Changements guidés par les buts en argumentation : Cadre théorique et outil

P. Bisquert bisquert@irit.fr
C. Cayrol ccayrol@irit.fr
F. Dupin de Saint-Cyr dupin@irit.fr
M.C. Lagasquie-Schiex lagasq@irit.fr

IRIT, Université Paul Sabatier,
118 route de Narbonne, 31062 Toulouse, France

Résumé :
Cet article définit un cadre théorique pour étudier le changement en argumentation. Ce cadre permet de prendre en compte le raisonnement d'un agent qui désire modifier un système d'argumentation cible afin d'atteindre certains buts. Les modifications sont des ajouts/retraits d'arguments ou d'attaques. L'agent est contraint par ses propres connaissances représentées par un deuxième système d'argumentation. Nous présentons un logiciel capable de calculer les opérations exécutables par un agent pour atteindre ses buts sur une cible donnée.

Mots-clés : Argumentation abstraite, Dynamique d'un système d'argumentation

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. The agent must respect some constraints induced by her own 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

1 Introduction

Un système d'argumentation abstrait [8] se compose d'un ensemble d'éléments nommés arguments et d'une relation binaire, nommée relation d'attaque, qui représente des conflits entre arguments. Une sémantique pour une telle structure permet de déterminer des ensembles d'arguments conjointement acceptables. Depuis quelques années, les chercheurs en argumentation ont commencé à s'intéresser à la dynamique de ces systèmes [6, 5, 7, 1, 10, 9].

Le changement en argumentation a amené des interrogations classiques : "*Comment le représenter ? Quelles en sont les conséquences ?*". De nouvelles questions se concentrant sur l'usage de tels changements ont aussi émergé : "*A qui sont destinés ces changements ? Comment les utiliser de manière effective pour arriver à un but précis ?*". Ce sont précisément ces deux questions : *le changement optimal* et *la prise en compte de l'audience*, qui nous ont amenés à proposer un nouveau cadre théorique.

Le cadre que nous allons décrire dans ce papier est proche de la planification classique. On désire modéliser le raisonnement d'un agent qui possède ses propres connaissances, un objectif, et qui a la possibilité d'agir sur un système cible. Ici, les connaissances de l'agent sont représentées sous la forme d'un système d'argumentation. La cible sur laquelle il peut agir est aussi un système d'argumentation. Par exemple, lors d'un débat public, les arguments échangés publiquement ainsi que leurs interactions peuvent constituer le système cible. Les moyens d'actions de l'agent sont des ajouts ou des retraits d'arguments ou d'attaques sur le système cible. Dans un débat public, l'ajout d'un argument consiste simplement à l'énoncer. Le retrait d'un argument paraît moins naturel mais il peut se produire [2] lors d'une objection ou d'une disqualification totale d'un énoncé que l'on ne reconnaît pas comme argument. L'ajout d'attaque peut venir de la découverte que deux arguments déjà énoncés sont en conflit, le retrait d'attaque peut être issu du constat que deux arguments précédemment jugés antagonistes sont en fait compatibles.

L'originalité de notre formalisation du changement réside dans l'utilisation de deux systèmes d'argumentation : un pour l'agent et un pour la cible. Cela introduit des limitations aux actions possibles de l'agent, (puisqu'il ne peut pas, par exemple, ajouter des arguments ou des attaques qu'il ne connaît pas) et cela rend le problème de la réalisation de ses objectifs moins trivial.

Nous proposons un outil logiciel qui prend en entrée le système d'argumentation d'un agent, un système cible et un but. Il fournit en sortie la liste des actions exécutables par l'agent afin de réaliser son but. Ce logiciel utilise des propriétés caractérisant le changement établies dans [7, 4, 2]. Dans cet article, nous nous sommes limités à présenter l'étude d'une sémantique particulière (la sémantique basique) mais le logiciel peut en traiter d'autres (les sémantiques préférée

et stable). D'autre part, le logiciel réalisé calcule pour l'instant uniquement certains types de changements.

Nous présentons en section 2 un exemple qui nous permettra d'illustrer notre cadre théorique exposé en section 3. En section 4, nous décrivons l'outil implémenté destiné à fournir les actions à effectuer par l'agent pour atteindre ses objectifs. Les résultats obtenus et les protocoles d'expérimentation utilisés sont donnés en section 5.

2 Exemple d'utilisation

Nous reprenons ici l'exemple décrit dans [4]. Il s'agit d'un exemple simple qui se prête bien à l'argumentation : une audience de tribunal. Bien entendu, si cet exemple permet d'illustrer aisément les notions mises en jeu, il ne résume pas à lui seul toutes les applications de notre travail. Cet exemple sera présenté en deux parties, la première se concentrant sur les protagonistes et leurs buts, la seconde illustrant le déroulement de l'audience.

2.1 Protagonistes

Cet exemple s'inspire du fonctionnement d'une audience de tribunal mettant en scène quatre entités aux rôles bien distincts qui interagissent pour déterminer si un argument x_0 est acceptable. Ici, l'argument x_0 signifie que l'inculpé n'est pas coupable.

- Le **procureur** veut faire rejeter l'argument x_0. Il a à sa disposition un ensemble d'arguments, éventuellement différents de ceux de son adversaire (l'avocat).
- L'**avocat de la défense**, disposant également d'un ensemble d'arguments, tente de faire accepter l'argument x_0.
- Le **juge** s'assure que le processus d'argumentation s'opère dans de bonnes conditions. Il intervient lorsqu'une objection est faite par un des participants ; il peut alors accepter cette objection (donc faire retirer l'argument correspondant), ou la rejeter.
- Les **jurés** [1] ont le mot de la fin. Leur rôle est d'écouter les arguments du procureur et de l'avocat et d'en tirer une conclusion concernant l'acceptabilité de l'argument x_0. Ils n'interviennent pas durant les échanges entre le procureur, l'avocat et le juge. Lorsque l'audience est terminée (*i.e.* lorsque ni le procureur, ni l'avocat ne peuvent, ou ne veulent, donner de nouveaux arguments), les jurés peuvent délibérer et déterminer si l'argument x_0 est acceptable ou non.

1. Bien qu'au pluriel, les jurés sont une seule et même entité décisionnaire.

Dans cet exemple, le procureur et l'avocat ne sont pas intéressés par le fait de se convaincre mutuellement. Par contre, ils souhaitent convaincre les jurés qui vont prendre la décision finale. Les deux protagonistes vont donc agir sur un système d'argumentation cible représentant l'état des connaissances des jurés, ce dernier étant vide au départ. Ce système représente le lieu central d'échanges entre le procureur et l'avocat, où chacun de ces derniers va placer à tour de rôle des arguments, de sorte qu'à la fin les jurés penchent en sa faveur.

