Fundamenta Informaticae XXI (2001) 1001–1025 1001
IOS Press

A Schema for Generating Relevant Logic Programming Semantics and its Applications in Argumentation Theory*

Juan Carlos Nieves
Universitat Politècnica de Catalunya
Software Department (LSI), Spain
jcnieves@lsi.upc.edu

Mauricio Osorio
Universidad de las Américas - Puebla. CENTIA. México
osoriomauri@googlemail.com

Claudia Zepeda
Benemérita Universidad Atónoma de Puebla
Facultad de Ciencias de la Computación. México
czepedac@gmail.com

Abstract. In the literature, there are several approaches which try to perform common sense reasoning. Among them, the approaches which have probably received the most attention the last two decades are the approaches based on logic programming semantics with negation as failure and argumentation theory. Even though both approaches have their own features, it seems that they share some common behaviours which can be studied by considering the close relationship between logic programming semantics and extension-based argumentation semantics.

In this paper, we will present a general recursive schema for defining new logic programming semantics. This schema takes as input any basic logic programming semantics, such as the stable model semantics, and gives as output a new logic programming semantics which satisfies some desired properties such as relevance and the existence of the intended models for every normal program. We will see that these new logic programming semantics can define candidate extension-based argumentation semantics. These new argumentation semantics will overcome some of the weakness of the

Address for correspondence: Universitat Politècnica de Catalunya. Software Department (LSI). c/Jordi Girona 1-3, E08034, Barcelona, Spain. jcnieves@lsi.upc.edu

*This document is an improved and extended version of the paper [38].

extension-based argumentation semantics based on admissible sets. In fact, we will see that some of these new argumentation semantics have similar behaviour to the extension-based argumentation semantics built in terms of strongly connected components.

Keywords: Non-monotonic reasoning, extension-based argumentation semantics and logic programming semantics, logic programming.

1. Introduction

In the non-monotonic reasoning community, it is well-accepted that the stable model semantics (also called answer set semantics) [26] represents a prominent approach for performing non-monotonic reasoning. In fact it has given place to a new approach of logic programming with *negation as failure*. Usually an answer set program can be seen as a specification of a problem where each stable model of a program P represents possible solutions to the problem. Nowadays, there are efficient solvers such as dlv, clasp, and smodel. The efficiency of these answer set solvers have increased the list of the stable model semantics' applications, *e.g.*, planning, bioinformatics, argumentation theory, etc.

Even though the stable model semantics enjoys a good reputation in the non-monotonic reasoning community, it is also well-known that the stable model semantics does not satisfy some desired properties pointed out by several authors [7, 17, 43, 46]. Among these properties we can mention the existence of intended models in some normal logic programs. Take for instance the basic program: $P = \{a \leftarrow \neg a, a \leftarrow a\}$. Note that this program is consistent from the point of view of classical logic, P has one model namely $\{a\}$. We follow the point of view of Schlipf [46], that proposes that logic programming with *negation as failure* should extend classical logic not replace it. Furthermore there are several approaches [7, 18, 20, 32, 39, 43, 46] which consider the above program consistent and that it should derive "a". For the stable semantics, the program P is inconsistent. Of course that this is not necessarily a weakness of the stable semantics. It all depends on the intended use of our logic programs. In the case of an approach motivated by *argumentation theory* it is desirable that normal programs (representing a dispute among arguments) always give an answer that infers the *winning* arguments in a dispute among arguments.

Argumentation theory has become an increasingly important and exciting research topic in Artificial Intelligence (AI), with research activities ranging from developing 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 [21], 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 arguments is supported by four extension-based argumentation semantics: *stable semantics*, *preferred semantics*, *grounded semantics*, and *complete semantics*[1]. The central notion of these semantics is the *acceptability of the arguments*. Even thought each of these argumentation semantics represents different patterns of selection of arguments, all of them are based on the basic concept of *admissible set*. Informally speaking, an admissible set presents a coherent and defendable point

[1]It is worth mentioning that recently Dung *et al*, [22] defined another argumentation semantics which is called *ideal semantics*. This semantics can be regarded as an intermediate semantics between the grounded and preferred semantics.

of view in a conflict between arguments. According to Bench-Capon and Dunne [5], 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 been illustrated in the literature [4, 5, 9, 45]. Authors such 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[2]. 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 argument a is attacked by itself and argument b is attacked by argument a. Intuitively, some authors as Prakken and Vreeswijk [45] suggest that one can expect that argument b can be considered as an acceptable argument since it is attacked by argument a which is attacked by itself. However, the preferred semantics is unable to infer 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 argument b as acceptable.

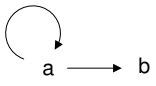

Figure 1. Graph representation of the argumentation framework $AF = \langle\{a, b\}, \{(a, a), (a, b)\}\rangle$.

Another interesting argumentation framework which has been commented on in the literature [45, 4] is presented in Figure 2. The preferred semantics *w.r.t.* this argumentation framework is only able to infer the empty set. Some authors, such as Prakken and Vreeswijk [45], Baroni *et al*[4], suggest that argument e can be considered as an acceptable argument since it is attacked by argument d which is attacked by three arguments: a, b, c. Observe that the arguments a, b and c form a cycle of attacks.

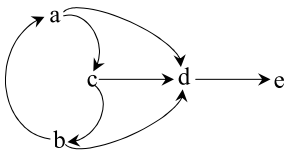

Figure 2. Graph representation of the argumentation framework $AF = \langle\{a, b, c, d, e\}, \{(a, c), (c, b), (b, a), (a, d), (c, d), (b, d), (d, e)\}\rangle$.

The stable argumentation semantics defined by Dung in [21] is also considered as another proper argumentation semantics. However, this semantics has been criticized by some authors such as Bench-

[2]This argumentation framework has received special attention in the literature in order to commented the problem of the self-defeated arguments [44, 45] and to point out some of the problems of the Dung's argumentation semantics [4].

Capon and Dunne [5], Caminada [9] because there are argumentation frameworks where no stable extensions exists, as in the case of odd-length cycles. In fact, the argumentation frameworks of Figure 1 and Figure 2 are two examples where the given argumentation frameworks has no stable extensions.

Since Dung's approach was introduced in [21], it was viewed as a special form of logic programming with *negation as failure*. For instance, in [21] it was proved that the grounded semantics can be characterized by the well-founded semantics [25] and the stable argumentation semantics can be characterized by the stable model semantics [26]. Also in [11], it was proved that the preferred semantics can be characterized by the p-stable semantics. In fact, the preferred semantics can be also characterized by the minimal models and the stable models of a logic program [35]. Furthermore, Caminada [48] proved the correspondence between complete extensions and 3-valued stable models of a logic program.

We can recognize two major branches for improving Dung's approach[3]. 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: *strongly connected components* (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 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 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 [21, 35, 36, 11]. In fact Dung in [21] introduced a metainterpreter P_{AF} for generating argumentation systems. When we have a logic program which represents an argumentation framework, we can split this program into subprograms where each subprogram could represent a part of an argumentation framework. For instance, let us consider a version of the mapping Ψ_{AF} introduced in [35, 11] in order to represent the argumentation framework of Figure 1 as the logic program P:

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

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 *winning/acceptable* arguments by *negation as failure*. This program can also be inferred from Dung's mapping P_{AF} [21] 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 argument a is defeated if argument a is not defeated. The second clause of P says that argument b is defeated if argument a is not defeated. The third clause of P says that argument a is acceptable if argument a is not defeated and the last clause of P says that argument b is acceptable if argument b is not defeated.

[3]By improving Dung's approach, we mean to define new extension-based argumentation which are not necessarily defined in terms admissible sets. However, it is expected that these new extension-based argumentation semantics are able to infer coherent points of views from a conflict of arguments.

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. This suggests a semantics dependency between P_1, P_2 and P_3. By managing this semantics dependency, one can define a whole semantics interpretation of P. For illustrating this idea, let us consider the logic programming semantics MM^{*^r}, one of the semantics introduced in this paper. In order to infer MM^{*^r}, 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 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 $d(a)$ true and $d(b)$ false. It is easy to see that the only minimal model of P_3' is: $\{acc(b)\}$. Therefore, $MM^{*^r}(P)$ 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 a unique model for P which is $\{acc(b), d(a)\}$. This model suggests that we can consider argument b as acceptable and the argument a as defeated. It is worth mentioning that the stable models of P correspond to the stable extensions of argumentation framework of Figure 1; however, the program P does not have stable models.

The idea of splitting a logic program into its component, in order to define logic programming semantics, has been explored by some authors in logic programming [18, 31, 24]. For instance, by splitting a logic program, Dix and Müller in [18] 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.

