Goal-driven Changes in Argumentation: A theoretical framework and a tool

Pierre Bisquert, Claudette Cayrol, Florence Dupin de Saint-Cyr,
Marie-Christine Lagasquie-Schiex

Goal-driven Changes in Argumentation: A theoretical framework and a tool

PIERRE BISQUERT
CLAUDETTE CAYROL
FLORENCE DUPIN DE SAINT-CYR
MARIE-CHRISTINE LAGASQUIE-SCHIEX

Tech. Report IRIT
RR- -2013-33- -FR

July 2013

Abstract

This paper defines a new framework for dynamics in argumentation. In this framework, an agent can change an argumentation system (the target system) in order to achieve some desired goal. Changes consist in addition/removal of arguments or attacks between arguments and are constrained by the agent's knowledge encoded by another argumentation system. We present a software that computes the possible change operations for a given agent on a given target argumentation system in order to achieve some given goal.

Keywords: Abstract argumentation theory, dynamics in argumentation

Contents

1 Introduction

An abstract argumentation system (AS) [1] is composed of a set of arguments linked by a binary relation, called attack relation, which represents conflicts between arguments. A "semantics" for such a structure defines the way to select subsets of arguments jointly acceptable. In the last few years, the dynamics of these systems has been garnering increased attention from the researchers in argumentation [2, 3, 4, 5, 6, 7]. Change in argumentation raises classical questions: "*How to represent a change? What are its consequences?*". But also new questions concerning the use of such changes emerge : "*Who is concerned by these changes? How to use them in an effective way to achieve a precise goal?*". These two questions concerning *the audience* and *optimal change* led us to propose a new theoretical framework that is close to classical planning. We want to model the reasoning of an agent who has her own knowledge, a goal and the possibility of acting on a target system. Here, the agent's knowledge is represented by an AS. The target on which she can operate is also an AS. For example, during a debate, the publicly exchanged arguments and their interactions may constitute the target system. An agent can act on the target for adding or removing arguments or attacks. In a public debate, adding an argument simply amounts to expressing it. The removal of an argument [8] may be due to an objection or may come from the rejection of a given statement not recognized as a proper argument. The addition (resp. removal) of attack may come from the discovery that two already stated arguments are in conflict (resp. compatible).

The originality of our proposal lies in the use of two AS: one for the agent and one for the target. This introduces some limitations into the set of possible actions of the agent, (since, for instance, she cannot add arguments or attacks if she does not know them), thus some trivial ways to realize a goal may not be allowed for the agent.

We propose a software tool which takes as input the AS of an agent, a target system and a goal. It provides as output the list of actions executable by the agent in order to achieve her goal. This software uses properties characterizing changes that were established in [4, 8]. In this article, we only use a particular semantics (the grounded semantics) but the software can handle others of them (the preferred and stable semantics). In addition, at the current stage, the software handles only some types of change.

In Sect.2, we present an example which will enable us to illustrate our theoretical framework described in Sect.3. In Sect.4, we present the implemented tool which provides the actions to be carried out by the agent in order to achieve her goal. The experimentation protocols and the results are given in Sect.5.

2 The court hearing example

This example, already described in [8], is a court hearing which fits well with an argumentation framework. However, while this example illustrates our definitions easily, it does not cover all the possible applications of our work. We present first the protagonists and their goals then we describe the course of the hearing.

2.1 Protagonists

In this example, four entities are interacting in order to determine if an argument x_0 is acceptable; here, the argument x_0 expresses that the defendant is not guilty. The four entities have quite distinct roles:

- the **prosecutor** wants to obtain the rejection of the argument x_0 given a set of arguments.
- the **defense lawyer**, with his set of arguments (possibly different from those of his adversary, the prosecutor), tries to obtain the acceptance of the argument x_0.
- the **judge** ensures that the process of argumentation takes place under good conditions. When an objection is made by one of the participants, he can accept this objection (thus the corresponding argument is removed), or reject it.
- the **jury** has the last word. Its role is to listen to the arguments of the prosecutor and lawyer. When the hearing is finished (*i.e.* when neither the prosecutor, nor the lawyer can, or want to, give new arguments), the jury deliberates and determines whether the argument x_0 is acceptable or not.

In this example, the prosecutor and the lawyer are not interested by convincing each-other. However, they wish to convince the jury who will make the final decision. Thus, both protagonists will act on a target AS representing the state of knowledge of the jury. The target AS, which is empty at the beginning, represents the central place of exchanges between the prosecutor and the lawyer, where each one will place his arguments in turn, trying to make the jury lean in his favor at the end.

2.2 Protocol

We present a protocol governing the exchange of arguments in the example of the hearing.

1. First of all, the argument x_0, giving the subject of the hearing, is set up. By convention, the defense lawyer speaks first and states the innocence of his client. The jury AS (the target) is modified.

2. the hearing is a sequence of actions made by the agents (either the prosecutor, or the lawyer) according to their own AS. We consider two types of action:

   - an *additive action* is the addition of arguments and/or attacks directly in the jury AS;
   - a *suppressive action* is the removal of one or more arguments and/or attacks from the jury AS. Let us note the particular case of an objection: it consists in asking the judge to rule on the legality of some arguments and/or attacks uttered by the adversary. If the objection is rejected, no modification of the jury AS is carried out. If the objection is accepted, arguments or attacks objected are removed from the jury AS.

