A General Schema For Generating Argumentation Semantics From Logic Programming Semantics

Juan Carlos Nieves Mauricio Osorio*

Abstract

In this paper, by considering the idea that argumentation semantics can be viewed as a special form of logic programming semantics with negation as failure, we show that any logic programming semantics as the stable model semantics, the minimal models, *etc.*, can define candidate argumentation semantics. These new argumentation semantics will overcome some of the problems of the Dung's argumentation semantics that have been discussed in the literature.

The new argumentation semantics are based on a new recursive framework for logic programming semantics. This framework generalizes any logic programming semantics in order to build logic programming semantics which are always defined, satisfy the property of relevance and agree with the stable semantics for the class of stratified programs.

1 Introduction

Argumentation theory has become an increasingly important and exciting research topic in Artificial Intelligence (AI), with research activities ranging from developing

*The authors are named by alphabetic order.

Juan Carlos Nieves — Universitat Politècnica de Catalunya. Software Department (LSI), C/Jordi Girona 1-3, E08034, Barcelona, Spain. E-mail: jcnieves@lsi.upc.edu

Mauricio Osorio — Universidad de las Américas - Puebla. CENTIA, Sta. Catarina Mártir, Cholula, Puebla, 72820 México. E-Mail: osoriomauri@googlemail.com)

theoretical models, prototype implementations, and application studies [5]. The main purpose of argumentation theory is to study the fundamental mechanism, humans use in argumentation, and to explore ways to implement this mechanism on computers.

Dung's approach, presented in [13], is a unifying framework which has played an influential role on argumentation research and AI. This approach is mainly orientated to manage the interaction of arguments. The interaction of the arguments is supported by four abstract argumentation semantics: *stable semantics*, *preferred semantics*, *grounded semantics*, and *complete semantics*. The central notion of these semantics is the *acceptability of the arguments*. According to Bench-Capon and Dunne, the three principal abstract argumentation semantics introduced by Dung are the grounded, preferred and stable semantics. However, these semantics exhibit a variety of problems which have illustrated in the literature [30, 4, 6, 7, 5]. Authors as P. Baroni *et al*, have suggested that in order to overcome Dung's abstract argumentation semantics problems, it is necessary to define flexible argumentation semantics which are not necessarily based on admissible sets [4].

According to Baroni *et al*, in [4] the preferred semantics is regarded as the most satisfactory approach; however, they have also pointed out that the preferred semantics produces some questionable results in some cases concerning cyclic attack relations [4]. For instance, let us consider the argumentation framework that appears in Figure 1[1]. In this argumentation framework there are two arguments: a and b. The arrows in the figure represent conflicts between the arguments. We can see that the argument a is attacked by itself and the argument b is attacked by the argument a. Intuitively, some authors as Prakken and Vreeswijk [30] suggest that one can expect that the argument b can be considered as an acceptable argument since it is attacked by the argument a which is attacked by itself. However, the preferred semantics is unable to infer the argument b as an acceptable argument — the only preferred extension of the argumentation framework of Figure 1 is the empty set. In fact, none of the argumentation semantics suggested by Dung is able to infer the argument b as acceptable.

Another interesting argumentation framework which has been commented on literature [30, 4] is presented in Figure 2. The preferred semantics *w.r.t.* this argumentation

[1]This argumentation framework has received special attention in the literature in order to commented the problem of the self-defeated arguments [29, 30] and to point out some of the problems of the Dung's argumentation semantics [4].

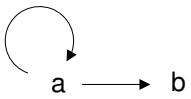

Figure 1: Graph representation of the argumentation framework $AF = \langle\{a, b\}, \{(a, a), (a, b)\}\rangle$.

framework is only able to infer the empty set. Some authors, as Prakken and Vreeswijk [30], Baroni *et al*[4], suggest that the argument e can be considered as an acceptable argument since it is attacked by the argument d which is attacked by three arguments: a, b, c. Observe that the arguments a, b and c form a cyclic of attacks.

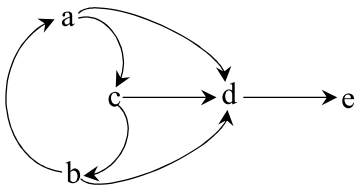

Figure 2: Graph representation of the argumentation framework $AF = \langle\{a, b, c, d, e\}, \{(a, c), (c, b), (b, a), (a, d), (c, d), (b, d), (d, e)\}\rangle$.

The stable argumentation semantics defined by Dung in [13] is also considered as another proper argumentation semantics. However, this semantics has been criticized by some authors as Bench-Capon and Dunne [5], Caminada [7] because frequently this semantics is undefined . For instance, the argumentation framework of Figure 3 has no stable extensions; however, it has a preferred extension, $\{c\}$ — in fact the argumentation frameworks of Figure 1 and Figure 2 are two examples where the stable argumentation semantics is also undefined.

The solutions to the problems of the argumentation semantics suggested by Dung are really diverse some researchers have focused on improving the stable argumentation semantics [7], some other researchers have focused on improving the preferred semantics [25], and still other researchers have focused on improving the concept of *admissible set* which is the basis of the argumentation semantics suggested by Dung

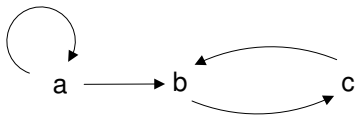

Figure 3: Graph representation of the argumentation framework $AF = \langle\{a, b, c\}, \{(a, a), (a, b), (b, c), (c, b)\}\rangle$.

[16, 4].

It is worth mentioning that since Dung's approach was introduced in [13], it was viewed as a special form of logic programming with *negation as failure*. For instance, in [13] it was proved that the grounded semantics can be characterized by the well-founded semantics [14] and the stable argumentation semantics can be characterized by the stable model semantics [15]. Also in [9], it was proved that the preferred semantics can be characterized by the pstable semantics. In fact, the preferred semantics can be also characterized by the minimal models and the stable models of a logic program [22, 23].

We can recognize two major branches for improving Dung's approach. On the one hand, we can take advantage of graph theory; on the other hand, we can take advantage of logic programming with negation as failure.

With respect to graph theory, the approach suggested by Baroni *et al*, in [4] is maybe the most general solution defined until now for improving Dung's approach. This approach is based on a solid concept in graph theory which is a *strongly connected component* (SCC). Based on this concept, Baroni *et al*, describe a recursive approach for generating new argumentation semantics. For instance, the argumentation semantics CF2 suggested in [4] is able to infer the argument b as an acceptable argument of the argumentation framework of Figure 1. Also CF2 is able to infer the extensions: $\{a, e\}$, $\{b, e\}$, $\{c, e\}$ from the argumentation framework of Figure 2. This means that CF2 regards the argument e as an acceptable argument.

As we commented, argumentation theory can be viewed as a special form of logic programming with negation as failure [13, 22, 24, 23, 9]. In fact Dung in [13] introduced a metainterpreter P_{AF} for generating argumentation systems. When we have a logic program which represents an argumentation framework, it is natural to think that

we can split this program into subprograms where each subprogram could represent a part of an argumentation framework. For instance, let us consider a single version of the mapping Ψ_{AF} introduced in [22, 21, 23, 9] in order to represent the argumentation framework of Figure 1 as the logic program P:

$$d(a) \leftarrow \neg d(a).$$
$$d(b) \leftarrow \neg d(a).$$
$$acc(a) \leftarrow \neg d(a).$$
$$acc(b) \leftarrow \neg d(b).$$

We want to point out that we are only considering the negative clauses of the mapping Ψ_{AF} and two clauses more in order to infer the acceptable arguments by *negation as failure*. This program can be also inferred from Dung's mapping P_{AF} [13] by considering the grounding instance of P_{AF} and applying the well-known principle of partial evaluation to P_{AF}. In fact, this codification can be regarded as the common point between the mappings Ψ_{AF} and P_{AF}.

The intended meaning of the first clause of P says that the argument a is defeated if the argument a is not defeated. The second clause of P says that the argument b is defeated if the argument a is not defeated. The third clause of P says that the argument a is acceptable if the argument a is not defeated and the last clause of P says that the argument b is acceptable if the argument b is not defeated.

Notice that the program P can be split into three subprograms, *i.e.* P_1, P_2 and P_3, where P_1 is:

$$d(a) \leftarrow \neg d(a).$$

P_2 is:

$$d(b) \leftarrow \neg d(a).$$
$$acc(a) \leftarrow \neg d(a).$$

and P_3 is:

$$acc(b) \leftarrow \neg d(b).$$

We can see that P_2 depends on P_1 because the atom $d(a)$ is defined in the program P_1. In the same way, P_3 depends on P_1 and P_2. Hence, in order to infer the semantics

of P_2, we have to infer the semantics of P_1 before. For instance, let us consider the minimal models of P_1. It is easy to see that the only minimal model of P_1 is: $\{d(a)\}$. Hence, in order to infer the semantics of P_2 based on the minimal models of P_1, we can remove from P_2 any clause that contains $\neg\ d(a)$ in their bodies — let P_2' be the reduced program. Notice that P_2' is an empty program; hence, the only minimal model of P_2' is the empty model *i.e.* the atoms $d(b)$ and $acc(a)$ are considered as false. Now, for inferring the semantics of P_3, we consider the minimal models of P_1 union the minimal models of P_2'. We can infer the semantics of P_3 based on the model $\{d(a)\}$ — let P_3' be the reduced program by considering $d(a)$ as true and $d(b)$ as false. It is easy to see that the only minimal model of P_3' is: $\{acc(b)\}$. Therefore, the semantics of P will be the union of the minimal models of P_1 ($\{d(a)\}$) union the minimal models of P_2' ($\emptyset$) union the minimal models of P_3' ($\{acc(b)\}$). Hence, we have an unique model for P which is $\{acc(b), d(a)\}$. This model suggests that we can consider the argument b as acceptable and the argument a as defeated.

