 2007], the fact that they identify reasons for conflicts yields a close relationship to our tableau-based approach. In principle, (immediate) reasons can easily be extracted from tableau rules, albeit our tableau frameworks do not incorporate conflict-driven learning. The state-based calculus by Lierler [Lierler 2011], inspired by a similar approach [Nieuwenhuis et al. 2006] to Satisfiability Modulo Theories (SMT; [Barrett et al. 2009]), allows for characterizing several atom-based ASP solvers that incorporate conflict-driven learning.

A major issue in ASP solving is the treatment of unfounded sets [Van Gelder et al. 1991; Leone et al. 1997], which can be captured by loop formulas [Lin and Zhao 2004; Lee 2005]. As the number of (non-redundant) loop formulas may be exponential [Lifschitz and Razborov 2006],[20] ASP solvers use dedicated procedures to check [Simons et al. 2002; Calimeri et al. 2006] and possibly also extract [Lin and Zhao 2004; Giunchiglia et al. 2006; Lin et al. 2006; Anger et al. 2006; Gebser et al. 2007; Drescher et al. 2008] (violated) loop formulas relative to assignments. To our knowledge, no existing ASP solver implements backward inference via tableau rule *WFJ*, unless loop formulas have been recorded. Unfortunately, the approach in this direction suggested in [Chen et al. 2008; 2009] is computationally too complex (quadratic) to be beneficial in practice.[21] Our generic tableau framework does not distinguish loops (in tableau rules $U\uparrow$ and $U\downarrow$), which could however be done based on loops for propositional theories [Ferraris et al. 2006]. Loops and loop formulas for first-order normal programs have been defined in [Chen et al. 2006; Lee and Meng 2008]. There are also direct characterizations (not referring to grounding) of answer sets or stable models, respectively, for first-order theories [Pearce and Valverde 2005; Ferraris et al. 2007]. To our knowledge, they have not yet been used as a basis for proof-theoretic frameworks for the construction of answer sets, albeit Pearce and Valverde [Pearce and Valverde 2006] axiomatize entailment, (strong) equivalence, and validity in quantified here-and-there logics.

8. DISCUSSION

In contrast to the area of SAT, where the proof-theoretic foundations of SAT solvers are well-understood (cf. [Beame and Pitassi 1998; Beame et al. 2004; Pipatsrisawat and Darwiche 2011]), the literature on ASP solvers is generally too specific in terms of procedures or particular solving strategies. We addressed this deficiency by introducing tableau frameworks that provide us with

[20] Lifschitz and Razborov [Lifschitz and Razborov 2006] show that, under widely accepted assumptions in complexity theory, any semantics-preserving polynomial translation of normal programs to propositional theories must extend the input vocabulary. For instance, *lp2sat* [Janhunen and Niemelä 2011] implements a sub-quadratic translation based on a binary representation of level mappings [Janhunen 2006; Niemelä 2008] (which are considered wrt "non-tight" programs); the SMT-based approach of *lp2diff* [Janhunen et al. 2009] harnesses difference logic [Nieuwenhuis and Oliveras 2005] to encode level mappings in linear space.

[21] While known implementations of the well-founded operator take quadratic (worst case) time in the size of a program (cf. [Berman et al. 1995]), the atoms deducible by *WFN* (in a fixed branch) can be computed in linear time, e.g., by means of the Dowling-Gallier algorithm [Dowling and Gallier 1984].

formal means for characterizing and analyzing computations of ASP solvers. This is accomplished by associating specific tableau calculi with the approaches of ASP solvers, rather than their solving procedures. In fact, tableau calculi abstract from implementation details and admit identifying fundamental inference patterns. The latter can, in principle, be exploited to precisely render the constraints propagated by a solver in order to use them, e.g., for conflict-driven learning.