   It is also possible for an agent to *do nothing*. It is the case, for instance, when the agent does not have any more argument to advance, or when the jury AS is appropriate for her.

3. a side effect of an action is to inform the adversary: if one of the two protagonists was not aware of one argument uttered by the other then he will add it to its own system. In the same vein, if an argument is considered to be illegal after an objection accepted by the judge, the two agents should occult it in order to no more use it.

4. When the agents both decide that they do not have anything to add, the process stops and the jury deliberates thanks to its AS.

This protocol highlights two types of change:

- a change operated by an agent on a target system other than her own,
- a change operated by an agent on her own system.

This last type of change can be studied by using directly the work of [4] and will not be treated here. In this paper, we investigate only the first type of change by defining a theoretical framework in which an agent chooses an action according to a target system, her own system and her goals.

3 Classical Argumentation Framework and New Definitions

This work is based on various concepts that we will present and illustrate progressively on the example of Sect.2. We will need in particular the notions of *argumentation system (AS)* and *change operation* respectively introduced in [1] and [4]. Beyond the concept of change and the modifications it implies, we are interested in what can cause this change, *i.e.*, why and how a target AS is modified. For this purpose, we will introduce the concept of *goal* of an agent and we will focus on finding the operations an agent can apply in order to achieve her goal.

3.1 Abstract Argumentation

Let us consider a set Arg of symbols (denoted by lowercase letters). The set Arg and a relation $R \subseteq Arg \times Arg$ enable us to define the set of all possible arguments with their interactions, which we call *reference universe*. More precisely, Arg represents a potentially infinite set of arguments available in a particular domain (*e.g.* if the domain is a knowledge base then Arg is the set of all arguments that can be built upon the formulas of the base). It is also possible, following the example below, to suppose that Arg is provided explicitly.

Ex. 1: *During an hearing concerning a defendant (Mr. X), several arguments may be examined in order to determine his culpability. Table 1 presents this set of arguments i.e., the set Arg. The relation R is represented in the graph of Fig.1.*

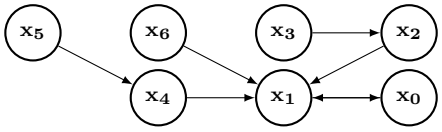


Figure 1: *Reference universe of Mr. X case.*

We slightly modify the AS definition of [1] in order to take into account a reference universe.

Def. 1: *An* argumentation system (AS) *on the universe $\langle Arg, R\rangle$ is a partial subgraph of $\langle Arg, R\rangle$, i.e. a pair $\langle A, R_A\rangle$, where $A \subseteq Arg$ is a finite nonempty set of arguments and $R_A \subseteq R \cap A \times A$ is called* attack relation. *Let $a, b \in A$, aR_Ab means that a attacks b.*
$\langle A, R_A\rangle$ is represented by an argumentation graph *G whose vertices and edges correspond respectively to the arguments and the attacks. $x \in G$ is a shortcut for $x \in A$.*

x_0	*Mr. X is not guilty of the murder of Mrs. X*
x_1	*Mr. X is guilty of the murder of Mrs. X*
x_2	*Mr. X's business associate has sworn that he met him at the time of the murder.*
x_3	*Mr. X associate's testimony is suspicious due to their close working business relationship*
x_4	*Mr. X loves his wife, so he cannot be her killer.*
x_5	*Mr. X has a reputation for being promiscuous.*
x_6	*Mr. X had no interest to kill Mrs. X, since he was not the beneficiary of her life insurance*

Table 1: *Arguments concerning Mr. X case.*

Given a universe $\langle Arg, R\rangle$, $G_k = \langle A_k, R_{A_k}\rangle$ denotes the AS of the agent k on this universe, and represents the part of the reference universe known by k.

Ex.1 (cont.) *The prosecutor does not know all the arguments of the universe (given in Fig.1). Fig.2 illustrates the arguments and the attacks that he knows (G_{pros}).*

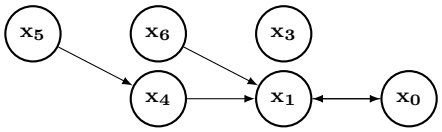


Figure 2: *AS of the prosecutor (G_{pros}).*

The acceptable sets of arguments ("extensions") are computed using a "semantics" based on the following notions:

Def. 2: *Given an AS $\langle A, R_A\rangle$, let $a \in A$ and $S \subseteq A$*

- *S attacks a iff*[1] *$\exists x \in S$ s.t.*[2] *xR_Aa.*

[1] if and only if
[2] such that

- S *is* conflict free *iff* $\nexists a, b \in S$ *s.t.* aR_Ab.
- S defends *an argument* a *iff* S *attacks any argument attacking* a. *The set of the arguments defended by* S *is denoted by* $\mathcal{F}(S)$; $\mathcal{F}$ *is called the* characteristic function *of* $\langle A, R_A \rangle$. *More generally,* S indirectly defends a *iff* $a \in \bigcup_{i \geq 1} \mathcal{F}^i(S)$.
- S *is an* admissible set *iff it is both conflict free and defends all its elements.*

In this article, we focus on one semantics among those proposed by [1]:

Def. 3 (Grounded semantics): *Let* $\mathcal{E} \subseteq A$, $\mathcal{E}$ *is the only* grounded extension *iff* $\mathcal{E}$ *is the smallest fixed point (wrt*[3] $\subseteq$*) of the characteristic function* $\mathcal{F}$.

