An Entailment Procedure for Kleene Answer Set Programs

Patrick Doherty[1] and Andrzej Szałas[1,2(✉)]

[1] Department of Computer and Information Science,
Linköping University, SE-581 83 Linköping, Sweden
{patrick.doherty,andrzej.szalas}@liu.se
[2] Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland
andrzej.szalas@mimuw.edu.pl

Abstract. Classical Answer Set Programming is a widely known knowledge representation framework based on the logic programming paradigm that has been extensively studied in the past decades. Semantic theories for classical answer sets are implicitly three-valued in nature, yet with few exceptions, computing classical answer sets is based on translations into classical logic and the use of SAT solving techniques. In this paper, we introduce a variation of Kleene three-valued logic with strong connectives, R_3, and then provide a sound and complete proof procedure for R_3 based on the use of signed tableaux. We then define a restriction on the syntax of R_3 to characterize Kleene ASPs. Strongly-supported models, which are a subset of R_3 models are then defined to characterize the semantics of Kleene ASPs. A filtering technique on tableaux for R_3 is then introduced which provides a sound and complete tableau-based proof technique for Kleene ASPs. We then show a translation and semantic correspondence between Classical ASPs and Kleene ASPs, where answer sets for normal classical ASPs are equivalent to strongly-supported models. This implies that the proof technique introduced can be used for classical normal ASPs as well as Kleene ASPs. The relation between non-normal classical and Kleene ASPs is also considered.

1 Introduction

Classical Answer Set Programs (ASP) [1,3,15–17,22] belong to the family of rule-based logic programming languages. The semantic framework for classical ASPs is based on the use of stable model semantics. There are two characteristics intrinsically associated with the construction of stable models (answer sets) for answer set programs. Any member of an answer set is supported through facts and chains of rules and those members are in the answer set only if generated minimally in such a manner. These two characteristics, supportedness and minimality, provide the essence of answer sets. Classical ASPs allows two kinds of

This work is partially supported by the Swedish Research Council (VR) Linnaeus Center CADICS, the ELLIIT network organization for Information and Communication Technology, the Swedish Foundation for Strategic Research (CUAS, SymbiCloud Project), and Vinnova NFFP6 Project 2013-01206.

C. Sombattheera et al. (Eds.): MIWAI 2016, LNAI 10053, pp. 24–37, 2016.
DOI: 10.1007/978-3-319-49397-8_3

negation, classical or strong negation and default or weak negation. Additionally, answer sets are implicitly partial and that partiality provides epistemic overtones to the interpretation of disjunctive rules and default negation.

There has been much work in providing numerous definitions [24] of what an answer set is and then relating particular definitions to other formalisms such as Circumscription [27] or Default Logic [32]. This body of research has provided much insight into the nature of answer sets. Additionally, the relationship with other formalisms has provided a basis for algorithms that generate answer sets for classical ASPs in addition to inference procedures (for implementations see, e.g., CLASP [13], ASSAT [25], CMODELS [23], SMODELS [36], DLV [21]). The majority of these approaches often use syntactic translations and encodings of classical ASPs into a classical two-valued framework using standard model theory. The benefit is that one can appeal to the broad body of techniques from classical logic.

In this paper, we are interested in Kleene Answer Set programs (Kleene ASPs) introduced in [7]. Our approach is based on an extension of the three-valued logic with strong connectives, K_3, of Kleene [19], extended with an external negation connective, $\sim$, characterizing default negation and an implication connective, $\rightarrow_s$, suitable for Kleene ASPs. This logic, which we call R_3, has been considered in [7]. Kleene ASPs are directly represented as a fragment of the R_3 language. The semantics of Kleene ASPs is based on the use of strongly supported models which turn out to be a subset of the R_3 models. It was stated previously that the intuitions behind classical ASP semantics are based on supportedness and minimality. One of the purposes in introducing Kleene ASPs and strongly supported models is to provide a weaker interpretation of non-normal ASPs where the minimality assumption on disjunction is relaxed. Separation of supportedness and minimality technically turns out to have complexity advantages when dealing with non-normal Kleene ASPs. As a derivative of this work, it can be shown that there is an equivalence between the strongly supported models of a normal Kleene ASP and the answer sets of a normal classical ASP. Consequently, these proof techniques may also be of wider interest.

The following are the main results in this paper:[1]

- We present a semantic tableaux procedure for R_3 based on the use of signed formulas that is both sound and complete for R_3. It can be used both for model generation and entailment.
- The general tableaux procedure for R_3 is only sound for Kleene ASPs. For completeness, the following method is proposed:

  A filtering procedure on tableaux branches is introduced based on the definition of strong supportedness. For both normal and non-normal Kleene ASPs, the filtering procedure is shown to be sound and complete. For normal ASP^{K}s it is tractable and for non-normal ones it is in Π_1^P.

[1] For clarity, we restrict the results to the propositional case, but they are easily generalized to the first-order case with certain restrictions.