2.2 Protocole

Nous présentons un protocole gérant l'échange d'arguments dans l'exemple de l'audience.

1. Tout d'abord, l'argument x_0 de l'audience est mis en place. Par convention, c'est l'avocat de la défense qui prend la parole en premier et énonce l'innocence de son client. Cette action prend pour cible le système d'argumentation des jurés, qui est modifié.
2. S'ensuit une séquence d'actions proposées par les différents agents (soit le procureur, soit l'avocat) en fonction de leur propre système d'argumentation. Deux types d'actions sont possibles :
   - Les *actions additives*, c'est-à-dire l'ajout d'un ou plusieurs arguments et/ou attaques directement dans le système d'argumentation des jurés.
   - Les *actions suppressives*, c'est-à-dire le retrait d'un ou plusieurs arguments et/ou attaques du système d'argumentation des jurés, (par exemple objection). Notons le comportement particulier de l'objection : elle consiste à demander au juge de statuer sur la légalité d'arguments et/ou d'attaques de l'adversaire. Si l'objection est rejetée, aucune modification du système des jurés n'est effectuée. Si l'objection est acceptée, les arguments ou attaques objectés sont retirés du système des jurés.

   Il est aussi possible pour un agent de ne *rien faire*. Ceci peut notamment arriver lorsqu'un agent n'a plus d'argument à avancer,

ou lorsque le système des jurés est dans un état lui convenant.

3. Un effet de bord des actions est le fait que si un des deux protagonistes n'a pas connaissance de l'argument que son adversaire énonce, il va l'ajouter à son propre système. De même, si un argument est jugé illégal suite à une objection acceptée par le juge, les deux agents doivent l'occulter afin de ne plus l'utiliser.
4. Lorsque les deux agents décident de ne plus rien faire, le jeu s'arrête et les jurés délibèrent grâce à leur système d'argumentation.

Ce protocole met en évidence deux types de changement :
– changement opéré par un agent sur un système cible autre que le sien,
– changement opéré par un agent sur son propre système.
Ce dernier type de changement peut être étudié en utilisant directement les travaux de [7] et ne sera par conséquent pas traité ici.
Dans ce papier, nous allons donc nous consacrer uniquement à l'étude du premier type de changement. Nous présentons ainsi un cadre théorique dans lequel un agent choisit une action en fonction d'un système cible, de son propre système et de ses buts.

3 Cadre formel et nouvelles définitions

Ce travail se base sur diverses notions que nous présenterons et illustrerons au fur et à mesure sur l'exemple de la section 2. Nous aurons notamment besoin des *systèmes d'argumentation* et des *opérations de changement* présentés dans [8] et [7].

Au-delà de la notion de changement et des modifications qu'il implique, nous nous intéressons à ce qui peut provoquer ce changement, c'est-à-dire, pourquoi et comment un système d'argumentation cible est modifié. Nous introduirons donc la notion de *but* d'un agent et nous étudierons ce qu'un agent peut faire en termes d'opérations pour réaliser son but.

3.1 Argumentation abstraite

Considérons un ensemble Arg de symboles (notés par des lettres minuscules), représentant un ensemble d'arguments.

L'ensemble Arg, ainsi qu'une relation $R \subseteq Arg \times Arg$, nous permettent de définir l'ensemble des arguments possibles avec leurs interactions, ce que nous appelons *univers de référence*. Plus précisément, Arg représente un ensemble potentiellement infini d'arguments utilisables dans un domaine particulier (*e.g.* si le domaine est une base de connaissances alors Arg est l'ensemble de tous les arguments qui peuvent être construits à partir des formules de la base). Il est également possible, à l'instar de l'exemple ci-dessous, de supposer que Arg est fourni explicitement.

Exemple 1 *Lors d'une audience de tribunal concernant un prévenu (M. X), de multiples arguments peuvent être mis en jeu pour déterminer sa culpabilité. La table 1 présente cet ensemble d'arguments i.e., l'ensemble Arg. La relation R est représentée dans le graphe de la figure 1.*

x_0	*M. X n'est pas coupable d'homicide volontaire avec préméditation sur la personne de M^{me} X, sa femme.*
x_1	*M. X est coupable d'homicide volontaire avec préméditation sur la personne de M^{me} X, sa femme.*
x_2	*L'accusé a un alibi, sa secrétaire ayant juré sur l'honneur qu'elle l'avait vu à l'heure du crime.*
x_3	*La personnalité effacée et influençable de la secrétaire ne peut que mettre en doute la véracité de ses propos.*
x_4	*M. X aime si fort sa femme qu'il l'a demandée en mariage une deuxième fois. Or, un homme qui aime sa femme ne saurait en être le meurtrier.*
x_5	*L'accusé est un homme connu pour être vénal et son "amour" pour une femme très riche ne pourrait être qu'appât du gain.*
x_6	*L'accusé n'aurait eu aucun intérêt à tuer sa femme, puisqu'il n'était pas le bénéficiaire de l'énorme assurance vie contractée par celle-ci.*

TABLE 1 – *Arguments mis en jeu lors de l'audience de tribunal.*

Nous proposons une variante de la définition d'un système d'argumentation au sens de [8] qui prend en compte l'univers de référence.

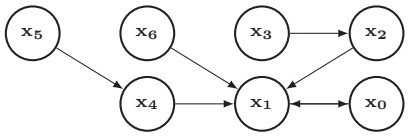


FIGURE 1 – *Illustration de l'univers de référence de l'audience.*

Définition 1 *Un* système d'argumentation sur l'univers $\langle Arg, R\rangle$ *est un sous-graphe partiel de* $\langle Arg, R\rangle$*, c'est-à-dire une paire* $\langle A, R_A\rangle$*, où* $A \subseteq Arg$ *est un ensemble non vide fini d'arguments et* $R_A \subseteq R \cap A \times A$ *est appelée* relation d'attaque.
Soit $a, b \in A$*,* aR_Ab *signifie que* a *attaque* b*.*
$\langle A, R_A\rangle$ *est représenté par un* graphe d'argumentation G *dont les sommets et les arcs correspondent respectivement aux arguments et aux attaques.*
Si x *est un argument,* $x \in G$ *est un abus de langage signifiant* $x \in A$*.*

Dans la suite de ce travail, étant donné un univers $\langle Arg, R\rangle$, $G_k = \langle A_k, R_{A_k}\rangle$ désigne le système d'argumentation de l'agent k sur cet univers, ce qui représente la partie de l'univers de référence connue par k.

