Stability, Supportedness, Minimality and Kleene Answer Set Programs *

Patrick Doherty[1] and Andrzej Szałas[1,2]

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

Abstract. 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. The semantic framework for Answer Set Programs is based on the use of stable model semantics. There are two characteristics intrinsically associated with the construction of stable models 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 stable models. Additionally, answer sets are implicitly partial and that partiality provides epistemic overtones to the interpretation of disjunctive rules and default negation. This paper is intended to shed light on these characteristics by defining a semantic framework for answer set programming based on an extended first-order Kleene logic with weak and strong negation. Additionally, a definition of strongly supported models is introduced, separate from the minimality assumption explicit in stable models. This is used to both clarify and generate alternative semantic interpretations for answer set programs with disjunctive rules in addition to answer set programs with constraint rules. An algorithm is provided for computing supported models and comparative complexity results between strongly supported and stable model generation are provided.

1 Introduction

Answer Set Programming (ASP) [2, 4–6, 16, 17] is a knowledge representation framework based on the logic programming paradigm that uses an answer set/stable model semantics for logic programs as its basis. There are a number of extensions to the language of ASP that provide increased expressivity relative to standard Prolog with negation as failure. ASP allows two kinds of negation, classical or "strong" negation and

* 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 Project), the EU FP7 project SHERPA (grant agreement 600958), and Vinnova NFFP6 Project 2013-01206.

default or "weak" negation. Additionally, it is extended to allow disjunctive heads in rules.

A very attractive feature of ASP is the use of an open world assumption as default in its semantic theory rather than the closed world assumption present in standard Prolog and most variants of Datalog. The open world assumption arises naturally in ASP since its semantic theory is intrinsically based on partial interpretations or models. Additionally, one can syntactically encode local closed world assumptions for particular relations in an answer set program in a fine-grained manner when needed.

ASP also includes constraint rules, rules whose heads are false. Constraint rules have been shown to be quite useful as a model filtering technique in various applications of ASP. Interestingly, the ASP semantics for constraint rules (and existing implementations) makes implicit use of a technique associated with filtered circumscription where answer sets are first generated for a subset of rules (non-constraint rules) in an answer set program and then these answer sets are filtered with the remaining constraint rules. There are other ways to interpret the semantics of constraint rules that are equally intuitive and will be considered.

In [16] the following informal principles that guide such a construction by a rational reasoner are pointed out. During the construction of an answer set S for an answer set program Π,

1. S must satisfy the rules of Π in the sense that any atom in S is in the head of a rule r of Π and the chain of rules used to satisfy the atom should be grounded in facts of Π;
2. the construction of S does not include any atoms that are not forced to be in S except through the explicit use of chains of rules grounded in facts.

The first principle describes a form of chained rule *support* for any atom in S while the second principle describes a *minimality* principle for any answer set S for an answer set program Π. In fact, a supported, minimal model for Π is a stable model for Π in the technical sense.

On the surface, both supportedness and the minimality principle make intuitive sense, especially in the context of normal answer set programs (those with non-disjunctive heads) and without constraint rules. When an answer set program is extended with either rules with disjunctive heads or constraint rules, or both, there are equally intuitive semantics that provide partial models for such programs that are not necessary minimal in the sense used for stable model semantics.

In order to explore these distinctions in the context of semantic alternatives for answer set programs which allow both rules with disjunctive heads and constraint rules, the underlying formalism has to be able to make a distinction between *supportedness* and *minimality*. Additionally, one would like the underlying semantic theory to elucidate the use of partial interpretations explicitly in the logical apparatus used. This implies the use of a multi-valued logic as a semantic basis for answer set programming.

The underlying logic used as a basis for a semantic theory for answer set programs that may include both classical and default negation, disjunctive heads in rules, and constraint rules will be a well known first-order three-valued logic proposed by Kleene.

This logic uses the strong connectives for disjunction, conjunction. implication and (strong) negation and is denoted by $\mathcal{K}_3$. $\mathcal{K}_3$ is then extended with a nonmonotonic (weak) negation connective,not, in addition to a conditional connective that supports the intuitive reading of ASP rules. Rules are also generalized to include arbitrary first-order Kleene formulas in the bodies of rules.

Interestingly, this logic is sound for ASP programs with finite domains in the following sense. Let Π be an answer set program and $M_{asp}(\Pi)$ the stable models of Π using the stable model semantics. Additionally, let $M_{KL^\star}(\Pi)$ be the partial models of Π using the modified Kleene logic and $Trans()$ a straightforward translation function that takes a partial model from $\mathcal{K}_3$ and returns an ASP model consisting of positive and (classically) negative literals. Then:

$$M_{asp}(\Pi) \subseteq \{m \mid m' \in M_{KL^\star}(\Pi) \wedge m = \mathit{Trans}(m')\},$$

where:

$$Trans(m') \stackrel{\text{def}}{=} \{\ell \mid m'(\ell) = \mathrm{T}\} \cup \{\neg\ell \mid m'(\ell) = \mathrm{F}\}. \tag{1}$$

Given that this is the case, definitions will be provided that allow us to distinguish between strongly supported models for an ASP program Π and minimal, supported models for Π. For normal ASP's, supported models and stable models are equivalent. For ASPs with disjunctive rules, one can define a semantics in terms of only strongly supported models, or strongly supported, minimal models. In the latter case, equivalence is shown between the strongly supported, minimal models of an answer set program with disjunctive rules and its stable models.

