Bipolarité en argumentation : acceptabilité et algorithmes

M. Mardi
C. Cayrol
M.C. Lagasquie-Schiex

Octobre 2005

Rapport IRIT/2005-20-R

Résumé

L'argumentation en intelligence artificielle est un processus cognitif qui repose tout d'abord sur la construction d'arguments puis sur la sélection des arguments les plus acceptables en fonction de leurs interactions. Les systèmes d'argumentation les plus représentés sont les systèmes "unipolaires", c'est-à-dire constitués d'une seule interaction : la contrariété qui reflète un aspect "négatif" des interactions d'arguments.

Dans ce rapport, nous souhaitons étendre ce processus d'argumentation en prenant en compte une interaction supplémentaire qui refléterait un aspect "positif" de l'interaction d'arguments. Nous allons donc étudier des systèmes d'argumentation "bipolaires", c'est-à-dire constitués de deux types d'interactions : la contrariété et l'appui.

Le cadre abstrait adéquat pour étudier de tels systèmes bipolaires va être obtenu en partant du cadre formel proposé initialement par Dung pour les systèmes unipolaires, et en l'étendant par la prise en compte d'une relation d'appui.

Dans ce document, nous utiliserons ce cadre pour proposer diverses sémantiques d'acceptabilité engendrées par l'introduction de cette nouvelle interaction. Et nous proposerons des algorithmes d'énumération d'ensembles qui respectent certaines de ces sémantiques.

Table des matières

Table des figures

Liste des Algorithmes

Chapitre 1

Introduction

L'Intelligence Artificielle représente une branche de l'informatique dont le but est de simuler un comportement complexe, intelligent, souvent associé à l'humain. Les plus grands domaines de recherche en Intelligence Artificielle sont l'acquisition et la représentation des connaissances, qui interviennent par exemple dans les processus liés à l'expérience, et les modes de raisonnement permettant la résolution rapide et appropriée d'un problème quelconque.

L'argumentation, en Intelligence Artificielle, est un processus cognitif qui repose sur l'étude des arguments en fonction de leurs interactions. On trouve des applications dans de nombreux domaines comme le traitement de bases de connaissances incohérentes, les domaines d'aide à la décision, à la négociation. L'argumentation est aussi étudiée dans d'autres disciplines comme la philosophie, la psychologie ou la linguistique.

Les arguments et les interactions étudiés dans un processus d'argumentation sont représentés dans un cadre appelé *système d'argumentation* qui, bien souvent, induit un graphe orienté où les nœuds sont les arguments et les arcs sont les interactions. De nombreux travaux ont été effectués sur des systèmes d'argumentation dans lesquels n'est considérée qu'une seule sorte d'interaction : la *contrariété* (appelée aussi *attaque*). En particulier, le cadre de Dung [Dun95] dont l'abstraction permet l'étude de nombreux systèmes formels de raisonnement. Néanmoins, les études récentes (cf. [KP01], [Ver02], [CLS04], [CLS05] et [DS04]) semblent se tourner vers les systèmes d'argumentation *bipolaires*, des systèmes où l'évaluation des arguments dépend de la contrariété et de l'*appui*. Dans ce rapport, nous nous plaçons dans le cadre abstrait proposé par C. Cayrol et M.C. Lagasquie-Schiex [CLS04], lui-même inspiré de celui de Dung.
Le processus d'argumentation consiste à étudier les arguments d'un système d'argumentation afin de déterminer les plus acceptables. Il existe différentes méthodes d'étude et à chaque méthode est associée une sémantique, c'est-à-dire un ensemble de règles et de contraintes. Généralement, on évalue des ensembles d'arguments qui doivent respecter une certaine cohérence.
De nombreuses sémantiques ont été proposées pour les systèmes d'argumentation unipolaires, comme la sémantique *stable* ou *préférée*, mais peu de travaux similaires ont été faits pour les systèmes d'argumentation bipolaires.

L'objectif de ce travail est donc d'étudier et de définir des sémantiques d'acceptabilité utilisables dans un cadre bipolaire et de proposer des algorithmes permettant l'énumération d'ensembles d'arguments qui respectent certaines de ces sémantiques.

Le rapport se décompose en trois grandes parties :
La première partie rassemble la section 2 page 3, où l'on rappelle le cadre unipolaire abstrait proposé par Dung [Dun95], et la section 3 page 7 qui contient les définitions du cadre bipolaire proposé par C. Cayrol et M.C. Lagasquie-Schiex [CLS04].

La deuxième partie rassemble la section 4 page 9, où l'on propose une amélioration de la cohérence donnée par C. Cayrol et M.C. Lagasquie-Schiex [CLS04] et où l'on interprète les notions définies dans un cadre concret, et la section 5 page 19 qui énumère les différentes sémantiques.
Enfin, la dernière partie, la section 6 page 31, est la partie algorithmique décrivant l'énumération des ensembles d'arguments acceptables dans un système d'argumentation bipolaire. Nous ne nous intéressons néanmoins qu'à certaines sémantiques définies dans les parties précédentes.

Chapitre 2

Le système d'argumentation unipolaire de Dung

2.1 Cadre de Dung [Dun95]

Voici le cadre proposé par Dung [Dun95] pour l'argumentation basée sur un seul type d'interaction, l'attaque.

Définition 1 (Système d'argumentation)
Un système d'argumentation *est un couple* $<A,R>$*, où* A *est un ensemble d'arguments et* R *une relation binaire sur* A *représentant une relation de contrariété ou d'attaque.*
Soit a_i *et* $a_j \in A$, $a_i R a_j$ *(ou* $(a_i, a_j) \in R$*) signifiera que* a_j *est contrarié/attaqué par* a_i*. Un système d'argumentation est* bien-fondé *si et seulement s'il n'existe pas de séquence infinie* $a_0, a_1, \ldots, a_n, \ldots$ *telle que* $\forall i$*,* $a_i \in A$ *et* $(a_i, a_{i+1}) \in R$*.*

Dung [Dun95] représente un système d'argumentation par un graphe orienté où chaque couple $(a_i, a_j) \in R$ est représenté graphiquement par un arc du nœud a_i vers le nœud a_j (cf figure 2.1).

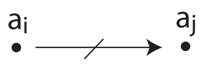


FIG. 2.1 – Représentation graphique d'une attaque

Exemple 1 *Le système d'argumentation*
$<A=\{a, b, c, d, e\}, R=\{(a, b), (b, a), (a, e), (d, e), (b, c), (e, c)\}>$ *est représenté par la figure suivante.*

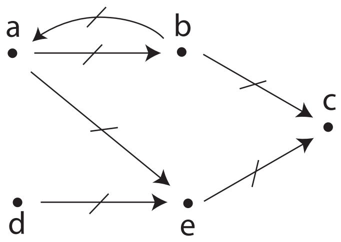


FIG. 2.2 – Exemple 1 : un système d'argumentation unipolaire

2.2 Acceptabilité de Dung [Dun95]

Étant donné un système d'argumentation, un des problèmes importants est la détermination des arguments acceptables pour ce système, c'est-à-dire quels sont les ensembles d'arguments que pourrait garder un agent rationnel. Dung [Dun95] propose différentes sémantiques pour l'acceptabilité des arguments dans un système d'argumentation. Les ensembles d'arguments acceptables seront appelés *extensions*. Ces extensions doivent respecter quelques propriétés de base que Dung [Dun95] définit ainsi :

Définition 2 (Ensemble sans-conflit)
Un ensemble S d'arguments, $S \subseteq A$, est sans-conflit *s'il n'existe pas d'argument a, $b \in S$ tels que a attaque b ($(a,b) \in R$).*

Exemple 1 page précédente – Suite *L'ensemble $\{a, b\}$ n'est pas sans-conflit. L'ensemble $\{a, c, d\}$ est sans-conflit.*

Définition 3 (Défense collective)
Un ensemble S d'arguments, $S \subseteq A$, défend (collectivement) un argument *$a \in A$ si et seulement si $\forall b \in A$ tel que $(b,a) \in R$, $\exists c \in S$ tel que $(c,b) \in R$. On dit aussi que l'argument a est* acceptable *pour S.*
Un ensemble S d'arguments, $S \subseteq A$, défend (collectivement) tous ses éléments si et seulement si $\forall a \in S$, si $\exists b \in A$ tel que $(b,a) \in R$ alors $\exists c \in S$ tel que $(c,b) \in R$.

Exemple 1 page précédente – Suite *Les ensembles $\{a\}$ et $\{a, d\}$ défendent collectivement c et l'ensemble $\{b\}$ défend collectivement b. L'ensemble $\{a, c\}$ défend collectivement tous ses éléments.*

A partir de ces propriétés, Dung [Dun95] définit plusieurs sémantiques pour l'acceptabilité.
Un ensemble d'arguments est admissible si cet ensemble peut se défendre contre toute attaque.

Définition 4 (Ensemble admissible)
Un ensemble S d'arguments, $S \subseteq A$, est admissible *si et seulement si S est sans-conflit et S défend tous ses éléments.*

Exemple 1 page précédente – Suite *Les ensembles $\{a, c\}$ et $\{a, c, d\}$ sont admissibles.*

A partir de cette notion, Dung [Dun95] propose les sémantiques préférée et stable dont les extensions sont définies ainsi :

Définition 5 (Extension préférée)
Un ensemble S d'arguments, $S \subseteq A$, est une extension préférée *si et seulement si S est maximal pour l'inclusion parmi les ensembles admissibles.*

Exemple 1 page précédente – Suite *L'ensemble $\{a, c, d\}$ est une extension préférée.*

Définition 6 (Extension stable)
Un ensemble S d'arguments, $S \subseteq A$, est une extension stable *si et seulement si S est sans-conflit et S attaque tout argument n'appartenant pas à S.*

Exemple 1 page 3 – Suite *Les ensembles* $\{b,d\}$ *et* $\{a,c,d\}$ *sont des extensions stables.*

Notons que la notion *stable* garantit la défense collective. En effet, si un ensemble est stable alors il attaque tous les éléments qui ne sont pas dans cet ensemble, en particulier les éléments qui attaquent l'ensemble.
Dung [Dun95] propose alors quelques propriétés :

Propriété 1 *Soit* $<A,R>$ *un système d'argumentation, on a :*
- *Tout ensemble admissible de* $<A,R>$ *est contenu dans une extension préférée de* $<A,R>$*.*
- $<A,R>$ *possède au moins une extension préférée.*
- *Si* $<A,R>$ *est bien-fondé alors il possède une et une seule extension préférée qui est aussi la seule extension stable.*
- *Toute extension stable est aussi préférée (La réciproque est fausse).*

Nous allons nous inspirer des travaux de Dung [Dun95] pour proposer des sémantiques d'acceptabilité dans le cadre des systèmes d'argumentation bipolaires, c'est-à-dire des systèmes d'argumentation comportant deux types d'interaction : l'attaque et l'appui.

Chapitre 3

Un système d'argumentation bipolaire (SABP)

Un système d'argumentation bipolaire étend le système d'argumentation introduit par Dung (voir [Dun95]). La perspective d'un système d'argumentation muni d'une seule relation, l'attaque, est assez restrictive. De façon naturelle et empirique, nous pouvons facilement évaluer le rôle important de l'accord ou de l'appui dans une argumentation. Ainsi, ne parle-t-on pas de « témoin à charge » et de « témoin à décharge » dans un tribunal ?
Donc, en considérant qu'un système d'argumentation n'est pas seulement constitué d'une seule relation d'attaque, les systèmes bipolaires prennent en compte deux sortes d'interaction entre arguments, la relation d'attaque et la relation d'appui. Des travaux ont été effectués dans cette perspective, en particulier, C. Cayrol et M.C. Lagasquie-Schiex [CLS04] qui proposent donc de rajouter la relation d'appui au cadre présenté en section 1.

Définition 7 (SABP : système d'argumentation bipolaire)
Un système d'argumentation bipolaire *(noté SABP) est un triplet* $< A, R_{att}, R_{app} >$ *avec :*
- A *: un ensemble d'arguments,*
- R_{att} *une relation binaire sur A représentant la relation d'attaque*
- R_{app} *une relation binaire sur A représentant la relation d'appui.*

Soit deux arguments a_1, $a_2 \in$ A. On dira a_1 *attaque* a_2 (resp. a_1 *appuie* a_2) si $a_1 R_{att} a_2$ (resp. $a_1 R_{app} a_2$). Chaque couple $(a_i, a_j) \in R_{att}$ sera graphiquement représenté comme indiqué sur la figure 2.1 page 3 et chaque couple $(b_i, b_j) \in R_{app}$ sera graphiquement représenté comme indiqué sur la figure 3.1.

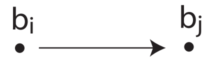


FIG. 3.1 – Représentation graphique d'un appui

Exemple 2 *Le graphe suivant représente le SABP :*
$<A=\{a,b,c,d,e,f\}$, $R_{att} = \{(a,b),(b,c),(b,e),(e,c)\}$, $R_{app} = \{(d,a),(e,f)\}>$.

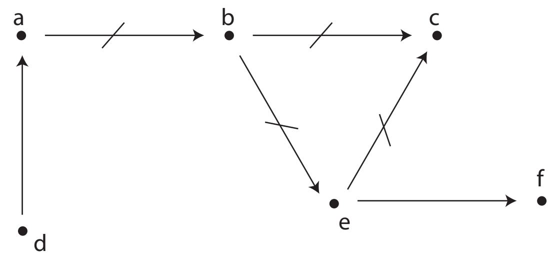


Fig. 3.2 – Exemple 2 : un SABP

Chapitre 4

Cohérence et défense dans un SABP : l'existant et les modifications proposées

Des travaux analogues à ceux de Dung [Dun95] ont été effectués par C. Cayrol et M.C. Lagasquie-Schiex [CLS04] dans le cadre de systèmes d'argumentation bipolaires. Après un bref rappel de la sémantique proposée (avec quelques modifications de terminologie), nous modifierons ce cadre pour élargir la notion de cohérence et améliorer la notion de défense.
Rappelons avant tout ce que signifie une *argumentation cohérente* : il s'agit de ne pas accepter de prendre en compte deux arguments entre lesquels il existerait un conflit.
Dans le cadre de Dung, cela se traduit juste par le refus de faire cohabiter dans un même ensemble acceptable deux arguments, dont l'un attaque l'autre.
Dans le cadre d'un système bipolaire, nous distinguons au moins deux formes d'incohérence pour un ensemble d'arguments (représentées sur la figure 4.1) :

- une incohérence *interne* (voir schéma de gauche), dans laquelle la source du conflit est interne à l'ensemble (deux arguments qui appartiennent à l'ensemble sont en conflit entre eux)[1]
- et une incohérence *externe* (voir schéma de droite), dans laquelle la source du conflit est externe à l'ensemble (deux arguments qui appartiennent à l'ensemble sont en conflit au sujet d'un autre argument qui n'appartient pas à l'ensemble – l'un l'appuie et l'autre l'attaque).

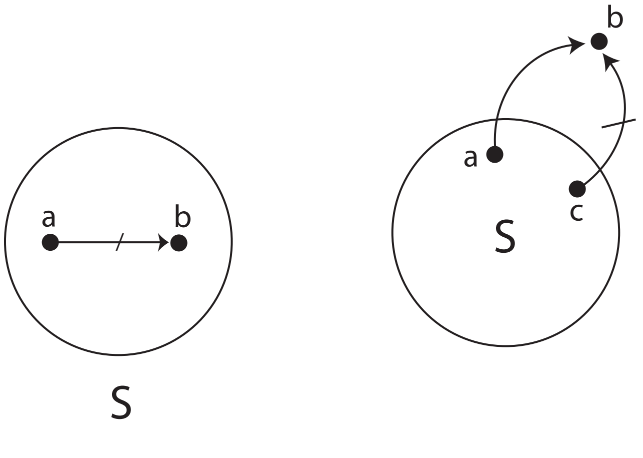


Fig. 4.1 – Les ensembles incohérents d'après [CLS04]

[1]Bien évidemment, la notion de conflit proposée par Dung entre dans le cadre de l'incohérence interne.

4.1 Sémantique proposée dans [CLS04]

Soit $<A,R_{att},R_{app}>$ un SABP. Nous définissons les relations d'attaque et d'appui suivantes sur lesquelles reposeront les différentes sémantiques d'acceptabilité.

