Argumentation-Based Semantics for Logic Programs with First-Order Formulae

Phan Minh Dung[1], Tran Cao Son[2], and Phan Minh Thang[3]

[1]Department of Computer Science, Asian Institute of Technology, Thailand
[2]Department of Computer Science, New Mexico State University, Las Cruces, NM, USA
[3]Department of Computer Science, Burapha University International College, Thailand

Abstract. This paper studies different semantics of logic programs with first order formulae under the lens of argumentation framework. It defines the notion of an *argumentation-based answer set* and the notion of an *argumentation-based well-founded model* for programs with first order formulae. The main ideas underlying the new approach lie in the notion of a *proof tree* supporting a conclusion given a program and the observation that proof trees can be naturally employed as arguments in an argumentation framework whose stable extensions capture the program's well-justified answer semantics recently introduced in [23]. The paper shows that the proposed approach to dealing with programs with first order formulae can be easily extended to a generalized class of logic programs, called programs with *FOL-representable atoms*, that covers various types of extensions of logic programming proposed in the literature such as weight constraint atoms, aggregates, and abstract constraint atoms. For example, it shows that argumentation-based well-founded model is equivalent to the well-founded model in [27] for programs with abstract constraint atoms. Finally, the paper relates the proposed approach to others and discusses possible extensions.

1 Introduction

Answer set semantics for logic programs [12] is one of the most widely adopted semantics for logic programs—i.e., logic programs that allow negation as failure in the body of the rules. It is a natural extension of the minimal model semantics of positive logic programs to the case of normal logic programs. Answer set semantics provides the theoretical foundation for *answer set programming* [18, 16] which has proved to be useful in several applications such as diagnosis, bioinformatics, planning, etc. (see, e.g., [1, 2, 6, 3, 11, 14]).

A set of atoms S is an *answer set* of the program Π if S is the minimal model of the positive program Π^S (the *reduct of Π with respect to S*), obtained from the *Gelfond-Lifschitz transformation* by (*i*) removing from Π all the rules whose body contains a negation as failure literal $not\ b$ which is false in S (i.e., $b \in S$); and (*ii*) removing all the negation as failure literals from the remaining rules.

One of the most interesting properties of answer sets that can be derived from the above definition is that each atom in an answer set is *non-circular justifiable*, i.e., for each atom a there exists a proof tree for a that does not involve a in any of the proof step.

The successes of answer set programming (ASP) and the needs for a more expressive and simple modeling language led to several extensions of the language such as weight constraint atoms [19], aggregates atoms (e.g, [10, 20]), abstract constraint atoms (e.g., [17, 25]), logic programs with first order formulae (e.g., [4, 23]). The notion of an answer set has been extended to various extensions of logic programming and one of the contentious issue in this endeavor is related to the circular justifiability of atoms belonging to an answer set. This problem has been discussed extensively in the literature and can be seen in the following example.

Example 1. Consider the program Π_1 with aggregates discussed in [23]:

$$p(1) \leftarrow \tag{1}$$

$$p(2) \leftarrow p(-1). \tag{2}$$

$$p(-1) \leftarrow \text{SUM}(\{X : p(X)\}) \geq 1. \tag{3}$$

where $\text{SUM}(\{X : p(X)\}) \geq 1$ represents an aggregate atom; informally, it is true in an interpretation I if $\Sigma_{p(x) \in I} x \geq 1$. This program has an answer set $\{p(1), p(2), p(-1)\}$ according to [10] but does not admit any answer set according to many other definitions (e.g., [20, 25, 23]). The issue of this answer set, as discussed in [23], lies in that $p(2)$ is circular justified by the sequence $p(2) \Leftarrow p(-1) \Leftarrow \text{SUM}(\{X : p(X)\}) \geq 1 \Leftarrow p(2)$. $\Diamond$

It is easy to see that the five Herbrand interpretations of the above program $\{p(2)\}$, $\{p(2), p(1)\}$, $\{p(2), p(-1)\}$, $\{p(2), p(1), p(-1)\}$, and $\{p(1)\}$, where elements not belonging to an interpretation are assumed to be false, are the only ones satisfying the atom $\text{SUM}(\{X : p(X)\}) \geq 1$. Let us denote with λ the atom $\text{SUM}(\{X : p(X)\}) \geq 1$ and C_λ be the collection of rules:

$$\lambda \leftarrow p(2), \neg p(1), \neg p(-1). \tag{4}$$

$$\lambda \leftarrow p(2),\ p(1), \neg p(-1) \tag{5}$$

$$\lambda \leftarrow p(2),\ p(-1), \neg p(1). \tag{6}$$

$$\lambda \leftarrow p(2),\ p(1),\ p(-1). \tag{7}$$

$$\lambda \leftarrow p(1), \neg p(2), \neg p(-1). \tag{8}$$

These rules basically provide the definition for the atom $\text{SUM}(\{X : p(X)\}) \geq 1$, i.e., they define when it is true. Let Π_λ be the program obtained from Π_1 by replacing $\text{SUM}(\{X : p(X)\}) \geq 1$ with λ. It is easy to check that the program $\Pi_\lambda \cup C_\lambda \cup \{\neg p(X) \leftarrow \ \mathit{not}\ p(X) \mid X \in \{1, 2, -1\}\}$ does not have an answer set. As such, it is reasonable to conclude that Π_1 is inconsistent. This argument is similar to the one used in [24] to show that Π_1 is inconsistent, i.e., Π_1 does not have an answer set.

It is interesting to observe that if we were to construct a SLD-proof[1] for $p(-1)$ given the program Π_1, assuming that $\mathit{not}\ p$ implies $\neg p$ (i.e., assuming the negation-as-failure rule or NAF-rule) we will eventually have to make use of the rules in C_λ.

[1] SLD stands for "**s**elective **l**inear **d**efinite" (see, e.g., [15]).

For example, a proof tree using (8) is depicted in Figure 1(left); a proof tree using (5) is depicted in Figure 1(right) where T_1 denotes a proof tree for $p(-1)$. It is easy to verify that every tree supporting $p(-1)$ from any set of assumptions, which is a subset of $\{not\, p(1),\ not\ p(2),\ not\ p(-1)\}$, under the program Π_1 is inconsistent in the sense that it assumes that $p(-1)$ is false ($not\ \ p(-1)$) to conclude that $p(-1)$ is true. In fact, the dependence discussed in [23] can be extracted from these proof trees.

