Attack Semantics for Abstract Argumentation

Serena Villata
INRIA
Sophia Antipolis, France
serena.villata@inria.fr

Guido Boella
Dipartimento di Informatica
Università di Torino, Italia
guido@di.unito.it

Leendert van der Torre
CSC
University of Luxembourg
leendert@vandertorre.com

Abstract

In this paper we conceptualize abstract argumentation in terms of successful and unsuccessful attacks, such that arguments are accepted when there are no successful attacks on them. We characterize the relation between attack semantics and Dung's approach, and we define an SCC recursive algorithm for attack semantics using attack labelings.

1 Introduction

Dung [1995] conceptualizes abstract argumentation in terms of attacks among arguments, together with the distinction among acceptable and unacceptable arguments. In this paper we raise the following question.

Research question. How to conceptualize argumentation in terms of successful and unsuccessful attacks, such that arguments are accepted if and only if there are no successful attacks on them?

For example, if "a suspect is innocent since he was on holidays" is attacked by "a witness saw him at the crime scene" and the suspect is not found guilty, then we say that the argument of the defense was accepted, because attacks on the innocence of the suspect were not successful.

We address several challenges. First, attacks are not only successful if they come from accepted arguments, since when two arguments attack each other, then it may be that neither argument is accepted, and thus that both attacks are successful. Therefore we may have that an argument is not accepted but its attacks are successful, which corresponds to the notion of undecided arguments in argument labelings [Caminada, 2006]. The following example illustrates the basic concepts and their relations.

Example 1 *Consider the argumentation framework with arguments* $\{a, b, c\}$*, attacks* $a \rightarrow a, a \rightarrow b, b \rightarrow c, c \rightarrow b$ *and extension* $\{c\}$*, visualized in Figure 1. Argument semantics says that* a *and* b *are rejected and* c *is accepted, and labeling semantics says that* a *is undecided,* b *is out and* c *is in. Attack semantics says that* $a \rightarrow a$*,* $a \rightarrow b$*, and* $c \rightarrow b$ *are successful, and the other attack is unsuccessful. Argument* c *is accepted since there are no successful attacks on it. The attacks by the in and undecided arguments are successful.*

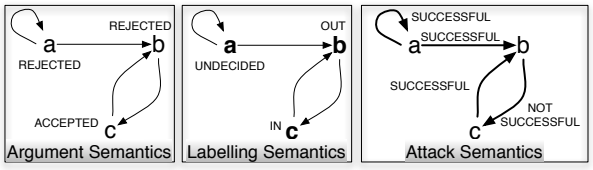

Figure 1: Successful attacks by *in* and *undecided* arguments.

Second, in attack semantics an argument is accepted when there are no successful attacks on it, but this is too weak to characterize admissibility semantics. Consider an argumentation framework with a single argument and an empty attack relation. There are two admissible extensions, since the single argument can be accepted or rejected in admissibility semantics. However, since there are no attacks, the distinction between these two admissible extensions cannot be represented by attack semantics. We therefore start with a general approach called argument - attack semantics, and we restrict attack semantics to complete and stronger semantics.

Third, Baroni *et al.* [2005] observe that "SCC-recursiveness can be assumed as a basic unifying concept in argumentation theory." Moreover, the SCC recursive scheme uses a generalization of Dung's semantics, in which the acceptability of arguments is considered with respect to what we call the qualified arguments. This raises the question how to define a recursive scheme for attack semantics, and how to define a corresponding generalization for attack semantics. Our generalization represents the unqualified arguments as arguments which are attacked from outside the argumentation framework. Moreover, we define also a constructive algorithm for SCC. We introduce an attack labeling approach for the algorithm, distinguishing among attacks which are successful because their attacking argument is accepted, and attacks which are successful although their attacking argument is not accepted. An attack is *in* when its attacker is *in*, *undecided* when its attacker is *undecided*, and *out* when its attacker is *out*. Consequently, an attack is successful when it is *in* or *undecided*, and unsuccessful when it is *out*.

The paper is organized as follows. Section 2 recalls Dung's argument semantics and introduces attack semantics and labeling, and Section 3 introduces the constructive SCC recursive procedure for attack semantics.

2 Semantics

Section 2.1 recalls Dung's semantics, distinguishing among accepted and rejected *arguments*. Our terminology differs from the usual one to contrast the standard approach with the new attack semantics explained in the following, distinguishing among successful and unsuccessful *attacks*. We start Section 2.2 with a combined argument - attack semantics for the general case, and Section 2.3 introduces attack semantics for complete and stronger semantics. In Section 2.4 we discuss attack labelings, which we use to define Algorithm 2 in Section 3.

2.1 Argument semantics

Dung [1995] introduces what we call here argument semantics, associating with each graph a set of so-called argument extensions, which are subsets of the nodes of the graph. Some authors therefore call it extension based semantics [Baroni and Giacomin, 2007].

Our presentation follows Baroni *et al.* [2005], where the acceptability of arguments is considered with respect to a designated subset of arguments, since we need this generalization to define the SCC recursive algorithm in Section 3. This set, which we call the set of qualified arguments, contains the arguments that an extension may consist of. Intuitively, it is used to filter out arguments that do not qualify for acceptance. This is necessary when we evaluate only a subset of arguments, but at the same time know that some arguments in this subset cannot be accepted due to attacks from outside the subset.

Definition 1 assumes an underlying mechanism of argument generation defining a set of arguments [Baroni and Giacomin, 2007], which is typically infinite, and which we call the universe of arguments and represent by U. An acceptance function $\mathcal{E}$ is a function that associates sets of acceptable arguments with a finite set of arguments $A \subseteq U$, a set of arguments produced by a reasoner at a given instant of time, a binary relation $\rightarrow \subseteq A \times A$, representing the attack relation among these arguments, and what we call here the qualified arguments $Q \subseteq A$.