Définition 8 (Attaque/Appui direct par un ensemble)
Soit $S \subseteq A$, soit $a \in A$. On dira que :
- S attaque directement *a ssi $\exists b \in S$ tel que $bR_{att}a$;*
- S appuie directement *a ssi $\exists b \in S$ tel que $bR_{app}a$.*

Exemple 2 page 7 – Suite *L'ensemble $\{b, e\}$ attaque directement l'argument c et appuie directement f.*

Définition 9 (Attaque appuyée)
Une attaque appuyée *est une séquence a_1 R_1 a_2 R_2 $\ldots R_{n-1}$ a_n, $n \geq 3$ avec $\forall i = 1\ldots n$, a_i est un argument de A et $\forall j = 1\ldots n\text{-}2$, $R_j = R_{app}$ et $R_{n-1} = R_{att}$.*

Autrement dit, une attaque appuyée est une attaque directe précédée d'une séquence d'appuis. Avec cette définition, [CLS04] propose donc de considérer que "les amis de nos ennemis" sont aussi "nos ennemis".

Exemple 2 page 7 – Suite *Il existe une attaque appuyée de d vers b.*

A l'aide des définitions des attaques directes, des appuis directs et de l'attaque appuyée, nous donnons la définition la plus générale de l'attaque et de l'appui complexes.

Définition 10 (Attaque/Appui complexe par un ensemble)
Soit $S \subseteq A$, soit $a \in A$. On dira que :
- S attaque de manière complexe *a ssi $\exists b \in S$ tel que :*
  - *soit $bR_{att}a$,*
  - *soit un des chemins de b vers a est une attaque appuyée.*
- S appuie de manière complexe *a ssi $\exists b \in S$ tel que un des chemins de b vers a est une séquence d'appuis directs.*

Exemple 2 page 7 – Suite *L'ensemble $\{b, e\}$ attaque de manière complexe c et l'ensemble $\{d\}$ attaque de manière complexe b.*

En utilisant les définitions 8 et 10, on peut alors vérifier la cohérence *interne*, représentée par les définitions de *sans-conflit-direct* et de *sans-conflit-complexe* et la cohérence *externe*, représentée par la notion d'*ensemble sûr* et d'*ensemble complexe-sûr* [2] :
Les définitions de sans conflit-direct et de directement sûr concernent les relations directes.

Définition 11 (Ensemble sans conflit-direct)
Soit $S \subseteq A$. S est sans conflit-direct *ssi il n'existe pas d'arguments a et $b \in S$ tels que $\{a\}$ attaque directement b.*

La notion de sans conflit-direct correspond exactement à la notion de sans-conflit définie par Dung.

Définition 12 (Ensemble directement sûr)
Soit $S \subseteq A$. S est directement sûr *ssi il n'existe pas d'arguments $b \in A$ et $c,d \in S$ tels que $\{c\}$ attaque directement b et, soit $\{d\}$ appuie directement b, soit $b \in S$.*

[2]La terminologie utilisée ici est légèrement différente de celle donnée dans [CLS04]. Nous avons rajouté le mot "complexe" pour éviter toute ambiguïté avec les définitions de Dung et avec la suite.

Exemple 3 *Sur l'exemple de la figure 4.2, l'ensemble* $\{b, c, d\}$ *n'est pas sans conflit-direct car l'attaque directe de b vers c représente le conflit « interne ». En revanche, l'ensemble* $\{b, d\}$ *est sans conflit-direct.*
L'ensemble $\{f, h\}$ *n'est pas directement sûr. L'ensemble* $\{e, f\}$ *est directement sûr.*

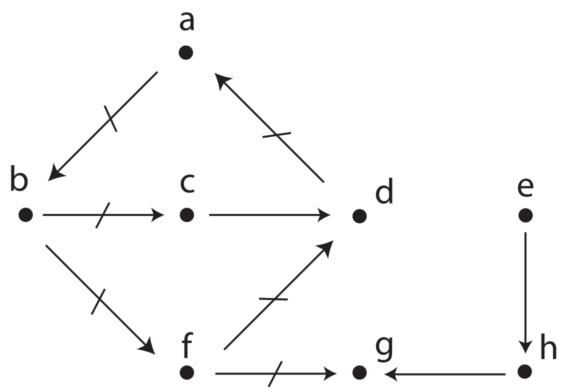


Fig. 4.2 – Exemple 3

Les définitions de sans conflit-complexe et de complexe sûr concernent les relations complexes.

Définition 13 (Ensemble sans-conflit-complexe)
Soit $S \subseteq A$. *S est* sans-conflit-complexe *ssi il n'existe pas d'arguments a et b de S tels que* $\{a\}$ *attaque de manière complexe b.*

Définition 14 (Ensemble complexe-sûr)
Soit $S \subseteq A$. *S est* complexe-sûr *ssi il n'existe pas d'argument b de A tel que S attaque de manière complexe b et, soit S appuie de manière complexe b, soit* $b \in S$.

Par définition, un ensemble complexe-sûr est sans-conflit-complexe, tout comme un ensemble sûr est aussi un ensemble sans-conflit direct.

Exemple 2 page 7 – Suite *L'ensemble* $\{d, b\}$ *n'est pas sans-conflit-complexe. En revanche, les ensembles* $\{a, c\}$ *et* $\{d, a, e, f\}$ *par exemple sont sans-conflit-complexe.*
Les ensembles $\{a, c\}$ *et* $\{d, a, e, f\}$ *sont complexe-sûrs.*

Nous rappelons la définition de la notion d'admissibilité d'un ensemble d'arguments proposé dans [CLS04] qui s'appuie sur la notion de défense complexe.

Définition 15 (Défense complexe par un ensemble)
Soit $S \subseteq A$. *Soit* $a \in A$. *S* défend de manière complexe[3] *a ssi* $\forall b \in A$, *si* $\{b\}$ *attaque de manière complexe a alors* $\exists c \in S$ *tel que* $\{c\}$ *attaque de manière complexe b.*

Définition 16 (Ensemble complexe-admissible)
Soit $S \subseteq A$. *S est* complexe-admissible[3] *ssi S est complexe-sûr et défend de manière complexe tous ses éléments.*

Exemple 2 page 7 – Suite *L'ensemble* $\{d, a, e, f\}$ *est complexe-admissible alors que l'ensemble* $\{e, f\}$ *ne l'est pas.*

[3]Ces notions ont été définies par [Dun95] dans le cadre d'un système d'argumentation unipolaire ; elles ne prenaient donc en compte initialement que les attaques directes. [CLS04] les a étendues à un système d'argumentation bipolaire en utilisant la notion d'attaque et de défense complexe donnée par les définitions 10 page précédente et 15.

4.2 Critique de la sémantique proposée dans [CLS04])

4.2.1 Amélioration de la cohérence

Nous avons vu que la cohérence implique, de façon naturelle, l'absence d'arguments en conflit dans un ensemble[4] et l'absence d'ensembles attaquant et appuyant un même argument[5].
Néanmoins, la notion d'attaque définie par [CLS04] pourrait être renforcée pour mieux s'assurer de l'absence d'ensembles incohérents. Par exemple, l'ensemble S de la figure 4.3 est sans-conflit-complexe (et même admissible au sens de la définition 16 page précédente) mais il attaque un argument b alors que le singleton $\{b\}$ constitué de cet argument appuie un argument de S.

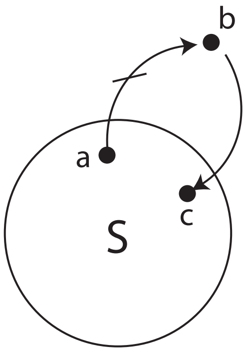


Fig. 4.3 – Ensemble cohérent pour [CLS04]

Cette configuration pourrait donc être considérée comme étant aussi un conflit entre a et c et donc interne à l'ensemble S.
Prendre en compte ce nouveau type de conflit nous amène à proposer une nouvelle attaque :

Définition 17 (Attaque détournée)
Une attaque détournée *est une séquence a_1 R_1 a_2 R_2 ...R_{n-1} a_n pour $n \geq 3$ avec $\forall i = 1 \ldots n$, a_i est un argument de A, $R_1 = R_{att}$ et $\forall j = 2 \ldots n-1$, $R_j = R_{app}$. Autrement dit, c'est une attaque directe suivie d'une séquence d'appuis.*

Exemple 2 page 7 – Suite *Il existe une attaque détournée entre b et f.*

Puis en utilisant cette nouvelle attaque, on peut étendre la définition 10 page 10 de manière à avoir une notion d'attaque encore plus riche :

Définition 18 (Attaque complexe+ par un ensemble)
Soit $S \subseteq A$, soit $a \in A$. On dira que : S attaque de manière complexe+ *a ssi $\exists b \in S$ tel que :*
- *soit $bR_{att}a$,*
- *soit un des chemins de b vers a est une attaque appuyée,*
- *soit un des chemins de b vers a est une attaque détournée.*

Nous déclarons, par cette définition, qu'une attaque détournée, une attaque appuyée et une attaque directe seront considérées comme des cas d'attaque. Si cela semble tout à fait clair dans le cas d'une attaque directe, en revanche, il peut arriver que, sur certains exemples, les attaques détournées ou appuyées ne possèdent pas vraiment le sens d'attaque que l'on désirait ; cet aspect sera illustré en particulier en section 4.3 page ci-contre à l'aide d'exemples. En effet, il peut arriver que, suivant les exemples, les "amis de nos ennemis" et les "ennemis de nos amis" soient ou ne soient pas nos "ennemis".

[4]La cohérence interne.
[5]La cohérence externe.

4.2.2 Amélioration de la défense

Un autre point évoqué dans [CLS04] est susceptible d'amélioration : la notion de défense.
En effet, le fait de proposer une défense qui répond à une attaque complexe (voir la définition 15 page 11) peut amener à des résultats contre-intuitifs (et à plus forte raison, si on l'étend à la prise en compte d'une attaque complexe+) :

Exemple 4 *Si on applique la définition de la défense proposée dans [CLS04] (définition 15 page 11), on obtient :*
- *dans le premier exemple de la figure 4.4, l'argument a_0 ne peut être défendu que par a_4, ce qui paraît une défense bien "lointaine" ;*
- *et dans le second exemple de la figure 4.4, l'argument a_1 ne peut se défendre face à l'argument a_3, ce qui paraît contre-intuitif.*

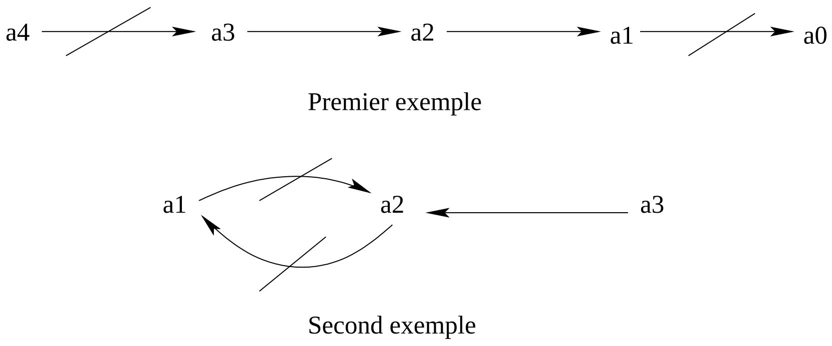


Fig. 4.4 – Fonctionnement de la défense dans [CLS04]

Nous allons donc reprendre exactement la définition de Dung [Dun95] sur la défense en insistant sur l'aspect *direct* de l'attaque.

Définition 19 (Défense collective)
Un ensemble S d'arguments, $S \subseteq A$, défend (collectivement) *un argument $a \in A$ si et seulement si $\forall b \in A$ tel que $\{b\}$ attaque directement a, $\exists c \in S$ tel que $\{c\}$ attaque directement b.*
Un ensemble S d'arguments, $S \subseteq A$, défend (collectivement) *tous ses éléments si et seulement si $\forall a \in S$, si $\exists b \in A$ tel que $\{b\}$ attaque directement a alors $\exists c \in S$ tel que $\{c\}$ attaque directement b.*

A l'aide de ces nouvelles définitions d'attaque et de défense, nous allons pouvoir maintenant définir des propriétés d'acceptabilité dans un système d'argumentation bipolaire. Ces propriétés sont inspirées des travaux de Dung [Dun95] sur l'acceptabilité dans un système d'argumentation unipolaire.
Toutefois, avant de passer à la partie acceptabilité, voyons sur quelques exemples la signification des différentes attaques choisies.

4.3 Validité du cadre abstrait

Il est évident que le travail précédent reste abstrait car il n'exploite pas la forme des arguments et il ne définit que très grossièrement les types d'interactions entre arguments. Il devient alors intéressant d'étudier le sens des notions définies précédemment lorsque l'on est ramené à un cadre plus concret. Ceci permet de vérifier si les résultats obtenus dans un cadre concret correspondent à ceux attendus dans le cadre abstrait.
Nous obtenons un cadre concret en instanciant le cadre abstrait. Pour cela, nous utiliserons et nous rappellerons les définitions d'arguments et d'interactions proposées dans [CLS04].

4.3.1 Instanciation du cadre abstrait

4.3.1.1 Définition d'un argument

Soit Λ un langage logique (par exemple, la logique des prédicats ou la logique propositionnelle). Soit $\vdash$ la relation d'inférence associée à Λ.

Définition 20 (Argument)
Un argument *est un couple (S,C) avec :*
– S un support (ensemble consistant de formules de Λ),
– C une formule consistante de Λ,
respectant les contraintes suivantes :
– $S \vdash C$,
– S minimal pour obtenir C ($\forall \varphi_i \in S$, $S \backslash \varphi_i \nvdash C$)

Exemple 5 *Voici des exemples d'argument :*
– $(\{voiture, voiture \vee train \rightarrow \neg retard\}, \neg retard)$
– $(\{train, voiture \vee train \rightarrow \neg retard\}, \neg retard)$
En revanche, le couple $(\{voiture, \neg train, voiture \vee train \rightarrow \neg retard\}, \neg retard)$ n'est pas un argument car $\{voiture, \neg train, voiture \vee train \rightarrow \neg retard\}$ n'est pas minimal.

4.3.1.2 Définition des relations

Avec cette définition d'argument, nous rappelons plusieurs types d'attaques et d'appuis parmi les plus pertinents proposés par [CLS04].

Définition 21 (Attaque d'une partie du support de a_2 à l'aide de la conclusion de a_1)
Soit $a_1 = (S_{a_1}, C_{a_1})$ et $a_2 = (S_{a_2}, C_{a_2})$ deux arguments. a_1 attaque1 directement a_2 *(noté $a_1 \not\rightarrow a_2$) si et seulement si $\neg C_{a_1} \in S_{a_2}$.*

Définition 22 (Attaque de la conclusion de a_2 à l'aide de la conclusion de a_1)
Soit $a_1 = (S_{a_1}, C_{a_1})$ et $a_2 = (S_{a_2}, C_{a_2})$ deux arguments. a_1 attaque2 directement a_2 *(noté $a_1 \overset{*}{\not\leftrightarrow} a_2$ car l'attaque2 est symétrique) si et seulement si $\neg C_{a_1} \equiv C_{a_2}$.*

Définition 23 (Appui d'une partie du support de a_2 à l'aide de la conclusion de a_1)
Soit $a_1 = (S_{a_1}, C_{a_1})$ et $a_2 = (S_{a_2}, C_{a_2})$ deux arguments. a_1 appuie1 directement a_2 *(noté $a_1 \rightarrow a_2$) si et seulement si $C_{a_1} \in S_{a_2}$.*

Définition 24 (Appui de la conclusion de a_2 à l'aide de la conclusion de a_1)
Soit $a_1 = (S_{a_1}, C_{a_1})$ et $a_2 = (S_{a_2}, C_{a_2})$ deux arguments. a_1 appuie2 directement a_2 *(noté $a_1 \overset{*}{\leftrightarrow} a_2$ car l'appui2 est symétrique) si et seulement si $C_{a_1} \equiv C_{a_2}$.*

Tout autre attaque ou appui peut se ramener à ceux définis précédemment.

Exemple 6 *Soit les quatre arguments suivants :*
– $a_1 = (\{voiture, voiture \vee train \rightarrow \neg retard\}, \neg retard)$
– $a_2 = (\{train, voiture \vee train \rightarrow \neg retard\}, \neg retard)$
– $a_3 = (\{voiture, voiture \rightarrow \neg train\}, \neg train)$
– $a_4 = (\{voiture\}, voiture)$
Nous obtenons le graphe suivant :

Remarque : L'attaque (resp. l'appui) sur conclusion peut être traduite en attaque sur support (resp. l'appui sur support) grâce à la création d'un nouvel argument. Cette initiative tend à rendre le système d'argumentation plus homogène.