Exemple 1 (suite) *Le procureur ne connaît pas tous les arguments présents dans l'univers (représenté dans la figure 1), la figure 2 illustre les arguments et les attaques qu'il connaît (*G_{proc}*).*

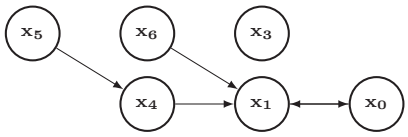


FIGURE 2 – *Système d'argumentation du procureur (*G_{proc}*).*

Les ensembles acceptables d'arguments ("extensions") sont déterminés à l'aide de sémantiques dont la définition repose essentiellement sur les notions suivantes :

Définition 2 *Étant donné un système d'argumentation* $\langle A, R_A\rangle$*, soit* $a \in A$ *et* $S \subseteq A$
– S *attaque* a *ssi*[2] $\exists x \in S$ *tel que* xR_Aa*.*
– S *est* sans conflit *ssi il n'existe pas* $a, b \in S$ *tels que* aR_Ab*.*
– S défend *un argument* a *ssi* S *attaque tout argument attaquant* a*. L'ensemble des arguments défendus par* S *est noté* $\mathcal{F}(S)$*;* $\mathcal{F}$ *est appelée la* fonction caractéristique *de* $\langle A, R_A\rangle$*. Plus généralement,* S défend indirectement a *ssi* $a \in \bigcup_{i\geq 1} \mathcal{F}^i(S)$.
– S *est un ensemble* admissible *ssi il est à la fois sans conflit et défend tous ses éléments.*

L'ensemble des extensions de $\langle A, R_A\rangle$ sous une sémantique donnée est noté $\mathbf{E}$ (avec $\mathcal{E}_1, \ldots, \mathcal{E}_n$ dénotant les extensions). Dans cet article, nous ne nous intéressons qu'à une seule sémantique parmi celles proposées par [8] :

Définition 3 (Sémantique basique) *Soit* $\mathcal{E} \subseteq A$*,* $\mathcal{E}$ *est l'unique* extension basique *ssi* $\mathcal{E}$ *est le plus petit point fixe (par rapport à* $\subseteq$*) de la fonction caractéristique* $\mathcal{F}$*.*

Exemple 1 (suite) *L'extension basique représentant les arguments acceptables pour le procureur est* $\{x_0, x_3, x_5, x_6\}$.

Par ailleurs, le statut d'un argument est fonction de sa présence dans les extensions de la sémantique choisie. En l'occurrence, un argument est "accepté" s'il apparaît dans l'extension basique et "rejeté" sinon.

Ainsi, dans le cas du procureur, x_0 est accepté et x_1 rejeté. Mais, bien qu'il ne possède pas les arguments pour prouver indubitablement que le prévenu est coupable, le procureur souhaite que x_1 soit accepté chez les jurés. La section suivante va nous permettre de voir comment le procureur peut agir sur le système des jurés afin de faire accepter x_1.

3.2 Changement en argumentation

Selon [7], un changement élémentaire est soit l'ajout/retrait d'un argument avec un ensemble d'attaques le concernant, soit l'ajout/retrait d'une attaque. Afin d'être plus concis, nous considérons uniquement dans cet article les opérations concernant l'ajout ou le retrait d'un argument.

Ainsi, nous raffinons la notion d'opération élémentaire au sens de [7] en quatre points : nous définissons sa *syntaxe* ; puis nous la ramenons à un agent pour déterminer si une opération est

2. si et seulement si

autorisée ou non pour celui-ci, c'est-à-dire s'il connaît les éléments mis en jeu dans l'opération. Nous prenons ensuite en compte la cible pour déterminer si l'opération est *exécutable* : l'ajout (respectivement le retrait) d'argument n'est exécutable que si cet argument est absent (respectivement présent) dans le système cible. Enfin, nous définissons l'*impact* d'une opération sur un système d'argumentation.

La restriction à des opérations élémentaires ne fait pas perdre en généralité puisqu'une opération quelconque peut se ramener à une séquence d'opérations élémentaires, appelée *programme* dans la définition 5. Par ailleurs, nous supposons dans ce travail que tous les systèmes considérés sont relatifs au même univers $\langle Arg, R\rangle$.

Définition 4 *Soit k un agent, $G_k = \langle A_k, R_{A_k}\rangle$ son système d'argumentation et $G = \langle A, R_A\rangle$ un système d'argumentation quelconque.*
- *Une* opération élémentaire *est un triplet $o = \langle op, arg, att\rangle$ où $op \in \{\oplus, \ominus\}$, $arg \subseteq Arg$, $att \subseteq R$ et*
  - *soit $op = \oplus$ avec $|arg| = 1$ et $\forall (x, y) \in att$, $(x \neq y)$ et $(x \in arg$ ou $y \in arg)$,*
  - *soit $op = \ominus$ avec $|arg| = 1$ et $att = \varnothing$.*
- *Une opération élémentaire $\langle op, arg, att\rangle$ est* autorisée pour *k ssi $arg \subseteq A_k$ et $att \subseteq R_{A_k}$* [3].
- *Une opération* exécutable par k sur G *est une opération élémentaire $\langle op, arg, att\rangle$ autorisée pour k telle que :*
  - *si $op = \oplus$, alors $arg \not\subset A$ et $\forall (x, y) \in att$, $(x \in A$ ou $y \in A)$,*
  - *si $op = \ominus$, alors $arg \subseteq A$.*
- *Une opération $o = \langle op, arg, att\rangle$ exécutable par k sur G fournit un nouveau système d'argumentation $G' = o(G) = \langle A', R_{A'}\rangle$ tel que :*
  - *si $op = \oplus$ alors $G' = \langle A \cup arg, R_A \cup att\rangle$,*
  - *si $op = \ominus$ alors $G' = \langle A \setminus arg, R_A \setminus \{(x, y) \in R_A | x \in arg$ ou $y \in arg\}\rangle$.*

Exemple 1 (suite) *À partir de l'univers de référence représenté par la figure 1, voici une liste non exhaustive d'opérations* élémentaires *:*
- $\langle \oplus, \{x_2\}, \{(x_2, x_1)\}\rangle$,
- $\langle \oplus, \{x_2\}, \{(x_2, x_1), (x_3, x_2)\}\rangle$,
- $\langle \oplus, \{x_6\}, \varnothing\rangle$,
- $\langle \oplus, \{x_6\}, \{(x_6, x_1)\}\rangle$.
- $\langle \ominus, \{x_4\}, \varnothing\rangle$,
- $\langle \ominus, \{x_2\}, \varnothing\rangle$.