Definition 1 (Argument semantics) *Let U be the universe of arguments. An acceptance function*

$$\mathcal{E} : 2^U \times 2^{U \times U} \times 2^U \rightarrow 2^{2^U}$$

is a partial function which is defined for each argumentation framework $\langle A, \rightarrow \rangle$ with finite $A \subseteq U$ and $\rightarrow \subseteq A \times A$, together with qualified arguments $Q \subseteq A$, and which associates with argumentation framework $\langle A, \rightarrow \rangle$ and qualified arguments Q sets of subsets of A: $\mathcal{E}(\langle A, \rightarrow \rangle, Q) \subseteq 2^A$. If $a \rightarrow b$ we say that a attacks b, a is an attacker of b, or b is attacked by a. If $E \in \mathcal{E}(AF, Q)$, then we say that E is an extension of AF and Q under the $\mathcal{E}$ semantics, and we say that the arguments in E are accepted and the arguments not in E are rejected.

Admissibility semantics in Definition 2 accepts only arguments that are defended by other accepted arguments, and therefore $\mathcal{E}(\langle A, \rightarrow \rangle, Q) \subseteq 2^Q$. Other authors have introduced semantics for which this property does not hold, e.g., Baroni *et al.* [2005] introduced CF1 and CF2 semantics. They observe also that Dung's semantics can be recovered by setting Q to A.

Definition 2 (Admissibility semantics) *Let $AF = \langle A, \rightarrow \rangle$ be an argumentation framework, and let $Q \subseteq A$ and $E \subseteq Q$ be sets of arguments. cf(E) if and only if there are no arguments $a, b \in E$ such that $a \rightarrow b$. E defends a if and only if $\forall b \in A$ such that $b \rightarrow a$, $\exists c \in E$ such that $c \rightarrow b$. Let $D(E) = \{a \in Q \mid E \text{ defends } a\}$.*

- $E \in \mathcal{E}_{admiss}(AF, Q)$ *if and only if* cf(E) *and* $E \subseteq D(E)$.
- $E \in \mathcal{E}_{compl}(AF, Q)$ *if and only if* cf(E) *and* $E = D(E)$.
- $E \in \mathcal{E}_{ground}(AF, Q)$ *if and only if E is smallest in $\mathcal{E}_{compl}(AF, Q)$ w.r.t. set inclusion.*
- $E \in \mathcal{E}_{pref}(AF, Q)$ *if and only if E is largest in $\mathcal{E}_{admiss}(AF, Q)$ w.r.t. set inclusion.*
- $E \in \mathcal{E}_{stable}(AF, Q)$ *if and only if* cf(E) *and* $\forall b \in A \backslash E \; \exists a \in E : a \rightarrow b$.

If $E \in \mathcal{E}_{admiss}(AF, Q)$ we say that E is an admissible extension of AF and Q, etc.

Example 2 *Consider $AF = \langle A, \rightarrow \rangle$ with $A = \{a, b, c, d, e\}$ and $\{a \rightarrow b, b \rightarrow a, b \rightarrow c, c \rightarrow d, d \rightarrow e, e \rightarrow c\}$ visualized in Figure 2. The complete extensions are $\mathcal{E}_{compl}(AF, A) = \{\emptyset, \{a\}, \{b, d\}\}$. $\emptyset$ is the unique grounded extension, $\{a\}$ and $\{b, d\}$ are the preferred extensions, and $\{b, d\}$ is the stable extension. In the complete extension $\emptyset$ the arguments are rejected, because they are not defended. In the extension $\{b, d\}$ the other arguments are rejected, because they are attacked by an accepted counterargument.*

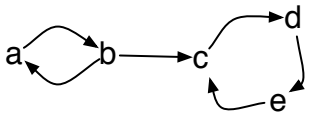

Figure 2: The argumentation framework of Example 2.

2.2 Attack - argument semantics

We generalize the argument semantics by selecting from the graph not only a set of nodes, but also a set of edges. This represents intuitively that attacks can be successful or unsuccessful. A similar kind of intuition is formalized in extended argumentation frameworks with second or higher order attacks by [Barringer *et al.*, 2005; Modgil and Bench-Capon, 2008; Boella *et al.*, 2009; Baroni *et al.*, 2011b], where attacks are treated as arguments which can be accepted and rejected too. In this paper we do not consider second or higher order attacks, though our generalization may be of use for such extended argumentation frameworks too.

Definition 3 represents argument - attack semantics by a function associating with each graph a set of its so-called

argument-attack extensions, which are sets of nodes and sets of edges from the graph. An argument-attack extension can contain an edge between arguments a and b, without containing the arguments a and b themselves. In other words, the argument attack extensions do not have to be subgraphs.

Since we aim to rephrase all concepts in terms of attack, the qualified arguments are implicitly represented as arguments not attacked from outside the argumentation framework: $Q = \{a \in A \mid \not\exists b \in U \setminus A : b \rightarrow a\}$. We therefore extend $\rightarrow$ to range over $U \times A$, and we do not add Q as a third parameter to the acceptance function $\mathcal{F}$.

Definition 3 (Argument - attack semantics) *An acceptance function $\mathcal{F} : 2^U \times 2^{U \times U} \rightarrow 2^{2^U \cup 2^{U \times U}}$ is a partial function which is defined for each argumentation framework $\langle A, \rightarrow \rangle$ with finite $A \subseteq U$ and $\rightarrow \subseteq U \times A$, which associates with argumentation framework $\langle A, \rightarrow \rangle$ sets of subsets of A and $\rightarrow$: $\mathcal{F}(\langle A, \rightarrow \rangle) \subseteq 2^{A \cup \rightarrow}$. For each attack $a \rightarrow b$, we say that a is the attacking argument, and b is the attacked argument. Moreover, for an argument a, we say that all its attacks are the attacks $a \rightarrow b$, and we say that all the attacks on a are all attacks $b \rightarrow a$.*

We *can* enforce additional constraints on argument attack semantics, just as additional constraints are sometimes enforced on argument semantics. The qualified constraint says that the accepted arguments depend only on whether an argument is attacked by an argument outside of A, not on the identity of the attackers outside the argumentation framework. The attacker dependence constraint says that if one attack by an argument is successful, then all attacks by this argument are successful. The successful attack constraint says that attacks by accepted argument are successful (but not necessarily vice versa). The unsuccessful attack constraint says that the attacks by arguments which are attacked by an accepted argument, are not successful. The defended attack constraint says that if there is no accepted argument attacking the attacker of an attack, then the attack is successful. The success of an attack in terms of the acceptance of arguments constraint says that an attack $b \rightarrow c$ is unsuccessful if and only if there is an accepted argument a attacking b. Therefore, an attack $b \rightarrow c$ is successful if and only if either its attacking argument b is accepted, or for all arguments a attacking argument b, argument a is rejected.

