Rewriting Rules for the Computation of Goal-Oriented Changes in an Argumentation System

Dionysios Kontarinis[1], Elise Bonzon[1], Nicolas Maudet[2], Alan Perotti[3], Leon van der Torre[4], and Serena Villata[5]

[1] LIPADE, Université Paris Descartes,
{dionysios.kontarinis,elise.bonzon}@parisdescartes.fr
[2] LIP6, Université Pierre et Marie Curie, nicolas.maudet@lip6.fr
[3] Turin University, perotti@unito.it
[4] University of Luxembourg, leon.vandertorre@uni.lu
[5] INRIA, Sophia Antipolis, serena.villata@inria.fr

Abstract. When several agents are engaged in an argumentation process, they are faced with the problem of deciding how to contribute to the current state of the debate in order to satisfy their own goal, ie. to make an argument under a given semantics accepted or not. In this paper, we study the minimal changes (or target sets) on the current state of the debate that are required to achieve such a goal, where changes are the addition and/or deletion of attacks among arguments. We study some properties of these target sets, and propose a Maude specification of rewriting rules which allow to compute all the target sets for some types of goals.

1 Introduction

Debates are pursued with the aim to obtain at the end a set of *accepted arguments*. As in [1–3], we assume that such debates are represented by a central, dynamic argumentation system which is modified by the agents' locutions. During these debates, each agent tries to argue in such a way that his own *argumentative goals* belong to the final set of accepted arguments in the central system. Given the number of participants and of proposed arguments, it is a challenging task to identify the part of the debate to focus on and to compute possible modifications which can affect the current state of the debate, in order to achieve a given argumentative goal.

We assume in this work, as it is done in [1–3], that the participating agents may disagree on the existence of binary attacks between some pairs of arguments. But what can cause such a disagreement? First, in the framework of value-based argumentation [4], a *defeat* relation between two arguments holds if there is a conflict between those arguments, and if the value promoted by the attacking argument is higher than the value promoted by the other argument. Therefore, if two agents order these values differently, they may disagree on the existence of this defeat relation. A similar type of reasoning is applied in preference-based argumentation [5]. Second, a usual phenomenon in everyday argumentation is the disagreement on the existence of a conflict between some pairs of arguments. Often, the claim of an argument does not explicitly contradict one of the premises of another argument (nor its claim), but it may still be considered that it is

attacking the latter. This is due to the use of enthymemes (arguments whose internal structures are not fully defined), as stated in [6].

In this work, for the sake of simplicity, we consider that the set of arguments and some attacks between those arguments are fixed in the debate. This can be done, for example, as follows. In the first phase of the debate, agents (following a given protocol) put forward all the arguments and attacks they consider relevant to the subject of the debate. Then follows a voting phase on the arguments and attacks which have been proposed by the agents. Afterwards, we assume, as it is done in [2], that all the arguments approved by a majority (for example) of agents are fixed and considered in the debate, as well as all the attacks on which a quasi-unanimity of agents have agreed. From that point on, the debate focuses on the attacks which have caused disagreement among the agents.

In our context, a *move* modifies the current state of the debate by either adding or removing attacks. A *successful move* brings about the acceptance or rejection of a particular argumentative goal, that is, ensures that a designated argument belongs (or not) to some (all) extension(s). Here we shall focus on (subset-)minimal successful moves, called *target sets* [7].

We acknowledge that, in some cases, focusing on minimal change may not be the best strategy for a debating agent. For example, if an agent is uncertain whether he will be able to assert additional moves during the debate, then it may be preferable for him to assert all the moves he can, as soon as possible. Also, if we consider a framework where the agents' personal beliefs are dynamic, a non-minimal move by an agent may be preferable for him, if it provokes some wished changes to the beliefs of the other agents. However, we believe that minimality is a useful notion in the study of argumentation dynamics. An agent may be motivated to find a minimal change satisfying his goal, because this would be the easiest, fastest way to do it. Moreover, it can be a good strategy for agents in a debate, as it minimizes their commitments.

Our first contribution in this paper is to provide some general properties of such (minimal) successful moves. We then put forward a set of rewriting rules for the Maude system [8], which exploit the recent attack semantics of [9] to compute target sets, and provide some properties of the resulting procedure.

Our work is inspired by proof theories for abstract argumentation frameworks, as in the work of [10], which treat the problem of how to prove the acceptance (or non-acceptance) of an argument under some semantics. The main new elements introduced here are the following. First, we consider dynamic systems where (several) attacks can be added and removed. Second, we focus on minimal change required to achieve acceptance (or non-acceptance) of an argument and we analyze the properties of minimal change.

Recently, the question of the *dynamics of argumentation systems* has been studied by several authors [11–15]. Baumann [14] studies different types of *expansions*, that is, different ways to modify the current system. For instance the most general kind, *arbitrary modifications*, allows the addition of new arguments, as well as the addition/removal of attacks. Formally, the problem studied is as follows: given a current argumentation system (AS), given a "goal set" E, find a minimal expansion such that E belongs to at least one extension of the modified AS. The notion of minimality differs from ours, since it relies on a pseudometric measuring the distance (in terms of number of differences between AS). Another key difference is that our modifications are typically constrained,

and also restricted to adding/removing attacks. Most importantly, our work focuses on the design of a procedure which returns the target sets in a given situation. The recent work of [16, 17] is also closely related. They share with us the view that the possible modifications of the system may be constrained, and also investigate practical computation techniques to enforce argumentative goals. In theory, their model caters for *sequences* of basic modifications of the system (and allows addition and removal of arguments as well). One important difference is that they do not focus on minimal changes. In practice, they design a tool which relies on characterization results for the dynamics of argumentation systems studied in [16]. The current implementation is restricted to single modifications [17].

The paper is organized as follows. Section 2 provides some basic background on abstract argumentation theory [18] and the notion of acceptability. Section 3 formalizes the notions of *successful moves* and of *target sets*, and highlights some important properties that they exhibit. In Section 4 we give the specification of rewriting rules to be used with the Maude system. We study some key properties that can (or cannot) be guaranteed with our approach. Finally, Section 5 concludes.

2 Background

In this section, we provide the basic concepts of abstract argumentation frameworks, as proposed by Dung [18], in which the exact content of arguments is left unspecified. In the definition of argumentation system we provide here, the difference compared to [18] is that we do not only have the standard attack relation (here denoted R), but we also have a relation R^+ which denotes the attacks which can be added to the system, and a relation R^- which denotes the attacks which can be removed from the system.

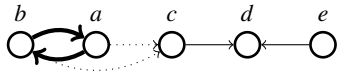
Definition 1. *We define an* **argumentation system** *as a tuple* $AS = \langle A, R, R^+, R^- \rangle$*, where* A *is a finite set of arguments,* $R \subseteq A \times A$ *is a binary attack relation between arguments,* $R^+ \subseteq A \times A$*, with* $R^+ \cap R = \{\}$*, contains the pairs of arguments which can be added in* R*, and* $R^- \subseteq R$ *contains the pairs of arguments which can be removed from* R*.*

As stated in the introduction, we assume that the arguments of such a system have been fixed on a previous phase where the agents have put forward the arguments they thought were pertinent for the debate, and have then voted on the arguments. Moreover, attacks in $R \setminus R^-$ are supposed to be attacks on which a quasi unanimity of agents have agreed (these attacks are not questioned anymore), whereas the validity of attacks in R^+ and R^- is still debated. For convenience, we will denote $At = R \cup R^+$ the set of attacks which are either on the system, or can be added to it. Note that we will only consider systems having a finite number of arguments, so $|A|$ is finite.