– Due to the equivalence between strongly supported models for a normal Kleene ASP and answer sets for its corresponding normal classical ASP, the proof techniques proposed here can be used directly for normal classical ASPs. Since classical answer sets are strongly supported, the proposed proof techniques remain sound for disjunctive classical ASPs.

Structure of the Paper. The paper is structured as follows. In Sect. 2 the augmented Kleene three-valued logic, R_3, is specified. Section 3 presents the general tableaux procedure for R_3. Section 4 is devoted to Kleene ASPs and its relation to classical ASPs. It is assumed the reader is familiar with classical ASP definitions (see, e.g., [24]). In Sect. 5, a sound and complete entailment procedure for Kleene ASPs, using semantic tableaux and filtering, is presented. Section 6 concludes with related work and conclusions.

2 Three-Valued Logic R_3

The basis for our approach begins with the three-valued logic with strong connectives, K_3, of Kleene [19], extended with an external negation connective, $\sim$, characterizing default negation.[2] We assume that constants T (*true*), F (*false*), U (*unknown*) are part of the language. The implication connective, $\rightarrow_s$, is defined using, $\sim$, as:[3]

$$A \rightarrow_s B \ = \sim A \vee \neg \sim B.$$

We denote this logic by R_3.

Truth tables for connectives of R_3 are shown in Table 1, where $\neg, \vee, \wedge$ are the strong connectives of K_3, $\sim$ is the weak or external negation connective and $\rightarrow_s$ is a newly defined implication for ASP rules.

Table 1. Truth tables for R_3 connectives.

	$\neg$	$\sim$
T	F	F
F	T	T
U	U	T

$\vee$	F	U	T
F	F	U	T
U	U	U	T
T	T	T	T

$\wedge$	F	U	T
F	F	F	F
U	F	U	U
T	F	U	T

$\rightarrow_s$	F	U	T
F	T	T	T
U	T	T	T
T	F	F	T

Let $\mathcal{P}$ be the set of propositional variables. By a *valuation of propositional variables* we understand a mapping $\mathcal{P} \longrightarrow \{T, U, F\}$. If A is an R_3 formula and v is a valuation then the *truth value* of a formula, denoted by $v(A)$, is defined inductively extending v by using the semantics of the connectives provided in Table 1.

By a *positive literal* (or an *atom*) we mean any propositional variable of $\mathcal{P}$. A *negative literal* is an expression of the form $\neg r$, where $r \in \mathcal{P}$. A *(classical)*

[2] Strong negation, conjunction and disjunction have also been used in [26].
[3] The implication connective, $\rightarrow_s$, is in fact equivalent to that used in [35].

literal is a positive or a negative literal. For a literal $\ell = \neg p$, by $\neg\ell$ we understand p. A set of literals is *consistent* when it does not contain a propositional variable together with its negation. An *interpretation* is any consistent set of literals.

Any interpretation I defines a three-valued valuation v_I by:

$$v_I(p) \stackrel{\text{def}}{=} \begin{cases} \text{T when } p \in I; \\ \text{F when } \neg p \in I; \\ \text{U otherwise.} \end{cases} \tag{1}$$

Also, given v, one can construct a corresponding interpretation I_v by setting:

$$I_v \stackrel{\text{def}}{=} \{p \mid p \in \mathcal{P} \text{ and } v(p) = \text{T}\} \cup \{\neg p \mid p \in \mathcal{P} \text{ and } v(p) = \text{F}\}. \tag{2}$$

In the rest of the paper, we will freely switch between interpretations and valuations, using (1) and (2).

An interpretation I is a *model* of a set of formulas S iff for all $A \in S$, $v_I(A) = \text{T}$, where v_I is the valuation corresponding to I.

By convention, the empty conjunction is T and the empty disjunction is F.

3 Signed Tableaux for Three-Valued Logic $\mathbf{R_3}$

We follow the signed tableaux style of [6], extended with rules for $\rightarrow_s$.[4] The generalization to multivalued logics is derived from [37].

By an *information constraint lattice* we mean the lattice $L \stackrel{\text{def}}{=} \langle \mathcal{S}; \sqcup, \sqcap \rangle$ where:

- $\mathcal{S} = \{[\text{T}], [\text{F}], [\text{U F T}], [\text{U T}], [\text{U F}], [\text{U}], [\ ]\}$ is the set of signs which also contains the element $[\ ]$, representing *contradiction*; moving upwards in the lattice should be interpreted as tightening for possible truth values for a formula;
- $\sqcup$ ($\sqcap$) are the join (meet) operation on $\mathcal{S}$ defined respectively as the least upper bound and greatest lower bound w.r.t. the ordering shown in Fig. 1.

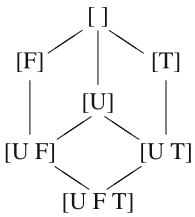

Fig. 1. Information constraint lattice.