Definition 4 (Constraints)

Qualified *For all $\langle A, \rightarrow_1 \rangle$ and $\langle A, \rightarrow_2 \rangle$, if same attacks in A: $\rightarrow_1 \cap (A \times A) = \rightarrow_2 \cap (A \times A)$ and same attacked from outside: $\{a \in A \mid \exists b \in U \setminus A : b \rightarrow_1 a\} = \{a \in A \mid \exists b \in U \setminus A : b \rightarrow_2 a\}$, then same accepted arguments $\mathcal{F}(\langle A, \rightarrow_1 \rangle) \cap A = \mathcal{F}(\langle A, \rightarrow_2 \rangle) \cap A$.*

Attacker dependence *$\forall E \cup \hookrightarrow \in \mathcal{F}(\langle A, \rightarrow \rangle)$: If $a \rightarrow b$ and $a \rightarrow c$, then either both attacks are successful, or none of them.*

Successful attack *$\forall E \cup \hookrightarrow \in \mathcal{F}(\langle A, \rightarrow \rangle)$: if both $a \in E$ and $a \rightarrow b$, then $a \hookrightarrow b$;*

Unsuccessful attack *$\forall E \cup \hookrightarrow \in \mathcal{F}(\langle A, \rightarrow \rangle)$: if $a \in E$, $a \rightarrow b$ and $b \rightarrow c$, then we do not have $b \hookrightarrow c$, i.e. $b \not\hookrightarrow c$;*

Defended attack *$\forall E \cup \hookrightarrow \in \mathcal{F}(\langle A, \rightarrow \rangle)$: if $b \rightarrow c$ and $\not\exists a \in E : a \rightarrow b$, then $b \hookrightarrow c$;*

Success in terms of acceptance *$\forall E \cup \hookrightarrow \in \mathcal{F}(\langle A, \rightarrow \rangle)$: $b \hookrightarrow c$ if and only if $b \rightarrow c$ and $\not\exists a \in E : a \rightarrow b$.*

Proposition 1 *Success in terms of acceptance implies attacker dependence, successful attack, unsuccessful attack, and the defended attack constraint.*

In the following, we consider argument attack semantics together with the constraint that defines success in terms of acceptance. However, this does not necessarily imply that also acceptance can be defined in terms of success. For example, consider again the argumentation framework with a single argument and an empty attack relation. There may be two extensions under the success in terms of acceptance constraint, since the single argument can be accepted or rejected. However, since there are no attacks, the acceptance of this argument cannot be represented by the success of attacks.

The following definition uses the constraint to relate argument semantics to argument attack semantics.

Definition 5 (Relation) *The argument semantics $\mathcal{E}$ and argument attacks semantics $\mathcal{F}$ are equivalent, written as $\mathcal{E} = \mathcal{F}$, if $E \cup \hookrightarrow \in \mathcal{F}(\langle A, \rightarrow \rangle)$ if and only if*

- *$E \in \mathcal{E}(\langle A, \rightarrow \cap A \times A \rangle, Q)$, for qualified arguments $Q = \{a \in A \mid \not\exists b \in U \setminus A : b \rightarrow a\}$, and*
- *$b \hookrightarrow c \Leftrightarrow b \rightarrow c \wedge \not\exists a \in E : a \rightarrow b$.*

We write $\mathcal{F}_{admiss}$ for the acceptance function equivalent to $\mathcal{E}_{admiss}$, etc.

The following example illustrates grounded, preferred and stable semantics using argument attack semantics.

Example 3 (Continued from Example 2) *The grounded extension $\emptyset$ has only successful attacks, the preferred extension $\{a\}$ has $\{a \hookrightarrow b, c \hookrightarrow d, d \hookrightarrow e, e \hookrightarrow c\}$ as successful attacks, and the stable extension $\{b, d\}$ has $\{b \hookrightarrow a, b \hookrightarrow c, d \hookrightarrow e\}$.*

Proposition 2 *If the universe of arguments U is infinite, then for every $\mathcal{E}$, there is an $\mathcal{F}$ equivalent to it.*

Proof. *By construction. Assume U is infinite, and consider $\langle A, \rightarrow_1 \rangle$ and Q. Since U is infinite and A is finite, there exists an argument a in $U \setminus A$. Let $\rightarrow_2 = \rightarrow_1 \cup \{a \rightarrow_2 b \mid b \in A \setminus Q\}$.*

Proposition 3 *If the universe of arguments U is infinite, then for every $\mathcal{F}$ with success in terms of acceptance, and the qualified constraint, there is precisely one $\mathcal{E}$ equivalent to it.*

Proof (sketch). *The construction in Proposition 2 maps each element in $\mathcal{E}$ to an $\mathcal{F}$ satisfying the two constraints.*

Proposition 4 characterizes defense and reinstatement as the failure of attacks from arguments which are attacked by accepted arguments: if an argument a is accepted, then for all arguments b attacked by a, the attacks of b are unsuccessful.

Proposition 4 (Defense) *If $E \cup \hookrightarrow \in \mathcal{E}(\langle A, \rightarrow \rangle)$, then for all arguments a, b, c we cannot have $a \in E$, $a \rightarrow b$ and $b \hookrightarrow c$.*

Proof. *Assume $a \in E$ and $a \rightarrow b$. Since E is conflict free, $\not\exists a' \in E : a' \rightarrow a$, and consequently $a \hookrightarrow b$ due to the successful in terms of acceptance constraint. Therefore we cannot have $b \hookrightarrow c$ due to the same constraint.*

Proposition 5 shows that attacks from outside the argumentation framework are always successful.

Proposition 5 (Context) *If $E \cup \hookrightarrow \in \mathcal{E}(\langle A, \rightarrow \rangle)$, then for all arguments a and b we have that $a \rightarrow b$ and $a \notin A$ implies $a \hookrightarrow b$.*

Proof (sketch). *Follows from the fact that there are no attacks on arguments outside A, together with the success in terms of acceptance constraint.*

The following theorems are proven directly from the definitions presented thus far, but could also have been proven via the alternative characterization using labeling semantics discussed in Section 2.4. An extension is admissible if and only if for all accepted arguments, none of the attacks on it are successful.