An alternative semantics for ASPs is provided by simply appealing to the use of strongly supported models. The gain here is that the semantic intuitions are equally convincing, yet the complexity in constructing answer sets for ASPs with disjunctive rules is lower. An algorithm for generating strongly supported models for ASP's is also provided. Comparative complexity results are provided for stable models and supported models.

The paper concludes with a discussion of constraint rules and ASPs. Two alternative ways to generate answer sets for ASPs that include constraints are provided. In the first case, one simply generates the strongly supported, (minimal) models of the ASP using Kleene semantics and translates these into answer sets using the $Trans()$ function. In the other case, one partitions an ASP into two sets, C and NC, representing the constraint rules and other rules, respectively. One then generates the supported minimal models for NC using Kleene semantics and then filters these with the constraint rules in C leaving only those models that satisfy the constraints rules in C, too.

The latter case appears to provide the current semantics for ASPs with constraint rules and existing implementations of ASPs appear to follow this semantics. Interestingly, the former case is equally feasible and seems to make more sense in the context of ASPs that have an underlying Kleene semantics. A constraint rule is simply a rule like any other in an ASP and filtering is implicit in the model construction for the full ASP. These two approaches do not necessarily generate the same models for an ASP.

The paper is structured as follows. In Section 2 we define an extended first-order language and its modified Kleene three-valued semantics. In Section 3 we introduce ASP^K

and define its semantics using strongly supported models. We also consider ASP^K_{min} admitting only minimal strongly supported models. Section 4 is devoted to alternative semantics for constraint rules. In Section 5 we discuss minimality and its effects in the context of standard ASP stable model semantics and ASP^K strongly supported model semantics. Section 6 presents an algorithm for computing strongly supported models and comparative complexity results between standard ASP, ASP^K and ASP^K_{min}. Finally, Section 7 concludes the paper.

2 First-Order Formulas with Default Negation

Answer set rules in the formalism to be introduced will be generalized to allow arbitrary first-order formulas with default negation in their bodies. In this section, the language of first-order formulas used is introduced. Additionally, the underlying semantics for this language will be a modified first-order three-valued Kleene logic $\mathcal{K}_3$ with weak and strong negation that is also described.

Let *D* be a finite set of constants, called the *domain*. In the current paper we deal with finite domains only and assume that these domains consist of constant symbols. We further assume that V is the set of individual variables and $\mathcal{R}$ is the set of relation symbols. The number of arguments of $r \in \mathcal{R}$ is denoted by $n(r)$.

Definition 1. By a *positive literal* (or an *atom*) we mean any expression of the form $r(a_1, \ldots, a_{n(r)})$, where $r \in \mathcal{R}$ and $a_1, \ldots, a_{n(r)} \in D \cup V$. A *negative literal* is an expression of the form $\neg\ell$, where ℓ is a positive literal. A *literal* is a positive or a negative literal. A *ground literal* is a literal without variables. A set of literals is *consistent* if it contains no literal ℓ together with its negation $\neg\ell$.[3] $\lhd$

Definition 2. By *Kleene first-order formulas*, KFOL, we understand formulas of first-order logic with an additional connective '*not*', called *default negation*:

$$\begin{aligned}\langle \text{KFOL}\rangle ::= {} & \text{T} \mid \text{F} \mid \text{U} \mid \langle \mathit{Atom}\rangle \mid \neg\langle \text{KFOL}\rangle \mid \mathit{not}\ \langle \text{KFOL}\rangle \mid \\ & \langle \text{KFOL}\rangle \vee \langle \text{KFOL}\rangle \mid \langle \text{KFOL}\rangle \wedge \langle \text{KFOL}\rangle \mid \langle \text{KFOL}\rangle \Rightarrow \langle \text{KFOL}\rangle \\ & \exists\langle V\rangle\langle \text{KFOL}\rangle \mid \forall\langle V\rangle\langle \text{KFOL}\rangle \mid (\langle \text{KFOL}\rangle)\end{aligned}$$ $\lhd$

The semantics of KFOL is three-valued, with the set of truth values $\{\text{T}, \text{F}, \text{U}\}$ ordered by '$<$' defined as follows:

$$\text{F} < \text{U} < \text{T}. \tag{2}$$

For the propositional connectives, we use the semantics of Kleene's system with strong connectives [18]. We denote the logic as $\mathcal{K}_3$.

Definition 3. For $u, w \in \{\text{T}, \text{F}, \text{U}\}$ we define:

$$u \vee w \stackrel{\text{def}}{=} \max\{u, w\}, \quad u \wedge w \stackrel{\text{def}}{=} \min\{u, w\}, \tag{3}$$

[3] We always remove double strong negations using $\neg(\neg\ell) \stackrel{\text{def}}{=} \ell$.

where $\max, \min$ are maximum and minimum w.r.t. ordering (2).

