International Journal of Approximate Reasoning 64 (2015) 54–74

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

Semantic-based construction of arguments: An answer set programming approach

Esteban Guerrero, Juan Carlos Nieves *, Helena Lindgren

Department of Computing Science, Umeå University, SE-901 87, Umeå, Sweden

ARTICLE INFO

Article history:
Received 15 May 2014
Received in revised form 22 June 2015
Accepted 29 June 2015
Available online 7 July 2015

Keywords:
Argumentation
Logic programming
Well-founded semantics
Argumentation tools
Stable model semantics
Answer set programming

ABSTRACT

In this paper, we introduce an argumentation approach which takes an extended logic program as input and gives a set of arguments with the respective disagreements among them as output. We establish the notion of an argument under the Well-Founded semantics and Stable semantics inferences, allowing us to identify arguments with stratified programs as support, even when the *input* for the argument engine is a non-stratified program. We propose a set of rationality postulates for argument-based systems under extended logic programs, which are based on a definition of closure for a set of clauses that considers the well-known Gelfond–Lifschitz reduction. We establish the conditions under which our approach satisfies these principles. In addition, we present a standalone argumentation-tool based on the XSB system which implements our argumentation approach.

1. Introduction

Common-sense reasoning is the type of reasoning that a person often performs when reasoning about what to do by evaluating the potential results of the different actions that (s)he can take [1]. Since the inception of this concept in the Artificial Intelligence field by authors such as McCarthy [2], common-sense reasoning has been a fundamental goal for different reasoning approaches like *non-monotonic reasoning* (NMR). NMR captures and represents *defeasible inference*, i.e., the kind of inference in everyday life in which conclusions are drawn tentatively, reserving the right to retract them in the light of further information [3]. Indeed, a number of "non-monotonic" logics [4–7] have been developed for capturing common-sense knowledge. In this setting, *Argumentation Theory* has emerged as a formalism for dealing with non-monotonic reasoning. Dung demonstrated in his seminal paper [8], that many of the major approaches to non-monotonic reasoning in Artificial Intelligence and Logic Programming (LP) are different forms of argumentation. In this scenario, different extensions of the so-called *abstract argumentation frameworks* have been developed [9–12], providing bases and analytic tools for non-monotonic reasoning, disregarding the structure or nature of the arguments themselves. An argument is the basic and fundamental element of these frameworks with a common structure *support-conclusion*. At the same time, different approaches [13–17] have instantiated these abstract frameworks into argument-based systems (ABS), building arguments, applying a given argumentation semantics and identifying accepted conclusions. In this setting, the underlying formalisms for knowledge representation play an important role. This is because the argument structure and its underlying formalism define conditions and restrictions with respect to the management of the knowledge bases; and furthermore, in the "quality" of the conclusions of such argument-based systems.

* Corresponding author.
E-mail addresses: esteban@cs.umu.se (E. Guerrero), jcnieves@cs.umu.se (J.C. Nieves), helena@cs.umu.se (H. Lindgren).

http://dx.doi.org/10.1016/j.ijar.2015.06.009
0888-613X/

In practice, knowledge bases are sometimes inconsistent, due to the presence of rules having exceptions, or because the available knowledge comes from several sources that do not necessarily agree [18], i.e., in sensor-based systems. *Extended logic programs* (ELP) [19] use both strong negation $\neg$ and negation-as-failure *not*, representing common-sense knowledge through logic programs. ELP deals directly with incomplete information instead of predicate calculus and circumscription, as its (axiom system) syntax and semantics are more complete and computationally superior [20]. ELP is not the only approach capturing incomplete information and exceptions; other non-monotonic logic approaches have also been introduced in the literature [21–23]. Two major semantics for ELP have been defined: (1) answer set semantics [19], an extension of *Stable model semantics*, and (2) a version of the Well-Founded Semantics (WFS) [24]. Answer set programming (ASP) is a form of declarative programming oriented towards difficult, primarily NP-hard, search problems [25]. ASP is based on the Stable model (answer sets) semantics to the analysis of negation as failure, but since WFS can be viewed as an efficient and skeptical approximation of Stable [26–28], it is treated as an approximation to answer set semantics.

In argumentation literature, there is an imbalance between theoretical research and pragmatic and experimental developments. Indeed, there is a limited availability of ABS formalisms with their implementations, i.e., showing a practical use in scenarios with incomplete and inconsistent information. Moreover, some of the ABS mentioned above fail to satisfy logical closure and consistency, which are important properties in any logical reasoning system [29,30]. Against this background, we introduce an *argumentation approach* for building arguments based on the answer set programming paradigm. Our approach takes a knowledge base captured by an extended logic program as input, obtaining as a result a set of *consistent* arguments, where an *argument* entails a relationship support-conclusion. In addition, we introduce a set of rationality postulates for ABS under extended logic programs, serving as a quality recommendation for logic-based argument systems, specifically in terms of logic programming approaches.

Essentially, our argumentation approach differs from previous approaches in: (1) our language for representing knowledge is based on extended logic programs [19], allowing us to deal naturally and conveniently with incomplete information; (2) we take advantage of the remarkable properties of answer set semantics, particularly a skeptical non-monotonic reasoning semantics: the WFS [24]; (3) the proposed rationality postulates are based on the well-known Gelfond–Lifschitz reduction [31], which is the core for defining the Stable model semantics; (4) Unlike other argumentation approaches to building arguments, which are mainly syntactic-based or proof-oriented, our approach is based on logic programming semantics inferences, i.e., Stable and WFS semantics.

WFS and Stable semantics capture the common-sense notion of *negation as failure*, leading to a natural treatment of exceptions. The *relevance* property, which is satisfied by WFS, allows us to identify only those clauses relevant to building a stratified support, inferring a conclusion and avoiding problems associated with the so-called *conflict propagation* [30] or *contamination* [32,33]. Furthermore, because WFS is polynomial time computable, we have developed an implementation, which, to our knowledge, is the first argumentation engine using this approach.

In summary, the following technical contributions are made:

- A notion of argument under the Well-Founded semantics and Stable semantics inference is proposed with a stratified program as support, using any extended logic program (stratified or not) as input.
- A set of rationality postulates for argument-based systems based on extended logic programs.
- A prototype tool which may be downloaded and tested.

Regarding to the first contribution, we show also that our characterization of an argument under WFS and Stable semantics coincides, by considering a stratified program as argument support. Moreover, since different well-acceptable semantics such as Completion semantics [34,35], Perfect model semantics [36,37] and PStable semantics [38] among others, also coincide with the stable semantics in the class of stratified programs, we can use our argument building approach with different logic programming semantics and obtain the same set of arguments.

The rest of the paper is structured as follows. In Section 2, we recall the theoretical background used throughout the paper. We introduce the argument and attack relationship concepts in Section 3 and we study some of the properties of these concepts. Rationality postulates for argumentation-based systems under ELP are explored in Section 4. Our argumentation tool and its architecture are described in Section 5. We present a discussion about the related work in the state-of-the-art in Section 6. In Section 7 a summary with our contributions and future work is presented. In Appendix A, the proofs of our formal result are presented.

2. Background

In this section some background about logic programs and argumentation semantics is introduced. We assume that the reader is familiar with basic terms of Logic Programming with *negation as failure*. A basic introduction about this topic can be found in [27].

2.1. Logic programs: syntax

Let us introduce the language of a propositional logic, which is constituted by propositional symbols: $p_0, p_1, \ldots$; connectives: $\wedge, \leftarrow, \neg,\ not, \top$; and auxiliary symbols: $(\ ,\ )$, in which $\wedge, \leftarrow$ are 2-place connectives, $\neg,\ not$ are 1-place connectives

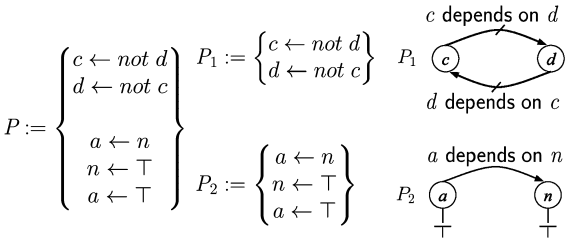

Fig. 1. A program containing a cycle through negation.

and $\top$ is a 0-place connective. The propositional symbols $\top$ and the propositional symbols of the form $\neg p_i$ ($i \geq 0$) stand for the indecomposable propositions, which we call *atoms*, or *atomic propositions*. The atoms of the form $\neg a$ are also called *extended atoms* in the literature. In order to simplify the presentation, we call them atoms as well. The negation symbol $\neg$ is regarded as the so-called *strong negation* in the Answer Set Programming literature [27], and the negation symbol *not* as *negation as failure*. A literal is an atom, a (called a positive literal), or the negation of an atom *not* a (called a negative literal). A (propositional) extended normal clause, C, is denoted:

$$a \leftarrow b_1 \wedge \cdots \wedge b_j \wedge not\ b_{j+1} \wedge \cdots \wedge not\ b_{j+n} \tag{1}$$

in which $j + n \geq 0$, a is an atom, and each b_i ($1 \leq i \leq j + n$) is an atom. We use the term *rule* as a synonym of *clause* indistinctly.

When $j + n = 0$, the clause is an abbreviation of $a \leftarrow \top$ (a *fact*), such that $\top$ is the propositional atom that always evaluates to true. In a slight abuse of notation, we sometimes write the clause (1) as a $a \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^-$, in which $\mathcal{B}^+ := \{b_1, \ldots, b_j\}$ and $\mathcal{B}^- := \{b_{j+1}, \ldots, b_{j+n}\}$. An extended logic program P is a finite set of extended normal clauses. When $n = 0$, the clause is called an *extended definite clause*. By $\mathcal{L}_P$, we denote the set of atoms which appear in P. The handling of strong negation in our logic programs will be done as it is usually done in Answer Set Programming literature [27]. Essentially, each atom of the form $\neg a$ is replaced by a new atom symbol a' that does not appear in the language of the program. A program without extended atoms will be called a *normal logic program*. Therefore, we can induce a normal logic program from an extended normal logic program by replacing each extended atom with a new symbol. For instance, let P be the program: $a \leftarrow q$; $\neg q \leftarrow r$, then, by replacing each extended atom with a new atom symbol, we will have: $a \leftarrow q$; $q' \leftarrow r$. In order not to allow inconsistent models of a logic program, usually a constraint of the form: $f \leftarrow q, q', not\ f$ is added, such that f does not appear in the program.

Let us introduce a concept which establishes a notion of *dependency* between atoms from $\mathcal{L}_P$. This concept is defined as follows:

Definition 1. (See [39].) Given an extended logic program P, a relationship: 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 set of dependencies of an atom x, denoted by dependencies_of(x), corresponds to the set $\{a \mid x \text{ depends on } a\}$. The two-place relationship *depends on* is the transitive closure of *depends immediately on*.

The concept of *dependency* as described in Definition 1 is essential in analyzing *connectivity* between clause literals. The dependencies_of(x) definition is straightforwardly extended to arbitrary atoms by setting dependencies_of($\neg x$) := dependencies_of(x) [39,40].

When positive and negative literals are *connected* generating cycles, they are "deadlocked", each waiting for the other to either succeed or fail. An example of these cycles is the loop formed by the literals c and d in the program P as shown in Fig. 1.[1] Cycles through literals have been analyzed in logic programming literature in various works [41–43]. In this setting, a class of logic programs called *stratified* programs has been identified. Informally speaking, an extended logic program presenting no cycles through negative literals will be called **stratified** and is defined as follows:

Definition 2. (See [44].) An extended logic program P is called **stratified**, if it is decomposable as $P = P_1 \cup \cdots \cup P_n$, such that the following holds (for $i = 1, \ldots, n$):

1. If a literal x occurs *positively* in a clause in P_i, then all clauses containing x in their heads are contained in $\bigcup_{j \leq i} P_j$.
2. If a literal x occurs *negatively* in a clause in P_i, then all clauses containing x in their heads are contained in $\bigcup_{j < i} P_j$.

[1] Graphical notation: we represent a dependency relationship through negation as failure with the arrow: ↷ and a dependency through positive atoms with the arrow: ⌒.

An example of a *stratified program* is the subprogram P_2 introduced in Fig. 1, in contrast with P_1, which contains a cycle through the negative literal c and d and is not a stratified logic program.