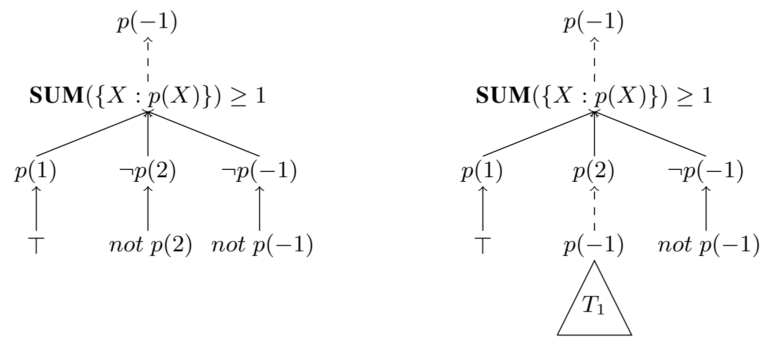

Fig. 1: Possible Proof Trees for $p(-1)$

It is not difficult to see that a proof tree constructed from a given program—informally defined as above—represents an argument supporting the conclusion of the literal at its root. Furthermore, there can be several arguments constructible from a program. The question is then which arguments should be *acceptable*, a central question in the studies of argumentation framework.

It is worth noticing that argumentation framework is another line of research that has its root in the study of logic programming and nonmonotonic reasoning. In fact, the landmark paper [9] originated from a paper studying the acceptability semantics of logic programming [8]. The proposed argumentation framework in [9] laid the foundation for the development of several argumentation-based theories and applications. Within logic programming, it has been showed in [9] that different semantics of argumentation frameworks such as grounded extensions and stable extensions correspond to the well-founded and answer set semantics of normal logic programs. In a recent paper, [22] showed that the 3-valued stable model of logic programming [21] can also be viewed as a semi-stable labeling of a corresponding assumption-based argumentation framework [5].

Can argumentation-based semantics be extended to more generalized logic programs? The main purpose of this paper is to investigate different semantics of logic programs with first order formulae under the view of argumentation framework. The advantages of this study are twofold. First, it shows that the traditional approach to studying the semantics of logic programs using argumentation framework can be generalized to more generalized programs. Second, it allows for the definition of the well-founded semantics for programs with first order formulae that has not been studied thus far. To the best of our knowledge, this is the first time the notion of a well-founded model for logic programs with first order formulae is discussed.

To summarize, the paper contributes to both areas of logic programming and argumentation framework. Regarding logic programming, the paper proposes to consider proof trees as arguments and to use different semantics of argumentation framework as the semantics of the original program. The paper then extends the proposed approach to a generalized class of logic programs, called programs with FOL-representable atoms, which covers several extensions of logic programs such as programs with aggregates, programs with abstract constraint atoms, and programs with weight constraint atoms. Regarding argumentation framework, the paper demonstrates that its principle is applicable in various extensions of logic programming. The equivalent results in this paper indicate that argumentation-based semantics can be used as a means to study different approaches to defining semantics of those extensions.

2 Background

In this section, we review the basics of argumentation framework and logic programs with first order formulae.

2.1 Argumentation Framework

An abstract *argumentation framework* (AF) [9] AF is a pair $(Args, Atts)$ where $Args$ is a set of abstract entities called arguments and $Atts \subseteq Args \times Args$ is the attack relation between arguments. An argument $a \in Args$ attacks an argument $b \in Args$ if $(a, b) \in Atts$. a is called an attacker of b if a attacks b. The set of arguments $S \subseteq Args$ *attacks* b if there exists $a \in S$ such that a attacks b. a (respectively S) *defends* an argument c against its attacker b if a (respectively S) attacks b. S is *conflict free* if it does not attack itself. S is *admissible* if it is conflict free and defends against every of its attackers. The *characteristic function* of AF is defined by

$$F_{AF}(S) = \{a \mid a \in Args, S \text{ defends } a\}.$$

Since F_{AF} is a monotonic function, the sequence

$$F_{AF}(\emptyset), F_{AF}(F_{AF}(\emptyset)), \ldots, F^n_{AF}(\emptyset) = \underbrace{F_{AF}(\ldots F_{AF}(F_{AF}}_{n\ times}(\emptyset))), \ldots$$

converges to its least fixpoint, denoted by *lfp*(F_{AF}). By this definition, it is easy to see that *lfp*(F_{AF}) is unique.

Given an $AF = (Args, Atts)$, a conflict free set of arguments $S \subseteq Args$ is a *stable extension* of AF if it attacks each argument $A \not\in S$; a *grounded extension* of AF if $S =$ *lfp*(F_{AF}); and a *preferred extension* of AF if it is a subset-maximal admissible set of AF. The focus of this paper is the two semantics associated with stable and grounded extensions of argumentation framework.

2.2 Logic Programs with FOL-Formulae

Let $\Sigma = (\mathcal{C}, \mathcal{P}, \mathcal{F})$ be a signature with finite set of constants $\mathcal{C}$, finite set of predicate symbols $\mathcal{P}$, and finite set of function symbols $\mathcal{F}$. We assume that $\mathcal{P}$ contains two 0-ary symbols $\top$ and $\bot$, denoting *truth* and *falsity* respectively. Let L_Σ be the first-order logic language with equality over Σ. We will make use of the usual notions in first-order logic (FOL) such as term, atom, literal, interpretation, satisfaction of a formula w.r.t. an interpretation etc. without precise definition. We refer the readers to [15, 23] for detail.

A *logic program with FOL-formulae* is a finite set of rules of the form $\phi \leftarrow \psi$ where ϕ and ψ are classical first order logic formulae in L_Σ such that $\phi \neq \top$ and $\psi \neq \bot$. For a rule $r = \phi \leftarrow \psi$, $head(r)$ and $body(r)$ denote ϕ and ψ respectively. When ϕ is an atom and ψ is a conjunction of literals, we say that the rule is a *normal rule*. A program is *normal* if every of its rules is normal.

Given a program Π, H_Π denotes the Herbrand base of Π excluding $\top$ and $\bot$. For a program Π, $ground(\Pi)$ denotes the set of ground instantiations of rules in Π using the set of constants occurring in Π. By an interpretation of Π, we mean a Herbrand interpretation. In this paper, we will assume that $ground(\Pi)$ is finite.

A *partial interpretation* of Π is a pair (P, Q) such that $P, Q \subseteq H_\Pi$ and $P \cap Q = \emptyset$. Given a Herbrand interpretation I of Π, we denote with $\neg I^-$ the set $\{\neg a \mid a \in H_\Pi \setminus I\}$ and say that I satisfies a rule $r \in ground(\Pi)$ if $I \cup \neg I^- \models body(r)$ implies $I \cup \neg I^- \models head(r)$ ($\models$ denotes the usual logical entailment relation). I *satisfies* a program Π (or I is a *model* of Π) if it satisfies every rule in $ground(\Pi)$.