Strong Kleene negation, $\neg$, is defined as:

$$\neg \mathrm{F} \stackrel{\text{def}}{=} \mathrm{T}, \quad \neg \mathrm{U} \stackrel{\text{def}}{=} \mathrm{U}, \quad \neg \mathrm{T} \stackrel{\text{def}}{=} \mathrm{F} \tag{4}$$

Implication $\Rightarrow$ is defined by:

$$u \Rightarrow w \stackrel{\text{def}}{=} \neg u \vee w. \tag{5}$$

$\triangleleft$

We then extend $\mathcal{K}_3$ logic with two additional *external* connectives for default negation and the implication connective used in rules.

Default negation will be defined as *weak* or *external* negation, '*not*':

$$not\ \mathrm{F} \stackrel{\text{def}}{=} \mathrm{T}, \quad not\ \mathrm{U} \stackrel{\text{def}}{=} \mathrm{T}, \quad not\ \mathrm{T} \stackrel{\text{def}}{=} \mathrm{F}. \tag{6}$$

Weak negation is a nonmonotonic connective with the intuitive reading *absence of truth*. Both types of negation have been used in the study of presupposition in natural language [12].

To define the semantics of rules we will also use another implication:

$$u \leftarrow w \stackrel{\text{def}}{=} \begin{cases} \mathrm{F} \text{ for } w = \mathrm{T} \text{ and } u \in \{\mathrm{F}, \mathrm{U}\}; \\ \mathrm{T} \text{ otherwise.} \end{cases} \tag{7}$$

This implication connective is also discussed in [23].

Remark 1.

- A similar logic has been considered in [22]. However, we use different implications here in addition to using two negation connectives. Also, our definition of satisfiability of rules (Definition 8) is different. 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 [8, 14, 15].
- In [21] the logic of here-and-there (HT) is used to define the semantics of ASP. 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 ASP models it is assumed that these sets are equal, so $-1, 1$ become redundant. Therefore, in the context of ASP one actually does not have to use full N_5 as it reduces to the three-valued logic of Kleene $\mathcal{K}_3$ with the additional implication (7) used in the current paper, with $-2, 0, 2$ of N_5 corresponding to F, U, T of $\mathcal{K}_3$, respectively. $\triangleleft$

Definition 4. For a given set of relation symbols $\mathcal{R}$ and a set of constants *D*, by an *interpretation* over $\mathcal{R}$ and *D* we mean any finite consistent set of ground literals (positive or negative) with relation symbols from $\mathcal{R}$ and constants from *D*. $\triangleleft$

Note that $Trans()$, defined in (1), allows us to easily switch between three-valued Kleene interpretations and interpretations in the sense of Definition 4.

Definition 5.

- Given a domain D, a *valuation of variables* (*valuation*, in short) is a mapping $v : V \longrightarrow D$.
- For $A \in$ KFOL and valuation v, by $v(A)$ we mean a formula obtained from A by replacing every free variable x in A by the constant $v(x)$. $\triangleleft$

Definition 6. For a given set of relation symbols $\mathcal{R}$, domain D, interpretation I over D, $\mathcal{R}$, and valuation v, the *value of a* KFOL *formula A w.r.t. I and v*, denoted by A^I_v, is defined as follows:

- for $t \in \{\text{T}, \text{F}, \text{U}\}$ we have $t^I_v \stackrel{\text{def}}{=} t$;
- $r(a_1, \ldots, a_{n(r)})^I_v \stackrel{\text{def}}{=} \begin{cases} \text{T} \text{ when } r(a'_1, \ldots, a'_{n(r)}) \in I, \\ \quad \text{where } a'_i = a_i \text{ for } a_i \in D \text{ and } a'_i = v(a_i) \text{ for } a_i \in V; \\ \text{F} \text{ when } \neg r(a'_1, \ldots, a'_{n(r)}) \in I, \text{ where } a'_i \text{ are as above;} \\ \text{U} \text{ otherwise;} \end{cases}$
- for $A \circ B$ and $\circ A$, where $\circ$ is a propositional connective, we use definitions of connectives (3)–(7), respectively;
- $\exists x[A(x)] \stackrel{\text{def}}{=} \max\limits_{a \in D}\{A(a)\}$, $\forall x[A(x)] \stackrel{\text{def}}{=} \min\limits_{a \in D}\{A(a)\}$, where $\max, \min$ are maximum and minimum w.r.t. ordering (2). $\triangleleft$

In the case of expressions without variables, in Definition 6 the valuation v becomes redundant, so we sometimes write A^I rather than A^I_v.

3 Kleene Answer Set Programs

Kleene answer set programs can now be defined as a set of rules where arbitrary first-order formulas are allowed in the bodies of rules.

Definition 7. A *Kleene answer set program* consists of a finite set of rules of the following form, where $\ell_1, \ldots, \ell_k$ are (positive or negative) literals and A is a Kleene first-order formula or the empty symbol:

$$\ell_1 \vee \ldots \vee \ell_k \leftarrow A. \tag{8}$$

The disjunction $\ell_1 \vee \ldots \vee \ell_k$ is called the *head* and A is called the *body* of (8). Variables occurring in the head of a rule should also occur free in the rule's body.