From now on, we will focus on the attack relations more than on the arguments. We will then need the following definition.

Definition 2. *Let* $AS = \langle A, R, R^+, R^- \rangle$*, and* $x = (a, b) \in At$*. We refer to the argument* a *as the* **tail of the attack** x*, denoted by* $tail(x) = a$*, and we refer to the argument* b *as the* **head of the attack** x*, denoted by* $head(x) = b$*.*
Let $x, y \in At$*. We will say that* x **hits** y*, denoted by* $hits(x, y)$*, if* $head(x) = tail(y)$*.*

Example 1. Let $AS = \langle A, R, R^+, R^- \rangle$ be an argumentation system such that $A = \{a,b,c,d,e\}$, $R = \{(a,b),(b,a),(c,d),(e,d)\}$, $R^+ = \{(a,c),(b,c)\}$, $R^- = \{(c,d),(e,d)\}$. This system can be represented as follows:



Non-removable attacks are represented by thick arrows, removable attacks by simple arrows, and addable attacks by dotted arrows. For the sake of convenience, we will denote an attack $(a,b) \in At$ simply as ab. We have that $tail(ba) = b$, $head(ba) = a$ and also $hits(ab,ba)$, as well as $hits(ba,ab)$.

In Dung's framework, the *acceptability of an argument* depends on its membership to some sets, called extensions.

Definition 3. *Let $AS = \langle A, R, R^+, R^- \rangle$ and $C \subseteq A$. The set C is* **conflict-free** *iff $\nexists x \in R$ such that $tail(x) \in C$ and $head(x) \in C$. An argument $a \in A$ is* **acceptable w.r.t.** *C iff $\forall x \in R$: if $head(x) = a$, then $\exists y \in R$ such that $hits(y,x)$ and $tail(y) \in C$.*

Several types of extensions have been defined by Dung [18].

Definition 4. *Let $C \subseteq A$ be conflict-free. C is an* **admissible extension** *iff each argument of C is acceptable w.r.t. C. C is a* **preferred extension** *iff it is a maximal (w.r.t. $\subseteq$) admissible extension. C is a* **complete extension** *iff every argument in C is acceptable w.r.t. C, and $\forall x \in A$: if x is acceptable w.r.t. C, then $x \in C$. C is a* **grounded extension** *iff it is the minimal (w.r.t. $\subseteq$) complete extension. Admissible, preferred, complete and grounded semantics are from now on denoted Adm, Pref, Comp and Gr, respectively.*

The next question is to decide, given a semantics, which arguments are acceptable.

Definition 5. *Let $AS = \langle A, R, R^+, R^- \rangle$ and $a \in A$. Argument a is said* **credulously accepted** *w.r.t. system AS under semantics $S \in \{Adm, Pref, Comp, Gr\}$, denoted $S_\exists(a,AS)$, iff a belongs to at least one extension of AS under the S semantics. Argument a is said* **sceptically accepted** *w.r.t. AS under semantics $S \in \{Adm, Pref, Comp, Gr\}$, denoted $S_\forall(a,AS)$, iff a belongs to all the extensions of AS under the S semantics.*

As $\{\}$ is always an admissible extension, $Adm_\forall(a,AS)$ does not hold for any $a \in A$. So, sceptical acceptability under admissible semantics is not an interesting notion, and we will not refer to it anymore. As there always exists a unique grounded extension, there is no difference between credulous and sceptical acceptability for grounded semantics. If $a \in A$ is accepted under the grounded semantics, we simply denote this by $Gr(a,AS)$. Moreover, as stated in [18], an argument $a \in A$ belongs to the grounded extension if and only if it is sceptically accepted under the complete semantics (thus, $Gr(a,AS) \Leftrightarrow Comp_\forall(a,AS)$). We will use this latter notation to refer to the grounded extension.

In the rest of the paper, we will denote by $Sem = \{Adm, Pref, Comp\}$ the set of admissible, preferred and complete semantics. Moreover, for the sake of readability, if there is no danger of confusing which argumentation system we refer to, we will simply write $\forall S \in Sem$, $S_\exists(a)$, or $S_\forall(a)$, without mentioning the AS.

The following property states that the set of arguments credulously accepted under the admissible semantics are the same as those accepted under the preferred, or the complete semantics.

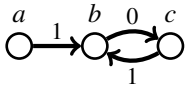
Property 1. [18] Let $AS = \langle A, R, R^+, R^- \rangle$ be an argumentation system, and $a \in A$. It holds that $Adm_{\exists}(a, AS) \Leftrightarrow Pref_{\exists}(a, AS) \Leftrightarrow Comp_{\exists}(a, AS)$.

The case of sceptical acceptability is a bit different. Every argument sceptically accepted under complete semantics is also sceptically accepted under preferred semantics, but the inverse does not hold in the general case.

Property 2. [18] Let $AS = \langle A, R, R^+, R^- \rangle$ be an argumentation system, and $a \in A$. It holds that $Comp_{\forall}(a, AS) \Rightarrow Pref_{\forall}(a, AS)$. The inverse does not necessarily hold.

As we have just seen, Dung's semantics [18] are stated in terms of sets of arguments, but it is also possible to express them using argument *labeling* [19, 20]. Roughly, an argument is *in* if all its attackers are *out*, it is *out* if it has at least an attacker *in*, otherwise it is *undec*. Villata et al. [9] introduce *attack semantics* where arguments are accepted when there are no *successful* attacks on them. An attack x is '1' when $tail(x)$ is *in*, '?' when $tail(x)$ is *undec*, and '0' when $tail(x)$ is *out*. An attack is called *successful* when it is '1' or '?', and *unsuccessful* when it is '0'.

Example 2. Let $AS = \langle A, R, R^+, R^- \rangle$ be an argumentation system, with $A = \{a, b, c\}$, $R = \{(a, b), (b, c), (c, b)\}$, $R^+ = R^- = \{\}$.



In argument semantics, an extension for a semantics $S \in Sem$ contains a and c. Thus, b is rejected (*out*) whereas a and c are accepted (*in*). In attack semantics, the attacks $(a, b) \in R$ and $(c, b) \in R$ are successful, whereas $(b, c) \in R$ is unsuccessful.

Boella et al. [7] propose a new kind of labelling, called *conditional labelling*. The idea is to provide the agents with a way to discover the arguments they should attack to get a particular argument accepted or rejected. Given a conditional labelling, the agents have complete knowledge about the consequences of the attacks they may raise on the acceptability of each argument without having to recompute the labelling for each possible set of attacks they may raise.

3 Argumentative goals and target sets

In this work, we consider that attacks are the core components of an argumentation system and thus prefer to commit to the attack semantics. As said before, we assume that the arguments of a system cannot change (neither new arguments can be added, nor arguments can be removed). Instead, we consider that the only change that can happen is the addition of new attacks and the removal of some attacks already in the system. A central notion, related to this type of change, is the following notion of atom.

