Extension-Based Argumentation Semantics via Logic Programming Semantics with Negation as Failure

Juan Carlos Nieves and Ignasi Gomez-Sebastià

Universitat Politècnica de Catalunya
Software Department (LSI)
c/Jordi Girona 1-3, E08034, Barcelona, Spain
{jcnieves,igomez}@lsi.upc.edu

Abstract. Extension-based argumentation semantics have been shown to be a suitable approach for performing practical reasoning. Since extension-based argumentation semantics were formalized in terms of relationships between atomic arguments, it has been shown that extension-based argumentation semantics (such as the grounded semantics and stable semantics) can be characterized by logic programming semantics with negation as failure. Recently, it has been shown that argumentation semantics such as the preferred semantics and the CF2 semantics can be characterized in terms of logic programming semantics. In this paper, we make a short overview *w.r.t.* recent results in the close relationship between extension-based semantics and logic programming semantics with negation as failure. We also show that there is enough evidence to believe that the use of declarative approaches based on logic programming semantics with negation as failure is a practical approach for performing practical reasoning following an argumentation reasoning approach.

1 Introduction

Argumentation theory is a formal discipline within Artificial Intelligence (AI) where the aim is to make a computer assist in or perform the act of argumentation. During the last years, argumentation has been gaining increasing importance in Multi-Agent Systems (MAS), mainly as a vehicle for facilitating *rational interaction* (*i.e.*interaction which involves the giving and receiving of reasons) [4, 15]. A single agent may also use argumentation techniques to perform its individual reasoning because it needs to make decisions under complex preferences policies, in a highly dynamic environment.

Argumentation theory is also regarded as an approach of non-monotonic reasoning since it formalizes non-monotonic reasoning as the construction and comparison of arguments for and against certain conclusions. Non-monotonicity arises from the fact that new information may give rise to new counterarguments that defeat the original argument. Inference in argumentation models is defined relative to a set of arguments and a binary attack relation between them. Typically, they classify arguments in three classes: the *acceptable* or justified arguments, the *defeated* or overruled arguments, and the *ties*, *i.e.*, the arguments, which are involved in an irresolvable conflict.

Although several approaches have been proposed for capturing representative patterns of inference in argumentation theory, Dung's approach, presented in [7], is a unifying framework which has played an influential role on argumentation research and AI.

The kernel of Dung's framework is supported by four extension-based argumentation semantics (we will refer to them also as abstract argumentation semantics): *grounded semantics*, *stable semantics*, *preferred semantics*, and *complete semantics*. When Dung introduced his argumentation approach, he proved that it can be regarded as a special form of logic programming with *negation as failure*. In fact, he showed that the grounded and stable semantics can be characterized by the well-founded and stable model semantics respectively. This result has at least two main implications:

1. It defines a general method for generating metainterpreters for argumentation systems
2. It defines a general method for studying abstract argumentation semantics' properties in terms of logic programming semantics' properties.

As we can see, the study of abstract argumentation semantics in terms of logic programming semantics has important implications for closing the gap between theoretical results and argumentation systems. Despite these implications, until 2007 the only extension-based argumentation semantics characterized in terms of logic programming semantics were the grounded and stable semantics. Recently, there are novel results which have shown that extension-based argumentation semantics (such as the preferred) semantics can be characterized in terms of logic programming semantics [5, 11]. In fact, there are results which show that not only a logic programming approach can characterize extension-based argumentation semantics but also it can define new extension based argumentation semantics [12].

In this paper, we present a survey of recent results *w.r.t.* the close relationship between extension-based semantics and logic programming semantics with negation as failure. These results are in two directions:

1. The identification of dual characterizations between extension-based argumentation semantics and logic programming semantics. These results represent an extension of Theorem 17 presented in [7]. These results are based on regarding argumentation frameworks into logic programs.
2. The implementation of logic programming metainterpreters for inferring extension-based argumentation semantics. These metainterpreters are based on the platform of Answer Set Programming (ASP)[1] and a characterization of extension-based argumentation semantics in terms of labels (IN, OUT, UNDEC). This labeling process has shown to be a practical approach for defining interactive algorithms for inferring extension-based argumentation semantics [10]. However, there are results that show that the declarative specification of a labeling process defines an easy approach for computing extension-based argumentation semantics [17, 16].

The rest of the paper is divided as follows: In §2, the syntax of valid logic programs in ASP is introduced, some basic concepts of the Dung's argumentation approach are defined and the characterization of extension-based argumentation semantics in terms of the labels: IN, OUT, UNDEC is presented. In §3, some recent results *w.r.t.* the relationship between extension-based argumentation semantics and logic programming

[1] Answer Set Programming is a novel approach of logic programming which was suggested by the non-monotonic reasoning community [1].

semantics are presented. In §4, some approaches for implementing logic programming metainterpreters for inferring extension-based argumentation semantics are presented. Finally, in the last section some conclusions are presented.

2 Background

In this section, we present some basic concepts on: a) the syntax of logic programs b) extension-based argumentation semantics c) a labeling approach for characterizing extension-based argumentation semantics.