The explicit representation of rule bodies and further composite language constructs, such as cardinality constraints and disjunctions, in assignments has several benefits. For one, it allows us to characterize SAT-based and also native atom- or rule-based ASP solving approaches in a closer fashion. In fact, even in atom-based solvers, such as *dlv*, *smodels*, and *smodels*$_{cc}$, which (logically) work on assignments over atoms only, inferences rely on the valuations of rule bodies (cf. Section 4). Hence, the decision of whether or not to include composite language constructs in assignments mainly affects the available cut objects. In this respect, the consideration of atoms as well as rule bodies may lead to exponentially smaller (best-case) complexity than obtained with restricted approaches. This also applies to cardinality constraints in logic programs, while it is open whether branching on disjunctive heads can be the source of an exponential separation. The potential of exponential proof complexity decreases due to extending the range of cut objects is also confirmed by related investigations (cf. [Järvisalo et al. 2005; Järvisalo and Oikarinen 2008; Järvisalo and Junttila 2009]). However, it is well-known that uncontrolled cut applications are prone to inefficiency, and restricting them to (sub)formulas occurring in the input showed to be an effective way to "tame" the cut [D'Agostino et al. 1999]. Our tableau calculi adopt such input restrictions, which is in line with the fact that current ASP solvers do not "invent" new cut objects.

The simple class of normal programs is, in principle, sufficient to represent all NP-problems in ASP [Marek and Truszczyński 1991], and it can be regarded as the core language shared by virtually all ASP solvers. On the other hand, practical experience shows that language extensions, such as *dlv*'s aggregates [Faber et al. 2008] or *smodels*' cardinality and weight constraints [Simons et al. 2002], are important for effective modeling (cf. [Truszczyński 2007]). Hence, we presented a generic tableau framework and illustrated its extension to composite language constructs on two (sophisticated) examples: cardinality constraints and disjunctive heads. The corresponding inference rules follow two major objectives: first, characterizing models of logic programs and, second, verifying that true atoms are non-circularly supported. Different notions of support are possible, given that atoms derived via composite language constructs in heads of rules may be subject to minimization, as with disjunctions, or not, as with cardinality constraints allowing for "choices." Such issues must be settled first in order to then devise appropriate inference patterns.[22]

For conflict-driven learning ASP solvers, it is not only important to know valid inferences, but also how the propagated constraints look like. Our generic tableau framework provides means to study such aspects of composite language constructs in the course of specifying tableau rules for them, and it also provides a ready-to-use basis for checking the soundness and completeness of sophisticated inference patterns. Interestingly, conditions allowing for the falsification of atoms that cannot be non-circularly supported inherently characterize unfounded sets for extended classes of logic programs. In particular, we are unaware of any pre-existing direct unfounded set definition (not relying on compilation to basic language constructs) for disjunctive programs admitting cardinality constraints in heads as well as bodies of rules, while the proviso of generic tableau rule $U\uparrow$ provides such a definition in view of Theorem 5.6. Based on our methodology, the consideration of unfounded sets could be extended to further composite language constructs.

[22]The available language constructs also affect computational complexity; for instance, it increases by one level in the polynomial time hierarchy with disjunctive heads or negative weights within weight constraints (cf. [Ferraris 2005]).

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library by visiting the following URL: `http://www.acm.org/pubs/citations/journals/tocl/20YY-V-N/p1-URLend`.

Acknowledgments. We are grateful to Christian Anger, Philippe Besnard, Martin Brain, Yuliya Lierler, Robert Mercer, Richard Tichy, and the anonymous referees for many helpful suggestions. The first author especially acknowledges the members of his thesis committee, Gerhard Brewka, Tomi Janhunen, and Torsten Schaub, for their invaluable support.

This work was partially funded by the German Science Foundation (DFG) under grants SCHA 550/8-1/2.

REFERENCES

ANGER, C., GEBSER, M., JANHUNEN, T., AND SCHAUB, T. 2006. What's a head without a body? In *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI'06)*, G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, Eds. IOS Press, 769–770.

ANGER, C., GEBSER, M., LINKE, T., NEUMANN, A., AND SCHAUB, T. 2005. The `nomore++` approach to answer set solving. In *Proceedings of the Twelfth International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, G. Sutcliffe and A. Voronkov, Eds. Lecture Notes in Artificial Intelligence, vol. 3835. Springer-Verlag, 95–109.