Let Π be a program with FOL-formulae and I be an interpretation of Π. Let $f\Pi^I$ be the program obtained from $ground(\Pi)$ by (*i*) eliminating all the rules whose bodies are not satisfied by I; and (*ii*) adding the negative literals in $\neg I^-$ as constraints to the resulting program. For two first order theories O and N, let $T_\Pi(O, N) = \{head(r) \mid r \in ground(\Pi), O \cup N \models body(r)\}$. A model I of a program Π is a *well-justified answer set* of Π if $lfp(T_{f\Pi^I}(\emptyset, \neg I^-)) \cup \neg I^- \models a$ for every $a \in I$ where $lfp(T_{f\Pi^I}(\emptyset, \neg I^-))$ denotes the least fixpoint of the function $T_\Pi(., \neg I^-)$.

Example 2. Consider the program from [23]:

$$a \vee (\neg b \wedge c) \leftarrow \neg a \wedge (\neg c \vee c).$$
$$d \leftarrow c.$$

Let us call the program Π_2. It is easy to see that $I = \{c, d\}$ is a model of Π_2 since $\neg I^- = \{\neg a, \neg b\}$ and for each rule r in Π_2, $I \cup \neg I^- \models body(r)$ and $I \cup \neg I^- \models head(r)$. It is also easy to see that $f\Pi_2^I$ consists of Π_2 together with the clause $\neg a \leftarrow$. $\Diamond$

Observe that the interpretation $I' = \{a, d\}$ is also a model of Π_2. However, d cannot be justified by this model. It can be seen that only I is considered as a well-justified answer set of Π_2.

3 Argumentation Framework and Logic Programs with FOL-Formulae

In this section, we define different semantics for logic programs with FOL formulae. In the following, whenever we refer to a program, we mean a logic program with FOL formula whose signature is assumed to be known.

3.1 Argumentation-Based Semantics for Logic Programs with FOL Formulae

Let Π be a program and L_Π be the set of ground formulae formed over H_Π. We extend the program Π (and hence, $ground(\Pi)$) with rules: (*i*) $\bot \leftarrow a \wedge \neg a$, denoted by $\mathbf{F}_a$, for each atom a in H_Π; (*ii*) $\neg a \leftarrow \ \ not\ a$, denoted by $\mathbf{A}_a$, where not is the default negation and is not a symbol in the language of Π for each atom a in H_Π. Intuitively, $\mathbf{F}_a$ indicates that if both a and $\neg a$ are provable then Π is inconsistent as it derives falsity; $\mathbf{A}_a$ encodes the negation-as-failure rule that says that if a is not provable then $\neg a$ can be concluded.

Definition 1 (Proof Tree). A *proof tree* (or *tree*) for a formula σ w.r.t. a program Π is a finite tree with nodes labelled by formulae in $L_\Pi \cup \{\top, \bot\} \cup \{not\, a \mid a \in H_\Pi\}$ such that

1. the root is labelled by σ;
2. for every node N labelled by φ if N is not a leaf node and has n children, labelled by $\varphi_1, \ldots, \varphi_n$ then one of the following conditions is satisfied:
   - $\varphi_1 \wedge \ldots \wedge \varphi_n \models \varphi$; or
   - $\varphi \leftarrow \varphi_1 \wedge \ldots \wedge \varphi_n$ is in $ground(\Pi)$.
3. a leaf of the tree must be either $\top$ or $\ not\ a$ for some $a \in H_\Pi$.

Intuitively, a tree represents a possible derivation of the formula at its root given the rules of the program and the assumptions made at its leaves. The formula labeled an interior node is either a logical consequence of the conjunction of formulae labeled its children (first case of Item 2) or it is the head of a ground rule whose body is the conjunction of formulae labeled its children (second case of Item 2).

For a tree T and a node N in T, $label(N)$ denotes the formula that labels N. The *conclusion* of a tree T, denoted by $Concl(T)$, is the formula labelling its root. The support of a tree T, denoted by $Support(T)$, is the set $\{label(N) \mid N$ is a leaf, $label(N) \neq \top\}$. A tree T is *strict* if $Support(T) = \emptyset$. For a set of trees S, $Concl(S) = \{Concl(T) \mid T \in S\}$.

Example 3. In Figure 2, we can see different types of nodes:
- the node whose label is $a \vee (\neg b \wedge c)$ whose connections to their children are dashed lines are constructed from rules in the program (constructed using the second case in Item 2 of Def. 1, via $a \vee (\neg b \wedge c) \leftarrow \neg a \wedge (\neg c \vee c)$ of Π_2).
- the node whose label is $\neg a$ with a unique child whose label is $\ not\ a$ (constructed using the second case in Item 2 of Def. 1, via the rule $\mathbf{A}_a$).
- the node whose label is the valid formula $\neg c \vee c$ with a unique child whose label is $\top$ (constructed using the first case in Item 2 of Def. 1).

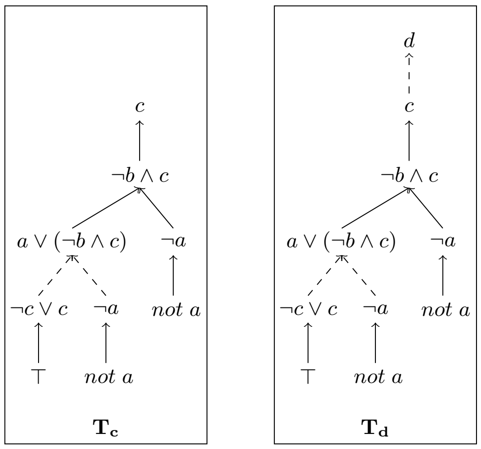

Fig. 2: Two Possible Proof Trees for Program in Example 2

Observe that for each $a \in H_{\Pi}$, we can create a tree whose conclusion is $\neg a$ and whose set of supports is $\{not\ a\}$ (Figure 3) using the rule $\mathbf{A}_a$. Abusing the notation, we will refer to this tree as $\mathbf{A}_a$.

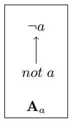

Fig. 3: $\mathbf{A}_a$ – NAF-Tree

Definition 2 (Induced AF). Let Π be a program. The argumentation framework induced by Π, $AF_{\Pi} = (Args_{\Pi}, Atts_{\Pi})$, is defined as follows:

- $Args_{\Pi}$ is the set of all proof trees whose roots are labeled with elements in $H_{\Pi} \cup \{\neg a \mid a \in H_{\Pi}\} \cup \{\top, \bot\}$.
- $Atts_{\Pi} = \{(A, B) \mid \ not\ Concl(A) \in Support(B)\}$, i.e., $A \in Args_{\Pi}$ attacks an argument $B \in Args_{\Pi}$ if and only if $not\ Concl(A) \in Support(B)$.