The idea of spitting a logic program into its component, in order to define logic programming semantics, has been explored by some authors in logic programming [12]. For instance, by splitting a logic program, Dix and Müller in [12] combine ideas of the stable model semantics and the well-founded semantics in order to define a skeptical logic programming semantics which satisfies the property of relevance and the general principle of partial evaluation.

In the first part of this paper, we will formalize a recursive general schema for constructing new logic programming semantics. This recursive general schema is based on the idea of splitting a logic program into its components. The new logic programming semantics have as main properties that they are always defined for any logic program and they satisfy the property of relevance.

In the second part of this paper, by considering the idea that argumentation semantics can be viewed as a special form of logic programming semantics with negation as failure and the schema presented in the first part of the paper, we show that any logic programming semantics as the stable model semantics, the minimal models, *etc.*, can define candidate argumentation semantics. These new argumentation semantics will overcome some of the problems of the Dung's argumentation semantics that have been discussed in the literature. In fact, we will see that some of our new argumentation semantics have similar behavior to the argumentation semantics defined in terms *strongly*

connected components [4].

The rest of the paper is divided as follows: In §2, we present some basic concept *w.r.t.* logic programming and argumentation theory. In §3, we introduce our new recursive general schema for defining new logic programming semantics. In §4, we define how to construct new argumentation semantics based on the approach presented in §3. In §5, we will point out some of the main common points between our approach and Baroni *et al*'s approach. Finally in the last section, we present our conclusion and ourline our future work.

2 Background

In this section, we define the syntax of the logic programs that we will use in this paper. In terms of logic programming semantics, we present the definition of the stable model semantics and the pstable model semantics. After that, we present a short description of the Dung's argumentation approach.

2.1 Logic programming: Syntax and some operations

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 $\neg a$ (negative literal). Given a set of atoms $\{a_1, ..., a_n\}$, we write $\neg\{a_1, ..., a_n\}$ to denote the set of atoms $\{\neg a_1, ..., \neg a_n\}$. A normal clause, *C*, is denoted as $a \leftarrow l_1, \ldots, l_n$, where $n \geq 0$, a is an atom, and each l_i is a literal. When $n = 0$ the clause is an abbreviation of $a \leftarrow \top$[2], where $\top$ is $\neg\bot$. Sometimes, we denote a normal clause *C* by $a \leftarrow \mathcal{B}^+,\ \neg\mathcal{B}^-$, where $\mathcal{B}^+$ contains all the positive body atoms and $\mathcal{B}^-$ contains all the negative body atoms. We also use $body(C)$ to denote $\mathcal{B}^+ \cup \neg\mathcal{B}^-$. A normal program P is a finite set of normal clauses, formally a normal program is a conjunction of its normal clauses. When $\mathcal{B}^- = \emptyset$, the clause is called definite. A finite set of definite clauses is called definite program.

Given a normal program P, we will call *non-trivial tautology* $C \in P$ if C is at one of the following forms:

[2]or simply a.

$$C \quad = \quad a \leftarrow (\mathcal{B}^+ \cup \{a\}),\ \neg\mathcal{B}^- \text{ such that } \mathcal{B}^+ \neq \emptyset \text{ or } \mathcal{B}^- \neq \emptyset \tag{1}$$

$$C \quad = \quad a \leftarrow (\mathcal{B}^+ \cup \{x\}),\ \neg(\mathcal{B}^- \cup \{x\}) \text{ such that } x \in \mathcal{L}_P \tag{2}$$

For instance, the clauses $a \leftarrow a, \neg b$ and $b \leftarrow a, \neg a$ are two non-trivial tautologies. The clauses of the form $x \leftarrow x$ will be called *trivial tautologies*.

We denote by $HEAD(P)$ the set $\{a | a \leftarrow \mathcal{B}^+,\ \neg\mathcal{B}^- \in P\}$. 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 programs defined over $\mathcal{L}$.

Remark 1 *We want to point out that*

- *whenever we consider logic programs, our negation* $\neg$ *corresponds to the default negation "not" used in* Logic Programming*;*
- *and whenever we treat a logic program as a theory, our negation* $\neg$ *corresponds to the negation of* classical logic.

A program P induces a notion of $dependency$ between atoms from $\mathcal{L}_P$. We say that $a\ depends\ immediately\ on\ b$ if and only if b appears in the body of a clause in P, such that a appears in its head. The two place relation $depends\ on$ is the transitive closure of $depends\ immediately\ on$. The $dependencies\ of\ an\ atom\ x$ is defined by:

$dependencies\text{-}of(x)$ is the set $\{a | x\ depends\ on\ a\}$

In order to illustrate this definition, let us consider the following normal program, denoted by RE (Running Example):

$e \leftarrow e.$

$c \leftarrow c.$

$a \leftarrow \neg b, c.$

$b \leftarrow \neg a, \neg\ e.$

$d \leftarrow b.$

We can see that $\mathcal{L}_P = \{a, b, c, d, e\}$. Now let us infer the dependency relations between the atoms of $\mathcal{L}_P$:

$dependencies\text{-}of(a) = \{a, b, c, e\}$
$dependencies\text{-}of(b) = \{a, b, c, e\}$
$dependencies\text{-}of(c) = \{c\}$
$dependencies\text{-}of(d) = \{a, b, c, e\}$
$dependencies\text{-}of(e) = \{e\}$

We define an equivalence relation $\equiv$ between atoms of $\mathcal{L}_P$ as follows: $a \equiv b$ if and only if $a = b$ or (a *depends-on* b and b *depends-on* a). We write $[a]$ to denote the equivalent class induced by the atom a. By considering again the normal program RE, we can see that:

$$[a] = [b] = \{a, b\} \quad [d] = \{d\}$$
$$[c] = \{c\} \quad [e] = \{e\}$$

We take $\leq_P$ to denote the partial order induced by $\equiv_P$ on its equivalent classes. Hence, $[a] \leq [b]$ if and only if b *depends-on* a. For instance, by considering the equivalent classes of the program RE, the following relations hold: $\{c, e\} \leq \{a, b\} \leq \{d\}$. By considering the relation $\leq_P$, each atom of $\mathcal{L}_P$ is assigned an order as follows:

- An atom a is *of order* 0, if $[a]$ is minimal in $\leq_P$.
- An atom a is *of order* $n + 1$, if n is the maximal order of the atoms of which a depends such that n is the order of the atom b and $b \neq a$.

We say that a program P is of order n if n is the maximum order of its atoms. By considering again the normal program RE, we can see that:

a is of order 1 $\quad$ d is of order 2
b is of order 1 $\quad$ e is of order 0
c is of order 0

this means that RE is a program of order 2.

We can also break a program P (of order n) into the disjoint union of programs P_i ($0 \leq i \leq n$) such that P_i is the set of rules such that the head of each atom is of order i (with respect to P). We say that $P_0, \ldots, P_n$ are the relevant modules of P. In order to illustrate these ideas, let us consider the following table, where RE_0, RE_1, RE_2 are the respective relevant modules of the normal program RE:

RE	RE_0	RE_1	RE_2
$e \leftarrow e.$	$e \leftarrow e.$		
$c \leftarrow c.$	$c \leftarrow c.$		
$a \leftarrow \neg b, c.$		$a \leftarrow \neg\, b, c.$	
$b \leftarrow \neg a, \neg\, e.$		$b \leftarrow \neg a, \neg e.$	
$d \leftarrow b.$			$d \leftarrow b.$

There is an interesting class of normal programs that is called *stratified programs*. This class of normal logic programs satisfies certain syntactic conditions *w.r.t.* the occurrence of their positive and negative literals. The stratified logic programs have gained a lot of importance in connection with the search for nice declarative semantics for logic programs and the treatment of negative information in logic programming — the interested reader can find some interesting results *w.r.t.* stratified programs in [19, 2]. The formal definition of a stratified logic program is defined as follows:

Definition 1 *[2] Let P be a normal logic program. P is called a stratified logic program if for any clause $a \leftarrow l_1, \ldots, l_m, \neg l_{m+1}, \ldots, \neg l_{nn}$ in P_i, $0 \leq i \leq n$, then*

1. *for every l_j, $1 \leq j \leq m$ there is a P_q such that $l_j \in \mathcal{L}_{P_q}$ and $q \leq i$, and*
2. *for every l_j, $m+1 \leq j \leq nn$, there is a P_q such that $l_j \in \mathcal{L}_{P_q}$ and $q < i$.*

Observe that the first condition only says that the positive literals of the clause can appear in any component lower than or equal to P_i, and the second condition says that the negative literals must appear in a clause which belongs to a component strictly lower than P_i. For instance, one can see that the logic program RE, introduced above, is not a stratified normal programs. However, the component RE_0 is a stratified logic program by itself.