2.1 Syntax of Logic Programs

A signature $\mathcal{L}$ is a finite set of elements that we call atoms. A literal is an atom, a (positive literal), or the negation of an atom $not\ a$ (negative literal). Given a set of atoms $\{a_1, \ldots, a_n\}$, $not\ \{a_1, \ldots, a_n\}$ denotes the set of literals $\{not\ a_1, \ldots, not\ a_n\}$. A disjunctive clause is a clause of the form:

$$a_1 \vee \cdots \vee a_m \leftarrow a_{m+1}, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_n$$

where a_i is an atom, $1 \leq i \leq n$. When $n = m$ and $m > 0$, the disjunctive clause is an abbreviation of the fact $a_1 \vee \cdots \vee a_m \leftarrow \top$ where $\top$ is an atom that always evaluate to true. When $m = 0$ and $n > 0$ the clause is an abbreviation of $\bot \leftarrow a_1, \ldots, a_j, not\ \ a_{j+1}, \ldots, not\ \ a_n$where $\bot$ is an atom that always evaluate to false. Clauses of this form are called constraints (the rest non-constraints). A disjunctive logic program is a finite set of disjunctive clauses.

Sometimes, we denote a clause C by $\mathcal{A} \leftarrow \mathcal{B}^+, \neg\mathcal{B}^-$, where $\mathcal{A}$ contains all the head atoms, $\mathcal{B}^+$ contains all the positive body literals and $\mathcal{B}^-$ contains all the negative body literals. We also use $body(C)$ to denote $\mathcal{B}^+, \neg\mathcal{B}^-$. When $\mathcal{B}^- = \emptyset$, the clause C is called a positive disjunctive clause. A set of positive disjunctive clauses is called positive disjunctive logic program. When $\mathcal{A}$ is a singleton set, the clause can be regarded as a normal clause. A normal program is a finite set of normal clauses. Also, when $\mathcal{A}$ is a singleton set and $\mathcal{B}^- = \emptyset$, the clause can be regarded as a definite clause. A finite set of definite clauses is called a definite logic program.

We denote by $\mathcal{L}_P$ the signature of P, *i.e.*, the set of atoms that occur in P. Given a signature $\mathcal{L}$, we write $Prog_{\mathcal{L}}$ to denote the set of all the programs defined over $\mathcal{L}$.

A signature $\mathcal{L}$ is a finite set of elements that we call atoms. A *literal* is either an atom a, called *positive literal*; or the negation of an atom $\neg a$, called *negative literal*. Given a set of atoms $\{a_1, ..., a_n\}$, we write $not\ \{a_1, ..., a_n\}$ to denote the set of atoms $\{not\ a_1, \ldots, not\ a_n\}$. A *normal* clause, C, is a clause of the form

$$a \leftarrow b_1, \ldots, b_n,\ not\ b_{n+1}, \ldots,\ not\ b_{n+m}$$

where a and each of the b_i are atoms for $1 \leq i \leq n + m$. In a slight abuse of notation we will denote such a clause by the formula $a \leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^-$ where the set $\{b_1, \ldots, b_n\}$ will be denoted by $\mathcal{B}^+$, and the set $\{b_{n+1}, \ldots, b_{n+m}\}$ will be denoted by $\mathcal{B}^-$. We define a *normal program* P, as a finite set of normal clauses.

If the body of a normal clause is empty, then the clause is known as a *fact* and can be denoted just by: $a \leftarrow \top$ where $\top$ denotes a atom which always evaluate to true. We also use $body(C)$ to denote $\mathcal{B}^+ \cup \neg\mathcal{B}^-$.

We write $\mathcal{L}_P$, to denote the set of atoms that appear in the clauses of P. We denote by $HEAD(P)$ the set $\{a | a \leftarrow \mathcal{B}^+,\ \neg\mathcal{B}^- \in P\}$.

2.2 Extension-based Argumentation Semantics

We are going to present a short introduction of Dung's argumentation approach. A fundamental Dung's definition is the concept called argumentation framework which is defined as follows:

Definition 1. *[7] An argumentation framework is a pair $AF = \langle AR, attacks\rangle$, where* AR *is a set of arguments, and* attacks *is a binary relation on* AR*,* i.e.attacks $\subseteq AR \times AR$*.*

Following Dung's reading, we say that A attacks B (or B is attacked by A) if $attacks(A, B)$ holds. Similarly, we say that a set S of arguments attacks B (or B is attacked by S) if B is attacked by an argument in S.

Definition 2. *[7] A set S of arguments is said to be conflict-free if there are no arguments A, B in S such that A attacks B.*

Definition 3. *[7] (1) An argument $A \in AR$ is acceptable with respect to a set S of arguments if and only if for each argument $B \in AR$: If B attacks A then B is attacked by S. (2) A conflict-free set of arguments S is admissible if and only if each argument in S is acceptable w.r.t. S.*

The (credulous) semantics of an argumentation framework is defined by the notion of *preferred extensions*.

Definition 4. *[7] A preferred extension of an argumentation framework AF is a maximal (w.r.t. inclusion) admissible set of AF.*

Another relevant semantics that Dung introduced is the stable semantics of an argumentation framework which is based in the notion of stable extension.

Definition 5. *[7] A conflict-free set of arguments S is called a stable extension if and only if S attacks each argument which does not belong to S.*

Dung proved an important relationship between preferred extensions and stable extensions.

Lemma 1. *[7] Every stable extension is a preferred extension, but no vice versa.*

Dung also defined a skeptical semantics which is called grounded semantics and it is defined in terms of a *characteristic function*.