In general, the set of arguments and the set of attacks of AF_{Π} can be infinite since there are infinitely many formulae that can be constructed from $L_{\Pi} \cup \{\top, \bot\} \cup \{\ not\ a \mid a \in H_{\Pi}\}$ and repetitions are allowed in the construction of proof trees. For instance, if we replace $\neg b \wedge c$, the formula associated with the child of the root of the tree $\mathbf{T}_c$ in Fig. 2, by $(\neg b \wedge c) \wedge (a \vee \neg a)$, then we will receive a new proof tree for c; or, if we replace the child $not\ a$ of $\neg a$ by the tree $\mathbf{A}_a$, then we also receive a new proof tree for c. This property is important from the computational aspect, i.e., any system for computing different semantics of logic programs with FOL-formulae as defined in Definitions 4–5 will need to deal with this problem. We observe that this problem can be dealt with by tightening the definition above, e.g., by considering equivalence between formulae as a single formula or disallowing repetitions. The presence of infinitely many arguments isn AF_Π, however, is irrelevant to the definition of the semantics of argumentation framework, we will keep the above definition as it. Addressing this issue is important but it is outside the scope of this paper and we leave this as a future task.

Observe that not all proof trees are used as arguments in AF_Π. This is because we are only interested in interpretations of Π. We prove some properties of stable extensions of AF_Π.

Proposition 1. *Let Π be a program and AF_Π be the AF induced by Π. For every stable extension S of AF_Π,*

1. *$\top \in Concl(S)$;*
2. *if $\bot \in Concl(S)$ then $\{a, \neg a\} \subseteq Concl(S)$ for every $a \in H_\Pi$;*
3. *if $\bot \in Concl(S)$ then S is the only stable extension of AF_Π; and*
4. *if $\bot \notin Concl(S)$ then for every atom $a \in H_\Pi$, $\{a, \neg a\} \setminus Concl(S)$ is a singleton.*

Proof.

1. The first item is trivial since $\top$ is a single node tree whose support is empty and hence cannot be attacked.
2. If $\bot \in Concl(S)$ then there exists a tree $T \in S$ with $\bot = Concl(T)$. Since $\bot \models \varphi$ for any formula φ, we can easily construct a tree T_φ with $Concl(T_\varphi) = \varphi$ and $Support(T_\varphi) = Support(T)$. Since T is not attacked by S, T_φ is not attacked by S. Hence $T_\varphi \in S$. Restricting φ to literals of the program, we get the conclusion of the proposition.
3. Assume that there exists a stable extension S' of AF_Π and $S' \neq S$. Consider some $T \in S \setminus S'$. S' attacks T implies that $Support(T) \neq \emptyset$. Assume that $not\ \ a \in Support(T)$. Since $\bot \in S$ then from the previous item, we have that S contains a tree T_a such that $a = Concl(T_a)$. It means that T_a attacks T, i.e., S is not conflict free. Contradiction.
4. Assume the contrary. There are two cases:

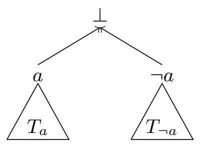

Fig. 4: Tree $T_\bot$

- $\{a, \neg a\} \setminus Concl(S) = \emptyset$. This means that S contains two trees T_a and $T_{\neg a}$ such that $Concl(T_a) = a$ and $Concl(T_{\neg a}) = \neg a$. From these two trees, we can construct a tree $T_\bot$ as in Figure 4. Obviously, this tree is not attacked by S since its set of supports is $Support(T_a) \cup Support(T_{\neg a})$ and S is conflict free by definition. In other words, $T_\bot \in S$ because S is a stable extension, i.e., $\bot \in Concl(S)$. Contradiction.

- $\{a, \neg a\} \setminus Concl(S) = \{a, \neg a\}$. This means that S does not contain any tree T with $Concl(T) = a$. As such, S does not attack the tree $\mathbf{A}_a$. Since S is a stable extension, $\mathbf{A}_a \in S$ which implies that $\neg a \in Concl(S)$. A contradiction with the assumption that $\{a, \neg a\} \setminus Concl(S) = \{a, \neg a\}$. □

The above proposition shows that programs consisting of arguments supporting $\bot$ would have a single stable extension, if one exists. Intuitively, such programs should not be used in making conclusions, especially programs with strict proof trees supporting contradictory conclusions. We characterize this class of programs as *incoherent* as follows.

Definition 3 (Coherent Programs). A program Π is said to be *incoherent* if there are strict arguments supporting both a and $\neg a$ for some $a \in H_\Pi$ in its induced argumentation framework AF_Π.

Π is *coherent* if it is not incoherent.

It is easy to check that $\Pi_3 = \{a \leftarrow \top. \quad \neg a \leftarrow \top.\}$ is incoherent. Coherent programs satisfy the following property.

Proposition 2. *Let Π be a program. It holds that*

1. *If Π is coherent then $Concl(S)$ is consistent for each stable extension S of AF_Π.*
2. *If there is a stable extension S of AF_Π such that $Concl(S)$ is consistent then Π is coherent.*

Proof. The first item follows from the second and fourth items of Proposition 1 and the fact that if Π is incoherent then $\bot \in Concl(S)$ for every stable extension S. The second item follows immediately from the first item. □

Proposition 2 suggests that it is reasonable to focus on coherent programs. In fact, all normal logic programs as defined in [12] are coherent since they do not allow classical negation. As such, *from now on if nothing is explicitly stated, by a program we mean a coherent program*. Having defined the argumentation framework of a program Π, we now define answer sets.

Definition 4 (Argumentation-Based Answer Set). Let Π be a program and $AF_\Pi = (Args_\Pi, Atts_\Pi)$ be the AF induced by Π. An interpretation I of Π is an *argumentation-based answer set* (or *AB-answer set*, for short) of Π if there exists a stable extension S of AF_Π such that $I \cup \neg I^- \cup \{\top\} = Concl(S)$.

The above definition is in line with the recently adopted convention in defining answer sets which requires that answer sets are consistent sets of literals. It also indicates that a program might not have an AB-answer set. In that case we say that the program is inconsistent. Generalizing the result on well-founded model of normal logic programs in [9], we next define well-founded model for programs with FOL formulae.