Now we introduce a single reduction for any normal program. The idea of this reduction is to remove from a normal program any atom which has already fixed to some true value. In fact this reduction is based on a pair of sets of atoms $\langle T; F\rangle$ such that the set T contains the atoms which can be considered as true and the set F contains the atoms which can be considered as false. Formally, this reduction is defined as follows:

Let $\langle T; F\rangle$ be a pair of sets of atoms. The reduction $R(P, \langle T; F\rangle)$ is obtained by 4 steps:

1. We replace every atom x that occurs in the bodies of P by 1 if $x \in T$ as well as we replace every atom x that occurs in the bodies of P by 0 if $x \in F$;

2. We replace every occurrence of $\neg 1$ by 0 and $\neg\ 0$ by 1;

3. Every clause with a 0 in its body is removed;

4. Finally we remove every occurrence of 1 in the body of the clauses.

Note that this is not the Gelfond-Lifschitz reduction (it will be presented in Section 2.2.1). For instance, let P be the normal program $RE \setminus RE_0$. This means that P is:

$a \leftarrow \neg b, c.$
$b \leftarrow \neg a, \neg e.$
$d \leftarrow b.$

Then, $R(P, \langle\{c\}; \{e\}\rangle)$ is:

$a \leftarrow \neg b.$
$b \leftarrow \neg a.$
$d \leftarrow b.$

2.2 Logic Programming: Semantics

Let P be a normal program. An interpretation I is a mapping from $\mathcal{L}_P$ to $\{0, 1\}$, where the generalization of I to connectives is as follows:

1. $I(a \wedge b) = min\{I(a), I(b)\}$,

2. $I(a \vee b) = max\{I(a), I(b)\}$,

3. $I(a \leftarrow b) = 0$ if and only if $I(b) = 1$ and $I(a) = 0$,

4. $I(\neg a) = 1 - I(a)$,

5. $I(\bot) = 0$,

6. $I(\top) = 1$.

It is standard to use sets of atoms to represent interpretations. The set corresponds exactly to those atoms that evaluate to 1. An interpretation M is called a (2-valued)

model of P if and only if for each clause $c \in P$, $M(c) = 1$. Finally, M is a minimal model of P if it does not exist a model M' of P such that $M' \subset M$ [31].

A semantics SEM is a (partial) mapping from the class of all programs into de powerset of the set of (2-valued) models. SEM assigns to every program P either a set of (2-valued) models of P or remains undefined.

Given a set of interpretations Q and a signature $\mathcal{L}$, we define Q $restricted\ to$ $\mathcal{L}$ as $\{M \cap \mathcal{L} | M \in Q\}$. For instance, let Q be $\{\{a, c\}, \{c, d\}\}$ and $\mathcal{L}$ be $\{c, d, e\}$, hence Q $restricted\ to$ $\mathcal{L}$ is $\{\{c\}, \{c, d\}\}$.

Let P be a program and $P_0, \ldots, P_n$ its relevant modules. We say that a semantics S satisfies the property of relevance if for every i, $0 \leq i \leq n$, $S(P_0 \cup \cdots \cup P_i) = S(P)$ restricted to $\mathcal{L}_{P_0 \cup \cdots \cup P_i}$.

Some logic programming semantics that we will consider are *the minimal model semantics* (denoted by MM), *the stable semantics* [15] (denoted by $Stable$), *the revised stable semantics* [28] (denoted by $RevStable$) and *the pstable semantics* [26] (denoted by $Pstable$). In the next subsections, we will define the stable model semantics and the pstable model semantics. It is worth mentioning that the minimal model semantics of a logic program P is given by the minimal models of P.

2.2.1 Stable model semantics.

The stable model semantics was defined in terms of the so called *Gelfond-Lifschitz reduction* [15] and it is usually studied in the context of syntax dependent transformations on programs. The following definition of a stable model for normal programs was presented in [15]:

Let *P* be any normal program. For any set $S \subseteq \mathcal{L}_P$, let P^S be the definite program obtained from *P* by deleting

(i) each rule that has a formula $\neg l$ in its body with $l \in S$, and then

(ii) all formulæ of the form $\neg l$ in the bodies of the remaining rules.

Clearly P^S does not contain $\neg$. Hence *S* is a stable model of *P* if and only if *S* is a minimal model of P^S.

In order to illustrate this definition let us consider the following example:

Example 1 *Let $S = \{b\}$ and P be the following logic program:*

$b \leftarrow \neg a.$ $\qquad$ $b \leftarrow \top.$

$c \leftarrow \neg b.$ $\qquad$ $c \leftarrow a.$

We can see that P^S is:

$b \leftarrow \top.$ $\qquad$ $c \leftarrow a.$

Notice that P^S has three models: $\{b\}$, $\{b, c\}$ and $\{a, b, c\}$. Since the minimal model amongst these models is $\{b\}$, we can say that S is a stable model of P.

2.2.2 Pstable model semantics.

Before to define the pstable semantics (introduced in [26]), we define some basic concepts. Logical inference in classic logic is denoted by $\vdash$. Given a set of proposition symbols S and a theory (a set of well-formed formulae) Γ, $\Gamma \vdash S$ if and only if $\forall s \in S$, $\Gamma \vdash s$. When we treat a logic program as a theory, each negative literal $\neg a$ is regarded as the standard negation operator in classic logic. Given a normal program P, if $M \subseteq \mathcal{L}_P$, we write $P \Vdash M$ when: $P \vdash M$ and M is a classical 2-valued model of P.

The Pstable semantics is defined in terms of a single reduction which is defined as follows:

Definition 2 *[26] Let P be a normal program and M a set of literals. We define*

$$RED(P, M) = \{l \leftarrow \mathcal{B}^+, \neg(\mathcal{B}^- \cap M) | l \leftarrow \mathcal{B}^+, \neg\mathcal{B}^- \in P\}$$

Let us consider the set of atoms $M_1 = \{a, b\}$ and the following normal program P_1:

$a \leftarrow \neg\, b, \neg\, c.$

$a \leftarrow b.$

$b \leftarrow a.$

We can see that $RED(P, M)$ is:

$a \leftarrow \neg b.$

$a \leftarrow b.$

$b \leftarrow a.$

By considering the reduction RED, it is defined the semantics pstable for normal programs.

Definition 3 *[26] Let P be a normal program and M a set of atoms. We say that M is a pstable model of P if $RED(P, M) \Vdash M$. We use Pstable to denote the semantics operator of pstable models.*

Let us consider again M_1 and P_1 in order to illustrate the definition. We want to verify whether M_1 is a pstable model of P_1. First, we can see that M_1 is a model of P_1, *i.e.* $\forall\ C \in P_1$, M_1 evaluates C to true. Now, we have to prove each atom of M_1 from $RED(P_1, M_1)$ by using classical inference, *i.e.* $RED(P_1, M_1) \vdash M_1$. Let us consider the proof of the atom a, which belongs to M_1, from $RED(P_1, M_1)$.

1. $(a \vee b) \rightarrow ((b \rightarrow a) \rightarrow a)$	Tautology
2. $\neg\, b \rightarrow a$	Premise from $RED(P_1, M_1)$
3. $a \vee b$	From 2 by logical equivalency
4. $(b \rightarrow a) \rightarrow a)$	From 1 and 3 by Modus Ponens
5. $b \rightarrow a$	Premise from $RED(P_1, M_1)$
6. a	From 4 and 5 by Modus Ponens

The proof for the atom b, which also belongs to M_1, is similar to the proof of the atom a. Then we can conclude that $RED(P_1, M_1) \Vdash M_1$. Hence, M_1 is a *pstable model* of P_1.

2.3 Argumentation Theory: Dung's approach

The fundamental Dung's definition is the concept called argumentation framework which is defined as follows:

Definition 4 *[13] An argumentation framework is a pair $AF = \langle AR, attacks \rangle$, where* AR *is a finite set of arguments, and* attacks *is a binary relation on* AR*,* i.e. attacks $\subseteq AR \times AR$*. We write $\mathcal{AF}_{AR}$ to denote the set of all the argumentation frameworks defined over AR.*

Any argumentation framework could be regarded as a directed graph. For instance, if $AF = \langle \{a, b, c\}, \{(a, b), (b, c)\} \rangle$, then AF is represented as in Figure 4. We say that *a attacks b* (or b is attacked by a) if $attacks(a, b)$ holds. 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. For instance in Figure 4, $\{a\}$ attacks b.