Definition 6. *[7] The characteristic function, denoted by F_{AF}, of an argumentation framework $AF = \langle AR, attacks\rangle$ is defined as follows:*
$F_{AF} : 2^{AR} \rightarrow 2^{AR}$
$F_{AF}(S) = \{A |$ *A is acceptable w.r.t. S* $\}$

Definition 7. *[7] The grounded extension of an argumentation framework AF, denoted by GE_{AF}, is the least fixed point of F_{AF}*

Dung defined the concept of complete extension which provides the link between preferred extensions (credulous semantics), and grounded extension (skeptical semantics).

Definition 8. *[7] An admissible set S of arguments is called complete extension if and only if each argument which is acceptable w.r.t. S, belongs to S.*

In [7], a general method for generating metainterpreters in terms of logic programming for argumentation systems was suggested. This is the first approach which regards an argumentation framework as a logic program. This metainterpreter is divided in two units: Argument Generation Unit (AGU), and Argument Processing Unit (APU). The AGU is basically the representation of the attacks upon an argumentation framework and the APU consists of two clauses. In order to define these clauses, let us introduce the predicate *d(x)*, where the intended meaning of *d(x)* is: "the argument *x* is *defeated*" and the predicate *acc(x)*, where the intended meaning of $acc(x)$ is: "the argument *x* is acceptable".

(C1) $acc(x) \leftarrow not\ d(x)$
(C2) $d(x) \leftarrow attack(y, x), acc(y)$

The first one (C1) suggests that the argument x is acceptable if it is not defeated and the second one (C2) suggests that an argument is defeated if it is attacked by an acceptable argument. Formally, the Dung's metainterpreter is defined as follows:

Definition 9. *Given an argumentation framework $AF = \langle AR, attacks \rangle$, P_{AF} denotes the logic program defined by $P_{AF} = APU + AGU$ where $APU = \{C1, C2\}$ and $AGU = \{attacks(a, b) \leftarrow \top | (a, b) \in attacks\}$*

For each extension E of AF, $m(E)$ is defined as follows:
$m(E) = AGU \cup \{acc(a) | a \in E\} \cup \{d(b) | b$ is attacked by some $a \in E\}$
Based on P_{AF}, Dung was able to characterize the stable semantics and the grounded semantics.

Theorem 1. *Let AF be an argumentation framework and E be an extension of AF. Then*

1. *E is a stable extension of AF if and only if $m(E)$ is an answer set of P_{AF}*
2. *E is a grounded extension of AF if and only if $m(E) \cup \{not\ defeat(a) | a \in E\}$ is the well-founded model of P_{AF}*

This result is really important in argumentation semantics, in fact it has at least two main implications:

1. It defines a general method for generating metainterpreters for argumentation systems and
2. It defines a general method for studying abstract argumentation semantics' properties in terms of logic programming semantics' properties.

As we can see, the study of abstract argumentation semantics in terms of logic programming semantics has important implications. However, Dung only characterized the grounded semantics and the stable semantics in terms of logic programming semantics.

In fact, we will see that by regarding an argumentation framework as a logic program, we can also define new argumentation semantics.

2.3 Extension-based Argumentation Semantics via Labeling

The Dung approach presented in the previous section has given rise to an extensive body research [4]. An one of most active research issue is dialogue process for deciding acceptability of arguments. A common approach for deciding acceptability of arguments is by considering a *labeling* process. The labeling approach is a suitable approach for characterizing argumentation semantics [9, 10].

The labeling process is based in a labeling mapping. In this paper, we are going to consider the mapping presented in [10] which considers three labels: For instance the labeling mapping defined in [10] is just a set of labels (one per argument in the framework) where each label can represent three different values: *in*, *out* and *undec*. This mapping is defined as follows: Given an argumentation framework $AF = \langle AR, attacks \rangle$, for any argument $a \in AR$ one can define functions $IN(a)$, $OUT(a)$ and $UNDEC(a)$ that return true if the argument a is labeled in, out and $undec$ respectively, and false otherwise. Also, the sets IN, OUT and $UNDEC$ can be defined as the sets containing all the arguments in the framework labeled in, out and $undec$ respectively.

A special type of labeling is the *reinstatement labeling*. A reinstatement labeling is a labeling that, given an Argumentation framework:

$$AF = \langle AR, attacks \rangle \textit{ and } attacks(a, b)\ iff\ (a, b) \in attacks$$

meets the following two properties:

$$\forall a \in AR : OUT(a) = true\ iff\ \exists b \in AR : attacks(b, a) \wedge IN(b) = true$$

$$\forall a \in AR : IN(a) = true\ iff\ \forall b \in AR : attacks(b, a) \textit{ then } OUT(b) = true$$

Some authors have shown that by considering a mapping process one can characterize the extension-based argumentation semantics defined by Dung in [7] and also some other new argumentation semantics that have been defined by the argumentation community. Some of these characterizations are [10]:

Caminada's work introduces the following mapping between labeling constraints and argumentation frameworks:

- Complete extensions are equal to reinstatement labellings.
- Grounded extensions are equal to reinstatement labellings, where the set IN is minimal. That is, there is no other possible reinstatement labeling in the same Argumentation Framework with a set IN' such that $IN' \subseteq IN$.
- Preferred extensions are equal to reinstatement labellings, where the set IN is maximal. That is, there is no other possible reinstatement labeling in the same Argumentation Framework with a set IN' such that $IN \subseteq IN'$.

– Stable extensions are equal to reinstatement labellings, where the set $UNDEC$ is empty. That is, $\forall a \in AR : a \notin UNDEC$.
– Semi-stable extensions are equal to reinstatement labellings, that have a minimal $UNDEC$ set. That is, there is no other possible reinstatement labeling in the same Argumentation Framework with a set $UNDEC'$ such that: $UNDEC' \in UNDEC$.