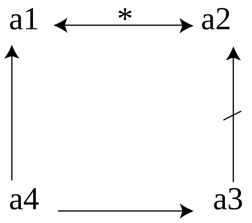


FIG. 4.5 – Exemple 6

En effet, supposons deux arguments $a_1 = (S_{a_1}, C_{a_1})$ et $a_2 = (S_{a_2}, C_{a_2})$ tels que $a_1 \overset{*}{\not\to} a_2$ (resp. $a_1 \overset{*}{\leftrightarrow} a_2$). Donc, $C_{a_2} \equiv \neg C_{a_1}$ (resp. $C_{a_2} \equiv C_{a_1}$). Nous pouvons alors créer un argument $a_3 = (S_{a_3}, C_{a_3})$ tel que $S_{a_3} = \{C_{a_1}\}$, $C_{a_3} = C_{a_1}$ et on a alors $a_1 \to a_3 \not\leftarrow a_2$ (resp. $a_1 \to a_3 \leftarrow a_2$) en interdisant les attaques et les appuis sur conclusion (donc en autorisant seulement les définitions 21 page précédente et 23 page ci-contre).

Exemple 6 page précédente – Suite *Nous obtenons cinq arguments :*
- a_1=($\{voiture, voiture \vee train \to \neg retard\}, \neg retard$)
- a_2=($\{train, voiture \vee train \to \neg retard\}, \neg retard$)
- a_3=($\{voiture, voiture \to \neg train\}, \neg train$)
- a_4=($\{voiture\}, voiture$)
- a_5=($\{\neg retard\}, \neg retard$)

Nous obtenons le graphe suivant : Le graphe ne contient que des attaques et des appuis sur support.

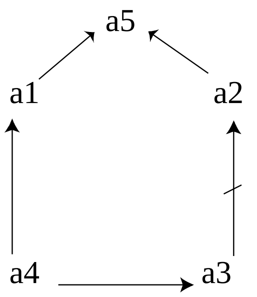


FIG. 4.6 – Exemple 6 suite

Maintenant qu'ont été définis les arguments et leurs interactions, nous allons interpréter les notions de base du cadre abstrait sur ces instances.

4.3.2 Interprétation de l'attaque

La définition 21 page ci-contre définit le sens logique d'une attaque directe entre deux arguments. Dans ce contexte, nous verrons qu'une attaque détournée ou appuyée ne peut pas toujours être interprétée comme une attaque.

4.3.2.1 Attaque appuyée

Soit trois arguments a_1, a_2 et a_3. Avec les définitions et les notations précédentes, une attaque appuyée de a_1 sur a_3 est représentée par : $a_1 \to a_2 \not\to a_3$. L'interprétation de l'attaque appuyée ne nous permet pas d'affirmer dans **tous les cas** que l'argument a_1 *attaque* a_3 au sens où il y aurait une inconsistance logique entre a_1 et a_3.

Exemple 7 *Trois personnes sont successivement mises en présence d'une partie d'un même animal pendant un court moment, leurs commentaires ont été relevés :*
- *L'agent 1 affirme avoir reconnu une vipère. Il sait qu'une vipère est un serpent. Il en déduit donc qu'il était en présence d'un serpent.*
- *L'agent 2 a cru voir un serpent dont les dents seraient manquantes. Comme il sait que les serpents sans dents ne peuvent pas mordre, il déduit que l'animal en question ne peut pas être l'auteur de morsure.*
- *L'agent 3 a failli être mordu par l'animal et n'a pas eu le temps de le voir. Il le croit dangereux.*

Nous obtenons les trois arguments suivants :
- a_1=$(\{vip\grave{e}re,\ vip\grave{e}re \rightarrow serpent\},\ serpent)$
- a_2=$(\{serpent,\ dents\_manquantes,\ serpent \wedge dents\_manquantes \rightarrow \neg morsure\}, \neg morsure)$
- a_3=$(\{morsure,\ morsure \rightarrow danger\},\ danger)$

Dans cet exemple, nous ne pouvons pas interpréter l'attaque appuyée comme une *attaque* de l'argument a_1 sur l'argument a_3. En effet, l'argument a_1 ne peut contredire l'argument a_3 d'aucune manière. Le simple fait d'être en présence d'un serpent ne remet pas en cause l'éventualité d'une morsure. Les arguments a_1 et a_3 semblent donc indépendants.
En revanche, l'agent 1 semble confirmer les dires de l'agent 2. En échangeant leurs informations, les agents 1 et 2 semblent former une *coalition* contre l'agent 3. Ainsi, grâce à cette coalition l'argument a_1 *attaque* l'argument a_3 par l'intermédiaire de l'argument a_2, l'agent 1 <u>ET</u> l'agent 2 pouvant déduire que l'animal ne pouvait pas mordre.
Ainsi, l'interprétation d'une attaque appuyée dépend directement de la faculté des arguments à s'allier via la relation d'appui :

> Si nous refusons l'interprétation des *coalitions*, les arguments a_1 et a_3 sont indépendants et l'attaque appuyée n'a pas le sens d'une *attaque*.
>
> Par contre, si les arguments qui s'appuient représentent une *coalition* alors l'attaque appuyée a bien le sens d'une attaque.

4.3.2.2 Attaque détournée

Soit trois arguments a_0, a_1, a_2. L'attaque détournée de a_0 sur a_2 est représentée par : $a_0 \not\rightarrow a_1 \rightarrow a_2$.
De façon similaire à l'attaque appuyée les interprétations influencent les résultats.

Exemple 8 *Trois personnes sont successivement mises en présence d'une partie d'un même animal pendant un court moment, leurs commentaires ont été relevés :*
- *L'agent 0 a vu un ver de terre. Il sait donc que ce n'est pas une vipère.*
- *L'agent 1 affirme avoir reconnu une vipère. Il sait qu'une vipère est un serpent. Il en déduit donc qu'il était en présence d'un serpent.*
- *L'agent 2 a cru voir un serpent dont les dents seraient manquantes. Comme il sait que les serpents sans dents ne peuvent pas mordre, il déduit que l'animal en question ne peut pas être l'auteur de morsure.*

Nous obtenons les trois arguments suivants :
- a_0=$(\{ver\_de\_terre,\ ver\_de\_terre \rightarrow\ \neg vip\grave{e}re\},\ \neg vip\grave{e}re)$
- a_1=$(\{vip\grave{e}re,\ vip\grave{e}re \rightarrow serpent\},\ serpent)$
- a_2=$(\{serpent,\ dents\_manquantes,\ serpent \wedge dents\_manquantes \rightarrow \neg morsure\}, \neg morsure)$

Comme pour le cas d'une attaque appuyée, le sens de l'attaque détournée dépend de l'interprétation de la relation d'appui. En effet, nous ne pouvons pas conclure que le fait d'être en présence d'un ver de terre et non d'une vipère remet en cause le fait que l'animal en question ne mord pas, il pourrait même confirmer ce dernier si l'agent 0 sait que les vers de terre ne mordent pas.
Dans le cas de *coalition* entre les arguments a_1 et a_2, si les agents 1 et 2 partagent leurs informations, ils peuvent tous les deux déduire que l'animal ne mord pas. L'argument a_0, en disant que l'animal n'était pas une vipère, contredirait alors leur déduction et attaquerait les arguments a_1 et a_2.

Donc, comme dans le cas de l'attaque appuyée, l'attaque détournée a un sens qui dépend directement de celui donné à l'appui (coalition ou pas) :

> Si nous refusons l'interprétation des *coalitions*, les arguments a_0 et a_2 sont indépendants et l'attaque détournée n'a pas le sens d'une *attaque*.
>
> Par contre, si les arguments qui s'appuient représentent une *coalition* alors l'attaque détournée a bien le sens d'une attaque.

4.3.3 Interprétation de l'appui

La définition 23 page 14 donne une interprétation logique d'un appui direct entre deux arguments. Soit trois arguments a_1, a_2 et a_4. La séquence d'appui de a_1 sur a_4 est représentée par : $a_1 \rightarrow a_2 \rightarrow a_4$.

Exemple 9 *Trois personnes sont successivement mises en présence d'une partie d'un même animal pendant un court moment, leurs commentaires ont été relevés :*
- *L'agent 1 affirme avoir reconnu une vipère. Il sait qu'une vipère est un serpent. Il en déduit donc qu'il était en présence d'un serpent.*
- *L'agent 2 a cru voir un serpent dont les dents seraient manquantes. Comme il sait que les serpents sans dents ne peuvent pas mordre, il déduit que l'animal en question ne peut pas être l'auteur de morsure.*
- *l'agent 4 a touché l'animal et l'animal n'a pas cherché à le mordre. Il en conclut que cet animal n'est pas dangereux.*

Nous obtenons les trois arguments suivants :
- a_1=($\{vipère,\ vipère \rightarrow serpent\},\ serpent$)
- a_2=($\{serpent,\ dents\_manquantes,\ serpent \wedge dents\_manquantes \rightarrow \neg morsure\}, \neg morsure$)
- a_0=($\{\neg morsure, \neg morsure \rightarrow \neg danger\}, \neg danger$)

Dans cet exemple, il est évident que chaque argument n'appuie qu'une partie d'un autre argument. Le fait de savoir que l'animal est une vipère ne paraît pas aider à conclure qu'il n'y a pas de danger (au contraire, si l'agent 1 sait que les vipères sont en général des animaux dangereux).
Par contre, dès qu'on considère que les agents forment une coalition alors ils sont susceptibles d'échanger leurs informations et ils peuvent tous déduire qu'il n'y a pas de danger.
On a donc ici aussi un sens différent suivant l'interprétation choisie :

> Si nous refusons l'interprétation des *coalitions*, les arguments a_1 et a_4 sont indépendants et l'appui complexe n'a pas le sens d'un *appui*.
>
> Par contre, si les arguments qui s'appuient représentent une *coalition* alors l'appui complexe a bien le sens d'un appui.

4.3.4 Interprétation de la défense

Soit quatre arguments a_0, a_1, a_2 et a_3. La défense complexe de a_3 par a_0 est représentée par : $a_0 \not\rightarrow a_1 \rightarrow a_2 \not\rightarrow a_3$.

Exemple 10 *Quatre personnes sont successivement mises en présence d'une partie d'un même animal pendant un court moment, leurs commentaires ont été relevés :*
- *L'agent 0 a vu un ver de terre. Il sait donc que ce n'est pas une vipère.*
- *L'agent 1 affirme avoir reconnu une vipère. Il sait qu'une vipère est un serpent. Il en déduit donc qu'il était en présence d'un serpent.*
- *L'agent 2 a cru voir un serpent dont les dents seraient manquantes. Comme il sait que les serpents sans dents ne peuvent pas mordre, il déduit que l'animal en question ne peut pas être l'auteur de morsure.*
- *L'agent 3 a failli être mordu par l'animal et n'a pas eu le temps de le voir. Il le croit dangereux.*

Nous obtenons les trois arguments suivants :
- a_0=($\{ver\_de\_terre,\ ver\_de\_terre \rightarrow\ \neg vipère\},\ \neg vipère$)

– a_1=($\{vipère,\ vipère \rightarrow serpent\},\ serpent$)
– a_2=($\{serpent,\ dents\_manquantes,\ serpent \wedge dents\_manquantes \rightarrow \neg morsure\}, \neg morsure$)
– a_3=($\{morsure,\ morsure \rightarrow danger\},\ danger$)

a_3 est directement attaqué par a_2, mais le fait de connaître a_0 (l'animal est un ver de terre) ne permet pas de “réinstaller” l'argument a_3 (qui conclut à l'existence d'un danger). Et cela, même si on considère que a_1 et a_2 forment une coalition.

> Donc dans le cas de la défense complexe, aucune interprétation ne conduit vraiment au sens d'une défense telle que l'avait conçu Dung.

4.3.5 Conclusion

Si nous considérons que des arguments qui s'appuient (une séquence d'appuis par exemple) peuvent se réunir implicitement sous une même conclusion, ou sous une même *coalition*[6], alors les notions d'attaque appuyée et détournée ainsi que la notion d'appui complexe ont bien un intérêt. Dans le cas contraire, seuls les attaques directes et les appuis directs seront considérés.
Et en ce qui concerne la défense, seule la défense directe paraît justifiée.

[6]Notion évoquée dans la section *Perspectives*

Chapitre 5

L'acceptabilité dans un SABP

Nous nous intéressons à la sémantique préférée basée sur les concepts de cohérence et de défense, puis, à la sémantique stable, basée sur la cohérence et l'attaque des éléments extérieurs à l'ensemble. Nous donnerons pour cela les notions de base utilisées (entre autres les différents types de conflit possibles), puis les définitions pour l'admissibilité (de la plus générale à la plus spécifique) et enfin les sémantiques préférées et stables pour le cadre bipolaire.

5.1 Notions de base : types de conflits, clôture et indépendance

Chaque sémantique doit reposer d'abord sur une notion de cohérence. Nous donnons alors plusieurs définitions de cohérence basées sur les relations directes (attaque directe et appui direct) et plus ou moins complexes (attaque complexe et complexe+ et appui complexe). Certaines de ces notions ont déjà été rappelées en section 4.1 page 10 puisqu'elles sont issues de [CLS04]. Nous les rappelons ici quand même afin d'avoir toutes les définitions de cohérence (les anciennes et les nouvelles) énumérées dans une même section.

Rappel de la définition 11 page 10 (Ensemble sans conflit-direct)
Soit $S \subseteq A$. S est sans conflit-direct *ssi il n'existe pas d'arguments a et $b \in S$ tels que $\{a\}$ attaque directement b.*

Rappel de la définition 12 page 10 (Ensemble directement sûr)
Soit $S \subseteq A$. S est directement sûr *ssi il n'existe pas d'arguments $b \in A$ et $c,d \in S$ tels que $\{c\}$ attaque directement b et, soit $\{d\}$ appuie directement b, soit $b \in S$.*

Les définitions de sans-conflit-complexe (resp. sans-conflit-complexe+) et de complexe-sûr (resp. complexe+-sûr) concernent les relations d'attaque complexe (resp. complexe+).

Définition 25 (Ensemble sans-conflit-complexe (resp. sans-conflit-complexe+))
Soit $S \subseteq A$. S est sans-conflit-complexe *(resp.* sans-conflit-complexe+*) ssi il n'existe pas d'arguments a et b de S tels que $\{a\}$ attaque de manière complexe (resp. complexe+) b.*

Définition 26 (Ensemble complexe-sûr (resp. complexe+-sûr))
Soit $S \subseteq A$. S est complexe-sûr *(resp.* complexe+-sûr*) ssi il n'existe pas d'argument b de A tel que S attaque de manière complexe (resp. complexe+) b et, soit S appuie de manière complexe b, soit $b \in S$.*

Exemple 3 page 11 – Suite *L'ensemble $\{b, d\}$ est sans-conflit-complexe, mais il n'est pas sans-conflit-complexe+.*
L'ensemble $\{c, a\}$ n'est pas sans-conflit-complexe, et il n'est pas sans-conflit-complexe+.
L'ensemble $\{e, f\}$ est sans-conflit-complexe+ et sans-conflit-complexe, mais ni complexe-sûr, ni complexe+-sûr.
Les ensembles $\{a, f\}$ et $\{a, e, h, g\}$ sont complexe+-sûrs.

Les notions suivantes, concernant la relation R_{app}, sont indépendantes des différentes définitions d'*attaque* et d'*appui*.

Définition 27 (Ensemble clos pour la relation R_{app})
Soit $S \subseteq A$, S est clos pour la relation d'appui R_{app} *ssi $\forall$ $a \in A$, si S appuie de manière complexe a alors $a \in S$. Autrement dit, un ensemble est clos pour R_{app} ssi il contient tous les éléments qu'il appuie de manière complexe.*

La notion de clôture sous-entend que nous n'avons aucune raison de ne pas accepter un argument que nous appuyons.

Définition 28 (Ensemble indépendant pour la relation R_{app})
Soit $S \subseteq A$, S est indépendant pour la relation d'appui R_{app} *ssi $\forall$ $a \in A$, si S appuie de manière complexe a ou si $\{a\}$ appuie de manière complexe S alors $a \in S$. Autrement dit, un ensemble est indépendant pour R_{app} ssi il contient tous les arguments qu'il appuie de manière complexe et qui l'appuient de manière complexe.*

La notion d'indépendance sous-entend que nous n'avons aucune raison de ne pas accepter un argument que nous appuyons ou qui nous appuie.

Exemple 3 page 11 – Suite *L'ensemble $\{e, h\}$ n'est pas clos pour la relation R_{app}. L'ensemble $\{g, h\}$ est clos pour la relation R_{app}.*
L'ensemble $\{g, h\}$ n'est pas indépendant pour la relation R_{app}. L'ensemble $\{e, g, h\}$ est indépendant pour la relation R_{app}.

Nous trouvons alors différentes propriétés sur les ensembles respectant ces différentes notions de base.