Parmi ces quelques opérations élémentaires, le procureur n'est pas autorisé *à utiliser celles concernant des arguments ou des attaques qu'il ne connaît pas. Ainsi, il ne pourra pas utiliser les opérations suivantes :*
- $\langle \oplus, \{x_2\}, \{(x_2, x_1)\}\rangle$,
- $\langle \oplus, \{x_2\}, \{(x_2, x_1), (x_3, x_2)\}\rangle$,
- $\langle \ominus, \{x_2\}, \varnothing\rangle$.

Supposons que $G_{jurés}$, donné par la figure 3, est le système des jurés (et donc la cible du procureur).

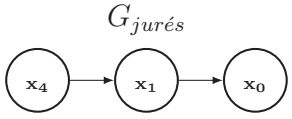


FIGURE 3 – *Système d'argumentation des jurés*

Ce dernier dispose donc, entre autres, des opérations exécutables *suivantes sur $G_{jurés}$:*
- $\langle \ominus, \{x_4\}, \varnothing\rangle$,
- $\langle \oplus, \{x_5\}, \{(x_5, x_4)\}\rangle$,
- $\langle \oplus, \{x_6\}, \varnothing\rangle$,
- $\langle \oplus, \{x_6\}, \{(x_6, x_1)\}\rangle$.

*Et, enfin, si l'on considère l'*impact *de la seconde de ces opérations sur le système des jurés, ce dernier devient alors $\langle \{x_0, x_1, x_4, x_5\}, \{(x_1, x_0), (x_4, x_1), (x_5, x_4)\}\rangle$:*

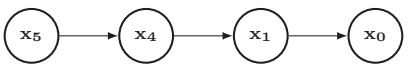


FIGURE 4 – *$G_{jurés}$ après $\langle \oplus, \{x_5\}, \{(x_5, x_4)\}\rangle$*

Nous considérons ci-dessous les suites d'opérations exécutables par un agent sur un système d'argumentation, nommées *programmes*, donnant la possibilité à un agent d'effectuer plusieurs opérations élémentaires en séquence.

Définition 5 *Soit k un agent et G un système d'argumentation quelconque. Un* programme *p exécutable par k sur G est une suite ordonnée finie de m opérations $(o_1, \cdots, o_m)$ telle que :*
- *$m = 1$: o_1 est exécutable par k sur G. On a alors $p(G) = o_1(G)$.*
- *$m > 1$: $(o_1, \cdots, o_{m-1})$ est un programme p' exécutable par k sur G tel que $p'(G) = G'$ et o_m est exécutable par k sur G'. On a alors $p(G) = o_m(G')$.*
- *Par extension, une suite vide est aussi un programme. On a alors $p(G) = G$.*

3. Dans le cas d'un ajout d'argument, il peut n'être fourni qu'une partie des attaques connues ; il est donc possible pour un agent d'effectuer un "mensonge par omission", par exemple pour des raisons stratégiques. L'agent n'est par contre pas autorisé à fournir des arguments ou des attaques inconnus ; il ne peut donc mentir de manière "active" ; ceci pourra être l'objet de travaux futurs.

Il reste maintenant à définir ce que pourrait être un programme réalisant un but précis.

3.3 Buts et programmes

Comme nous l'avons déjà souligné, un agent a la possibilité d'agir sur un système cible pour atteindre des objectifs. Ces objectifs seront formellement traduits par des buts. Un but est représenté par une expression d'un langage qui exprime des conditions à satisfaire dans des systèmes d'argumentation (celui de la cible et éventuellement celui de l'agent). Pour représenter ces buts, nous utiliserons les symboles apparaissant dans la typologie des propriétés du changement présentée dans [3], notamment :
– les arguments (x, y, z, etc.),
– les extensions ($\mathcal{E}_i$ et $\mathcal{E}'_i$),
– les ensembles d'extensions ($\mathbf{E}$ et $\mathbf{E}'$) ainsi que leur cardinal ($|\mathbf{E}|$ et $|\mathbf{E}'|$),
– les ensembles de toutes les extensions contenant un argument particulier x ($\mathbf{E}_x$ et $\mathbf{E}'_x$) et leur cardinal ($|\mathbf{E}_x|$ et $|\mathbf{E}'_x|$),
– les opérateurs de comparaison classique ($=$, $<$, $>$, etc.),
– les quantificateurs $\forall$ et $\exists$,
– l'appartenance ($\in$) et l'inclusion ($\subseteq$),
– l'union ($\cup$) et l'intersection ($\cap$) d'ensembles,
– les opérateurs logiques classiques ($\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\neg$).

Nous noterons b_k le but de l'agent k.

Exemple 1 (suite) *Le but b_{proc} du procureur peut être représenté par $(x_1 \in \mathcal{E}')$ avec $\mathcal{E}'$ représentant l'extension basique dans le système des jurés.*

Maintenant que le but du procureur peut être formellement représenté, nous pouvons donner la notion de programme réalisant un but.

Définition 6 *Soit k un agent, b_k son but et G un système d'argumentation quelconque. Un* programme p de k sur G réalisant b_k *est un programme exécutable par k sur G tel que $p(G) = G'$ et b_k est satisfait dans G'* [4].

Exemple 1 (suite) *Si l'on considère le but b_{proc} représenté par $(x_1 \in \mathcal{E}')$, les programmes $(\langle \oplus, \{x_5\}, \{(x_5, x_4)\}\rangle)$ et $(\langle \ominus, \{x_4\}, \varnothing\rangle)$ du procureur sur $G_{jurés}$ réalisent b_{proc}.*

4. Ou dans (G, G') si le but exprime une condition sur les deux systèmes.

Comment déterminer les opérations qui réalisent le but d'un agent ? C'est ce que nous allons étudier dans la section suivante.

3.4 Caractérisations du changement

Une approche "naïve" est de calculer, pour chaque opération exécutable, l'ensemble des extensions du système cible modifié et de vérifier que le but est bien réalisé. Une autre approche plus efficace utilise les *caractérisations du changement* qui ont été étudiées dans des travaux tels que [7] et [2]. Une caractérisation est une propriété donnant des conditions nécessaires et/ou suffisantes permettant de réaliser un but particulier pour un type d'opération et une sémantique donnés.

Nous donnons deux exemples de caractérisations :

Caractérisation 1 ([3]) *Lors de l'ajout d'un argument z sous la sémantique basique, si z n'est pas attaqué par G et z défend indirectement x et $x \notin \mathcal{E}$, alors $x \in \mathcal{E}'$.*

Cette caractérisation concerne un but de type *appartenance forcée* [5] d'un argument x. Elle précise également l'opération concernée : il s'agit de l'ajout d'un argument z. Ainsi, grâce à cette caractérisation, nous savons (sans avoir besoin de tout recalculer) que si une opération ajoute un argument z sous la sémantique basique, tel qu'il n'est pas attaqué et qu'il défend indirectement un autre argument x qui n'était pas accepté, alors x deviendra accepté.