The property of relevance is a highly desirable property in any logic programming semantics. In fact, Dix and Müller in [19] pointed out that the reason of the anomalous behavior of stable model semantics is the failure of relevance. Also one of its main implications of relevance is that it allows us to define top-down algorithms for answering queries from a knowledge base. In the context of argumentation semantics, relevance can also play an important role for identifying the attack-dependencies that exist in argumentation framework [4].

In the first part of the paper, given that the existence of models for all normal logic program could play an important role in some applications such as argumentation theory, we define a general approach for extending any logic programming semantics S, such as the stable model semantics, in order to define

a new logic programming semantics S^* which will always have intended models for every normal logic program. This approach extends S for normal logic program in a natural way. S^* is equivalent to S for a logic program P whenever P has models in S. By considering the idea of splitting, we will formalize a recursive general schema for constructing new logic programming semantics. This schema takes as input any basic logic programming semantics S, such as the stable model semantics, and gives as output a new logic programming semantics S^r which satisfies some desired properties such as relevance and the existence of the intended models for every normal logic program. For instance, we will be able to define an alternative version of the stable model semantics which will always have intended models for every normal logic program, and satisfy the property of relevance.

In the second part of this paper, 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 of *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 show that there is a close relationship between the approach presented in [4] and the approach presented in this paper. In the last section, we present our main conclusions.

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 p-stable model semantics. After that, we present a short description of Dung's argumentation approach.

2.1. Syntax and some operations

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 $\neg\{a_1, ..., a_n\}$ to denote the set of atoms $\{\neg a_1, ..., \neg a_n\}$. A *normal* clause, C, is a clause of the form

$$a \leftarrow b_1 \wedge \ldots \wedge b_n \wedge \neg b_{n+1} \wedge \ldots \wedge \neg b_{n+m}$$

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. When $\mathcal{B}^- = \emptyset$, the clause can be regarded as a definite clause. We define a *definite program*, as a finite set of definite clauses. If the body of a clause is empty, then the clause is known as a *fact* and can be denoted just by: $a \leftarrow$.

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 $\{a \mid a \leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^- \in P\}$. Given a signature $\mathcal{L}$, we write $Prog_{\mathcal{L}}$ to denote the set of all normal programs defined over $\mathcal{L}$.

Remark 2.1. We want to point out that our negation symbol, $\neg$, corresponds to "*not*" in the standard use of *Logic Programming*.

The following definitions of dependency, proposed by Dix and Müller in [18], are inspired on the idea of splitting a logic program into its component in order to define a logic programming semantics. 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 set of dependencies of an atom x, denoted by *dependencies-of*(x), corresponds to the set $\{a \mid x \ depends \ on \ a\}$. We define an equivalence relation $\equiv$ between atoms of $\mathcal{L}_P$ as follows:

$$a \equiv b \text{ if and only if } a = b \text{ or } (a \text{ depends on } b \text{ and } b \text{ depends on } a)$$

We write $[a]$ to denote the equivalent class induced by the atom a.

Example 2.1. Let us consider the following normal program,

$$S = \{e \leftarrow e, \ \ c \leftarrow c, \ \ a \leftarrow \neg b \wedge c, \ \ b \leftarrow \neg a \wedge \neg e, \ \ d \leftarrow b\}.$$

The dependency relations between the atoms of $\mathcal{L}_S$ are as follows: *dependencies-of*$(a) = \{a, b, c, e\}$; *dependencies-of*$(b) = \{a, b, c, e\}$; *dependencies-of*$(c) = \{c\}$; *dependencies-of*$(d) = \{a, b, c, e\}$; and *dependencies-of*$(e) = \{e\}$. We can also see that, $[a] = [b] = \{a, b\}$, $[d] = \{d\}$, $[c] = \{c\}$, and $[e] = \{e\}$.

We take $<_P$ to denote the strict partial order induced by $\equiv$ on its equivalent classes. Hence, $[x] <_P [y]$, if and only if, there exists $w \in [x]$ and $z \in [y]$ such that z *depends-on* w and $[x]$ is not equal to $[y]$. By considering the relation $<_P$, each atom of $\mathcal{L}_P$ is assigned an order as follows:

- An atom x *is of order* 0, if $[x]$ is minimal in $<_P$.
- An atom x *is of order* $n + 1$, if n is the maximal order of the atoms on which x depends.

We say that a program P is of order n, if n is the maximum order of its atoms. We can also break a program P of order n into the disjointed union of programs P_i with $0 \leq i \leq n$, such that P_i is the set of rules for which the head of each clause is of order i (*w.r.t.* P). The empty program has order 0. We say that $P_0, \ldots, P_n$ are the *relevant modules* of P or the *components* of P.

Let us consider the following example in order to illustrate some of the concepts.

Example 2.2. By considering the equivalent classes of the program S in Example 2.1, the following relations hold: $\{c\} <_S \{a, b\}$, $\{e\} <_S \{a, b\}$, and $\{a, b\} <_S \{d\}$. We can also see that: d is of order 2, a is of order 1, b is of order 1, e is of order 0, and c is of order 0. This means that S is a program of order 2. The following table illustrates how the program S can be broken into the disjointed union of the following relevant modules or components S_0, S_1, S_2:

S	S_0	S_1	S_2
$e \leftarrow e.$	$e \leftarrow e.$		
$c \leftarrow c.$	$c \leftarrow c.$		
$a \leftarrow \neg b \wedge c.$		$a \leftarrow \neg b \wedge c.$	
$b \leftarrow \neg a \wedge \neg e.$		$b \leftarrow \neg a \wedge \neg e.$	
$d \leftarrow b.$			$d \leftarrow b.$

Now we introduce a single reduction for any normal program. Informally speaking, the idea of this reduction is to remove from a normal program any atom which has already been assigned some true value. 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 P be a normal logic program and $A = \langle T; F\rangle$ be a pair of sets of atoms. The reduction $R(P, A)$ is obtained by 2 steps:

1. Let $R'(P, A)$ be the program obtained in the following steps:

   (a) We replace every atom x that occurs in the bodies of P by 1 if $x \in T$, and we replace every atom x that occurs in the bodies of P by 0 if $x \in F$;

   (b) we replace every occurrence of $\neg 1$ by 0 and $\neg\ 0$ by 1;

   (c) every clause with a 0 in its body is removed;

   (d) finally we remove every occurrence of 1 in the body of the clauses.

2. $R(P, A) = norm_{\mathcal{CS}}(R'(P, A))$ such that $\mathcal{CS}$ is a rewriting system formed by the transformation rules: RED^+, RED^-, $Success$, $Failure$ and $Loop$ (Due to lack of space, we do not present the definition of these transformation rules; however, their definition can be founded in [20]); and $norm_{\mathcal{CS}}(P)$ denotes the uniquely determined normal form of a program P with respect to the system $\mathcal{CS}$.

We want to point out that this reduction does not coincide with the Gelfond-Lifschitz reduction [26] (it will be presented in Section 2.2.2). On the other hand, it is important to recall that the rewriting system $\mathcal{CS}$ can characterize the Well-founded Semantics [20].

Example 2.3. Let us consider the normal program S of Example 2.1 and the relevant module S_0 of S described in Example 2.2. Let P be the normal program $(S \setminus S_0) \cup \{b \leftarrow e, m \leftarrow n, n \leftarrow m\} = \{a \leftarrow \neg b \wedge c,\ b \leftarrow \neg a \wedge \neg e,\ d \leftarrow b,\ b \leftarrow e,\ m \leftarrow n,\ n \leftarrow m\}$, and let A be the pair of sets of atoms $\langle\{c\};\{e\}\rangle$. Thus, $R'(P, A) = \{a \leftarrow \neg b,\ b \leftarrow \neg a,\ d \leftarrow b,\ m \leftarrow n,\ n \leftarrow m\}$. Hence, $R(P, A) = \{a \leftarrow \neg b,\ b \leftarrow \neg a,\ d \leftarrow b\}$.

2.2. Logic Programming Semantics

Here, we present the definitions of a couple of logic programming semantics and some of their properties. Note that we only consider 2-valued logic programming semantics.

Definition 2.1. A logic programming semantics SEM is a mapping from the class of all programs into the powerset of the set of (2-valued) models.

We sometimes refer to *logic programming semantics* as *semantics*, when no ambiguity arises. The semantics that we consider in this paper are: *the minimal model semantics* (denoted by MM), *the stable model semantics* [26] (denoted by $stable$), and *the p-stable model semantics* [39] (denoted by $p\text{-}stable$). We will review these semantics in the next subsections. From now on, we assume that the reader is familiar with the notion of an *interpretation* [13].