Propriété 2

1. *Un ensemble sans-conflit-complexe (resp. complexe-sûr) est sans conflit-direct (resp. directement sûr).*
2. *Un ensemble sans-conflit-complexe+ (resp. complexe+-sûr) est sans conflit-complexe (resp. complexe sûr).*
3. *Soit $S \subseteq A$, si S est directement sûr (resp. complexe sûr, complexe+-sûr) alors S est sans conflit-direct (resp. sans-conflit-complexe, sans-conflit-complexe+).*
4. *Soit $S \subseteq A$, si S est sans conflit-direct (resp. sans-conflit-complexe, sans-conflit-complexe+) et clos pour la relation R_{app} alors S est directement sûr (resp. complexe-sûr, complexe+-sûr).*
5. *Soit $S \subseteq A$, si S est indépendant pour la relation R_{app} alors S est clos pour la relation R_{app}.*
6. *Si $R_{app} = \varnothing$ alors S est sans-conflit-complexe ssi S est sans-conflit-direct et S est sans-conflit-complexe+ ssi S est sans-conflit-direct.*

Preuve :

1. Évident par la définition 10 page 10.
2. Évident par la définition 18 page 12.
3. Évident par les définitions 11 page 10, 12 page 10, 25 page précédente et 26 page précédente.

4. Soit S un ensemble d'arguments clos pour la relation R_{app} et sans conflit-direct (resp. sans-conflit-complexe, sans-conflit-complexe+). S est clos, donc il n'existe pas d'argument $a \in A \backslash S$ tel que S appuie directement (resp. appuie de manière complexe) et attaque directement (resp. attaque de manière complexe, attaque de manière complexe+) a. Donc S est directement sûr (resp. complexe-sûr, 3 complexe-sûr).
5. Évident par les définitions 27 page précédente et 28 page ci-contre.
6. Évident d'après les définitions 11 page 10 et 25 page 19.

□

Les liens d'inclusion entre ces différents types d'ensemble sont représentés sur la figure 5.1.

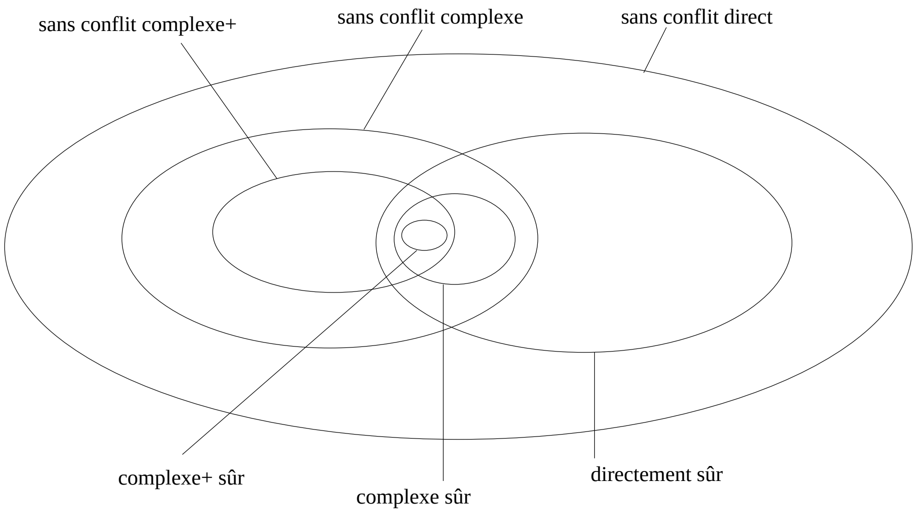


FIG. 5.1 – Liens entre les différents ensembles sans-conflit et sûrs

5.2 Notion d'admissibilité

La sémantique admissible proposée par Dung [Dun95] repose sur un concept de *cohérence* et sur la défense directe (définition 19 page 13). Nous en proposons plusieurs variantes à l'aide des définitions précédentes.
La terminologie utilisée sera la suivante :
- les sémantiques utilisant la notion de sans-conflit-direct seront dites *faibles* (elles ne prennent en compte que les attaques directes) ;
- les sémantiques utilisant la notion de sans-conflit-complexe seront dites *moyennes* (elles prennent en compte les attaques directes et les attaques appuyées) ;
- les sémantiques utilisant la notion de sans-conflit-complexe+ seront dites *fortes* (elles prennent en compte toutes les attaques possibles – directes, appuyées et détournées –).

Pour les sémantiques faiblement (resp. moyennement, fortement) d-admissibles, la cohérence est représentée par la notion de sans-conflit-direct (resp. sans-conflit-complexe, sans-conflit-complexe+).

Définition 29 (Ensemble faiblement (resp. moyennement, fortement) d-admissible) *Soit $S \subseteq A$, S est* faiblement (resp. moyennement, fortement) d-admissible[1] *ssi S est sans conflit-direct (resp. sans-conflit-complexe, sans-conflit-complexe+), et défend tous ses éléments.*

Exemple 3 page 11 – Suite *L'ensemble $\{a, b, d\}$ n'est pas faiblement d-admissible. En revanche,*

[1] « d » signifie « au sens de Dung »

les ensembles $\{b, d, g\}$ et $\{a, f\}$ sont faiblement d-admissibles.
L'ensemble $\{b, d, g\}$ est moyennement d-admissible mais pas fortement d-admissible. En revanche, l'ensemble $\{a, f\}$ est fortement d-admissible.

Pour les sémantiques faiblement/moyennement/fortement s-admissibles, la *cohérence* est représentée par la notion de sûreté.

Définition 30 (Ensemble faiblement (resp. moyennement, fortement) s-admissible)
Soit $S \subseteq A$, S est faiblement (resp. moyennement, fortement) s-admissible[2] *ssi S est directement sûr (resp. complexe sûr, complexe+ sûr), et défend tous ses éléments.*

Exemple 3 page 11 – Suite *L'ensemble $\{a, f, h\}$ n'est pas faiblement s-admissible, l'ensemble $\{a, f, e\}$ est faiblement s-admissible, mais ni moyennement, ni fortement s-admissible. L'ensemble $\{a, f\}$ est fortement s-admissible.*

Pour les sémantiques faiblement/moyennement/fortement c-admissibles, la *cohérence* est représentée par la notion de clôture pour la relation d'appui associée à la notion de sans-conflit.

Définition 31 (Ensemble faiblement (resp. moyennement, fortement) c-admissible)
Soit $S \subseteq A$, S est faiblement (resp. moyennement, fortement) c-admissible[3] *ssi S est faiblement (resp. moyennement, fortement) d-admissible et clos pour la relation d'appui R_{app}.*

Exemple 3 page 11 – Suite *L'ensemble $\{b, d, g, e\}$ n'est pas faiblement c-admissible, il est faiblement d-admissible mais il n'est pas clos pour la relation R_{app}.*
Les ensembles $\{b, d, g, h\}$ et $\{b, d, g\}$ sont faiblement et moyennement c-admissibles, mais pas fortement c-admissibles.
L'ensemble $\{a, f, e\}$ n'est pas fortement c-admissible car non clos pour la relation R_{app}.
L'ensemble $\{a, f\}$ est fortement c-admissible.

Enfin, pour les sémantiques faiblement/moyennement/fortement i-admissibles, la *cohérence* est représentée par la notion d'indépendance pour la relation d'appui associée à la notion de sans-conflit.

Définition 32 (Ensemble faiblement/moyennement/fortement i-admissible)
Soit $S \subseteq A$, S est faiblement (resp. moyennement, fortement) i-admissible[4] *ssi S est faiblement (resp. moyennement, fortement) d-admissible et indépendant pour la relation d'appui R_{app}.*

Exemple 3 page 11 – Suite *L'ensemble $\{b, d, g\}$ n'est pas faiblement i-admissible.*
L'ensemble $\{a, f\}$ est faiblement, moyennement et fortement i-admissible dans l'exemple 3 page 11.

Par la propriété 2 page 20, nous obtenons :

Conséquence 1

1. *Soit $S \subseteq A$, si S est fortement d,s,c,i-admissible alors S est moyennement d,s,c,i-admissible.*
2. *Soit $S \subseteq A$, si S est moyennement d,s,c,i-admissible alors S est faiblement d,s,c,i-admissible.*
3. *Soit $S \subseteq A$, si S est faiblement s-admissible (resp. moyennement, fortement s-admissible) alors S est faiblement d-admissible (resp. moyennement, fortement d-admissible).*
4. *Soit $S \subseteq A$, si S est faiblement c-admissible (resp. moyennement, fortement c-admissible) alors S est faiblement s-admissible (resp. moyennement, fortement s-admissible).*

[2] « s » signifie « sûr »
[3] « c » signifie « clos pour R_{app} »
[4] « i » signifie « indépendant pour R_{app} »

5. *Soit $S \subseteq A$, si S est faiblement i-admissible (resp. moyennement, fortement i-admissible) alors S est faiblement c-admissible (resp. moyennement, fortement c-admissible).*

Nous avons aussi une propriété d'existence évidente :

Propriété 3 *Quelle que soit la classe de sémantique donnée (faible, moyenne ou forte) et quel que soit $x \in \{d, s, c, i\}$, il existe toujours au moins un ensemble x-admissible dans cette classe.*

> **Preuve :** La preuve repose sur le fait que l'ensemble vide a toutes les propriétés (sans-conflit-direct, sans-conflit-complexe, sans-conflit-complexe+, directement sûr, complexe-sûr, complexe+-sûr, clos pour R_{app}, indépendant pour R_{app}). □

Le schéma de la figure 5.2 résume les implications entre les différentes définitions pour l'admissibilité dans chaque classe d'acceptabilité (faible, moyenne, forte).

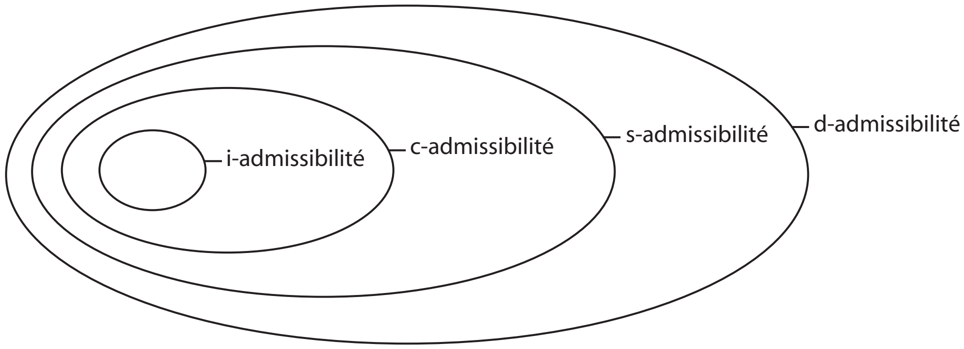


FIG. 5.2 – Relation entre les définitions d'admissibilité dans une classe d'acceptabilité donnée (forte, moyenne, faible)

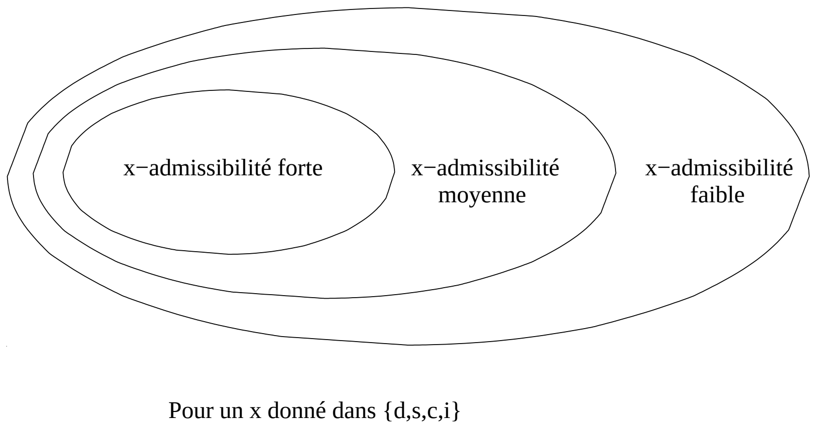


FIG. 5.3 – Relation entre les classes d'acceptabilité pour une admissibilité x donnée

5.3 Extensions Préférées

Avec les précédentes notions d'admissibilité et en étendant les propositions de [Dun95], nous pouvons proposer de nouvelles sémantiques pour l'acceptabilité.
Les extensions faiblement préférées concernent les définitions d'admissibilité basées sur les relations directes.

Définition 33 (Extensions faiblement préférées)
Soit $S \subseteq A$, S est une extension faiblement d-préférée *(resp.* faiblement s-préférée, faiblement c-préférée, faiblement i-préférée*) ssi S est maximal pour l'inclusion parmi les ensembles faiblement d-admissibles (resp. faiblement s-admissibles, faiblement c-admissibles, faiblement i-admissibles).*

Exemple 3 page 11 – Suite *Les ensembles $\{b, d, e, g, h\}$ et $\{a, c, e, f, h\}$ sont faiblement d-préférés, les ensembles $\{b, d, e, g, h\}$, $\{a, e, f\}$ sont faiblement s-préférés, les ensembles $\{a, f\}$ et $\{b, d, e, g, h\}$ sont faiblement c-préférés et seul $\{a, f\}$ est faiblement i-préféré.*

Les extensions moyennement préférées concernent les définitions d'admissibilité basées sur les attaques complexes.

Définition 34 (Extensions moyennement préférées)
Soit $S \subseteq A$, S est une extension moyennement d-préférée *(resp.* moyennement s-préférée, moyennement c-préférée, moyennement i-préférée*) ssi S est maximal pour l'inclusion parmi les ensembles moyennement d-admissibles (resp. moyennement s-admissibles, moyennement c-admissibles, moyennement i-admissibles).*

Exemple 3 page 11 – Suite *Les ensembles $\{b, d, e, g, h\}$ et $\{a, e, f, h\}$ sont moyennement d-préférés, les ensembles $\{b, d, e, g, h\}$, $\{a, f\}$ sont moyennement s-préférés et aussi moyennement c-préférés. Et seul $\{a, f\}$ est moyennement i-préféré.*

Les extensions fortement préférées concernent les définitions d'admissibilité basées sur les attaques complexes+.

Définition 35 (Extensions fortement préférées)
Soit $S \subseteq A$, S est une extension fortement d-préférée *(resp.* fortement s-préférée, fortement c-préférée, fortement i-préférée*) ssi S est maximal pour l'inclusion parmi les ensembles fortement d-admissibles (resp. fortement s-admissibles, fortement c-admissibles, fortement i-admissibles).*

Exemple 3 page 11 – Suite *L'ensemble $\{a, e, f, h\}$ est fortement d-préféré, les ensembles $\{e, h\}$ et $\{a, f\}$ sont fortement s-préférés. L'ensemble $\{e, h\}$ n'est pas fortement c-préféré. L'ensemble $\{a, f\}$ est fortement c-préféré et fortement i-préféré.*

Remarque : Au travers des exemples, on constate rapidement que les liens d'inclusion qui existaient entre les admissibilités disparaissent dès lors que l'on impose la maximalité pour l'inclusion :
- S s-préférée $\not\Rightarrow$ S d-préférée (quelle que soit la classe étudiée – faible, moyenne ou forte –) : voir l'exemple 3 page 11 avec les ensembles
  - $\{a, e, f\}$ qui est faiblement s-préféré et pas faiblement d-préféré ($\{a, c, e, f, h\}$ est faiblement d-préféré),
  - $\{a, f\}$ qui est moyennement s-préféré et pas moyennement d-préféré ($\{a, e, f, h\}$ est moyennement d-préféré),
  - $\{a, f\}$ qui est fortement s-préféré et pas fortement d-préféré ($\{a, e, f, h\}$ est fortement d-préféré).
- S c-préférée $\not\Rightarrow$ S s-préférée (quelle que soit la classe étudiée – faible, moyenne ou forte –) :
  - dans l'exemple 3 page 11, l'ensemble $\{a, f\}$ est faiblement c-préféré et pas faiblement s-préféré ($\{a, e, f\}$ est faiblement s-préféré),
  - dans l'exemple représenté par $a \rightarrow b \not\leftarrow c \rightarrow d \not\leftarrow e$, l'ensemble $\{e\}$ est moyennement c-préféré et pas moyennement s-préféré ($\{a, e\}$ est moyennement s-préféré),
  - dans l'exemple représenté par $a \rightarrow b \not\leftarrow c \rightarrow d \not\leftarrow e$, l'ensemble $\{e\}$ est fortement c-préféré et pas fortement s-préféré ($\{a, e\}$ est fortement s-préféré).