The empty head evaluates to F. Rules with the empty head are called *constraints*. The empty body evaluates to T. Rules with the empty body are called *facts* and are written without the symbol '$\leftarrow$'. $\triangleleft$

Remark 2. Note that in Definition 7 we require a rather weak form of safety. Safety in the context of answer set programming is usually defined by requiring that in every rule each variable appearing in the rule appears in at least one positive literal in the body of that rule (see, e.g., [1, 3]). However, this notion of safety is rather restrictive as it disallows, e.g., important rules for closing the world locally, such as:

$$\neg p(X) \leftarrow not\ p(X). \tag{9}$$

When rules allow for more complex expressions, different versions of safety are considered [7]. On the other hand, when we fix domains as finite sets of constant symbols, the problems related to non-safety disappear. Namely, if D is the domain, whenever a rule uses a variable, the rule implicitly involves the "domain checking" atom for each variable. For example, (9), in fact, expresses:

$$\neg p(X) \leftarrow d(X) \wedge not\ p(X). \tag{10}$$

The positive atom $d(X)$ in (10) expresses the fact that the value of X is in D. If there are more variables, rules semantically behave as if such "domain checks" were added for each variable appearing in the rule. ◁

Definition 8. An interpretation I is a *model* of (8) if for every valuation $v : V \longrightarrow D$,

$$\big((\ell_1 \vee \ldots \vee \ell_k) \leftarrow A\big)^I_v = \mathrm{T}. \tag{11}$$

I is a model of an ASP^K program if it is a model of every rule of the program. ◁

Definition 9. A model I of an ASP^K program Π is *minimal* if there is no model J of Π such that $J \subsetneq I$. ◁

The following definitions generalize known definitions of well-supported models for standard normal logic programs with a two-valued semantics [10, 11] to the case of ASP^K programs. This generalization is called *strongly supported* models. Our formulation concentrates on derivability rather than on the existence of a certain well-founded ordering, as in [10, 11]. The intuition is that whenever a (positive or negative) literal belongs to a strongly supported model for a program Π then there should be a finite derivation of the literal starting from facts of Π and, if needed, using rules of Π. Of course, our definition can also be given in terms of well-founded relations as in [10, 11], but the definition used here simplifies presentation and proofs.

Let us start with a definition of the value of a formula w.r.t. two interpretations: the first one for evaluating formulas outside of the scope of 'not ' and the second one for evaluating formulas of the form '$not\ C$'.

Definition 10. Given a domain D, interpretations I, N and a valuation v, the *value of a* KFOL *formula* A *w.r.t.* D, I, N, v, denoted by $A^{I,N}_v$, is defined as follows:[4]

$$A^{I,N}_v \stackrel{\text{def}}{=} \mathbb{A}^I_v,$$

where $\mathbb{A}$ is obtained from A by substituting subformulas of the form '$not\ C$' by truth values $(not\ C^N_v)$.[5] ◁

[4] Recall that $\mathbb{A}^I_v$ is defined in Definition 6.

[5] Note that C^N_v is a truth value, so $(not\ C^N_v)$ is a truth value, too.

Strongly supported models can now be defined.

Definition 11. Let Π be an ASP^K program. A model N for Π is *strongly supported* provided that there is a sequence of interpretations I_i with $i = 0, \ldots, m$ for some $m \in \omega$, such that $N = \bigcup_{0 \leq i \leq m} I_i$, and:

(i) for every fact of the form '$\ell_1 \vee \ldots \vee \ell_k$', at least one of literals of $\ell_1, \ldots, \ell_k$ is in I_0;

(ii) for every $0 < n \leq m$, every rule '$\ell_1 \vee \ldots \vee \ell_k \leftarrow A$' in Π and every valuation v, if $A_v^{\bigcup_{0 \leq j \leq n-1} I_j, N} = \mathrm{T}$ then a (possibly empty) subset of $\{v(\ell_1), \ldots, v(\ell_k)\}$ is included in I_n;

(iii) for $i = 0, \ldots, m$, I_i can only contain literals obtained by applying points (i) and (ii) specified above. ◁

The following examples illustrate various aspects of Definition 11.

Example 1. Let the domain be $D_1 = \{a, b\}$ and let Π_1 be the program:

$r(X) \leftarrow p(X).$
$p(a).$

Then, using the notation of Definition 11, we have that $I_0 = \{p(a)\}$, $I_1 = \{r(a)\}$. Of course, $N = I_0 \cup I_1 = \{p(a), r(a)\}$ is a model for Π_1 so it is a strongly supported model. On the other hand, $\{p(a), r(a), r(b)\}$ is a model of Π_1 but is not strongly supported. ◁

Example 2. Let the domain be $D_2 = \{a, b\}$ and let Π_2 be the program:

$r(X) \leftarrow \neg q(X) \wedge not\ p(X).$
$\neg q(a).$
$q(b).$

Then $N = \{\neg q(a), q(b), r(a)\}$ is a strongly supported model for Π_2. Using again the notation of Definition 11, we have that $N = I_0 \cup I_1$, where $I_0 = \{\neg q(a), q(b)\}$, $I_1 = \{r(a)\}$. This follows from the fact that for $v(X) = a$:

$$\begin{aligned}\big(\neg q(X) \wedge not\ p(X)\big)_v^{I_0, N} &= (\neg q(X))_v^{I_0} \wedge not\ p(X)_v^N \\ &= \neg q(a)^{I_0} \wedge not\ p(a)^N = \mathrm{T}.\end{aligned}$$ ◁