In the following sections, we are going to present some recent results which extend Theorem 1 and its implications.

3 Extension-Based Semantics as Logic Programming Semantics with Negation as Failure

In this section, we are going to introduce some recent results which extend Theorem 1. For this aim, we are going to introduce some simple mappings in order to regard argumentation frameworks in terms of logic programs.

Definition 10. *Let $AF = \langle AR, attacks \rangle$ be an argumentation framework, then P^1_{AF}, P^2_{AF} and P^3_{AF} are defined as follows:*

$$P^1_{AF} = \bigcup_{a \in AR} \{ \bigcup_{b:(b,a) \in attacks} \{d(a) \leftarrow not\ d(b)\}$$

$$P^2_{AF} = \bigcup_{a \in AR} \{ \bigcup_{b:(b,a) \in attacks} \{d(a) \leftarrow \bigwedge_{c:(c,b) \in attacks} d(c)\}\}$$

$$P^3_{AF} = \bigcup_{a \in AR} \{acc(a) \leftarrow not\ d(a)\}$$

The intuitive ideas behind these mappings are:

– P^1_{AF} suggests that an argument a will be defeated if one of its adversaries is not defeated. It is not hard to see that P^1_{AF} is equivalent to P_{AF} (introduced in Definition 9).
– P^2_{AF} suggests that the argument a is defeated when all the arguments that defend[2] a are defeated.
– P^3_{AF} suggest that any arguments which is not defeated is accepted.

The conditions captured by P^1_{AF} and P^2_{AF} are standard settings in argumentation theory for expressing the acceptability of an argument. In fact, these conditions are also standard setting in Defeasible Logic [8].

In order to present some relevant properties of the defined mappings, let us define the function tr as follows: Given a set of arguments E, $tr(E)$ is defined as follows: $tr(E) = \{acc(a) | a \in E\} \cup \{d(b) | b\ is\ an\ argument\ and\ b \notin E\}$

Now, let us introduce a theorem that essentially summarizes some relevant characterizations of extension-based argumentation semantics in terms of logic programming semantics with negation as failure.

[2] We say that c defends a if b attacks a and c attacks b.

Theorem 2. *Let AF be an argumentation framework and E be a set of arguments. Then:*

- *E is the grounded extension of AF iff $tr(E)$ is the well-founded model of $\Psi^1_{AF} \cup \Psi^2_{AF} \cup \Psi^3_{AF}$.*
- *E is a stable extension of AF iff $tr(E)$ is a stable model of $Psi^1_{AF} \cup \Psi^2_{AF} \cup \Psi^3_{AF}$.*
- *E is a preferred extension of AF iff $tr(E)$ is a p-stable model of $\Psi^1_{AF} \cup \Psi^2_{AF} \cup \Psi^3_{AF}$.*

This theorem was proved in [5]. Observe that essentially, this theorem is extending Theorem 1 and suggests that one can capture three of the well accepted argumentation semantics in one single logic program and three different logic programming semantics. The interesting part of this kind of results is that one can explore meta-interpreters of these argumentation semantics in terms of solvers of logic programming semantics . Also, one can explore non-monotonic reasoning properties of argumentation semantics in terms of their corresponding logic programming semantics. For instance, a possible research issue in this line is to explore the non-monotonic properties of the preferred semantics via p-stable semantics. It is worth mentioning that the p-stable semantics is based on paraconsistent logics. This suggests that one can characterize the preferred semantics in terms of paraconsistent logics.

The identification of non-monotonic reasoning properties which could represent a good-behavior of an argumentation semantics represents a hot topic since recently the number of new argumentation semantics in the context of Dungs argumentation approach has increased. The new semantics are motivated by the fact that the extension based semantics introduced by Dung in [7] have exhibited a variety of problems common to the grounded, stable and preferred semantics [14, 3].

One of the approaches that has been emerging for building new extension-based argumentation semantics was introduced in [3]. This approach is based on a solid concept in graph theory: strongly connected components (SCC). In [3], several alternative argumentation semantics were introduced; however, from them, CF2 is an argumentation semantics that has shown to have good-behavior [2].

In [12], an new approach for building argumentation semantics was introduced. In this approach the author suggests to use any logic programming semantics. By considering this approach, the authors were able to characterize the argumentation semantics CF2.

Theorem 3. *[12] Let AF be an argumentation framework and E be a set of arguments. Then, E is a CF2 extension AF iff $tr(E)$ is MM^r model of $\Psi^1_{AF} \cup \Psi^3_{AF}$.*

This theorem suggests that the relationship between the approach presented in [3] and [12] is close. In fact, both Theorem 2 and Theorem 3 suggest that not only one can characterize argumentation semantics in terms of logic programming semantics but also one can define new argumentation semantics in terms of logic programming semantics.

4 Inferring Extension-Based Semantics via NonMonotonic Reasoning Tools

So far, we have shown that logic programming semantics can be a suitable approach for exploring properties of extension-based argumentation semantics and defining new argumentation semantics. In this section, we show that the use of nonmonotic tools represent a potential approach for building intelligent systems based on an argumentation reasoning process.