Definition 5 (Argumentation-Based Well-Founded Model). Let Π be a program and $AF_\Pi = (Args_\Pi, Atts_\Pi)$ be the AF induced by Π. A partial interpretation (P, Q) of Π is the *argumentation-based well-founded model* (or *AB-well-founded model*) of Π if $P \cup \{\neg a \mid a \in Q\} \cup \{\top\} = Concl(S)$ where S is the grounded extension of AF_Π.

Intuitively, if (P, Q) is the well-founded model of Π, then P (Q) is the collection of atoms that are true (false) given Π. It is worth noticing that the notion of a well-founded model for logic programs with first-order formulae has not been defined in the literature.

It follows directly from the property of grounded extension and stable extensions of argumentation framework that the well-founded model is unique and every AB-answer set contains the AB-well-founded model.

Proposition 3. *Let Π be a program. Then, the AB-well-founded model of Π, (P, Q), always exists and is unique; and for every AB-answer set I of Π, $P \subseteq I$ and $Q \subseteq H_\Pi \setminus I$.*

Finally, it is not difficult to derive from the results of Theorems 49 and 50 in [9] that, for normal programs, the above defined AB-answer sets and AB-well-founded model coincide with the classical stable models (as in [12]) and well-founded model (as in [26]). In this sense, this paper lifts the work in [9] to logic programs with FOL-formula.

3.2 Argumentation-Based Answer Sets Are Well-Justified

In this section we will focus on the well-justified provability of elements in an AB-answer set. In the following, we make use of terminologies related to trees such as ancestor, subtree, path, etc. without definitions. For precise definitions, the readers are referred to [7]. A pair of two different nodes N and N' in a tree T is a *cycle* if $\mathit{label}(N) = \mathit{label}(N')$ and N is an ancestor of N'. T is *circular* if it has a cycle; *non-circular* if it does not have a cycle. We prove some properties related to the non-circularity of trees.

Lemma 1. *For each tree T, there exists a non-circular tree T' such that $\mathit{Concl}(T) = \mathit{Concl}(T')$ and $\mathit{Support}(T') \subseteq \mathit{Support}(T)$.*

Proof. Let $k = 0$ and $T_k = T$. If T_k is non-circular then the proposition is proved. Otherwise, T_k has a cycle (N, N'). Clearly, replacing the subtree rooted at N by the subtree rooted at N' in T_k results in a tree T_{k+1} that has at least one cycle less than T_k and $\mathit{Concl}(T_k) = \mathit{Concl}(T_{k+1})$ and $\mathit{Support}(T_k) \subseteq \mathit{Support}(T_{k+1})$. Furthermore, the number of nodes of T_{k+1} is less than the number of nodes of T_k. As such, there exists a finite n such that T_n is non-circular. Clearly, we have that $\mathit{Concl}(T) = \mathit{Concl}(T_n)$ and $\mathit{Support}(T_n) \subseteq \mathit{Support}(T)$. $\square$

Lemma 2. *Let Π be a program and $AF_\Pi = (\mathit{Args}_\Pi, \mathit{Atts}_\Pi)$ be the AF induced by Π. Let S be a stable extension of AF_Π. For each $T \in S$ there exists a non-circular T' in S such that $\mathit{Concl}(T') = \mathit{Concl}(T)$.*

Proof. It follows from Lemma 1 that there exists a non-circular T' such that $\mathit{Concl}(T) = \mathit{Concl}(T')$ and $\mathit{Support}(T') \subseteq \mathit{Support}(T)$. We will show that T' belongs to S. Assume that $T' \notin S$. Since S is a stable extension of AF_Π, S attacks T'. Since $\mathit{Support}(T') \subseteq \mathit{Support}(T)$, S attacks T. Contradiction because S is conflict free and $T \in S$. $\square$.

The above lemmas allow us to prove an important property of AB-answer sets.

Theorem 1. *Let Π be a program and I be an AB-answer set of Π. Then, for each atom $a \in I$, there exists a non-circular proof tree T such that $Concl(T) = a$ and $\{\neg b \mid \ not\ b \in Support(T)\} \subseteq \neg I^-$.*

Proof. I is an argumentation-based answer set of Π iff there exists a stable extension S of AF_Π such that $I \cup \neg I^- \cup \{\top\} = Concl(S)$. This implies that there exists some tree in $T' \in S$ such that $Concl(T') = a$. It follows from Lemma 2 that there exists a non-circular $T \in S$ such that $Concl(T) = Concl(T')$. Since S does not attack T, it means that each $\mathbf{A}_b$. where $\ not\ b \in Support(T)$ belongs to S, i.e., $\{\neg b \mid not\ b \in Support(T)\} \subseteq Concl(S)$. By the definition of I, we can conclude that $\{\neg b \mid \ not\ b \in Support(T)\} \subseteq \neg I^-$. $\Box$.

To prove the non-circularity of AB-answer sets, we need some additional notations. Let T be a tree. The level of a node N, denoted by $l(N)$, in T is defined as follows: (i) if N is a leaf then $l(N) = 0$; (ii) if N is not a leaf then $l(N) = 1 + \max\{l(i) \mid i$ is a child of $N\}$. The level of a tree T is defined by the level of its root and is denoted by $l(T)$. For a program Π and its induced AF_Π, let S be a stable extension of AF_Π. The level of S, denoted by $l(S)$, is defined as $\max\{l(T) \mid T \in S\}$. The *kernel* of S, denoted by $k(S)$, is the collection of trees in S such that $Concl(S) = Concl(k(S))$ and for each $z \in Concl(S)$ there exists a unique $T \in k(S)$ such that $z = Concl(T)$, $l(T) = \min\{l(T') \mid T' \in S, Concl(T') = z\}$.[2] For each stable extension S of AF_Π, let $\langle S_i \rangle_{i=0}^{l(S)}$ be the sequence of literals $S_i = \{Concl(T) \mid T \in k(S), l(T) = i\}$.

Proposition 4. *Let Π be a program and $AF_\Pi = (Args_\Pi, Atts_\Pi)$ be the AF induced by Π. Let S be a stable extension of AF_Π and $0 \leq i \leq l(S)$. Then, for every tree $T \in k(S)$ such that $Concl(T) \in \bigcup_{0 \leq j < i} S_j$, T does not contain any node whose label is $z \in S_t$ for $t \geq i$.*

Proof. Suppose the contrary. Consider $T_z \in k(S)$ such that $Concl(T_z) = z$. It implies that T contains a subtree T' such that $Concl(T') = z$. It is easy to see that $T' \in S$. Since $l(T') < l(T) < i = l(T_z)$, it means that $T_z \notin k(S)$. This is a contradiction. $\Box$

A consequence of the above proposition is that each tree with level i in the kernel of a stable extension of AF_Π can be constructed using only literals that were the conclusions of trees whose levels are smaller than i. This demonstrates that AB-answer sets are indeed well-justified. Theorem 2 in Subsection 5.1 formalizes this result in precise terms.