2.2. Logic programs: semantics

In this section, we introduce a couple of well-known logic programming semantics, namely answer set semantics [19], an extension of *Stable model semantics*, and a version of the Well-Founded Semantics. To this end, we start presenting some definitions of interpretations and models.

An *interpretation* of a propositional signature $\mathcal{L}_P$ is a function from $\mathcal{L}_P$ to {false,true}. A partial interpretation (also called 3-valued interpretation), based on a signature $\mathcal{L}_P$, is a disjoint pair of sets $\langle I_1, I_2\rangle$, such that $I_1 \cup I_2 \subseteq \mathcal{L}_P$. A partial interpretation is total if $I_1 \cup I_2 = \mathcal{L}_P$. An interpretation I of a given logic program P is a *model* for P iff $I(C) = true$ for each clause $C \in P$. I is a *minimal model* of P if a model I' of P different from I such that $I' \subset I$ does not exist.

2.2.1. Well-founded semantics

In this section, we present a standard definition of the well-founded semantics in terms of rewriting systems. We start presenting a definition *w.r.t.* 3-valued logic semantics.

Definition 3 *(SEM).* (See [45].) For normal logic program P, we define $\text{HEAD}(P) = \{a |\ a \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^-\ \in P\}$ – the set of all head-atoms of P. We also define $\text{SEM}(P) = \langle P^{true}, P^{false}\rangle$, where $P^{true} := \{p |\ p \leftarrow \top \in P\}$ and $P^{false} := \{p |\ p \in \mathcal{L}_P \backslash \text{HEAD}(P)\}$. $\text{SEM}(P)$ is also called model of P.

In order to present a characterization of the well-funded semantics in terms of rewriting systems, we define some basic transformation rules for normal logic programs. To this end, let us introduce the following notation: given a signature $\mathcal{L}$, $\text{Prog}_{\mathcal{L}}$ denotes the set of all normal programs defined over $\mathcal{L}$.

Definition 4 *(Basic transformation rules).* (See [45].) A transformation rule is a binary relation on $\text{Prog}_{\mathcal{L}}$. The following transformation rules are called *basic*. Let a program $P \in \text{Prog}_{\mathcal{L}}$ be given.

***RED*$^+$:** This transformation can be applied to P if there is an atom a which does not occur in $\text{HEAD}(P)$. **RED**$^+$ transforms P to the program where all occurrences of *not* a are removed.

***RED*$^-$:** This transformation can be applied to P, if there is a rule $a \leftarrow \top \in P$. **RED**$^-$ transforms P to the program where all clauses that contain *not* a in their bodies are deleted.

Success: Suppose that P includes a fact $a \leftarrow \top$ and a clause $q \leftarrow body$ such that $a \in body$. Then we replace the clause $q \leftarrow body$ with $q \leftarrow body \setminus \{a\}$.

Failure: Suppose that P contains a clause $q \leftarrow body$ such that $a \in body$ and $a \notin \text{HEAD}(P)$. Then we erase the given clause.

Loop: We say that P_2 results from P_1 by $Loop_A$ if, by definition, there is a set A of atoms such that:

1. for each rule $a \leftarrow body \in P_1$, if $a \in A$, then $body \cap A \neq \emptyset$,
2. $P_2 := \{a \leftarrow body \in P_1 | body \cap A = \emptyset\}$,
3. $P_1 \neq P_2$.

One can observe that the application of the loop transformation depends on the set A which is basically an *unfounded set* [24]. Informally speaking, an unfounded set consists of positive atoms which are not possibly true, i.e., they cannot be derived by assuming all negative literals to be true. In this sense, one can see that by considering the greatest unfounded set of P_1 for defining A, the loop transformation maximizes the number of rules which can be removed from P_1. This means that if P_2 results from P_1 by using the maximal A, one cannot apply once again Loop A to P_2. In [46], a general method for computing the maximal A of a given logic program was introduced. The algorithm for computing the maximal A (which means the greatest unfounded set of P_1) works as follows:

1. $P' = \{a \leftarrow \mathcal{B}^+ | a \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^- \in P_1\}$
2. Let MM_P be the minimal model of P'.[2]
3. $A = \mathcal{L}_{P_1} \setminus MM_P$

By considering A, one can get P_2 as follows:

$$P_2 = \{a \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^- | a \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^- \in P_1 \text{ and } \mathcal{B}^+ \cap A = \emptyset\}$$

In the rest of the paper, whenever the loop transformation is applied, we assume that the greatest unfounded set was used for defining A.

[2] Since P' is a definite program P' has a unique minimal model.

Let CS_0 be the rewriting system such that it contains the transformation rules: RED^+, RED^-, *Success*, *Failure*, and *Loop*. We denote the uniquely determined normal form of a program P with respect to the system CS_0 by $norm_{CS_0}(P)$. CS_0 induces a logic programming semantics as follows: Let P be a normal logic program:

$$\mathrm{SEM}_{CS_0}(P) := \mathrm{SEM}(norm_{CS_0}(P))$$

In [47], it was showed that $\mathrm{SEM}_{CS_0}(P)$ characterizes the Well-Founded Semantics [48]. This characterization is formalized as follows:

Lemma 2.1. *(See [47].) CS_0 is a confluent rewriting system. It induces a 3-valued semantics that is the Well-founded Semantics.*

Let us observe that Lemma 2.1 is characterizing WFS for the class of normal logic programs. Hence, to use this definition in the class of extended logic programs, we assume that the extended logic programs are transformed into normal logic programs by replacing extended atoms with new symbols.

WFS performs a skeptical reasoning approach, satisfying a good set of expected properties in the context of the non-monotonic reasoning, such as being both polynomial time computable and always defined. Nowadays, there are several solvers which can efficiently compute WFS i.e., DLV,[3] SMODELS,[4] XSB.[5]

2.2.2. Stable model semantics

Stable model semantics is one of the most influential logic programming semantics in the non-monotonic reasoning community and is defined as follows:

Definition 5. (See [31].) Let P be a normal logic program. For any set $S \subseteq \mathcal{L}_P$, let P^S be the definite logic program obtained from P by deleting

(i) each rule that has a formula *not* l in its body with $l \in S$, and then
(ii) all formulæ of the form *not* l in the bodies of the remaining rules.

Hence S is a stable model of P iff S is a minimal model of P^S. $STABLE(P)$ denotes the set of stable models of P

From this point on, whenever we say Gelfond–Lifschitz (GL) reduction, we mean the reduction P^S. As we can observe, GL reduction is the core of the stable model semantics.

2.3. Argumentation semantics

In this section, we introduce the definition of some argumentation semantics (an overview about abstract argumentation semantics can be found in [49]). To this end, we begin by defining the basic structure of an argumentation framework.

Definition 6. (See [8].) 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 say that a *attacks* b (or b is attacked by a) if $attacks(a, b)$ holds. Similarly, we say that a set S of arguments attacks b (or b is attacked by S) if b is attacked by an argument in S.

Let us observe that an argumentation framework is a simple structure that captures the conflicts of a given set of arguments. In order to select *coherent points of view* from a set of conflicts of arguments, Dung introduced a set of *patterns of selection of arguments*. These patterns of selection of arguments were called *argumentation semantics*. The sets of arguments suggested by an argumentation semantics are called *extensions*. Dung defined his argumentation semantics based on the basic concept of *admissible sets*:

Definition 7. (See [8].) 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. 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.

[3] http://www.dbai.tuwien.ac.at/proj/dlv/.
[4] http://www.tcs.hut.fi/Software/smodels/.
[5] http://xsb.sourceforge.net/.

Definition 8. (See [8].) Let $AF := \langle AR, attacks \rangle$ be an argumentation framework. An admissible set of arguments $S \subseteq AR$ is:

- *stable* if and only if S attacks each argument which does not belong to S.
- *preferred* if and only if S is a maximal (*w.r.t.* inclusion) admissible set of AF.
- *complete* if and only if each argument, which is acceptable with respect to S, belongs to S.
- the *grounded* extension of AF if and only if S is the minimal (*w.r.t.* inclusion) complete extension of AF.

The grounded semantics definition presented above is not the one originally introduced in [8]; however, it was shown that the grounded extension can be defined in terms of complete extensions [50]. Hereon whenever we say *Dung's standard semantics* we refer to the argumentation semantics introduced by Definition 8.

3. Answer set based argumentation

In this section, we introduce the central definition of an argument and establish some of its fundamental properties, such as the *stratified support* of each argument. We start defining the process of building arguments regarding WFS and stable semantics, then we introduce the concept of *attack* between arguments.

3.1. Building arguments

Let us start introducing the notion of an *argument* based on WFS:

Definition 9. Given an extended logic program P and $S \subseteq P$. $Arg_P = \langle S, g \rangle$ is an **argument** under WFS, if the following conditions hold:

1. $WFS(S) = \langle T, F \rangle$ such that $g \in T$.
2. S is minimal w.r.t. the set inclusion satisfying 1.
3. $\nexists\, g \in \mathcal{L}_P$ such that $\{g, \neg g\} \subseteq T$ and $WFS(S) = \langle T, F \rangle$.

By $\mathcal{A}rg_P$ we denote the set of all of the arguments built from P.

Let us now examine some properties of the arguments built according to Definition 9. First of all, let us observe that, given an argument $\langle S, g \rangle$, S is usually called *support* and g *conclusion*; moreover, more than one argument with the same conclusion could exist. An argument built under Definition 9 entails a relationship between *support* and *conclusion*.

Let us observe that the *support* sets are minimal subsets of the *relevant* clauses *w.r.t.* the atom g (Definition 10). *Relevance* is a property that WFS satisfies [39] and states, informally speaking, that it is perfectly reasonable that the truth-value of an atom, with respect to any semantics, only depends on the *subprogram* formed from the *relevant clauses* with respect to that specific atom [40]. An example of relevance is introduced in the program P of Fig. 1, where the truth-values of atoms c and d do not depend on the values of a and n. Let us define a mechanism for obtaining those *relevant clauses w.r.t.* a given atom.

Definition 10. (See [39].) Let P be an extended logic program and $x \in \mathcal{L}_P$. $\mathsf{rel\_rules}(P, x)$ is a function which returns the set of clauses that contain an $a \in \mathsf{dependencies\_of}(x)$ in their heads.

The function $\mathsf{rel\_rules}(P, x)$ minimizes the search space for establishing *candidate clauses* for supporting arguments *w.r.t.* an atom x. Each *relevant rule* centralizes a set of knowledge for creating/supporting arguments. This concept resembles the idea of *epicenter* in [51],[6] where arguments are built using *candidate clauses* contained only in the same *epicenter*. Let us illustrate this important concept with an example:

Example 1. Let us consider the program P introduced in Fig. 1. The relevant rules for inferring the atom c (the epicenter of c) are: $\mathsf{rel\_rules}(P, c) = \{c \leftarrow not\ d,\ d \leftarrow not\ c\}$, presented in Fig. 2.

Bearing in mind that the support of each argument with conclusion x is a subset of the relevant clauses of x, let us observe that the support of each argument constructed according to Definition 9 is *a stratified logic program*.

Proposition 1. *Let P be an extended logic program. If $\langle S, g \rangle \in \mathcal{A}rg_P$, then S is a stratified program.*[7]

[6] In [51], the authors define an epicenter as a part of a *focal graph* which is defined as a subgraph of the closed graph for a knowledge base (the full graph), which is specified by a clause from the knowledge base and corresponds to the part of the closed graph that contains such a clause.

[7] Proofs for propositions can be found in Appendix A.

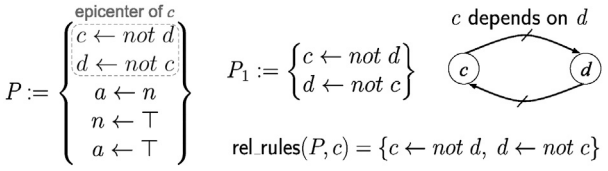

Fig. 2. Example of relevant rules for a given atom.

The importance of Proposition 1 lies in the fact that, given any extended logic program P, we can unequivocally identify the existence of a set of clauses supporting a given conclusion under WFS. Indeed, we can obtain a stratified program as support of an argument from any (stratified or not) input program.