2.2.1. Minimal model semantics

An interpretation M is called a (2-valued) *model* of P if and only if for each clause $c \in P$, $M(c) = 1$. It is clear that M is a characteristic function, and by a slight abuse of notation we can also regard M as a set of atoms, namely those to which M assigns the value 1 [32]. We say that M is a *minimal model* of P if and only if there does not exist a model M' of P such that $M' \subset M$, $M' \neq M$ [13]. We will denote by $MM(P)$ the set of all the minimal models of a given logic program P. Usually MM is called *minimal model semantics*.

Example 2.4. Let P be the normal program $\{a \leftarrow \neg b,\ \ b \leftarrow \neg a,\ \ a \leftarrow \neg c,\ \ c \leftarrow \neg a\}$. As we can see, P has five models: $\{a\}$, $\{b, c\}$, $\{a, c\}$, $\{a, b\}$, $\{a, b, c\}$; however, P has just two minimal models: $\{b, c\}$, $\{a\}$. Hence $MM(P) = \{\ \{b, c\}, \{a\}\ \}$.

2.2.2. Stable model semantics

The stable model semantics was defined in terms of the so called *Gelfond-Lifschitz reduction* [26] 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 [26].

Definition 2.2. 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 each rule that has a formula $\neg l$ in its body with $l \in S$, and then 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.

Example 2.5. Let $S = \{b\}$ and P be the following program: $\{b \leftarrow \neg a,\ \ c \leftarrow \neg b,\ \ b \leftarrow,\ \ c \leftarrow a\}$. Notice that P^S has three models: $\{b\}$, $\{b, c\}$ and $\{a, b, c\}$. Since the minimal model among these models is $\{b\}$, we can say that S is a stable model of P.

2.2.3. p-stable semantics

Before defining the p-stable semantics (introduced in [39]), 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 program as a theory, each negative literal $\neg a$ is regarded as the standard negation operator in classical 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 p-stable semantics is defined in terms of a single reduction which is defined as follows:

Definition 2.3. [39] Let P be a normal program and M a set of literals. We define

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

Example 2.6. Let us consider the set of atoms $M_1 = \{a, b\}$ and the following normal program P_1: $\{a \leftarrow \neg b \wedge \neg c,\ \ a \leftarrow b,\ \ b \leftarrow a\}$. We can see that $RED(P, M)$ is: $\{a \leftarrow \neg b,\ \ a \leftarrow b,\ \ b \leftarrow a\}$.

Next we present the definition of the p-stable semantics for normal programs.

Definition 2.4. [39] Let P be a normal program and M a set of atoms. We say that M is a *p-stable model* of P if $RED(P, M) \Vdash M$. We use p-stable to denote the semantics operator of p-stable models.

Example 2.7. Let us consider again P_1 and M_1 of Example 2.6. Let us verify whether M_1 is a p-stable model of P_1. First, we can see that M_1 is a model of P_1, for each clause C of P_1, M_1 evaluates C to true. We also can verify that $RED(P_1, M_1) \vdash M_1$. Then we can conclude that $RED(P_1, M_1) \Vdash M_1$. Hence, M_1 is a *p-stable model* of P_1.

2.2.4. Some properties of logic programming semantics

Informally, if a logic program has models under a particular logic programming semantics SEM, then we say that the program is $SEM\_consistent$. In the case that each normal logic program always has models under a given semantics, then this semantics is called *Always Consistent* (AC[4]).

Definition 2.5. Let P be a program and SEM be a semantics. We say that P is $SEM\_consistent$ if $SEM(P) \neq \emptyset$, otherwise we say that P is $SEM\_inconsistent$. A semantics SEM is *AC* if for every normal program P, $SEM(P) \neq \emptyset$.

We illustrate the above definition using the stable semantics.

Example 2.8. Let P_1 be the following program: $\{a \leftarrow \neg a\}$. Since P_1 does not have stable-models $stable(P_1) = \emptyset$. Hence, $stable(P_1)$ is $stable\_inconsistent$. Now, let P_2 be the following program: $\{a \leftarrow \neg b,\quad b \leftarrow \neg a\}$. Since P_2 has two stable-models, namely $\{a\}$ and $\{b\}$, then $stable(P_2) = \{\{a\}, \{b\}\}$. Hence, $stable(P_2)$ is $stable\_consistent$.

Finally, we present the property of relevance. Informally speaking, when it is necessary to find an answer to a query, the property of relevance allows us to compute just the partial models that sustain the answer to the query instead of computing the whole model. Pereira et al. [30] and Dix et al. [17, 18] are some of the authors who have discussed the importance of the property of relevance in logic programming semantics. Now we present the formal definition of the property of relevance.

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

Definition 2.6. Let P be a program and $P_0, \ldots, P_n$ its relevant modules. We say that a semantics SEM satisfies the *property of relevance* if for every i, $0 \leq i \leq n$, $SEM(P_0 \cup \cdots \cup P_i) = SEM(P)$ restricted to $\mathcal{L}_{P_0 \cup \cdots \cup P_i}$.

[4]Perhaps *Always Consistent* is not the best name but it is a shorthand to express this property. Thus some readers could not agree with this name.

Example 2.9. Let $P_0 = \{a \leftarrow \neg b,\ \ b \leftarrow \neg a\}$ and $P_1 = \{p \leftarrow \neg p,\ \ p \leftarrow \neg a\}$ be the relevant modules of the program $P = \{a \leftarrow \neg b,\ \ b \leftarrow \neg a,\ \ p \leftarrow \neg p,\ \ p \leftarrow \neg a\}$. We can verify that the stable semantics is not relevant since $stable(P_0) \neq stable(P)$ restricted to $\mathcal{L}_{P_0}$.

2.3. Argumentation theory: Dung's approach

A fundamental definition of Dung's approach is the concept called argumentation framework, which is defined as follows.

Definition 2.7. [21] 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.

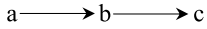

Figure 3. Graph representation of the argumentation framework $AF = \langle\{a, b, c\}, \{(a, b), (b, c)\}\rangle$.

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 3. We say that *a attacks b* (or b is attacked by a) if $attacks(a, b)$ hold. 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 3, $\{a\}$ attacks b.

Definition 2.8. [21] 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 3. We will denote by $max\_conflict\_freeSets(AF)$ the set of maximal conflict free sets (*w.r.t.* set inclusion) of an argumentation framework AF.

Definition 2.9. [21] 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. 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 Dung's approach which has played an influential role on argumentation research is the preferred semantics [21]. This semantics is defined as follows.

Definition 2.10. [21] A *preferred extension* of an argumentation framework AF is a maximal (*w.r.t.* inclusion) admissible set of AF.

The admissible sets of Figure 3 are $\{a\}$ and $\{a, c\}$, then the only preferred extension is $\{a, c\}$.

Remark 2.2. There are other interesting argumentation semantics introduced by Dung, such as the *stable semantics* and the *grounded semantics* [21], we omit their definition due to lack of space.

3. Construction of new logic programming semantics

In this section we construct elaborated semantics, each of which is based on a *basic logic programming semantics*. Our new semantics will satisfy the following suitable properties: *AC* and *relevance*. We start presenting a useful definition that indicates when a semantics S is a *basic logic programming semantics*.

Definition 3.1. Let S be a semantics and P be a normal program. We say that S is a *basic semantics* if it satisfies the following property: if $x \in \mathcal{L}_P$ and $x \leftarrow \in P$, then P is $S\_consistent$.

Let us remark that every well known semantics such as the logic programming semantics mentioned in Section 2.2, namely MM, $stable$, and $p\text{-}stable$, are basic semantics. In fact in all these basic semantics there is exactly one intended model that corresponds to $\mathcal{L}_P$. For instance, let P be the following program: $\{a \leftarrow,\ \ b \leftarrow,\ \ a \leftarrow b,\ \ b \leftarrow \neg a\}$. Observe that all the atoms of the program P appear as facts in P; hence P will always have the model $\{a, b\}$.

Clearly, we can see that a semantics AC is also a basic semantics.

3.1. Semantics always defined

It is sometimes desirable that a logic programming semantics of a normal program could be AC [7, 46, 20, 18, 39, 43]; for instance, the case when a logic program is used for modeling an argumentation semantics [21, 11, 35]. We want to point out that according to Bench-Capon and Dunne there are three main problems which have been reported in the context of argumentation semantics [5]:

(P1) *Emptiness*: this problem happens when even though an extension satisfying the prescribed conditions always exists, there are argumentation frameworks for which the only such extension is the empty set. This problem can arise with both the grounded and preferred semantics.

(P2) *Non-existence*: there are argumentation semantics as the stable argumentation semantics that when it exists is never empty, but there are argumentation frameworks for which no extension meeting the required criteria exist.

(P3) *Multiplicity*: in an argumentation framework AF there may be several incompatible extensions, *i.e.*S_1 and S_2 which are well-defined extensions of AF but with $S_1 \cup S_2$ failing to be so. This problem does not happen with Dung's grounded semantics; however, argumentation frameworks are easily constructed in which both the preferred and stable semantics exhibit this phenomenon [5].