Caractérisation 2 ([3]) *Lors du retrait d'un argument z sous la sémantique basique, si $\mathcal{E} \neq \varnothing$ et z est attaqué par G, alors $\mathcal{E}' \neq \varnothing$.*

Le but, de type *extension non vide*, porte ici sur l'évolution de l'extension. Cette caractérisation nous permet de savoir, sans tout recalculer, que si une opération retire un argument z alors que ce dernier est attaqué par au moins un argument de G et que l'extension n'est pas vide avant le changement, alors l'extension obtenue après le changement ne sera pas vide. Ceci peut, par exemple, être utile lorsque l'on veut s'assurer que la discussion ne se révèlera pas vaine.

5. L'appartenance forcée ("enforcement", voir [1]) consiste à faire en sorte qu'un argument qui était rejeté devienne accepté. Ainsi, on force l'appartenance d'un argument à une extension.

Ceci conclut notre cadre théorique. La section suivante présente l'outil associé.

4 Présentation de l'outil

Cet outil s'organise autour de deux modules spécifiques : un gestionnaire de systèmes d'argumentation, et un moteur de calcul des opérations de changement. Les sorties du premier module servent d'entrée au deuxième module.

4.1 Le gestionnaire de systèmes d'argumentation

Nous présentons ici la facette argumentative "pure" du programme, qui concerne la création de différents systèmes d'argumentation. Nous avons implémenté la notion de système d'argumentation, et lui avons adjoint la possibilité de calculer ses propres extensions. Ce module est implémenté en utilisant un langage à objet (*Python 2.7*) pour la flexibilité et la facilité que cela procure dans l'implémentation des notions utilisées.

Ainsi, pour créer un système d'argumentation, il faut fournir :
– un ensemble d'arguments,
– un ensemble d'attaques.
Ce module gère la cohérence des données fournies en vérifiant par exemple qu'un argument n'apparaît qu'une seule fois, ou qu'une attaque n'existe que si les arguments concernés existent aussi. Ce module renvoie des informations concernant les systèmes d'argumentation et lance si nécessaire le calcul des extensions relativement à une sémantique pouvant être fournie en paramètre.

Conformément à notre cadre théorique, l'outil permet donc de créer deux systèmes d'argumentation, celui de l'agent et celui de sa cible. Cette création s'effectue en fournissant la liste des arguments et des attaques de ces deux systèmes. De plus, la sémantique utilisée dans le système cible est aussi précisée (elle servira à vérifier la satisfaction des buts de l'agent).

Les systèmes d'argumentation de l'agent et de la cible, ainsi que les extensions de ce dernier sont alors transmis au deuxième module.

4.2 Le moteur de calcul

Ce deuxième module est destiné à calculer des "opérations de changement". Plus précisément, il permet de répondre à la question *"Quelles sont les opérations exécutables par l'agent sur le système cible réalisant son but ?"*.

Le moteur de calcul prend en entrée les systèmes d'argumentation de l'agent et de la cible, ainsi que l'ensemble d'extensions de la cible pour une sémantique donnée. En plus de cela, il est nécessaire de lui fournir :
– un but correspondant à ce que l'agent cherche à satisfaire dans le système cible,
– une base de caractérisations permettant de vérifier si une opération réalise le but fourni.
Notons que l'utilisateur peut filtrer les résultats en forçant le type d'opération attendu, il peut ne s'intéresser qu'aux ajouts d'arguments par exemple.

Le cœur de ce module est une règle qui :
– génère toutes les opérations exécutables par l'agent sur la cible et réalisant le but,
– et produit, pour chacune des opérations, la caractérisation justifiant ce résultat.
Cette règle comporte deux parties, l'une se chargeant de construire les opérations, et l'autre vérifiant que l'opération correspond aux *desiderata* de l'agent.

Une vision synthétique de l'outil, et donc de l'articulation entre les deux modules de l'outil, est donnée par la figure 5.

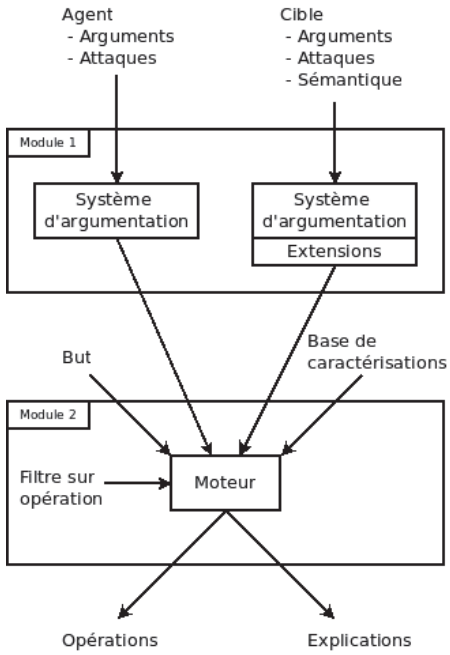


FIGURE 5 – *Architecture schématique de l'outil.*

Notons que nous utilisons dans ce second module un langage de programmation logique

(*Prolog*) pour deux raisons primordiales : les caractérisations se traduisent naturellement en règles logiques, ce qui facilite l'implémentation, et le mécanisme d'unification nous permet de générer et de contraindre facilement des opérations à partir des systèmes d'argumentation et du but de l'agent.

Construction des opérations La construction des opérations découle directement de la définition 4. Ainsi, nous générons directement une opération exécutable, puis nous calculons son impact. Ceci permet d'éviter de considérer des opérations non autorisées ou non exécutables et ainsi d'optimiser le temps de calcul.

Vérification des opérations Lorsqu'une opération est générée par les règles de construction, celle-ci est traitée grâce aux caractérisations (par exemple, les caractérisations 1 ou 2). Ainsi, pour une opération donnée, s'il existe une caractérisation correspondant au type de l'opération et au but recherché par l'agent et si les conditions de cette caractérisation sont satisfaites, alors cette opération sera fournie à l'utilisateur avec la caractérisation correspondante à titre d'explication.

Notons que l'outil ne traite pour l'instant que des programmes réduits à une opération élémentaire.

4.3 Application à l'exemple

Nous illustrons l'utilisation de l'outil sur l'exemple de l'audience. Pour ce faire, considérons la situation suivante : le procureur doit faire en sorte que l'argument x_1 soit accepté. Il a à sa disposition les arguments présentés dans la figure 2 et doit modifier le système d'argumentation des jurés, représenté par la figure 3.