Let us illustrate part of the process for building arguments according to Definition 9, by considering the program introduced in Fig. 1 and inferring arguments for atom c.

Example 2. Given the program P introduced in Example 1, an evaluation of relevant rules for c is:

$$\text{rel\_rules}(P, c) = \{c \leftarrow not\ d,\ d \leftarrow not\ c\}$$

All the possible sub-sets for building an argument support are given by:

$$2^{\text{rel\_rules}(P,c)} = \{\{\},\ \{c \leftarrow not\ d\},\ \{d \leftarrow not\ c\}, \{c \leftarrow not\ d,\ d \leftarrow not\ c\}\}$$

Let us define $S_0 = \{\}$, $S_1 = \{c \leftarrow not\ d\}$, $S_2 = \{d \leftarrow not\ c\}$ and $S_3 = \{c \leftarrow not\ d,\ d \leftarrow not\ c\}$. By considering each subset S_i $(0 \leq i \leq 3)$ under WFS evaluation, we obtain:

$$WFS(S_0) = \langle\{\}, \{\}\rangle$$
$$WFS(S_1) = \langle\{c\}, \{d\}\rangle$$
$$WFS(S_2) = \langle\{d\}, \{c\}\rangle$$
$$WFS(S_3) = \langle\{\}, \{\}\rangle$$

According to these evaluations, it is clear that the only candidate set to be a support set for an argument with conclusion c is S_1. Since S_1 is consistent and minimal, an argument for c considering P is:

$$Arg = \langle S_1, c\rangle = \langle\{c \leftarrow not\ d\},\ c\rangle$$

Given that *WFS* infers total interpretations from stratified logic programs, we have the following straightforward corollary with respect to arguments.

Corollary 1. *Let P be an extended logic program and $Arg \in \mathcal{A}rg_P$ such that $Arg = \langle S,\ g\rangle$. If $WFS(S) = \langle T,\ F\rangle$, then $T \cup F$ is a total interpretation of S.*

Moreover, since it is known that every stratified logic program has exactly one stable model [31], we have the following corollary.

Corollary 2. *Let P be an extended logic program and $Arg \in \mathcal{A}rg_P$ such that $Arg = \langle S,\ g\rangle$. S has a unique stable model.*

Corollary 2 establishes a relevant relationship between WFS and stable semantics in the construction of arguments. Indeed, the notion of argument introduced in Definition 9 can be expressed under the stable semantics by considering only minimal subsets of the original program that are stratified:

Definition 11. Given an extended logic program P and $S \subseteq P$. $Arg_P = \langle S,\ g\rangle$ is an **argument** under the stable semantics, if the following conditions hold:

1. $g \in M$ such that $M \in STABLE(S)$,
2. S is a stratified logic program,
3. S is minimal w.r.t. the set inclusion satisfying 1,
4. $\nexists\, g \in \mathcal{L}_P$ such that $\{g, \neg g\} \subseteq M$ and $M \in STABLE(S)$.

In order to show the coincidences between Definitions 9 and 11, let us consider the following example:

$$R := \left\{ \begin{array}{c} a \leftarrow not\ b \\ b \leftarrow not\ a \\ c \leftarrow not\ c \wedge not\ a \\ d \leftarrow not\ d \wedge not\ b \end{array} \right\}$$

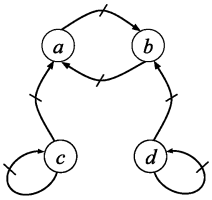

Fig. 3. Example program, coincidence between arguments under WFS and STABLE evaluation.

Table 1
WFS and stable models evaluation of Example 3.

Conclusion	a	
Rel. rules	$\{a \leftarrow not\ b;\ b \leftarrow not\ a\}$	
WFS		*STABLE*
$WFS(S_0) = \langle\{\}, \{\}\rangle$		$STABLE(S_0) = \{\}$
$WFS(S_1) = \langle\{a\}, \{b\}\rangle$		$STABLE(S_1) = \{\{a\}\}$
$WFS(S_2) = \langle\{b\}, \{a\}\rangle$		$STABLE(S_2) = \{\{b\}\}$
$WFS(S_3) = \langle\{\}, \{\}\rangle$		$STABLE(S_3) = \{\{a\}, \{b\}\}$
Arguments for a		
$\langle\{\underbrace{a \leftarrow not\ b}_{S_1}\}, a\rangle$		$\langle\{\underbrace{a \leftarrow not\ b}_{S_1}\}, a\rangle$

Conclusion	c	
Rel. rules	$\{c \leftarrow not\ c,\ not\ a;\ a \leftarrow not\ b;\ b \leftarrow not\ a\}$	
WFS		*STABLE*
$WFS(S'_0) = \langle\{\}, \{\}\rangle$		$STABLE(S'_0) = \{\}$
$WFS(S'_1) = \langle\{\}, \{a\}\rangle$		$STABLE(S'_1) = \{\}$
$WFS(S'_2) = \langle\{a\}, \{b\}\rangle$		$STABLE(S'_2) = \{\{a\}\}$
$WFS(S'_3) = \langle\{b\}, \{a\}\rangle$		$STABLE(S'_3) = \{\{b\}\}$
$WFS(S'_4) = \langle\{a\}, \{b, c\}\rangle$		$STABLE(S'_4) = \{\{a\}\}$
$WFS(S'_5) = \langle\{b\}, \{a\}\rangle$		$STABLE(S'_5) = \{\}$
$WFS(S'_6) = \langle\{\}, \{\}\rangle$		$STABLE(S'_6) = \{\{a\}, \{b\}\}$
$WFS(S'_7) = \langle\{\}, \{\}\rangle$		$STABLE(S'_7) = \{\{a\}\}$
Arguments for c		
No arguments		No arguments

Example 3. Let R be the logic program introduced in Fig. 3. The sets of all possible clauses involved in the inference of the atoms a and c are:

$$2^{\text{rel\_rules}(R,a)} = \{\underbrace{\{\}}_{S_0}, \underbrace{\{a \leftarrow not\ b\}}_{S_1}, \underbrace{\{b \leftarrow not\ a\}}_{S_2}, \underbrace{\{a \leftarrow not\ b,\ b \leftarrow not\ a\}}_{S_3}\}$$

$$\begin{aligned} 2^{\text{rel\_rules}(R,c)} = \{&\underbrace{\{\}}_{S'_0}, \underbrace{\{c \leftarrow not\ c \wedge not\ a\}}_{S'_1}, \underbrace{\{a \leftarrow not\ b\}}_{S'_2}, \underbrace{\{b \leftarrow not\ a\}}_{S'_3}, \\ &\underbrace{\{c \leftarrow not\ c \wedge not\ a,\ a \leftarrow not\ b\}}_{S'_4}, \\ &\underbrace{\{c \leftarrow not\ c \wedge not\ a,\ b \leftarrow not\ a\}}_{S'_5}, \\ &\underbrace{\{a \leftarrow not\ b,\ b \leftarrow not\ a\}}_{S'_6}, \\ &\underbrace{\{c \leftarrow not\ c \wedge not\ a,\ a \leftarrow not\ b,\ b \leftarrow not\ a\}}_{S'_7}\} \end{aligned}$$

In Table 1, the WFS and stable models evaluations are summarized as well as the set of arguments obtained for the conclusions a and c.

Table 2
Set of possible arguments built from program R in Example 3.

Arguments for program R	
WFS	*STABLE*
$\langle\{a \leftarrow not\ b\}, a\rangle$	$\langle\{a \leftarrow not\ b\}, a\rangle$
$\langle\{b \leftarrow not\ a\}, b\rangle$	$\langle\{b \leftarrow not\ a\}, b\rangle$

We now follow with some remarks with respect to these evaluations of WFS and stable model semantics. First, let us observe the evaluations of the potential sets of clauses, which can be a support of an argument with a conclusion a under the inference of WFS. We can see that S_1 is the only set that infers a; moreover, it is minimal. Hence, $\langle S_1, a\rangle$ is the only argument with conclusion a under the inference of WFS. On the other hand, we can see that both $STABLE(S_1)$ and $STABLE(S_3)$ have a stable model which contains the atom a. However, S_3 is not a stratified logic program, and is not minimal. Therefore, once again, the only argument with conclusion a under the inference of stable models is $\langle S_1, a\rangle$.

Observing the evaluations of the sets of clauses which can be a support of an argument with conclusion c, we can observe that none of them have a model which contains atom c. Hence, there is no argument with conclusion c under either WFS or stable model semantics.

The sets of all possible arguments which can be built from R with respect to WFS and stable model semantics are depicted in Table 2.

We can establish the coincidence between Definition 9 and Definition 11 for building arguments through the following proposition:

Proposition 2. *Let P be an extended logic program, $Arg_{WFS}(P)$ be the set of arguments from P according to Definition 9 and $Arg_{STABLE}(P)$ be the set of arguments from P according to Definition 11. The following condition holds:*

$$Arg_{WFS}(P) = Arg_{STABLE}(P)$$

Proposition 2 is quite important, since it is arguing that we can use WFS and Stable semantics to build arguments from an extended logic program and obtain the same set of arguments. Indeed, Definition 11 identifies the basic requirements for using a wide family of logic programming semantics to build arguments and coincide with our approach. Let us remember that several well-accepted semantics, such as Completion semantics [34,35,52], Perfect model semantics [36,37] and PStable semantics [38],[8] among others, coincide with the stable semantics in the class of stratified semantics. Therefore, we can use Definition 11 with different logic programming semantics and obtain the same set of arguments.

An interesting behavior of our construction of arguments is that, by adding a set of clauses to the knowledge base (i.e. an extended logic program P), the number of arguments increases monotonically. This property is described as follows:

Proposition 3 *(Monotonous argument building). Let P and G be extended logic programs, the following condition holds: $\mathcal{A}rg_P \subseteq \mathcal{A}rg_{P\cup G}$.*

An intuitive and direct consequence of Proposition 3 is a monotonous increase of the candidate clauses for supporting arguments, even when this expansion involves atoms that have already been inferred.

Now let us observe that the support of an argument can be a subset of another argument. In this sense, we define the concept of *sub-arguments*.

Definition 12. Let $A = \langle S_A, g_A\rangle$, $B = \langle S_B, g_B\rangle$ be two arguments. A is a sub-argument of B if and only if $S_A \subset S_B$.

Example 4. Let us consider the program P introduced in Fig. 1. We can see that there are two arguments for the atom a and one argument for the atom n:

$$\begin{aligned} Arg_1 &= \langle\{a \leftarrow \top\}, a\rangle \\ Arg_2 &= \langle\{a \leftarrow n, n \leftarrow \top\}, a\rangle \\ Arg_3 &= \langle\{n \leftarrow \top\}, n\rangle \end{aligned}$$

As we can see, Arg_3 is a sub-argument of Arg_2.

Another property of the support of an argument is that it does not contain *tautologies*.

[8] In [28], a survey about Fixpoint semantics (approaches based on fixpoint theory) is introduced, analyzing coincidences between different approaches including WFS, Stable among others.

Proposition 4. *Let $A = \langle S, g\rangle$ be an argument. If $a \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^- \in S$ then $a \notin \mathcal{B}^+$ and $\mathcal{B}^+ \cap \mathcal{B}^- = \emptyset$.*

In order to illustrate this proposition, let us consider the following program:

$$a \leftarrow b \wedge not\ b \qquad a \leftarrow a \qquad a \leftarrow \top$$

We can think that $a \leftarrow b \wedge not\ b$ is a tautology which could suggest an argument of the form: $\langle\{a \leftarrow b \wedge not\ b\}, a\rangle$. However, this argument is not feasible to be built given that $WFS(\{a \leftarrow b \wedge not\ b\}) = \langle\{\}, \{a, b\}\rangle$. Moreover, even though $a \leftarrow a$ can be regarded as a tautology in classical logic, the argument $\langle\{a \leftarrow a\}, a\rangle$ cannot be built since $WFS(\{a \leftarrow a\}) = \langle\{\}, \{a\}\rangle$. The only argument which can be built is: $\langle\{a \leftarrow \top\}, a\rangle$.

A direct corollary from Proposition 4 and Proposition 1 is that the support of an argument does not contain cycles between their both positive and negative literals.