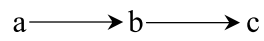

Figure 4: Graph representation of the argumentation framework $AF = \langle\{a, b, c\}, \{(a, b), (b, c)\}\rangle$.

Definition 5 [13] *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.*

For instance, the sets $\{a\}$, $\{b\}$, and $\{a, c\}$ are conflict-free sets *w.r.t.* Figure 4.

Definition 6 [13] *(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.*

One of the semantics of the Dung's approach which has played an influential role on argumentation research is the preferred semantics. This semantics is defined as follows:

Definition 7 *[13] A preferred extension of an argumentation framework AF is a maximal (*w.r.t. *inclusion) admissible set of AF.*

The admissible sets of Figure 4 are $\{a\}$ and $\{a, c\}$, then the only preferred extension is $\{a, c\}$. Another interesting semantics which was introduced in [13] is the *stable semantics*.

Definition 8 *A conflict-free set of arguments A is called a stable extension if and only if S attacks each argument which does not belong to S.*

We can see that the only stable extension of the argumentation framework of Figure 4 is $\{a, c\}$.

Remark 2 *Observe that there is a logic programming semantics that is called* stable model semantics *(presented in Section 2.2.1) and an argumentation semantics that is called* stable semantics *(Definition 8). Please be careful not to misunderstand each concept.*

Dung suggested other two argumentation semantics, however we do not present their definitions here.

3 Construction of new logic programming semantics

In this section we construct elaborated logic programming semantics based on a simpler logic programming semantics. We have in mind that our new logic programming semantics satisfy the following suitable properties:

1. *They should be always defined.* In both approaches logic programming and argumentation theory, it is appreciated that logic programming semantics (resp. argumentation semantics) can be always defined [5, 8, 28, 12]. Hence, our new logic programming semantics will be always defined.

2. *They should satisfy the relevance property.* The relevance property is an important property in order to define well-behaved semantics in logic programming [11, 12].

3. *They should agree with $Stable$ for the class of stratified programs.* On the one hand, stable model semantics is probably the most accepted logic programming semantics in the last two decades, and on the other hand, the class of stratified logic programs is the class of logic programs where the most accepted logic programming semantics agree [2, 19].

4. *They should be useful to model argumentation problems.* As we commented in §1, one of the main objectives of this paper is to suggest a general schema for constructing argumentation semantics based on logic programming semantics.

We assume that every semantics S satisfies the following trivial property:

For every program P such that every atom that appears in P also occurs as a fact of P then $S(P)$ is defined.

Every well known semantics satisfies this basic property such as $Stable$, MM, $RevStable$, and $Pstable$. In fact in all these semantics there is exactly one intended model that is $\mathcal{L}_P$. For instance, let P be the following program:

$a \leftarrow \top.$ $\quad\quad$ $b \leftarrow \top.$

$a \leftarrow b.$ $\quad\quad$ $b \leftarrow \neg a.$

Observe that all the atoms of the program P appear as facts in P; hence P will always have the model $\{a, b\}$.

Remark 3 *For the construction of our semantics we assume that our programs do not include* non-trivial tautologies *(see §2) . If a program includes them, we simply remove them in the very first stage of our construction of our semantics.*

3.1 Semantics always defined

It is sometimes desirable that a semantics of a normal program is always defined; for instance, the case when a program is modeling an argumentation problem. Given a particular semantics S, we show how to construct a semantics based on S that is always defined.

First of all, we will define some concepts *w.r.t.* the notion of generalized S model.

Definition 9 *Let S be a logic programming semantics, P be a logic program and A be a set of atoms (called abductives) such that $A \subseteq \mathcal{L}_P$.*

- *We say that M_B is a generalized S model of P* w.r.t. *A if $M \in S(P \cup B)$ where $B \subseteq A$ and $M \subseteq \mathcal{L}_P$.*
- *We define a partial order between generalized S models (*w.r.t. *A) of a program according to the set inclusion* w.r.t. *the subindex B. We say that M is a minimal generalized S model of P* w.r.t. *A if there exists a set of atoms B, such that M_B is a generalized S model of P* w.r.t. *A and M_B is minimal* w.r.t. *the partial order just defined.*
- *We write S^* to denote the minimal generalized S semantics, where $A = \mathcal{L}_P$. Namely $S^*(P)$ is the collection of minimal generalized S models of P* w.r.t. *$\mathcal{L}_P$.*

Observe that in our definition we are not instantiating the definition to a particular logic programming semantics.

The concept of generalized S model is closely related to the semantics of *abductive logic programming* [18, 17], in particular to the concept of *generalized answer set*. It

has also been explored for different logic programming approaches as in [1, 27]. For instance, the authors in [27] consider two partial order relations between the generalized models for defining minimal generalized models, one by considering the set inclusion *w.r.t.* the subindexes of the generalized models (as in Definition 9) and another one *w.r.t.* the cardinality of the subindexes of the generalized models[3].

In order to illustrate Definition 9, take the program C:

$$p \leftarrow \neg p.$$

We can see that $Stable(C)$ is undefined, however $Stable^*(C) = \{\{p\}\}$. Note that $\{p\}_{\{p\}}$ is the unique generalized stable model of C. This is because $\{p\}$ is a stable model of $C \cup \{p\}$. Moreover, $\{p\}$ is the unique minimal generalized model of C. Observe also that $Pstable(C) = Pstable^*(C) = \{\{p\}\}$.

Take the program D:

$$a \leftarrow \neg b.$$
$$b \leftarrow \neg c.$$
$$c \leftarrow \neg a.$$

Observe that $Pstable(D)$ is undefined; however, one can see that $\{a, b\}$ is a pstable model of $P \cup \{a\}$, $\{b, c\}$ is a pstable model of $P \cup \{b\}$ and $\{c, a\}$ is a pstable model of $P \cup \{c\}$. Since the models $\{a, b\}_{\{a\}}$, $\{b, c\}_{\{b\}}$, $\{c, a\}_{\{c\}}$ are the three minimized generalized pstable models of D, $Pstable^*(D) = \{\{a, b\}, \{b, c\}, \{a, c\}\}$. The same situation happens to $Stable$. $Stable(D)$ is undefined; however, $Stable^*(D) = \{\{a, b\}$, $\{b, c\}$, $\{a, c\}\}$.

The following lemma insures that any semantics induced by Definition 9 will be defined — the proof of the lemma is immediate thanks to the basic property defined at the beginning of this section.

Lemma 1 *For every semantics S and program P, $S^*(P)$ is defined.*

One important property of the semantics induced by Definition 9 is that the concept of generalized model will be important only in the case that the initial semantics S is undefined.

[3] In [27], a generalized model is called a $\mathcal{L}$-completion.

Lemma 2 *For every semantics S and program P. If $S(P)$ is defined then $S^*(P) = S(P)$.*

Proof: If $S(P)$ is defined then it is clear that $M \in S(P)$ iff $M_{\{\}}$ is a minimal generalized S model of P. ■

This lemma insures that whenever a semantics S is defined, it will be the same to S^* *e.g.*, $Pstable(C) = Pstable^*(C) = \{\{p\}\}$. The following lemma makes some observations *w.r.t.* MM, $RevStable$, $Stable$, $Pstable$ and their induced semantics based on the concept of generalized model.

Lemma 3 *MM and MM^* are the same semantics. $RevStable$ and $RevStable^*$ are the same semantics. $Stable$ is different from $Stable^*$. $Pstable$ is different from $Pstable^*$. $Stable^*$, $Pstable^*$, and MM^* are 3 different semantics.*

Proof: MM is the same as MM^* follows because MM is always defined (Lemma 2). For the same reason $RevStable$ and $RevStable^*$ are the same semantics. Note that program C already defined shows that $Stable$ is different from $Stable^*$. Program D shows that $Pstable$ is different from $Pstable^*$.

Consider Program E:

$$a \leftarrow \neg b.$$
$$b \leftarrow \neg a.$$
$$p \leftarrow \neg b.$$
$$p \leftarrow \neg p.$$

$Stable^*(E) = Stable(E) = \{\{p, a\}\}$. $Pstable^*(E) = Pstable(E) = \{\{p, a\}, \{p, b\}\}$. $MM^*(E) = MM(E) = \{\{p, a\}, \{p, b\}\}$. This program shows that $Pstable^*$ is different from $Stable^*$ and that MM^* is different from $Stable^*$

Consider Program F:

$$a \leftarrow \neg b.$$
$$b \leftarrow \neg a.$$
$$u \leftarrow a.$$
$$x \leftarrow \neg y, u.$$
$$y \leftarrow \neg z, u.$$
$$z \leftarrow \neg x, u.$$

One can see that $Pstable^*(F) = \{\{b\}\}$ and $MM^*(F) = \{\{b\}, \{a, u, x, y\}, \{a, u, x, z\}, \{a, u, y, z\}\}$. Hence, this program shows that $Pstable^*$ is different from MM^*. ■

3.2 Constructing relevant semantics

It is sometimes desirable that a semantics satisfies the relevance property. Relevance is a fundamental property when we are interested in defining the semantics of a split logic program. Given a semantics S we show how to construct a relevant semantics based on S that we denote by S^r.