4 Programs With FOL-Representable Atoms

Since the construction of a proof tree makes use of logical inference (first case of Item 2 of Definition 1), it is easy to see that AB-answer sets are preserved under equivalent transformations. This stipulates that the notion of AB-answer sets can be extended to allow extended atoms whose truth values can be defined via a formula in the language of the given program. In fact this is true for many well-known extensions of logic programs

[2] Intuitively, a tree T belongs to the kernel of S if T belongs to S and the level of T is minimal wrt trees in S supporting the same conclusion.

such as weight constraint atoms (e.g., [19]), aggregates (e.g., [10]), or abstract constraint atoms (e.g., [17]). Let us quickly review some basic notions of these types of extensions of logic programming.

- A weight constraint atoms [19] of a program Π is of the form $l\ [p_1 : q_1 = w_1, \ldots, p_n : q_n = w_n]\ u$ where l and u are two real numbers such that $l \leq u$, p_i is either a or $not\ \ a$ for some atom $a \in H_\Pi$, q_i is an atom in H_Π, and w_i is a real number. $p_i : q_i$ is called a *conditional literal*. Given a weight constraint atom C and an interpretation I, the weight of the formula $[p_1 : q_1 = w_1, \ldots, p_n : q_n = w_n]$, denoted by $W(C)$, is calculated[3] and C is declared to be true w.r.t. I if $l \leq W(C) \leq u$.
- An aggregate atom (e.g, [10]) of a program Π is of the form $f(S) \prec T$ where T is a term, $\prec \in \{=, <, >, \leq, \geq\}$, and $f(S)$ is an aggregate term with f is an aggregate function symbol ($\#\texttt{count}, \#\texttt{sum}, \ldots$) and S is a set term. In fact, following this syntax the aggregate atom in Π_1 is written as $\#\texttt{sum}\{X : p(X)\} \geq 1$. Given an aggregate atom $f(S) \prec T$ and an interpretation I, the set term S is evaluated and the value for $f(S)$ is calculated; the atom is true w.r.t. T if the evaluation of $f(S) \prec T$ returns true.
- An abstract constraint atom (e.g., [17, 25]) of a program Π is of the form (D, C) where C is a set of subsets of H_Π and D is a subset of H_Π. (D, C) is true w.r.t. an interpretation I if $I \cap D \in C$.

In short, each of these extensions to logic program starts by defining the syntax of a new type of atoms and associating a method for evaluating the truth value of a new atom w.r.t. an interpretation. Usually, this method allows for the identification of all interpretations of the program that satisfies the atom, i.e., each atom a is associated with a set $\mathcal{I}_a$ of interpretations satisfying a. As such, we can identify a with $t_a = \bigvee_{I \in \mathcal{I}_a} (\bigwedge_{b \in I} b \wedge \bigwedge_{b \notin I} \neg b)$ and $\neg a$ with $\neg t_a$. As an example, the aggregate atom $\lambda = \textsc{Sum}(\{X : p(X)\}) \geq 1$ in Example 1 is associated to the formula $t_\lambda = \bigvee_{i=1}^{5} (I_i \wedge \neg I_i^-)$ where $I_1, \ldots, I_5$ are the five interpretations satisfying λ detailed in Section 1.

The method of evaluation of extended atoms is then used to define when a rule (or a program with extended syntax) is satisfied given an interpretation which, in turn, is used in defining answer sets although the approach to define the semantics of programs with these new types of atoms might be different (e.g., by using a two step definition similar to the original definition of answer sets [10], or using a mathematical operator [20, 17], etc.). The key distinction between previous approaches in dealing with these extensions lies in the requirement whether or not t_a needs to be proved or can be assumed. For examples, the approaches in [20, 25, 23] seem to require that the formulae related to the extended atoms are provable; on the other hand, the approaches in [10, 17] seem to allow they to be assumed.

It is easy to recognize that the approach developed in the previous section requires provability of the conclusions. As it turns out, it can be easily adapted to any of the proposed extensions of logic programs. Instead of generalizing the approach for each extension separately, we next propose a generalization of the AB-semantics which can

[3] Precise formula for computing $W(C)$ is not really important for the discussion. It can be found in [19].

be instantiated to each of the above discussed extensions of logic programs. Given a signature $\Sigma = (\mathcal{C}, \mathcal{P}, \mathcal{F})$. An expression of the form $\gamma[\alpha]$ where γ is a 0-ary predicate symbol that does not occur in the language L_Σ and $\alpha \in L_\Sigma$ is called a *FOL-representable atom* w.r.t. Σ. Intuitively, the formula α is a definition of γ in the language over Σ. For instance, $(\textsc{Sum}(\{X : p(X)\}) \geq 1)[t_\lambda]$ is a FOL-representable atom over the language of Π_1.

Definition 6. *Let $\Sigma = (\mathcal{C}, \mathcal{P}, \mathcal{F})$ be a signature and Γ be a set of FOL-representable atoms w.r.t. Σ. A* program with FOL-representable atoms *over (Σ, Γ) is a set of rules of the form $\psi \leftarrow \phi$ where $\psi \in L_\Sigma$ and ϕ is a formula in the language over the signature $\Sigma' = (\mathcal{C}, \mathcal{P} \cup \{\gamma \mid \gamma \in \Gamma\}, \mathcal{F})$.*

We will now extend the notion of AB-semantics to programs with FOL-representable atoms.

Definition 7. *Let Π be a program with FOL-representable atoms over (Σ, Γ) and Π' be Π extended with the set of rules $\{\gamma \leftarrow \alpha \mid \gamma[\alpha] \in \Gamma\} \cup \{\neg\gamma \leftarrow \neg\alpha \mid \gamma[\alpha] \in \Gamma\}$. Let $\Lambda = \{\lambda \mid \lambda[\alpha] \in \Gamma\}$. We define (*i*) an interpretation I of Π is an AB-answer set of Π iff there exists some $\Lambda_p \subseteq \Lambda$ such that $I \cup \Lambda_p$ is an AB-answer set of Π'; and (*ii*) a partial interpretation (P, Q) of Π is the AB-well-founded model of Π iff there exists some $\Lambda_p, \Lambda_n \subseteq \Lambda$ such that $\Lambda_p \cap \Lambda_n = \emptyset$ and $(P \cup \Lambda_p, Q \cup \Lambda_n)$ is the AB-well-founded model of Π'.*

Intuitively, the semantics of programs with FOL-representable atoms is defined by transforming them to programs with FOL-formulae without extended features. By adding $\gamma \leftarrow \alpha$ and $\neg\gamma \leftarrow \neg\alpha$ to the original program, we essentially add rules to the construction of proof trees which allow for the derivation of FOL-representable atoms. The former (latter) rule allows for the conclusion of γ ($\neg\gamma$). The usefulness of this definition is illustrated in the next section.