Definition 6. *Let $AS = \langle A, R, R^+, R^- \rangle$ be an argumentation system, $x \in At = R \cup R^+$ be an attack, and $d \in A$ be an argument. An* **atom** *of AS is defined as follows:*

$$Atom(AS) ::= \top \mid \bot \mid (x,+,\#) \mid (x,-,\#) \mid (x,1,\#) \mid (x,0,\#) \mid (x,?,\#) \mid (x,1,*) \mid (x,0,*) \mid (x,?,**) \mid (x,?,*) \mid PRO(d) \mid CON(d)$$

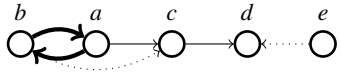
Atoms $\top$, $\bot$, $(x,+,\#)$, $(x,-,\#)$, $(x,1,\#)$, $(x,0,\#)$ *and* $(x,?,\#)$ *are called* **closed atoms**, *whereas atoms* $(x,1,*)$, $(x,0,*)$, $(x,?,**)$, $(x,?,*)$, $PRO(d)$ *and* $CON(d)$ *are called* **open atoms**.

The atom $(x,+,\#)$ (resp. $(x,-,\#)$) indicates the action of adding (resp. removing) the attack x from the system. The atom $(x,1,*)$ (resp. $(x,?,*)$, resp. $(x,0,*)$) indicates that we must find a way for attack x to become '1' (resp. '?', resp. '0'). [6] On the other hand, the atom $(x,1,\#)$ (resp. $(x,?,\#)$, resp. $(x,0,\#)$), indicates that we have already found a way for attack x to become '1' (resp. '?', resp. '0'). $PRO(d)$ and $CON(d)$ are two specific atoms regarding the acceptability status of d. Their exact meaning will be explained later. Finally, the atom $\bot$ indicates failure, whereas $\top$ indicates success.

By using the atoms $(x,+,\#)$ and $(x,-,\#)$, we define the notion of move on a system:

Definition 7. *Let* $AS = \langle A,R,R^+,R^-\rangle$ *and* $m = \{(x,s,\#) \mid x \in At,\ s \in \{+,-\}\}$ *be a set of atoms. m is called* **move on** *AS iff* $\forall (x,+,\#) \in m,\ x \in R^+$, *and* $\forall (x,-,\#) \in m,\ x \in R^-$. *The* **resulting system of playing move** *m* **on** *AS is the argumentation system* $\Delta(AS,m) = \langle A,R_m,R_m^+,R_m^-\rangle$, *such that: (1)* $x \in R_m$ *iff either* $x \in R$ *and* $(x,-,\#) \notin m$, *or* $(x,+,\#) \in m$. *(2)* $x \in R_m^+$ *iff either* $x \in R^+$ *and* $(x,+,\#) \notin m$, *or* $(x,-,\#) \in m$. *(3)* $x \in R_m^-$ *iff either* $x \in R^-$ *and* $(x,-,\#) \notin m$, *or* $(x,+,\#) \in m$.

Example 1, cont. The move $m = \{(ed,-,\#),(ac,+,\#)\}$ on AS will lead to the following system $\Delta(AS,m)$:



If we are able to play a move on $AS = \langle A,R,R^+,R^-\rangle$, we may be motivated to play it by the desire to satisfy a specific goal. Let us formally define this notion of goal.

Definition 8. *Let Systems be a set of argumentation systems, and Props be a set of properties, such that each property can refer to any* $AS \in Systems$. *We define the function* f*: Props* $\times$ *Systems* $\rightarrow \{true, false\}$, *such that* $\forall P \in Props, \forall AS \in Systems$, *it holds that* $f(P,AS) = true$ *iff P, when referring to AS, holds; otherwise* $f(P,AS) = false$.
A property P may be chosen as a **positive goal**: *we say that goal P is satisfied in AS iff* $f(P,AS) = true$. *A negated property* $\neg P$ *may be chosen as a* **negative goal**: *we say that goal* $\neg P$ *is satisfied in AS iff* $f(P,AS) = false$ *(that is iff* $f(\neg P,AS) = true$*).*

If a specific (positive or negative) goal is not satisfied in AS, then we search for possible moves m on AS leading to a modified system $\Delta(AS,m)$ in which that goal is satisfied. Any move on AS which achieves this is called a *successful move*. Such a succesful move is called a *target set* if the changes induced by it on AS are minimal.

Definition 9. *Let* $AS = \langle A,R,R^+,R^-\rangle$, *and Props be a set of properties. Let m be a move on AS,* $P \in Props$, *and g be a goal, that is P or* $\neg P$. *m is called* **successful move for goal** *g iff goal g is satisfied in* $\Delta(AS,m)$, *that is if* $f(g,\Delta(AS,m)) = true$. *m is called* **target set for goal** *g iff m is minimal w.r.t.* $\subseteq$ *among all the successful moves for g.*

[6] The atom $(x,?,**)$ is similar to $(x,?,*)$, their difference is explained later.

Let us now describe the types of goals that we focus on. Let $AS = \langle A, R, R^+, R^- \rangle$, m be a move on AS, $X \in \{\exists, \forall\}$ and $S \in Sem$. We focus on the acceptance of a single argument $d \in A$ called the *issue*, and we consider these two types of goals: (1) $S_X(d)$ is a positive goal, with $\mathbb{M}^S_X = \{m \mid S_X(d) \text{ is satisfied in } \Delta(AS, m)\}$. (2) $\neg S_X(d)$ is a negative goal, with $\mathbb{M}^S_{\neg X} = \{m \mid \neg S_X(d) \text{ is satisfied in } \Delta(AS, m)\}$.

Example 1, cont. Let $d \in A$ be the issue.
d does not belong to any admissible extension of AS. The goal $S_\exists(d)$ consisting in placing d in some admissible (or preferred, or complete) extension has three target sets: $\mathbb{T}^S_\exists = \{\{(ed,-,\#),(cd,-,\#)\},\{(ed,-,\#),(ac,+,\#)\},\{(ed,-,\#),(bc,+,\#)\}\}$.
Moreover, we have $\{(ed,-,\#),(bc,+,\#),(ac,+,\#)\} \in \mathbb{M}^S_\exists$: this move is successful for $S_\exists(d)$, but it is not a target set, as it is not minimal. Now, regarding sceptical preferred semantics, it holds that $\mathbb{T}^{Pref}_\forall = \{\{(ed,-,\#),(cd,-,\#)\},\{(ed,-,\#),(bc,+,\#),(ac,+,\#)\}\}$.
Finally, as far as grounded semantics is concerned, $\mathbb{T}^{Comp}_\forall = \{\{(ed,-,\#),(cd,-,\#)\}\}$.

We now provide some properties of succesful moves and of target sets.

Property 3. It holds that

$$\mathbb{M}^{Comp}_\forall \subseteq \mathbb{M}^{Pref}_\forall \subseteq \mathbb{M}^S_\exists \text{ and } \mathbb{M}^S_{\neg\exists} \subseteq \mathbb{M}^{Pref}_{\neg\forall} \subseteq \mathbb{M}^{Comp}_{\neg\forall}$$

Proof. Let us begin with the case of the positive goals. If move $m \in \mathbb{M}^{Comp}_\forall$, then d is accepted in $AS' = \Delta(AS, m)$ under complete semantics (using sceptical acceptability), so d belongs in all the complete extensions of AS', therefore in all the preferred extensions of AS'. So, it holds that $m \in \mathbb{M}^{Pref}_\forall$. Thus, we have proved that $\mathbb{M}^{Comp}_\forall \subseteq \mathbb{M}^{Pref}_\forall$. Moreover, if $m \in \mathbb{M}^{Pref}_\forall$, then d belongs in all the preferred extensions of AS', therefore d belongs in at least one preferred extension of AS' (so, it also belongs in at least one admissible, and in at least one complete extension of AS'). Thus, it holds that $m \in \mathbb{M}^S_\exists$, and we have proved that $\mathbb{M}^{Pref}_\forall \subseteq \mathbb{M}^S_\exists$. As a result, $\mathbb{M}^{Comp}_\forall \subseteq \mathbb{M}^{Pref}_\forall \subseteq \mathbb{M}^S_\exists$. The proof is similar in the case of negative goals. It is omitted due to the lack of space.