Corollary 3. *Let $A = \langle S, g\rangle$ be an argument and S be decomposable as $S = S_1 \cup \cdots \cup S_n$. The following conditions hold (for $i = 1, \ldots, n$):*

1. *If a literal* x *occurs positively in a clause in P_i, then all clauses containing* x *in their heads are contained in $\bigcup_{j<i} P_j$.*
2. *If a literal* x *occurs negatively in a clause in P_i, then all clauses containing* x *in their heads are contained in $\bigcup_{j<i} P_j$.*

3.2. Argument attacks

Relationships between arguments are defined by the concept of *attack*. Intuitively, an attack between arguments emerges whenever there is a *disagreement* between arguments. Considering our construction of an argument, the attack relationship between arguments is defined as follows:

Definition 13 *(Attack relationship between arguments).* Let $A = \langle S_A, g_A\rangle$, $B = \langle S_B, g_B\rangle$ be two arguments such that $WFS(S_A) = \langle T_A, F_A\rangle$ and $WFS(S_B) = \langle T_B, F_B\rangle$. We say that A attacks B if one of the following conditions holds:

1. $a \in T_A$ and $\neg a \in T_B$.
2. $a \in T_A$ and $a \in F_B$.

$At(\mathcal{A}rg)$ denotes the set of attack relationships between the arguments belonging to the set of arguments $\mathcal{A}rg$.

Let us observe that this definition of attack relationship between arguments is totally based on *semantic interpretations* of the supports of the given arguments. In particular, Definition 13 considers the well-founded models of the supports of the arguments without considering the internal *syntactic structure* of the arguments. In other words, the definition of an attack is based only on the information which is inferred from the supports of the arguments.

In terms of sub-arguments and attacks, we can observe the following properties:

Proposition 5. *Let A, B and C be arguments such that B is a sub-argument of A. The following conditions hold:*

a) *If B attacks C then A attacks C.*
b) *If C attacks B then C attacks A.*

The properties of attacks and sub-arguments which are identified by Proposition 5 are not really unexpected since the clauses of the support of the sub-argument B also appear in the support of the argument A.

Let us observe that, whenever the knowledge base increases, the number of attacks can also increase.

Proposition 6. *Let P and G be extended logic programs. Then, the following condition holds: $At(\mathcal{A}rg_P) \subseteq At(\mathcal{A}rg_{P\cup G})$*

An intuitive consequence of Proposition 6 establishes a monotonous behavior of the attack relationships, increasing the number of possible attacks when the knowledge base is expanded.

Now that we have defined the concepts of arguments and attacks, let us define the concept of argumentation framework with respect to an extended logic program as follows:

Definition 14. Let P be an extended logic program. The resulting argumentation framework w.r.t. P is the tuple: $AF_P = \langle \mathcal{A}rg_P, At(\mathcal{A}rg_P)\rangle$.

As we can observe, the resulting argumentation framework from an extended logic program resembles the classic definition of an argumentation framework introduced in Definition 6. One interesting property that satisfies any argumentation

framework AF_P is that AF_P has no self-attacked arguments. In order to formalize this property, let us define the equality between arguments. Given two arguments $A = \langle S_A, c_A \rangle$ and $B = \langle S_B, c_B \rangle$, we say that $A = B$ *iff* $S_A = S_B$ and $c_A = c_B$.

Proposition 7. *Let P be an extended logic program and $AF_P = \langle Arg_P, At(Arg_P) \rangle$ be the resulting argumentation framework w.r.t. P. If $(A, B) \in At(Arg_P)$, then $A \neq B$.*

Proposition 7 argues that, with our approach, an argument cannot be attacked by itself. In other words, by considering a semantic-oriented approach (e.g., using WFS or Stable models semantics) we avoid self-attacked arguments.

Following Dung's style, we define an argumentation semantics SEM_{Arg} as a function from an argumentation framework $AF_P = \langle Arg_G, At(Arg_G) \rangle$ to $2^{2^{Arg_P}}$. Hence, SEM_{Arg} can be instantiated with any of the argumentation semantics introduced in Section 2.3. This means that we can apply Dung's style semantics with the argumentation framework induced by an extended logic program.

4. Closure and consistency analysis

In argumentation theory literature, different authors have analyzed the *rationality* of different argumentation systems, e.g., ASPIC, ASPIC$^+$ [29,30,53]. In [29], four postulates were introduced: *sub-argument closure*, *closure*, *direct consistency* and *indirect consistency*, which can be used to judge the quality of a rule-based argumentation system. These postulates are formulated in terms of an eventual and intuitive *conclusion* of an argumentation-based system and its final *output*.

In this section, we study the four postulates introduced by Caminada and Amgoud [29]. To this end, we introduce *new definitions* of these postulates in terms of extended normal logic programs and arguments. Moreover, we identify the conditions under which our argumentation system satisfies these postulates. We start with the postulate of *sub-argument closure*.

Postulate 1 *(Sub-argument closure). Let* P *be an extended logic program, $AF_P = \langle Arg_P, At(Arg_P) \rangle$ be the resulting argumentation framework from* P *and SEM_{Arg} be an argumentation semantics such that $E \in SEM_{Arg}(AF_P)$. If $B, A \in Arg_P$, B is a sub-argument of A and $A \in E$ then $B \in E$.*

Intuitionally this postulate argues that whenever an argumentation semantics suggests that a given argument belongs to a given extension E, its respective sub-arguments also must belongs to E. We can show that our approach satisfies sub-argument closure by considering Dung's standard semantics.

Proposition 8. *Let* P *be an extended logic program, $AF_P = \langle Arg_P, At(Arg_P) \rangle$ be the resulting argumentation framework from* P *and SEM_{Arg} be an argumentation semantics. If SEM_{Arg} is a Dung's standard semantics, then AF_P satisfies sub-argument closure w.r.t. SEM_{Arg}.*

The sub-argument closure is only concerned about sub-arguments without taking care of their conclusions. In order to evaluate the conclusions of an argumentation semantics with respect to an argumentation system the concept of closure with respect to a set of clauses takes relevance. In order to define a closure condition of a set of clauses, we will consider the Gelfond–Lifschitz reduction (see Definition 5).

Definition 15 *(Closure of a set of clauses).* Let P be an extended logic program and $S \subseteq \mathcal{L}_P$. The closure of S under the set of extended normal clauses $G \subseteq P$, denoted by $Cl_G(S)$, is the smallest set such that:

- $S \subseteq Cl_G(S)$.
- if $a \leftarrow b_1 \wedge \cdots \wedge b_j \in G^S$ and for all $b_i \in Cl_G(S)$ such that $i \in \{1, \ldots, j\}$ then $a \in Cl_G(S)$.

If $S = Cl_G(S)$, then S is said to be closed under the set G.

Definition 15 is not the same to the definition of closure introduced by [29], which is defined in terms of the so-called *strict rules*. Since normal clauses contain *negation as failure*, we use the Gelfond–Lifschitz transformation to reduce a set of extended normal clauses into a definite program.

In order to formalize the other three rationality postulates introduced by [29] in terms of extended logic programs, let us define the function Conc(*Arg*), which returns the conclusion of a given argument *Arg*, i.e., given an argument $Arg = \langle S, g \rangle$, then $\text{Conc}(Arg) = g$.

Definition 16 *(Consistent set of clauses).* Let P be an extended logic program and $G \subseteq P$. G is consistent if and only if $\nexists\, A,\ B \in Arg_G$, such that A attacks B.

One of the key concepts when evaluating the consistency of conclusions of argumentation semantics with respect to an *argumentation framework* is the set of *justified conclusions* which are suggested by the given argumentation semantics. This set of justified conclusions will be denoted by the set Output and is defined in terms of the resulting argumentation framework built from a given extended logic program.

Definition 17 *(Justified conclusions).* Let P be an extended logic program, $AF_P = \langle \mathcal{A}rg_P, At(\mathcal{A}rg_P)\rangle$ be the resulting argumentation framework from P and SEM_{Arg} be an argumentation semantics.

If $\text{SEM}_{Arg}(AF_P) = \{E_1, \ldots, E_n\}(n \geq 1)$, then

- $\text{Concs}(E_i) = \{\text{Conc}(A) \mid A \in E_i\}(1 \leq i \leq n)$.
- $\text{Output} = \bigcap_{i=1\ldots n} \text{Concs}(E_i)$.

Let us note that the notion of *justified conclusions*, Output in Definition 17 (derived from [29]), coincides with a skeptical approach. This concept is used to define the postulate of *Closure* as follows:

Postulate 2 *(Closure). Let* P *be an extended logic program,* $AF_P = \langle \mathcal{A}rg_P, At(\mathcal{A}rg_P)\rangle$ *be the resulting argumentation framework from* P *and* SEM_{Arg} *be an argumentation semantics.* Output *is the set of justified conclusions with respect to* $SEM_{Arg}(AF_P) = \{E_1, \ldots, E_n\}$. AF_P *satisfies closure w.r.t.* SEM_{Arg} *iff:*

1. $\text{Concs}(E_i) = Cl_G(\text{Concs}(E_i))$ *for each* $1 \leqslant i \leqslant n$, *such that* $G = \bigcup_{\langle S,g\rangle \in E_i} S$.
2. $\text{Output} = Cl_{G'}(\text{Output})$ *such that* $\text{Output}' = \bigcap_{i=1\ldots n} E_i$ *and* $G' = \bigcup_{\langle S,g\rangle \in Output'} S$.

We can see that the first condition is considering an extended logic program G which is the union of the argument-supports which appear in a given extension. Hence, G is the extended logic program which infers all the conclusions of a given extension. In this sense, closure aims to show that the set of conclusions of the arguments which appear in an extension is the only information that can be inferred by considering the whole knowledge base which appears in a given extension. Indeed, below we show that the information which is inferred from G is also consistent (see Propositions 11 and 12). The second condition of closure considers an extended logic program G' which is the union of the argument-supports which infers Output. This means that G' is denoting a subset of all the programs G which appears in all the extension of $\text{SEM}_{Arg}(AF_P)$.

Let us observe that if we consider a set of conflict-free set of arguments E, the resulting program G of the union of argument-supports which appear in E is a stratified logic program. Indeed, we can infer the well founded model of G by considering the well-founded models of the argument-supports which appears in E.

Proposition 9. *Let* E *be a conflict-free set of arguments and* $G = \bigcup_{\langle S,g\rangle \in E} S$. *The following conditions hold:*

1. G *is a stratified logic program.*
2. $WFS(G) = \bigsqcup_{\langle S,g\rangle \in E} WFS(S)$ *such that* $\langle T_1, F_1\rangle \sqcup \langle T_1, F_1\rangle = \langle T_1 \cup T_2, F_1 \cup F_2\rangle$

Proposition 9 identifies a class of argumentation semantics which satisfies closure. This class of argumentation semantics will be *conflict free semantics*. We say that an argumentation semantics is conflict-free if for all $E \in \text{SEM}_{Arg}(AF_P)$, $A, B \in E$ do not exist such that A attacks B.

Proposition 10. *Let* P *be an extended logic program,* $AF_P = \langle \mathcal{A}rg_P, At(\mathcal{A}rg_P)\rangle$ *be the resulting argumentation framework from* P *and* SEM_{Arg} *be an argumentation semantics. If* SEM_{Arg} *is a conflict-free semantics, then* AF_P *satisfies closure w.r.t.* SEM_{Arg}.

Another postulate which was introduced by [29] is *direct consistency*. This postulate argues that the conclusions of an extension is a consistent set of atoms; moreover, the set Output is also a consistent set of atoms. We introduce a definition of this postulate in terms of extended logic programs as follows:

Postulate 3 *(Direct Consistency). Let* P *be an extended logic program,* $AF_P = \langle \mathcal{A}rg_P, At(\mathcal{A}rg_P)\rangle$ *be the resulting argumentation framework from* P *and* SEM_{Arg} *be an argumentation semantics.* Output *is the set of justified conclusions with respect to* $SEM_{Arg}(AF_P) = \{E_1, \ldots, E_n\}$. AF_P *satisfies direct consistency w.r.t.* SEM_{Arg} *iff:*

1. $\text{Concs}(E_i)$ *is consistent for each* $1 \leqslant i \leqslant n$.
2. Output *is consistent.*

As we can observe, satisfying direct consistency is not really hard in our approach. Indeed, by considering argumentation semantics which are conflict-free, we can guarantee the satisfaction of direct consistency.