ANGER, C., GEBSER, M., AND SCHAUB, T. 2006. Approaching the core of unfounded sets. In *Proceedings of the Eleventh International Workshop on Nonmonotonic Reasoning (NMR'06)*, J. Dix and A. Hunter, Eds. Number IFI-06-04 in Technical Report Series. Clausthal University of Technology, Institute for Informatics, 58–66.

APT, K., BLAIR, H., AND WALKER, A. 1987. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann Publishers, Chapter 2, 89–148.

BABOVICH, Y. AND LIFSCHITZ, V. 2003. Computing answer sets using program completion. Unpublished draft; available at `http://www.cs.utexas.edu/users/tag/cmodels.html`.

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

BARAL, C., GRECO, G., LEONE, N., AND TERRACINA, G., Eds. 2005. *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*. Lecture Notes in Artificial Intelligence, vol. 3662. Springer-Verlag.

BARRETT, C., SEBASTIANI, R., SESHIA, S., AND TINELLI, C. 2009. Satisfiability modulo theories. See Biere et al. [2009], Chapter 26, 825–885.

BEAME, P., KAUTZ, H., AND SABHARWAL, A. 2004. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research 22*, 319–351.

BEAME, P. AND PITASSI, T. 1998. Propositional proof complexity: Past, present, and future. *Bulletin of the European Association for Theoretical Computer Science 65*, 66–89.

BERMAN, K., SCHLIPF, J., AND FRANCO, J. 1995. Computing the well-founded semantics faster. In *Proceedings of the Third International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'95)*, V. Marek and A. Nerode, Eds. Lecture Notes in Artificial Intelligence, vol. 928. Springer-Verlag, 113–126.

BIERE, A., HEULE, M., VAN MAAREN, H., AND WALSH, T., Eds. 2009. *Handbook of Satisfiability*. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press.

BONATTI, P. 2001. Resolution for skeptical stable model semantics. *Journal of Automated Reasoning 27,* 4, 391–421.

BREWKA, G. AND LANG, J., Eds. 2008. *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*. AAAI Press.

CALIMERI, F., FABER, W., PFEIFER, G., AND LEONE, N. 2006. Pruning operators for disjunctive logic programming systems. *Fundamenta Informaticae 71,* 2-3, 183–214.

CHEN, X., JI, J., AND LIN, F. 2008. Computing loops with at most one external support rule. See Brewka and Lang [2008], 401–410.

CHEN, X., JI, J., AND LIN, F. 2009. Computing loops with at most one external support rule for disjunctive logic programs. See Erdem et al. [2009], 130–144.

CHEN, Y., LIN, F., WANG, Y., AND ZHANG, M. 2006. First-order loop formulas for normal logic programs. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, P. Doherty, J. Mylopoulos, and C. Welty, Eds. AAAI Press, 298–307.

CLARK, K. 1978. Negation as failure. In *Logic and Data Bases*, H. Gallaire and J. Minker, Eds. Plenum Press, 293–322.

COOK, S. AND RECKHOW, R. 1979. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic 44,* 1, 36–50.

D'AGOSTINO, M., GABBAY, D., HÄHNLE, R., AND POSEGGA, J., Eds. 1999. *Handbook of Tableau Methods*. Kluwer Academic Publishers.

DAVIS, M., LOGEMANN, G., AND LOVELAND, D. 1962. A machine program for theorem-proving. *Communications of the ACM 5*, 394–397.

DAVIS, M. AND PUTNAM, H. 1960. A computing procedure for quantification theory. *Journal of the ACM 7*, 201–215.

DIX, J., FURBACH, U., AND NIEMELÄ, I. 2001. Nonmonotonic reasoning: Towards efficient calculi and implementations. See Robinson and Voronkov [2001], 1241–1354.

DOWLING, W. AND GALLIER, J. 1984. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming 1*, 267–284.

DRESCHER, C., GEBSER, M., GROTE, T., KAUFMANN, B., KÖNIG, A., OSTROWSKI, M., AND SCHAUB, T. 2008. Conflict-driven disjunctive answer set solving. See Brewka and Lang [2008], 422–432.

EÉN, N. AND SÖRENSSON, N. 2004. An extensible SAT-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, E. Giunchiglia and A. Tacchella, Eds. Lecture Notes in Computer Science, vol. 2919. Springer-Verlag, 502–518.