Property 4. If m is a move such that $m \in \mathbb{T}^{Comp}_\forall$ and $m \in \mathbb{T}^S_\exists$, then $m \in \mathbb{T}^{Pref}_\forall$ (1)
Moreover, if m is a move such that $m \in \mathbb{T}^S_{\neg\exists}$ and $m \in \mathbb{T}^{Comp}_{\neg\forall}$, then $m \in \mathbb{T}^{Pref}_{\neg\forall}$ (2)

Proof. (1) By contradiction, let $m \in \mathbb{T}^{Comp}_\forall$, $m \in \mathbb{T}^S_\exists$ and assume that $m \notin \mathbb{T}^{Pref}_\forall$. Now, $m \in \mathbb{T}^{Comp}_\forall$ implies that $m \in \mathbb{M}^{Comp}_\forall$ (as m is minimal w.r.t. $\subseteq$ among the moves in $\mathbb{M}^{Comp}_\forall$). Then, from $m \in \mathbb{M}^{Comp}_\forall$ it follows that $m \in \mathbb{M}^{Pref}_\forall$ (from Property 3). Moreover, we assumed that $m \notin \mathbb{T}^{Pref}_\forall$, so there must exist another move $m' \subset m$, such that $m' \in \mathbb{T}^{Pref}_\forall$ (and, of course, $m' \in \mathbb{M}^{Pref}_\forall$). From $m' \in \mathbb{M}^{Pref}_\forall$, we get that $m' \in \mathbb{M}^S_\exists$ (from Property 3). Finally, from $m' \in \mathbb{M}^S_\exists$ and $m \in \mathbb{T}^S_\exists$, it follows that $m \subseteq m'$. Contradiction, since above we had $m' \subset m$. Therefore, $m \in \mathbb{T}^{Pref}_\forall$.
(2) Similar proof for the case of negative goals. It is omitted due to the lack of space.

Figure 1 graphically represents the links between the set of succesful moves and the target sets for the positive and the negative goals. The meaning of the sets $\mathcal{M}^{PRO}$ and $\mathcal{M}^{CON}$ will be explained in Section 4.

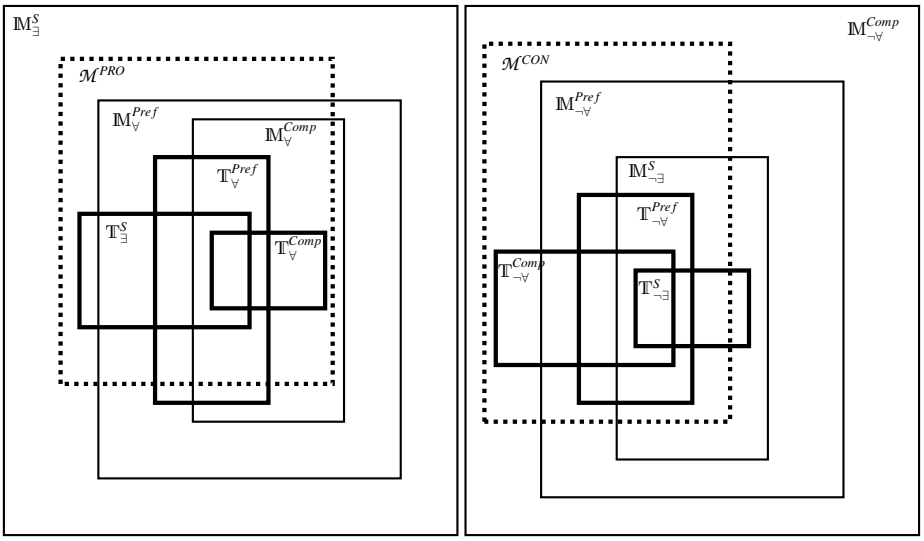


Fig. 1. On the left: The sets of successful moves and target sets for the positive goals, and $\mathcal{M}^{PRO}$. On the right: The sets of successful moves and target sets for the negative goals, and $\mathcal{M}^{CON}$.

Having highlighted some properties of the successful moves and of the target sets, we define in the following section our rewriting procedure which computes target sets.

4 Computing Target Sets and Successful Moves

In this section we provide a set of rewriting rules which help us to compute, for any system $AS = \langle A, R, R^+, R^- \rangle$, all the target sets for some types of goals. In order to do this, we have used the Maude [7] system [8] which is based on rewriting logic. This section is arranged as follows: we start by explaining what Maude is and why it is useful for the type of computations we want to make. Then, we analyze the core component of our program, its set of rules. Afterwards, we explain the rewriting procedure of Maude, in the context of our program. Finally, we prove some important properties.

4.1 The Maude rewriting system and the intuition behind our program

Maude is both a declarative programming language and a system. It is based on rewriting logic and it can model systems and the actions within those systems. Maude is a high-level, expressive language, which can model from biological systems to programming languages, including itself. A program in Maude is a logical theory, and a computation made by that program is logical deduction using the axioms of the theory.

Our Maude program, presented in Appendix A, is given as input a term which describes an argumentation system $AS = \langle A, R, R^+, R^- \rangle$ and contains either $PRO(d)$ or

[7] http://maude.cs.uiuc.edu

$CON(d)$, with $d \in A$. If we want to ensure the (positive) goal of accepting argument d under some semantics, we start with $PRO(d)$. Otherwise, if we want to ensure the (negative) goal of rejecting d, we start with $CON(d)$. Maude starts from these atoms and, based on a set of rewriting rules and equations, rewrites the initial term, thus producing new terms, which are, in turn, rewritten. The system stops when all the computed terms are non-rewritable. We will see that every term of the output corresponds to a move on the initial system AS. Their connection with the status of d is detailed in Property 6.

4.2 The Rewriting Rules

Before explaining the rules of our program, we must provide two more basic definitions. The notion of atom is central in what follows. The connectors $\wedge$ and $\vee$ are used in order to link atoms, forming conjuncts and formulas.

Definition 10.
$Conjunct ::= Atom \mid (Conjunct \wedge Conjunct)$;
$Formula ::= Conjunct \mid (Formula \vee Formula)$
*Let $Conjuncts$ denote the set of all possible conjuncts, and let $Formulas$ denote the set of all possible formulas. A conjunct which contains at least one open atom is called **open conjunct**. Otherwise, it is called **closed conjunct**. A formula which contains at least one open conjunct is called **open formula**. Otherwise, it is called **closed formula**.*

We now proceed to the analysis of the program's rules. There exist two types of rules: **Atom expansions**, or **rewriting rules**, indicated by '=>', and **atom simplifications**, or **equations**, indicated by '='. In our program, an atom expansion replaces two atoms appearing in an open conjunct by some other atoms, whereas an atom simplification replaces two atoms found in the same conjunct by a single atom.