Ex.1 (cont.) *The grounded extension representing the acceptable arguments for the prosecutor is* $\{x_0, x_3, x_5, x_6\}$.

In addition, the status of an argument is defined wrt its presence in the extensions of the selected semantics. In our particular case, an argument is "accepted" if it appears in the grounded extension and "rejected" otherwise.

Thus, for the prosecutor, x_0 is accepted and x_1 rejected. But, although he does not have the arguments to prove undoubtedly that the defendant is guilty, the prosecutor wishes that x_1 would be accepted by the jury. In the following section, we will see how the prosecutor can act on the jury AS in order to make x_1 accepted.

3.2 Change in argumentation

According to [4], an elementary change is either the addition/removal of an argument with a set of related attacks, either the addition/removal of one precise attack. In order to be more concise, we only consider here the operations concerning the addition or the removal of an argument. Moreover, we refine the concept of elementary operation in the sense of [4] in four steps: first, we define its *syntax*; then we relate the operation to an agent in order to determine if it is *authorized* or not for her, *i.e.* if she knows the elements involved in the operation. We then take into account the target to determine if the operation is *executable*: the addition (resp. removal) of an argument is only achievable if this argument is absent (resp. present) in the target system. Lastly, we define the *impact* of an operation on an AS. The restriction to elementary operations is done without loss of generality since any operation can be represented by a sequence of elementary operations, called *program* in Def.5. In addition, we suppose in this work that all the systems considered are relative to the same universe $\langle Arg, R \rangle$.

[3] with respect to

Def. 4: *Let k be an agent, $G_k = \langle A_k, R_{A_k} \rangle$ her AS and $G = \langle A, R_A \rangle$ any AS.*

- *an* elementary operation *is a triplet $o = \langle op, arg, att \rangle$ where $op \in \{\oplus, \ominus\}$, $arg \subseteq Arg$, $att \subseteq R$ and*
  - *if $op = \oplus$ then $|arg| = 1$ and $\forall (x, y) \in att,\ (x \neq y)$ and $(x \in arg$ or $y \in arg)$,*
  - *if $op = \ominus$ then $|arg| = 1$ and $att = \varnothing$.*
- *an elementary operation $\langle op, arg, att \rangle$ is* authorized for *k if $arg \subseteq A_k$ and $att \subseteq R_{A_k}$* [4].
- *an* executable *operation* by *k* on *G is an elementary operation $\langle op, arg, att \rangle$ authorized for k s.t.:*
  - *if $op = \oplus$, then $arg \not\subset A$ and $\forall (x, y) \in att,\ (x \in A$ or $y \in A)$,*
  - *if $op = \ominus$, then $arg \subseteq A$.*
- *an operation $o = \langle op, arg, att \rangle$ executable by k on G provides a new AS $G' = O(G) = \langle A', R_{A'} \rangle$ s.t.:*
  - *if $op = \oplus$ then $G' = \langle A \cup arg, R_A \cup att \rangle$,*
  - *if $op = \ominus$ then $G' = \langle A \setminus arg, R_A \setminus \{(x, y) \in R_A$ s.t. $x \in arg$ or $y \in arg\} \rangle$.*

Ex.1 (cont.) *Given the reference universe of Fig.1, here is a non-exhaustive list of* elementary *operations:*

- $\langle \oplus, \{x_2\}, \{(x_2, x_1)\} \rangle$
- $\langle \oplus, \{x_2\}, \{(x_2, x_1), (x_3, x_2)\} \rangle$
- $\langle \oplus, \{x_6\}, \{(x_6, x_1)\} \rangle$
- $\langle \oplus, \{x_6\}, \varnothing \rangle$
- $\langle \ominus, \{x_4\}, \varnothing \rangle$
- $\langle \ominus, \{x_2\}, \varnothing \rangle$

Among these elementary operations, the agent (the prosecutor) is not authorized *to use those concerning arguments or attacks that she does not know. Thus, she will not be able to use the following operations:*

- $\langle \oplus, \{x_2\}, \{(x_2, x_1)\} \rangle$,
- $\langle \oplus, \{x_2\}, \{(x_2, x_1), (x_3, x_2)\} \rangle$,
- $\langle \ominus, \{x_2\}, \varnothing \rangle$.

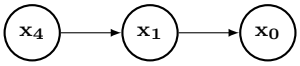


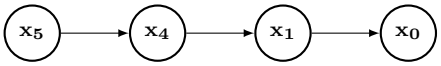
Figure 3: *AS of the jury (G_{jury}).*

Let G_{jury} be the jury AS (Fig.3).