Le procureur doit donc tout d'abord fournir à l'outil les données nécessaires, à savoir :
- l'ensemble des arguments et des attaques qu'il connaît : $\{x_0, x_1, x_3, x_4, x_5, x_6\}$ et $\{(x_0, x_1), (x_1, x_0), (x_4, x_1), (x_5, x_4), (x_6, x_1)\}$,
- l'ensemble des arguments et attaques connus par les jurés (le système cible) : $(\{x_0, x_1, x_4\}, \{(x_1, x_0), (x_4, x_1)\})$,
- la sémantique utilisée par les jurés, que nous supposons ici être la sémantique basique.

L'outil déduira de ces données l'ensemble des extensions pour le système des jurés ($\mathbf{E} = \{\{x_0, x_4\}\}$).

Le procureur doit aussi préciser le but qu'il souhaite voir satisfait :

"x_1 doit appartenir à l'extension basique."

Et enfin il doit fournir une base de caractérisations. Nous supposerons pour les besoins de cet exemple que cette base se résume à l'unique caractérisation 1.

L'outil génère alors les opérations exécutables par le procureur sur le système des jurés puis calcule leurs impacts. L'outil génère par exemple les opérations :
- $\langle \ominus, \{x_4\}, \varnothing \rangle$, qui passe avec succès les étapes de construction et dont l'impact est le système ayant pour ensemble d'arguments $\{x_0, x_1\}$ et pour ensemble d'attaques $\{(x_1, x_0)\}$,
- $\langle \oplus, \{x_5\}, \{(x_5, x_4)\} \rangle$, qui passe également avec succès les étapes de construction et dont l'impact est le système $(\{x_0, x_1, x_4, x_5\}, \{(x_1, x_0), (x_4, x_1), (x_5, x_4)\})$.

Notons que les opérations concernant des arguments ou attaques inconnus de l'agent, ou non exécutables sur le système cible, ne sont pas générées par l'outil. Ainsi, l'opération $\langle \ominus, \{x_6\}, \varnothing \rangle$ n'est pas générée car l'argument x_6 n'est pas présent dans le système des jurés. Ceci assure le caractère fini du processus.

Les opérations générées sont en même temps examinées au travers des caractérisations. Si une opération ne correspond à aucune des caractérisations, elle est rejetée ; si elle est concernée par une ou plusieurs caractérisations, alors l'outil renvoie tous les couples (opération, caractérisation).

Dans notre exemple, l'opération $\langle \ominus, \{x_4\}, \varnothing \rangle$ est rejetée car son type (retrait d'un argument) ne correspond pas à celui spécifié dans l'unique caractérisation disponible. Par contre, le type de l'opération $\langle \oplus, \{x_5\}, \{(x_5, x_4)\} \rangle$ correspond ; l'outil doit donc vérifier que cette dernière satisfait les contraintes spécifiées dans la caractérisation, à savoir que z, en l'occurrence apparié à l'argument x_5, n'est pas attaqué et qu'il doit défendre indirectement x (ce dernier apparié à x_1) tel que x n'appartient pas à l'extension.[6] Ces conditions étant vérifiées, l'opération réalise effectivement le but du procureur.

6. Les concepts nécessaires aux caractérisations ont été implémentés. Nous comptons ainsi parmi eux les concepts d'attaque indirecte par un argument, de défense directe et indirecte par un argument, d'attaque et de défense d'ensembles, etc.

5 Expérimentations

L'objectif de notre outil est de trouver un programme (au sens de la définition 5) réalisant un but. Rappelons que nous nous restreignons dans un premier temps aux *programmes ne contenant qu'une seule opération*. Le travail aurait pu être fait directement en calculant l'impact d'une opération puis l'ensemble des extensions du graphe obtenu. Mais l'outil est conçu dans le but d'éviter de recalculer des extensions, ce qui peut être très coûteux. En effet, si calculer l'impact d'une opération est aisé en termes d'arguments et d'attaques (car n'impliquant que des traitements "simples" d'ensembles), il est bien plus lourd de recalculer les ensembles d'extensions ([9]). Notre idée est de générer des opérations exécutables et de les "tester" grâce aux caractérisations, plutôt que de calculer les extensions pour chacun des systèmes résultants puis de vérifier que le but sélectionné est satisfait. Le gain en termes de temps et d'espace reste encore à évaluer, ce que nous comptons faire dans des travaux futurs.

Avant d'analyser les résultats produits par l'outil (section 5.2), nous présentons, en section 5.1, les protocoles expérimentaux utilisés, qui seront ensuite critiqués en section 5.3.

5.1 Protocoles expérimentaux

Les deux protocoles expérimentaux mis en place visent à vérifier en premier lieu la correction des résultats, et dans un deuxième temps à mettre en lumière des lacunes au niveau des caractérisations.

Hypothèse d'inclusion. Ces deux protocoles supposent que le système cible est un sous-graphe partiel du système de l'agent. Cette hypothèse paraît naturelle dans le cas de l'exemple de l'audience : le procureur et l'avocat entendent publiquement ce qui a été dit au cours de l'audience. Ils sont donc au courant de tous les arguments présents dans le système d'argumentation des jurés (leur cible)[7].

Génération aléatoire. Ce protocole vise à générer aléatoirement deux systèmes (un pour l'agent et un pour la cible en respectant l'hypothèse d'inclusion) et un but à satisfaire :
- un ensemble d'arguments de taille n est créé, avec n ayant une valeur aléatoire entre 2 et 20,
- un ensemble d'attaques entre ces arguments de taille nb_att est créé, avec nb_att ayant une valeur aléatoire entre $n * 0.5$ et $n * 1.5$.

Ceci constitue le premier système. Il va servir de base pour créer un sous-graphe partiel représentant la cible :
- deux nombres $suppr\_arg$ et $suppr\_att$ sont générés aléatoirement, avec $suppr\_arg$ ayant une valeur entre 0 et $n-1$ et $suppr\_att$ ayant une valeur entre 0 et $nb\_att - 1$,
- $suppr\_arg$ arguments sont retirés aléatoirement de l'ensemble d'arguments, et pour chaque argument retiré nous soustrayons à $suppr\_att$ le nombre d'attaques le concernant,
- $suppr\_att$ attaques sont retirées aléatoirement.

Ceci nous permet d'obtenir le deuxième système.