We assume in this subsection the following property:

> *For every program, if an atom occurs in a program then it should also occur in the head of some rule.*

Later, we show how to deal with programs that do not satisfy this property. Given Q and R both sets of interpretations, we define

$$Q * R := \{M_1 \cup M_2 | M_1 \in Q, M_2 \in R\}$$

Definition 10 *Let S be a semantics that is always defined. We define the associate S^r semantics recursively as follow: Given a program P of order 0, $S^r(P) = S(P)$. For a program P of order $n > 0$ we define*

$$S^r(P) = \bigcup_{M \in S(P_0)} \{M\} * S^r(R(P \setminus P_0, \langle M; \mathcal{L}_{P_0} \setminus M \rangle))$$

For the reader who knows $STABLE^{rel}$'s definition presented in [12], we want to point out that S^r is similar to $STABLE^{rel}$ *w.r.t.* the relevance property; however, $STABLE^{rel}$ has a skeptical construction and S^r has a construction by scenarios.

Consider the program E defined before. We illustrate our definition to compute $Stable^{*^r}(E)$. Recall that E is:

$$a \leftarrow \neg b.$$
$$b \leftarrow \neg a.$$
$$p \leftarrow \neg b.$$
$$p \leftarrow \neg p.$$

Then E_0 is:

$$a \leftarrow \neg b.$$
$$b \leftarrow \neg a.$$

$Stable(E_0) = \{\{a\}, \{b\}\}$. Since $Stable(E_0)$ has two models, there are two cases to consider:

1. First, consider M be $\{a\}$. We can see that $E \setminus E_0$ is:

   $$p \leftarrow \neg b.$$
   $$p \leftarrow \neg p.$$

   Moreover, we can see that $R(E \setminus E_0, \langle M; \mathcal{L}_{E_0} \setminus M\rangle)$ is:

   $$p.$$
   $$p \leftarrow \neg p.$$

   Hence, $S^r(R(E \setminus E_0, \langle M; \mathcal{L}_{E_0} \setminus M\rangle))$ is: $\{\{p\}\}$. So, $\{M\} * S^r(R(E \setminus E_0, \langle M; \mathcal{L}_{E_0} \setminus M\rangle))$ is: $\{\{p, a\}\}$.

2. Now, consider M be $\{b\}$. In this case, we can see that $R(E \setminus E_0, \langle M; \mathcal{L}_{E_0} \setminus M\rangle)$ is:

   $$p \leftarrow \neg p.$$

   Observe that $Stable^*$ of this reduced program is $\{\{p\}\}$. Hence, $S^r(R(E \setminus E_0, \langle M; \mathcal{L}_{E_0} \setminus M\rangle))$ is: $\{\{p\}\}$. So, $\{M\} * S^r(R(E \setminus E_0, < M; \mathcal{L}_{E_0} \setminus M >))$ is: $\{\{p, b\}\}$.

Therefore $Stable^{*^r}(E) = \{\{p, a\}, \{p, b\}\}$.

3.2.1 The general case.

We have shown how to construct a relevant semantics for programs where every atom in the signature of the program occurs in the head of some rule of that program. Now, why is it important that every atom in the signature of the program must occur in the head of some rule of that program? In order to answer this question, let us consider the program J:

$$a \leftarrow \neg b.$$

and let S be a logic programming semantics. Now, let us suppose we want to infer $S^r(J)$. Hence, the first step, in order to infer $S^r(J)$, is to split J into its components. Since a dependents on b, but d does not dependent on a, $[b] \leq_P [a]$. This means that J has to be split into two components: J_0 and J_1. Remember that J_0 will contain all the clauses whose head is the atom b and J_1 will contain all the clauses whose head is the atom b. It is obvious that J_0 is an empty component, but this is a problem because we cannot apply Definition 10 in order to infer $S^r(J)$. A simple way to avoid this problem is to add the tautology $b \leftarrow b$ to J. For instance, let K be the program:

$$b \leftarrow b.$$
$$a \leftarrow \neg b.$$

Observe that like J, K has also two components; however, none of them is empty. This idea will suggest a natural generalization of Definition 10 in order to infer S^r for any logic program.

Definition 11 *Let P be a normal logic program. We define*

$$S^r(P) = S^r(P \cup \{x \leftarrow x : x \in \mathcal{L}_P \setminus HEAD(P)\})$$

This completes our construction of relevant semantics. Consider again the program J:

$$a \leftarrow \neg b.$$

Since b does not occur in the head of some rule, we add the tautology $b \leftarrow b$, obtaining K:

$$b \leftarrow b.$$
$$a \leftarrow \neg b.$$

Now we can proceed as before to obtain our intended semantics. $MM^{*^r}(K) = Stable^{*^r}(K) = Pstable^{*^r}(K) = \{\{a\}\}$. Hence, $MM^{*^r}(J) = Stable^{*^r}(J) = Pstable^{*^r}(J) = \{\{a\}\}$.

Observe that the logic programming semantics induced by Definition 11 are different *w.r.t.* the logic programming semantics induced by Definition 9. For instance, the following lemmas formalize the differences *w.r.t.* some semantics.

Lemma 4 *$Stable^*$ is different from $Stable^{*^r}$. $Pstable^*$ is different from $Pstable^{*^r}$.*

Proof: Program E shows that $Stable^*$ is different from $Stable^{*^r}$. Program F shows that $Pstable^*$ is different from $Pstable^{*^r}$.

■

Lemma 5 *$Stable^{*^r}$, $Pstable^{*^r}$, and MM^{*^r} are 3 different semantics.*

Proof:

Consider Program L:

$$a \leftarrow \neg b.$$
$$b \leftarrow \neg a.$$
$$p \leftarrow \neg b.$$
$$p \leftarrow \neg p.$$
$$b \leftarrow \neg p.$$

Then $Stable^{*^r}(L) = \{\{p, a\}\}$. However $Pstable^{*^r}(L) = \{\{p, a\}, \{p, b\}\}$. Hence $Stable^{*^r}$ is different from $Ptable^{*^r}$. This example also show that $Stable^{*^r}$ is different from MM^{*^r}.

Consider Program R:

$$x \leftarrow \neg y.$$
$$y \leftarrow \neg z.$$
$$z \leftarrow \neg x.$$
$$x \leftarrow \neg u.$$
$$d \leftarrow \neg z.$$
$$u \leftarrow \neg d.$$

This example shows that $Stable^{*^r}$ is different from MM^{*^r}. $Stable^{*^r}(R) = \{\{x, y, d\}\}$. However $\{u, x, z\} \in MM^{*^r}(R)$. This example also shows that $Pstable^{*^r}$ is different from MM^{*^r}, since $Pstable^{*^r}(R) = Stable^{*^r}(R)$.

■

As we commented in Section 2.1, the stratified logic programs define a class of logic programs which have interesting properties *w.r.t.* the search for nice declarative semantics. In particular we can observe that *Stable*, *Pstable*, MM^{*^r}, as well as all our refined versions for *Stable* and *Pstable* agree with respect to the class of stratified programs.

Lemma 6 *Let P be a stratified program. Hence,*
$Stable(P) = Pstable(P) = MM^{*^r}(P) = Stable^{*^r}(P) = Pstable^{*^r}(P)$

Proof:

The proof is by induction on the number of components of P *e.g.*, $P = P_0 \cup \cdots \cup P_n$.

Base Step By definition of stratified program, one can see that if $P = P_0$, then P is a *definite program i.e.* P does not have negative literals. Hence, $MM(P) = Stable(P) = Pstable(P)$. Since $MM(P)$ is always defined, $Stable(P) = Pstable(P) = MM^{*^r}(P) = Stable^{*^r}(P) = Pstable^{*^r}(P)$.

Inductive Step Now, let us suppose that P has n components.

Let us suppose that it is true that $Stable(P_k) = Pstable(P_k) = MM^{*^r}(P_k) = Stable^{*^r}(P_k) = Pstable^{*^r}(P_k)$ for some $k < n$.

By definition, $S^r(P) = \bigcup_{M \in S(P_0)} \{M\} * S^r(R(P \setminus P_0, \langle M; \mathcal{L}_{P_0} \setminus M \rangle))$

Observe that since P is a stratified program, hence for any $M \in S^r(P_k)$, $R(P_{k+1} \setminus P_k, \langle M; \mathcal{L}_{P_{k+1}} \setminus M \rangle)$ is a *definite program*. Therefore by inductive hypothesis, $Stable(P) = Pstable(P) = MM^{*^r}(P) = Stable^{*^r}(P) = Pstable^{*^r}(P)$.

■

As final result of this subsection, we can introduce the following lemma whose proof is immediate by Lemma 1.

Lemma 7 *For every semantics S and program P, S^{*^r} is defined.*

This lemma insures that any logic programming semantics induced by Definition 9 and Definition 11 will be defined for any logic program.