Theorem 1 (Admissible) *$E \cup \hookrightarrow \in \mathcal{F}_{admiss}(AF)$ if and only if $cf(E)$ and $\forall a \in E \ \nexists b \in U : b \hookrightarrow a$.*

Proof. *$\Rightarrow$: follows directly from the success in terms of acceptance constraint. $\Leftarrow$: We prove the contrapositive. Assume the extension $E \cup \hookrightarrow$ is not admissible, then there is an accepted argument $a \in E$ which is not defended against an attack from argument b: $b \rightarrow a$ and $\nexists c \in E : c \rightarrow b$. From the definition of successful attack, this implies that the attack of b on a is successful: $b \hookrightarrow a$.*

A complete extension is an admissible extension, in which an argument is accepted if and only if none of the attacks on it are successful.

Theorem 2 (Complete) *$E \cup \hookrightarrow \in \mathcal{F}_{compl}(AF)$ if and only if $cf(E)$ and $a \in E \Leftrightarrow \nexists b \in U : b \hookrightarrow a$.*

Proof. *By Definition 2, E is a set of arguments c such that for all arguments b attacking c there is an argument $a \in E$ attacking b. $\Leftrightarrow$ E is a set of arguments c such that there is no argument b attacking c for which there is no argument $a \in E$ attacking b. $\Leftrightarrow$ E is the set of arguments for which there is no argument b successfully attacking it. The latter step follows directly from success in terms of acceptance.*

Theorem 3 characterizes complete semantics in terms of the success of attacks, which is the basis for the attack semantics in Section 2.3. In complete semantics, an attack $c \rightarrow d$ is successful if and only if for each successful attack $b \rightarrow c$, there is a successful attack $a \rightarrow b$.

Theorem 3 (Complete: Success propagation) *$E \cup \hookrightarrow \in \mathcal{F}_{compl}(AF)$ if and only if for each $c \rightarrow d$ we have $c \hookrightarrow d$ if and only if for each $b \hookrightarrow c$ we have an $a \hookrightarrow b$.*

Proof (sketch). *Follows from Theorem 2 together with the success in terms of acceptance.*

A grounded extension is a complete extension with a maximal set of successful attacks.

Theorem 4 (Grounded) *$E \cup \hookrightarrow \in \mathcal{F}_{ground}(AF)$ if and only if $E \cup \hookrightarrow \in \mathcal{F}_{compl}(AF)$ and there is no $E' \cup \hookrightarrow' \in \mathcal{F}_{compl}(AF)$ with $\hookrightarrow \subset \hookrightarrow'$.*

Proof (sketch). *Follows from Proposition 2,3 and Theorem 2.*

A preferred extension is an admissible extension with a minimal set of successful attacks.

Theorem 5 (Preferred) *$E \cup \hookrightarrow \in \mathcal{F}_{pref}(AF)$ if and only if $E \cup \hookrightarrow \in \mathcal{F}_{admiss}(AF)$ and there is no $E' \cup \hookrightarrow' \in \mathcal{F}_{admiss}(AF)$ with $\hookrightarrow' \subset \hookrightarrow$.*

Proof (sketch). *Follows from Proposition 2,3 and Theorem 2.*

A stable extension has no successive attacks.

Theorem 6 (Stable) *$E \cup \hookrightarrow \in \mathcal{F}_{stable}(AF)$ if and only if $cf(E)$ and there are no arguments a, b, c with $a \hookrightarrow b$ and $b \hookrightarrow c$.*

Proof (sketch). *Follows from Proposition 2,3 and Theorem 2.*

2.3 Attack semantics

From now on, in this paper we consider only complete semantics or one of its refinements, such as grounded, preferred or stable semantics. We introduce the following attack semantics, a function that associates with each graph a set of its attack extensions, which are sets of edges from the graph.

Definition 6 (Attack semantics) *An acceptance function $\mathcal{A} : 2^U \times 2^{U \times U} \rightarrow 2^{2^{U \times U}}$ is a partial function which is defined for each argumentation framework $\langle A, \rightarrow \rangle$ with finite $A \subseteq U$ and $\rightarrow \subseteq U \times A$, which associates with argumentation framework $\langle A, \rightarrow \rangle$ sets of subsets of $\rightarrow$: $\mathcal{A}(\langle A, \rightarrow \rangle) \subseteq 2^{\rightarrow}$. If $a \rightarrow b$ and $b \rightarrow c$ we say that $a \rightarrow b$ is a predecessor of $b \rightarrow c$, and that $b \rightarrow c$ is a successor of $a \rightarrow b$.*

Theorem 3-6 can be rephrased in terms of attack semantics, because for complete semantics (and stronger) an argument is accepted when there are no successful attacks on it (Theorem 2).

Corollary 1 (Complete Attack Semantics)

$\hookrightarrow \in \mathcal{A}_{compl}(AF)$ if and only if for each $c \rightarrow d$ we have $c \hookrightarrow d$ if and only if for each $b \hookrightarrow c$ we have an $a \hookrightarrow b$.

$\hookrightarrow \in \mathcal{A}_{ground}(AF)$ if and only if $\hookrightarrow \in \mathcal{A}_{compl}(AF)$ and there is no $\hookrightarrow' \in \mathcal{A}_{compl}(AF)$ with $\hookrightarrow \subset \hookrightarrow'$.

$\hookrightarrow \in \mathcal{A}_{pref}(AF)$ if and only if $\hookrightarrow \in \mathcal{A}_{compl}(AF)$ and there is no $\hookrightarrow' \in \mathcal{A}_{compl}(AF)$ with $\hookrightarrow' \subset \hookrightarrow$.

$\hookrightarrow \in \mathcal{A}_{stable}(AF)$ if and only if $\hookrightarrow \in \mathcal{A}_{compl}(AF)$ and there are no arguments a, b, c with $a \hookrightarrow b$ and $b \hookrightarrow c$.

Example 4 *Consider the argumentation framework $AF = \langle A, \rightarrow \rangle$ with arguments $A = \{a, b, c, d, e, f\}$ and attacks $\{a \rightarrow b, b \rightarrow a, b \rightarrow c, c \rightarrow d, d \rightarrow e, e \rightarrow f\}$, visualized in Figure 3. Assume that we know that a and b are rejected. Then the attacks $a \rightarrow b$ and $b \rightarrow a$ are successful, and consequently we can derive by attack propagation that all attacks are successful.*

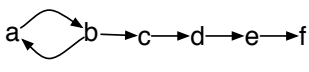

Figure 3: The argumentation framework of Example 4.