EITER, T. AND GOTTLOB, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence 15,* 3-4, 289–323.

ERDEM, E., LIN, F., AND SCHAUB, T., Eds. 2009. *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*. Lecture Notes in Artificial Intelligence, vol. 5753. Springer-Verlag.

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

FABER, W. 2005. Unfounded sets for disjunctive logic programs with arbitrary aggregates. See Baral et al. [2005], 40–52.

FABER, W., PFEIFER, G., AND LEONE, N. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence 175,* 1, 278–298.

FABER, W., PFEIFER, G., LEONE, N., DELL'ARMI, T., AND IELPA, G. 2008. Design and implementation of aggregate functions in the DLV system. *Theory and Practice of Logic Programming 8,* 5-6, 545–580.

FERRARIS, P. 2005. Answer sets for propositional theories. See Baral et al. [2005], 119–131.

FERRARIS, P., LEE, J., AND LIFSCHITZ, V. 2006. A generalization of the Lin-Zhao theorem. *Annals of Mathematics and Artificial Intelligence 47,* 1-2, 79–101.

FERRARIS, P., LEE, J., AND LIFSCHITZ, V. 2007. A new perspective on stable models. See Veloso [2007], 372–379.

FERRARIS, P. AND LIFSCHITZ, V. 2005. Weight constraints as nested expressions. *Theory and Practice of Logic Programming 5,* 1-2, 45–74.

FITTING, M. 1994. Tableaux for logic programming. *Journal of Automated Reasoning 13,* 2, 175–188.

FITTING, M. 2002. Fixpoint semantics for logic programming: A survey. *Theoretical Computer Science 278,* 1-2, 25–51.

FREEMAN, J. 1995. Improvements to Propositional Satisfiability Search Algorithms. Ph.D. thesis, University of Pennsylvania.

GARCIA DE LA BANDA, M. AND PONTELLI, E., Eds. 2008. *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*. Lecture Notes in Computer Science, vol. 5366. Springer-Verlag.

GEBSER, M., KAMINSKI, R., KAUFMANN, B., AND SCHAUB, T. 2009. On the implementation of weight constraint rules in conflict-driven ASP solvers. In *Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09)*, P. Hill and D. Warren, Eds. Lecture Notes in Computer Science, vol. 5649. Springer-Verlag, 250–264.

GEBSER, M., KAMINSKI, R., KÖNIG, A., AND SCHAUB, T. 2011. Advances in *gringo* series 3. In *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, J. Delgrande and W. Faber, Eds. Lecture Notes in Artificial Intelligence, vol. 6645. Springer-Verlag, 345–351.

GEBSER, M., KAUFMANN, B., NEUMANN, A., AND SCHAUB, T. 2007. Conflict-driven answer set solving. See Veloso [2007], 386–392.

GEBSER, M., KAUFMANN, B., NEUMANN, A., AND SCHAUB, T. 2008. Advanced preprocessing for answer set solving. In *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, M. Ghallab, C. Spyropoulos, N. Fakotakis, and N. Avouris, Eds. IOS Press, 15–19.

GEBSER, M., LEE, J., AND LIERLER, Y. 2007. Head-elementary-set-free logic programs. In *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, C. Baral, G. Brewka, and J. Schlipf, Eds. Lecture Notes in Artificial Intelligence, vol. 4483. Springer-Verlag, 149–161.

GEBSER, M. AND SCHAUB, T. 2006a. Characterizing ASP inferences by unit propagation. In *Proceedings of the First International Workshop on Search and Logic: Answer Set Programming and SAT (LaSh'06)*, E. Giunchiglia, V. Marek, D. Mitchell, and E. Ternovska, Eds. 41–56.

GEBSER, M. AND SCHAUB, T. 2006b. Tableau calculi for answer set programming. In *Proceedings of the Twenty-second International Conference on Logic Programming (ICLP'06)*, S. Etalle and M. Truszczyński, Eds. Lecture Notes in Computer Science, vol. 4079. Springer-Verlag, 11–25.