– S i-préférée $\not\Rightarrow$ S c-préférée (quelle que soit la classe étudiée – faible, moyenne ou forte –) :
  – dans l'exemple représenté par $a \not\rightarrow b \rightarrow c \not\rightarrow d \leftarrow e \leftarrow f$, l'ensemble $\{a\}$ est faiblement i-préféré et pas faiblement c-préféré ($\{a, c\}$ est faiblement c-préféré),
  – dans l'exemple représenté par $a \not\rightarrow b \rightarrow c \not\rightarrow d \leftarrow e \leftarrow f$, l'ensemble $\{a\}$ est moyennement i-préféré et pas moyennement c-préféré ($\{a, c\}$ est moyennement c-préféré),
  – dans l'exemple représenté par la figure 5.4, l'ensemble $\varnothing$ est fortement i-préféré et pas fortement c-préféré ($\{a, c\}$ et $\{a, b, d\}$ sont fortement c-préférés).

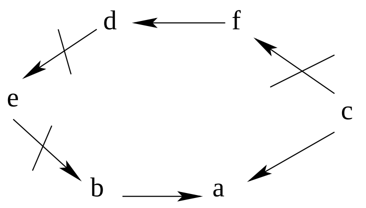


Fig. 5.4 – Exemple montrant que S fortement i-préféré n'est pas toujours fortement c-préféré

Comme pour les sémantiques admissibles, nous avons une propriété d'existence évidente :

Propriété 4 *Quelle que soit la classe de sémantique donnée (faible, moyenne ou forte) et quel que soit $x \in \{d, s, c, i\}$:*

1. *tout ensemble x-admissible est inclus dans une extension x-préférée ;*
2. *il existe toujours au moins une extension x-préférée dans cette classe.*

Preuve :

1. Il suffit de constater que toute chaîne d'ensembles x-admissibles admet un majorant pour la relation d'inclusion ensembliste. Plus précisément, pour une classe donnée, soit $Y_1 \subseteq Y_2 \subseteq \ldots \subseteq Y_n \subseteq \ldots$ une chaîne d'ensembles x-admissibles, on montre aisément que $\cup_{i \geq 1} Y_i$ est encore un ensemble x-admissible. Cela repose sur le fait que les différentes définitions (attaque directe, appuyée, détournée, séquence d'appuis) utilisent des séquences finies d'arcs d'attaque ou d'appui.
2. La preuve repose sur le fait qu'il existe toujours au moins un ensemble x-admissible (éventuellement l'ensemble vide – voir la propriété 3 page 23) et que l'on peut appliquer le résultat 4. 1.

□

On a aussi la conséquence suivante :

Conséquence 2 *Pour une classe de sémantique donnée (faible, moyenne ou forte), et pour $x = s$ (resp. $x = c$, $x = i$) et $y = d$ (resp. $y = s$, $y = c$), on a :*
si $S \subseteq A$ est x-préféré pour la classe donnée, alors $\exists S' \subseteq A$ y-préféré pour la classe donnée et tel que $S \subseteq S'$.

Preuve : Si S est x-préféré alors S est x-admissible et par la conséquence 1 page 22, S est aussi y-admissible. D'après le point 1 de la propriété 4, S est alors inclus dans une extension y-préférée. □

5.4 Extensions stables

En exploitant la propriété 2 page 20, on constate que les notions de clôture et d'indépendance pour la relation d'appui, dans le cadre *stable*, peuvent être retrouvées à partir des notions de sans-conflit et de sûreté. On aura donc à définir seulement les extensions d-stables et s-stables.

Définition 36 (Extensions faiblement (resp. moyennement, fortement) d-stables) *Soit $S \subseteq A$. S est un ensemble* faiblement (resp. moyennement, fortement) d-stable *ssi S est sans conflit-direct (resp. sans-conflit-complexe, sans-conflit-complexe+) et $\forall a \notin S$, S attaque directement (resp. de manière complexe, complexe+) a.*

Définition 37 (Extensions faiblement (resp. moyennement, fortement) s-stables) *Soit $S \subseteq A$. S est un ensemble* faiblement (resp. moyennement, fortement) s-stable *ssi S est directement sûr (resp. complexe-sûr, complexe+-sûr) et $\forall a \notin S$, S attaque directement (resp. de manière complexe, complexe+) a.*

Exemple 11 *Le graphe suivant représente le SABP :*
$<A=\{a,b,c,d\},\ R_{att} = \{(a,b),(c,d),(d,c)\},\ R_{app} = \{(d,b)\}>$. *Sur ce graphe, les ensembles*

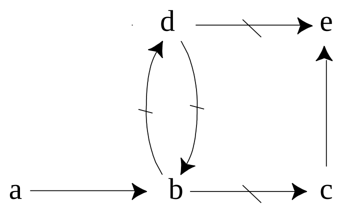


FIG. 5.5 – Exemple 11

$\{a,d,c\}$ et $\{a,b,e\}$ sont faiblement d-stables.
L'ensemble $\{a,d,c\}$ n'est pas faiblement s-stable, alors que l'ensemble $\{a,b,e\}$ est faiblement s-stable.
Seul l'ensemble $\{a,b,e\}$ est moyennement d-stable et moyennement s-stable.
Et il n'existe aucun ensemble fortement d-stable ou fortement s-stable.

Nous obtenons les propriétés suivantes classées suivant les classes de sémantiques concernées :

Propriété 5

1. *Pour les liens internes à la classe faible :*
   (a) *Soit $S \subseteq A$, si S est une extension faiblement s-stable alors S est faiblement d-stable.*
   (b) *Soit $S \subseteq A$, si S est une extension faiblement d-stable alors S est faiblement d-admissible.*
   (c) *Soit $S \subseteq A$, S est une extension faiblement d-stable et S est clos pour la relation R_{app}[5] ssi S est une extension faiblement s-stable.*
   (d) *Soit $S \subseteq A$, si S est une extension faiblement d-stable et indépendante pour la relation R_{app}[6] alors S est une extension faiblement s-stable. La réciproque est fausse.*
2. *Pour les liens internes à la classe moyenne :*
   (a) *Soit $S \subseteq A$, si S est une extension moyennement s-stable alors S est moyennement d-stable.*
   (b) *Soit $S \subseteq A$, si S est une extension moyennement d-stable alors S n'est pas toujours moyennement d-admissible.*
   (c) *Soit $S \subseteq A$, S est une extension moyennement d-stable et S est clos pour la relation R_{app}[7] ssi S est une extension moyennement s-stable.*
   (d) *Soit $S \subseteq A$, si S est une extension moyennement d-stable et indépendante pour la relation R_{app}[8] alors S est une extension moyennement s-stable. La réciproque est fausse.*

[5]Ce qui pourrait correspondre au fait que S est faiblement c-stable, si on avait défini la c-stabilité faible.
[6]Ce qui pourrait correspondre au fait que S est faiblement i-stable, si on avait défini la i-stabilité faible.
[7]Ce qui pourrait correspondre au fait que S est moyennement c-stable, si on avait défini la c-stabilité moyenne.
[8]Ce qui pourrait correspondre au fait que S est moyennement i-stable, si on avait défini la i-stabilité moyenne.

3. Pour les liens internes à la classe forte :

(a) Soit $S \subseteq A$, si S est une extension fortement s-stable alors S est fortement d-stable.

(b) Soit $S \subseteq A$, si S est une extension fortement d-stable alors S n'est pas toujours fortement d-admissible.

(c) Soit $S \subseteq A$, si S est une extension fortement s-stable alors S est indépendante pour la relation R_{app}.

(d) Soit $S \subseteq A$. S est une extension fortement d-stable et indépendante pour la relation R_{app}[9] ssi S est une extension fortement s-stable.

(e) Soit $S \subseteq A$. S est une extension fortement d-stable et S est clos pour la relation R_{app}[10] ssi S est une extension fortement s-stable.

Preuve :

1. Pour la classe faible :
   (a) Conséquence de la propriété 2 page 20.
   (b) Soit $S \subseteq A$ un ensemble d'arguments faiblement d-stable. Supposons deux arguments a et b tels que $a \in A$ et $b \in S$. Si $\{a\}$ attaque directement b alors $a \in A \backslash S$ puisque S est sans conflit-direct. Or, S attaque directement a car S est faiblement d-stable. Donc S défend b et donc S est faiblement d-admissible.
   (c) ($\Rightarrow$)Soit S un ensemble faiblement d-stable et clos pour la relation R_{app}. Par la propriété 2 page 20, S est directement sûr. Donc, S est directement sûr et attaque tous les arguments n'appartenant pas à S. S est faiblement s-stable.
   ($\Leftarrow$) Soit S un ensemble faiblement s-stable. Par la propriété 5.1a, S est faiblement d-stable. De plus, il n'existe pas d'argument $a \in A \backslash S$ tel que S appuie directement ou de manière complexe a, puisque S est s-stable. Donc, S est faiblement d-stable et clos pour la relation R_{app}.
   (d) ($\Rightarrow$)Preuve analogue à la partie ($\Rightarrow$) de celle de la propriété 5.1c.
   ($\Leftarrow$) Sur le contre-exemple suivant, $a \not\rightarrow b \rightarrow c$, l'ensemble $\{a, c\}$ est faiblement s-stable, donc faiblement d-stable, mais n'est pas indépendant pour la relation R_{app}.
2. Pour la classe moyenne :
   (a) Conséquence de la propriété 2 page 20.
   (b) Sur l'exemple de la figure 5.6, l'ensemble $\{a, d, e\}$ est moyennement d-stable mais n'est pas moyennement d-admissible (l'argument d n'étant pas défendu).

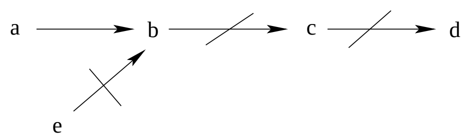


FIG. 5.6 – Exemple d'un ensemble moyennement d-stable et pas moyennement d-admissible

   (c) ($\Rightarrow$)Preuve analogue à la partie ($\Rightarrow$) de celle de la propriété 5.1c.
   ($\Leftarrow$) On sait déjà par la propriété 5.2a, qu'un ensemble moyennement s-stable est aussi moyennement d-stable. Vérifions maintenant qu'il est aussi clos. Supposons un ensemble $S \subseteq A$ d'arguments qui est moyennement s-stable. Soit $b \in A \backslash S$ un argument tel que S appuie de manière complexe b, alors, comme S est moyennement s-stable, S attaque de manière complexe b. Donc S appuie de manière complexe et attaque de manière complexe b. C'est impossible puisque S est complexe-sûr par définition. Donc S est clos pour la relation R_{app}.
   (d) ($\Rightarrow$)Conséquence de la propriété 2 page 20.
   ($\Leftarrow$) Il suffit d'utiliser le contre-exemple suivant : $a_1 \rightarrow a_2 \not\rightarrow b \rightarrow a_1$. Dans ce cas l'ensemble $\{a_1, a_2\}$ est moyennement s-stable et pourtant il n'est pas indépendant pour la relation R_{app}.

[9] Ce qui pourrait correspondre au fait que S est une extension fortement i-stable, si on avait défini la i-stabilité forte.

[10] Ce qui pourrait correspondre au fait que S est fortement c-stable, si on avait défini la c-stabilité forte.

3. Pour la classe forte :

   (a) Conséquence de la propriété 2 page 20.

   (b) Sur l'exemple suivant, $a \not\leftarrow b \leftarrow c \not\leftarrow d$, l'ensemble $\{a, d\}$ est fortement d-stable mais n'est pas fortement d-admissible (l'argument a n'étant pas défendu).

   (c) Supposons un ensemble $S \subseteq A$ d'arguments qui est fortement s-stable mais qui n'est pas indépendant pour la relation R_{app}.
      – Soit $b \in A \backslash S$ un argument tel que S appuie de manière complexe+ b, alors, comme S est fortement s-stable, S attaque de manière complexe+ b. Donc S appuie de manière complexe+ et attaque de manière complexe b. C'est impossible puisque S est complexe+-sûr par définition. Donc S est clos pour la relation R_{app}.
      – Soit $b \in A \backslash S$ un argument tel que $\{b\}$ appuie de manière complexe S, c'est-à-dire $\exists a \in S$ tel que $\{b\}$ appuie de manière complexe a. Or, S attaque de manière complexe b puisque S est fortement s-stable. S attaque b par attaque directe ou détournée seulement puisque S est clos. Donc S attaque de manière complexe+ a (par attaque détournée). C'est impossible puisque S est complexe+-sûr par définition. Donc, il n'existe pas d'argument $b \in A \backslash S$ tel que $\{b\}$ appuie de manière complexe S.

      S est clos et il n'existe pas d'argument $b \in A \backslash S$ tel que $\{b\}$ appuie de manière complexe S, S est donc indépendant pour la relation R_{app}.

   (d) ($\Rightarrow$)Preuve analogue à la partie ($\Rightarrow$) de celle de la propriété 5.1c.
   ($\Leftarrow$)Conséquence des propriétés 5.3a et 5.3c.

   (e) ($\Rightarrow$)Conséquence de la propriété 2 page 20. ($\Leftarrow$)Conséquence de la propriété 5.3d.

□

Comme dans le cadre unipolaire, nous avons aussi une propriété de non-existence évidente :

Propriété 6 *Quelle que soit la classe de sémantique donnée (faible, moyenne ou forte) et quel que soit $x \in \{d, s\}$, il n'existe pas toujours une extension x-stable dans cette classe.*

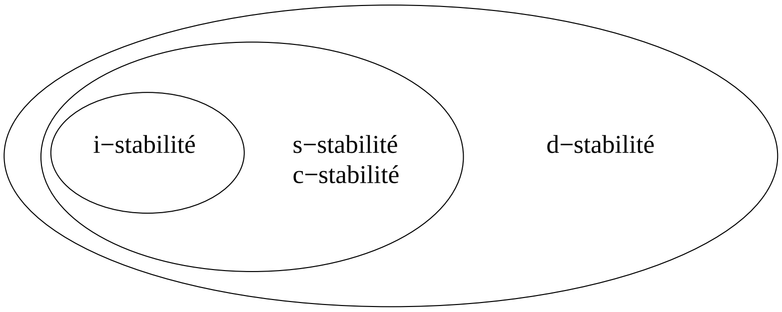
> **Preuve :** La preuve repose sur le contre-exemple suivant (qui est aussi celui utilisé dans le cadre unipolaire par Dung pour montrer la non-existence systématique des extensions stables).
>
> Soit le système d'argumentation réduit à un seul argument a et aux relations $R_{att} = \{(a, a)\}$ et $R_{app} = \varnothing$. Dans ce cas, il ne peut pas y avoir d'extension stable quel que soit le type de stabilité puisque qu'il n'existe que deux ensembles possibles :
> – $\{a\}$ qui contient un conflit direct (et donc ne pourra pas être stable),
> – et $\varnothing$ qui ne possède aucun conflit mais ne peut pas attaquer l'argument a (et donc ne pourra pas non plus être stable).
>
> □

Les liens entre les différentes sémantiques stables peuvent être représentées par la figure 5.7 page 30.

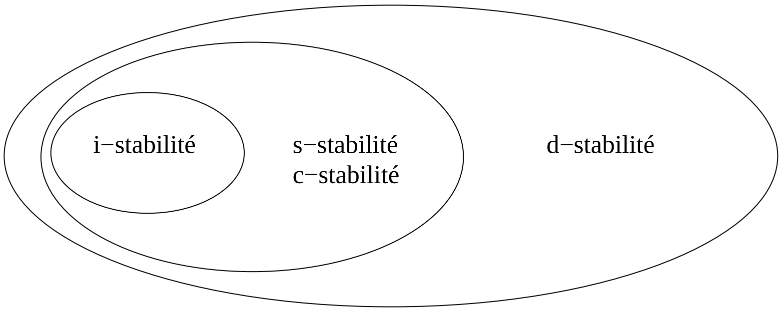
5.5 Synthèse

Voici un tableau qui reprend toutes les notions définies précédemment :

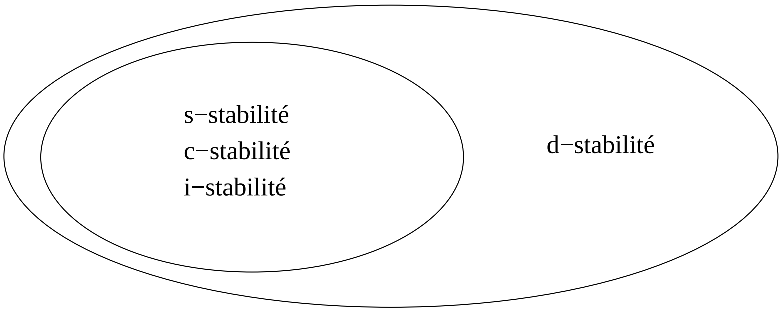
	Extensions préférées												Extensions stables					
	Faibles				Moyennes				Fortes				Faibles		Moyennes		Fortes	
	d	s	c	i	d	s	c	i	d	s	c	i	d	s	d	s	d	s
Prise en compte de l'attaque directe	X	X	X	X									X	X				
Prise en compte de l'attaque complexe					X	X	X	X							X	X		
Prise en compte de l'attaque complexe+									X	X	X	X					X	X
Sans conflit (cohérence interne)	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Sûr (cohérence externe)		X				X				X				X		X		X
Clos pour la relation R_{app}			X	X			X	X			X	X		X		X		X
Indépendant pour la relation R_{app}				X				X				X						X
Défense (par rapport à une attaque directe)	X	X	X	X	X	X	X	X	X	X	X	X	X	X				



Cas de la classe faible



Cas de la classe moyenne



Cas de la classe forte

Fig. 5.7 – Lien entre les différentes sémantiques stables

Chapitre 6

Algorithmique

Dans cette section, nous allons nous intéresser à l'algorithmique du calcul d'extension pour la sémantique préférée. Nous allons pour cela nous inspirer des algorithmes pour la sémantique préférée dans le cadre de systèmes d'argumentation unipolaires proposés par S. Doutre [Dou02].