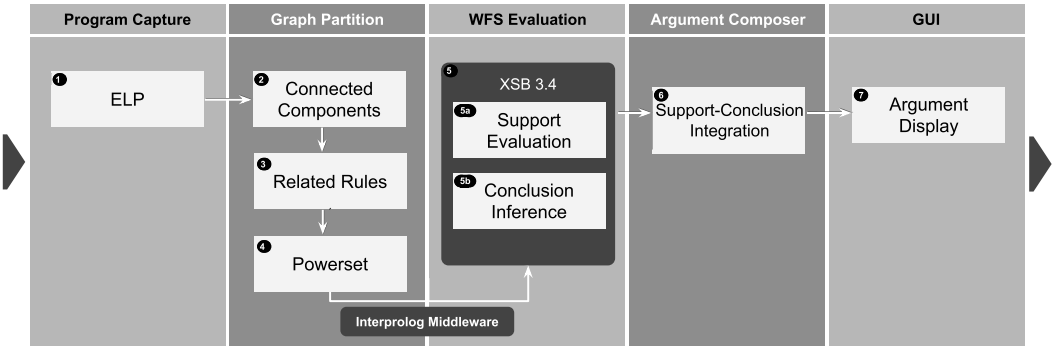

Fig. 4. Implementation workflow of the Argument Engine.

Proposition 11. *Let* P *be an extended logic program,* $AF_P = \langle \mathcal{A}rg_P, At(\mathcal{A}rg_P)\rangle$ *be the resulting argumentation framework from* P *and* SEM_{Arg} *be an argumentation semantics. If* SEM_{Arg} *is a conflict-free semantics, then* AF_P *satisfies direct consistency w.r.t.* SEM_{Arg}.

The last postulate which we are going to consider is indirect consistency. This postulate is defined as follows:

Postulate 4 *(Indirect consistency). Let* P *be an extended logic program,* $AF_P = \langle \mathcal{A}rg_P, At(\mathcal{A}rg_P)\rangle$ *be the resulting argumentation framework from* P *and* SEM_{Arg} *be an argumentation semantics.* Output *is the set of justified conclusions with respect to* $SEM_{Arg}(AF_P) = \{E_1, \ldots, E_n\}$. AF_P *satisfies indirect consistency w.r.t.* SEM_{Arg} *iff:*

1. $Cl_G(\text{Concs}(E_i))$ *is consistent for each* $1 \leqslant i \leqslant n$ *such that* $G = \bigcup_{\langle S,g\rangle \in E_i} S$.
2. $Cl_{G'}(\text{Output})$ *is consistent such that* $\text{Output}' = \bigcap_{i=1\ldots n} E_i$ *and* $G' = \bigcup_{\langle S,g\rangle \in Output'} S$.

Intuitively, Postulate 4 establishes that the closure of the clauses for inferring conclusions must be consistent. By considering that *indirect consistency* is based on the *closure* notion, we find that our argumentation approach, which uses *relevant rules* under a WFS evaluation, fulfills these two conditions. This remark is captured as follows:

Proposition 12. *Let* P *be an extended logic program,* $AF_P = \langle \mathcal{A}rg_P, At(\mathcal{A}rg_P)\rangle$ *be the resulting argumentation framework from* P *and* SEM_{Arg} *be an argumentation semantics. If* SEM_{Arg} *is a conflict-free semantics, then* AF_P *satisfies indirect consistency w.r.t.* SEM_{Arg}.

5. Implementation

So far we have introduced an argumentation approach which takes as input an extended logic program and gives as output a set of arguments and a set of attacks between them. Moreover, we have identified conditions in which the suggested argumentation approach satisfies some basic postulates. These postulates help to judge the quality of the conclusions of the suggested argumentation approach considering Dung's argumentation semantics style.

In this section, an argumentation tool is described which implements the suggested argumentation approach. The implementation of this argumentation engine is based on WFS and Java, using InterProlog [54] as middleware for calling the XSB solver [55], as it is shown in Fig. 4. Our argument engine captures an extended logic program using Prolog code as input. By considering a program as a directed graph, the argument engine performs a graph analysis checking connected atoms. Relevant clauses for a particular atom are produced by obtaining subsets of their connected atoms and generating all their possible subsets. The WFS analysis is performed by using InterProlog function calls which transform XSB procedures into Java objects and conversely. Graph analysis information regarding the evaluated support for a given conclusion is stored in an embedded database. The integration of support and conclusion for building an argument is one of the last steps of the workflow (Fig. 4), storing the sets of generated arguments in the database and finally displaying those arguments in a graphical user interface.

The argument engine exports the built arguments in a JSON format.[9] An implementation exporting arguments in an argument interchange format [56] will be part of the future versions of our argument engine. The full sources of our argumentation engine are available at https://github.com/logic-programming-argumentation/wfs-argument-engine, as well as installation manuals and examples are available also at http://www8.cs.umu.se/~esteban/wfsArgEngine.

6. Discussion and related work

In this section, we are going to identify some related works which are close to the results presented in this article.

[9] JavaScript Object Notation, http://json.org/.

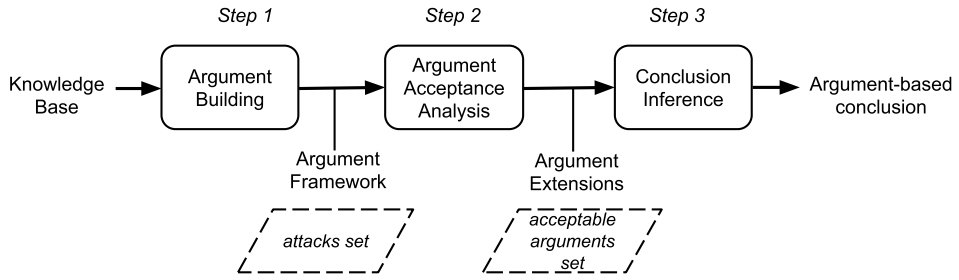

Fig. 5. Meta-interpreter for an argumentation system.

6.1. Management of incomplete information, exceptions and inconsistent knowledge bases with ELP under Answer Sets

Extended logic programs (ELP) [19] capture incomplete information as well as exceptions, using strong negation and negation-as-failure (NAF). Indeed, ELP is not the only approach for capturing defaults and partial information. Some other approaches [21–23] capture incomplete information and exceptions as well. ELP is the knowledge representation language for our approach for two main reasons: semantic expressiveness and the existence of efficient tools for computing ELP semantics, such as XSB [55], and a number of ASP solvers, such as: DLV[10] and SMODELS,[11] among others. Two major semantics for ELP have been defined: (1) answer set semantics [19], an extension of *Stable model semantics*; (2) a version of the Well-Founded Semantics (WFS) [24]. While both coincide in the class of *stratified* programs, they behave quite differently in *non-stratified* programs. In our approach, a correspondence between them is obtained by the fact that every argument built in our approach has a stratified logic program as support (see Proposition 2). A comparative review of different semantics [57] shows that the data complexity of logic programs under WFS is polynomial, allowing us to feasibly evaluate programs.

WFS has shown to be a strong approach to fulfilling all the principles of *well-behaved semantics* [39], such as *Relevance*, among others (Stable semantics introduced in [31] does not satisfy the relevance principle[12]). The authors in [30] show that classical propositional proof systems, propagate conflicts throughout the knowledge bases to unrelated parts (similar to the concept of *contamination* in [32,33]). The principle of *Relevance* allows us to define the notion of epicenter, which is related to the set of relevant clauses that can be part of the support structure of an argument. This notion is important in defining conditions for satisfying closure and consistency. In [51], the authors introduce the notion of a *focal graph* which is defined as a set of relevant clauses to a specific clause in a knowledge base. This concept is also revised, with no specific name, in the analysis of consistency in different approaches [29,30,59].

The notion of *inconsistency* is very important in the study of knowledge-based systems. Inconsistency may be present for different reasons, as was highlighted in [60] for instance:

- The knowledge base includes default rules (i.e., “birds fly”, “penguins are birds”, “penguins do not fly”) and facts (i.e., “Tweety is a bird”) and, later, new information is received (i.e., “Tweety is a penguin”) that contradicts a plausible conclusion that could be previously derived from the knowledge base;
- In model-based diagnosis, where a knowledge base contains a description of the normal behavior of a system. Failure detection occurs when observations conflict with the normal functioning mode of the system and the hypothesis that the components of the system are working well; this leads to diagnose which component(s) fail(s);
- Several consistent knowledge bases pertaining to the same domain but coming from n different information sources are available. For instance, each source is a reliable specialist in some aspect of the concerned domain, but is less reliable in other aspects. A straightforward way of building a global base Σ is to concatenate the knowledge bases Σ_i $(1 \leq i \leq n)$ provided by each source. Even if Σ_i is consistent, it is unlikely that $\Sigma_1 \cup \Sigma_2 \ldots \cup \Sigma_n$ will also be consistent.

The underlying formalisms for knowledge representation in argument-based systems play an important role in the management of these causes of inconsistency. This observation is evident if we consider an argument-based system as a three-steps architecture (Fig. 5), extending the meta-interpreter proposed by Dung in [8]. An important part of argument-based systems relies on computing all acceptable sets of argument extensions (output of Step 2 in Fig. 5). Different ASP approaches rely on the mapping of an argumentation framework into a logic program whose answer sets are in one-to-one correspondence with the sets of arguments suggested by an argumentation semantics of the original framework. A survey of different approaches using ASP for argumentation is introduced in [61].

[10] www.dbai.tuwien.ac.at/proj/dlv/ or www.dlvsystem.com/.
[11] www.tcs.hut.fi/Software/smodels/.
[12] Different extensions of the Stable semantics [31] fulfilling some properties including Relevance, have been proposed in the literature [40,58].

Table 3
Comparative table for argument building approaches.

Project	Theoretical formalism		Implementation	
	Lang.[a]	Formalism[b]	Approach	Availability[c]
TOAST	Unspecified	ASPIC+ is an extension of ASPIC framework	Java implemented + Web client/service	Yes on-line
GORGIAS	LPwNAF	Abductive reasoning used for admissibility analysis	SWI-Prolog based	Yes on-line
CASAPI	Classic LP	Assumption-based argumentation through dispute derivations	Based on an extension of Prolog	Yes on-line
Efstathiou and Hunter	Classic LP	Support trees are built for finding proofs	Direct implementation based on their algorithms	Not available
DeLP client	DeLP	DeLP considers weak and strict rules; construction of argument structures is non-monotonic	Based on an extension of Prolog + Web client	Yes on-line

[a] The expressiveness language for the argument-based formalism.
[b] The theoretical underlying argument-based formalism.
[c] Intended as a Web availability on February 2014.

6.2. Arguments with a stratified program as support

In argumentation literature, abstract argumentation frameworks ([8,9], among others) provide bases for exploring issues of defeasible reasoning, disregarding the structure or nature of the arguments. Our approach follows the line of abstract argumentation frameworks introduced in [8], as it is closer to approaches where the knowledge is coded in the structure of arguments and argumentation semantics is used to determine the acceptability of arguments. In this setting, Table 3 summarizes the key characteristics and remarks of some related approaches. TOAST is an implementation of ASPIC+ [62] (a further development of ASPIC [63]) that defines inference trees formed by applying two kinds of inference rules: strict and defeasible.

Our argument structure resembles *subgraphs* of a graph (the full program), using knowledge epicenters in order to support a conclusion. Efstathiou and Hunter developed an approach for building arguments using support trees to find proofs by contradiction [16]. This work is framed on classical propositional logic, limited in this manner to capturing incomplete information, i.e., exceptions. Prakken and Modgil construct arguments by chaining inference rules into trees [62,64], extending ASPIC with preference information for resolving conflicts between arguments, with the argument structure being crucial in determining how preferences must be applied to attacks.

On the other hand, in [9,65] Bondarenko et al. define a generalized framework for assumption-based reasoning: the Assumption-Based Argumentation (ABA), which is applicable as a generic approach to describing a range of formalizations of default reasoning. Even when ABA is a general-purpose argumentation framework that can be instantiated to support various specialized frameworks, ABA does not have explicit rebuttals and does not impose the restriction that arguments have consistent and minimal supports, as was pointed out by Dung et al. in [66]. Nevertheless, our approach is closely related to this work and, informally speaking, can be seen as an instance of ABA. Indeed, the concept of *assumption* has a direct relationship with negative literals in ELP and in logic programming in general, and can be understood as expressing the lack of evidence about them. Part of our future work is the establishment of a formal relationship between our approach and ABA.