[4] This signed approach to semantic tableaux for multi-valued logic was discovered independently, by [6,18] who generalized the technique for a larger class of multi-valued logics. The approach of [6] was used as a basis for [28].

A *signed formula* is any expression of the form $[s]A$, where $[s] \in \mathcal{S}$ in the lattice L. A valuation u *satisfies* a signed formula $[a \dots b]A$ if it satisfies the disjunction:

$$u(A) = a \text{ or } \dots \text{ or } u(A) = b, \tag{3}$$

where "or" in (3) is the classical disjunction.

A valuation u *satisfies* a set of signed formulas S iff u satisfies every formula in S. An interpretation is a *model* of S iff the corresponding valuation satisfies S.

3.1 Tableaux Construction Rules

Tableaux [37] are used to construct models for formulas. The construction of a tableau starts with a formula for which models (if any) are constructed. The tableau is then expanded according to rules provided in the subsequent subsections. The following types of rules are used, where ',' and '|' represent conjunction and disjunction, respectively:

$$\frac{[s]A}{[s']B} \qquad \frac{[s]A}{[s']B,\ [s'']C} \qquad \frac{[s]A}{[s']B \mid [s'']C}$$

The following theorem can be proved extending the corresponding proof for K_3 given in [6] by considering the new connective $\rightarrow_s$.

Theorem 1. *Tableau rules shown in Table 2 are sound and complete for R_3, i.e., a formula A is unsatisfiable iff there is a closed tableau for A.* ◁

Table 2. Tableaux rules.

Rules for conjunction - $\wedge$.

$$\frac{[\mathrm{T}]A \wedge B}{[\mathrm{T}]A,\ [\mathrm{T}]B} \qquad \frac{[\mathrm{F}]A \wedge B}{[\mathrm{F}]A \mid [\mathrm{F}]B} \qquad \frac{[\mathrm{U\,T}]A \wedge B}{[\mathrm{U\,T}]A,\ [\mathrm{U\,T}]B} \qquad \frac{[\mathrm{U\,F}]A \wedge B}{[\mathrm{U\,F}]A \mid [\mathrm{U\,F}]B}$$

Rules for disjunction - $\vee$.

$$\frac{[\mathrm{T}]A \vee B}{[\mathrm{T}]A \mid [\mathrm{T}]B} \qquad \frac{[\mathrm{F}]A \vee B}{[\mathrm{F}]A,\ [\mathrm{F}]B} \qquad \frac{[\mathrm{U\,T}]A \vee B}{[\mathrm{U\,T}]A \mid [\mathrm{U\,T}]B} \qquad \frac{[\mathrm{U\,F}]A \vee B}{[\mathrm{U\,F}]A,\ [\mathrm{U\,F}]B}$$

Rules for strong negation - $\neg$ and weak (external) negation - $\sim$.

$$\frac{[\mathrm{T}]\neg A}{[\mathrm{F}]A} \quad \frac{[\mathrm{F}]\neg A}{[\mathrm{T}]A} \quad \frac{[\mathrm{U\,T}]\neg A}{[\mathrm{U\,F}]A} \quad \frac{[\mathrm{U\,F}]\neg A}{[\mathrm{U\,T}]A} \quad \frac{[\mathrm{T}] \sim A}{[\mathrm{U\,F}]A} \quad \frac{[\mathrm{F}] \sim A}{[\mathrm{T}]A} \quad \frac{[\mathrm{U\,T}] \sim A}{[\mathrm{U\,F}]A} \quad \frac{[\mathrm{U\,F}] \sim A}{[\mathrm{T}]A}$$

Rules for implication - $\rightarrow_s$.

$$\frac{[\mathrm{T}]A \rightarrow_s}{[\mathrm{U\,F}]A} \quad \frac{\rightarrow_s [\mathrm{T}]B}{[\mathrm{T}]B} \quad \frac{[\mathrm{T}]A \rightarrow_s B}{[\mathrm{U\,F}]A \mid [\mathrm{T}]B} \quad \frac{[\mathrm{F}]A \rightarrow_s B}{[\mathrm{T}]A,\ [\mathrm{U\,F}]B} \quad \frac{[\mathrm{U\,T}]A \rightarrow_s B}{[\mathrm{U\,F}]A \mid [\mathrm{T}]B} \quad \frac{[\mathrm{U\,F}]A \rightarrow_s B}{[\mathrm{T}]A,\ [\mathrm{U\,F}]B}$$

3.2 Constructing Models Using Tableaux

A path in a tableau is *completed* if no tableau rule can extend the path. A tableau is *completed* when all its paths are completed.

Let $[s]A$ and $[s']A$ be signed formulas. A path in a tableau is *closed* if both $[s]A$ and $[s']A$ are in the path and $[s] \sqcup [s'] = [\,]$. A path is *open* when it is not closed.

A tableau $\mathcal{T}$ is *closed* if all its branches are closed. A tableau is *open* if it is not closed.