Le lecteur notera que ce chapitre est une première ébauche de l'algorithmique pour l'acceptabilité en argumentation bipolaire et qu'en conséquence seules quelques sémantiques ont été implémentées : calcul des ensembles faiblement d-admissibles, faiblement s-admissibles et des extensions faiblement d-préférées et faiblement s-préférées.

Toutefois, les algorithmes sont suffisamment génériques pour être applicables aux autres sémantiques, une fois que les conditions particulières à chaque sémantique auront été identifiées.

6.1 Algorithmes pour les systèmes d'argumentation unipolaires

6.1.1 Énumération des parties d'un ensemble

L'algorithme de calcul d'extensions est basé sur le principe d'énumération de parties. L'énumération des sous-ensembles d'un ensemble A est effectuée par l'exploration d'un arbre binaire dont les nœuds sont étiquetés par une partition de A en trois ensembles I, O et U où :

- I est un ensemble d'éléments qui sera inclus dans tout sous-ensemble de A trouvé au niveau d'une feuille descendante du nœud courant.
- O est un ensemble d'éléments qui ne seront dans aucun sous-ensemble de A trouvé au niveau d'une feuille descendante du nœud courant.
- $U = A \backslash (I \cup O)$ est un ensemble d'éléments dont le statut n'est pas encore décidé.

Si $U = \emptyset$ alors le nœud courant est une feuille.
La fonction $EnumParties$ permet donc d'énumérer tous les sous-ensembles d'un ensemble donné en argument. Elle fait appel à la fonction $EnumParties_{rec}$ qui, à l'aide de la fonction $Select$ dont le rôle est de choisir un élément arbitrairement, s'effectue récursivement sur deux nouvelles partitions : dans la première, x est ajouté à I (« IN ») et dans la seconde x est ajouté à O (« OUT »). Elle effectue ensuite l'union des deux résultats, c'est-à-dire l'union de l'ensemble des sous-ensembles contenant x et de l'ensemble des sous-ensembles ne contenant pas x.

Fonction $EnumParties_{rec}(\ I, O, U$: **ensembles disjoints**) : **ensemble de sous-ensembles**

Résultat : $\{S \mid I \subseteq S \subseteq (I \cup U)\}$

Si $(U = \emptyset)$ **Alors**

  Retourner $(\{I\})$

Sinon

  $x \leftarrow Select(U)$;

  Retourner $(EnumParties_{rec}(I \cup \{x\}, O, U \backslash \{x\}) \cup (EnumParties_{rec}(I, O \cup \{x\}, U \backslash \{x\})$

Fin Si

Fin

Algorithme 1: Énumération des parties d'un ensemble (fonction intermédiaire récursive)

Fonction $EnumParties(\ A$: **ensemble**) : **ensemble de sous-ensembles**

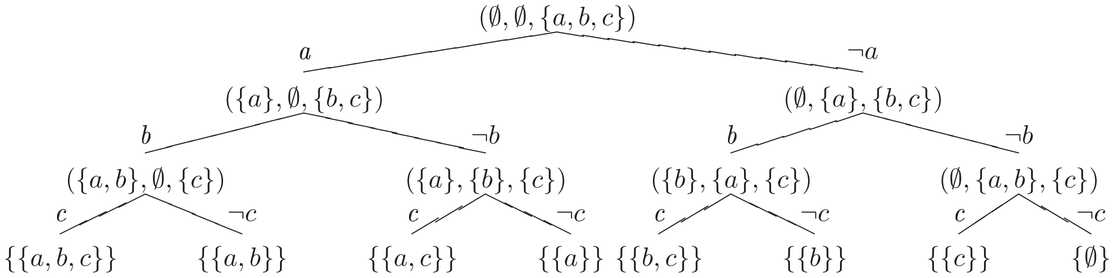
Résultat : $\{S \mid S \subseteq A\}$

Retourner $(EnumParties_{rec}(\emptyset, \emptyset, A))$;

Fin

Algorithme 2: Énumération des parties d'un ensemble

Exemple 12 *Le schéma suivant représente l'arbre binaire d'énumération des sous-ensembles de l'ensemble $A = \{a, b, c\}$ du système d'argumentation $< A, R >$ avec $R = \{(c, b), (b, a)\}$. Le graphe $c \not\rightarrow b \not\rightarrow a$ représente ce système d'argumentation. A chaque nœud est donné le triplet (I, O, U) et seul l'ensemble I est donné au niveau des feuilles de l'arbre binaire puisque $U = \emptyset$, l'ensemble O peut être déterminé d'après l'ensemble I, $O = A \backslash I$.*



6.1.2 Énumération des parties sans-conflit d'un ensemble

La fonction d'énumération des parties sans-conflit d'un ensemble repose sur la fonction précédente. Elle s'assure à chaque étape que l'ensemble I est *sans-conflit*, étant donné une relation d'attaque R, c'est-à-dire qu'il n'y a pas, dans un ensemble solution, deux arguments x et y de A tels que le couple (x, y) appartient à R.

Soit $R^+(x)$ l'ensemble des successeurs (resp. $R^-(x)$ l'ensemble des prédécesseurs) pour un argument x, défini par $R^+(x) = \{y | (x, y) \in R\}$ (resp. $R^-(x) = \{y | (y, x) \in R\}$) et $R^+(S) = \bigcup_{x \in S} R^+(x)$ (resp. $R^-(S) = \bigcup_{x \in S} R^-(x)$) pour un ensemble d'arguments. $R^+(S)$ est l'ensemble des arguments qui sont attaqués par les arguments de l'ensemble S et $R^-(S)$ est l'ensemble des arguments qui attaquent les arguments de l'ensemble S.

On définit alors l'ensemble $R^{+-}(x) = R^+(x) \bigcup R^-(x)$ quel que soit x.

Soit $Refl(A, R)$ l'ensemble des éléments réflexifs, selon la relation R, de l'ensemble A.

$Refl(A, R) = \{x | x \in A\ et\ (x, x) \in R\} = \{x | x \in A\ et\ x \cap R^+(x) \neq \emptyset\}$.
La fonction $EnumPartiesSansconflit$ permet d'énumérer tous les sous-ensembles *sans-conflit* d'un ensemble, en fonction d'une relation, donnés en argument. Elle fait appel à la fonction $EnumPartiesSansconflit_{rec}$ en supprimant tous les arguments réflexifs puisqu'ils ne font partie d'aucun ensemble *sans-conflit*. Cette dernière s'effectue récursivement sur deux nouvelles partitions : dans la première, x est ajouté à I et tous les prédécesseurs et successeurs sont ajoutés à O, et dans la seconde x est ajouté à O. Grâce à cela la fonction s'assure à chaque étape que l'ensemble I est *sans-conflit*. Elle effectue ensuite l'union des deux résultats, c'est-à-dire l'union de l'ensemble des sous-ensembles *sans-conflit* contenant x et de l'ensemble de ceux qui ne contiennent pas x.

Propriété 7 *Soit S un ensemble quelconque d'arguments, S est* sans-conflit *ssi* $S \cap R^+(S) = \emptyset$.

Preuve : La preuve est triviale. □

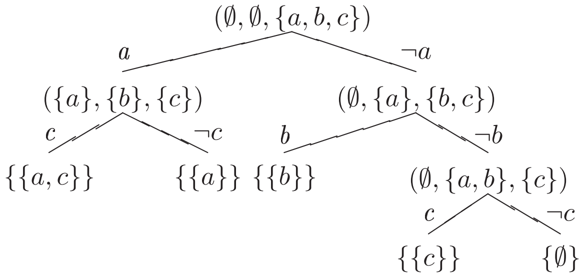
Fonction $EnumPartiesSansconflit_{rec}$(R : **relation binaire** ,I, O, U : **ensembles disjoints**) : **ensemble de sous-ensembles**
  Résultat : $\{S \mid I \subseteq S \subseteq ((I \cup U) \backslash R^{+-}(I))\}$
  Si ($U = \emptyset$) **Alors**
    Retourner ($\{I\}$)
  Sinon
    $x \leftarrow Select(U)$;
    Retourner ($EnumPartiesSansconflit_{rec}(R, I \cup \{x\}, O \cup R^{+-}(x), U \backslash (\{x\} \cup R^{+-}(x)))$ $\cup$ $(EnumPartiesSansconflit_{rec}(R, I, O \cup \{x\}, U \backslash \{x\}))$);
  Fin Si
Fin

Algorithme 3: Énumération des parties sans-conflit d'un ensemble (fonction intermédiaire récursive)

Fonction $EnumPartiesSansconflit$(A : **ensemble** ,R : **relation**) : **ensemble de sous-ensembles**
  Résultat : $\{S \mid S \subseteq A\ et\ S \cap R^+(S) = \emptyset\}$
  Retourner ($EnumPartiesSansconflit_{rec}(R, \emptyset, Refl(A, R), A \backslash Refl(A, R))$);
Fin

Algorithme 4: Énumération des parties sans-conflit d'un ensemble

Exemple 12 page précédente – Suite *L'arbre binaire suivant représente l'énumération de tous les sous-ensembles* sans-conflit *de l'ensemble A de l'exemple 12 page ci-contre.*



6.1.3 Énumération des sous-ensembles admissibles

La fonction d'énumération des sous-ensembles admissibles repose sur la fonction précédente. Elle effectue simplement un test sur chaque sous-ensemble *sans-conflit* (c'est-à-dire sur l'ensemble I de chaque feuille de l'arbre binaire) afin de déterminer s'il est *admissible*.
La fonction $EnumAdm$ permet d'énumérer tous les sous-ensembles *admissibles* d'un ensemble, en fonction d'une relation, donnés en argument. Elle fait appel à la fonction $EnumAdm_{rec}$ qui suit le même principe que la fonction $EnumPartiesSansconflit_{rec}$. Si le nœud courant est une feuille (l'ensemble I de ce nœud est donc *sans-conflit*), représenté par la condition $U = \emptyset$, et si l'ensemble des arguments qui attaquent I est inclus dans l'ensemble des arguments attaqués par I (c'est-à-dire si I se défend), représenté par la condition $R^-(I)\backslash R^+(I) = \emptyset$, alors l'ensemble I est admissible. De plus, si, à un nœud quelconque, un argument attaque l'ensemble I et que cet argument n'est attaqué ni par l'ensemble I ni l'ensemble U, alors l'énumération dans cette branche de la recherche est stoppée. En effet, sous cette condition, $\exists x \in (R^-(I)\backslash R^+(I))$ *tel que* $R^-(x) \subseteq O$, aucun ensemble contenant I ne peut être admissible et par conséquent aucune feuille descendante de ce nœud ne possède un ensemble I admissible (résultat démontré par S. Doutre [Dou02]).

Fonction $EnumAdm_{rec}($ R : **relation binaire** $,I,O,U$: **ensembles disjoints**) : **ensemble de sous-ensembles**
Résultat : $\{S \mid I \subseteq S \subseteq ((I \cup U)\backslash R^{+-}(I))\ et\ R^-(S) \subseteq R^+(I \cup U)\}$
Si $(U = \emptyset\ et\ R^-(I)\backslash R^+(I) = \emptyset)$ **Alors**
  Retourner $(\{I\})$
Sinon
  Si $(\exists x \in (R^-(I)\backslash R^+(I))\ tel\ que\ R^-(x) \subseteq O)$ **Alors**
    Retourner $(\{\})$
  Sinon
    $x \leftarrow Select(U)$;
    Retourner ($EnumAdm_{rec}(R, I \cup \{x\}, O \cup R^{+-}(x), U\backslash(\{x\} \cup R^{+-}(x)))$ $\cup$ $(EnumAdm_{rec}(R, I, O \cup \{x\}, U\backslash\{x\}))$);
  Fin Si
Fin Si
Fin

Algorithme 5: Énumération des sous-ensembles admissibles (fonction intermédiaire récursive)

Fonction $EnumAdm($ A : **ensemble** $,R$: **relation**) : **ensemble de sous-ensembles**
Résultat : $\{S \mid S \subseteq A\ et\ S \cap R^+(S) = \emptyset\ et\ R^-(S) \subseteq R^+(S)\}$
Retourner $(EnumAdm_{rec}(R, \emptyset, Refl(A,R), A\backslash Refl(A,R)))$;
Fin

Algorithme 6: Énumération des sous-ensembles admissibles

Remarque : Dans la partie *SINON* de la condition $\exists x \in (R^-(I)\backslash R^+(I))$ *tel que* $R^-(x) \subseteq O$, nous sommes certains que $U \neq \emptyset$ car si $\exists x \in (R^-(I)\backslash R^+(I))$ *tel que* $R^-(x) \not\subseteq O$ alors $\exists z \in R^-(x)$

tel que $z \notin I$ (car sinon $x \notin (R^-(I) \backslash R^+(I))$) et $z \notin O$ (car sinon $R^-(x) \subseteq O$). Donc z appartient à l'ensemble U.

Exemple 13 *Soit* $< A, R > = < \{a, b, c, d\}, \{(d, c), (c, d), (c, b), (b, b), (b, a)\} >$ *un système d'argumentation représenté graphiquement par : L'arbre binaire d'énumération des parties admissibles*

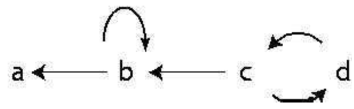
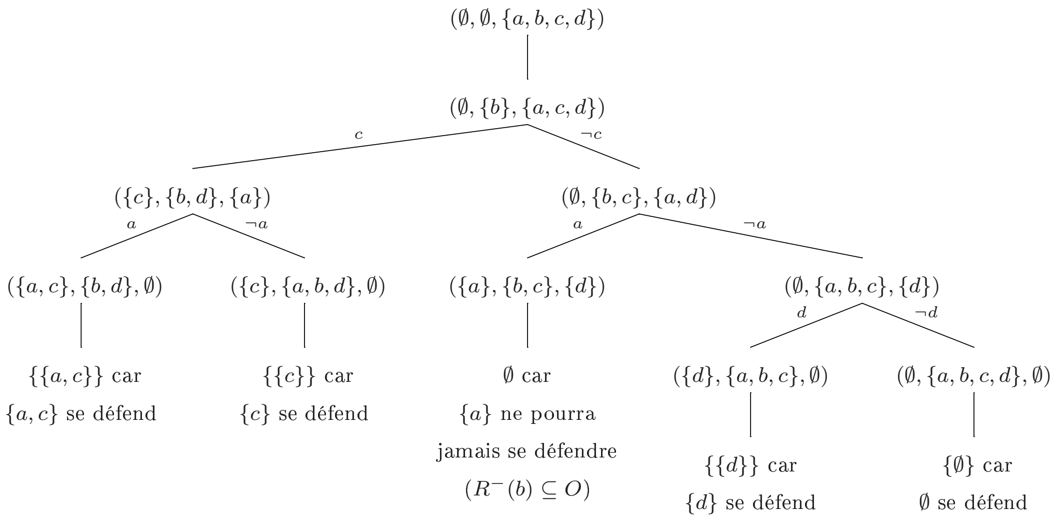


FIG. 6.1 – Exemple 13

de A est le suivant :



Le résultat obtenu est donc :
$\{\{a, c\}\} \cup \{\{c\}\} \cup \emptyset \cup \{\{d\}\} \cup \{\emptyset\} = \{\{a, c\}, \{c\}, \{d\}, \emptyset\}$.

6.1.4 Énumération des extensions préférées

Parmi les ensembles admissibles, on trouve les extensions préférées. L'algorithme d'énumération des sous-ensembles admissibles peut être modifié pour énumérer les extensions préférées. S. Doutre [Dou02] propose un algorithme intégrant des heuristiques pour effectuer l'énumération des extensions préférées dans un système d'argumentation unipolaire. Nous ne présenterons pas cet algorithme puisque nous restons dans le cadre d'algorithmes « naïfs »(c'est-à-dire sans heuristique). Nous proposerons ainsi des algorithmes pour l'énumération d'extensions faiblement préférées en nous inspirant uniquement de la fonction $EnumAdm$.