Here are some executable operations on G_{jury} by the agent:

- $\langle \ominus, \{x_4\}, \varnothing \rangle$
- $\langle \oplus, \{x_6\}, \varnothing \rangle$
- $\langle \oplus, \{x_6\}, \{(x_6, x_1)\} \rangle$
- $\langle \oplus, \{x_5\}, \{(x_5, x_4)\} \rangle$

Finally, the impact *of the last operation on the jury AS produces:*



We consider below the *programs*, *i.e.* the sequences of executable operations by an agent on an AS. This enables an agent to carry out several elementary operations in sequence.

Def. 5: *Let k be an agent and G any AS. An* executable program *p by k on G is an ordered finite sequence of m operations $(o_1, \cdots, o_m)$ s.t.:*

- *$m = 1$: o_1 is executable by k on G. In this case, $p(G) = o_1(G)$.*
- *$m > 1$: $(o_1, \cdots, o_{m-1})$ is an executable program p' by k on G s.t. $p'(G) = G'$ and o_m is executable by k on G'. In this case, $p(G) = o_m(G')$.*
- *By extension, an empty sequence is also a program. In this case, $p(G) = G$.*

Let us now define what could be a program achieving a precise goal.

3.3 Goals and programs

An agent can act on a target AS in order to achieve some goals. A goal is formally represented by a formula in a language that expresses conditions that may hold for some AS (e.g. the target AS).

For representing these goals, we use the symbols appearing in the typology of the change properties defined in [9][5]:

[4] In the case of addition of an argument, only a part of the known attacks may be provided; it is thus possible for an agent to carry out a "lie by omission", for example for strategic reasons. On the other hand, the agent is not authorized to provide unknown arguments or attacks; she cannot thus lie actively; this could be the object of future work.

[5] In this typology, several semantics are considered, thus unlike with the grounded semantics, there may be several sets of extensions. The set of the extensions of $\langle A, R_A \rangle$ under a given semantics is denoted by $\mathbf{E}$ (with $\mathcal{E}_1, \ldots, \mathcal{E}_n$ being the extensions).

- arguments (x, y, z, etc),
- extensions ($\mathcal{E}_i$, $\mathcal{E}'_i$),
- the set of extensions ($\mathbf{E}$, $\mathbf{E}'$), their cardinality ($|\mathbf{E}|$, $|\mathbf{E}'|$),
- the set of all the extensions containing a particular argument x ($\mathbf{E}_x$, $\mathbf{E}'_x$) and their cardinality ($|\mathbf{E}_x|$, $|\mathbf{E}'_x|$),
- classical comparison operators ($=$, $<$, $>$, etc),
- quantifiers $\forall$ and $\exists$,
- membership ($\in$) and inclusion ($\subseteq$),
- union ($\cup$) and intersection ($\cap$) of sets,
- classical logical operators ($\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\neg$).

We will note b_k the goal of the agent k.

Ex.1 (cont.) *The goal b_{pros} of the prosecutor can be represented by $(x_1 \in \mathcal{E}')$ with $\mathcal{E}'$ being the grounded extension of the jury AS.*

Then, we give the notion of a program achieving a goal.

Def. 6: *Let k be an agent, b_k her goal and G any AS. A* program p of k on G achieving b_k *is an executable program by k on G s.t. $p(G) = G'$ and b_k holds in G'* [6].

Ex.1 (cont.) *Considering the goal b_{pros} represented by $(x_1 \in \mathcal{E}')$, the programs $(\langle \oplus, \{x_5\}, \{(x_5, x_4)\}\rangle)$ and $(\langle \ominus, \{x_4\}, \varnothing\rangle)$ of the prosecutor on G_{jury} achieve b_{pros}.*

In the next section, we study how to compute the operations that an agent should do in order to achieve her goal.

3.4 Change characterizations

A "naive" approach is to compute, for each executable operation, the set of extensions of the modified target AS and to check if the goal is achieved. Another more efficient approach uses the *change characterizations* which were studied in [4] and [8]. A characterization is a property that gives necessary and/or sufficient conditions for achieving a particular goal wrt a kind of operation and a semantics. We give two examples of characterizations:

[6] or in (G, G') if the goal expresses a condition on both systems.

Charact. 1 ([9])**:** *When adding an argument z under grounded semantics, if z is not attacked by G and z indirectly defends x and $x \notin \mathcal{E}$, then $x \in \mathcal{E}'$.*

This characterization concerns the goal of "enforcement" of an argument x. Enforcement, introduced by [5], consists in ensuring that an argument which was rejected would be accepted. This characterization also specifies the operation involved: here the addition of an argument z. Thus, thanks to this characterization, we know (without requiring a new computation of the extensions) that if an operation adds an argument z under the grounded semantics, s.t. z is not attacked and indirectly defends another argument x which was not accepted, then x will become accepted.

Charact. 2 ([9])**:** *When removing an argument z under grounded semantics, if $\mathcal{E} \neq \varnothing$ and z is attacked by G, then $\mathcal{E}' \neq \varnothing$.*

Here the goal concerns the evolution of the non emptiness of the extension. This characterization enables us to know, without any computation, that if an operation removes an argument z s.t. z is attacked by at least one argument of G and knowing that the extension was not empty before the change, then the extension obtained after the change will not be empty. This can be useful when one wants to make sure that the discussion will not be fruitless.