In Section 1, we have already motivated that an argumentation semantics can be regarded as a logic programming semantics. Hence, we claim that the non-existence of extensions in an argumentation semantics can be tackle by considering logic programming semantics which are AC and specify a congruent patter of selection of arguments in a dispute among arguments. Therefore, we show how to construct a logic programming semantics that is AC, based on a particular basic semantics S.

We start defining some concepts *w.r.t.* the notion of generalized S model. The concept of generalized S model is closely related to the semantics of *abductive logic programming* [29, 28], in particular to the concept of *generalized answer set*. It has also been explored for different logic programming approaches

[5] We are not concerned with this issue in this paper.

as in [1, 40]. For instance, the authors in [40] 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 3.4) and another one *w.r.t.* the cardinality of the subindexes of the generalized models[6].

Given a basic semantics S, we present the definition of a generalized S model of a program.

Definition 3.2. Let S be a basic semantics, P be a normal program, $A \subseteq \mathcal{L}_P$ a set that we call of abductives, and let $B \subseteq A$. We say that M_B is a *generalized S model of P w.r.t. A*, if $M_B \in S(P \cup B)$.

Now, we define a partial order between a pair of generalized S models of a program.

Definition 3.3. Let S be a basic semantics, let P be a normal program, and $A \subseteq \mathcal{L}_P$. Let $B_1 \subseteq A$, $B_2 \subseteq A$, and let M_{B_1} and M_{B_2} be two generalized S models of P *w.r.t.* A. We define a *partial order between two generalized S models of P w.r.t. A* as follows: $M_{B_1} < M_{B_2}$, if $B_1 \subset B_2$.

Based on the partial order just defined, we define the minimal generalized S models of a program.

Definition 3.4. Let S be a basic semantics, and P be a normal program. Let $A \subseteq \mathcal{L}_P$, and $B \subseteq A$. We say that M_B 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 among the generalized S models of P *w.r.t.* A. We write S^* to denote the *minimal generalized S* semantics *w.r.t.* A, 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 the three previous definitions of this section, we do not instantiate any of them to a particular basic logic programming semantics. The following two examples illustrate Definition 3.2, Definition 3.3, and Definition 3.4, each of them based on a different basic semantics.

Example 3.1. Let C be the normal program $\{p \leftarrow \neg p\}$. We can see that C is $stable\_inconsistent$, 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 $p\text{-}stable(C) = p\text{-}stable^*(C) = \{\{p\}\}$.

Example 3.2. Let D be the normal program $\{a \leftarrow \neg b, \quad b \leftarrow \neg c, \quad c \leftarrow \neg a\}$. Observe that D is $p\text{-}stable\_inconsistent$; however, one can see that $\{a, b\}$ is a p-stable model of $P \cup \{a\}$, $\{b, c\}$ is a p-stable model of $P \cup \{b\}$ and $\{c, a\}$ is a p-stable model of $P \cup \{c\}$. Since the models $\{a, b\}_{\{a\}}$, $\{b, c\}_{\{b\}}$, $\{c, a\}_{\{c\}}$ are the three minimal generalized p-stable models of D, $p{-}stable^*(D) = \{\{a, b\}, \{b, c\}, \{a, c\}\}$. Observe that D is $stable\_inconsistent$ and $stable^*(D) = \{\{a, b\}, \{b, c\}, \{a, c\}\}$.

The following lemma assures that given a semantics S induced by Definition 3.4, each normal program is $S\_consistent$.

Lemma 3.1. For every basic semantics S and normal program P, $S^*(P)$ is AC.

[6] In [40], a generalized model is called a $\mathcal{L}$-completion.

Proof: Follows directly from Definition 3.1 and definition of a semantics AC given in Definition 2.5. ■

One important property of the semantics induced by Definition 3.4 is that the concept of a generalized S model is important only in the case that the basic semantics S is $S\_inconsistent$ (see Definition 2.5). The following lemma assures that given a normal logic program P and a basic semantics S, if P is $S\_consistent$, then $S(P)$ and $S^*(P)$ are the same, *e.g.*, considering Example 3.1, we can see that $p\text{-}stable(C) = p\text{-}stable^*(C) = \{\{p\}\}$.

Lemma 3.2. For every basic semantics S and logic program P. If P is $S\_consistent$ then $S^*(P) = S(P)$.

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

The following lemma makes some observations *w.r.t.* MM, $stable$, $p - stable$ and their induced semantics based on the concept of generalized model.

Lemma 3.3. MM and MM^* are the same semantics. $stable$ is different from $stable^*$. $p - stable$ is different from $p\text{-}stable^*$. $stable^*$, $p\text{-}stable^*$, and MM^* are three different semantics.

Proof: We can see that MM is the same as MM^* because MM is AC (Lemma 3.2). In Example 3.1, we can note that program C shows that $stable(C)$ is different from $stable^*(C)$. Also in Example 3.2, program D shows that $p\text{-}stable$ is different from $p\text{-}stable^*$. Now, let E be the normal program $\{a \leftarrow \neg b, \quad b \leftarrow \neg a, \quad p \leftarrow \neg b, \quad p \leftarrow \neg p\}$. We can verify that: $stable^*(E) = stable(E) = \{\{p, a\}\}$. $p\text{-}stable^*(E) = p\text{-}stable(E) = \{\{p, a\}, \{p, b\}\}$. $MM^*(E) = MM(E) = \{\{p, a\}, \{p, b\}\}$. This program shows that $p\text{-}stable^*$ is different from $stable^*$ and that MM^* is different from $stable^*$. Now, let F be the normal program $\{a \leftarrow \neg b, \ \ b \leftarrow \neg a, \ \ u \leftarrow a, \ \ x \leftarrow \neg y \wedge u, \ \ y \leftarrow \neg z \wedge u, \ \ z \leftarrow \neg x \wedge u\}$. One can see that $p\text{-}stable^*(F) = \{\{b\}\}$ and $MM^*(F) = \{\{b\}, \{a, u, x, y\}, \{a, u, x, z\}, \{a, u, y, z\}\}$. Hence, this program shows that $p\text{-}stable^*$ is different from MM^*. ■

3.2. Constructing relevant semantics

The property of relevance is a highly desirable property in any logic programming semantics. Pereira, *et al*, [30] and Dix, *et al*, [17, 18] are some of the authors who have discussed the importance of the property of relevance in logic programming semantics. One of its main implications of relevance is that it allows us to define top-down algorithms for answering queries from a knowledge base. This thanks that one can split a logic program into subprograms. Hence, computing models of a given semantics from a logic program can be simplified by considering only subprograms. In the context of argumentation semantics, relevance can also play an important role for identifying the attack-dependencies that exist in an argumentation framework [4]. In fact, following the idea of relevance Baroni *et al*, defined a partial order between the strongly connected components of the directed graph induced by an argumentation framework[7].

For constructing relevant semantics, we considering a process of splitting a program into its component. Hence, by considering an order between these components, we define a general recursive schema which takes as input a logic programming semantics S and gives as output a logic programming semantics S_c^r which satisfy relevance.

[7]We want to clarify that Baroni *et al*, do not use the concept of relevant by itself, they consider strongly connected components and the so-called directionality principle for following similar concept.

Before to introduce the recursive construction, we introduce a basic operator between sets: Given two sets of interpretations Q and R, we define $Q \uplus R := \{M_1 \cup M_2 | M_1 \in Q, M_2 \in R\}$.

Now we present the recursive definition of the relevant semantics S_c^r based on a given semantics S that is AC. Let us recall that an AC semantics is also a basic semantics. In order to understand this definition, it is useful to recall the definition *w.r.t.* the order of a normal program given in Section 2.1.

Definition 3.5. Let S be a semantics that is AC. We define the associated S_c^r semantics recursively as follows: Given a program P of order 0, $S_c^r(P) = S(P)$. For a program P of order $n > 0$ we define

$$S_c^r(P) = \bigcup_{M \in S(P_0)} \{M\} \uplus S_c^r(R(P \setminus P_0, \langle M; N \rangle))$$

where $N := (\mathcal{L}_{P_0} \cup \{a \in \mathcal{L}_P \mid a \notin Head(P)\}) \setminus M$.

It is important to observe that in each recursive call of the construction of the semantics S_c^r, the reduction R is applied. Since R reduces a given program in terms of the rewriting systems which characterizes the WFS semantics, the WFS semantics has an influence to semantics constructed by S_c^r. However, since R is not applied to the original program P, we cannot insure that S_c^r will contain the WFS model. If R is applied to the original program before using the construction S_c^r, S_c^r will define a semantics which contains the WFS model. For the particular results that will be presented in Section 4, this property is not relevant; however, it is a consideration that could be relevant for other applications.