GEBSER, M. AND SCHAUB, T. 2007. Generic tableaux for answer set programming. In *Proceedings of the Twenty-third International Conference on Logic Programming (ICLP'07)*, V. Dahl and I. Niemelä, Eds. Lecture Notes in Computer Science, vol. 4670. Springer-Verlag, 119–133.

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., LIERLER, Y., AND MARATEA, M. 2006. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning 36,* 4, 345–377.

GIUNCHIGLIA, E. AND MARATEA, M. 2005. On the relation between answer set and SAT procedures (or, between cmodels and smodels). In *Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05)*, M. Gabbrielli and G. Gupta, Eds. Lecture Notes in Computer Science, vol. 3668. Springer-Verlag, 37–51.

HÄHNLE, R. 2001. Tableaux and related methods. See Robinson and Voronkov [2001], 100–178.

JANHUNEN, T. 2006. Some (in)translatability results for normal logic programs and propositional theories. *Journal of Applied Non-Classical Logics 16,* 1-2, 35–86.

JANHUNEN, T. AND NIEMELÄ, I. 2011. Compact translations of non-disjunctive answer set programs to propositional clauses. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, M. Balduccini and T. Son, Eds. Lecture Notes in Computer Science, vol. 6565. Springer-Verlag, 111–130.

JANHUNEN, T., NIEMELÄ, I., AND SEVALNEV, M. 2009. Computing stable models via reductions to difference logic. See Erdem et al. [2009], 142–154.

JÄRVISALO, M. AND JUNTTILA, T. 2009. Limitations of restricted branching in clause learning. *Constraints 14,* 3, 325–356.

JÄRVISALO, M., JUNTTILA, T., AND NIEMELÄ, I. 2005. Unrestricted vs restricted cut in a tableau method for Boolean circuits. *Annals of Mathematics and Artificial Intelligence 44,* 4, 373–399.

JÄRVISALO, M. AND OIKARINEN, E. 2008. Extended ASP tableaux and rule redundancy in normal logic programs. *Theory and Practice of Logic Programming 8,* 5-6, 691–716.

KONCZAK, K., LINKE, T., AND SCHAUB, T. 2006. Graphs and colorings for answer set programming. *Theory and Practice of Logic Programming 6,* 1-2, 61–106.

LEE, J. 2005. A model-theoretic counterpart of loop formulas. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, L. Kaelbling and A. Saffiotti, Eds. Professional Book Center, 503–508.

LEE, J. AND MENG, Y. 2008. On loop formulas with variables. See Brewka and Lang [2008], 444–453.

LEONE, N., PFEIFER, G., FABER, W., 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. 2011. Abstract answer set solvers with learning. *Theory and Practice of Logic Programming 11,* 2-3, 135–169.

LIFSCHITZ, V. 2008. Twelve definitions of a stable model. See Garcia de la Banda and Pontelli [2008], 37–51.

LIFSCHITZ, V., PEARCE, D., AND VALVERDE, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic 2,* 4, 526–541.

LIFSCHITZ, V. AND RAZBOROV, A. 2006. Why are there so many loop formulas? *ACM Transactions on Computational Logic 7,* 2, 261–268.

LIFSCHITZ, V., TANG, L., AND TURNER, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25,* 3-4, 369–389.

LIN, F. AND ZHAO, Y. 2004. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence 157,* 1-2, 115–137.

LIN, Z., ZHANG, Y., AND HERNANDEZ, H. 2006. Fast SAT-based answer set solver. In *Proceedings of the Twenty-first National Conference on Artificial Intelligence (AAAI'06)*, Y. Gil and R. Mooney, Eds. AAAI Press, 92–97.

LINKE, T., ANGER, C., AND KONCZAK, K. 2002. More on `nomore`. In *Proceedings of the Eighth European Conference on Logics in Artificial Intelligence (JELIA'02)*, S. Flesca, S. Greco, N. Leone, and G. Ianni, Eds. Lecture Notes in Computer Science, vol. 2424. Springer-Verlag, 468–480.

LIU, L. AND TRUSZCZYŃSKI, M. 2006. Properties and applications of programs with monotone and convex constraints. *Journal of Artificial Intelligence Research 27*, 299–334.