Before to finish this section, we want to remember to the reader that there is an important preprocessing that must be applied to any logic program in order to apply the semantics introduced in this section. In Remark 3, we pointed out that we have assumed that our programs do not include *non-trivial tautologies*. This preprocessing has an important role in logic programming semantics as MM^{*^r}. For instance, let P be the following logic program:

$$a \leftarrow \neg b.$$
$$b \leftarrow a, \neg a. \quad \text{(non-trivial tautology)}$$

Observe that the atom b is implied by the formula $a \wedge \neg a$; hence, one can expect not to infer the atom b from P. However, one can see that P has a minimal model which contains the atom b. This means that $MM^{*^r} = \{\{a, b\}\}$; therefore, MM^{*^r} is inferring an unexpect model. It is worth to comment that this problem does not happen with semantics as $Stable$ and $Pstable$ — these semantics only infer from P the model $\{a\}$.

In order to avoid some kind of unexpect models, one can remove from P any *non-trivial tautology*. Observe that the clause $b \leftarrow a, \neg a$ is a non-trivial tautology. Hence by removing this non-trivial tautology from P, we can get the logic program P':

$$a \leftarrow \neg b.$$

In this case like the semantics $Stable$ and $Pstable$, MM^{*^r} only infers the model $\{a\}$ from P'.

It worth to comment that the constructive definition of relevant semantics presented in Definition 11 only considers trivial tautologies in order to insure that any atom of a given program appears in the head of some clause.

4 Construction of abstract argumentation semantics

In the previous section, we have defined a recursive general schema for constructing new logic programming semantics. As we mentioned in the introduction, we are interested in constructing new abstract argumentation semantics based on logic programming semantics. In this section, we will show how to take advantages of our approach for building new abstract argumentation semantics.

It is quite obvious that in order to regard an argumentation framework as a logic program, we require a function mapping which constructs a logic program from an argumentation framework. Hence let $\mathcal{M}$ be a mapping from the class of argumentation frameworks ($\mathcal{AF}_{AR}$) to the class of logic programs ($Prog_{\mathcal{L}}$). $\mathcal{M}$ assigns to every argumentation framework AF a logic program P. We are going to denote the image of AF under $\mathcal{M}$ as P_{AF} .

Now, we will define how any logic programming semantics can induce a *candidate* abstract argumentation semantics under a particular mapping.

Definition 12 *Let $AF = \langle AR, attacks \rangle$ be an argumentation framework and S be any logic programming semantics. The semantics S induces the candidate abstract argumentation semantics $S^{\mathcal{M}}_{Arg}$ as follows:*

$$S^{\mathcal{M}}_{Arg}(AF) = f(S^{*^r}(P_{AF}))$$

such that f is a mapping from $2^{\mathcal{L}_{P_{AF}}}$ to 2^{AR}.

Observe that for each model of P_{AF} the mapping f is defining an extension for the argumentation framework AF. Informally speaking we can say that $S^{\mathcal{M}}_{Arg}$ is the *candidate abstract argumentation semantics* induced by the logic programming semantics S under the mapping $\mathcal{M}$.

The following lemma is immediate thanks to Lemma 1 and Lemma 7.

Lemma 8 *For every logic programming semantics S, the candidate abstract argumentation semantics $S^{\mathcal{M}}_{Arg}$ is always defined*

In order to illustrate our general schema for constructing abstract argumentation semantics, let us introduce a simple mapping to regard an argumentation framework as a normal logic program. In this mapping, we use the predicate *d(x)*, where the intended meaning of *d(x)* is: "the argument *x* is defeated". This means that the argument x is attacked by an acceptable argument.

Definition 13 *Let $AF = \langle AR, attacks \rangle$ be an argumentation framework. We define:*

$$P_{AF} = \bigcup_{a \in AR} (\bigcup_{b:(b,a) \in attacks} d(a) \leftarrow \neg d(b))$$

The only condition which is captured by this program is that any argument will be defeated when anyone of its adversaries is not defeated. Observe that essentially P_{AF} is capturing the basic principle of *conflict-freeness* (see Definition 5); hence, one can insure that any candidate abstract argumentation semantics induced by P_{AF} at least will satisfy the principle of conflict-freeness. It worth to comment that according to Baroni

and Giacomin in [3], the principle of conflict-freeness is the minimal requirement to be satisfied by any argumentation semantics.

In order to illustrate the mapping P_{AF}, let AF be the argumentation framework of Figure 2. We can see that P_{AF} is:

$$\begin{array}{lll} d(a) \leftarrow \neg d(b). & d(d) \leftarrow \neg d(a). & d(e) \leftarrow \neg d(d). \\ d(b) \leftarrow \neg d(c). & d(d) \leftarrow \neg d(b). & \\ d(c) \leftarrow \neg d(a). & d(d) \leftarrow \neg d(c). & \end{array}$$

Two relevant properties of the mapping P_{AF} are:

1. The stable models of P_{AF} characterize the stable argumentation semantics (Definition 8) (see [13, 20] for details) and
2. by Theorem 17 of [13] one can see that the well founded model [14] of P_{AF} characterizes the grounded semantics defined in [13] (see [20] for details).

In order to construct our abstract argumentation semantics induced by any logic programming semantics and the mapping P_{AF}. Let us point out that $\mathcal{L}_{P_{AF}} = \{d(a) | a \in AR\}$. Hence given any logic programming semantics S and an argumentation framework AF:

$$f(S(P_{AF})) = \bigcup_{M \in S(P_{AF})} \{a | d(a) \in \mathcal{L}_{P_{AF}} \setminus M\}$$

We have already commented that the stable models of P_{AF} characterize the stable extensions of AF. For instance, we can see that the program P_{AF} of the argumentation framework of Figure 2 has no stable models, this means that this argumentation framework has no stable extensions. However, let us consider the argumentation semantics $Stable^{*^r}_{Arg}$ *w.r.t.* AF.

First of all, we have to compute the semantics $Stable^{*^r}$ *w.r.t.* P_{AF}. It is not difficult to see that

$$Stable^{*^r}(P_{AF}) = \{\{d(a), d(b), d(d)\}, \{d(b), d(c), d(d)\}, \{d(a), d(c), d(d)\}\}$$

Then we can see that

$$f(Stable^{*^r}(P_{AF})) = \{\{a, e\}, \{b, e\}, \{c, e\}\}$$

This means that

$$Stable^{*^r}_{Arg}(AF) = \{\{a, e\}, \{b, e\}, \{c, e\}\}$$

Observe that the extensions suggested by the argumentation semantics $Stable^{*^r}_{Arg}$ are the same to the extensions suggested by the semantics $CF2$ introduced by P. Baroni *et al*, in [4]. In fact $MM^{*^r}_{Arg}$ and $Pstable^{*^r}_{Arg}$ also coincide with CF2 in this example.

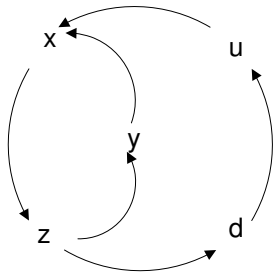

Figure 5: Graph representation of the argumentation framework $AF = \langle\{x, y, z, u, d\}, \{(x, z), (z, y), (y, x), (u, x), (z, d), (d, u)\}\rangle$.

Now let us consider the argumentation framework of Figure 5. It is not difficult to see that P_{AF} *w.r.t.* the argumentation framework of Figure 5 is:

$$d(x) \leftarrow \neg d(y).$$
$$d(y) \leftarrow \neg d(z).$$
$$d(z) \leftarrow \neg d(x).$$
$$d(x) \leftarrow \neg d(u).$$
$$d(d) \leftarrow \neg d(z).$$
$$d(u) \leftarrow \neg d(d).$$

Observe that P_{AF} is exactly (modulo notation) the program R which appears in the proof of Lemma 5. Since $Stable^{*^r}(P_{AF}) = \{\{d(x), d(y), d(d)\}\}$, therefore $Stable^{*^r}_{Arg}(AF) = \{\{u, z\}\}$. We can see that $Stable^{*^r}_{Arg}$ coincides with the preferred semantics, the stable argumentation semantics and $Pstable^{*^r}_{Arg}$ in this example. Now let us compute the argumentation semantics $MM^{*^r}_{Arg}$. Since $MM^{*^r}(P_{AF}) =$

$$\{\{d(y), d(z), d(u)\}, \{d(x), d(z), d(d)\}, \{d(u), d(x), d(z)\}, \{d(x), d(y), d(d)\}\}$$

then $MM^{*^r}_{Arg}(AF) = \{\{x, d\}, \{y, u\}, \{y, d\}, \{z, u\}\}$. Notice that $MM^{*^r}_{Arg}$ coincides again with the argumentation semantics CF2.

We want to remark that the argumentation semantics $MM^{*^r}_{Arg}$ is really close to the argumentation semantics CF2. Let us consider another example where $MM^{*^r}_{Arg}$ and CF2 coincide. Let AF be the argumentation framework of Figure 6[4].