Let us illustrate how the recursion in Definition 3.5 is given by means of the following example.

Example 3.3. Let us consider the program $E = \{a \leftarrow \neg b, \quad b \leftarrow \neg a, \quad p \leftarrow \neg b, \quad p \leftarrow \neg p\}$ defined in the proof of Lemma 3.3. Let S be the semantics $stable^*$ that is AC. We are going to compute the $stable_c^{*^r}(E)$ for the normal program E. So, according to Definition 3.5, since E is of order 1, then we need to obtain the following:
1) $stable^*(E_0)$,
2) $stable_c^{*^r}(R(E \setminus E_0, \langle M; N \rangle))$ for each $M \in stable^*(E_0)$, and
3) $stable_c^{*^r}(E) = \bigcup_{M \in stable^*(E_0)} \{M\} \uplus stable_c^{*^r}(R(E \setminus E_0, \langle M; N \rangle))$.

Obtaining $stable^*(E_0)$: Let us see that $E_0 = \{a \leftarrow \neg b, \quad b \leftarrow \neg a\}$ is the component of order 0 of program E. Thus $stable^*(E_0) = stable(E_0) = \{\{a\}, \{b\}\}$.

Obtaining $stable_c^{*^r}(R(E \setminus E_0, \langle M; N \rangle))$ for each $M \in stable^*(E_0)$: Since $stable^*(E_0)$ has two models, then we need to consider two cases.

1. First, consider M to be $\{a\}$. Let E' be the normal program $R(E \setminus E_0, \langle M; N \rangle)$ with $E \setminus E_0 = \{p \leftarrow \neg b, \quad p \leftarrow \neg p\}$, and $N = (\mathcal{L}_{E_0} \cup \{\}) \setminus M = (\{a, b\} \cup \{\}) \setminus \{a\} = \{b\}$. We can see that $E' = \{p \leftarrow\}$. Now we need to obtain $stable_c^{*^r}(E')$. Since E' is of order 0, then applying Definition 3.5, we can see that $stable_c^{*^r}(E') = stable^*(E') = \{\{p\}\}$. So, $stable_c^{*^r}(R(E \setminus E_0, \langle M; N \rangle))$ with $M = \{a\}$ corresponds to $\{M\} \uplus stable_c^{*^r}(E') = \{\{a\}\} \uplus \{\{p\}\} = \{\{a, p\}\}$.

2. Now, consider M to be $\{b\}$. In this case, we can verify that $R(E \setminus E_0, \langle M; N \rangle) = \{p \leftarrow \neg p\}$. Since $R(E \setminus E_0, \langle M; N \rangle)$ is of order 0, then applying Definition 3.5, we can see that $stable_c^{*^r}(R(E \setminus E_0, \langle M; N \rangle)) = stable^*(R(E \setminus E_0, \langle M; N \rangle)) = \{\{p\}\}$. So, $stable_c^{*^r}(R(E \setminus$

$E_0, \langle M; N\rangle))$ with $M = \{b\}$ corresponds to $\{M\} \uplus stable_c^{*^r}(R(E \setminus E_0, \langle M; N\rangle)) = \{\{b\}\} \uplus \{\{p\}\} = \{\{b, p\}\}$.

Obtaining $stable_c^{*^r}(E)$: It is easy to verify that $stable_c^{*^r}(E) = \bigcup_{M \in stable^*(E_0)}\{M\} \uplus stable_c^{*^r}(R(E \setminus E_0, \langle M; N\rangle)) = \{\{a, p\}, \{b, p\}\}$.

Let us observe that in Definition 3.5, the set N that is considered is defined in terms of the set $\{a \in \mathcal{L}_P \mid a \notin Head(P)\}$. The set N allows us to deal with normal programs where some of the atoms in the signature of the program do not occur in the head of some clause of that program. For instance, let us consider the program $J = \{a \leftarrow \neg b\}$ and the $stable^*$ semantics. Now, let us suppose we want to infer $stable_c^{*^r}(J)$. The first step, in order to infer $stable_c^{*^r}(J)$, is to obtain the components of J (see Section 2.1). Since a depends on b, but b does not depend on a, $[b] <_P [a]$. This means that J has 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 a. It is obvious that J_0 is an empty component, but this is not a problem when we apply Definition 3.5 in order to infer $stable_c^{*^r}(J)$. So, let us see how to obtain $stable_c^{*^r}(J)$. We can verify that J is of order 1, and $stable^*(J_0) = \{\ \}$. We also see that $J \setminus J_0$ is J and M is $\{\ \}$. Additionally, since we expect intuitively that b evaluates to false, then b must be in the set N that corresponds to the set of atoms considered as false. So, $N = (\mathcal{L}_{J_0} \cup \{x \in \mathcal{L}_J \mid x \notin Head(J)\}) \setminus M = (\{\ \} \cup \{b\}) \setminus \{\ \} = \{b\}$. Now, let us see that $R(J \setminus J_0, \langle M; N\rangle) = R(\{a \leftarrow \neg b\}, \langle\{\ \}; \{b\}\rangle)$ is $\{a\}$. Since $R(J \setminus J_0, \langle M; N\rangle)$ is of order 0, then applying recursively Definition 3.5, we can see that $stable_c^{*^r}(R(J \setminus J_0, \langle M; N\rangle)) = stable^*(R(J \setminus J_0, \langle M; N\rangle)) = \{\{a\}\}$. So, $stable_c^{*^r}(J) = \{M\} \uplus stable_c^{*^r}(R(J \setminus J_0, \langle M; N\rangle)) = \{\{\ \}\} \uplus \{\{a\}\} = \{\{a\}\}$.

For the reader who knows $STABLE^{rel}$'s definition presented by Dix et al. [18], we want to point out that the construction S_c^r is similar to $STABLE^{rel}$ *w.r.t.* the relevance property; however, $STABLE^{rel}$ has a skeptical construction and S_c^r has a construction by scenarios.

3.2.1. Analizing programs with tautologies

It is important to eliminate tautologies from the programs, since they can introduce non-desirable models. For example, if P is the program $\{a \leftarrow \neg b, \quad b \leftarrow a, b\}$, then the minimal models for this program are $\{a\}$ and $\{b\}$; however, after deleting the second rule, which is a tautology, it is clear that the second set, namely $\{b\}$ is not an intended minimal model.

In order to construct a relevant semantics based on a semantics S that also considers programs with tautologies, we need to extend the semantics S_c^r given in Definition 3.5. This extension considers removing all the tautologies of the original program and then applying to the new program the semantics S_c^r. We are going to consider two types of tautologies, and depending on the type, we define a set of tautologies from a normal program. Given a normal program P, we say that $Taut_1(P) = \{x \leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^- \mid x \in \mathcal{B}^+\}$ and $Taut_2(P) = \{x \leftarrow \mathcal{B}^+ \cup \neg\mathcal{B}^- \mid \mathcal{B}^+ \cap \mathcal{B}^- \neq \emptyset\}$.

Now, we define two different relevant semantics, $S^r(P)$ and S^r_{alt}, that consider programs with tautologies.

Definition 3.6. Let P be a normal program. We define $S^r(P) = S_c^r(P \setminus (Taut_1(P) \cup Taut_2(P)))$ and $S^r_{alt}(P) = S_c^r(P \setminus Taut_1(P))$.

This completes our construction of relevant semantics. Now we present an example of semantic $S^r(P)$ and some lemmas about some of the semantics defined until now.

Example 3.4. Consider the program $K = \{b \leftarrow b,\ a \leftarrow \neg b\}$. The semantics $S^r(K)$ of program K is the same when S corresponds to the MM^*, the $stable^*$, or the $p-stable^*$ semantics, namely, $MM^{*^r}(K) = stable^{*^r}(K) = p\text{-}stable^{*^r}(K) = \{\{a\}\}$.

Observe that the semantics induced by Definition 3.6 are different *w.r.t.* the semantics induced by Definition 3.4. The following lemmas formalize the differences *w.r.t.* some semantics.

Lemma 3.4. $stable^*$ is different from $stable^{*^r}$, and $p\text{-}stable^*$ is different from $p\text{-}stable^{*^r}$.

Proof: Program E in Example 3.3 shows that $stable^*$ is different from $stable^{*^r}$. Program F defined in the proof of Lemma 3.3 shows that $p\text{-}stable^*$ is different from $p\text{-}stable^{*^r}$. ■

Lemma 3.5. $stable^{*^r}$, $p\text{-}stable^{*^r}$, and MM^{*^r} are three different semantics.