2.4 Labeling semantics

In this section we show how to use labelings [Jakobovits and Vermeir, 1999; Caminada, 2006] to make the relation between argument and attack semantics more precise, which is useful for Algorithm 2 in Section 3. We consider the set of labelings of an argumentation framework satisfying the following conditions: an argument is in if and only if all its attackers are out, an argument is out if and only if there is an attacker that is in, and an argument is undecided if and only if none of its attackers is in, and at least one of its attackers is undecided.

We now introduce an attack labeling semantics. Whereas argument labelings partition rejected arguments into *out* and *undecided* arguments, attack labelings partition successful attacks into *in* and *undecided* attacks. Intuitively, an undecided attack is like a threat: it is not based on an accepted attacking argument, but it can still be used to reject the attacked argument.

Definition 7 (Attack labeling semantics) *Let U be the universe of arguments, and let $\mathcal{L}$ be the set of functions from relations over arguments $\rightarrow \subseteq U \times U$ to $\{in, out, und\}$. An acceptance function $\mathcal{E} : 2^U \times 2^{U\times U} \rightarrow 2^{\mathcal{L}(U\times U)}$ is a partial function which is defined for each argumentation framework $\langle A, \rightarrow\rangle$ with finite $A \subseteq U$ and $\rightarrow \subseteq U \times A$, which associates with argumentation framework $\langle A, \rightarrow\rangle$ sets of labelings of $\rightarrow$: $\mathcal{E}(\langle A, \rightarrow\rangle) \subseteq \mathcal{L}(\rightarrow)$.*

We define attack labeling analogous to argument labeling. An attack is *in* when its attacking argument is *in*, an attack is *undecided* when its attacking argument is *undecided*, and an argument is *out* when its attacking argument is *out*. An attack is successful when it is *in* or *undecided*, whereas an argument is accepted when it is *in*. For example, if an argument is rejected, but at least one of its attacks is successful, then the argument is undecided.

Example 5 (Continued from Example 3) *The grounded extension $\emptyset$ has only undecided attacks, the preferred extension $\{a\}$ has in attack $a \hookrightarrow b$ and undecided attacks $c \hookrightarrow d, d \hookrightarrow e, e \hookrightarrow c$, and the stable extension $\{b, d\}$ has in attacks $b \hookrightarrow a, b \hookrightarrow c, d \hookrightarrow e$ and no undecided attacks.*

We now rephrase the argument labeling conditions in terms of attack labeling. For complete semantics, each labeling corresponds to a complete extension, where the in arguments of the labeling are the accepted arguments of the extension. Analogously, an attack is in if and only if all its predecessors are out, an attack is out if and only if there is a predecessor that is in, and an attack is undecided if and only if none of its predecessors is in, and at least one of its predecessors is undecided.

Theorem 7 *Complete semantics is characterized by the following two conditions.*

1. *For all b and c we have $L(b \rightarrow c) =$out, if and only if there is an a such that we have $L(a \rightarrow b) =$in.*
2. *For all b and c we have $L(b \rightarrow c) =$in, if and only if for all a we have $L(a \rightarrow b) =$out.*

3 SCC recursive algorithm

3.1 SCC-recursive argumentation semantics

In this subsection, we describe [Baroni *et al.*, 2005]'s SCC-recursive scheme which is based on the graph theoretical notion of strongly connected component, i.e., SCCs provide a unique partition of a directed graph into disjoint parts where all nodes are mutually reachable. The authors observe that "the underlying basic ideas can be summarized as follows: (1) the argumentation framework is partitioned into its strongly connected components; they form a partial order which encodes the dependencies existing among them according to the directionality principle; (2) the possible choices for extensions within each initial strongly connected component are determined using a semantic-specific base function which returns the extensions of argumentation frameworks consisting of a single strongly connected component; (3) for each possible choice determined at step 2, according to the reinstatement principle, the nodes directly attacked within subsequent strongly connected components are suppressed and the distinction between defended and undefended nodes is (possibly) taken into account; (4) the steps 1–3 above are applied recursively on the restricted argumentation frameworks obtained at step 3."

Most presentations of the SCC recursive scheme (e.g., [Baroni and Giacomin, 2009; Baroni *et al.*, 2011a]) present only the following definition, and they observe that it is well founded. They also observe that it is quite complicated and its detailed illustration is beyond the scope of the paper; we also do not attempt to present it here in further detail, but refer the reader to [Baroni *et al.*, 2005].

Definition 8 ([Baroni *et al.*, 2005], Definition 20) *A given argumentation semantics is $SCC - recursive$ if and only if for any argumentation framework $AF = \langle A, \rightarrow\rangle$, $\mathcal{E}_S(AF) = GF(AF, A)$, where for any $AF = \langle A, \rightarrow\rangle$ and for any set $C \subseteq A$, the function $GF(AF, C) \subseteq 2^A$ is defined as follows: for any $E \subseteq A$, $E \in GF(AF, C)$ if and only if*

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

where

- *$SCCS_{AF}$ denotes the set of strongly connected components of AF (a maximal set such that in the transitive closure of $\rightarrow$, each argument of an SCC attacks all other ones),*
- *$BF_S(AF, C)$ is a function, called* base function*, that, given an argumentation framework $AF = \langle A, \rightarrow\rangle$ s.t. $|SCCS_{AF}| = 1$ and a set $C \subseteq A$, gives a subset of 2^A,*
- *$UP_{AF}(S, E) = \{a \in S \mid \nexists b \in E : b \notin S, b \rightarrow a\}$, all arguments that are not attacked by an accepted argument outside of S,*
- *$U_{AF}(S, E) = \{a \in UP_{AF}(S, E) \mid \forall b \notin E : b \notin S, b \rightarrow a \Rightarrow \exists c \in E : c \notin S, c \rightarrow b\}$, all arguments that are defended against attacks from outside of S, and*
- *$AF \downarrow S = \langle S, \rightarrow \cap E \times E)\rangle$, the restriction of AF to S.*

Baroni *et al.* [2005] show also that the base function for admissible, grounded, preferred and stable semantics are the ones given in Definition 2, and therefore that all these semantics are SCC recursive.