Let us briefly explain the intuition behind the expansion rules. Depending on whether we want to accept or reject the issue, we start from it and we navigate the attacks backwards, while adding and removing attacks, trying to enforce the status of the attacks relevant to the issue. When there exist more than one choice to achieve our goal, we try to explore all the possibilities (combinations of additions and removals). Very roughly, if at some point of the computation, the left side of an expansion rule appears, Maude replaces it with the right side of that rule. The same principle holds for equations.

So, when the initial goal is $PRO(d)$, we want to see the issue d accepted. To do so, we have to take each attack against d, one at the time, and either remove it (if it belongs to R^-), or make it '0' by making an attack which attacks it become '1'. On the other hand, when the initial goal is $CON(d)$, and we want to see d rejected, we have to either make one attack against d become '1', add such an attack if it is in R^+ (and ensure its succesfulness), or to make one attack against d become '?'. Let us see the rules in more detail:

Rules 1-3 say that if an attack is '1', then for every attack against it, either that attack is '0' (rule 1), or it is removed (rule 2), or (if it belongs to R^+) we introduce an atom $(x,0,\#)$ which will lead to a simplification if we later add this attack (rule 3), thus it can never become successful. Rules 4-5 say that if an attack is '0', then there exists an attack against it which is '1'. That attack is either already in the system (rule 4), or it is added

to it (rule 5). Rules 6-12 say that if an attack is '?', then two things hold: first, there exists at least one attack against it which is also '?' (rules 6 and 7). [8] Also, the rest of the attacks set against it are either '?', or '0', or removed (rules 8-10), or (if they belong to R^+) we introduce $(x,0,\#)$ and $(x,?,\#)$, which will lead to simplifications if we later add these attack and try to make them '1' (rules 11-12). Rules 13-15 say that in the PRO case every attack against the issue is either '0' (rule 13), is removed (rule 14), or (if it belongs to R^+) we introduce an atom $(x,0,\#)$ for the same reason as explained above (rule 15). Rules 16-19, finally, say that in the CON case there exists one attack against the issue which is either '1' (rules 16 and 17) or '?' (rules 18 and 19).

Now, as far as the simplification rules (equations) are concerned: Equation 1 says that if two identical atoms appear in the same conjunct, then one of them is deleted. Equation 2 performs a simplification related to the '?' status of an attack. Equation 3 says that if an open atom and a closed atom (which are otherwise identical) appear in the same conjuct, then the open atom is deleted. Equations 4-6 say that if two atoms referring to the same attack, but indicating different status, appear in the same conjunct, then $\bot$ is introduced. Equations 7-8 say that if an attack which cannot be attacked is set to be '?' or '0', then $\bot$ is introduced. Equations 9-10 are applied in case there exist no potential attacks against d. Equation 11, finally, says that the atom $\bot$ once it appears in a conjunct, it reduces that conjunct into $\bot$.

Also, notice the **and** operator in the program (corresponding to the $\wedge$ sign) which is declared as associative and commutative. This makes the firing of expansion and simplification rules easy, regardless of the position of the atoms in a conjunct.

Finally, we explain how an argumentation system is represented and passed as input to our program. We define the attacks of the system by using the following conventions. The name of an attack must be preceded by '. If attack $x \in R^+$ (resp. $x \in R^-$) then its name starts with '$+$' (resp. '$-$'). Also, by using the *hits*, *isNotHit* and *hitsArg* operators, we define how the attacks are related to each other (and to the issue). For example ('-cd hitsArg d) means that $head(-cd) = d$. Moreover, ('+bc hits '-cd) means that $hits(+bc,-cd)$. Finally, (isNotHit '-ed) means there is no attack against the attack -ed.

4.3 The Rewriting Procedure (*RP*)

Now we explain how the rewriting procedure of Maude works, not in general, but in the specific case of our program. Informally, its input is an argumentation system *AS*, either the atom $PRO(d)$ or the atom $CON(d)$, a set of expansion rules and a set of simplification rules. The rewriting procedure starts from $PRO(d)$ or $CON(d)$. All the applicable expansion rules are considered, one-by-one. For every applicable expansion rule, that rule is applied, and a set of new conjuncts is computed. In every new conjunct, simplification rules are applied repeatedly, until no more simplification rules are applicable. Once an "expansion-simplification" step is finished, all the conjuncts computed in the previous step are considered (one by one) and there follows another "expansion-simplification" step. These steps are repeated until, at some point, there are no conjuncts which can

[8] Note that if we only had atoms of the type $(x,?,*)$, but not of the type $(x,?,**)$, there would exist a possible rewriting making *all* the the attacks against x become '0' (for example), thus not achieving to make x become '?'.

Data: A system $AS = \langle A, R, R^+, R^- \rangle$, $initF = PRO(d)$ or $initF = CON(d)$, with $d \in A$, a set of expansion rules, a set of simplification rules.
Result: A set of moves $\mathcal{M}_d$.
Initialise formula $currF := initF$;
while *currF has an expandable conjunct* **do**
  Let Exp denote the set of all the expandable conjuncts of $currF$;
  foreach *conjunct* $C \in Exp$ **do**
    Initialise the set of conjuncts $repl_C := \{\}$;
    foreach *applicable rewriting rule rl on* C **do**
      if *rule rl applied on* C *gives* C' **then**
        while *a simplification can be applied on* C' **do**
          Choose such a simplification, and apply it on C' ;
        Add C' into the set $repl_C$;
    Replace C with $C'_1 \vee C'_2 \vee \cdots \vee C'_m$ in $currF$, s.t. $\forall i \in [1 \ldots m]$, $C'_i \in repl_C$;
Initialise the set of moves $\mathcal{M}_d := \{\}$;
foreach *conjunct* C *of* $currF$ **do**
  if $C \neq \bot$ **then**
    $m := \{(x,s,\#) \mid (x,s,\#)$ appears in C, and $s \in \{+,-\}\}$; Add m into the set $\mathcal{M}_d$;
return $\mathcal{M}_d$;

Algorithm 1: Maude's rewriting procedure, in the context of our program

be further expanded. Finally, from every non-expandable conjunct computed, just the $(x,+,\#)$ and $(x,-,\#)$ atoms are filtered. The formal definition of Maude's rewriting procedure, in the context of our program, is given in Algorithm 1.

In the rest of the paper, the set of returned moves will be denoted $\mathcal{M}_d^{PRO}$ if $initF = PRO(d)$, and $\mathcal{M}_d^{CON}$ if $initF = CON(d)$.

Example 1, cont. In order to represent the system AS of this example, we must run the Maude program with the following input:

```
> search PRO(d) and ('-cd hitsArg d) and ('-ed hitsArg d) and ('+bc hits '-cd)
and ('+ac hits '-cd) and ('ba hits '+ac) and ('ba hits 'ab) and ('ab hits '+bc)
and ('ab hits 'ba) and (isNotHit '-ed) =>! C:Conjunct .
```

Two important remarks: first, the "search" keyword tells Maude that whenever more than one rewriting rules are applicable, it must consider them all, one at a time, in a Breadth-First-Search way. This is essential in order to find all the possible rewritings. Second, by using `=>! C:Conjunct`, we tell Maude to continue the rewritings, until the obtained terms are non-rewritable conjuncts.