Given a tableau $\mathcal{T}$ for a formula A, to extract u satisfying A we look for an open branch. If such a branch does not exist then there is no model for A. Otherwise, for each propositional variable p appearing in the branch let:

$$\Sigma(p) \stackrel{\text{def}}{=} \{s \mid [s]p \text{ occurs in the branch}\} \text{ and let } \sigma(p) \stackrel{\text{def}}{=} \bigsqcup_{s \in \Sigma(p)} [s].$$

The satisfying valuations (models) are then defined by assigning, to each p occurring in the branch, a value from $\sigma(p)$. If for a propositional variable p, occurring in A, no formula of the form $[s]p$ occurs on a given branch then p is unconstrained and can be assigned any truth value. Note that, due to the form of rules, only signs [F], [T], [U F] and [U T] can appear in tableaux. If a proposition p does not occur in a branch, we assume that implicitly [U F T]p occurs in that branch.

Example 1. Consider the following tableau:

$$\dfrac{\dfrac{\dfrac{[\mathrm{T}]((p \vee \neg q) \wedge \sim q)}{[\mathrm{T}](p \vee \neg q),\ [\mathrm{T}](\sim q)}}{[\mathrm{U\,F}]q}}{[\mathrm{T}]p \quad | \quad \dfrac{[\mathrm{T}]\neg q}{[\mathrm{F}]q}}$$

The first branch contains [U F]q and [T]p, so $\sigma(q) = [\mathrm{U\,F}]$ and $\sigma(p) = [\mathrm{T}]$. The branch then encodes two models:

$$v_1(q) = \mathrm{U}, v_1(p) = \mathrm{T}, \text{ corresponding to } \{p\},$$
$$v_2(q) = \mathrm{F}, v_2(p) = \mathrm{T}, \text{ corresponding to } \{\neg q, p\}.$$

The second branch contains [U F]q and [F]q, so $\sigma(q) = \mathrm{F}$. Since p remains unconstrained, the branch encodes models:

$$u_1(q) = \mathrm{F}, u_1(p) = \mathrm{T}, \text{ corresponding to } \{\neg q, p\},$$
$$u_2(q) = \mathrm{F}, u_2(p) = \mathrm{F}, \text{ corresponding to } \{\neg q, \neg p\},$$
$$u_3(q) = \mathrm{F}, u_3(p) = \mathrm{U}, \text{ corresponding to } \{\neg q\}.$$

Of course, v_1, v_2, u_1, u_2, u_3 are all models for the starting formula $(p \vee \neg q) \wedge \sim q$.

$\triangleleft$

We have the following theorem which can be proved by extending the proof of the analogous theorem for K_3 given in [6].

Theorem 2. *Given an R_3 formula A, every interpretation satisfying A can be extracted from a completed tableau for $[T]A$.* ◁

4 (Kleene) Answer Set Programs

The syntax for Kleene Answer Set Programs, ASP^K, is identical for that of classical ASP programs. The semantics for Kleene ASP^K programs is based on the use of the augmented three-valued Kleene logic R_3 and strongly-supported models presented in [7]. The semantics for classical ASP programs is based on stable model semantics [24]. Correspondences between classical answer sets and strongly supported models will be considered later in this section. For the sake of clarity we consider propositional programs only.

By an ASP^K rule we understand an expression ϱ of the form:

$$\ell_1 \vee \ldots \vee \ell_k \leftarrow \ell_{k+1}, \ldots, \ell_m, not\ \ell_{m+1}, \ldots, not\ \ell_n, \tag{4}$$

where $n \geq m \geq k \geq 0$ and $\ell_1, \ldots, \ell_k, \ell_{k+1}, \ldots, \ell_m, \ell_{m+1}, \ldots \ell_n$ are (positive or negative) literals. The expression at the lefthand side of '$\leftarrow$' in (4) is called the *head* and the righthand side of '$\leftarrow$' is called the *body* of the rule. The rule is called *disjunctive* if $k > 1$.

An ASP^K *program* Π is a finite set of rules. A program is *normal* if each of its rules has at most one literal in its head. If a program contains a disjunctive rule, we call it *disjunctive* or *non-normal*.

An interpretation I *satisfies* a rule ϱ of the form (4), denoted by $I \models \varrho$, if whenever $\ell_{k+1}, \ldots, \ell_m \in I$ and $\ell_{m+1}, \ldots, \ell_n \notin I$, we have $\ell_i \in I$ for some $1 \leq i \leq k$. An interpretation I *satisfies* an ASP^K program Π, denoted by $I \models \Pi$, if for all rules $\varrho \in \Pi$, $I \models \varrho$.

We say that an Kleene ASP program Π *entails an* R_3 *formula* A, and denote it by $\Pi \models A$, provided that A is true in every strongly supported model of Π.