6.2 Algorithmes pour les systèmes d'argumentation bipolaires

A l'aide des algorithmes décrits dans la section précédente, nous allons déterminer des algorithmes permettant l'énumération des extensions *faiblement préférées* de la définition 33 page 24.
Pour cela, nous proposerons des algorithmes pour :

1. énumérer les sous-ensembles *sans conflit-direct*,

2. énumérer les sous-ensembles *directement sûrs*,
3. énumérer les sous-ensembles faiblement d-admissibles,
4. énumérer les sous-ensembles faiblement s-admissibles,
5. énumérer les extensions faiblement d-préférées et s-préférées.

Pour l'algorithme d'énumération des sous-ensembles *sans conflit-direct*, nous reprendrons la fonction $EnumPartiesSansconflit$ en lui apportant les modifications nécessaires pour pouvoir l'utiliser dans un cadre bipolaire. Pour celui de l'énumération des sous-ensembles *directement sûr*, nous proposerons un nouvel algorithme inspiré de la fonction $EnumPartiesSansconflit$. Les algorithmes pour l'énumération des sous-ensembles faiblement d-admissibles et s-admissibles et les extensions faiblement d-préférées et s-préférées se baseront respectivement sur les deux algorithmes précédents.

6.2.1 Algorithme d'énumération des sous-ensembles *sans conflit-direct*

Voici l'algorithme permettant l'énumération des sous-ensembles sans conflit-direct de A, appliqué à un système d'argumentation bipolaire $< A, R_{att}, R_{app} >$. L'algorithme est identique à celui de la fonction $EnumPartiesSansconflit$, la seule nouveauté est la présence de la relation binaire R_{app} dans la fonction principale. La relation binaire R a été remplacée par la relation binaire R_{att} et l'ensemble $R^{+-}_{att}(x)$ est renvoyé par la fonction $Conflit(x, R_{att})$. On rappelle qu'un ensemble S est *sans conflit-direct* ssi $S \cap R^{+}_{att}(S) = \emptyset$. La fonction $Refl$ est définie par $Refl(A, R) = \{x | x \in A \ et \ x \cap R^{+}(x) \neq \emptyset\}$ avec les nouvelles notations, la fonction devient :
$Refl(A, R_{att}) = \{x | x \in A \ et \ x \cap R^{+}_{att}(x) \neq \emptyset\}$.

Fonction $EnumPartiesSansconflitSABP_{rec}($ R_{att} : **relation binaire** $, I, O, U$: **ensembles disjoints**) : **ensemble de sous-ensembles**
  Résultat : $\{S \mid I \subseteq S \subseteq ((I \cup U) \backslash Conflit(I, R_{att}))\}$
  Si ($U = \emptyset$) **Alors**
    Retourner ($\{I\}$)
  Sinon
    $x \leftarrow Select(U)$;
    Retourner ($EnumPartiesSansconflitSABP_{rec}(R_{att}, I \cup \{x\}, O \cup Conflit(x, R_{att}), U \backslash (\{x\} \cup Conflit(x, R_{att}))) \cup (EnumPartiesSansconflitSABP_{rec}(R_{att}, I, O \cup \{x\}, U \backslash \{x\}))$;
  Fin Si
Fin

Algorithme 7: Énumération des parties sans-conflit d'un SABP (fonction intermédiaire récursive)

Fonction $EnumPartiesSansconflitSABP($ A : **ensemble** $, R_{att}, R_{app}$: **relations binaires**) : **ensemble de sous-ensembles**
  Résultat : $\{S \mid S \subseteq A \ et \ S \cap R^{+}_{att}(S)) = \emptyset\}$
  Retourner ($EnumPartiesSansconflitSABP_{rec}(R_{att}, \emptyset, Refl(A, R_{att}), A \backslash Refl(A, R_{att}))$);
Fin

Algorithme 8: Énumération des parties sans-conflit d'un SABP

Exemple 14 *Soit le SABP suivant : Voici l'arbre binaire d'énumération des sous-ensembles sans conflit-direct de ce système d'argumentation.*

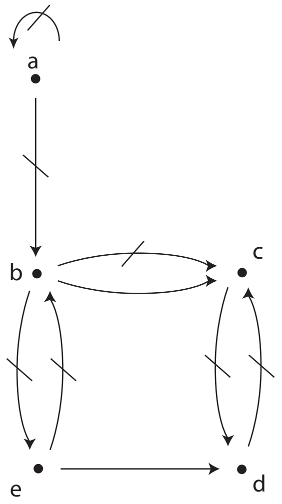
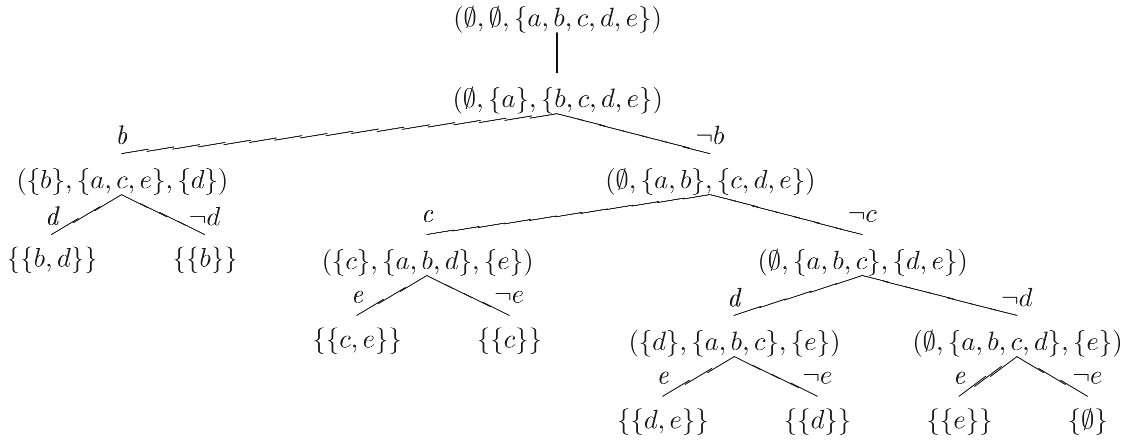


FIG. 6.2 – Exemple 14



6.2.2 Algorithme d'énumération des sous-ensembles *directement sûrs*

L'algorithme suivant permet d'énumérer les sous-ensembles directement sûrs. Il suit le même principe que la fonction précédente, c'est-à-dire qu'il crée, à chaque étape, des nœuds dont l'ensemble I est directement sûr.
La fonction $EnumPartiesSurSABP$ permet d'énumérer tous les sous-ensembles directement sûrs d'un ensemble, en fonction des deux relations R_{att} et R_{app}, donnés en argument. Elle fait appel à la fonction $EnumPartiesSurSABP_{rec}$ en supprimant tous les arguments non directement sûrs puisqu'ils ne peuvent pas faire partie d'un ensemble directement sûr. La fonction $EnumPartiesSurSABP_{rec}$ s'appelle récursivement sur deux nouvelles partitions : dans la première, x est ajouté à I et l'ensemble renvoyé par la fonction $NonSur$ est ajouté à O, et dans la seconde x est ajouté à O. Elle effectue ensuite l'union des deux résultats, c'est-à-dire l'union de l'ensemble des sous-ensembles directement sûrs contenant x et de l'ensemble de ceux qui ne contiennent pas x.

La fonction $NonSur(x, R_{att}, R_{app})$ renvoie un ensemble dont chaque élément interdit la propriété d'ensemble directement sûr lorsqu'il est associé à x. Autrement dit, pour tout élément y de $NonSur(x, R_{att}, R_{app})$, si S est un ensemble tel qu'il contient x et y alors S n'est pas directement sûr. La fonction $NonSur(x)$ est définie par :
$NonSur(x, R_{att}, R_{app}) = R^{+-}_{att}(x) \cup R^{-}_{app}(R^{+}_{att}(x)) \cup R^{-}_{att}(R^{+}_{app}(x))$ pour un argument x.
La fonction $NonSur(x, R_{att}, R_{app})$ renvoie donc un ensemble résultant de l'union de :

- $R^{+-}_{att}(x)$: l'ensemble des éléments qui attaquent x ou sont attaqués par x,
- $R^{-}_{app}(R^{+}_{att}(x))$: l'ensemble des éléments qui appuient ceux que x attaque,
- $R^{-}_{att}(R^{+}_{app}(x))$: l'ensemble des éléments qui attaquent ceux que x appuie.

On posera :
$NonSur(S, R_{att}, R_{app}) = \bigcup_{x\in S} NonSur(x, R_{att}, R_{app})$ pour un ensemble quelconque S[1].

Propriété 8 *Soit S un sous-ensemble quelconque d'arguments de A, S est directement sûr ssi $S \cap NonSur(S) = \emptyset$.*

Preuve :

- ($\Rightarrow$) Soit S un ensemble d'arguments directement sûr et y un argument tel que $y \in NonSur(S)$. Montrons que $y \notin S$:
  $\exists x \in S$ tel que $y \in NonSur(x)$ et
  - soit $y \in R^{+-}_{att}(x)$ et puisque $x \in S$ et S est directement sûr alors $y \notin S$.
  - soit $y \in R^{-}_{app}(R^{+}_{att}(x))$, donc il existe un argument z tel que $y \to z \not\leftarrow x$, et puisque S est directement sûr alors $y \notin S$.
  - soit $y \in R^{-}_{att}(R^{+}_{app}(x))$, donc il existe un argument z tel que $y \not\to z \leftarrow x$, et puisque S est directement sûr alors $y \notin S$.
- ($\Leftarrow$) Supposons que $S \cap NonSur(S) = \emptyset$, montrons que S est directement sûr.
  Supposons par l'absurde que S n'est pas directement sûr. Donc $\exists z \in A$ et $\exists x \in S$ tel que :
  - soit $x \not\to z$ et $z \in S$, alors $z \in S$ et $z \in NonSur(S)$ d'où $S \cap NonSur(S) \neq \emptyset$. Il y a contradiction puisque $S \cap NonSur(S) = \emptyset$.
  - soit $x \not\to z$ et $\exists y \in S$ tel que $y \to z$, alors $y \in S$ et $y \in R^{-}_{app}(R^{+}_{att}(x))$ donc $y \in NonSur(S)$. Il y a contradiction puisque $S \cap NonSur(S) = \emptyset$.

□

La fonction $Filtre(A, R_{att}, R_{app})$ détermine l'ensemble des éléments de l'ensemble A dont on sait qu'ils ne font pas partie d'un ensemble sûr. Nous définissons :
$Filtre(A, R_{att}, R_{app}) = \{x | x \in A \; et \; x \in NonSur(x, R_{att}, R_{app})\}$.
Ou, de façon équivalente :
$Filtre(A, R_{att}, R_{app}) = \{x|x \in A \; et \; x \cap R^{+}_{att}(x) \neq \emptyset\} \cup \{x|x \in A \; et \; (R^{+}_{att}(x) \cap R^{+}_{app}(x)) \neq \emptyset\}$
C'est-à-dire que $Filtre$ renvoie l'ensemble des éléments qui s'auto-attaquent ou qui attaquent et appuient un même autre argument.

[1]Sera noté $NonSur(S)$ lorsqu'il n'y aura pas d'ambigu¨é sur R_{att} et R_{app}.

Fonction $EnumPartiesSurSABP_{rec}($ R_{att}, R_{app} : **relations binaires** ,I, O, U : **ensembles disjoints**) : **ensemble de sous-ensembles**

Résultat : $\{S \mid I \subseteq S \subseteq ((I \cup U)\backslash NonSur(I))\}$

Si $(U = \emptyset)$ **Alors**

  Retourner $(\{I\})$

Sinon

  $x \leftarrow Select(U)$;

  Retourner ($EnumPartiesSurSABP_{rec}(R_{att}, R_{app}, I \cup \{x\}, O \cup NonSur(x, R_{att}, R_{app}),$ $U\backslash(\{x\} \cup NonSur(x, R_{att}, R_{app})))$ $\cup$ $(EnumPartiesSurSABP_{rec}(R_{att}, R_{app}, I, O \cup \{x\}, U\backslash$ $\{x\}))$) ;

Fin Si

Fin

Algorithme 9: Énumération des sous-ensembles sûrs d'un SABP (fonction intermédiaire récursive)

Fonction $EnumPartiesSurSABP($ A : **ensemble** ,R_{att}, R_{app} : **relations binaires**) : **ensemble de sous-ensembles**

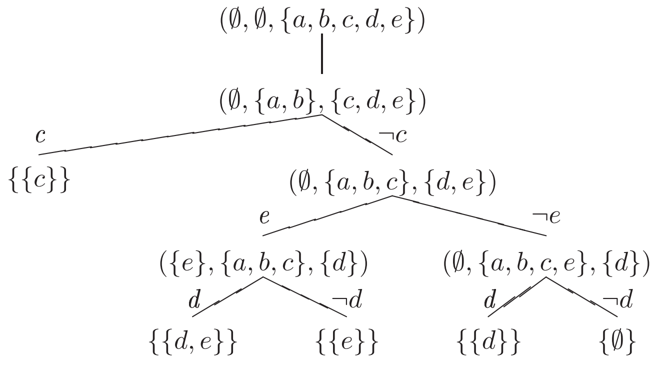
Résultat : $\{S \mid S \subseteq A \ et \ S \cap NonSsur(S) = \emptyset\}$

Retourner $(EnumPartiesSurSABP_{rec}(R_{att}, R_{app}, \emptyset, Filtre(A, R_{att}, R_{app}), A\backslash Filtre(A,$ $R_{att}, R_{app})))$;

Fin

Algorithme 10: Énumération des sous-ensembles sûrs d'un SABP

Exemple 14 page 36 – Suite *Voici l'arbre binaire d'énumération des sous-ensembles directement sûrs du système d'argumentation.*



Nous allons maintenant prouver la correction de l'algorithme $EnumPartiesSurSABP$. Cette preuve sera utilisée à la section 6.2.6 page 47 qui traite de la correction et la complétude de tous les algorithmes proposés. Pour la preuve de la correction de l'algorithme $EnumPartiesSurSABP$, nous avons besoin des définitions et des propriétés suivantes.

Propriété 9 *Étant donné un système d'argumentation bipolaire* $< A, R_{att}, R_{app} >$ *et deux arguments* $x, y \in A$, $y \in NonSur(x, R_{att}, R_{app})$ *ssi* $x \in NonSur(y, R_{att}, R_{app})$.

Preuve : Soit un système d'argumentation bipolaire $< A, R_{att}, R_{app} >$ et deux arguments $x, y \in A$ tels que $x \in NonSur(y, R_{att}, R_{app})$.
Donc $x \in (R_{att}^{+-}(y) \cup R_{app}^{-}(R_{att}^{+}(y)) \cup R_{att}^{-}(R_{app}^{+}(y)))$.
- Si $x \in R_{att}^{+-}(y)$ alors $y \in R_{att}^{+-}(x)$ et $y \in NonSur(x, R_{att}, R_{app})$,
- Si $x \in R_{app}^{-}(R_{att}^{+}(y))$ alors $y \in R_{att}^{-}(R_{app}^{+}(x))$ et $y \in NonSur(x, R_{att}, R_{app})$,
- Si $x \in R_{att}^{-}(R_{app}^{+}(y))$ alors $y \in R_{app}^{-}(R_{att}^{+}(x))$ et $y \in NonSur(x, R_{att}, R_{app})$.

$\square$

A une étape quelconque de l'algorithme, un triplet (I, O, U) est correct ssi I, O et U forment une partition de A et $NonSur(I, R_{att}, R_{app}) \subseteq O$. On appellera alors ce triplet un $\{R_{att}, R_{app}\}$-candidat.