4.1 Preferred Semantics by Positive Disjunctive Logic Programs

Taking advantage of the mappings presented in Definition 10, in [11], it was shown that by considering $\Psi^1_{AF} \cup \Psi^2_{AF}$ as a propositional formula its minimal models characterize the preferred extensions of the given argumentation framework. In fact, by transforming Ψ^1_{AF} into a positive disjunctive logic program one can characterize the preferred semantics in terms of stable model semantics as follows:

Definition 11. *Let $AF = \langle AR, attacks\rangle$ be an argumentation framework ,$a \in AR$ and $attacks(a,b)\ iff\ (a,b) \in attacks$. We define the transformation function $\Gamma(a)$ as follows:*

$$\Gamma(a) = \{ \bigcup_{b:attacks(b,a)} \{d(a) \vee d(b)\}\} \cup \{ \bigcup_{b:attacks(b,a)} \{d(a) \leftarrow \bigwedge_{c:attacks(c,b)} d(c)\}\}$$

Now we define the function Γ in terms of an argumentation framework.

Definition 12. *Let $AF = \langle AR, attacks\rangle$ be an argumentation framework. We define its associated general program as follows:*

$$\Gamma_{AF} = \bigcup_{a \in AR} \Gamma(a)$$

Theorem 4. *Let $AF = \langle AR, attacks\rangle$ be an argumentation framework and $S \subseteq AR$. S is a preferred extension of AF if and only if $compl(S)$ is a stable model of Γ_{AF}.*

This theorem suggest that one can use any disjunctive answer set solver as DLV [6] for inferring the preferred semantics. It is worth mentioning that by considering P^1_{AF}, P^2_{AF} and P^3_{AF} one can use any answer solver for computing the stable and grounded semantics.

4.2 Computing Extension-Based Argumentation Semantics by Labeling

Until now, we have shown that we can infer extension-based argumentation semantics by regarding an argumentation framework as a logic program. However, as we saw in §2.3 some extension-based argumentation semantics can be characterized by considering labeling process. The labeling represents an alternative approach for computing

extension-based argumentation semantics in terms of logic programming with negation as failure. In particular, there are results that show that by considering a labeling process one is able to compute the grounded, stable, preferred and complete argumentation semantics in terms of answer set semantics. An outstanding trend of computing such labellings is via answer set solvers.

The following two sections present studies that explore ASP applications that are able to compute various argumentation semantics, as well as decide whether an argument is sceptically or credulously accepted with reference to the chosen semantics. The idea behind these works is to transform an argumentation framework into a single normal logic program that references an argumentation semantics (either preferred, grounded, stable or complete). The answer set of the resulting program is a labeling [10] that expresses an argument-based extension for the chosen semantics.

Once programs to compute every available semantics are available, a procedure to decide if an argument is sceptically or credulously accepted in the semantic can be introduced. The following mapping between skeptical or credulous arguments, argumentation frameworks and argumentation semantics is defined. Given an argumentation framework $AF = \langle AR, attacks \rangle$, an argument $a \in AR$, and an argumentation semantics S: **a)** a is credulously justified if $a \in IN$ in, at least, one possible labeling that matches the restrictions defined by the semantic S. **b)** a is sceptically justified if $a \in IN$ in every possible labeling that matches the restrictions defined by the semantic S.

4.3 First Approach for Implementing an Argumentation MetaInterpreter in ASP

This section presents an approach for implementing an argumentation metainterpreter presented in [17] based on answer-set semantics to compute admissible, preferred, stable, semi-stable, complete, and grounded extensions.

The following programs are specified given an Argumentation Framework $AF = \langle AR, attacks \rangle$, two variables X and Y, the predicate symbol arg that denotes that the variable a is an argument of AF, and the functions $IN(X)$, $OUT(X)$ defined before.

The program formed by the set of rules Π_{CF} is able to compute complete extensions. The set of rules Π_{CF} is as follows:

$$IN(x) \leftarrow arg(x),\ not\ OUT(x). \quad OUT(x) \leftarrow arg(x),\ not\ IN(x).$$

$$\bot \leftarrow IN(x), IN(Y), defeat(X,Y). \quad defeat(X,Y) \leftarrow (X,Y) \in attacks.$$

The program formed by the set of rules $\Pi_{CF} \cup \Pi_{stable}$ is able to compute stable extensions. The set of rules Π_{stable} is as follows:

$$defeated(X) \leftarrow IN(Y), attack(Y,X) \in attacks. \quad \leftarrow OUT(X),\ not\ defeated(X).$$

The program formed by the set of rules $\Pi_{CF} \cup \Pi_{stable} \cup \Pi_{complete}$ is able to compute complete extensions. The set of rules $\Pi_{complete}$ is as follows:

$$not\_defended(X) \leftarrow defeat(Y,X), not\ defeated(Y). \quad \leftarrow OUT(X),\ not\ not\_defended(X).$$

Computing grounded, preferred and semi-stable extensions is a bit harder, because they require a labeling that maximizes or minimizes the elements in a set. In order to compute these extensions, this approach makes use of a *stratified* program [1]. This program defines the order operator ($<$) between the arguments in the domain, and derives

predicates for minimal, maximal and successor. For doing so, the following program, $\Pi_<$ is defined:

$lt(X,Y) \leftarrow arg(X), arg(Y), X < Y. \quad nsucc(X,Z) \leftarrow lt(X,Y), lt(Y,Z).$

$succ(X,Y) \leftarrow lt(X,Y), not\ nsucc(X,Y). \quad ninf(Y) \leftarrow lt(X,Y).$

$inf(X) \leftarrow arg(X), not\ ninf(X). \quad nsup(X) \leftarrow lt(X,Y). \quad sup(X) \leftarrow arg(X), not\ nsup(X).$