The concept of strong supportedness [7] builds on the principle of constructing models through chains of rules grounded in facts. When evaluating the body of a rule to determine whether it is applicable, literals outside the scope of *not* must be evaluated against the set of literals for which support has already been found, represented as an interpretation I. However, literals inside the scope of *not* must be evaluated against a strongly supported model candidate J, similarly to how *not* ℓ is evaluated against an answer set candidate S when a reduct Π^S is computed classically. We therefore evaluate formulas w.r.t. two interpretations. Given interpretations I and J, the *value of a formula A w.r.t.* (I, J), denoted by $(I, J)(A)$, is defined as follows:

$$(I, J)(A) \stackrel{\text{def}}{=} \begin{cases} \text{T when } I \models reduct^J(A); \\ \text{F when } I \models reduct^J(\neg A); \\ \text{U otherwise.} \end{cases} \tag{5}$$

where $reduct^J(A)$ (respectively, $reduct^J(\neg A)$) is a formula obtained from A ($\neg A$) by substituting subformulas of the form $not\ \ell$ by their truth values evaluated in J.

Now we are ready to define strong supportedness. An interpretation N is a *strongly supported model* of an ASP^K program Π provided that N satisfies Π and there exists a sequence of interpretations $I_0 \subseteq I_1 \subseteq \ldots \subseteq I_n$ where $n \geq 0$ such that $I_0 = \emptyset$, $N = I_n$, and:

1. for every $1 \leq i \leq n$ and every rule $\ell_1 \vee \ldots \vee \ell_k \leftarrow B$ of Π, if $\big(I_{i-1}, N\big)(B) = \mathrm{T}$ then a nonempty subset of $\{\ell_1, \ldots, \ell_k\}$ is included in I_i;
2. for $i = 1, \ldots, n$, I_i can only contain literals obtained by applying point 1.

A *Kleene Answer Set* for an ASP^K program is a strongly supported model. The terminology will be used interchangeably.

The following correspondences between classical answer sets with stable model semantics (for definitions see, e.g., [24]) and Kleene answer sets with strongly-supported model semantics relate the two semantics.

In [7] the following theorem is proved (see [7, point 2 of Theorem 1]).

Theorem 3. *For any normal ASP^K program Π, I is a classical answer set of Π iff I is a strongly supported model of Π.* ◁

The following theorem clarifies the role of strong supportedness in the context of disjunctive programs.

Theorem 4. *For any ASP^K program Π, if I is a classical answer set of Π then I is a strongly supported model of Π.* ◁

Theorem 3 allows us to use the filter-based tableau technique introduced in Sect. 5 below, not only for Kleene ASPs, but for classical normal ASPs, too. Theorem 4 shows that filtering remains sound for disjunctive classical ASPs.[5]

5 Filtering Technique for Kleene Answer Set Programs

To construct tableaux for Kleene ASP entailment we first translate Kleene ASPs into formulas of R_3 using the translation Tr defined as follows:

$$\begin{aligned} &Tr(\neg A) \stackrel{\text{def}}{=} \neg A,\ Tr(not\ A) \stackrel{\text{def}}{=} \sim A, \\ &Tr(A \circ B) \stackrel{\text{def}}{=} Tr(A) \circ Tr(B), \text{for } \circ \in \{\vee, \wedge\}, \\ &Tr(A \leftarrow B) \stackrel{\text{def}}{=} Tr(B) \rightarrow_s Tr(A). \end{aligned} \tag{6}$$

Rules are then translated as follows:

$$\begin{aligned} &Tr(\ell_0 \vee \ldots \vee \ell_i \leftarrow \ell_{i+1}, \ldots, \ell_j, not\ \ell_{j+1}, \ldots, not\ \ell_m) = \\ &\quad (\ell_{i+1} \wedge \ldots \wedge \ell_j \wedge \sim \ell_{j+1} \wedge \ldots \wedge \sim \ell_m) \rightarrow_s (\ell_0 \vee \ldots \vee \ell_i). \end{aligned} \tag{7}$$

[5] In understanding the theorems, recall that the syntax for ASP^K programs and classical ASP programs is identical.

For a Kleene ASP program Π, $Tr(\Pi) \stackrel{\text{def}}{=} \bigwedge_{r\in\Pi} Tr(r)$. By a *model of a Kleene ASP program* Π we understand any model of $Tr(\Pi)$.

Note that every Kleene answer set for a program Π is an R_3 model for $Tr(\Pi)$. Therefore, we have the following theorem.

Theorem 5. *Let Π be a Kleene ASP program, A be an R_3 formula, and $\mathcal{T}$ be a tableau for:*

$$[T]\big(Tr(\Pi) \wedge {\sim} A\big). \tag{8}$$

Then, if $\mathcal{T}$ is closed then $\Pi \models A$. ◁

That is, the tableaux procedure provided in Sect. 3.1 is sound for Kleene ASP entailment.