Example 6 (Continued from Example 2) *Consider again the example visualized in Figure 2. There are two SCC $\{a, b\}$ and $\{c, d, e\}$. The SCC $\{a, b\}$ is not attacked by anything, $\{a\}$ and $\{b\}$ are the complete extensions of this SCC. Considering $\{a\}$, the SCC we get is $\{c, d, e\}$. Considering, instead, $\{b, d\}$, since b is in, the SCC $\{c, d, e\}$ goes into recursion and becomes $\{d, e\}$, which consists of two SCCs, $\{d\}$ and $\{e\}$.*

3.2 Iterative procedure

Baroni *et al.* [2005] observe that Definition 8 has also a straightforward constructive interpretation, but they do not give a procedure. We believe that Algorithm 1 is easier to understand than Definition 8. The X label stands for unlabeled arguments in a partial labeling. Line 2 starts a loop where partial labelings are extended by first in lines 4-7 applying the so-called characteristic function, and then in lines 11-19 applying the base function BF. The characteristic function in lines 4-7 assigns the *in* label to the arguments attacked only by *out* arguments, and *out* to arguments attacked by at least one *in* argument, following [Caminada, 2006]. Lines 8-10 assign the label *und* to those arguments which are attacked by *out* and *und* arguments only (and therefore at least by one *und* argument, otherwise it would already have been labeled *in*); these lines are not strictly necessary, but speed up the algorithm. In line 11, $SCC(\langle A, \rightarrow\rangle, L)$ returns the sets of X-labeled arguments of A that (1) are maximal strongly connected components and (2) are not attacked by any X-labeled arguments. If such a set exists, then $BF(\langle A, \rightarrow\rangle, C)$ applies the base function to $\langle A, \rightarrow\rangle$ where C corresponds to arguments which are defended against attacks from outside of $\langle A, \rightarrow\rangle$.

Theorem 8 (Soundness and completeness) *The iterative procedure is sound and complete with respect to the SCC recursive scheme, in the sense that it returns all labelings that can be reconstructed using the base function.*

3.3 SCC for attack semantics

For argument semantics, the SCC scheme partitions a SCC into defended and undefended arguments. For attack semantics, instead, an SCC consists in a set of attacks, together with attacks from outside the SCC. Algorithm 2 is the counterpart of Algorithm 1 for attack semantics by using the relations shown in Section 2. Line 2 starts a loop where partial labelings are extended by first in lines 4-10 applying the characteristic function, and then in lines 14-21 applying the base function for attack semantics BF_{Att}. The characteristic function in lines 4-10 assigns the label *in* to the attacks whose predecessors are labeled *out*, and attacks with a predecessor which is labeled *out* are labeled *in*. In lines 11-13, the attacks whose predecessors are not X-labeled get the label *und*; also for attack semantics these lines are not strictly necessary, but speed up the algorithm. Finally, similarly to Algorithm 1, $SCC_{Att}(AF, L)$ returns the sets of X-labeled attacks that (1) form a SCC together with those attacks on

Input: $AF = \langle A, \rightarrow\rangle, BF$
Output: $\mathcal{L}$
1 $\mathcal{L} := \{L = \{L(a) = X | a \in A\}\}$
2 **while** $\exists L \in \mathcal{L} : \exists a \in A : L(a) = X$ **do**
3 $\quad \mathcal{L} = \mathcal{L} \setminus \{L\}$;
4 $\quad$ **while** $\exists a \in A : L(a) = X \wedge \forall b \rightarrow a : L(b) =$ *out* **do**
5 $\quad\quad L(a) :=$ *in*;
6 $\quad\quad \forall b : a \rightarrow b \Rightarrow L(b) :=$ *out*;
7 $\quad$ **end**
8 $\quad$ **while** $\exists a \in A : L(a) = X \wedge \forall b \rightarrow a : L(b) =$ *out* $\vee L(b) =$ *und* **do**
9 $\quad\quad L(a) :=$ *und*;
10 $\quad$ **end**
11 $\quad$ **if** $\exists S \in SCC(AF, L)$ **then**
12 $\quad\quad$ **foreach** $E \in BF(AF \downarrow S, \{s \in S : \nexists b \in A \setminus S : b \rightarrow s \wedge L(b) =$ *und*$\})$ **do**
13 $\quad\quad\quad L' := L$;
14 $\quad\quad\quad \forall a \in E \Rightarrow L(a) :=$ *in*;
15 $\quad\quad\quad \forall b \in S\ \exists a \in E : a \rightarrow b \Rightarrow L(b) :=$ *out*;
16 $\quad\quad\quad \forall b \in S : L(b) = X \Rightarrow L(b) :=$ *und*;
17 $\quad\quad\quad \mathcal{L} := \mathcal{L} \cup \{L'\}$;
18 $\quad\quad$ **end**
19 $\quad$ **end**
20 **end**

Algorithm 1: The SCC algorithm for argument semantics.

Input: $AF = \langle A, \rightarrow\rangle, BF_{Att}$
Output: $\mathcal{L}ab$
1 $\mathcal{L} = \{L = \{L(a) = X | a \in \rightarrow\}\}$
2 **while** $\exists L \in \mathcal{L} : \exists a \rightarrow b : L(a \rightarrow b) = X$ **do**
3 $\quad \mathcal{L} = \mathcal{L} \setminus L$;
4 $\quad$ **while** $\exists a \rightarrow b : L(a \rightarrow b) = X$ **do**
5 $\quad\quad$ **while** $\exists a \rightarrow b : L(a \rightarrow b) = X \wedge \forall c \rightarrow a : L(c \rightarrow a) =$ *out* **do**
6 $\quad\quad\quad L(a \rightarrow b) :=$ *in*;
7 $\quad\quad\quad$ **while** $\exists a \rightarrow b : L(a \rightarrow b) = X \wedge \exists c \rightarrow a : L(c \rightarrow a) =$ *in* **do**
8 $\quad\quad\quad\quad L(a \rightarrow b) :=$ *out*;
9 $\quad\quad\quad$ **end**
10 $\quad\quad$ **end**
11 $\quad\quad$ **while** $\exists a \rightarrow b : L(a \rightarrow b) = X \wedge \forall c \rightarrow a : L(c \rightarrow a) \neq X$ **do**
12 $\quad\quad\quad L(a \rightarrow b) :=$ *und*;
13 $\quad\quad$ **end**
14 $\quad\quad$ **if** $\exists \langle A', \rightarrow'\rangle \in SCC_{Att}(AF, L)$ **then**
15 $\quad\quad\quad$ **foreach** $\hookrightarrow' \in BF_{Att}(\langle A', \rightarrow'\rangle)$ **do**
16 $\quad\quad\quad\quad L' := L$;
17 $\quad\quad\quad\quad \forall a \rightarrow' b : a \not\hookrightarrow b \Rightarrow L(a \rightarrow b) :=$ *out*;
18 $\quad\quad\quad\quad \forall a \hookrightarrow b : \neg\exists c \hookrightarrow a \Rightarrow L(a \rightarrow b) :=$ *in*;
19 $\quad\quad\quad\quad \forall a \rightarrow' b : L(a \rightarrow b) = X \Rightarrow L(a \rightarrow b) :=$ *und*;
20 $\quad\quad\quad\quad \mathcal{L} := \mathcal{L} \cup \{L'\}$;
21 $\quad\quad\quad$ **end**
22 $\quad\quad$ **end**
23 $\quad$ **end**
24 **end**