Using the program above, the predicate $defend(X)$ is defined. This predicate is derived if for a given Argumumentation Framework $AF = \langle AR, attacks \rangle$, each argument $Y : Y \in AR \ \ defend(X,Y)$. In order ensure that $defend(X,Y)$ is checked for every available argument, variable Y ranges from the minimal to the maximal using successor relations. The program Π_{defend} that defines the above mentioned predicate, is as follows:

$defend\_upto(X,Y) \leftarrow inf(Y), arg(X), not\ defeat(Y,X).$

$defend\_upto(X,Y) \leftarrow inf(Y), in(Z), defeat(Z,Y), defeat(Y,X).$

$defend\_upto(X,Y) \leftarrow succ(Z,Y), defend\_upto(X,Z), not\ defeat(Y,X).$

$defend\_upto(X,Y) \leftarrow succ(Z,Y), defend\_upto(X,Z), in(V), defeat(V,Y), defeat(Y,X).$

$defend(X) \leftarrow sup(Y), defend\_upto(X,Y).$

The program formed by the set of rules $\Pi_< \cup \Pi_{defend} \cup \Pi_{ground}$ computes grounded extensions. The set of rules Π_{ground} is as follows:

$in(X) \leftarrow defended(X).$

Computing both preferred and semi-stable extensions will make use of the programs $\Pi_<$ and Π_{defend} defined above. However this process is a bit more complicated, because it requires computing maximal sets instead of minimal ones. For tackling this issue the analyzed approach makes use of a saturation technique. This technique consists in building a labeling S such that every other possible labeling T that complies with certain conditions (either maximal IN or $UNDEC$ labels) does not characterize an admissible extension.

First of all, in order to model the maximal sets that both preferred and semi-stable extensions require, the program Π_{defend} presented above is slightly modified, to include predicates inN and $outN$. The meaning of the predicates varies depending the type of semantic computed (either preferred or semi-stable) so they are defined later, in the program associated to this semantics. The program $\Pi_{undefeated}$ is defined as follows:

$undefeated\_upto(X,Y) \leftarrow inf(Y), outN(X), outN(Y).$

$undefeated\_upto(X,Y) \leftarrow inf(Y), outN(X), not\ defeat(Y,X).$

$undefeated\_upto(X,Y) \leftarrow succ(Z,Y), undefeated\_upto(X,Z), outN(Y).$

$undefeated\_upto(X,Y) \leftarrow succ(Z,Y), undefeated\_upto(X,Z), not\ defeat(Y,X).$

$undefeated(X) \leftarrow sup(Y), undefeated\_upto(X,Y).$

The program formed by the sets of rules $\Pi_{adm} \cup \Pi_< \cup \Pi_{undefeated} \cup \Pi_{preferred}$ computes preferred extensions. The set of rules $\Pi_{preferred}$ is as follows:

$eq\_upto(Y) \leftarrow inf(Y), in(Y), inN(Y). \quad eq\_upto(Y) \leftarrow inf(Y), out(Y), outN(Y).$

$eq\_upto(Y) \leftarrow succ(Z,Y), in(Y), inN(Y), eq\_upto(Z).$

$eq\_upto(Y) \leftarrow succ(Z,Y), out(Y), outN(Y), eq\_upto(Z).$

$eq \leftarrow sup(Y), eq\_upto(Y). \quad inN(X) \vee outN(X) \leftarrow out(X). \quad inN(X) \leftarrow in(X)$

$spoil \leftarrow eq. \quad spoil \leftarrow inN(X), inN(Y), defeat(X,Y).$

$spoil \leftarrow inN(X), outN(Y), defeat(Y,X), undefeated(Y).$

$inN(X) \leftarrow spoil, arg(X). \quad outN(X) \leftarrow spoil, arg(X). \quad \bot \leftarrow not\, spoil.$

Last but not least, the program formed by the sets of rules $\Pi_{adm} \cup \Pi_{<} \cup \Pi_{undefeated} \cup \Pi_{semi-stable}$ computes semi-stable extensions.. The set of rules $\Pi_{semi-stable}$ is as follows:

$ag(a) \leftarrow$. for any argument $a \in AR \quad def(a, b)$. for any pair $(a, b) \in attacks$

$eqplus\_upto(Y) \leftarrow inf(Y), in(Y), inN(Y).$

$eqplus\_upto(Y) \leftarrow inf(Y), in(Y), inN(X), defeat(X, Y).$

$eqplus\_upto(Y) \leftarrow inf(Y), in(X), inN(Y), defeat(X, Y).$

$eqplus\_upto(Y) \leftarrow inf(Y), in(X), inN(Z), defeat(X, Y), defeat(Z, Y).$

$eqplus\_upto(Y) \leftarrow inf(Y), out(Y), outN(Y), not\, defeated(Y), undefeated(Y).$

$eqplus\_upto(Y) \leftarrow succ(Z, Y), in(Y), inN(Y), eqplus\_upto(Z).$

$eqplus\_upto(Y) \leftarrow succ(Z, Y), in(Y), inN(X), defeat(X, Y), eqplus\_upto(Z).$

$eqplus\_upto(Y) \leftarrow succ(Z, Y), in(Y), inN(Y), defeat(X, Y), eqplus\_upto(Z).$

$eqplus\_upto(Y) \leftarrow succ(Z, Y), in(X), inN(U), defeat(X, Y), defeat(U, Y), eqplus\_upto(Z).$