5 Related Work and Discussion

The paper relates to works that extend the answer set semantics to more generalized logic programs and that study semantics of logic programs using argumentation. The main distinctions between our approach and previous approaches to defining semantics of logic programs using argumentation such as [5, 9, 13, 22] lie in our focus on generalized logic programs and the explicit use of the notion of a proof tree. Due to the space limitation, we will focus our discussion on the properties of the AB-semantics. Specifically, we show that AB-answer set semantics (resp. AB-well-founded model) is equivalent to well-justified answer set semantics for programs with FOL-formula [23] (resp. well-founded model for programs with abstract constraint atoms [27]). We note that these results, together with the results in [23, 27], show that the argumentation-based answer set (Definition 4) is equivalent to a number of previously defined semantics for various extensions of logic programs such as aggregates, description logics, or abstract constraint atoms.

5.1 Well-Justified Answer Sets for Logic Programs with FOL-formulae

The relationship between well-justified answer sets and AB-answer sets is proved in the next theorem.

Theorem 2. *Let Π be a program with FOL formulae. I is a well-justified answer set of Π iff I is an argumentation-based answer set of Π.*

Proof. Let I be a well-justified answer set of Π. Let S be the set of trees in AF_Π such that for every $T \in S$, $Support(T) \subseteq \{not\ b \mid b \notin I\}$. Because I is an answer set of Π, we can show that S is conflict free. Furthermore, for every $a \in I$, there exists a tree $T \in S$ such that $Concl(T) = a$. This also implies that S attacks every T' such that $Support(T') \setminus \{not\ b \mid b \notin I\} \neq \emptyset$, i.e., S attacks every T' does not belonging to it. Hence, S is a stable extension of AF_Π. This implies that I is an argumentation-based answer set of Π.

Let I be an argumentation-based answer set of Π and S be the stable extension of AF_Π. For a tree T and a rule $r \in ground(\Pi)$, r is applicable in T if r is used in the construction of T (in the second condition of Item 2, Definition 1). It is easy to see that $f\Pi^I$ is the set of rules in $ground(\Pi)$ applicable in S. By induction over the levels of trees in the kernel of S and Proposition 4, we can show that $lfp(T_{f\Pi^I}(\emptyset, \neg I^-)) \cup \neg I^- \models a$ for every $a \in I$. It means that I is a well-justified answer set of Π. $\square$

5.2 Well-Founded Semantics for Programs with Abstract Constraint Atoms

Well-Founded Semantics for programs with abstract constraint atoms [27] is defined for programs consisting of rules of the form

$$a \leftarrow A_1, \dots, A_k, \neg A_{k+1}, \dots, \neg A_m$$

where a is an atom and A_i's are abstract constraint atoms, each is of the form (D, C), D is a set of atoms and $C \subseteq 2^D$. Let S, J be two sets of atoms such that $S \cap J = \emptyset$. The S-prefixed power set $S \uplus J$ is $\{S' \mid S \subseteq S' \subseteq S \cup J\}$. $S \uplus J$ is maximal in an abstract constraint atom $A = (D, C)$ if $S \uplus J \subseteq C$ and there exists no $S' \uplus J' \subseteq C$ such that $S \uplus J \subset S' \uplus J'$. It has been shown that each $A = (D, C)$ can be represented by an $A' = (D, C^*)$ such that C^* is a set of maximal prefixed power sets in A.

Let $X = (P, Q)$ be a partial interpretation of Π and $A = (D, C^*)$ be an abstract constraint atom in abstract representation. X satisfies A, written $X \models A$, if for some $S \uplus J \in C^*$, $S \subseteq P$ and $D \setminus (S \cup J) \subseteq Q$. X falsifies A, written $X \Vdash A$, if for every $S \uplus J \in C^*$, $S \cap Q \neq \emptyset$ or $D \setminus (S \cup J) \cap Q \neq \emptyset$. X satisfies (resp. falsifies) $\neg A$ if X falsifies (resp. satisfies) A. X satisfies (resp. falsifies) a set Z if it satisfies (resp. falsifies) every member of Z. A set of atoms U is an unfounded set of Π with respect to X iff, for any $a \in U$ and any $r \in ground(P)$ with $head(r) = a$, either (*i*) for some $\neg A \in body(r)$, $X \Vdash \neg A$; or (*ii*) for some $A = (D, C^*) \in body(r)$, for any $S \uplus J \in C^*$, either $U \cap S \neq \emptyset$ or $X \Vdash S \cup \{\neg z \mid z \in (D \setminus (S \cup J))\}$. For a program Π, we define

$$T_{\Pi}(X) = \{head(r) \mid r \in \Pi,\ X \text{ satisfies } body(r)\}$$
$$U_{\Pi}(X) = \text{the greatest unfounded set of } \Pi \text{ w.r.t. } X$$
$$W_{\Pi}(X) = (T_{\Pi}(X), H_{\Pi} \setminus U_{\Pi}(X))$$

$\mathit{lfp}(W_{\Pi})$ is defined as the well-founded model of Π.

A program with abstract constraint atoms Π can be viewed as a program with FOL-representable atoms Π^* where each abstract constraint atom $A = (D, C^*)$ is replaced by λ_A whose FOL-representation is $\lambda_A[\alpha(D, C^*)]$ and $\alpha(D, C^*) = \bigvee_{S \uplus J \in C}(\bigwedge_{p \in S} p \wedge \bigwedge_{n \in D \setminus (S \cup J)} \neg n)$. Let $\Lambda(\Pi) = \{\lambda_{A[\alpha(D,C^*)]} \mid A = (D, C^*)$ is an abstract constraint atom in $\Pi\}$. The relationship between the well-founded model of Π and the AB-well-founded model of Π^* is proved in the next theorem.