Example 2. Let Π consists of rules:

$$q \leftarrow p.$$
$$q \leftarrow not\, p.$$

To show that $\Pi \models q$ we construct the following tableau:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{[\mathrm{T}]\big((p \rightarrow_s q) \wedge ({\sim}p \rightarrow_s q) \wedge {\sim}q\big)}{[\mathrm{T}]\big((p \rightarrow_s q) \wedge ({\sim}p \rightarrow_s q)\big),\ [\mathrm{T}]({\sim}q)}}{[\mathrm{T}](p \rightarrow_s q),\ [\mathrm{T}]({\sim}p \rightarrow_s q),\ [\mathrm{T}]({\sim}q)}}{[\mathrm{U\,F}]q}}{\cfrac{[\mathrm{U\,F}]p}{\cfrac{[\mathrm{U\,F}]\,{\sim}p \ \mid\ [\mathrm{T}]q}{[\mathrm{T}]p}} \quad \mid \quad [\mathrm{T}]q}$$

On the first branch $\sigma(p) = [\ ]$ and on the other two branches $\sigma(q) = [\ ]$. Thus the above tableau is closed. ◁

For completeness of ASP^K, a filtering technique is required to filter out non-strongly supported models associated with open branches. To decide whether $\Pi \models A$ we first construct a tableau $\mathcal{T}$ for signed formula (8). Then,

1. if $\mathcal{T}$ is closed then $\Pi \models A$; otherwise
2. *filtering*: eliminate every open branch of $\mathcal{T}$ encoding only non-strongly supported models of Π. If all open branches of $\mathcal{T}$ are eliminated then $\Pi \models A$, otherwise $\Pi \not\models A$.

The filtering described in point 2. Above is sound and complete for ASP^K, as stated in the following theorem.

Theorem 6. *Let Π be a Kleene ASP program, A be an R_3 formula, and $\mathcal{T}$ be a tableau for $[T]\big(Tr(\Pi) \wedge {\sim} A\big)$. Then, $\Pi \models A$ iff filtering eliminates every open branch of $\mathcal{T}$.* ◁

Example 3. Let program Π consist of a single rule: $q \leftarrow not\, p$. To show that $\Pi \models q$, we construct the following tableau:

$$\dfrac{\dfrac{\dfrac{[\mathrm{T}]\big((\sim p \rightarrow_s q) \wedge \sim q\big)}{[\mathrm{T}](\sim p \rightarrow_s q),\ [\mathrm{T}](\sim q)}}{[\mathrm{U\,F}]q}}{\dfrac{[\mathrm{U\,F}] \sim p}{[\mathrm{T}]p} \quad \mid \quad [\mathrm{T}]q}$$

The first branch is open and the second branch is closed. Therefore we have to check whether the first branch represents a strongly supported model for Π. Since the branch contains $[\mathrm{U\,F}]q$ and $[\mathrm{T}]p$, the candidates for a strongly supported model for Π are $\{p\}$ and $\{\neg q, p\}$.[6] Of course, $\{p\}$ and $\{\neg q, p\}$ are not strongly supported, so there are no open branches representing strongly supported models for Π. Therefore, Π indeed entails q. ◁

For non-normal programs an open branch may encode more than one strongly supported model, consequently the technique shown in Example 3 does not apply since one uses an assumption of equivalence between minimality and strong supportedness that only applies to normal programs. In this case one is required to filter all potential interpretations, including non-minimal ones, associated with an open branch.

To verify strong supportedness one can use Algorithm 1 provided in [7, p. 137]. Given a program Π and an interpretation I, this algorithm checks whether I is a strongly supported model of Π in deterministic polynomial time w.r.t. Π and I. Recall that due to the form of tableau rules shown in Table 2, only signs [F], [T], [U F] and [U T] (and implicitly [U F T]) can appear in tableaux. Therefore, to check whether a given open branch of a tableau encodes a strongly supported model, one can extract the candidate valuation v in such a way that whenever for a given proposition p, U is in $\sigma(p)$, we set $v(p) \stackrel{\text{def}}{=} \mathrm{U}$, otherwise $v(p)$ is the (uniquely determined) truth value from $\sigma(p)$. That way, for normal programs, v represents a minimal interpretation uniquely determined by this branch. By Theorem 3, for normal programs strong supportedness is equivalent to minimality, so we have the following theorem.

Theorem 7. *For any normal ASP^K program Π checking whether a given tableau branch for $[T]\Pi$ encodes a strongly supported model is tractable.* ◁

For non-normal programs, checking whether an open branch exists such that there is a valuation encoded by the branch defining a strongly supported model, is obviously in Σ_1^P. Therefore, checking whether a branch can be closed via filtering (does not encode a strongly supported model), is in Π_1^P (i.e., co-NP) as stated in the following theorem.

[6] Note that both $\{p\}$ and $\{\neg q, p\}$ are R_3 models for the considered formula, so the fact that q is entailed by Π cannot be proved using only rules provided in Sect. 3.1.