Proof: Let L be the following normal program: $\{a \leftarrow \neg b,\ b \leftarrow \neg a,\ p \leftarrow \neg b,\ p \leftarrow \neg p,\ b \leftarrow \neg p\}$. We can verify that $stable^{*^r}(L) = \{\{p, a\}\}$ and $p\text{-}stable^{*^r}(L) = \{\{p, a\}, \{p, b\}\}$. Hence $stable^{*^r}$ is different from $p\text{-}stable^{*^r}$[8]. The following example shows that $stable^{*^r}$ is different from MM^{*^r}. Let R be the following normal program: $\{x \leftarrow \neg y,\ y \leftarrow \neg z,\ z \leftarrow \neg x,\ x \leftarrow \neg u,\ d \leftarrow \neg z,\ u \leftarrow \neg d\}$. We can verify that $stable^{*^r}(R) = p\text{-}stable^{*^r}(R) = \{\{x, y, d\}\}$. However, $\{u, x, z\} \in MM^{*^r}(R)$. This example also shows that $p\text{-}stable^{*^r}$ is different from MM^{*^r}. ■

As a final result of this subsection, we can introduce the following lemma.

Lemma 3.6. For every semantics S and program P, S^{*^r} is AC.

Proof: Follows directly from Lemma 3.1 ■

This lemma assures that any logic programming semantics induced by Definition 3.4 and Definition 3.6 is defined for any program. As final result of this section, we want to formalize that the semantics $stable^{*^r}$, $p\text{-}stable^{*^r}$, and MM^{*^r} satisfy relevance.

Theorem 3.1. $stable^{*^r}$, $p\text{-}stable^{*^r}$, and MM^{*^r} satisfy relevance.

Proof: The proof is straightforward by definition of S^R. ■

4. Construction of extension based argumentation semantics

So far we have defined how to construct logic programming semantics which are AC and satisfy the property of relevance. Now in this section, we are going to show that these new argumentation semantics can induce new extension-based argumentation semantics. For this end, we map an argumentation framework into a logic program. This approach of mapping an argumentation approach into a logic program is a standard approach which was first introduced in [21] and has been explored in several contexts in [11, 35, 23, 47]. The main difference among these approaches is the purpose of each mapping. Some of them are introduced only for operational purpose [23, 47]. This means the mapping does not only require to specify an argumentation framework into a logic program, but it also introduces operational knowledge. The mappings introduced in [21, 11, 35] are thought in terms of the basic principles that a set

[8] This example also show that $stable^{*^r}$ is different from MM^{*^r}.

of *winning/acceptable* arguments must satisfy in a dispute among arguments. Since the purpose of the mappings in this paper will be the definition of new argumentation semantics, we claim that a suitable mapping for defining new argumentation semantics has to be the specification of basic principles of the argumentation inferences, *e.g.*, the basic principle of *conflict-freeness* (an intensive discussion of basic principles in argumentation theory can be found in [3]).

Let $\mathcal{M}$ be a mapping from $\mathcal{AF}_{AR}$ to $Prog_{\mathcal{L}}$, *i.e.*, a mapping from the set of all argumentation frameworks over a finite set of arguments AR to the set of all normal programs defined over a signature $\mathcal{L}$. $\mathcal{M}$ assigns to every argumentation framework AF a normal program denoted by P_{AF}.

In order to show how to define argumentation semantics in terms of basic principles, we will define an instance of $\mathcal{M}$. In this mapping, we use the predicate *d(x)*, where the intended meaning of *d(x)* is: "the argument *x* is defeated". We also use the predicate *a(x)*, where the intended meaning of *a(x)* is: "the argument *x* is accepted".

Definition 4.1. Let $AF = \langle AR, attacks\rangle$ be an argumentation framework, $P^1_{AF} = \{d(x) \leftarrow \neg d(y_1), \ldots, d(x) \leftarrow \neg d(y_n) \mid x \in AR \ and \ \{y_1, \ldots, y_n\} = \{y_i \in AR \mid (y_i, x) \in attacks\}\}$; and $P^2_{AF} = \bigcup_{x\in AR}\{a(x) \leftarrow \neg d(x)\}$. We define: $P_{AF} = P^D_{AF} \cup P^A_{AF}$.

The intended meaning of each clause $d(x) \leftarrow \neg d(y)$ is that argument x will be defeated when anyone of its adversaries y is not defeated. The expert reader in argumentation theory can observe that essentially, P^D_{AF} captures the basic principle of *conflict-freeness* (this means that any set of acceptable argument will not contain two arguments which attack each other, see Definition 2.8). The idea P^A_{AF} is just to infer that any argument x that is not defeated is accepted. It is worth commenting that according to Baroni and Giacomin [3], the principle of conflict-freeness is the minimal requirement to be satisfied by any argumentation semantics.

Example 4.1. Now, we illustrate the mapping P_{AF}, let AF be the argumentation framework of Figure 2. We can see that $P_{AF} = P^D_{AF} \cup P^A_{AF}$ is:

P^D_{AF} :			P^A_{AF} :	
$d(a) \leftarrow \neg d(b).$	$d(d) \leftarrow \neg d(a).$	$d(e) \leftarrow \neg d(d).$	$a(a) \leftarrow \neg d(a).$	$a(d) \leftarrow \neg d(d).$
$d(b) \leftarrow \neg d(c).$	$d(d) \leftarrow \neg d(b).$		$a(b) \leftarrow \neg d(b).$	$a(e) \leftarrow \neg d(e).$
$d(c) \leftarrow \neg d(a).$	$d(d) \leftarrow \neg d(c).$		$a(c) \leftarrow \neg d(c).$	

We point out that P_{AF} is similar to the mapping defined in [21]. In fact, we known that

1. The stable models of P_{AF} characterize the stable argumentation semantics (see [21, 33] for details), and

2. by Theorem 17 of [21], one can see that the well founded model semantics [25] of P_{AF} characterizes the grounded semantics defined in [21] (see [33] for details).

Now, we define how any logic programming semantics, S, can induce an abstract argumentation semantics under the mapping P_{AF}.

Definition 4.2. Let $AF = \langle AR, attacks \rangle$ be an argumentation framework and S be any logic programming semantics. The semantics S induces the *extension-based argumentation semantics* $S_{\mathcal{M}}$ as follows:

$$S_{\mathcal{M}}(AF) = f(S(P_{AF}))$$

where f is a mapping from $2^{2^{\mathcal{L}_{P_{AF}}}}$ to $2^{2^{AR}}$, such that $f(X) = \bigcup_{M \in X}\{t(M)\}$ with $t(M) = \{y \mid a(y) \in M\}$.

Our definition is written in terms of a mapping f given explicitly in the second row, but it could be applied to a different mapping. Observe that for each model of $S(P_{AF})$, the mapping f assigns a set of arguments from the argumentation framework AF. Now, we illustrate the definition of $S_{\mathcal{M}}(AF)$.

Example 4.2. Let us again consider the argumentation framework AF of Figure 2. Let S be the logic programming semantics $stable^{*^r}$. Let us consider the normal program P_{AF} previously obtained in Example 4.1. This means that we are going to obtain the extension-based argumentation semantics $stable^{*^r}_{\mathcal{M}}(AF)$. First of all, we have to obtain the semantics $stable^{*^r}$ of P_{AF}. It is not difficult to see that $stable^{*^r}(P_{AF})$ corresponds to the following set:

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

Hence, $stable^{*^r}_{\mathcal{M}}(AF) = f(stable^{*^r}(P_{AF})) = \{\{c, e\}, \{a, e\}, \{b, e\}\}$.

Let us point out that the models suggested by the argumentation semantics $stable^{*^r}_{\mathcal{M}}$ are the same as the extensions suggested by the semantics $CF2$ introduced by P. Baroni et al. [4]. Moreover, we can also verify that $MM^{*^r}_{\mathcal{M}}$ and $p - stable^{*^r}_{\mathcal{M}}$ also coincide with CF2 in this example.

The following lemma indicates that $S_{\mathcal{M}}$ is AC.

Lemma 4.1. For every logic programming semantics S, the extension-based argumentation semantics $S_{\mathcal{M}}$ is AC.

Proof: Follows directly from Lemma 3.1 and Lemma 3.6. ■

5. Relation between semantics $CF2$ and $MM^{*^r}_{\mathcal{M}}$

In this section, we will show that there is a close relationship between the approach presented in [4] and the approach presented in this paper. In particular, we show the relation between semantics $CF2$ and $MM^{*^r}_{\mathcal{M}}$.

5.1. Semantics $CF2$

We present some definitions *w.r.t.* the argumentation semantics CF2. The details of these definitions are presented in [4].

Definition 5.1. [4] Given an argumentation framework $AF = \langle AR, attacks \rangle$, the binary relation of *path-equivalence* between nodes, denoted as $PE_{AF} \subseteq (AR \times AR)$, is defined as follows:
— $\forall a \in AR$, $(a, a) \in PE_{AF}$,
— given two distinct nodes a, $b \in AR$, $(a, b) \in PE_{AF}$ if and only if there is a path from a to b and a path from b to a.