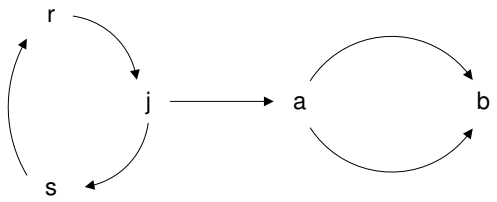

Figure 6: Graph representation of the argumentation framework $AF = \langle\{r, s, j, a, b\}, \{(r, j), (j, s), (s, r), (j, a), (a, b), (b, a)\}\rangle$.

In order to compute $MM^{*^r}_{Arg}(AF)$, first of all we have to map AF to P_{AF}:

$$
\begin{aligned}
d(r) &\leftarrow \neg d(s).\\
d(s) &\leftarrow \neg d(j).\\
d(j) &\leftarrow \neg d(r).\\
d(a) &\leftarrow \neg d(j).\\
d(a) &\leftarrow \neg d(b).\\
d(b) &\leftarrow \neg d(a).
\end{aligned}
$$

As we can see, $MM^{*^r}(P_{AF})$ has five models which are:

$$\{\{d(r), d(s), d(a)\}, \{d(j), d(r), d(a)\}, \{d(j), d(r), d(b)\},$$
$$\{d(s), d(j), d(a)\}, \{d(s), d(j), d(b)\}\}$$

This means that:

$$MM^{*^r}_{Arg}(AF) = \{\{j, b\}, \{s, b\}, \{s, a\}, \{r, b\}, \{r, a\}\}$$

As commented before, $MM^{*^r}_{Arg}(AF) = CF2(AF)$.

[4]This argumentation framework was introduced in [4].

5 Related work

As far this paper, we have commented that our approach for building new argumentation semantics is close related to Baroni *et al*'s approach presented in [4]. In this section, we will point out some of the main common points between our approach and Baroni *et al*'s approach.

As we know, Baroni *et al*'s approach is based on a solid concept in graph theory which is a *strongly connected component* (SCC)[5]. Based on the fact that any argumentation framework AF can be represented by a directed graph, Baroni *et al*'s approach consider the strongly connected components of AF in order to introduce a recursive definition for argumentation semantics. The behavior of the argumentation semantics will be mainly influenced by

- a basic argumentation semantics-specific function. This function is a pattern of inference of arguments *e.g.*, conflict-free sets, admissible sets, *etc.*

In §2, we saw that any logic program induces a notion of $dependency$ between atoms from $\mathcal{L}_P$. In fact, we saw that this dependency can define classes of atoms such that theses classes of atoms form a partial order which encodes the dependencies existing among the atoms. By considering theses classes of atoms, one can break any logic program into a disjoint union of subprograms. Based on this disjoint union of subprograms, we introduce a recursive definition for logic programming semantics.

In §4, we saw that by considering an argumentation framework as a logic program, one can use the recursive definition for logic programming semantics presented in §3 in order to induce candidate argumentation semantics. The behavior of the induce candidate argumentation semantics will be mainly influenced by two variables:

- the representation of an argumentation framework into a logic program and
- the basic logic programming semantics function *e.g.*, the minimal model semantics, the stable model semantics, the pstable semantics, *etc.*

Unlike to Baroni *et al*'s approach which always starts with the same representation of an argumentation framework into a directed graph, our approach allows to use

[5] A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex. The strongly connected components (SCC) of a directed graph are its maximal strongly connected subgraphs [10].

different representations of an argumentation framework in terms of logic programs. These different representations could induce different dependencies of atoms of the given logic program; therefore we can break a logic program into different disjoint subprograms. This means that we can break an argumentation framework into different components.

Even though one can consider different representations of an argumentation framework in terms of logic programs, we can consider single mappings of an argumentation framework into a logic program in order to split an argumentation framework similarly as it is done by Baroni *et al*'s approach. For instance, let us consider the argumentation framework AF of Figure 2. Observe that the graph representation of AF has three strongly connected components:

$$SCC^0_{AF} = \{a, b, c\}$$
$$SCC^1_{AF} = \{d\}$$
$$SCC^2_{AF} = \{e\}$$

By considering the relation of attack between sets of arguments and the so called *directionality principle* [4], Baroni *et al*'s approach defines a partial order between theses strongly connected components (let us denote this relation as $\leq_{SCC}$). $\leq_{SCC}$ defines the following relations between the above strongly connected components:

$$\{a, b, c\} \leq_{SCC} \{d\} \leq_{SCC} \{e\} \tag{3}$$

Essentially, these relations capture that SCC^0_{AF} attacks SCC^1_{AF} and SCC^1_{AF} attacks SCC^2_{AF}. Since there is not a strongly connected component that attacks SCC^0_{AF}, SCC^0_{AF} is called *initial*.

Now, let us consider the representation of AF in terms of normal logic programs according to the mapping presented in Definition 13. We can see that P_{AF} is:

$$d(a) \leftarrow \neg d(b). \qquad d(d) \leftarrow \neg d(a). \qquad d(e) \leftarrow \neg d(d).$$
$$d(b) \leftarrow \neg d(c). \qquad d(d) \leftarrow \neg d(b).$$
$$d(c) \leftarrow \neg d(a). \qquad d(d) \leftarrow \neg d(c).$$

Observe that this program induces the following classes of atoms:

$[d(a)] = \{d(a), d(b), d(c)\}$ $[d(d)] = \{d(d)\}$ $[d(e)] = \{d(e)\}$
$[d(b)] = \{d(a), d(b), d(c)\}$
$[d(c)] = \{d(a), d(b), d(c)\}$

such that

$d(a)$ is of order 0 $d(d)$ is order 1 $d(e)$ is of order 2
$d(b)$ is of order 0
$d(c)$ is of order 0

This means that we have the following relations of atoms:

$$\{d(a), d(b), d(c)\} \leq_p \{d(d)\} \leq_p \{d(e)\} \tag{4}$$

Observe that we have very similar relations between the partial orders: $\leq_{SCC}$ (3) and $\leq_p$ (4). In Baroni *et al*'s approach, the partial order $\leq_{SCC}$ is used for splitting an argumentation framework in order to define a recursive construction of argumentation semantics. In our approach, the partial order $\leq_p$ is used for splitting a logic program in order to define a recursive construction of logic programming semantics. If the given logic program captures an argumentation framework AF (as it is done by $\mathcal{P}_{AF}$), $\leq_p$ will support the construction of an recursive construction of argumentation semantics as well.

In Baroni *et al*'s approach, once an argumentation framework is partitioned into its strongly connected components and the relation between them is defined, the possible choices for extensions within each *initial strongly connected component* are determined using *a basic argumentation semantics-specific function e.g.*, conflict-free sets, admissible sets, *etc.*, which returns the extensions of the argumentation frameworks consisting of a single strongly connected components. For instance, let us consider the initial strongly connected component ($SCC^0_{AF} = \{a, b, c\}$) of the argumentation framework AF of Figure 2. SCC^0_{AF} will define the following subargumentation framework:

$$AF_{SCC^0_{AF}} = \langle \{a, b, c\}, \{(a, c), (c, b), (b, a)\} \rangle$$

Now let us consider as basic argumentation semantics-specific function, the function which returns all the possible conflict-free sets ($\mathcal{CF}$) of an argumentation framework. Hence we can see that $\mathcal{CF}(AF_{SCC^0_{AF}}) = \{\{a\}, \{b\}, \{c\}\}$.

Once a basic argumentation semantics-specific function S_{arg} was applied to all the initial strongly connected component of an argumentation framework, for each choice determined by S_{arg}

- the nodes directly attacked within subsequent strongly connected components are suppressed and
- a distinction between defended and undefended arguments is taken into account.

This changes are done according to the *reinstatement principle*. This basic principle prescribes that the arguments defeated by an extension E play no role in the selection of the arguments to be included in E [4].

In our approach, once the partial order $\leq_p$ is defined in terms of $\mathcal{L}_{P_{AF}}$, the program P_{AF} is split into its components. For instance, P_{AF} is splint as follows:

P_0	P_1	P_2
$d(a) \leftarrow \neg d(b).$	$d(d) \leftarrow \neg d(a).$	$d(e) \leftarrow \neg d(d).$
$d(b) \leftarrow \neg d(c).$	$d(d) \leftarrow \neg d(b).$	
$d(c) \leftarrow \neg d(a).$	$d(d) \leftarrow \neg d(c).$	

Observe that P_0 can be constructed from $AF_{SCC^0_{AF}}$. In fact $P_0 = \mathcal{P}_{AF_{SCC^0_{AF}}}$. Let us consider the minimal model semantics MM as basic logic programming semantics function in order to illustrate the relations of our approach with Baroni *et al*'s approach. The basic logic programming semantics function is applied recursively as it is presented in Definition 10. Now observe that

$$MM(P_0) = \{\{d(a), d(b)\}, \{d(b), d(c)\}, \{d(a), d(c)\}\}$$

therefore $\mathcal{F}(MM(P_0)) = \{\{a\}, \{b\}, \{c\}\}$. This means that $\mathcal{F}(MM(P_0)) = \mathcal{CF}(AF_{SCC^0_{AF}})$.