Et enfin, un but est choisi aléatoirement parmi $x \in \mathcal{E}'$, $x \notin \mathcal{E}'$, $z \in \mathcal{E}'$, $\mathcal{E} = \mathcal{E}'$, $\mathcal{E} \subseteq \mathcal{E}'$, $\mathcal{E} \subset \mathcal{E}'$, $\mathcal{E}' \subseteq \mathcal{E}$, $\mathcal{E}' \subset \mathcal{E}$, $\mathcal{E}' = \varnothing$ et $\mathcal{E}' \neq \varnothing$, où x appartient au système cible, z appartient au système de l'agent et $\mathcal{E}$ (resp. $\mathcal{E}'$) est l'extension basique de la cible avant (resp. après) le changement.

La génération d'opérations est alors lancée à partir de ces deux systèmes et du but (voir les résultats obtenus en section 5.2).

Génération systématique. Bien que la génération aléatoire soit efficace, elle peut laisser de côté des cas intéressants. Pour pallier ce problème, nous avons mis en place un autre protocole expérimental consistant à tester exhaustivement une liste de couples de systèmes.

Ainsi, pour n arguments, nous créons tous les couples de systèmes tels que :
- le premier système a un ensemble d'arguments arg_agent (numérotés de 1 à n) et un ensemble d'attaques att_agent (dont le nombre va de 0 à $n * (n-1)$).
- le deuxième système a un ensemble d'arguments arg_cible variant parmi tous les sous-ensembles possibles de arg_agent, et un ensemble d'attaques variant parmi tous les sous-ensembles possibles de att_agent restreint aux arguments de arg_cible.

Nous procédons alors à la génération d'opérations pour un but particulier donné (ici, on a

7. Le non respect de cette hypothèse signifierait que l'agent pourrait ne pas être d'accord avec la validité de certains arguments ou attaques entendus. On se pose là la question complexe de la différentiation entre connaissance d'un argument ou d'une attaque et croyance en sa validité. Cette question mériterait un approfondissement qui sort du cadre de cet article.

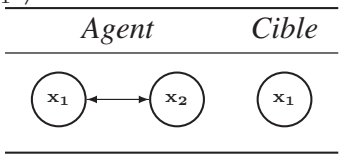
testé le but d'appartenance forcée pour chaque argument du système cible).

Notre protocole calcule l'extension basique du système résultant de l'opération. Ceci nous permet de constater les problèmes de "couverture" de l'outil, c'est-à-dire les cas où aucune opération n'est trouvée pour réaliser le but alors qu'il en existe une. Ces résultats de couverture sont présentés dans la section suivante.

5.2 Résultats

Pour la génération aléatoire : Pour chaque couple de systèmes généré (en tout cinq millions), le premier protocole expérimental affiche des informations concernant la génération des opérations exécutables réalisant un but choisi aléatoirement. L'exemple 2 propose un court extrait montrant un cas de réussite de génération d'opérations.

Exemple 2 *Considérons les systèmes suivants et le but $x_1 \notin \mathcal{E}'$:*



L'outil génère deux opérations : $\langle \oplus, \{x_2\}, \{(x_2, x_1)\}\rangle$ et $\langle \oplus, \{x_2\}, \{(x_2, x_1), (x_1, x_2)\}\rangle$ avec les caractérisations correspondantes (ici, il s'agit deux fois de la même – celle donnée par la proposition 16 de [3]).

Par ailleurs, le protocole propose également un récapitulatif des résultats, pour chacun des buts, en détaillant le nombre de cas où une solution est trouvée. La table 2 montre un exemple de résultats.

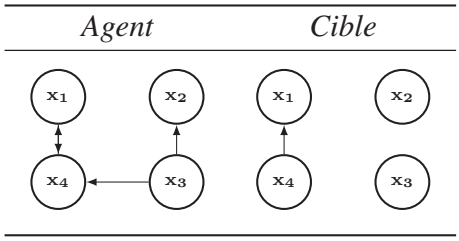
Selon le but considéré, l'outil trouve plus ou moins souvent des solutions. Plus précisément, il existe deux causes possibles amenant l'outil à ne pas trouver d'opérations exécutables :

- soit *il n'existe pas d'opération exécutable* réalisant le but cherché ; notons que ce cas inclut celui où il n'est pas possible de réaliser ce but avec une seule opération (*cf.* exemple 3).
- soit *il manque une caractérisation* à l'outil permettant de trouver au moins une opération (*cf.* exemple 4).

But	**Nb. de cas testés**	**Ensemble de sol.**		**%**
		Non vide	**Vide**	
$x \in \mathcal{E}'$	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

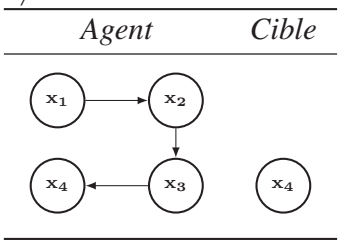
TABLE 2 – *Récapitulatif de résultats pour le protocole de génération aléatoire. Pour chaque but, la deuxième colonne indique le nombre total de couples de systèmes d'argumentation testés par l'outil. La troisième colonne (resp. quatrième colonne) indique le nombre de fois où l'outil a renvoyé un ensemble de solutions non vide (resp. vide) pour un couple de systèmes. Enfin, la cinquième colonne indique le pourcentage des cas où l'outil a renvoyé un ensemble de solutions non vide par rapport au total des cas testés pour un but particulier.*

Exemple 3 *Considérons les systèmes suivants et le but $\mathcal{E}' = \varnothing$:*



L'outil ne génère aucune opération car il est impossible d'avoir une opération exécutable réalisant le but recherché : d'une part, il n'existe pas d'argument attaquant en même temps x_2, x_3 et x_4 et étant attaqué par au moins l'un d'entre eux, et, d'autre part, il n'est pas possible de retirer tous les arguments en une seule opération.

Exemple 4 *Considérons les systèmes suivants et le but $\mathcal{E}' \neq \varnothing$:*



L'outil ne génère aucune opération, et pourtant, ne rien faire (entre autres choses) suffirait à réaliser le but ; il manque donc à l'outil au moins une caractérisation affirmant que lorsque l'on ne fait rien, les extensions ne changent pas (la

possibilité de ne rien faire n'avait pas été envisagée dans [3]).

L'ensemble des solutions fourni par l'outil peut donc être vide pour des raisons très différentes mais qui ne sont pas distinguées par ce protocole. C'est pour mettre en évidence des exemples relevant du deuxième cas (montrant des lacunes dans les caractérisations) que nous avons mis en place le deuxième protocole expérimental, dont nous donnons maintenant les résultats.

Pour la génération systématique : En considérant un but particulier, et en traitant tous les cas possibles, ce protocole nous a permis de nous concentrer sur les cas où le système ne trouve pas de solution et ainsi de pouvoir détecter les manques dans notre base de caractérisations.