This concludes our theoretical framework. The following section presents the tool that has been developed.

4 Presentation of the tool

This tool is organized around two specific modules: an AS handler, and an inference engine (for computing the change operations). The outputs of the first module are inputs for the second module.

4.1 The argumentation systems manager

We present here the argumentative facet of the program, which may handle the creation of various AS and enable the computation of the extensions. This module is encoded in an object language (*Python 2.7*) which is convenient for implementing the concepts used. Thus, creating an AS requires a set of arguments and a set of attacks.

This module handles the consistency of the input data by checking that any argument appears only once (with the same name), and that the input attacks relate existing arguments. This module returns information concerning the AS and launches (on request) the computation of the extensions wrt a semantics (which can

be provided as a parameter). In accordance with our theoretical framework, the tool makes it possible to create two AS, one for the agent and one for her target. This creation is carried out by providing the list of the arguments and attacks of these two systems. Moreover, the semantics used in the target system is also specified (it will be used to check the achievement of the goals of the agent). The agent AS and the target AS, as well as the extensions of this last, are then transmitted to the second module.

4.2 Inference Engine

This second module computes the change operations. More precisely, it allows to answer to the question *"What are the operations executable by the agent on the target system that achieve her goal?"*.

The inference engine receives the agent AS and the target AS, as well as the set of extensions of the target for a given semantics. Besides, it is necessary to provide a goal that the agent wants to achieve on the target system and a set of characterizations allowing to check if an operation achieves that goal. Let us note that the user can filter the results by forcing the type of operation, for instance if she is only interested in the additions of arguments.

The heart of this module is a rule which generates all the operations executable by the agent on the target and achieving the goal, and produces, for each operation, the characterization justifying this result. This rule contains two parts, one part builds the operations, and the other checks that the operations fit the *desiderata* of the agent.

A synthetic vision of the tool, and thus of the articulation between its two modules, is given in Fig.4.

This second module has been encoded with a logic programming language (*Prolog*) for two main reasons: the characterizations translate naturally into logical rules, and the mechanism of unification allows us to generate and easily filter the operations wrt the AS and the goal of the agent.

Construction of the operations The construction of the operations is a direct translation in *Prolog* of Def.4. Thus, we generate executable operations and their impact on the argumentation graph (it is either the addition or the removal of an argument). This makes it possible to avoid considering operations that are not authorized or not executable and thus to optimize the computing time.

Checking the operations After its generation by the construction rules, the operation is treated thanks to the characterizations (e.g., Charac.1 or 2). Thus, for a given operation, if there exists a characterization corresponding to the type of

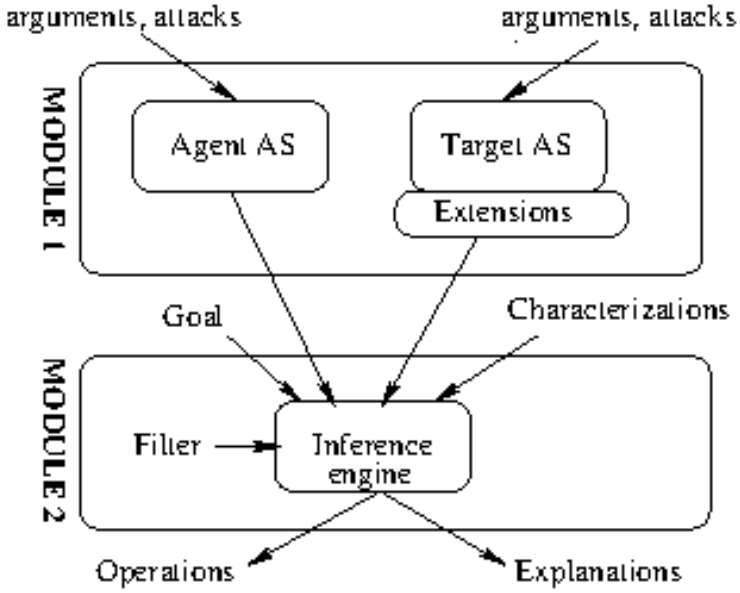


Figure 4: *Architecture of the tool.*

operation and the goal requested by the agent and if the conditions of this characterization are satisfied, then this operation will be provided to the user (with the corresponding characterization as an explanation).

Let us note that for the moment the tool only handles programs reduced to one elementary operation.

4.3 Application to the example

We illustrate the use of the tool on the example of the hearing: the prosecutor wants to have the argument x_1 accepted. He knows the arguments presented in Fig.2 and has to modify the jury AS, represented in Fig.3.

First of all, the necessary data must be provided, *i.e.* the sets of arguments and attacks that the prosecutor knows, the sets of arguments and attacks known by the jury (the target AS) and the semantics used by the jury (we suppose here that it is the grounded semantics).

The tool will deduce from these data the set of the extensions for the jury AS ($\mathbf{E} = \{\{x_0, x_4\}\}$).

The prosecutor must also specify his goal: "x_1 must belong to the grounded extension".

And finally a set of characterizations must be provided. In order to simplify the explanation, let us suppose that this set is reduced to the single characterization 1.