Given an argumentation framework $AF = \langle AR, attacks \rangle$, the *strongly connected components* of AF are the equivalent classes of nodes which are defined according to path-equivalence relaction. The set of the strongly connected components of AF is denoted as $SCCS_{AF}$. Given a node $a \in AR$, the strongly connected component a belongs to is denoted as $SCC_{AF}(a)$.

Definition 5.2. [4] Let $AF = \langle AR, attacks \rangle$ be an argumentation framework, and let $S \subseteq AR$ be a set of arguments. The *restriction* of AF to S is the argumentation framework $AF \downarrow_S = \langle S, attacks \cap (S \times S) \rangle$.

Considering an argumentation framework, $AF = \langle AR, attacks \rangle$, a set $E \subseteq AR$ and a strongly connected component $S \in SCCS_{AF}$. We will present the definition of some useful sets, the formal definition of these sets is in [4]. The set $D_{AF}(S, E)$ consists of the nodes of S attacked by E from outside S, the set $U_{AF}(S, E)$ consists of the nodes of S that are not attacked by E from outside S and are defended by E (i.e., their defeaters from outside S are all attacked by E), and $P_{AF}(S, E)$ consists of the nodes of S that are not attacked by E from outside S and are not defended by E (i.e., at least one of their defeaters from outside S is not attacked by E). Finally, $UP_{AF}(S, E) = (S \setminus D_{AF}(S, E)) = (U_{AF}(S, E) \cup P_{AF}(S, E))$.

Here, we define $GF(AF, C)$ for an argumentation framework $AF = \langle AR, attacks \rangle$ and a set $C \subseteq A$, representing the defended nodes of AF: two cases have to be considered in this respect.

If AF consists of exactly one strongly connected component, it does not admit a decomposition in which can be applied the directionality principle, therefore it has to be assumed that $GF(AF, C)$ coincides in this case with a *base function*, denoted as $BF_S(AF, C)$, that must be assigned in order to characterize a particular argumentation semantics S.

On the other hand, if AF can be decomposed into several strongly connected components, then, $GF(AF, C)$ is obtained by applying recursively GF to each strongly connected component of AF, deprived of the nodes in $D_{AF}(S, E)$. Formally, this means that for any $S \in SCCS_{AF}$, $(E \cap S) \in GF(AF \downarrow_{UP_{AF}(S,E)}, C')$, where C' represents the set of defended nodes of the restricted argumentation framework $AF \downarrow_{UP_{AF}(S,E)}$. The set C' can be determined by taking into account both the attacks coming from outside AF and those coming from other strongly connected components of AF.

Definition 5.3. [4] A given argumentation semantics S is *SCC-recursive* if and only if for any argumentation framework $AF = \langle AR, attacks \rangle$, $E_S(AF) = GF(AF, AR)$, where for any $AF = \langle AR, attacks \rangle$ and for any set $C \subseteq AR$, the function $GF(AF, C) \subseteq 2^{AR}$ is defined as follows: for any $E \subseteq AR$, $E \in GF(AF, C)$ if and only if

- in case $|SCCS_{AF}| = 1$, $E \in BF_S(AF, C)$,
- otherwise, $\forall S \in SCCS_{AF}(E \cap S) \in GF(AF \downarrow_{UP_{AF(S,E)}}, U_{AF}(S, E) \cap C)$.

where $BF_S(AF, C)$ is a function, called base function, that, given an argumentation framework $AF = \langle AR, attacks \rangle$ such that $|SCCS_{AF}| = 1$ and a set $C \subseteq AR$, gives a subset of 2^{AR}.

Definition 5.3 does not define any particular semantics, but defines a general structure, where if one changes the base function, $BF_S(AF, C)$, one can induce several semantics.

In particular, when $BF_S(AF, C)$ is the function that returns all sets that are free of maximal conflicts, $max\_conflict\_freeSets(AF)$, then $CF2$ is obtained.

5.2. Expressing $CF2$ using $MM^{*^r}_{\mathcal{M}}$

This section introduces Theorem 5.1 which formalizes the fact about the coincidence between the stratified argumentation semantics and the argumentation semantics $CF2$ introduced by P. Baroni et al. [4]. In order to present the proof of Theorem 5.1, we first show some lemmas.

The following lemma shows that the number of strongly connected components of an argumentation framework AF is the same as the number of components of the normal program P_{AF}.

Lemma 5.1. Let AF be an argumentation framework. If $P_{AF} = P^D_{AF} \cup P^A_{AF}$ such that P^D_{AF} is of order n, then $|SCC_{AF}| = n + 1$.

Proof: (sketch) Since the number of components of P^1_{AF} depends on the number of equivalent classes of atoms in $\mathcal{L}_{P_{AF}}$ and the number of strongly connected components depends on the number of equivalent classes of nodes in PE_{AF}, the proof follows from that fact that: The number of equivalent classes of atoms induced by the relation *depends-on* in $\mathcal{L}_{P_{AF}}$ is the same as the number of classes of nodes induced by the relation *path-equivalence* in PE_{AF}. ■

Lemma 5.2. Let $AF = \langle AR, Attacks\rangle$ be an argumentation framework. If $P_{AF} = P^D_{AF} \cup P^A_{AF}$ such that P^D_{AF} is of order 0, then $E \in max\_conflict\_freeSets(AF)$ if and only if $\{a(x)|x \in E\} \cup \{d(x)|x \in AR \setminus E\}$ is a model of $MM^{*^r}_{\mathcal{M}}(P_{AF})$.

Proof: Observations:

1. P_{AF} consists of two levels where P^D_{AF} is of order 0 and P^A_{AF} is of order 1. Moreover, M is a model of $MM^{*^r}_{\mathcal{M}}(P_{AF})$ if and only if there exists M_1 and M_2 such that $M = M_1 \cup M_2$, M_1 is a minimal model of P^D_{AF} and $M_2 = \{a(x)|a(x) \leftarrow \neg d(x) \in P^2_{AF}, d(x) \notin M_1, x \in AR\}$.
2. If E is a conflict free set of AF, then $M = \{d(x)|x \in E\}$ is a model of P^D_{AF}.
3. If M is a model of P^D_{AF}, then $E = \{x|d(x) \in M\}$ is a conflict free set of AF.

$\Rightarrow$ If E is a maximal conflict free set of AF, then, by Proposition 1 of [35] and Observation 2, $M_1 = \{d(x)|x \in AR \setminus E\}$ is a minimal model of P^D_{AF}. Hence, by Observation 1, $M_1 \cup \{a(x)|x \in E\}$ is a model of $MM^{*^r}_{\mathcal{M}}(P_{AF})$.

$\Leftarrow$ If $M = M_1 \cup M_2$ such that $E \subseteq AR$, $M_1 = \{d(x)|x \in AR \setminus E\}$, $M_2 = \{a(x)|x \in E\}$ and M is a model of $MM^{*^r}_{\mathcal{M}}(P_{AF})$ (from Observation 1, we known that M_1 is a minimal model of P^1_{AF}), then E is a maximal conflict-free set of AF (see Observation 3 and Proposition 1 of [35]).

■

Given the set of strongly connected components $SCC(AF)$, we denote by $\leq_{SCC}$ the partial order between strongly connected components defined in [4]. This partial order is induced by the so called directionality principle and the relation of attack between set of arguments.

Theorem 5.1. Given an argumentation framework $AF = \langle AR, Attacks\rangle$, where $E \in AR$. Then, $E \in MM^{*^r}_{\mathcal{M}}(P_{AF})$ if and only if $E \in CF2(AF)$.

Proof: (sketch) Since the construction of both semantics is recursive, the proof is by induction *w.r.t.* the number of components n of the normal logic program P_{AF}.

Base Step If $n = 0$, then AF has just one strongly connected component (Lemma 5.1); hence, $MM^{*^r}_{\mathcal{M}}(P_{AF}) = CF2(AF)$ by Lemma 5.2.

Inductive Step If $n > 0$, then the proof follows from the following observations:

1. The partial order $\leq_{SCC}$ and the partial order $<_P$ correspond to each other, since the equivalent classes correspond to P^D_{AF}.
2. The base function for the construction of $MM^{*^r}_{\mathcal{M}}(P_{AF})$ and $CF2(AF)$ are the same (Lemma 5.2).

■

6. Conclusions

During the last decades, logic programming semantics with negation as failure and argumentation theory have received a special attention for formalizing common sense reasoning. Even though both approaches have their own features, it seems that they share some common behaviours which can be studied by considering the close relationship between logic programming semantics with negation as failure and extension-based argumentation semantics. In this paper, we have showed that some results in logic programming semantics can suggest some general mechanism for tackling some weakness of the extension-based argumentation semantics based on admissible sets.