Example 3. Let the domain be $D_3 = \emptyset$ and let Π_3 be the program:

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

Then the only strongly supported model for Π_3 is $N = \{p\}$. Here $N = I_0 \cup I_1$, where $I_0 = \emptyset$ (there are no facts) and $I_1 = \{p\}$ since for the third rule we have:

$$(not\ q)^{I_0, N} = (not\ q^N) = not\ \mathrm{U} = \mathrm{T}.$$

On the other hand, $N' = \{p, q\}$ is not strongly supported. Again, $I_0 = \emptyset$, so for the first rule we have $q^{I_0,N} = \text{U}$, for the second rule we have $(not\ p)^{I_0,N} = \text{F}$ and, for the third rule, $(not\ q)^{I_0,N} = \text{F}$. Therefore, no rule produces new results. ◁

The following example shows that strongly supported models do not have to be minimal.

Example 4. Consider the following program Π_4 over domain $D_4 = \{a\}$:

$q(X) \leftarrow p(X).$
$p(a) \vee q(a).$

According to Definition 11, program Π_4 has two strongly supported models: $N = \{q(a)\}$ and $N' = \{p(a), q(a)\}$. Of course, N' is not minimal since $N \subsetneq N'$. ◁

The following definitions and theorem show the relation between stable models and strongly supported models. For a discussion of stable models see [19].

Definition 12. An interpretation I for a Kleene answer set program Π is a *stable model* for Π provided that I is a minimal strongly supported model for Π. ◁

We shall consider two versions of ASP.

Definition 13.

- By ASP^K we understand Kleene answer set programs with the semantics given by strongly supported models.
- By ASP$^K_{min}$ we understand Kleene answer set programs with the semantics given by stable models. ◁

Theorem 1.

1. For answer set programs with rules of the (traditional) form, where all ℓ_i are literals and $k > 0$:[6]

   $$\ell_1 \vee \ldots \vee \ell_k \leftarrow \ell_m, \ldots, \ell_n, not\ \ell_s, \ldots, not\ \ell_t,$$

   ASP$^K_{min}$ coincides with answer set programming in the traditional sense (as presented, e.g., in [5, 16, 19]).
2. If rules of an ASP^K program Π are all of the form '$\ell \leftarrow A$.', where ℓ is a literal, then a model I for Π is stable iff I is strongly supported.

Proof.
1. Let I be a stable model in the sense of Definition 12. Then I is both strongly supported and minimal.

[6] The case when $k = 0$ (constraints) is dealt with in Section 4.

Observe that strongly supported models are constructed in such a way that all formulas of the form '$not\ C$' are evaluated in the context of the final interpretation N (see Definition 11) and, for a given valuation v, they have fixed truth values $(not\ C_v^N)$. By assumption, C is a literal. In the traditional definition of stable models [19], the notion of reduct corresponds to substituting '$not\ C$' by $(not\ C_v^N)$ and removing redundant literals and rules. Now minimality guarantees stability in the traditional sense.

2. Of course, by definition, stability implies strong supportedness.

To prove that strong supportedness implies stability, suppose that there are two strongly supported models, J and J' such that $J \subsetneq J'$. By the construction of strongly supported models, $J' = J \cup K$, where literals in K are obtained by point (ii). of Definition 11. However, that would mean that J was not a model of Π as literals are added in (ii) only when there is a rule with true body and head not being true. Such literals are uniquely determined (by assumption heads contain single literals). ◁

4 Constraints

Given a standard ASP program Π, let $NC(\Pi)$ be the rules in Π that are not constraint rules and let $C(\Pi)$ be the rules in Π that are constraint rules, where $NC(\Pi) \cup C(\Pi) = \Pi$. There are at least two alternatives for providing a semantic theory for constraints.

- In the first, one first computes the stable models for $NC(\Pi)$ and then eliminates those models in $NC(\Pi)$ that do not satisfy the constraint rules in $C(\Pi)$.
- In the second, one simply computes the stable models of $NC(\Pi) \cup C(\Pi)$.

The first alternative is that used traditionally for standard ASP (see, e.g., [16]). It is the basis for many of the most prominent implementations of ASP in the literature and is in fact similar to filtered circumscriptive approaches. The second method, which appears to be as intuitive, is similar to non-filtered circumscriptive approaches. These methods apply equally well for strongly supported models and ASP^K. In Definition 11, the second alternative is used. However, one can easily adjust the definition to reflect the first alternative.

The following example shows that these approaches are not equivalent.

Example 5. Consider program Π_5 (see [2, Example 32]):

$$a \vee b \leftarrow . \tag{12}$$

$$a \vee c \leftarrow . \tag{13}$$

$$\leftarrow a \wedge (not\ b) \wedge (not\ c). \tag{14}$$

$$\leftarrow (not\ a) \wedge b \wedge c. \tag{15}$$

According to the standard ASP semantics which uses the first alternative, Π_5 has no stable models. This follows from the fact that stable models of Π_5 without constraints (i.e., with only rules (12), (13)) are $\{a\}$ and $\{b, c\}$, and these models do not satisfy the constraints (14), (15).