In [67], three dialectic proof procedures for finding a set of assumptions supporting a given belief are introduced. In this approach, an argument is defined as a deduction whose premises are all assumptions; in this manner, the attack relationship between arguments depends entirely on sets of assumptions. Dung et al. define a *backward argument* as a proof tree which can be seen as a top-down method of deduction from the root (conclusion) to the terminal nodes (assumptions). The authors in [67] illustrate how their backward arguments are a generalization of SLD resolution, which is the basis of proof procedures for logic programming. This approach to building top-down proof-tree arguments is also considered in [68], where different approaches of *answer set justifications* are introduced. Furthermore, Apt et al. characterize a deduction model from a logic program in two ways: 1) *backward-chaining*, based on a recursive, "top-down" chaining; and 2) *forward-chaining*, based on an inductive, "bottom-up" chaining [69]. The *backward argument* defined by Dung et al. in [67] follows the Apt' backward-chaining procedure.

In [70], semantic equivalences between argumentation frameworks and logic programs are presented. The argument structure considered in this work, as well as in [71], establishes a tree structure of clauses starting from a *root* (the conclusion), removing those clauses which generate infinite loops.

There is a major difference in the argument notion between approaches building tree-like proof procedures (like [67, 69,70,62] among others) and our approach. The construction of arguments proposed in our approach follows a *semantic interpretation of the argument support*. In this setting, it is worth differentiating between a *proof-oriented* argument construction and a *semantic-oriented* approach. In the first one, as in ABA, *syntactic* methods close to SLD resolution are used. In these approaches there are no semantic restrictions in the argument support, allowing arguments, for instance, of the form:

$\langle\{f \leftarrow not\ f\},\ f\rangle$ as in [70]. One can see that $\langle\{f \leftarrow not\ f\},\ f\rangle$ is a self-attacked argument. Our approach avoids this type of arguments through a semantic interpretation of the support. It is worth mentioning that WFS is empty and Stable semantics is undefined with the program $f \leftarrow not\ f$. In general, our approach avoids self-attacked arguments (Proposition 7).

In the literature, different transformations from logic programs into argumentation frameworks have been suggested [8,70]. For instance, let us observe that in Section 4.3 in [8], two transformation are introduced. Given a logic program P, let us denote by AF_{napif} and AF_{naff} the resulting argumentation frameworks using Dung transformations from Section 4.3 in [8], and let us denote by AF_{Wu} the resulting argumentation framework considering [70]. We can observe that given a logic program, the resulting argumentation frameworks AF_{napif}, AF_{naff} and AF_{Wu} are different to our resulting argumentation framework AF_P from a *cardinality* standpoint (number of arguments and attacks) and from the structure of arguments. Nevertheless, we can also observe that for some classes of logic programs, as the stratified ones, the argumentation-inference from AF_{napif}, AF_{naff}, AF_{Wu} and AF_P coincide. However, this observation is not for any program. For instance, let us consider the program $P = \{a \leftarrow not\ a;\ b \leftarrow not\ a\}$. Let SEM_{ground} denotes the grounded semantics. It is easy to see that $SEM_{ground}(AF_{napif}) = \{\{\}\}$, $SEM_{ground}(AF_{naff}) = \{\{\}\}$, $SEM_{ground}(AF_{Wu}) = \{\{\}\}$ and $SEM_{ground}(AF_P) = \{\{b\}\}$. We consider that the identification of conditions in which, AF_{napif}, AF_{naff}, AF_{Wu} and AF_P coincide in their argumentation inference, can impact in the implementation of real argumentation systems. Hence, this research issue is part of our future work.

6.3. Rationality postulates for argument-based systems based on ELP

In [17], an approach based on Defeasible Logic Programming (DeLP) is presented. DeLP deals with incomplete and contradictory information generating dialectical trees associated with a *warrant* procedure. According to Caminada et al., this work violates a set of rationality postulates introduced in [29]. Indeed, Caminada et al. state that some argumentation formalisms like [72,17,73] infer inconsistent conclusions.

We introduced a set of postulates for argumentation-based systems under extended logic programs, inspired by Caminada's work. In [29], the rationality postulates are intended for ASPIC-like systems, but not for other argument-based systems such as those under logic programming or assumption-based argumentation. We can argue that logic programs with negation as failure can be regarded as *defeasible theories* which are the ones considered in [29]. However, since there are different semantics for capturing negation as failure, one can define different constructions of arguments and different definitions of attacks depending on a given logic programming semantics. In this setting, we have aimed to follow Answer Set Programming roots for capturing negation as failure in an argumentation-based setting.

A set of conditions referred to as the *self-contradiction axiom* that guarantees the consistency property in both ASPIC-like and ABA systems is introduced in [30]. Dung et al. show that ABA systems satisfy the *ab-self-contradiction axiom*.[13] Moreover, in [30] it is proven that, for *definite* logic programs and for *normal* logic programs, the ab-self-contradiction axiom holds trivially. One of the main differences between the approach of Dung and Tang and ours, lies in the treatment of negative literals which are transformed into new atoms, in Dung and Tang approach, in order to capture normal programs as Assumption-Based Argumentation frameworks. Our definitions of closure and consistency follow the management of negation as failure as it is done in Answer Set Programming, based on the Gelfond–Lifschitz reduction. Nevertheless, we can observe that concepts as self-contradiction axiom can be defined in our terms. Given that this notion leads to important concepts as *compactness*, we will study the definition of self-contradiction axiom and their implications, in our approach, will be part of our future work.

7. Conclusions

In this paper, we introduce an approach to building arguments, taking as input an extended logic program and obtaining as result a set of *consistent* and *rational* arguments. Three major contributions are introduced in this paper:

- We establish the concept of argument by considering a semantic interpretation of the argument supports. This notion allows us to unequivocally identify arguments with stratified programs as support, even when the *input* for the argument engine is a non-stratified program. We have observed that our notation of argument can be based on any logic programming semantics which coincides with stable model semantics in the class of stratified logic programs.
- We introduce a set of rationality postulates for argument-based systems under extended logic programs. These postulates *judge* the consistency of the underlying formalisms for building arguments. These postulates are formulated in terms of argument conclusions, considering at the same time a skeptical output of the system. We introduce these concepts redefining quality criteria introduced by [29] in terms of normal logic programs and a definition of closure of a set of clauses that consider the well-known Gelfond–Lifschitz reduction.
- By taking advantage of the computational properties of WFS, we develop an implementation which, to our knowledge, is the first argumentation engine for extended logic programs. Therefore, our engine can serve as a starting point for other argument-based systems in a wide application spectrum, such as decision-making frameworks and context-based argument reasoning.

[13] *ab-* is a prefix implying that the axiom is applicable only in assumption-based systems.

As final remarks, we note a significant property of our argument approach, establishing that the support of all the arguments built according to our approach has a unique stable model (Corollary 2). This result allows us to establish a general process for building arguments by considering different logic programming semantics which coincide with Stable semantics in the class of stratified programs. From a practical and experimental perspective, our approach opens up a number of opportunities for using well-known ASP tools to build arguments.

In our approach, the *relevance* property satisfied by WFS is considered a fundamental feature. This notion allows us to centralize a set of knowledge for creating/supporting arguments in a so-called knowledge *epicenter*, minimizing the set space for establishing *candidate clauses*. This notion is relevant from theoretical as well as experimental points of view. The principle of relevance give us a "partition method" of a knowledge base without losing consistency under WFS evaluation.

On the other hand, there are a number of specific concerns that we want to further explore between proof procedures approaches, such as ABA and our approach, i.e., the overall computational cost of generating tree-like arguments ([67], among others) compared with our approach. This is a further analysis between *syntactic* (proof procedures) and *semantic* argumentation building (our approach).

Part of our future work is also to apply our argumentation engine in a real scenario application, i.e., sensor-based systems where inconsistency of observations of the world are captured by ELP. Furthermore, a *quality* comparison of our argument builder with other approaches is also part of our future work.

Acknowledgements

We are grateful to anonymous referees for their useful comments. This research has been partially supported by VINNOVA (The Swedish Governmental Agency for Innovation Systems, Project number 2008-03438) and the Swedish Brain Power Program.

Appendix A. Proofs

In this appendix, the proofs of the propositions are presented.

Proposition 1. *Let P be an extended logic program. If $\langle S, g\rangle \in \mathcal{A}rg_P$, then S is a stratified program.*

Proof. Let us start with the following two observations:

1. Since *WFS* satisfies *relevance* [39], then $S \subseteq \mathsf{rel\_rules}(P, g)$. Hence, if S is decomposable in $S_1 \cup \cdots \cup S_n$ such that S_1 is the lower rank, g appears in the head of a clause in S_n.
2. Since S is minimal and $WFS(S) = \langle T, F\rangle$ such that $g \in T$, then S is acyclic with respect to negative literals.

The proposition follows the previous two observations which basically check the conditions of a stratified logic program. □

Proposition 2. *Let P be an extended logic program, $Arg_{WFS}(P)$ be the set of arguments from P according to Definition 9 and $Arg_{STABLE}(P)$ be the set of arguments from P according to Definition 11. The following condition holds:*

$$Arg_{WFS}(P) = Arg_{STABLE}(P)$$

Proof. We start introducing the following observation:

1. If P is a stratified logic program then $WFS(P) = \langle T, F\rangle$ and $STABLE(P) = \{M\}$ such that $T = M$ and $F = \mathcal{L}_P \setminus M$.

Now let us present the prove:

=> Let us proof that if $\langle S, c\rangle \in Arg_{WFS}(P)$ then $\langle S, c\rangle \in Arg_{STABLE}(P)$. If $\langle S, c\rangle \in Arg_{WFS}(P)$, then, by Proposition 1, S is a stratified logic program. Moreover, by Definition 9, S is minimal with respect to set inclusion and $\nexists\, g \in \mathcal{L}_P$ such that $\{g, \neg g\} \subseteq T$ and $WFS(S) = \langle T, F\rangle$. Therefore, by Observation 1, $c \in M$ such that $M \in STABLE(P)$. It is clear that $\langle S, c\rangle$ satisfies the conditions of Definition 11. Therefore, $\langle S, c\rangle \in Arg_{STABLE}(P)$.

<= Let us prove that if $\langle S, c\rangle \in Arg_{STABLE}(P)$ then $\langle S, c\rangle \in Arg_{WFS}(P)$. If $\langle S, c\rangle \in Arg_{STABLE}(P)$, then by definition, S is a stratified and minimal with respect to set inclusion. Moreover $\nexists\, g \in \mathcal{L}_P$ such that $\{g, \neg g\} \subseteq M$ and $M \in STABLE(S)$. Hence, the proof is straightforward by Observation 1. □

Proposition 3. *Let P and G be extended logic programs Then, the following condition holds: $\mathcal{A}rg_P \subseteq \mathcal{A}rg_{P \cup G}$.*

Proof. There are two cases to verify:

1. If $\mathcal{L}_P \cap \mathcal{L}_G = \emptyset$, the proof is straightforward since $\mathcal{A}rg_{P\cup G}$ will be $\mathcal{A}rg_P \cup \mathcal{A}rg_G$
2. If $\mathcal{L}_P \cap \mathcal{L}_G \neq \emptyset$, let $I = \mathcal{L}_P \cap \mathcal{L}_G$ and $g \in I$. Since $\text{rel\_rules}(P \cup G, g) = \text{rel\_rules}(P, g) \cup \text{rel\_rules}(G, g)$, we can observe that if S is a minimal subset of $\text{rel\_rules}(P, g)$ such that $WFS(S) = \langle T, F\rangle$ and $g \in T$, then S is a minimal subset of $\text{rel\_rules}(P \cup G, g)$ such that $WFS(S) = \langle T, F\rangle$ and $g \in T$. Hence, if $\langle S, g\rangle \in \mathcal{A}rg_P$, then $\langle S, g\rangle \in \mathcal{A}rg_{P\cup G}$. □

Proposition 4. *Let $A = \langle S, g\rangle$ be an argument. If $a \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^- \in S$ then $a \notin \mathcal{B}^+$ and $\mathcal{B}^+ \cap \mathcal{B}^- = \emptyset$.*

Proof. Let us start observing:

1. Since S is minimal, if $a \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^- \in S$, then $\{a\} \cup \mathcal{B}^+ \subseteq T$ and $\mathcal{B}^- \subseteq F$ such that $WFS(S) = \langle T, F\rangle$.