Once this computation finishes, we obtain three conjuncts which correspond to moves on AS, as well as a fourth conjunct $\bot$. The moves corresponding to the three conjuncts are: $\mathcal{M}_d^{PRO} = \{\{(ed,-,\#),(cd,-,\#)\}, \{(ed,-,\#),(bc,+,\#)\}, \{(ed,-,\#),(ac,+,\#)\}\}$.

Now, let us highlight some properties of the *RP* procedure.

Property 5. The procedure *RP* always terminates.

Proof. *RP* starts with a conjunct containing $PRO(d)$ or $CON(d)$. It finds all the applicable expansion rules, therefore it computes a number of new conjuncts. We can see the

initial conjunct as the root of a tree and the new conjuncts as the children of the root. Gradually, *RP* will compute a tree whose nodes are conjuncts. We will prove that this tree has obligatorily a finite number of nodes. First, from the expansion rules it follows that every conjunct computed by *RP* has a finite number of atoms. Moreover, there is a finite number of applicable rules on every conjunct, so the branching factor of the tree is finite. Finally, we must prove that the depth of the tree is finite. From the set of rewriting rules, it follows that a conjunct will be expandable (that is not a leaf node), if it contains an open atom and an atom of the form (x hits y), or of the form (x hitsArg d). [9] Notice that every conjunct contains a finite number of (x hits y) and (x hitsArg d) atoms, because the number of arguments and attacks of *AS* is finite. Also, after the application of any expansion rule, the newly created conjunct contains one less (x hits y) or (x hitsArg d) atom than its parent-node. As a result, the depth of the tree cannot be greater than the initial number of (x hits y) and (x hitsArg d) atoms, which is finite. So, we have proved that *RP* always terminates.

At this point, we underline that the "search" keyword ensures that, after a simplification step, Maude tries every applicable rewriting rule. Therefore, the order in which the rules are checked (Maude uses an internal strategy to decide on the order) does not affect the results.

We now analyze the output of the rewriting procedure w.r.t. the different argumentative goals. We shall say that: (1) the procedure is *correct* for *successful moves* (resp. *target sets*) for goal g if every move it returns is successful (resp. a target set) for g ;
(2) the procedure is *complete* for *successful moves* (resp. *target sets*) for goal g if it returns *all* the successful moves (resp. the target sets) for g.

As shown by Figure 1, correctness for target sets is not satisfied: the procedure returns, for *PRO* or *CON*, some moves that are not target sets for any of the semantics. But in some cases we can ensure that the procedure is correct for successful moves—in that case moves only fail on the minimality criterion. In the same way, the completeness for successful moves is not satisfied: *RP* does not give all the successful moves for any semantics (in the general case). However, completeness for target sets can be obtained in some cases. Of course, the most interesting lines are those for which we have "Yes" in both columns: only successful moves are returned, and all the target sets are.

Property 6. The following table illustrates for which goals the rewriting procedure is correct for successful moves and/or complete for target sets.

Goal	*Correctness for successful moves*	*Completeness for target sets*
$S_{\exists}(d)$	Yes	Yes
$Pref_{\forall}(d)$	No	No
$Comp_{\forall}(d)$	No	Yes
$\neg S_{\exists}(d)$	No	No
$\neg Pref_{\forall}(d)$	No	?
$\neg Comp_{\forall}(d)$	Yes	Yes

[9] This means that it is quite possible for an open atom to be non-expandable. This is the case when no relevant (x hits y) or (x hitsArg d) atom is found in the same conjunct as the open atom.

Note that the completeness regarding the goal $\neg Pref_{\forall}(d)$ is left open so far. However, for the sake of readability, we draw Figure 1 assuming that the answer is "Yes".

Proof. There are counter-examples for the "No" entries of the table, omitted due to the lack of space. As far as the "Yes" cases are concerned, we only provide the proofs of completeness and correctness for $S_{\exists}(d)$.

Correctness of *RP* for $S_{\exists}(d)$: That is $\mathcal{M}_d^{PRO} \subseteq \mathbb{M}_{\exists}^S$. Let $m \in \mathcal{M}_d^{PRO}$. The move m correponds to some conjunct, denoted c_m, computed by *RP*. From c_m we can construct the set of arguments $D = \{x \mid (xy,1,s) \text{ is an atom of } c_m\}$. We will now prove that in $\Delta(AS,m) = \langle A, R_m, R_m^+, R_m^- \rangle$, it holds that D is an admissible set of arguments which defends argument d. First, let us assume that in $\Delta(AS,m)$ the set D is not conflict-free. In that case there exist two arguments $x_1, x_2 \in D$, such that $x_1x_2 \in R_m$. Now, $x_1, x_2 \in D$ implies that $\exists x_3, x_4 \in A$ such that $(x_1x_3,1,s)$ and $(x_2x_4,1,s)$ are atoms of c_m. Given that $(x_2x_4,1,s)$ appears in c_m, and that $x_1x_2 \in R_m$, it follows that atom $(x_1x_2,0,s)$ must also appear in c_m (from expansion rule 1). In turn, this means that $\exists x_5 \in A$ such that $(x_5x_1,1,s)$ also appears in c_m (from expansion rules 4,5). Similarly, given that $(x_1x_3,1,s)$ appears in c_m, it holds that $(x_5x_1,0,s)$ also appears in c_m. But, it is impossible for both $(x_5x_1,1,s)$ and $(x_5x_1,0,s)$ to appear in the same conjunct (as they would have been simplified into $\bot$). Therefore, we have proved that D is conflict-free. Second, let us assume that in the system $\Delta(AS,m)$, the set D does not defend all its elements. In that case $\exists x_1 \in D$ and $\exists x_2 \notin D$ such that $x_2x_1 \in R_m$, and no argument of D attacks x_2. $x_1 \in D$ implies that $\exists x_0 \in A$ such that atom $(x_1x_0,1,s)$ appears in c_m. So, it follows that atom $(x_2x_1,0,s)$ also appears in c_m (from expansion rule 1), and as a result, $\exists x_3 \in A$ such that atom $(x_3x_2,1,s)$ also appears in c_m. By definition of the set D, notice that $x_3 \in D$. Impossible, since we assumed that no argument of D attacks x_2 in $\Delta(AS,m)$. Therefore, we have proved that D defends all its elements. Given that D is conflict-free and it defends all its elements, it follows that D is an admissible set of arguments. Finally, since for every attack $xd \in R_m$ against the issue d, it holds that atom $(xd,0,s)$ appears in c_m (because of expansion rule 13), it holds that argument d is defended by the set D. From this, and from the fact that D is admissible in $\Delta(AS,m)$, it follows that $D \cup \{d\}$ is admissible in $\Delta(AS,m)$. Thus, $m \in \mathbb{M}_{\exists}^S$, and we have proved that $\mathcal{M}_d^{PRO} \subseteq \mathbb{M}_{\exists}^S$.