When using the second alternative, when all rules (12)–(15) participate in computing models, there are two stable models $\{a, b\}, \{a, c\}$, which are also strongly supported models. The explanation used in [2] is that these models are not minimal for Π_5 without constraints, so they should not be considered. On the other hand, these models are both minimal and strongly supported for the theory expressed by Π_5 (in which case $\{a\}, \{b, c\}$ are not models, so are not considered in checking minimality). Note also that $\{a, b, c\}$ is a strongly supported but non-minimal model when Π_5 is interpreted using ASP^K. ◁

Remark 3.

- Theorem 1(ii) holds when we allow constraints with the traditional semantics, since one first computes models and later eliminates those not satisfying constraints. Strongly supported models are in this case minimal and remain so even after filtering out some of them.
- Theorem 1(ii) also holds for the case when constraints participate in finding models: the models are in this case minimal models satisfying the whole program consisting of rules and constraints. ◁

5 Minimality Revisited

In this section, we consider some relationships between stability, minimality [13, 20], and strong supportedness. When focussing on normal ASPs that do not contain disjunctive rules or constraints, stable model construction naturally generates only minimal models due to the lack of choice in the iterated construction from base facts. On the other hand, when extending expressivity to include rules with disjunctive heads and constraint rules, choice in the iterated construction of models and the different alternatives that can be used in applying constraints, open up opportunities for making different semantic choices when interpreting ASPs.

These choices become very clear when one bases semantic theories for ASPs on an explicit three-valued logic together with distinguishing strong supportedness from minimality. As is often the case with semantic intuitions associated with nonmonotonic formalisms, due to the space of choices, one has a number of different alternatives to choose from. The approach taken in this paper is simply to clarify these choices in as lucid a manner as possible and provide mechanisms for leaving the choice up to the knowledge engineer.

In the case of enforcing minimality in ASP theories, there are a number of arguments, not against minimality in principle, but for enforcing strong supportedness instead. For the case of normal ASPs (no disjunctive heads), strong supportedness and minimality are equivalent. In the case of ASPs with disjunctive rules, there is a history in the nonmonotonic literature of viewing minimality assumptions applied to disjunctions with suspicion, both for intuitive and pragmatic reasons as some of the examples have shown.

One of the more interesting reasons for not only having the capability to distinguish between supportedness and minimality, but to also be able to only construct strongly

supported models in isolation, is a complexity argument (see Theorem 2 in Section 6). Complexity for constructing strongly supported models in the case of ASPs with disjunctive rules is lower than the complexity of constructing minimal, strongly supported models. Additionally, since one has the capability of applying local closed world assumptions to specific relations due to the default open world assumption associated with the ASP framework, one seems to have the best of both worlds. In the following, a number of examples are provided to further clarify the subtle relationships between stability, minimality and supportedness.

The following example is from [19].

Example 6. Consider the domain $D_6 = \{a, b\}$ and program Π_6:

$$r(X) \leftarrow p(X) \wedge not\ q(X).$$
$$p(a).\ \ p(b).\ \ q(a).$$

Then Π_6 has two minimal models:

$$I_0 = \{p(a), p(b), q(a), r(b)\}, I_1 = \{p(a), p(b), q(a), q(b)\}.$$

According to [19], I_0 is a "good" model (an answer set) while I_1 is "bad" (not an answer set). The explanation given in [19] is related to an argument based on program completion. A more direct explanation is that the fact $q(b)$ appearing in I_1 is not supported (not being a conclusion of the rule of Π_6). ◁

Example 7. Let the program domain be $D_7 = \{jack, john, xco\}$ and Π_7 consist of the following rule and facts:[7]

$$luckyBroker(X) \vee successfulBroker(X) \leftarrow \tag{16}$$
$$\forall Y(invests(X, Y) \Rightarrow makesProfit(X, Y)). \tag{17}$$
$$\neg luckyBroker(X) \leftarrow not\ \big(\forall Y(invests(X, Y) \Rightarrow makesProfit(X, Y))\big). \tag{18}$$
$$perfectBroker(X) \leftarrow successfulBroker(X) \wedge luckyBroker(X). \tag{19}$$
$$invests(jack, xco). \tag{20}$$
$$invests(john, xco). \tag{21}$$
$$makesProfit(jack, xco). \tag{22}$$

According to the ASP^K semantics, Π_7 has the following strongly supported models:

$$\big\{(20), (21), (22), luckyBroker(jack), \neg luckyBroker(john)\big\}, \tag{23}$$
$$\big\{(20), (21), (22), successfulBroker(jack), \neg luckyBroker(john)\big\}, \tag{24}$$
$$\big\{(20), (21), (22), luckyBroker(jack), successfulBroker(jack), \tag{25}$$
$$perfectBroker(jack), \neg luckyBroker(john)\big\}$$

[7] Observe that the right arrow $\Rightarrow$ is used in formulas in the antecedent of a rule according to the syntax of first-order formulas allowed in the antecedent to a rule, whereas the left arrow $\leftarrow$ is used to distinguish between the antecedent and consequent of a rule.