Algorithm 2: The SCC_{Att} algorithm for attack semantics.

it labeled *und*, and that (2) do not have an X-labeled predecessor, and $BF_{Att}(\langle A, \rightarrow \rangle, C)$ is the base function for attack semantics, as given in Corollary 1 for various kinds of semantics. We illustrate how algorithm SCC_{Att} works by means of Example 7.

Example 7 (Continued) *Consider again the argumentation framework visualized in Figure 2. We start with the following situation, where L and $\mathcal{L}$ are the partial labeling and the set of starting partial labelings for the SCC_{Att} algorithm, respectively:*

- $\mathcal{L} = \{L\}$;
	$a \rightarrow b$	$b \rightarrow a$	$b \rightarrow c$	$c \rightarrow d$	$d \rightarrow e$	$e \rightarrow c$;
L	X	X	X	X	X	X
- $SCC_{Att}(.) = \{\{a \rightarrow b, b \rightarrow a\}\}$;
- $BF_{Att}(.) = \{\{a \rightarrow b, b \rightarrow a\}, \{a \rightarrow b\}, \{b \rightarrow a\}\}$;

We obtain the following partial labelings:

- $\mathcal{L} = \{L_1, L_2, L_3\}$;
	$a \rightarrow b$	$b \rightarrow a$	$b \rightarrow c$	$c \rightarrow d$	$d \rightarrow e$	$e \rightarrow c$;
L_1	*und*	*und*	X	X	X	X
L_2	*in*	*out*	X	X	X	X
L_3	*out*	*in*	X	X	X	X

Now, we select the first partial labeling, L_1, in order to compute a complete labeling:

- $\mathcal{L} = \{L_1, L_2, L_3\}$;
	$a \rightarrow b$	$b \rightarrow a$	$b \rightarrow c$	$c \rightarrow d$	$d \rightarrow e$	$e \rightarrow c$;
L_1	*und*	*und*	X	X	X	X
L_1	*und*	*und*	*und*	X	X	X
- $SCC_{Att}(.) = \{\{c \rightarrow d, d \rightarrow e, e \rightarrow c\}\}$;
- $BF_{Att}(.) = \{\{c \rightarrow d, d \rightarrow e, e \rightarrow c\}\}$;
	$a \rightarrow b$	$b \rightarrow a$	$b \rightarrow c$	$c \rightarrow d$	$d \rightarrow e$	$e \rightarrow c$;
L_1	*und*	*und*	*und*	*und*	*und*	*und*

L_1 is the first attack labeling we get. Then, we consider the other two partial labelings. We start with L_2.

	$a \rightarrow b$	$b \rightarrow a$	$b \rightarrow c$	$c \rightarrow d$	$d \rightarrow e$	$e \rightarrow c$;
L_2	*in*	*out*	X	X	X	X
L_2	*in*	*out*	*out*	X	X	X
- $SCC_{Att}(.) = \{\{c \rightarrow d, d \rightarrow e, e \rightarrow c\}\}$;
- $BF_{Att}(.) = \{\{c \rightarrow d, d \rightarrow e, e \rightarrow c\}\}$;
	$a \rightarrow b$	$b \rightarrow a$	$b \rightarrow c$	$c \rightarrow d$	$d \rightarrow e$	$e \rightarrow c$;
L_2	*in*	*out*	*out*	*und*	*und*	*und*

Then we consider the last partial labeling, L_3.

	$a \rightarrow b$	$b \rightarrow a$	$b \rightarrow c$	$c \rightarrow d$	$d \rightarrow e$	$e \rightarrow c$;
L_3	*out*	*in*	X	X	X	X
L_3	*out*	*in*	*in*	X	X	X
L_3	*out*	*in*	*in*	*out*	X	X
L_3	*out*	*in*	*in*	*out*	*in*	X
L_3	*out*	*in*	*in*	*out*	*in*	*out*

Theorem 9 follows for complete semantics, when we have a one to one correspondence between argument labelings and attack labelings.

Theorem 9 (Soundness and completeness) *For complete semantics and its refinements, Algorithm 2 returns the set of all attack labelings.*

3.4 Implementation

The two algorithms above have been implemented, and we are collecting and evaluating preliminary results. The source code and additional information can be found at:

`http://argumentationpatterns.com`

The prototype has been developed using Python programming language v2.6., an interpreted, general-purpose high-level programming language, whose reference implementation (CPython) is an open source software. In particular, we developed the Python prototype, using the `python.graph` package[1] and exploiting its inner built-in features (e.g., search and traversal algorithms, cycle detection). In our implementation, the argumentation framework is a directed graph and the available SCCs are associated to the arguments attacking them, using dictionaries and lists.

The argumentation framework is characterized by the following structures:

1. a directed graph called `digraph`;
2. a list of strongly connected components. This is represented by means of a list of lists, each of them containing the nodes of the SCC itself;
3. a dictionary implementing the base function. The SCCs are the keys of the dictionary, and the arguments attacking each SCC are its values.