Theorem 8. *For any non-normal ASP^K program Π checking, in general, whether a given tableau branch for $[T]\Pi$ encodes a strongly supported model, is in Π_1^P.* ◁

Note that Theorem 8, ensures that our entailment procedure is in Π_1^P.

Remark 1. The technique of filtering can be applied to classical answer sets, too. First, one can filter out non-strongly supported models. This can be done "locally" node by node. Normal classical ASP programs are equivalent to normal ASP^K programs so for this case our procedure remains sound and complete. For non-normal classical ASP programs, the minimality requirement results in a higher complexity of reasoning [8,9], [7, Thm. 2, p. 137] (assuming that the polynomial hierarchy does not collapse). In this case, checking for non-minimality rather than for non-supportedness is required. It calls for pairwise comparisons with all models, perhaps encoded by other nodes in the constructed tableaux. Therefore, rather than checking for minimality, we could achieve completeness for non-normal classical ASPs by suitably adding a generalization of Clark's completion and loop formulas provided in [25] to the original ASP program. ◁

6 Related Work and Conclusions

There is a rich history of explicit use of partial interpretations and multi-valued logics as a basis for semantic theories for logic programs. Some related and additional representative examples are [5,11,12,20,30,31,34,38]. Supportedness is analyzed in many papers, starting from [10]. One of most recent generalizations, via grounded fixpoints, is investigated in [2]. However, grounded fixpoints are minimal (see [2, Proposition 3.8]) while strongly supported models do not have to be minimal.

In [33] a possible model semantics for disjunctive programs is proposed. It is formulated with the use of split programs and there can be exponentially many of them comparing to the original program. Similar semantics was independently proposed in [4] under the name of the possible world semantics. In comparison to [33], ASP^K programs allow for strong negation and a three-valued model-theoretic semantics is provided. The presence of both default and strong negation in ASP^K provides a tool to close the world locally in a contextual manner, more flexible than possible model negation proposed in [33]. Though defined independently and using different foundations, both semantics appear compatible on positive programs, so the results of the current paper apply to possible model semantics of [33], too.

Paper [14] defines a tableaux framework for classical ASP, using two (explicit) truth values T, F. They require a *cut* rule, whereas we do not. Loop formulas are explicit in some rules and supportedness is encoded as an additional set of inference rules. Our approach is very much in the spirit of Smullyan [37] and does not require special inference rules, although loop and completion formulas would have to be added to an ASP if one wanted to deal with classical non-normal ASPs.

In [29,30], the logic of here-and-there (HT) is used to define a direct declarative semantics for classical ASPs, although HT has greater generality and wider application. HT can be defined by means of a five-valued logic, N_5, defined over two worlds: h (here) and t (there), where the set of literals associated with h is included in the set of literals associated with t. N_5 uses truth values $\{-2, -1, 0, 1, 2\}$, where the values $-1, 1$ characterize literals associated with h and not associated with t. On the other hand, for classical ASP models it is assumed that these sets are equal, so $-1, 1$ become redundant. Therefore, in the context of classical ASPs one actually does not have to use full N_5 as it reduces to the three-valued logic R_3, with $-2, 0, 2$ of N_5 corresponding to F, U, T of R_3, respectively. Consequently, tableaux techniques used in [29] for N_5 could then be simplified when focus is on ASPs.

Kleene Answer set programs, ASP^K, and connectives used in R_3 have been proposed in [7]. The current paper introduces a sound and complete tableaux-based proof procedure for them. A filtering technique is introduced which, when added to the R_3 tableaux based proof procedure, provides a sound and complete proof procedure for Kleene ASPs. As a derivative result, it is shown that the proof procedure is also sound and complete for classical normal ASPs and remains sound for disjunctive classical ASPs.

References

1. Baral, C.: Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)
2. Bogaerts, B., Vennekens, J., Denecker, M.: Grounded fixpoints and their applications in knowledge representation. Artif. Intell. **224**, 51–71 (2015)
3. Brewka, G., Niemelä, I., Truszczynski, M.: Answer set optimization. In: Gottlob, G., Walsh, T. (eds.) Proceedings of the 18th IJCAI, pp. 867–872. Morgan Kaufmann (2003)
4. Chan, P.: A possible world semantics for disjunctive databases. IEEE Trans. Knowl. Data Eng. **5**(2), 282–292 (1993)
5. Denecker, M., Marek, V., Truszczynski, M.: Stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In: Minker, J. (ed.) Logic-Based Artificial Intelligence, pp. 127–144. Kluwer Academic Publishers, Dordrecht (2000)
6. Doherty, P.: A constraint-based approach to proof procedures for multi-valued logics. In: Proceedings of the 1st World Conference Fundamentals of AI (WOCFAI), pp. 165–178. Springer (1991)
7. Doherty, P., Szałas, A.: Stability, supportedness, minimality and Kleene Answer Set Programs. In: Eiter, T., Strass, H., Truszczyński, M., Woltran, S. (eds.) Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation. LNCS (LNAI), vol. 9060, pp. 125–140. Springer, Heidelberg (2015). doi:10.1007/978-3-319-14726-0_9
8. Eiter, T., Faber, W., Fink, M., Woltran, S.: Complexity results for answer set programming with bounded predicate arities and implications. Ann. Math. Artif. Intell. **51**(2–4), 123–165 (2007)