Completeness of *RP* for $S_{\exists}(d)$ (sketch of proof): We want to prove that $\mathbb{T}_{\exists}^S \subseteq \mathcal{M}_d^{PRO}$. Let $t \in \mathbb{T}_{\exists}^S$. We will prove that *RP* constructs a tree which has a leaf node containing all the atoms of t, and no additional $(x,+,\#)$, or $(x,-,\#)$ atoms. Let the set $\{x_1,\ldots,x_n\}$ contains the arguments attacking d in AS. Let $P = \{(x_1d,-,\#),\ldots,(x_nd,-,\#)\}$. t contains a subset of atoms $P' \subseteq P$, and cannot contain any atoms of the form $(xd,+,\#)$. Moreover, it is not difficult to prove that the tree has a node n (not a leaf, in the general case) which contains all the atoms of P', and no other $(x,+,\#)$ or $(x,-,\#)$ atoms. Let $\{x_k,\ldots,x_l\} \subseteq \{x_1,\ldots,x_n\}$ denote the arguments whose attacks against d remain in $\Delta(AS,t)$. According to the expansion rules for $PRO(d)$, the node n also contains the atoms $(x_kd,0,*)$, …, $(x_ld,0,*)$. Thus t contains the atoms of P' and some additional atoms, resulting from the expansions of $(x_kd,0,*)$, …, $(x_ld,0,*)$, so it can be denoted $t = P' \cup Q$. Note that every atom of Q refers to an attack necessarily "connected" to an argument of $\{x_k,\ldots,x_l\}$. Let us focus on the attacks against d which are not removed. Those attacking arguments must get attacked back, in order for d to be reinstated. At this point, it is not difficult to prove that, for every argument $x_i \in \{x_k,\ldots,x_l\}$: (1) It is impos-

sible for any $(yx_i, -, \#)$ atom to appear in t. (2) It is impossible for two atoms $(y_1x_i, +, \#)$ and $(y_2x_i, +, \#)$ to appear in t. As a result, for every argument $x_i \in \{x_k, \ldots, x_l\}$, t can only contain 0 or 1 atoms of the type $(yx_i, +, \#)$. Now we must make sure that *RP* computes all these possible combinations of attack additions reinstating d. When *RP* expands the node n, it creates a node for every possible combination of attack additions reinstating d. Thus, there will be below the node n a number of nodes which contain either 0 or 1 atoms of the type $(yx_i, +, \#)$ for every $x_i \in \{x_k, \ldots, x_l\}$. One of these nodes will obligatorily contain exactly the atoms of t which indicate attack additions against the arguments $\{x_k, \ldots, x_l\}$. Moreover, if such a node contains atom $(yx_i, +, \#)$, then it also contains atom $(yx_i, 1, *)$, as the added attacks must be '1'. *RP* continues to search the graph backwards, considering the indirect attackers (and defenders) of d, using the expansion rules for the $(yx_i, 1, *)$ atoms. Therefore, after a finite number of expansions, the procedure will compute a node which contains exactly the $(x, +, \#)$ and $(x, -, \#)$ atoms found in t. This last statement is true only if the simplification rules which produce $\perp$ cannot lead to the "loss of a target set". Two simplification rules can introduce $\perp$ here: the first one says that if there is a node n containing $(xy, 0, *)$, and no potential attacker of x in the system, $\perp$ is introduced. Having $(xy, 0, *)$ in n means that all the target sets found in the subtree below n must lead to a modified system where there is an attack against x. Since x has no potential attackers this can never happen. The second rule says that if node n contains both $(xy, 0, s)$ and $(xy, 1, s)$, $\perp$ is introduced. Let n a node containing both $(xy, 0, s)$ and $(xy, 1, s)$. Every eventual target set found in the subtree below n leads to a modified system in which some admissibe extension: (a) attacks the argument x (because of $(xy, 0, s)$), and (b) contains argument x (because of $(xy, 1, s)$). This is impossible.

5 Conclusion

The dynamics of argumentation systems is a central and compelling notion to address when debates are to be considered among users or agents. However, the task of computing which move to make in order to reach a given argumentative goal is difficult. In this paper we focus on complex simultaneous moves involving addition and retraction of attacks. We first proved a number of results related to the relation which holds among sets of successful moves and target sets. Then we described an approach based on a dedicated rewriting procedure within the Maude system, and proposed rules inspired from the attack semantics [9]. This approach provides the advantage of being relatively easy to design and interpret. This is an important feature if we consider a context where such moves are suggested to a user, since for instance traces can provide human-readable explanations of the result of the procedure. We then presented a number of results regarding the procedure (together with our rules): regarding positive goals, it is correct for successful moves and complete for target sets for *any* credulous semantics; while it is complete for target sets for the complete semantics, regardless of the type of goal considered (we recall that grounded semantics are included as a special case).

As far as potential extensions of this work are concerned, there are a number of possibilities. First, the efficiency of our rewriting procedure requires further investigation. We also plan to make some modifications of the procedure, in order to be correct and/or complete for more semantics. At this point, we note that adding the possibility to

explicitly add/remove arguments from a system would require the definition of some additional rules, but it would not significantly change the procedure. Finally, we will study the use of target sets in protocols for multi-agent debates. We wish to analyze the properties of such protocols, as well as the possible strategic considerations of the agents.

References

1. Bonzon, E., Maudet, N.: On the outcomes of multiparty persuasion. In: Proc. of AAMAS'11. (2011) 47–54
2. Kontarinis, D., Bonzon, E., Maudet, N., Moraitis, P.: Picking the right expert to make a debate uncontroversial. In: proc. of COMMA'12. (2012) 486–497
3. Egilmez, S., Martins, J., Leite, J.: Extending social abstract argumentation with votes on attacks. In: proc. of TAFA'13. (2013) To appear.
4. Bench-capon, T.J.M., Doutre, S., Dunne, P.E.: Value-based argumentation frameworks. In: Artificial Intelligence. (2002) 444–453
5. Amgoud, L., Cayrol, C.: On the acceptability of arguments in preference-based argumentation. In: proc. of UAI'98. (1998) 1–7
6. Modgil, S.: Revisiting abstract argumentation frameworks. In: proc. of TAFA'13. (2013) To appear.
7. Boella, G., Gabbay, D., Perotti, A., van der Torre, L., Villata, S.: Conditional labelling for abstract argumentation. In: Proc. of TAFA'11. (2011)
8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: The maude system. In: RTA'99. (1999) 240–243
9. Villata, S., Boella, G., van der Torre, L.: Attack semantics for abstract argumentation. In Walsh, T., ed.: proc. of IJCAI'11, IJCAI/AAAI (2011) 406–413
10. Modgil, S., Caminada, M.: Proof theories and algorithms for abstract argumentation frameworks. In: Argumentation in Artificial Intelligence. Springer US (2009) 105–129
11. Cayrol, C., de Saint-Cyr, F.D., Lagasquie-Schiex, M.C.: Change in abstract argumentation frameworks: Adding an argument. J. Artificial Intelligence Research (JAIR) **38** (2010) 49–84
12. Baumann, R., Brewka, G.: Expanding argumentation frameworks: Enforcing and monotonicity results. In: COMMA. (2010) 75–86
13. Liao, B., Jin, L., Koons, R.C.: Dynamics of argumentation systems: A division-based method. Artificial Intelligence **175**(11) (2011) 1790 – 1814
14. Baumann, R.: What does it take to enforce an argument? minimal change in abstract argumentation. In: ECAI. (2012) 127–132
15. Coste-Marquis, S., Konieczny, S., Mailly, J.G., Marquis, P.: On the revision of argumentation systems: Minimal change of arguments status. In: proc. of TAFA'13. (2013) To appear.
16. Bisquert, P., Cayrol, C., de Saint-Cyr, F.D., Lagasquie-Schiex, M.C.: Characterizing change in abstract argumentation systems. Technical report, IRIT-UPS (2013)
17. Bisquert, P., Cayrol, C., de Saint-Cyr, F.D., Lagasquie-Schiex, M.C.: Changements guidés par les buts en argumentation : Cadre théorique et outil. In: MFI'13. (2013)
18. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-persons games. Artificial Intelligence **77** (1995) 321–357
19. Jakobovits, H., Vermeir, D.: Robust semantics for argumentation frameworks. J. Log. Comput. **9**(2) (1999) 215–261
20. Caminada, M.: On the issue of reinstatement in argumentation. In: JELIA'06. (2006) 111–123