Like Baroni *et al*'s approach which reduces recursively the original argumentation framework by considering the results of the basic argumentation semantics-specific function, our approach applies a reduction to a logic program.

As we can see there are several key points that we can identified in common between Baroni *et al*'s approach and our approach, some them are:

1. Both approach consider an partial order for splitting the representation of an argumentation framework *i.e.* graph representation or logic program.

2. Both approach use a recursive definition for constructing the desired semantics (argumentation semantics / logic programming semantics).

A deep analysis is required in order to understand more about the relation between Baroni *et al*'s approach and our approach. In fact some interesting questions that we are going to consider in our future work are: What is the class of argumentation semantics that can be characterized in both approaches? What are the logic programming semantics that are more useful for building argumentation semantics? Which mappings of an argumentation framework into a logic program defines useful argumentation semantics?

With respect to the above questions, we know, by the moment, that the mapping of Definition 13 is a practical mapping that by considering the logic programming MM^{*^r} is able to suggest an argumentation semantics similar to CF2. It is worth to comment that CF2 is one of the most accepted argumentation semantics builded under the Baroni *et al*'s approach [3].

As final comment, we want to comment that any of the candidate argumentation semantics S_{arg} suggested under our approach can be explored their non-monotonic properties in terms of the logic programming semantics which induces S_{arg}. This is a relevant feature of our approach since many of the new argumentation semantics are only motivated by particular examples; hence, the identification of the non-monotonic reasoning properties, that a particular argumentation semantics satisfies, takes relevance in order to support the well-behaviour of an argumentation semantics.

6 Conclusions

Authors as Baroni *et al*, have suggested that in order to overcome Dung's abstract argumentation semantics problems, it is necessary to define flexible argumentation semantics which are not necessarily based on admissible sets [4]. For instance, Baroni *et al*, have pointed out that in any admissibility-based semantics odd-length cycles admit only the empty extension. Based on the fact that logic programming offers a wide liberty for modeling knowledge, we can construct abstract argumentation semantics by specifying the basic conditions that our new argumentation semantics must satisfy. For instance, the mapping introduced in Definition 13 only captures the restriction that

any argument will be defeated when anyone of its adversaries is not defeated. This single mapping is enough for characterizing argumentation semantics as the grounded semantics and the stable argumentation semantics. In fact, we also defined the abstract argumentation semantics $MM^{*^r}_{Arg}$ which is similar to CF2 — the semantics $MM^{*^r}_{Arg}$ is constructed under the mapping of Definition 13 and the logic programming semantics MM^{*^r}.

By considering the idea that argumentation semantics can be viewed as a special form of logic programming semantics with negation as failure, we introduce a general recursive approach for defining a new family of logic programming semantics which induce a new family of abstract argumentation semantics.

We have in mind that our new logic programming semantics satisfy the following suitable properties:

1. They should be always defined.
2. They should satisfy the relevance property.
3. They should agree with stable models semantics for the class of stratified programs.
4. They should be useful to model argumentation problems.

To study abstract argumentation semantics in terms of logic programming semantics is not new. In fact, since Dung's approach was introduced in [13], Dung proved that the grounded semantics can be characterized by the well-founded semantics (WFS) [14] and the stable argumentation semantics can be characterized by the stable model semantics [15]. This result 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.

Although the study of abstract argumentation semantics in terms of logic programming semantics has important implications, the only semantics that were studied before 2006 were the grounded semantics and the stable semantics. Nowadays, we know

that the preferred semantics can be characterized in terms of minimal models [22, 23], pstable models [9] and stable models [23].

In the process of characterizing an abstract argumentation semantics in terms of logic programming semantics, we have seen that an abstract argumentation semantics can be extended by considering logic programming semantics in order to overcome some of their limitations. For instance in [24], we saw that the grounded semantics can be extended based on WFS's extensions in order to overcome some their problems *w.r.t. emptiness*. An interesting property of the study done in [24] is that rewriting systems can be used for defining a declarative calculus of argumentation frameworks which allow to describe the interactions of arguments.

To explore the properties of the family of the abstract argumentation semantics which are induced by our approach is an issue for argumentation research. In fact, it is part of our future research. It is worth mentioning that thanks to the properties that the logic programming semantics hold, we can study the argumentation semantics that are constructed under these logic programming semantics *e.g.*, Lemma 8.

Acknowledgement

We are grateful to Pietro Baroni for their useful comments.

References

[1] M. Balduccini and M. Gelfond. Logic Programs with Consistency-Restoring Rules. In P. Doherty, J. McCarthy, and M.-A. Williams, editors, *International Symposium on Logical Formalization of Commonsense Reasoning*, AAAI 2003 Spring Symposium Series, Mar 2003.

[2] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Cambridge, 2003.

[3] P. Baroni and M. Giacomin. On principle-based evaluation of extension-based argumentation semantics. *Artificial Intelligence.*, 171(10-15):675–700, 2007.

[4] P. Baroni, M. Giacomin, and G. Guida. SCC-recursiveness: a general schema for argumentation semantics. *Artificial Intelligence*, 168:162–210, October 2005.

[5] T. J. M. Bench-Capon and P. E. Dunne. Argumentation in artificial intelligence. *Artificial Intelligence*, 171(10-15):619–641, 2007.

[6] M. Caminada. Contamination in formal argumentation systems. In *BNAIC 2005 - Proceedings of the Seventeenth Belgium-Netherlands Conference on Artificial Intelligence, Brussels, Belgium, October 17-18*, pages 59–65, 2005.

[7] M. Caminada. Semi-Stable semantics. In P. E. Dunne and T. J. Bench-Capon, editors, *Proceedings of COMMA*, volume 144, pages 121–130. IOS Press, 2006.

[8] M. Caminada and C. Sakama. On the existence of answer sets in normal extended logic programs. In *ECAI*, pages 743–744, 2006.

[9] J. L. Carballido, J. C. Nieves, and M. Osorio. Inferring Preferred Extensions by Pstable Semantics. *Iberoamerican Journal of Artificial Intelligence (Inteligencia Artificial)*, to appear.

[10] T. H. Cormen, C. E. Leiserson, R. L. Riverst, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[11] J. Dix. A classification theory of semantics of normal logic programs: II. weak properties. *Fundam. Inform.*, 22(3):257–288, 1995.

[12] J. Dix and M. Müller. Partial evaluation and relevance for approximations of stable semantics. In *ISMIS*, volume 869 of *Lecture Notes in Computer Science*, pages 511–520. Springer, 1994.

[13] 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.

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

[15] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *5th Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[16] H. Jakobovits and D. Vermeir. Robust semantics for argumentation frameworks. *Journal of logic and computation*, 9(2):215–261, 1999.

[17] A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. In D. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook in Artificial Intelligence and Logic Programming, Volume 5*, pages 235–324. Oxford University Press, Oxford, 1998.

[18] A. C. Kakas and P. Mancarella. Generalized stable models: A semantics for abduction. In *ECAI*, pages 385–391, 1990.

[19] J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1987.

[20] J. C. Nieves. *Modeling arguments and uncertain information — A non-monotonic reasoning approach*. PhD thesis, Software Department (LSI), Technical University of Catalonia, 2008.

[21] J. C. Nieves and M. Osorio. Inferring Preferred Extensions by Pstable Semantics. In *Proceedings of the Latin-American Workshop on Non-Monotonic Reasoning (LA-NMR07) Workshop*, volume 286 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[22] J. C. Nieves, M. Osorio, and U. Cortés. Inferring preferred extensions by minimal models. In G. Simari and P. Torroni, editors, *Argumentation and Non-Monotonic Reasoning (LPNMR-07 Workshop)*, pages 114–124, Arizona, USA, 2007.

[23] 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.

[24] J. C. Nieves, M. Osorio, and U. Cortés. Studying the grounded semantics by using a suitable codification. Research report LSI-08-6-R, Universitat Politècnica de Catalunya, Software Department (LSI), Barcelona, Spain, January 2008.

[25] J. C. Nieves, M. Osorio, U. Cortés, I. Olmos, and J. A. Gonzalez. Defining new argumentation-based semantics by minimal models. In *Seventh Mexican International Conference on Computer Science (ENC 2006)*, pages 210–220. IEEE Computer Science Press, September 2006.

[26] M. Osorio, J. A. Navarro, J. R. Arrazola, and V. Borja. Logics with Common Weak Completions. *Journal of Logic and Computation*, 16(6):867–890, 2006.

[27] M. Osorio and J. C. Nieves. $W_{s,c}$-Stable Semantics for Propositional Theories. In *International Conferences of CIC'2001*, pages 319–328, México, 2001.

[28] L. M. Pereira and A. M. Pinto. Revised stable models - a semantics for logic programs. In *EPIA*, pages 29–42, 2005.

[29] J. L. Pollock. Justification and defeat. *Artif. Intell.*, 67(2):377–407, 1994.

[30] 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.

[31] D. van Dalen. *Logic and structure*. Springer-Verlag, Berlin, 3rd., aumented edition edition, 1994.