The tool then generates the operations executable by the prosecutor on the jury AS and their impacts. For example:

- $\langle \ominus, \{x_4\}, \varnothing \rangle$, which successfully completes the construction step and whose impact is the system $(\{x_0, x_1\}, \{(x_1, x_0)\})$,
- $\langle \oplus, \{x_5\}, \{(x_5, x_4)\} \rangle$, which also successfully completes the construction step and whose impact is the system $(\{x_0, x_1, x_4, x_5\}, \{(x_1, x_0), (x_4, x_1), (x_5, x_4)\})$.

Let us note that the operations concerning the arguments or attacks unknown by the agent, or which are not executable on the target AS, are not generated by the tool. Thus, the operation $\langle \ominus, \{x_6\}, \varnothing \rangle$ is not generated because the argument x_6 is not present in the jury AS. This ensures the finiteness of the process. Generated operations are examined at the same time through the characterizations. If an operation does not correspond to any characterization, it is rejected; if it matches with one or more characterizations, then the tool returns all the pairs (operation, characterization).

In our example, the operation $\langle \ominus, \{x_4\}, \varnothing \rangle$ is rejected because its type (removal of an argument) does not correspond to that specified in the only characterization available. On the other hand, the type of the operation $\langle \oplus, \{x_5\}, \{(x_5, x_4)\} \rangle$ corresponds; the tool must thus check that the latter satisfies the constraints specified in the characterization, namely that z, in fact paired with the argument x_5, is not attacked and that it must defend indirectly x (this last being paired with x_1) such as x does not belong to the extension[7]. These conditions being satisfied, the operation realizes indeed the goal of the prosecutor.

5 Experiments

The aim of our tool is to find a "program" (in the sense of Def.5) achieving a goal. Let us point out that as a first step we restrict the search to the *programs containing only one operation*. This search could have been made directly by computing the impact of an operation and then by checking the set of the extensions of the resulting graph. Nevertheless, recomputing extensions can be very expensive (see [7]), whereas computing the impact of an operation is easy in terms of arguments and attacks (it just amounts to handle elementary operations on sets). Thus, our idea is to generate executable operations and "to check them" thanks to the characterizations, rather than compute the extensions for each resulting system and then check

[7]The notions appearing in the characterizations are implemented in the inference engine (for instance, indirect attack by an argument, direct and indirect defense by an argument, attack and defense of sets, etc).

that the goal is satisfied. The benefits in terms of time and of space still remain to be evaluated (this will be the subject of a future study). Before analyzing the results produced by the tool (Sect.5.2), we present, in Sect.5.1, our experimental protocols. In Sect.5.3, we discuss their drawbacks and benefits.

5.1 Experimental Protocols

We propose two experimental protocols for checking the soundness of the results. These protocols will also enable us to reveal some lacks in the set of characterizations.

5.1.1 Inclusion Hypothesis

Both protocols suppose that the target AS is a partial subgraph of the agent AS. This assumption seems natural in the court hearing example: the prosecutor and lawyer are aware of everything what is said during the hearing, so they know all the arguments that are present in the jury AS (their target)[8].

5.1.2 Random Generation

This protocol aims at randomly generating two systems (one for the agent and one for the target) respecting the inclusion hypothesis, and at randomly generating a goal to achieve. Thus:

- a set of arguments of size n is created, with n being a random value between 2 and 20,
- a set of attacks between these arguments of size nb_att is created, with nb_att being a random value between $n * 0.5$ and $n * 1.5$.

This constitutes the first AS. It will be used as a basis to create a partial subgraph representing the target AS:

- two numbers sup_arg and sup_att are generated randomly, with sup_arg being a value between 0 and $n - 1$ and sup_att being a value between 0 and $nb\_att - 1$,

[8] Removing this assumption would mean to consider that an agent may not agree with the validity of some presented arguments or attacks. This refers to the question of the differentiation between being aware of the existence of an information and believing in its validity. This question would deserve a deeper study which is out of the scope of this paper.

- sup_arg arguments are removed randomly from the set of arguments, and for each removed argument we subtract from sup_att the number of attacks involving this argument,

- sup_att attacks are removed randomly.

This enables us to obtain the second AS.

And finally, a goal is randomly selected among $x \in \mathcal{E}'$, $x \notin \mathcal{E}'$, $z \in \mathcal{E}'$, $\mathcal{E} = \mathcal{E}'$, $\mathcal{E} \subseteq \mathcal{E}'$, $\mathcal{E} \subset \mathcal{E}'$, $\mathcal{E}' \subseteq \mathcal{E}$, $\mathcal{E}' \subset \mathcal{E}$, $\mathcal{E}' = \varnothing$, $\mathcal{E}' \neq \varnothing$, where x belongs to the target AS, z belongs to the agent AS and $\mathcal{E}$ (resp. $\mathcal{E}'$) is the grounded extension of the target AS before (resp. after) the change.

The generation of operations is executed from these two systems and the goal (see the results obtained in Sect.5.2).

5.1.3 Systematic Generation

Although the random generation is efficient, it can leave aside some interesting cases. To solve this problem, we set up another experimental protocol consisting in exhaustively testing a list of pairs of AS.