LLOYD, J. 1987. *Foundations of Logic Programming*. Springer-Verlag.

MAREK, V. AND REMMEL, J. 2008. On the continuity of Gelfond-Lifschitz operator and other applications of proof-theory in ASP. See Garcia de la Banda and Pontelli [2008], 223–237.

MAREK, V. AND TRUSZCZYŃSKI, M. 1991. Autoepistemic logic. *Journal of the ACM 38,* 3, 588–619.

MARQUES-SILVA, J. AND SAKALLAH, K. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers 48,* 5, 506–521.

MOSKEWICZ, M., MADIGAN, C., ZHAO, Y., ZHANG, L., AND MALIK, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Thirty-eighth Conference on Design Automation (DAC'01)*. ACM Press, 530–535.

NIEMELÄ, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3-4, 241–273.

NIEMELÄ, I. 2008. Stable models and difference logic. *Annals of Mathematics and Artificial Intelligence 53,* 1-4, 313–329.

NIEUWENHUIS, R. AND OLIVERAS, A. 2005. DPLL(T) with exhaustive theory propagation and its application to difference logic. In *Proceedings of the Seventeenth International Conference on Computer Aided Verification (CAV'05)*, K. Etessami and S. Rajamani, Eds. Lecture Notes in Computer Science, vol. 3576. Springer-Verlag, 321–334.

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.

OLIVETTI, N. 1999. Tableaux for nonmonotonic logics. See D'Agostino et al. [1999], 469–528.

PEARCE, D. 1996. A new logical characterisation of stable models and answer sets. In *Proceedings of the Sixth Workshop on Non-Monotonic Extensions of Logic Programming (NMELP'96)*, J. Dix, L. Pereira, and T. Przymusinski, Eds. Lecture Notes in Computer Science, vol. 1216. Springer-Verlag, 57–70.

PEARCE, D., DE GUZMÁN, I., AND VALVERDE, A. 2000. A tableau calculus for equilibrium entailment. In *Proceedings of the Ninth International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'00)*, R. Dyckhoff, Ed. Lecture Notes in Computer Science, vol. 1847. Springer-Verlag, 352–367.

PEARCE, D. AND VALVERDE, A. 2005. A first order nonmonotonic extension of constructive logic. *Studia Logica 30,* 2-3, 321–346.

PEARCE, D. AND VALVERDE, A. 2006. Quantified equilibrium logic and the first order logic of here-and-there. Technical Report MA-06-02, University of Málaga.

PIPATSRISAWAT, K. AND DARWICHE, A. 2011. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence 175,* 2, 512–525.

PURDOM, P. 1970. A transitive closure algorithm. *BIT Numerical Mathematics 10*, 76–94.

ROBINSON, A. AND VORONKOV, A., Eds. 2001. *Handbook of Automated Reasoning*. Elsevier and MIT Press.

SIMONS, P. 2000. Extending and Implementing the Stable Model Semantics. Ph.D. thesis, Helsinki University of Technology.

SIMONS, P., NIEMELÄ, I., AND SOININEN, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138,* 1-2, 181–234.

SYRJÄNEN, T. Lparse 1.0 user's manual. `http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz`.

TRUSZCZYŃSKI, M. 2007. Comments on modeling languages for answer-set programming. In *Proceedings of the First Workshop on Software Engineering for Answer Set Programming (SEA'07)*, M. De Vos and T. Schaub, Eds. Number CSBU-2007-05 in Technical Report Series. University of Bath, Department of Computer Science, 3–11.

VAN GELDER, A., ROSS, K., AND SCHLIPF, J. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38,* 3, 620–650.

VELOSO, M., Ed. 2007. *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*. AAAI Press/MIT Press.

WARD, J. AND SCHLIPF, J. 2004. Answer set programming with clause learning. In *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, V. Lifschitz and I. Niemelä, Eds. Lecture Notes in Artificial Intelligence, vol. 2923. Springer-Verlag, 302–313.

ZHANG, L., MADIGAN, C., MOSKEWICZ, M., AND MALIK, S. 2001. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'01)*. 279–285.

Received July 2010; accepted August 2011