We use two algorithms classes provided by `python.graph`, concerning searching (e.g., `depth_first_searching` used for breadth-first search) and accessibility. The base function BF is computed starting from the attackers of a SCC from outside the SCC itself. After the graph is built and the BF is computed, the labeling can start by iterating on each label to be set. First of all, all arguments are labelled X. Then, each argument is labelled IN or OUT according to:

- being part of an attacked SCC;
- attacking a SCC;
- attacking/being attacked by a labelled argument;

At the end, remaining unlabeled arguments are labelled as UND, and the final set of labels is returned.

The algorithm always converges in a few steps, depending on the number of the attacked SCCs and on the number of attack relations. Preliminary results show also that maintaining a cache for intermediate labelings leads to a benefit in terms of number of operations (around 10% less operations of graph access).

Future work will be described on the website. For example, we are now exploring how to modify the algorithms for creating, accessing and manipulating graphs (e.g., by adopting NetworkX, a Python package for the creation, and manipulation of complex networks), in order to avoid cross checks in graph cycles management.

[1] http://code.google.com/p/python-graph/

4 Concluding remarks

To represent and reason with argumentation there is a choice between Dung's two valued argument evaluations [Dung, 1995] and three value labelings [Caminada, 2006]. Some people find it easier to work with the former, because it is the simplest model, and some other people find it easier to work with the latter, because the value of a label depends on the value of its attackers only. We find it more natural to conceptualize argumentation and dialogue in terms of successful and unsuccessful attacks, and we suspect that some other people might find it useful to use combinations of these approaches.

In this paper we therefore introduce attack semantics, that conceptualizes argumentation in terms of successful and unsuccessful attacks, such that arguments are accepted if and only if there are no successful attacks on them. We introduce also attack labeling semantics and general argument attack semantics. We compare these semantic approaches in detail with Dung's traditional extension based semantics, which we call argument semantics to contrast it with our attack semantics, and Jakobovits-Vermeir-Caminada labeling semantics. Since we focus on attack semantics coinciding with traditional semantics, there exist meaningful instantiations of our abstract theory. We discuss also limitations of attack semantics, in particular it cannot be used for admissible semantics (here, argument attack semantics or argument attack labelings must be used).

Moreover, in this paper we introduce a constructive SCC recursive procedure for attack semantics. In our opinion, SCC techniques are not useful only for computational reasons, but in particular also for conceptual reasons. For instance, as discussed extensively by Baroni *et. al* [2005], SCC techniques can be used to implement directionality. We believe this may be useful, when argumentation theory is further generalized to partial acceptance and partial success.

Extended argumentation frameworks with second or higher order attacks by [Barringer *et al.*, 2005; Modgil and Bench-Capon, 2008; Boella *et al.*, 2009; Baroni *et al.*, 2011b] treat attacks as arguments which can be accepted and rejected too. Assume, for instance, the $AFRA$ based argument α representing the attack $a \rightarrow b$. The acceptance of α corresponds to the success of the attack $a \rightarrow b$. Moreover, in meta-argumentation [Boella *et al.*, 2009] the attack $a \rightarrow b$ is flattened into four arguments $accept(a), X_{ab}, Y_{ab}$ and $accept(b)$ with $accept(a) \rightarrow X_{ab}$, $X_{ab} \rightarrow Y_{ab}$, and $Y_{ab} \rightarrow accept(b)$. If there are no other arguments attacking the auxiliary arguments X_{ab} and Y_{ab}, then we have that the label of the attack argument Y_{ab} is identical to the label of $accept(a)$.

Concerning further research, one may consider a common framework for iterated belief revision and abstract argumentation, such that the revision of b by a corresponds to attacks of a on b. In particular, one could define generalizations of belief revision with revision cycles. In iterated belief revision there are no revision cycles, but in argumentation theory there can be attack cycles among the arguments. Therefore the challenge is to introduce cycles in iterated belief revision too, and give them an intuitive explanation. Moreover, another challenge consists in introducing partial acceptance and success in abstract argumentation.

Acknowledgements

The first author acknowledges support of the DataLift Project ANR-10-CORD-09 founded by the French National Research Agency.

References

[Baroni and Giacomin, 2007] Pietro Baroni and Massimiliano Giacomin. On principle-based evaluation of extension-based argumentation semantics. *Artif. Intell.*, 171(10-15):675–700, 2007.

[Baroni and Giacomin, 2009] Pietro Baroni and Massimiliano Giacomin. Semantics of abstract argument systems. In I. Rahwan and G. Simari, editors, *Argumentation in Artificial Intelligence*, pages 25–44, 2009.

[Baroni *et al.*, 2005] Pietro Baroni, Massimiliano Giacomin, and Giovanni Guida. Scc-recursiveness: a general schema for argumentation semantics. *Artif. Intell.*, 168(1-2):162–210, 2005.

[Baroni *et al.*, 2011a] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *Knowledge Engineering Review*, 2011.

[Baroni *et al.*, 2011b] Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Giovanni Guida. AFRA: Argumentation framework with recursive attacks. *Int. J. Approx. Reasoning*, 52(1):19–37, 2011.

[Barringer *et al.*, 2005] Howard Barringer, Dov M. Gabbay, and John Woods. Temporal dynamics of support and attack networks: From argumentation to zoology. In *Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday, LNCS 2605, Springer*, pages 59–98, 2005.

[Boella *et al.*, 2009] Guido Boella, Dov M. Gabbay, Leendert van der Torre, and Serena Villata. Meta-argumentation modelling I: Methodology and techniques. *Studia Logica*, 93(2-3):297–355, 2009.

[Caminada, 2006] Martin Caminada. On the issue of reinstatement in argumentation. In *Procs. of Logics in Artificial Intelligence, 10th European Conference, JELIA, LNCS 4160, Springer*, pages 111–123, 2006.

[Dung, 1995] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.

[Jakobovits and Vermeir, 1999] Hadassa Jakobovits and Dirk Vermeir. Robust semantics for argumentation frameworks. *J. Log. Comput.*, 9(2):215–261, 1999.

[Modgil and Bench-Capon, 2008] Sanjay Modgil and Trevor J. M. Bench-Capon. Integrating object and meta-level value based argumentation. In *Procs. of Computational Models of Argument, COMMA, Frontiers in Artificial Intelligence and Applications, vol. 172, IOS Press*, pages 240–251, 2008.