Thus, for n arguments, we create all the pairs of AS s.t.:

- the first AS has a set of arguments arg_agent (indexed from 1 to n) and a set of attacks att_agent (the number goes from 0 to $n * (n - 1)$).

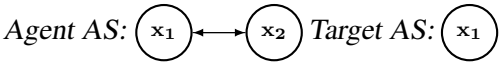
- the second AS has a set of arguments arg_target varying among all the possible subsets of arg_agent, and a set of attacks varying among all the possible subsets of att_agent restricted to the arguments of arg_target.

We carry out the generation of operations for a given kind of goals: we have successively considered the enforcement of each of the arguments of the target AS. Our protocol computes the grounded extension of the system resulting from the operation. This enables us to reveal the "covering" problems of the tool, *i.e.* cases where no operation is found that achieves the goal whereas there is one (see the following section).

5.2 Results

For the random generation For each pair of AS generated (in all five million), the first experimental protocol returns information concerning the generation of the executable operations achieving a goal chosen randomly. Ex.2 proposes a short extract showing a case of successful generation of operations.

Ex. 2: *Consider the following AS and the goal $x_1 \notin \mathcal{E}'$:*



The tool generates two operations: $\langle \oplus, \{x_2\}, \{(x_2, x_1)\}\rangle$ and $\langle \oplus, \{x_2\}, \{(x_2, x_1), (x_1, x_2)\}\rangle$ with the corresponding characterizations (here, it is twice the same – corresponding to Prop.16 of [9]).

In addition, the protocol also proposes a summary of the results, for each goal, by detailing the number of cases where a solution is found. Tab.2 shows an example of such results.

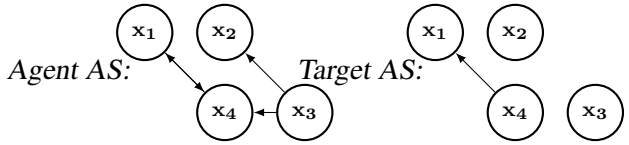
Goal	**Nb. of cases tested**	**Set of solutions**		**%**
		Non empty	**Empty**	
$x \in \mathcal{E}'$	499216	443010	56206	88.7
$x \notin \mathcal{E}'$	500009	397706	102303	79.6
$z \in \mathcal{E}'$	500196	443931	56265	88.8
$\mathcal{E} = \mathcal{E}'$	499770	389933	109837	78.0
$\mathcal{E} \subseteq \mathcal{E}'$	499116	499116	0	100.0
$\mathcal{E} \subset \mathcal{E}'$	500697	489562	11135	97.8
$\mathcal{E}' \subseteq \mathcal{E}$	499860	435600	64260	87.1
$\mathcal{E}' \subset \mathcal{E}$	499546	402207	97339	80.5
$\mathcal{E}' = \varnothing$	500728	27222	473506	5.4
$\mathcal{E}' \neq \varnothing$	500862	279162	221700	55.7
Total	5000000	3807449	1192551	76.1

Table 2: *Summary of results for the protocol of random generation. For each goal, the second column gives the total number of pairs of AS tested by the tool. The third column (resp. fourth column) gives the number of times where the tool returned a nonempty (resp. empty) set of solutions. Lastly, the fifth column gives the percentage of the cases where the tool returned a nonempty set of solutions compared to the total number of cases tested for a particular goal.*

Whether the tool finds solution or not depends on the goal considered. More precisely, there are two possible explanations when no executable operation is found:

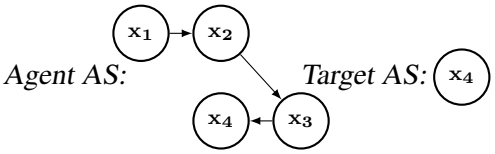
- *there does not exist any executable operation* achieving the goal; let us note that this case includes the case where it is not possible to achieve this goal with *only one* operation (*cf* Ex.3).
- *a characterization is missing* in order to find at least one operation (*cf* Ex.4).

Ex. 3: *Consider the following AS and the goal $\mathcal{E}' = \varnothing$:*



The tool does not generate any operation because there exist no executable operation achieving the goal: on the one hand, there exists no argument attacking at the same time x_2, x_3 and x_4 and being attacked by at least one of them, and, on the other hand, it is not possible to remove all the arguments in only one operation.

Ex. 4: *Consider the following AS and the goal $\mathcal{E}' \neq \varnothing$:*



The tool does not generate any operation, and yet, doing nothing would be enough to achieve the goal; thus there is a gap in the set of characterizations (the possibility of doing nothing had not been considered in [9]).

The set of solutions provided by the tool can thus be empty for very different reasons but which are not distinguished by the protocol. The second experimental protocol has been set up in order to highlight examples revealing lacks in the set of characterizations (results given below).

For the systematic generation By considering a particular goal, and by treating all the possible cases, this protocol enabled us to concentrate on the cases where the tool does not find a solution and thus to detect the lacks in our set of characterizations. Thus, we have enriched the set of characterizations, so that currently, for the goal corresponding to the enforcement of one argument in the grounded extension, the automatic generation for $n = 3$ or 4 does not detect any lack of characterization.