According to the ASP^K_{min} semantics, Π_7 has only (stable) models (23) and (24). But model (25) makes perfect intuitive sense and it is questionable whether or not it should be omitted. ◁

Recall that one can close the world locally, using rules of the form (9). However, such closures minimize relations in the classical sense (by minimizing their positive instances while maximizing negative ones). In Definition 9, minimality is defined w.r.t. both positive and negative instances. Since the rule (9) adds literals, in general such closures do not preserve the set of stable models nor strongly supported models and may seriously affect the result, as the following example shows.

Example 8. Let the domain be $D_8 = \{a\}$ and let Π_8 be the program:

$$r(X) \leftarrow not\ p(X).$$
$$s(X) \leftarrow \neg p(X).$$

The only strongly supported model for Π_8, being also its stable model, is $\{r(a)\}$.
By closing the relation p in Π_8 by rule (9), we obtain a program with the only strongly supported (and stable) model $\{\neg p(a), r(a), s(a)\}$. ◁

6 Computing Strongly Supported Models and Complexity Results

Algorithm 1 allows us to compute strongly supported models. The intuition is that we first guess a candidate for a strongly supported model and then we check whether the model is strongly supported (i.e., can be generated from facts, using rules).

In the following theorem we consider data complexity. In the answer set programming literature, expression (program) complexity is more common, mainly due to the use of grounding. On the other hand, ASP^K is a database language (by restriction to finite domains) and we do not use grounding. Therefore data complexity is more relevant here.

Theorem 2.

1. Checking whether an ASP^K program has a strongly supported model is Σ^P_1-complete (i.e., NP-complete).
2. Checking whether an ASP^K_{min} program has a stable model is Σ^P_2-complete.

Proof.

1. To prove the first claim it suffices to observe that ASP^K is at least as expressive as ASP, so the considered problem is NP-hard. To show that it is in NP, we use Algorithm 1 which runs in time polynomial w.r.t. the size of the domain D (assuming the input program Π is fixed, so has size bounded by a constant).
2. To prove the second claim we first show hardness of the considered problem for Σ^P_2 and then we show that the problem actually is in Σ^P_2.

Algorithm 1: Computing strongly supported models for ASP^K programs.

Input: An ASP^K program Π with domain D, consisting of a set of rules S.
Output: A nondeterministically computed strongly supported model N for Π.

/* For notation see Definition 10 */

generate nondeterministically an interpretation N with constants from D;
if there is $r \in S$ and a valuation v of variables in D with $r_v^N \neq \mathrm{T}$ **then**
  reject N; /* N is not a model for P */
  stop;

/* otherwise N is a candidate for a strongly supported model. */

/* In the remaining part of the algorithm we verify whether N */
/* is a strongly supported model by generating a supported interpretation I */
/* and checking whether $N = I$. */

/* We use the fact that during computations I cannot decrease and always $I \subseteq N$. */

set $I = \emptyset$;
repeat
  set $W = \{(r, v) \mid r \in S$ and v is valuation of variables in D with $r_v^{I,N} \neq \mathrm{T}\}$;
  foreach $(r, v) \in W$ **do**
    /* Let $r =$ '$\ell_1 \vee \ldots \vee \ell_k \leftarrow A$.' */
    if $N \cap \{v(\ell_1), \ldots, v(\ell_k)\} = \emptyset$ **then reject** N; **stop**;
      /* no subset of $\{v(\ell_1), \ldots, v(\ell_k)\}$, when added to I, can make $r_v^{I,N} = \mathrm{T}$ */
      /* without violating the invariant $I \subseteq N$ */
    else set $I = I \cup (N \cap \{v(\ell_1), \ldots, v(\ell_k)\})$;
until $W = \emptyset$;
if $N = I$ **then accept** N as a strongly supported model for Π
else reject N;

(a) By results of [9, Section 3],[8] checking whether an ASP$^K_{min}$ program has a stable model is Σ_2^P-hard.
(b) To show that checking whether an ASP$^K_{min}$ program has a stable model is in Σ_2^P, we encode the problem by a second-order formula of the form:

$$\exists \bar{P} \forall \bar{R}\ A(\bar{P}, \bar{R}), \tag{26}$$

where $\exists \bar{P} \forall \bar{R}$ are all second-order quantifiers used.
To check whether an ASP^K program Π has a stable model we first guess the model and then check the model for minimality:
  – guessing the model can be expressed by using existential second-order quantifiers $\exists \bar{P}$, where $\bar{P}$ are all relations in Π;
  – checking the guessed model for minimality can be done in a manner similar to circumscription, where quantifiers $\forall \bar{R}$ are used – for details of how the suitable formula can look like see, e.g., [13].

[8] With a suitable encoding allowing one to move from expression complexity to data complexity.

As a result, one obtains a formula of the required form (26), meaning that the considered problem is indeed in Σ_2^P. ◁

The above theorem shows that the minimality requirement raises complexity from NP (i.e., Σ_1^P) to Σ_2^P in terms of the polynomial hierarchy.

Remark 4. Note that Algorithm 1 treats rules and constraints uniformly.