Theorem 3. *For a program with abstract constraint atoms Π, (P, Q) is its well-founded model iff there exists a pair (A_p, A_n) such that $A_p, A_n \subseteq \Lambda(\Pi)$, $A_p \cap A_n = \emptyset$, and $(P \cup A_p, Q \cup A_n)$ is the AB-well-founded model of Π^*. In addition, $(P, Q) \models \alpha(D, C^*)$ for every $\lambda_{A[\alpha(D,C^*)]} \in A_p$ and $(P, Q) \models \neg\alpha(D, C^*)$ for every $\lambda_{A[\alpha(D,C^*)]} \in A_n$.*

Proof. (Sketch) Let AF_{Π^*} be the argumentation framework induced by the program Π^*. Let $G^i = \{T \mid T \in \mathit{lfp}(F_{AF_{\Pi^*}}),\ l(T) = i\}$. We can prove by induction over i that (*i*) if $a \in H_{\Pi}$ then $a \in W^i_{\Pi}(\emptyset)$ iff there exists a tree $T_a \in G^i$ such that $Concl(T_a) = a$; (*ii*) for every $A = (D, C^*)$, $W^i_{\Pi}(\emptyset) \models A$ iff there exists a tree $T_{\lambda_{A[\alpha(D,C^*)]}} \in G^i$ such that $Concl(T_{\lambda_{A[\alpha(D,C^*)]}}) = \lambda_A$; and (*iii*) for every $A = (D, C^*)$, $W^i_{\Pi}(\emptyset) \models \neg A$ iff there exists a tree $T_{\neg\lambda_{A[\alpha(D,C^*)]}} \in G^i$ such that $Concl(T_{\lambda_{A[\alpha(D,C^*)]}}) = \neg\lambda_A$. This proves the theorem. $\square$

6 Conclusions

In this paper, we defined different argumentation-based semantics for programs with FOL-formulae. The key idea behinds our approach lies in the notion of a proof tree. We proved that the proposed semantics captured the well-justified semantics for programs with FOL-formulae defined in [23] and the well-founded semantics for programs with abstract constraint atoms defined in [27]. To the best of our knowledge, this is the first proposal of argumentation-based semantics for generalized programs that exhibits this equivalence. The proposed framework also sheds new light on the answer set semantics for various extensions of logic programs in the literature by showing that they can be viewed as programs with FOL-formulae extended with FOL-representable atoms, indirectly providing a means for comparison among approaches to defining semantics of logic programs with extensions. Finally, we note that our focuses in this paper is on the two most well-known semantics of logic programs as well as the most recently introduced semantics. Due to the results in [9, 22], we expect that similar results for 3-valued stable models and complete extensions can be established. In addition, the results established in Theorems 2-3 and the results in [23, 27] show that equivalence between AB-defined semantics and other approaches in the literature such as [20, 24] can also be established. This will be reported in the future extended version of the paper.

Finally, we note that the paper focuses on the development of argumentation-based semantics for logic programs with FOL-formulae. It does not address question relating to the computation of the semantics. For example, how to construct arguments (or proof-trees) of a program?; is the set of arguments always infinite?; etc. We leave this for the future work.

References

[1] M. Balduccini, M. Gelfond, R. Watson, and M. Nogueira. The USA-Advisor: a case study in answer set planning. In *Lectures Notes in Artificial Intelligence (Proceedings of the Sixth International Conference on Logic Programming and Nonmonotonic Reasoning, LP-NMR'01)*, volume 2173, pages 439–442. Springer-Verlag, 2001.

[2] C. Baral. *Knowledge Representation, reasoning, and declarative problem solving with Answer sets*. Cambridge University Press, Cambridge, MA, 2003.

[3] C. Baral, A. Provetti, and T. C. Son, editors. *Theory and Practice of Logic Programming Special Issue on Answer Set Programming*, volume 3. Cambridge Univeristy Press, 2003.

[4] M. Bartholomew, J. Lee, and Y. Meng. First-order extension of the FLP stable model semantics via modified circumscription. In *IJCAI 2011*, pages 724–730. IJCAI/AAAI, 2011.

[5] A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artif. Intell.*, 93:63–101, 1997.

[6] B. Chisham, E. Pontelli, T. C. Son, and B. Wright. Cdaostore: A phylogenetic repository using logic programming and web services. In J. P. Gallagher and M. Gelfond, editors, *Technical Communications*, ICLP 2011, pages 209–219.

[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, 2nd Edition*. MIT Press, Cambridge, MA, 2001.

[8] P. M. Dung. An Argumentation-Theoretic Foundations for Logic Programming. *J. Log. Program.*, 22:2, 151–171, 1995.

[9] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321 – 357, 1995.

[10] W. Faber, N. Leone, and G. Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In JELIA 2004, pages 200–212. Springer, 2004.

[11] M. Gebser, C. Guziolowski, M. Ivanchev, T. Schaub, A. Siegel, S. Thiele, and P. Veber. Repair and prediction (under inconsistency) in large biological networks with answer set programming. In KR'10, pages 497–507. AAAI Press, 2010.

[12] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In LPICS, pages 1070–1080, 1988.

[13] A. Kakas and P. Mancarella. Generalized stable models: a semantics for abduction. In *Proceedings of ECAI-90*, pages 385–391, 1990.

[14] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1–2):39–54, 2002.

[15] J. Lloyd. *Foundations of logic programming*. Springer Verlag, 1987. Second, extended edition.

[16] V. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-year Perspective*, pages 375–398, 1999.

[17] V. W. Marek, I. Niemelä, and M. Truszczynski. Logic programs with monotone abstract constraint atoms. *TPLP*, 8(2):167–199, 2008.

[18] I. Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.

[19] I. Niemelä, P. Simons, and T. Soininen. Stable model semantics for weight constraint rules. In *LPNMR*, pages 315–332, 1999.

[20] N. Pelov, M. Denecker, and M. Bruynooghe. Partial stable models for logic programs with aggregates. In *LPNMR*, pages 207–219. Springer, 2004.

[21] T. Przymusinski. Stable semantics for disjunctive programs. *New generation computing*, 9(3,4):401–425, 1991.

[22] C. Schulz and F. Toni. Logic programming in assumption-based argumentation revisited - semantics and graphical representation. In AAAI 2015, pages 1569–1575. AAAI Press, 2015.

[23] Y. Shen, K. Wang, T. Eiter, M. Fink, C. Redl, T. Krennwallner, and J. Deng. FLP answer set semantics without circular justifications for general logic programs. *Artif. Intell.*, 213:1–41, 2014.

[24] T. C. Son and E. Pontelli. A Constructive Semantic Characterization of Aggregates in Answer Set Programming. *Theory and Practice of Logic Programming*, 7(03):355–375, 2007.

[25] T. C. Son, E. Pontelli, and P. H. Tu. Answer Sets for Logic Programs with Arbitrary Abstract Constraint Atoms. In *AAAI*, 2006.

[26] A. Van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650, 1991.

[27] Y. Wang, F. Lin, M. Zhang, and J. You. A well-founded semantics for basic logic programs with arbitrary abstract constraint atoms. In AAAI 2012.