9. Eiter, T., Gottlob, G.: Complexity results for disjunctive logic programming and application to nonmonotonic logics. In: Miller, D. (ed.) Proceedings of the Logic Programming, pp. 266–278 (1993)
10. Fages, F.: Consistency of Clark's completion and existence of stable models. Methods Logic Comput. Sci. **1**, 51–60 (1994)
11. Fitting, M.: A Kripke-Kleene semantics for logic programs. J. Logic Program. **2**(4), 295–312 (1985)
12. Fitting, M.: The family of stable models. J. Logic Program. **17**(2–4), 197–225 (1993)
13. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: from theory to practice. Artif. Intell. **187**, 52–89 (2012)
14. Gebser, M., Schaub, T.: Tableau calculi for logic programs under answer set semantics. ACM Trans. Comput. Log. **14**(2), 15:1–15:40 (2013)
15. Gelfond, M., Kahl, Y.: Knowledge Representation, Reasoning, and the Design of Intelligent Agents - The Answer-Set Programming Approach. Cambridge University Press, Cambridge (2014)
16. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R., Bowen, K. (eds.) Proceedings of the International Logic Programming, pp. 1070–1080. MIT Press (1988)
17. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**(3/4), 365–386 (1991)
18. Hähnle, R.: Uniform notation of tableau rules for multiple-valued logics. In: ISMVL, pp. 238–245 (1991)
19. Kleene, S.C.: On a notation for ordinal numbers. Symbolic Logic **3**, 150–155 (1938)
20. Kunen, K.: Negation in logic programming. J. Log. Program. **4**(4), 289–308 (1987)
21. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Logic **7**(3), 499–562 (2006)
22. Leone, N., Rullo, P., Scarcello, F.: Disjunctive stable models: unfounded sets, fixpoint semantics, and computation. Inf. Comput. **135**(2), 69–112 (1997)
23. Lierler, Y.: Relating constraint answer set programming languages and algorithms. Artif. Intell. **207**, 1–22 (2014)
24. Lifschitz, V.: Thirteen definitions of a stable model. In: Blass, A., Dershowitz, N., Reisig, W. (eds.) Fields of Logic and Computation. LNCS, vol. 6300, pp. 488–503. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15025-8_24
25. Lin, F., Zhao, Y.: ASSAT: computing answer sets of a logic program by SAT solvers. Artif. Intell. **157**(1–2), 115–137 (2004)
26. Łukasiewicz, J.: O logice trójwartościowej (in Polish). Ruch filozoficzny **5**, 170–171 (1920). English translation: On three-valued logic. In: Borkowski, L. (ed.) Selected works by Jan Łukasiewicz, pp. 87–88. North-Holland, Amsterdam (1970)
27. McCarthy, J.: Circumscription - a form of non-monotonic reasoning. Artif. Intell. **13**(1–2), 27–39 (1980)
28. Murray, N., Rosenthal, E.: Adapting classical inference techniques to multiple-valued logics using signed formulas. Fundam. Inform. **21**(3), 237–253 (1994)
29. Pearce, D.: Equilibrium logic. Ann. Math. AI **47**(1–2), 3–41 (2006)
30. Pearce, D., Guzmán, I.P., Valverde, A.: Computing equilibrium models using signed formulas. In: Lloyd, J., et al. (eds.) CL 2000. LNCS (LNAI), vol. 1861, pp. 688–702. Springer, Heidelberg (2000). doi:10.1007/3-540-44957-4_46
31. Przymusinski, T.: Stable semantics for disjunctive programs. New Gener. Comput. **9**(3/4), 401–424 (1991)

32. Reiter, R.: A logic for default reasoning. Artif. Intell. **13**(1–2), 81–132 (1980)
33. Sakama, C., Inoue, K.: An alternative approach to the semantics of disjunctive logic programs and deductive databases. J. Autom. Reasoning **13**(1), 145–172 (1994)
34. Seipel, D., Minker, J., Ruiz, C.: A characterization of the partial stable models for disjunctive databases. In: Małuszyński, J. (ed.) Logic Programming Symposium, pp. 245–259 (1997)
35. Shepherdson, J.C.: A sound and complete semantics for a version of negation as failure. Theor. Comput. Sci. **65**(3), 343–371 (1989)
36. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artif. Intell. **138**(1–2), 181–234 (2002)
37. Smullyan, R.: First-Order Logic. Dover Publications, Mineola (1968)
38. Stamate, D.: Assumption based multi-valued semantics for extended logic programs. In: 36th IEEE International Symposium on ISMVL, p. 10. IEEE Computer Society (2006)