$eqplus\_upto(Y) \leftarrow succ(Z, Y), out(Y), outN(Y), not\, defeated(Y), undefeated(Y), eqplus\_upto(Z).$

$eqplus \leftarrow sup(Y), eqplus\_upto(Y). \quad inN(X) \vee outN(X) \leftarrow arg(X). \quad spoil \leftarrow eqplus.$

$spoil \leftarrow inN(X), inN(Y), defeat(X, Y).$

$spoil \leftarrow inN(X), outN(Y), defeat(Y, X), undefeated(Y).$

$spoil \leftarrow inN(X), outN(X), undefeated(X).$

$spoil \leftarrow inN(Y), defeat(Y, X), outN(X), undefeated(X).$

$inN(X) \leftarrow spoil, arg(X). \quad outN(X) \leftarrow spoil, arg(X). \quad \bot \leftarrow not\, spoil.$

4.4 Second Approach for Implementing an Argumentation MetaInterpreter in ASP

The work presented in this section[16] is an approach for computing Dung's standard argumentation semantics as well as semi-stable semantics via ASP.

The base of all this programs is a set of rules defining the Argumentation Framework, that is, both the available arguments and the attack relations between them. Given an argumentation framework $AF = \langle AR, attacks \rangle$, and two constants a and b the following ASP program , known as Π_{AF} can be defined:

$ag(a) \leftarrow$. for any argument $a \in AR \quad def(a, b)$. for any pair $(a, b) \in attacks$

The program formed by the set of rules $\Pi_{AF} \cup \Pi_{Complete}$ is able to compute complete extensions. Given two variables X and Y, the predicate symbols ng and arg and the functions $IN(x), OUT(x), UNDEC(x)$ defined before, the set of rules $\Pi_{Complete}$ is as follows:

$IN(x) \leftarrow ag(x),\ not\, ng(x). \quad ng(x) \leftarrow IN(y),\ def(y, x). \quad ng(x) \leftarrow UNDEC(y),\ def(y, x).$

$OUT(x) \leftarrow IN(y),\ def(y, x). \quad UNDEC(x) \leftarrow ag(x),\ not\, IN(x)\ ,\ not\, OUT(x).$

The program formed by the set of rules $\Pi_{AF} \cup \Pi_{Complete} \cup \Pi_{Stable}$ is able to compute stable extensions. The set of rules Π_{Stable} is as follows:

$\bot \leftarrow UNDEC(x)$

Computing grounded, preferred and semi-stable extensions is a bit harder, because it implies the need to minimize or maximize the elements in certain sets. In order to be able to compute such semantics, the following program, known as Π_{AF_MAX} is defined:

$m1(Lt) \leftarrow L$. for any set of arguments $Lt : \forall a \in Lt IN(a) \cup UNDEC(a) \quad m2(Lt, j) \leftarrow \top. \quad cno(j) \leftarrow \top.$
$1 \geq j \geq \xi$ where ξ is the number of possible labellings in the framework, and Lt is a set of arguments labeled IN or UNDEC in the labeling number j.
$i(Lt)$ for any set of arguments $Lt : \forall a \in Lt\, IN(a) \quad u(Lt)$ for any set of arguments $Lt : \forall a \in Lt\, UNDEC(a)$

The program formed by the set of rules $\Pi_{AF} \cup \Pi_{AF_M AX} \cup \Pi_{Complete} \cup \Pi_{Preferred}$ is able to compute preferred extensions.. The set of rules $\Pi_{Preferred}$ is as follows:

$c(Y) \leftarrow cno(Y), m1(X), i(X), not\, m2(X, Y). \quad d(Y) \leftarrow m2(X, Y), i(X), not\, m1(X).$
$\bot \leftarrow d(Y), not\, c(Y).$

The program formed by the set of rules $\Pi_{AF} \cup \Pi_{AF_M AX} \cup \Pi_{Complete} \cup \Pi_{Grounded}$ is able to compute grounded extensions. The set of rules $\Pi_{Grounded}$ is as follows:

$c(Y) \leftarrow cno(Y), m1(X), i(X), not\, m2(X, Y). \quad d(Y) \leftarrow m2(X, Y), i(X), not\, m1(X).$
$\bot \leftarrow c(Y), not\, d(Y).$

The program formed by the set of rules $\Pi_{AF} \cup \Pi_{AF_M AX} \cup \Pi_{Complete} \cup \Pi_{semi-stable}$ is able to compute semi-stable extensions.
The set of rules $\Pi_{semi-stable}$ is as follows:

$c(Y) \leftarrow cno(Y), m1(X), u(X), not\, m2(X, Y). \quad d(Y) \leftarrow m2(X, Y), u(X), not\, m1(X).$
$\bot \leftarrow c(Y), not\, d(Y).$

4.5 Discussion *w.r.t.* Label Process

Labeling provides an easy and intuitive approach to argumentation theory. It is based on simple and easy principles that are simpler to explain and understand than extension-based approaches.

Computing the labels that are to be assigned to nodes via an ASP program allows to build and interpreter that processes the Argumentation Framework as input, in contrast to using a fixed logic program which depends on the Argumentation Framework to process. In this sense, the interpreter is easier to understand, extend and debug. Moreover, it eases the process of formally proving the correspondence between answer sets and extensions. Finally, having an interpreter which is independent of the Argumentation Framework allows to easily change the framework without having to re-generate the logic program. However, it must be noted that a full decoupling between the input Argumentation Framework and the programs used to compute its labellings has not been achieved. Both works present programs that are bounded to the Argumentation Framework when computing grounded, preferred and semi-stable extension. In the first work analyzed if the Argumentation Framework changes, predecessor, successor and maximal properties of arguments must be re-built, whereas in the second one constant ξ must be updated.