The algorithm can be easily modified for the case when constraints are separated. Namely, S in the algorithm should consist of all rules with nonempty heads (not being constraints). After generating strongly supported models one should check whether such models satisfy the constraints and reject models not satisfying them. ◁

7 Conclusions

This paper has explored the subtle relationship between stability, supportedness and minimality in the context of Answer Set Programming. This has been done by making a formal distinction between two characteristics of stable models, strong supportedness and minimality. This was done by introducing a modified first-order three-valued Kleene Logic used as the semantic basis for interpreting Kleene answer set programs in the language of ASP^K. Strongly supported models were then defined. It was shown that strongly supported models and stable models are equivalent in the context of normal answer set programs, where no constraint rules or disjunctive rules are allowed. With the addition of disjunctive rules, use of strongly supported models as a semantic basis differs from stable models due in part to the separation of support and minimality. An argument is presented for using (strongly) supported models as a semantic interpretation of disjunctive answer set programs. The argument is not exclusive since one can combine minimality and supportedness if so desired. One of the advantages of not doing this is based on a complexity argument. Expressiveness of answer set programs can be extended to the use of arbitrary first-order formulas in the antecedents of rules without any modification to the underlying semantics. A non-deterministic algorithm for generating strongly supported models is also provided. Additionally, an analysis of constraint rules is provided with consideration of two alternative approaches to their application, leading to two different semantic interpretations of answer set programs with constraint rules.

References

1. Alviano, M., Faber, W., Leone, N., Perri, S., Pfeifer, G., Terracina, G.: The disjunctive datalog system DLV. In: de Moor, O., Gottlob, G., Furche, T., Sellers, A. (eds.) Datalog Reloaded. LNCS, vol. 6702, pp. 282–301. Springer (2010)
2. Baral, C.: Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge University Press (2003)
3. Bonatti, P., Calimeri, F., Leone, N., Ricca, F.: Answer set programming. In: Dovier, A., Pontelli, E. (eds.) A 25-Year Perspective on Logic Programming: Achievements of the Italian Association for Logic Programming, GULP. LNCS, vol. 6125, pp. 159–182. Springer (2010)

4. Brewka, G.: Preferences, contexts and answer sets. In: Dahl, V., Niemelä, I. (eds.) Proc. 23rd International Conference, ICLP 2007. LNCS, vol. 4670, p. 22. Springer (2007)
5. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM 54(12), 92–103 (2011)
6. Brewka, G., Niemelä, I., Truszczynski, M.: Answer set optimization. In: Gottlob, G., Walsh, T. (eds.) Proc. 18th IJCAI. pp. 867–872. Morgan Kaufmann (2003)
7. Cabalar, P., Pearce, D., Valverde, A.: A revised concept of safety for general answer set programs. In: Erdem, E., Lin, F., Schaub, T. (eds.) Proc. 10th Int. Conf. Logic Programming and Nonmonotonic Reasoning. LNCS, vol. 5753, pp. 58–70. Springer (2009)
8. Denecker, M., Marek, V., Truszczynski, M.: Stable operators, well-founded fixpoints and applications in nnonmonotonic reasoning. In: Minker, J. (ed.) Logic-based Artificial Intelligence, pp. 127–144. Kluwer Academic Pub. (2000)
9. Eiter, T., Gottlob, G.: Complexity results for disjunctive logic programming and application to nonmonotonic logics. In: Miller, D. (ed.) Logic Programming, Proceedings of the 1993 International Symposium. pp. 266–278 (1993)
10. Fages, F.: A new fixpoint sematics for general logic programs compared with the well-founded and stable model semantics. New Generation Computing 9, 425–443 (1991)
11. Fages, F.: Consistency of Clark's completion and existence of stable models. Methods of Logic in Computer Science 1, 51–60 (1994)
12. Fenstad, J.E.: Situations, Language and Logic. D. Reidel Publishing Company (1987)
13. Ferraris, P., Lifschitz, V.: On the minimality of stable models. In: Balduccini, M., Son, T. (eds.) Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday. LNCS, vol. 6565, pp. 64–73. Springer (2011)
14. Fitting, M.: A Kripke-Kleene semantics for logic programs. J. Logic Programming 2(4), 295–312 (1985)
15. Fitting, M.: The family of stable models. J. Logic Programming 17(2/3&4), 197–225 (1993)
16. Gelfond, M., Kahl, Y.: Knowledge Representation, Reasoning, and the Design of Intelligent Agents - The Answer-Set Programming Approach. Cambridge University Press (2014)
17. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R., Bowen, K. (eds.) Proc. of Int'l Logic Programming. pp. 1070–1080. MIT Press (1988)
18. Kleene, S.C.: On a notation for ordinal numbers. Symbolic Logic 3, 150–155 (1938)
19. 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 (2010)
20. Lonc, Z., Truszczynski, M.: Computing minimal models, stable models and answer sets. TPLP 6(4), 395–449 (2006)
21. Pearce, D.: Equilibrium logic. Annals of Mathematics and AI 47(1-2), 3–41 (2006)
22. Przymusinski, T.: Stable semantics for disjunctive programs. New Generation Comput. 9(3/4), 401–424 (1991)
23. Shepherdson, J.C.: A sound and complete semantics for a version of negation as failure. Theoretical Computer Science 65(3), 343 – 371 (1989)