A Maude's listing

```
mod RP_PROCEDURE is
protecting QID .
********************* SORTS AND SUBSORTS
sorts Attack Argument Sign Atom Conjunct .
subsort Atom < Conjunct . subsort Qid < Attack .
********************* CONSTANTS
ops top btm : -> Atom [ctor] .
ops + - 1 ? 0 * ** # : -> Sign [ctor] .
ops d : -> Argument [ctor] .
********************** VARIABLES
vars X Y : Attack .
vars S T : Sign .
var At : Atom .
********************* OPERATORS
op ___ : Attack Sign Sign -> Atom [ctor] .
op PRO_ : Argument -> Atom [ctor] .
op CON_ : Argument -> Atom [ctor] .
op _hits_ : Attack Attack -> Atom [ctor] .
op _hitsArg_ : Attack Argument -> Atom [ctor] .
op isNotHit_ : Attack -> Atom [ctor] .
op isNotHitArg_ : Argument -> Atom [ctor] .
op _and_ : Conjunct Conjunct -> Conjunct [ctor assoc comm] .
********************* EQUATIONS - SIMPLIFICATION RULES
eq (X S T) and (X S T) = (X S T) .   *** Eq. 1
eq (X S **) and (X S *) = (X S *) .  *** Eq. 2
eq (X S *) and (X S #) = (X S #) .  *** Eq. 3
eq (X 0 S) and (X 1 T) = btm .   *** Eq. 4
eq (X 0 S) and (X ? T) = btm .   *** Eq. 5
eq (X ? S) and (X 1 T) = btm .   *** Eq. 6
eq (X 0 *) and isNotHit(X) = btm .   *** Eq. 7
eq (X ? S) and isNotHit(X) = btm .  *** Eq. 8
eq PRO(d) and isNotHitArg(d) = top .  *** Eq. 9
eq CON(d) and isNotHitArg(d) = btm .  *** Eq. 10
eq At and btm = btm .     *** Eq. 11
********************* REWRITING RULES - EXPANSION RULES
--------- Expansion rules for (X 1 *) atoms (rules 1, 2 and 3) ---------
*** RULE 1: The attack Y is on the system.
crl [expand_X1*_with_Y0*] : (X 1 *) and (Y hits X) =>
(X 1 *) and (Y 0 *) if not (substr(string(Y),0,1) == "+") .
*** RULE 2: The attack Y is removable.
crl [expand_X1*_with_Y-#_Y0#] : (X 1 *) and (Y hits X) =>
(X 1 *) and (Y - #) and (Y 0 #) if (substr(string(Y),0,1) == "-") .
*** RULE 3: The attack Y is addable.
crl [expand_X1*_with_Y0#] : (X 1 *) and (Y hits X) =>
(X 1 *) and (Y 0 #) if (substr(string(Y),0,1) == "+") .
--------- Expansion rules for (X 0 *) atoms (rules 4 and 5) ---------
*** RULE 4: The attack Y is on the system.
```

```
crl [expand_X0*_with_Y1*] : (X 0 *) and (Y hits X) =>
(X 0 #) and (Y 1 *) if not (substr(string(Y),0,1) == "+") .
*** RULE 5: The attack Y is addable.
crl [expand_X0*_with_Y+#_Y1*] : (X 0 *) and (Y hits X) =>
(X 0 #) and (Y + #) and (Y 1 *) if (substr(string(Y),0,1) == "+") .
--------- Expansion rules for (X ? **),(X ? *) atoms (rules 6-12) ---------
*** RULE 6: Sign **, the attack Y is on the system.
crl [expand_X?**_with_Y?**] : (X ? **) and (Y hits X) =>
(X ? *) and (Y ? **) if not (substr(string(Y),0,1) == "+") .
*** RULE 7: Sign **, the attack Y is addable.
crl [expand_X?**_with_Y+#_Y?**] : (X ? **) and (Y hits X) =>
(X ? *) and (Y + #) and (Y ? **) if (substr(string(Y),0,1) == "+") .
*** RULE 8: Sign *, the attack Y is on the system.
crl [expand_X?*_with_Y?**] : (X ? *) and (Y hits X) =>
(X ? *) and (Y ? **) if not (substr(string(Y),0,1) == "+") .
*** RULE 9: Sign *, the attack Y is on the system.
crl [expand_X?*_with_Y0*] : (X ? *) and (Y hits X) =>
(X ? *) and (Y 0 *) if not (substr(string(Y),0,1) == "+") .
*** RULE 10: Sign *, the attack Y is removable.
crl [expand_X?*_with_Y-#_Y0#] : (X ? *) and (Y hits X) =>
(X ? *) and (Y - #) and (Y 0 #) if (substr(string(Y),0,1) == "-") .
*** RULE 11: Sign *, the attack Y is addable.
crl [expand_X?*_with_Y0#] : (X ? *) and (Y hits X) =>
(X ? *) and (Y 0 #) if (substr(string(Y),0,1) == "+") .
*** RULE 12: Sign *, the attack Y is addable.
crl [expand_X?*_with_Y?#] : (X ? *) and (Y hits X) =>
(X ? *) and (Y ? #) if (substr(string(Y),0,1) == "+") .
--------- Expansion rules for PRO, CON atoms (rules 13-19) ---------
*** RULE 13: PRO, and the attack Y is on the system.
crl [expand_PRO_with_Y0*] : PRO(d) and (Y hitsArg d) =>
PRO(d) and (Y 0 *) if not (substr(string(Y),0,1) == "+") .
*** RULE 14: PRO, and the attack Y is removable.
crl [expand_PRO_with_Y-#_Y0#] : PRO(d) and (Y hitsArg d) =>
PRO(d) and (Y - #) and (Y 0 #) if (substr(string(Y),0,1) == "-") .
*** RULE 15: PRO, and the attack Y is addable.
crl [expand_PRO_with_Y0#] : PRO(d) and (Y hitsArg d) =>
PRO(d) and (Y 0 #) if (substr(string(Y),0,1) == "+") .
*** RULE 16: CON, and the attack Y is on the system.
crl [expand_CON_with_Y1*] : CON(d) and (Y hitsArg d) =>
(Y 1 *) if not (substr(string(Y),0,1) == "+") .
*** RULE 17: CON, and the attack Y is addable.
crl [expand_CON_with_Y+#_Y1*] : CON(d) and (Y hitsArg d) =>
(Y + #) and (Y 1 *) if (substr(string(Y),0,1) == "+") .
*** RULE 18: CON, and the attack Y is on the system.
crl [expand_CON_with_Y?**] : CON(d) and (Y hitsArg d) =>
(Y ? **) if not (substr(string(Y),0,1) == "+") .
*** RULE 19: CON, and the attack Y is addable.
crl [expand_CON_with_Y+#_Y?**] : CON(d) and (Y hitsArg d) =>
(Y + #) and (Y ? **) if (substr(string(Y),0,1) == "+") .
endm
```