It must be noted that treating the labeling as a classical graph labeling problem can be useful in argumentation semantics that allow different types of attack relations between arguments (e.g. support and collective attacks,attacks on attacks and so on). In this case, the labeling approach provides a basis for propagating the label of a node through node connections.

5 Conclusions

In this paper, we have presented a survey of resent results *w.r.t.* the dual relationship between extension-based argumentation semantics and logic programming semantics.

In particular, we have presented results which explore the characterization of extension-based argumentation semantics in terms of logic programming semantics with

negation as failure (§3). We saw that the argumentation semantics based on admissible sets can be characterized by well-acceptable logic programming semantics. In fact, there are results that suggest that extension-based argumentation semantics based on strongly connected components such as CF2 can also be characterized by logic programming semantics with negation as failure. These results suggest that there are behaviours which are shared by both approaches of reasoning. It is worth mentioning that argumentation theory and logic programming semantics with negation as failure represent two successful approaches for performing *common sense reasoning*. A possible interesting research issue can be the identification of the non-monotonic reasoning properties which are shared by both approaches.

The specification of argumentation frameworks in terms of logic programs can also define some approaches for inferring extension-based argumentation semantics in order to construct argumentation systems. For instance, we have presented results that show that answer set programming is suitable approach for building Argumentation MetaInterpreters (see §4).

We also presented two approaches for exploring extension-based argumentation semantics in terms of logic programming and a labelling process. On the one hand, the first approach [17] presents a more complex but more versatile code due to the usage of the stratified programming and saturation techniques. On the other hand, the second one [16] presents a more simple code, but it has to be meta-coded by an external program due to being bound to the number of available arguments in the argumentation framework. Both approaches are based on mapping the argumentation framework to a labeling system, this facilitates the computational treatment of the argumentation semantics in terms of declarative specifications.

There are still several open issues to explore in the close relationship between argumentation theory and logic programming with negation as failure. One of the most appealing issues is exploring the inference of extension-based argumentation semantics based on strongly connected components in terms of answer set models. By the moment, we know that the logic programming semantics MM^r can infer the argumentation semantics CF2 [13].

References

1. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Cambridge, 2003.
2. P. Baroni and M. Giacomin. On principle-based evaluation of extension-based argumentation semantics. *Artificial Intelligence.*, 171(10-15):675–700, 2007.
3. P. Baroni, M. Giacomin, and G. Guida. SCC-recursiveness: a general schema for argumentation semantics. *Artificial Intelligence*, 168:162–210, October 2005.
4. T. J. M. Bench-Capon and P. E. Dunne. Argumentation in artificial intelligence. *Artificial Intelligence*, 171(10-15):619–641, 2007.
5. J. L. Carballido, J. C. Nieves, and M. Osorio. Inferring Preferred Extensions by Pstable Semantics. *Iberoamerican Journal of Artificial Intelligence (Inteligencia Artificial) ISSN: 1137-3601*, 13(41):38–53, 2009 (doi: 10.4114/ia.v13i41.1029).
6. S. DLV. Vienna University of Technology. http://www.dbai.tuwien.ac.at/proj/dlv/, 1996.

7. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.
8. G. Governatori, M. J. Maher, G. Antoniou, and D. Billington. Argumentation semantics for defeasible logic. *J. Log. Comput.*, 14(5):675–702, 2004.
9. H. Jakobovits and D. Vermeir. Robust semantics for argumentation frameworks. *Journal of logic and computation*, 9(2):215–261, 1999.
10. S. Modgil and M. Caminada. *Argumentation in Artificial Intelligence*, chapter Proof Theories and Algorithms for Abstract Argumentation Frameworks, pages 105–129. Springer, 2009.
11. J. C. Nieves, M. Osorio, and U. Cortés. Preferred Extensions as Stable Models. *Theory and Practice of Logic Programming*, 8(4):527–543, July 2008.
12. J. C. Nieves, M. Osorio, and C. Zepeda. Expressing Extension-Based Semantics based on Stratified Minimal Models. In H. Ono, M. Kanazawa, and R. de Queiroz, editors, *Proceedings of WoLLIC 2009, Tokyo, Japan*, volume 5514 of *FoLLI-LNAI subseries*, pages 305–319. Springer Verlag, 2009.
13. M. Osorio, A. Marin-George, and J. C. Nieves. Computing the Stratified Minimal Models Semantic. In *LANMR'09*, pages ?–?, 2009.
14. H. Prakken and G. A. W. Vreeswijk. Logics for defeasible argumentation. In D. Gabbay and F. Günthner, editors, *Handbook of Philosophical Logic*, volume 4, pages 219–318. Kluwer Academic Publishers, Dordrecht/Boston/London, second edition, 2002.
15. I. Rahwan and P. McBurney. Argumentation technology: Introduction to the special issue. *IEEE Intelligence Systems*, 22(6):21–23, 2007.
16. K. N. Toshiko Wakaki. *New Frontiers in Artificial Intelligence*, volume 5447/2009 of *Lecture Notes in Computer Science*, pages 254–269. 2009.
17. S. W. Uwe Egly, Sarah Alice Gaggl. Answer-set programming encodings for argumentation frameworks. Technical report, DBAI, 2008.