$a \notin \mathcal{B}^+$: Let us suppose that $a \in \mathcal{B}^+$. By Lemma 2.1, WFS is close with respect to the Loop transformation rule. If S_2 is the resulting program after applying the loop transformation rule to S, $a \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^- \notin S_2$, then $WFS(S_2) = \langle T_2, F_2 \cup \{a\}\rangle$. This is a contradiction because $WFS(S_2) = WFS(S_1)$ and according to Observation 1, $a \in T$ such that $WFS(S) = \langle T, F\rangle$.

$\mathcal{B}^+ \cap \mathcal{B}^- = \emptyset$: The proof is direct by Observation 1 since $T \cap F = \emptyset$. □

Proposition 5. *Let A, B and C be arguments such that B is a sub-argument of A. The following conditions hold:*

a) *If B attacks C then A attacks C.*
b) *If C attacks B then C attacks A.*

Proof. Let us start introducing the following observation:

1. Let $A = \langle S_A, g_B\rangle$ and $B = \langle S_B, g_B\rangle$ such that B is a sub-argument of A. If $WFS(S_A) = \langle T_A, F_A\rangle$ and $B = \langle T_B, F_B\rangle$, then $T_B \subseteq T_A$ and $F_B \subseteq F_A$.

Considering Observation 1, the proof is direct by the fact that the definition of attacks are based on set-contains. □

Proposition 6. *Let P and G be extended logic programs Then, the following condition holds:*

1. $|At(\mathcal{A}rg_P)| \leqslant |At(\mathcal{A}rg_{P\cup G})|$

Proof. The proof is straightforward, by both the fact that the number of arguments does not decrease whenever an extended logic program is extended with new clauses (Proposition 3) and the definition of attacks which is based on the well-founded model of the support of each argument. □

Proposition 7. *Let P be an extended logic program and $AF_P = \langle \mathcal{A}rg_P, At(\mathcal{A}rg_P)\rangle$ be the resulting argumentation framework w.r.t. P. If $(A, B) \in At(\mathcal{A}rg_P)$, then $A \neq B$.*

Proof. The proof is by contradiction. Let us suppose that there is an argument $A = \langle S, c\rangle \in \mathcal{A}rg_P$ and $(A, A) \in At(\mathcal{A}rg_P)$. According to Definition 13, there two cases to verify:

1. If $(A, A) \in At(\mathcal{A}rg_P)$ by Condition 1 of Definition 13, then $\{c, \neg c\} \in T$ such that $WFS(S) = \langle T, F\rangle$. This is a contradiction because T is consistent.
2. If $(A, A) \in At(\mathcal{A}rg_P)$ by Condition 2 of Definition 13, then $c \in T$ and $c \in F$ such that $WFS(S) = \langle T, F\rangle$. This is a contradiction because $T \cap F = \emptyset$. □

Proposition 8. *Let* P *be an extended logic program, $AF_P = \langle \mathcal{A}rg_P, At(\mathcal{A}rg_P)\rangle$ be the resulting argumentation framework from* P *and SEM_{Arg} be an argumentation semantics. If SEM_{Arg} is a Dung's standard semantics, then AF_P satisfies sub-argument closure w.r.t. SEM_{Arg}.*

Proof. Since grounded, preferred and stable extensions are also complete extensions, the proof will be only concerned on complete extensions.

Let SEM_C denotes the complete semantics, $E \in \text{SEM}_C(AF_P)$, $A, B \in \mathcal{A}rg_P$ and B be a sub-argument of A. Then if $A \in E$, we have to proof that $B \in E$.

The proof will be by contradiction. Hence, let us suppose that $B \notin E$. Hence there are two cases to see:

1. $E \cup \{B\}$ is not conflict-free: If $E \cup \{B\}$ is not conflict-free, then $\exists\, C \in E$ such that either C attacks B or B attacks C. Suppose C attacks B, then, by Proposition 5, C attacks A. This is a contradiction because $A \in E$. Suppose B attacks C, then, by Proposition 5, A attacks C. This is contradiction again because $A \in E$.
2. If $C \in Arg_P$ such that C attacks B, then E does not attack C: If C attacks B, then, by Proposition 5, C attacks A. Since $A \in E$, then E attacks A, this is a contradiction. □

Proposition 9. *Let E be a conflict-free set of arguments and $G = \bigcup_{\langle S,g\rangle \in E} S$. The following conditions hold:*

1. *G is a stratified logic program.*
2. *$WFS(G) = \bigsqcup_{\langle S,g\rangle \in E} WFS(S)$ such that $\langle T_1, F_1\rangle \sqcup \langle T_2, F_2\rangle = \langle T_1 \cup T_2, F_1 \cup F_2\rangle$*

Proof. Let us start with the following observation:

1. Let $\langle S, a\rangle$ be an argument. If $a \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^- \in S$, then $a \leftarrow \top \in norm_{CS_0}(S)$.

Since $\cup$ and $\sqcup$ are binary operators, let us consider only two arguments: $A_1 = \langle S_1, a_1\rangle$ and $A_2 = \langle S_2, a_2\rangle$ such that $\{A_1, A_2\}$ is a conflict-free set and $S = S_1 \cup S_2$. Let us prove each case of the proposition considering these two arguments:

1. We have to prove that S is a stratified logic program. Let us suppose that S is not a stratified program. Then r_a and r_b appear in S such that $r_a = a \leftarrow \mathcal{B}_a^+ \wedge not\ (\mathcal{B}_a^- \cup \{b\})$ and $r_b = b \leftarrow \mathcal{B}_b^+ \wedge not\ (\mathcal{B}_b^- \cup \{a\})$. Then there two cases to see:
   (a) r_a and r_b appear together in either S_1 or S_2. This is a contraction because both S_1 and S_2 are stratified logic programs.
   (b) r_a appears in S_1 and r_b appears in S_2. This is a contradiction because $\{A_1, A_2\}$ is a conflict-free set.
2. We have to prove that $WFS(S) = WFS(S_1) \sqcup WFS(S_2)$. By Lemma 2.1, it is equivalent to show that $norm_{CS_0}(S) = norm_{CS_0}(S_1) \cup norm_{CS_0}(S_2)$. Let us suppose that $norm_{CS_0}(S) \neq norm_{CS_0}(S_1) \cup norm_{CS_0}(S_2)$. There are two cases to see:
   (a) $\exists\, a \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^- \in norm_{CS_0}(S)$ and $a \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^- \notin norm_{CS_0}(S_1) \cup norm_{CS_0}(S_2)$. Since S is a stratified logic program: $a \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^- = a \leftarrow \top$. Then $a \leftarrow \top \notin norm_{CS_0}(S_1) \cup norm_{CS_0}(S_2)$. This is a contradiction by Observation 1 and the fact that $S = S_1 \cup S_2$.
   (b) $\exists\, a \leftarrow \top \in norm_{CS_0}(S_1) \cup norm_{CS_0}(S_2)$ and $a \leftarrow \top \notin norm_{CS_0}(S)$. Suppose that $a \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^- \in norm_{CS_0}(S)$ such that $\mathcal{B}^+ \cup \mathcal{B}^- \neq \emptyset$. Then $\exists\, b \in \mathcal{B}^+ \cup \mathcal{B}^-$ such that $b \leftarrow \mathcal{B}_a^+ \wedge not\ (\mathcal{B}_a \cup \{n\})^-$ and $n \leftarrow \mathcal{B}_b^+ \wedge not\ (\mathcal{B}_b \cup \{b\})^-$ appears in $norm_{CS_0}(S)$. This is a contradiction because S is a stratified logic program. Suppose that $a \notin \text{HEAD}(norm_{CS_0}(S))$. Then $a \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^-$ was removed from S by either Loop, RED$^-$ or Failure. Loop cannot be applied to S because S has not cycles with positive literals. RED$^-$ cannot be applied because $\{A_1, A_2\}$ is conflict-free set. If failure was applied then $\exists\, b \in \mathcal{B}^+$ such that $b \notin HEAD(norm_{CS_0}(S))$. This is a contradiction because $a \leftarrow \top \in norm_{CS_0}(S_1) \cup norm_{CS_0}(S_2)$ and Observation 1. □

Proposition 10. *Let* P *be an extended logic program, $AF_P = \langle Arg_P, At(Arg_P)\rangle$ be the resulting argumentation framework from* P *and SEM_{Arg} be an argumentation semantics. If SEM_{Arg} is a conflict-free semantics, then AF_P satisfies closure w.r.t. SEM_{Arg}.*

Proof. We start introducing the following notation:

$G_E = \bigcup_{\langle S,g\rangle \in E} S$ such that $E \in \text{SEM}_{Arg}(AF_{Arg})$.
$G' = \bigcup_{\langle S,g\rangle \in Output'} S$ such that $\text{Output}' = \bigcap_{E \in \text{SEM}_{Arg}(AF_{Arg})} E$.

Now, let us introduce the following observations:

1. If P is a stratified logic program, $CL_P(S) = S$ iff S is a stable model of P.
2. By Proposition 9, if $E \in \text{SEM}_{Arg}(AF_{Arg})$, $Concs(E) = T$ such that $WFS(G_E) = \langle T, F\rangle$.
3. By Proposition 9, M is a stable model of G_E iff $WFS(G_E) = \langle M, \mathcal{L}_p \setminus M\rangle$.

1) By Observation 1 and Proposition 9, the proof is reduced to show that if $E \in \text{SEM}_{Arg}(AF_{Arg})$ then $Concs(E)$ is a stable model of G_E. Let us suppose that $Concs(E)$ is not a stable model of G_E. Since G_E is a stratified logic program (Proposition 9), G_E has a unique stable model which is denoted by M. Then $M \neq Concs(E)$. If $M \neq Concs(E)$, there are two cases to see:
   1. $\exists\, a \in M$ such that $a \notin Concs(E)$. If $a \in M$ and $a \notin Concs(E)$, then $a \in F$ such that $WFS(G_E) = \langle T, F\rangle$. This a contradiction of Observation 3.
   2. $\exists\, a \in Concs(E)$ such that $a \notin M$. If $a \in Concs(E)$ and $a \notin M$. Then $a \in \mathcal{L}_p \setminus M$. Hence then $a \in F$ such that $WFS(G_E) = \langle T, F\rangle$. It is a contradiction of Observation 2.

2) Since G' is a subset of each G_E induced by $E \in SEM_{Arg}(AF_{Srg})$, the proof is direct by 1). □

Proposition 11. *Let* P *be an extended logic program,* $AF_P = \langle \mathcal{A}rg_P, At(\mathcal{A}rg_P)\rangle$ *be the resulting argumentation framework from* P *and* SEM_{Arg} *be an argumentation semantics. If* SEM_{Arg} *is a conflict-free semantics, then* AF_P *satisfies direct consistency w.r.t.* SEM_{Arg}.

Proof. Let us start introducing the following notation:

$G_E = \bigcup_{\langle S,g\rangle \in E} S$ such that $E \in \text{SEM}_{Arg}(AF_{Arg})$.
$G' = \bigcup_{\langle S,g\rangle \in Output'} S$ such that $\text{Output}' = \bigcap_{E \in \text{SEM}_{Arg}(AF_{Arg})} E$.

The proposition holds by Proposition 9 and the fact that both $WFS(G_E)$ and $WFS(G')$ are consistent models. □

Proposition 12. *Let* P *be an extended logic program,* $AF_P = \langle \mathcal{A}rg_P, At(\mathcal{A}rg_P)\rangle$ *be the resulting argumentation framework from* P *and* SEM_{Arg} *be an argumentation semantics. If* SEM_{Arg} *is a conflict-free semantics, then* AF_P *satisfies indirect consistency w.r.t.* SEM_{Arg}.