5.3 Discussing the protocols

Our protocol of random generation enables us to test a large number of examples (with various goals). However, the chosen examples are not necessarily representative of all the cases.

Concerning the systematic generation, it is very expensive: for a generation based on n arguments, there are $2^{n*(n-1)}$ possible AS for the agent and in the worst case $2^{n*(n-1)} + (n * 2^{(n-1)*(n-2)}) + \cdots + (n * 2^0)$ possible cases for the target AS. The total number of possible pairs is about $2^{n*(n-1)} * (2^{n*(n-1)} + (n *$

$2^{(n-1)*(n-2)}) + \cdots + (n * 2^0))$. This is tolerable for a small number of arguments. For example, for $n = 3$, there are $2^6 * (2^6 + 3 * 2^2 + 3) = 5056$ possible pairs. But this becomes prohibitive as soon as they are more than five arguments. This problem is partly due to the useless generation of isomorphic pairs of graphs.

The second disadvantage of this protocol is that it depends on the choice of the goal. It is thus necessary to redefine a protocol with each new goal.

6 Discussion and conclusion

We presented a theoretical framework and a tool able to find a change operation which achieves a goal given a target AS and given arguments and attacks from a "source" AS (representing the knowledge of an agent). We studied the behavior of this tool by means of two experimental protocols. Nevertheless the experiments are not finished, since our protocols do not allow to evaluate all the possible examples exhaustively.

Future works can be declined along three axes. The first axis relates to the tool itself:

- Since computing the extensions after change is very expensive (in spite of the progress made in [7]), our objective is to show that our approach is less expensive. That will require to determine the complexity of the inference engine algorithm (written in Prolog), one difficulty being to take into account the check of the applicability of some characterizations [9].

- In addition, we could consider programs with more than only one elementary operation. Considering sequences of operations will make it possible to carry out more complex modifications and thus to find more solutions.

- We evoked the use of the tool to locate gaps of characterization; it could be interesting to develop a sharper analyzer which would detect all the executable operations than the tool does not find.

The second axis relates to the experiments carried out thanks to the tool and its applications:

- The continuation of experiments, such as those presented in this article, is important in order to create "benchmarks" in the argumentation field.

[9] Indeed some characterizations use complex concepts: for example the indirect defense of an argument by a set.

- It could be interesting to consider real cases of argumentation to establish other more concrete "benchmarks". In particular, online debates (on social networks for example) seem well adapted to our problematic.
- The previous point constitutes a track of applications of our tool, which could enable a user to be guided in his choice of arguments to utter during an online debate.

The third and last axis relates to theoretical points:

- At the time we developed our experimentation protocol, we assumed the inclusion of the target AS in the agent AS. This questions the complete or incomplete knowledge of the system on which the agent wants to act. Moreover the question of how the agent can update her own system deserves a thorough study.
- It seems necessary to extend our hearing example in order to allow a real interaction between the prosecutor and the lawyer. In a more general way, this implies that we study the changes operated by an agent on her own system when another agent carries out a modification of the target AS.
- Lastly, up to that point, we limited ourselves to a persuasion dialog, with agents having contradictory goals. It could be interesting to consider other types of dialogs bringing into play a coalition of agents cooperating to achieve a goal, for example several lawyers trying to flesh out their pleadings.

References

[1] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artificial Intelligence*, vol. 77, no. 2, pp. 321–358, 1995.

[2] G. Boella, S. Kaci, and L. van der Torre, "Dynamics in argumentation with single extensions: Attack refinement and the grounded extension," in *Proc. of AAMAS*, 2009, pp. 1213–1214.

[3] ——, "Dynamics in argumentation with single extensions: Abstraction principles and the grounded extension," in *Proc. of ECSQARU (LNAI 5590)*, 2009, pp. 107–118.

[4] C. Cayrol, F. Dupin de Saint Cyr, and M.-C. Lagasquie-Schiex, "Change in abstract argumentation frameworks: Adding an argument," *JAIR*, vol. 38, pp. 49–84, 2010.

[5] R. Baumann and G. Brewka, "Expanding argumentation frameworks: Enforcing and monotonicity results," in *Proc. of COMMA*. IOS Press, 2010, pp. 75–86.

[6] M. O. Moguillansky, N. D. Rotstein, M. A. Falappa, A. J. García, and G. R. Simari, "Argument theory change through defeater activation," in *Proc. of COMMA 2010*. IOS Press, 2010, pp. 359–366.

[7] B. Liao, L. Jin, and R. C. Koons, "Dynamics of argumentation systems: A division-based method," *Artificial Intelligence*, vol. 175, no. 11, pp. 1790 – 1814, 2011.

[8] P. Bisquert, C. Cayrol, F. Dupin de Saint-Cyr, and M.-C. Lagasquie-Schiex, "Duality between addition and removal," in *Advances on Computational Intelligence*, vol. 297. Springer, 2012, pp. 219–229.

[9] ——, "Characterizing change in abstract argumentation systems," IRIT, UPS, Toulouse, France, Tech. Rep., 2013, ftp://ftp.irit.fr/pub/IRIT/ADRIA/Rapport-IRIT-2013-22.pdf.