In terms of logic programming with negation as failure, we have defined a general approach for extending any logic programming semantics S, such as the stable model semantics, in order to define a new logic programming semantics S^* which will be *always_consistemt (AC)* (Definition 3.4). S^* is equivalent to S for a logic program P whenever P has models in S (Lemma 3.2). By considering a process of splitting a logic program into its components, we have formalized a recursive general schema for constructing new logic programming semantics. This schema takes as input any basic logic programming semantics S, such as the stable model semantics, and gives as output a new logic programming semantics S^r which satisfies some desired properties such as relevance and AC (Lemma 3.6, Theorem 3.1).

In terms of extension-based argumentation semantics, we showed that by considering a logic programming semantics S which is AC, S can induce an extension-based argumentation semantics which is AC (Lemma 4.1). In fact, we showed that by considering S^r in terms of minimal models, S^r can characterize CF2 that is an extension-based argumentation semantics based on strongly connected components (Theorem 5.1). This suggests that our approach for generating new argumentation semantics can define new argumentation semantics with similar behaviour to the argumentation semantics defined in [4].

In logic programming side, we want to point out that the definition of logic programming semantics which are AC is not necessary a requirement for many application; however, it seem that in the exploration of formalisms for capturing common sense reasoning sometimes it is a highly desirable property. We can also see that the property of relevance has a strong influence in the definition of general approaches for capturing common sense reasoning. As we have cited, relevance (in other terms) can be found in other approaches for capturing practical reasoning such as in [4].

In the argumentation theory side, we want to point out that the identification of basic principles such as the conflict freeness can help to determine when an argumentation semantics can be considered really a well-behaved. It seems that the basic directionality principle introduced in [4] can be also motivated in terms of the property of *relevance*. This suggests that some well-accepted properties of the logic programming semantics can motivate some basic principles for the argumentation semantics.

Exploring the properties of the family of the 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.

References

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

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

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

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

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

[6] Brass, S., Zukowski, U., Freitag, B.: Transformation-Based Bottom-Up Computation of the Well-Founded Model., *NMELP*, 1996.

[7] Brewka, G.: An Abductive Framework for Generalized Logic Programs, *LPNMR*, 1993.

[8] Caminada, M.: Contamination in Formal Argumentation Systems., *BNAIC 2005 - Proceedings of the Seventeenth Belgium-Netherlands Conference on Artificial Intelligence, Brussels, Belgium, October 17-18*, 2005.

[9] Caminada, M.: Semi-Stable Semantics, *Proceedings of COMMA* (P. E. Dunne, T. J. Bench-Capon, Eds.), 144, IOS Press, 2006.

[10] Caminada, M., Sakama, C.: On the Existence of Answer Sets in Normal Extended Logic Programs., *ECAI*, 2006.

[11] Carballido, J. L., Nieves, J. C., Osorio, M.: Inferring Preferred Extensions by Pstable Semantics, *Revista Iberomericana de Inteligencia Artificial*, **13**(41), 2009, 38–53.

[12] Cormen, T. H., Leiserson, C. E., Riverst, R. L., Stein, C.: *Introduction to Algorithms*, Second edition, MIT Press, 2001.

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

[14] Denecker, M., Pelov, N., Bruynooghe, M.: Ultimate Well-founded and Stable Semantics for Logic Programs with Aggregates, *In Proceedings of ICLP-01, LNCS 2237*, Springer, 2001.

[15] Dix, J.: A Framework for Representing and Characterizing Semantics of Logic Programs, *KR*, 1992.

[16] Dix, J.: A Classification Theory of Semantics of Normal Logic Programs: I. Strong Properties., *Fundam. Inform.*, **22**(3), 1995, 227–255.

[17] Dix, J.: A Classification Theory of Semantics of Normal Logic Programs: II. Weak Properties., *Fundam. Inform.*, **22**(3), 1995, 257–288.

[18] Dix, J., Müller, M.: Partial Evaluation and Relevance for Approximations of Stable Semantics, *ISMIS*, 869, Springer, 1994.

[19] Dix, J., Müller, M.: The Stable Semantics and its Variants: A Comparison of Recent Approaches, *KI*, 1994.

[20] Dix, J., Osorio, M., Zepeda, C.: A general theory of confluent rewriting systems for logic programming and its applications., *Ann. Pure Appl. Logic*, **108**(1-3), 2001, 153–188.

[21] Dung, P. M.: On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games., *Artificial Intelligence*, **77**(2), 1995, 321–358.

[22] Dung, P. M., Mancarella, P., Toni, F.: Computing ideal sceptical argumentation, *Artif. Intell.*, **171**(10-15), 2007, 642–674.

[23] Egly, U., Gaggl, S. A., Woltran, S.: ASPARTIX: Implementing Argumentation Frameworks Using Answer-Set Programmin, *International Conference of Logic Programming (ICLP)* (M. G. de la Banda, E. Pontelli, Eds.), 5366, Springer, 2008.

[24] Ferraris, P., Lee, J., Lifschitz, V., Palla, R.: Symmetric Splitting in the General Theory of Stable Models, *IJCAI*, 2009.

[25] Gelder, A. V., Ross, K. A., Schlipf, J. S.: The Well-Founded Semantics for General Logic Programs., *Journal of the ACM*, **38**(3), 1991, 620–650.

[26] Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming, *5th Conference on Logic Programming* (R. Kowalski, K. Bowen, Eds.), MIT Press, 1988.

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

[28] Kakas, A. C., Kowalski, R. A., Toni, F.: The role of Abduction in Logic Programming, in: *Handbook in Artificial Intelligence and Logic Programming, Volume 5* (D. Gabbay, C. J. Hogger, J. A. Robinson, Eds.), Oxford University Press, Oxford, 1998, 235–324.

[29] Kakas, A. C., Mancarella, P.: Generalized Stable Models: A Semantics for Abduction, *ECAI*, 1990.

[30] L. M. Pereira, A. M. P.: Layer Supported Models of Logic Programs (extended version). Submitted to LANMR'09. Online: http://centria.fct.unl.pt/ lmp/ (Last consulted 13/08/09).

[31] Lifschitz, V., Turner, H.: Splitting a Logic Program, *ICLP*, 1994.

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

[33] Nieves, J. C.: *Modeling arguments and uncertain information — A non-monotonic reasoning approach*, Ph.D. Thesis, Software Department (LSI), Technical University of Catalonia, 2008.

[34] Nieves, J. C., Osorio, M., Cortés, U.: Inferring Preferred Extensions by Minimal Models, *Argumentation and Non-Monotonic Reasoning (LPNMR-07 Workshop)* (G. Simari, P. Torroni, Eds.), Arizona, USA, 2007.

[35] Nieves, J. C., Osorio, M., Cortés, U.: Preferred Extensions as Stable Models, *Theory and Practice of Logic Programming*, **8**(4), July 2008, 527–543.

[36] Nieves, J. C., Osorio, M., Cortés, U.: *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.

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

[38] Nieves, J. C., Osorio, M., Zepeda, C.: Expressing Extension-Based Semantics Based on Stratified Minimal Models, *WoLLIC 2009, Tokyo, Japan, June 21-24, 2009. Proceedings*, 5514, Springer, 2009.

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

[40] Osorio, M., Nieves, J. C.: $W_{s,c}$-Stable Semantics for Propositional Theories, *International Conferences of CIC'2001*, México, 2001.

[41] Pereira, L. M., Pinto, A. M.: Revised Stable Models - A Semantics for Logic Programs, *EPIA*, 3808, Springer, 2005.

[42] Pereira, L. M., Pinto, A. M.: Approved Models for Normal Logic Programs, *LPAR 2007*, 4790, Springer, 2007.

[43] Pereira, L. M., Pinto, A. M.: Layer Supported Models of Logic Programs, *LPNMR 2009*, 5753, Springer, 2009.

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

[45] Prakken, H., Vreeswijk, G. A. W.: Logics for defeasible argumentation, in: *Handbook of Philosophical Logic* (D. Gabbay, F. Günthner, Eds.), vol. 4, second edition, Kluwer Academic Publishers, Dordrecht/Boston/London, 2002, 219–318.

[46] Schlipf, J. S.: Formalizing a Logic for Logic Programming, *Ann. Math. Artif. Intell.*, **5**(2-4), 1992, 279–302.

[47] Wakaki, T., Nitta, K.: Computing Argumentatoin Semantics in Answer Set Progamming, *JSAI'2008*, 5447, 2009.

[48] Wu, Y., Caminada, M., Gabbay, D. M.: Complete Extensions in Argumentation Coincide with 3-Valued Stable Models in Logic Programming, *Studia Logica*, **93**(2-3), 2009, 383–403.