Nous avons ainsi pu compléter notre base de caractérisations de telle sorte qu'actuellement la génération automatique pour $n = 3$ ou $= 4$ ne détecte plus aucun manque de caractérisation pour le but correspondant à l'appartenance forcée d'un argument à l'extension basique.

5.3 Critiques des protocoles

Notre protocole de génération aléatoire a permis de tester un grand nombre d'exemples (avec différents buts). Cependant, nous n'avons pas étudié la pertinence du choix de ces exemples, *i.e.* les choix effectués ne sont pas forcément représentatifs de tous les cas.

Quant à la génération systématique, elle est très coûteuse : pour une génération concernant n arguments, il y a $2^{n*(n-1)}$ systèmes possibles pour l'agent et dans le pire des cas $2^{n*(n-1)} + (n * 2^{(n-1)*(n-2)}) + \cdots + (n * 2^0)$ cas possibles pour la cible. Le nombre de couples possibles total est de l'ordre de $2^{n*(n-1)} * (2^{n*(n-1)} + (n * 2^{(n-1)*(n-2)}) + \cdots + (n * 2^0))$. Ceci est tolérable pour un petit nombre d'arguments. Par exemple, pour $n = 3$, il y a $2^6 * (2^6 + 3 * 2^2 + 3) = 5056$ couples possibles. Mais cela devient rédhibitoire à partir de cinq arguments. Ce problème est en partie dû à la génération inutile de couples de graphes isomorphes.

Le deuxième inconvénient de ce protocole est qu'il dépend du choix du but. Il faut donc redéfinir un protocole à chaque nouveau but.

6 Discussion et conclusion

Nous avons présenté un cadre théorique et un outil permettant de trouver une opération de changement qui réalise un but donné sur un système d'argumentation cible à partir d'arguments ou d'attaques prises dans un système d'argumentation "source" (représentant les connaissances d'un agent). Nous avons étudié le comportement de cet outil au moyen de deux protocoles expérimentaux. Néanmoins les expérimentations ne sont pas terminées, puisque nos protocoles ne permettent pas d'évaluer exhaustivement tous les exemples possibles.

Les perspectives de ces travaux sont prometteuses et peuvent se décliner suivant trois axes. Le premier axe concerne l'outil lui-même :

– Nous nous pencherons tout d'abord sur la complexité. Puisqu'un calcul des extensions après changement est très coûteux (malgré les progrès faits dans [9]), notre objectif est de montrer que notre approche l'est moins. Cela nécessitera la détermination de la complexité de l'algorithme du moteur de calcul de l'opération (écrit en Prolog), une des difficultés étant d'intégrer à ce calcul la complexité liée à la vérification de l'applicabilité de certaines caractérisations [8].
– Par ailleurs, nous prévoyons de lever la restriction de ne considérer qu'une seule opération élémentaire. Intégrer des séquences d'opérations permettra d'effectuer des modifications plus complexes et donc de trouver plus de solutions.
– Nous avons évoqué l'utilisation de l'outil pour repérer des lacunes de caractérisation ; il pourrait être intéressant de développer un analyseur plus fin qui détecterait toutes les opérations exécutables que l'outil ne trouve pas.

Le deuxième axe concerne les expérimentations réalisées grâce à l'outil et ses applications :

– La poursuite d'expérimentations, telles celles présentées dans cet article, est importante puisqu'elle permet de créer des "benchmarks" dans le domaine de l'argumentation.
– Il pourrait être intéressant de considérer des cas réels d'argumentation pour établir d'autres "benchmarks", plus concrets. En particulier, les débats en ligne (sur des réseaux sociaux par exemple) semblent bien se prêter à cet exercice.

8. En effet certaines caractérisations utilisent des notions difficiles à calculer : par exemple la défense indirecte d'un argument par un ensemble.

– Le point précédent constitue également une piste d'applications, permettant éventuellement à un utilisateur d'être guidé dans son choix d'arguments à avancer lors de débats en ligne.

Le troisième et dernier axe concerne les avancées théoriques :

– Lors de l'étude de nos protocoles d'expérimentation, nous avons posé l'hypothèse d'inclusion du système cible dans le système d'argumentation de l'agent. Cette question de la connaissance complète ou non du système sur lequel l'agent veut agir et de la mise à jour par l'agent de son propre système mérite d'être étudiée et approfondie.
– Il paraît nécessaire d'étendre notre application du procès de façon à permettre une réelle interaction entre le procureur et l'avocat. De manière plus générale, ceci implique que nous étudiions les changements opérés par un agent sur son propre système lorsque un autre agent effectue une modification du système cible.
– Enfin, nous nous sommes jusque-là limités à un dialogue de persuasion, avec des agents ayant des buts contradictoires. Il pourrait être intéressant de considérer d'autres types de dialogues mettant en jeu une coalition d'agents coopérant pour réaliser un but en commun, par exemple plusieurs avocats tentant d'étoffer une défense.

Références

[1] R. Baumann and G. Brewka. Expanding argumentation frameworks : Enforcing and monotonicity results. In *Proc. of COMMA*, pages 75–86. IOS Press, 2010.

[2] P. Bisquert, C. Cayrol, F. Dupin de Saint-Cyr, and M.-C. Lagasquie-Schiex. Duality between addition and removal. In *Advances on Computational Intelligence*, volume 297 of *Communications in Computer and Information Science*, pages 219–229. Springer, 2012.

[3] P. Bisquert, C. Cayrol, F. Dupin de Saint Cyr, and M.-C. Lagasquie-Schiex. Characterizing change in abstract argumentation systems. Technical report, IRIT, UPS, Toulouse, France, 2013. ftp ://ftp.irit.fr/pub/IRIT/ADRIA/Rapport-IRIT-2013-22.pdf.

[4] P. Bisquert, C. Cayrol, F. Dupin de Saint Cyr Bannay, and M.-C. Lagasquie-Schiex. Changement dans un système d'argumentation : suppression d'un argument. *RIA*, 26(3/2012) :225–254, 2012.

[5] G. Boella, S. Kaci, and L. van der Torre. Dynamics in argumentation with single extensions : Abstraction principles and the grounded extension. In *Proc. of ECSQARU (LNAI 5590)*, pages 107–118, 2009.

[6] 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*, pages 1213–1214, 2009.

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

[8] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2) :321–358, 1995.

[9] B. Liao, L. Jin, and R. C. Koons. Dynamics of argumentation systems : A division-based method. *Artificial Intelligence*, 175(11) :1790 – 1814, 2011.

[10] 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*, pages 359–366. IOS Press, 2010.