Proof. The proposition holds by Proposition 9 and once again by the fact that both $WFS(G_E)$ and $WFS(G')$ are consistent models. □

References

[1] J. McCarthy, P. Hayes, Some Philosophical Problems from the Standpoint of Artificial Intelligence, Stanford University USA, 1968.
[2] J. McCarthy, Programs with Common Sense, Defense Technical Information Center, 1963.
[3] C. Strasser, G.A. Antonelli, Non-monotonic logic, in: E.N. Zalta (Ed.), The Stanford Encyclopedia of Philosophy, winter 2014 edition, 2014.
[4] J. McCarthy, Circumscription – a form of non-monotonic reasoning, Artif. Intell. 13 (1) (1980) 27–39.
[5] D.V. McDermott, Nonmonotonic logic II: nonmonotonic modal theories, J. ACM 29 (1) (1982) 33–57, http://dx.doi.org/10.1145/322290.322293.
[6] R.C. Moore, Semantical considerations on nonmonotonic logic, Artif. Intell. 25 (1) (1985) 75–94.
[7] R. Reiter, A logic for default reasoning, Artif. Intell. 13 (1) (1980) 81–132.
[8] P.M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and *n*-person games, Artif. Intell. 77 (2) (1995) 321–357.
[9] A. Bondarenko, P.M. Dung, R.A. Kowalski, F. Toni, An abstract, argumentation-theoretic approach to default reasoning, Artif. Intell. 93 (1) (1997) 63–101.
[10] L. Amgoud, C. Cayrol, A reasoning model based on the production of acceptable arguments, Ann. Math. Artif. Intell. 34 (1–3) (2002) 197–215.
[11] S. Modgil, M. Caminada, Proof theories and algorithms for abstract argumentation frameworks, in: Argumentation in Artificial Intelligence, Springer, 2009, pp. 105–129.
[12] L. Amgoud, P. Besnard, Bridging the gap between abstract argumentation systems and logic, in: Scalable Uncertainty Management, Springer, 2009, pp. 12–27.
[13] M. Snaith, C. Reed, Toast: online ASPIC+ implementation, in: Fourth International Conference on Computational Models of Argument, COMMA 2012, IOS Press, 2012, pp. 509–510.
[14] N. Demetriou, A.C. Kakas, Argumentation with abduction, in: Proceedings of the Fourth Panhellenic Symposium on Logic, vol. 26, 2003, pp. 177–184.
[15] D. Gartner, F. Toni, Casapi: a system for credulous and sceptical argumentation, in: Proc. of Argumentation and Non-Monotonic Reasoning (ArgNMR), 2007, pp. 80–95.
[16] V. Efstathiou, A. Hunter, Algorithms for generating arguments and counterarguments in propositional logic, Int. J. Approx. Reason. 52 (6) (2011) 672–704.
[17] A.J. García, G.R. Simari, Defeasible logic programming: an argumentative approach, Theory Pract. Log. Program. 4 (1–2) (2004) 95–138.
[18] S. Benferhat, D. Dubois, H. Prade, Reasoning in inconsistent stratified knowledge bases, in: 26th International Symposium on Multiple-Valued Logic, 1996. Proceedings, IEEE, 1996, pp. 184–189.
[19] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (3–4) (1991) 365–385.
[20] C. Baral, M. Gelfond, Representing concurrent actions in extended logic programming, in: The 13th International Joint Conference on Artificial Intelligence (IJCAI), vol. 2, Morgan Kaufmann Publishers Inc., 1993, pp. 866–873.
[21] J. Lobo, J. Minker, A. Rajasekar, Foundations of Disjunctive Logic Programming, The MIT Press, 1992.
[22] A.C. Kakas, R.A. Kowalski, F. Toni, Abductive logic programming, J. Log. Comput. 2 (6) (1992) 719–770.
[23] D. Nute, Defeasible logic, in: Web Knowledge Management and Decision Support, Springer, 2003, pp. 151–169.
[24] A. Van Gelder, K.A. Ross, J.S. Schlipf, The well-founded semantics for general logic programs, J. ACM 38 (3) (1991) 619–649.
[25] V. Lifschitz, What is answer set programming? in: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13–17, 2008, 2008, pp. 1594–1597.
[26] G. Brewka, Well-founded semantics for extended logic programs with dynamic preferences, J. Artif. Intell. Res. 4 (1) (1996) 19–36.
[27] C. Baral, Knowledge Representation, Reasoning and Declarative Problem Solving, Cambridge University Press, 2003.
[28] M. Fitting, Fixpoint semantics for logic programming a survey, Theor. Comput. Sci. 278 (1) (2002) 25–51.
[29] M. Caminada, L. Amgoud, On the evaluation of argumentation formalisms, Artif. Intell. 171 (5) (2007) 286–310.
[30] P.M. Dung, P.M. Thang, Closure and consistency in logic-associated argumentation, J. Artif. Intell. Res. 49 (2014) 79–109.
[31] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R. Kowalski, K. Bowen (Eds.), 5th Conference on Logic Programming, MIT Press, 1988, pp. 1070–1080.
[32] M.W. Caminada, W.A. Carnielli, P.E. Dunne, Semi-stable semantics, J. Log. Comput. (2011) 1–48.
[33] Y. Wu, Between argument and conclusion, argument-based approaches to discussion, inference and uncertainty, PhD thesis report, Universite du Luxembourg, 2012.
[34] K.L. Clark, Negation as failure, in: Logic and Data Bases, Springer, 1978, pp. 293–322.
[35] M. Fitting, A Kripke–Kleene semantics for logic programs, J. Log. Program. 2 (4) (1985) 295–312, http://dx.doi.org/10.1016/S0743-1066(85)80005-4.
[36] T.C. Przymusinski, Every logic program has a natural stratification and an iterated least fixed point model, in: Proceedings of the Eighth ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems, ACM, 1989, pp. 11–21.
[37] T.C. Przymusinski, On the declarative and procedural semantics of logic programs, J. Autom. Reason. 5 (2) (1989) 167–205.
[38] M. Osorio, J.A. Navarro, J.R. Arrazola, V. Borja, Logics with Common Weak Completions, vol. 16, 2006, pp. 867–890.
[39] J. Dix, A classification theory of semantics of normal logic programs: II. Weak properties, Fundam. Inform. 22 (3) (1995) 257–288.

[40] J. Dix, M. Müller, Partial evaluation and relevance for approximations of the stable semantics, in: Methodologies for Intelligent Systems, Springer, 1994, pp. 511–520.
[41] T.C. Przymusinski, Every Logic Program Has a Natural Stratification and an Iterated Least Fixed Point Model (Extended Abstract), ACM Press, 1989, pp. 11–21.
[42] A. Van Gelder, Negation as failure using tight derivations for general logic programs, J. Log. Program. 6 (1–2) (1989) 109–133.
[43] V. Lifschitz, A. Razborov, Why are there so many loop formulas? ACM Trans. Comput. Log. 7 (2) (2006) 261–268.
[44] J. Dix, A classification theory of semantics of normal logic programs: I. Strong properties, Fundam. Inform. 22 (3) (1995) 227–255.
[45] J. Dix, M. Osorio, C. Zepeda, A general theory of confluent rewriting systems for logic programming and its applications, Ann. Pure Appl. Logic 108 (1–3) (2001) 153–188.
[46] S. Brass, J. Dix, B. Freitag, U. Zukowski, Transformation-based bottom-up computation of the well-founded model, Theory Pract. Log. Program. 1 (5) (2001) 497–538.
[47] S. Brass, U. Zukowski, B. Freitag, Transformation-based bottom-up computation of the well-founded model, in: Non-Monotonic Extensions of Logic Programming, Springer, 1997, pp. 171–201.
[48] A.V. Gelder, K.A. Ross, J.S. Schlipf, The well-founded semantics for general logic programs, J. ACM 38 (3) (1991) 620–650.
[49] J.C. Nieves, M. Osorio, U. Cortés, An overview of argumentation semantics, Comput. Sist. 12 (1) (2008) 65–88.
[50] P. Baroni, M. Caminada, M. Giacomin, An introduction to argumentation semantics, Knowl. Eng. Rev. 26 (4) (2011) 365–410.
[51] V. Efstathiou, A. Hunter, Algorithms for effective argumentation in classical propositional logic: a connection graph approach, in: Foundations of Information and Knowledge Systems, Springer, 2008, pp. 272–290.
[52] K. Kunen, Negation in logic programming, J. Log. Program. 4 (4) (1987) 289–308.
[53] S. Modgil, H. Prakken, The ASPIC$^+$ framework for structured argumentation: a tutorial, Argument Comput. 5 (1) (2014) 31–62, http://dx.doi.org/10.1080/19462166.2013.869766.
[54] M. Calejo, Interprolog: towards a declarative embedding of logic programming in Java, in: Logics in Artificial Intelligence, Springer, 2004, pp. 714–717.
[55] T. Swift, D.S. Warren, Xsb: extending prolog with tabled logic programming, CoRR, arXiv:1012.5123.
[56] C.I. Chesñevar, J. McGinnis, S. Modgil, I. Rahwan, C. Reed, G.R. Simari, M. South, G. Vreeswijk, S. Willmott, Towards an argument interchange format, Knowl. Eng. Rev. 21 (4) (2006) 293–316, http://dx.doi.org/10.1017/S0269888906001044.
[57] C. Baral, M. Gelfond, Logic programming and knowledge representation, J. Log. Program. 19 (1994) 73–148.
[58] J. Dix, Semantics of logic programs: their intuitions and formal properties. An overview, in: Logic, Action, and Information, 1996, pp. 241–327.
[59] M. Caminada, Semi-stable semantics, in: 1st International Conference on Computational Models of Argument (COMMA), vol. 144, IOS Press, 2006, pp. 121–130.
[60] S. Benferhat, D. Dubois, H. Prade, Some syntactic approaches to the handling of inconsistent knowledge bases: a comparative study part 1: the flat case, Stud. Log. 58 (1) (1997) 17–45, http://dx.doi.org/10.1023/A:1004987830832.
[61] F. Toni, M. Sergot, Argumentation and answer set programming, in: Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning, Springer, 2011, pp. 164–180.
[62] H. Prakken, An abstract framework for argumentation with structured arguments, Argument Comput. 1 (2) (2010) 93–124.
[63] L. Amgoud, L. Bodenstaff, M. Caminada, P. McBurney, S. Parsons, H. Prakken, J. van Veenen, G. Vreeswijk, Final review and report on formal argumentation system. Deliverable d2. 6, Tech. rep., ASPIC IST-FP6-002307, 2006.
[64] S. Modgil, H. Prakken, A general account of argumentation with preferences, Artif. Intell. 195 (2013) 361–397.
[65] A. Bondarenko, F. Toni, R.A. Kowalski, An assumption-based framework for non-monotonic reasoning, in: LPNMR, vol. 93, 1993, pp. 171–189.
[66] P.M. Dung, R.A. Kowalski, F. Toni, Assumption-based argumentation, in: Argumentation in Artificial Intelligence, Springer, 2009, pp. 199–218.
[67] P.M. Dung, R.A. Kowalski, F. Toni, Dialectic proof procedures for assumption-based, admissible argumentation, Artif. Intell. 170 (2) (2006) 114–159.
[68] C. Schulz, F. Toni, Justifying answer sets using argumentation, Theory Pract. Log. Program. (2015), http://dx.doi.org/10.1017/S1471068414000702.
[69] K.R. Apt, H.A. Blair, A. Walker, Towards a Theory of Declarative Knowledge, IBM TJ Watson Research Center, 1986.
[70] Y. Wu, M. Caminada, D.M. Gabbay, Complete extensions in argumentation coincide with 3-valued stable models in logic programming, Stud. Log. 93 (2–3) (2009) 383–403.
[71] M. Caminada, S. Sá, J. Alcântara, On the equivalence between logic programming semantics and argumentation semantics, in: Symbolic and Quantitative Approaches to Reasoning with Uncertainty, Springer, 2013, pp. 97–108.
[72] H. Prakken, G. Sartor, Argument-based extended logic programming with defeasible priorities, J. Appl. Non-Class. Log. 7 (1–2) (1997) 25–75.
[73] G. Governatori, M.J. Maher, G. Antoniou, D. Billington, Argumentation semantics for defeasible logic, J. Log. Comput. 14 (5) (2004) 675–702.