Définition 38 ($\{R_{att}, R_{app}\}$-candidat)
Soit un système d'argumentation bipolaire $< A, R_{att}, R_{app} >$, un $\{R_{att}, R_{app}\}$-candidat *est un triplet (I, O, U) d'ensembles disjoints tels que $NonSur(I, R_{att}, R_{app}) \subseteq O$.*

Propriété 10 *Étant donné un système d'argumentation bipolaire $< A, R_{att}, R_{app} >$, un $\{R_{att}, R_{app}\}$-candidat (I, O, U) et un élément x de U tel que $x \notin Filtre(A, R_{att}, R_{app})$, les triplets suivants sont des $\{R_{att}, R_{app}\}$-candidats :*
- $(I \cup \{x\}, O \cup NonSur(x, R_{att}, R_{app}),\ U \backslash (\{x\} \cup NonSur(x, R_{att}, R_{app})))$
- $(I, O \cup \{x\}, U \backslash\ \{x\})$

Preuve : Nous utiliserons dans cette preuve les propriétés ensemblistes suivantes. A, B, C et D sont des ensembles :

$$A \cup (B \backslash A) = A \cup B$$
$$(A \backslash C) \cap (B \cup C) = (A \cap B) \backslash C$$
$$(A \backslash C) \cap (B \cup D) = (A \cap B) \backslash C \text{ si } D \subseteq C.$$

1. Prouvons que :
   $(I', O', U') = (I \cup \{x\}, O \cup NonSur(x, R_{att}, R_{app}),\ U \backslash (\{x\} \cup NonSur(x, R_{att}, R_{app})))$
   est un $\{R_{att}, R_{app}\}$-candidat.
   - Prouvons d'abord que les ensembles I', O' et U' sont disjoints :
     $I' \cap U' = (I \cup \{x\}) \cap (U \backslash (\{x\} \cup NonSur(x, R_{att}, R_{app})))$
     est un sous-ensemble inclus dans :
     $(I \cup \{x\}) \cap (U \backslash (\{x\})) = (I \cap U) \backslash \{x\} = \emptyset$
     puisque I et U sont des ensembles disjoints par hypothèse.

     $$\begin{aligned} I' \cap O' &= (I \cup \{x\}) \cap (O \cup NonSur(x, R_{att}, R_{app})) \\ &= (I \cap O) \cup (I \cap NonSur(x, R_{att}, R_{app})) \cup (\{x\} \cap O) \cup \\ &\quad\ (\{x\} \cap NonSur(x, R_{att}, R_{app})) \\ &\quad\ x \notin Filtre(A, R_{att}, R_{app}) \text{ donc, par la définition de la fonction} \\ &\quad\ Filtre,\ x \notin NonSur(x, R_{att}, R_{app}), \\ &= (I \cap O) \cup (I \cap NonSur(x, R_{att}, R_{app})) \cup (\{x\} \cap O) \\ &\quad\ x \in U \text{ et } O, U \text{ sont des ensembles disjoints donc } x \notin O, \\ &= (I \cap O) \cup (I \cap NonSur(x, R_{att}, R_{app})) \\ &\quad\ I \text{ et } O \text{ sont des ensembles disjoints,} \\ &= I \cap NonSur(x, R_{att}, R_{app}) \end{aligned}$$

     Supposons que $I \cap NonSur(x, R_{att}, R_{app}) \neq \emptyset$, il existe donc un argument $y \in I$ tel que $y \in NonSur(x, R_{att}, R_{app})$. Par la propriété 9 page précédente, on a $x \in NonSur(y, R_{att}, R_{app})$. Nous savons que $NonSur(I, R_{att}, R_{app}) \subseteq O$ donc x est dans l'ensemble O. C'est impossible puisque x est dans l'ensemble U et les ensembles U et O sont disjoints. Finalement, $I \cap NonSur(x, R_{att}, R_{app}) = \emptyset$ et donc $I' \cap O' = \emptyset$.

     $$\begin{aligned} U' \cap O' &= (U \backslash (\{x\} \cup NonSur(x, R_{att}, R_{app}))) \cap \\ &\quad\ (O \cup NonSur(x, R_{att}, R_{app})) \\ &= (U \cap O) \backslash (\{x\} \cup NonSur(x, R_{att}, R_{app})) \end{aligned}$$

     est un sous-ensemble de $U \cap O = \emptyset$.
   - Prouvons maintenant que $NonSur(I', R_{att}, R_{app}) \subseteq O'$:

     $$\begin{aligned} NonSur(I', R_{att}, R_{app}) &= NonSur(I \cup \{x\}, R_{att}, R_{app}) \\ &= NonSur(I, R_{att}, R_{app}) \cup NonSur(x, R_{att}, R_{app}) \\ &\subseteq O \cup NonSur(x, R_{att}, R_{app}) \\ &= O' \end{aligned}$$

2. Prouvons que :

$$(I'', O'', U'') = (I, O \cup \{x\}, U \backslash \{x\})$$

est un $\{R_{att}, R_{app}\}$-candidat.

– Il est aisé de démontrer que (I'', O'', U'') est un triplet d'ensembles disjoints car I, O et U sont disjoints, $I'' = I$ et O'' est construit en ajoutant à O un élément x qui est enlevé de U pour obtenir U''.
– Montrons maintenant que $NonSur(I'', R_{att}, R_{app}) \subseteq O''$:
$NonSur(I'', R_{att}, R_{app}) = NonSur(I, R_{att}, R_{app}) \subseteq O \subseteq O \cup \{x\} = O''$

□

Nous avons prouvé que, pour un système d'argumentation bipolaire $< A, R_{att}, R_{app} >$, à chaque étape d'itération les deux ensembles créés, auxquels la fonction est réappliquée, sont des $\{R_{att}, R_{app}\}$-candidats. Autrement dit, les fils d'un nœud $\{R_{att}, R_{app}\}$-candidat sont aussi des $\{R_{att}, R_{app}\}$-candidats. Ainsi, puisque le triplet initial $(\emptyset, Filtre(A, R_{att}, R_{app}), A \backslash Filtre(A, R_{att}, R_{app}))$ est un $\{R_{att}, R_{app}\}$-candidat alors, par itération, toutes les feuilles de l'arbre binaire sont des $\{R_{att}, R_{app}\}$-candidats. De plus, si une feuille (I, O, U), avec $U = \emptyset$, est un $\{R_{att}, R_{app}\}$-candidat alors I est directement sûr. En effet, $NonSur(I, R_{att}, R_{app}) \subseteq O$ et les ensembles I et O sont disjoints, donc $I \cap NonSur(I, R_{att}, R_{app}) = \emptyset$, et, d'après la propriété 8 page 38, I est directement sûr. Donc l'algorithme $EnumPartiesSurSABP$ est correct.

6.2.3 Algorithme générique d'énumération des sous-ensembles de la sémantique désirée

Voici l'algorithme générique d'énumération des sous-ensembles respectant la sémantique recherchée, les fonctions $CondVrai$, $CondFaux$, $Incoherent$ et $Filtre$ seront détaillées pour chaque sémantique. Cet algorithme générique effectue un test sur chaque feuille, dont l'ensemble I respecte la cohérence désirée, c'est-à-dire la cohérence de *sans conflit-direct* ou de *directement sûr*. Il servira alors pour l'énumération des sous-ensembles faiblement d-admissibles, des sous-ensembles faiblement s-admissibles, des extensions faiblement d-préférées et des extensions faiblement s-préférées.

$EnumGenericSABP$ est la fonction principale. Elle fait appel à la fonction $EnumGeneric$-$SABP_{rec}$ qui retourne l'ensemble des sous-ensembles respectant la sémantique recherchée. A chaque étape, elle vérifie si un ensemble correct, au sens de la sémantique recherchée, peut être trouvé au nœud courant ou dans une feuille descendante du nœud courant :

– lorsque le nœud courant est une feuille ($U = \emptyset$), $CondVrai(R_{att}, R_{app}, I, O)$ renvoie *vrai* si l'ensemble I de la feuille courante est un ensemble solution.
– $CondFaux(R_{att}, R_{app}, I, O, U)$ renvoie *vrai* si aucun ensemble solution ne peut être trouvé à partir du nœud courant.

La fonction $Incoherent(x, R_{att}, R_{app})$ renvoie l'ensemble des éléments qui, associés à x, ne respectent pas la cohérence désirée et devront donc être éliminés si l'on décide de choisir x dans la construction d'un ensemble solution.

La fonction $Filtre$ détermine l'ensemble des éléments dont on sait qu'ils ne pourront jamais faire partie d'un ensemble solution.

Fonction $EnumGenericSABP_{rec}($ R_{att}, R_{app} : **relations binaires** ,I, O, U : **ensembles disjoints**) : **ensemble de sous-ensembles**

Résultat : $\{S \mid I \subseteq S \subseteq (I \cup U)$ et S est $cohérent\}$
Si $(U = \emptyset)$ **Alors**
  Si $(CondVrai(R_{att}, R_{app}, I, O))$ **Alors**
    Retourner $(\{I\})$
  Sinon
    Retourner $(\{\})$
  Fin Si
Sinon
  Si $(CondFaux(R_{att}, R_{app}, I, O, U))$ **Alors**
    Retourner $(\{\})$
  Sinon
    $x \leftarrow Select(U)$;
    Retourner $(EnumGenericSABP_{rec}(R_{att}, R_{app}, I \cup \{x\}, O \cup Incoherent(x, R_{att}, R_{app}), U \setminus (\{x\} \cup Incoherent(x, R_{att}, R_{app}))) \cup (EnumGenericSABP_{rec}(R_{att}, R_{app}, I, O \cup \{x\}, U \setminus \{x\}))$;
  Fin Si
Fin Si
Fin

Algorithme 11: Énumération générique dans un SABP (fonction intermédiaire récursive)

Fonction EnumGenericSABP(A : **ensemble** ,R_{att}, R_{app} : **relations binaires**) : **ensemble de sous-ensembles**

Résultat : $Ensemble\ de\ sous-ensembles\ respectant\ la\ sémantique\ désirée$
Retourner $(EnumGenericSABP_{rec}(R_{att}, R_{app}, \emptyset, Filtre(A, R_{att}, R_{app}), A \setminus Filtre(A, R_{att}, R_{app})))$;
Fin

Algorithme 12: Énumération générique dans un SABP

6.2.4 Algorithme d'énumération des sous-ensembles faiblement admissibles

6.2.4.1 Énumération des sous-ensembles faiblement d-admissibles

L'algorithme d'énumération des sous-ensembles faiblement d-admissibles repose sur l'algorithme générique défini précédemment.

- $CondVrai(R_{att}, R_{app}, I, O)$ renvoie $vrai$ si l'ensemble I de la feuille courante est un ensemble faiblement d-admissible, c'est-à-dire un ensemble sans conflit-direct qui défend tous ses éléments. Pour que les ensembles I de toutes les feuilles de l'arbre binaire soient sans conflit-direct, il faut définir la fonction $Incoherent$:
  $Incoherent(x, R_{att}, R_{app}) = Conflit(x, R_{att})$, $Conflit$ est la fonction définie dans l'algorithme de la fonction $EnumPartiesSansconflitSABP$.
  Nous sommes maintenant assuré que toutes les feuilles possèdent un ensemble I sans conflit. Un tel ensemble est faiblement d-admissible ssi il défend tous ses éléments. Le test est effectué dans la fonction $CondVrai$:

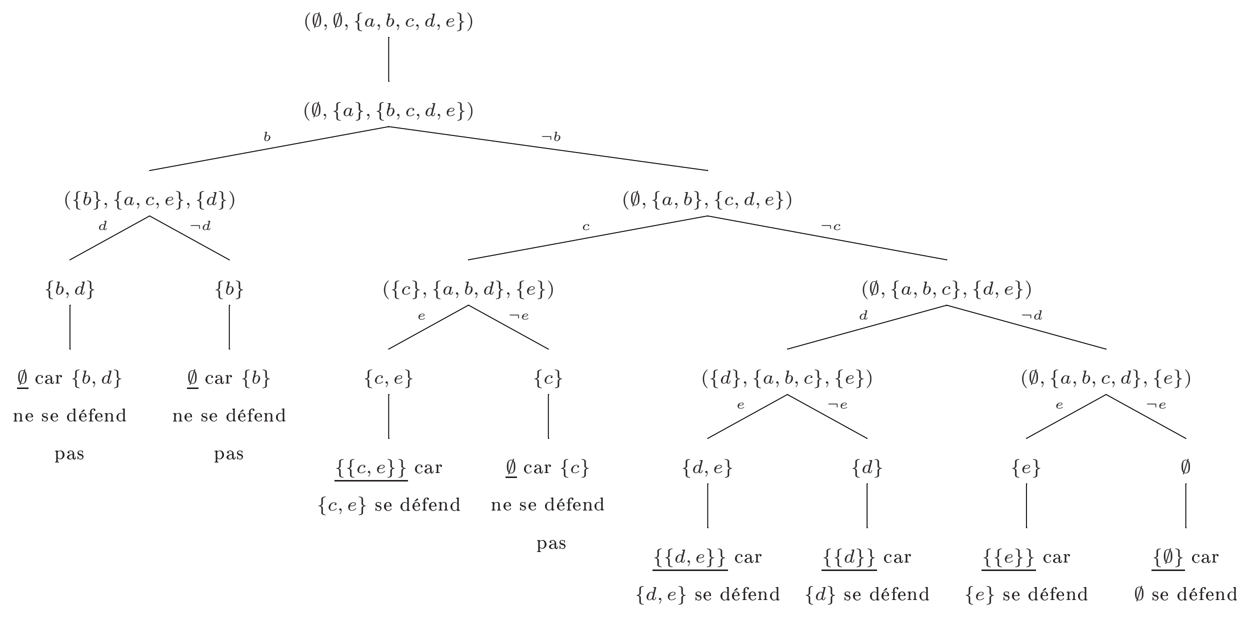
$CondVrai(R_{att}, R_{app}, I, O) = R^-_{att}(I) \subseteq R^+_{att}(I)$.
Ainsi, la fonction $CondVrai(R_{att}, R_{app}, I, O)$ renvoie *vrai* lorsque l'ensemble I est faiblement d-admissible.

– $CondFaux(R_{att}, R_{app}, I, O, U)$ renvoie *vrai* si aucun sous-ensemble faiblement d-admissible ne peut être trouvé à partir du nœud courant. Puisque la notion d'ensemble faiblement d-admissible dans un système d'argumentation bipolaire correspond à la notion d'ensemble admissible dans un système d'argumentation unipolaire, la condition sous laquelle aucun sous-ensemble faiblement d-admissible ne peut être trouvée à partir du nœud courant reste identique à celle de l'algorithme $EnumAdm$. Nous avons donc :
$CondFaux(R_{att}, R_{app}, I, O, U) = \exists x \in (R^-_{att}(I) \backslash R^+_{att}(I))$ tel que $R^-_{att}(x) \subseteq O$.

– La fonction $Filtre(A, R_{att}, R_{app}) = Refl(A, R_{att})$ puisque nous sommes certains que les éléments réflexifs, c'est-à-dire les éléments qui s'auto-attaquent, ne font partie d'aucun ensemble faiblement d-admissible. Un argument réflexif ne peut pas assurer la cohérence *sans conflit-direct* pour les ensembles qui le contiennent.

Avec ces définitions, l'algorithme correspond à celui présenté en section 6.1.3 page 34 sur l'énumération des ensembles admissibles dans un cadre unipolaire.

Exemple 14 page 36 – Suite *Voici l'arbre binaire d'énumération des sous-ensembles faiblement d-admissibles :*



Le résultat est :
$\emptyset \cup \emptyset \cup \{\{c, e\}\} \cup \emptyset \cup \{\{d, e\}\} \cup \{\{d\}\} \cup \{\{e\}\} \cup \{\emptyset\} = \{\{c, e\}, \{d, e\}, \{d\}, \{e\}, \emptyset\}$.

6.2.4.2 Énumération des sous-ensembles faiblement s-admissibles

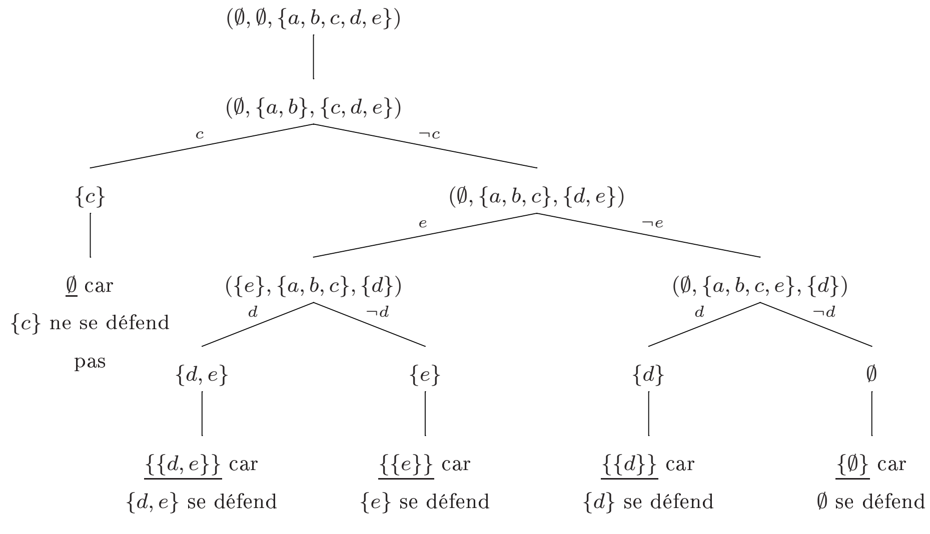
L'algorithme d'énumération des sous-ensembles faiblement s-admissibles est obtenu en définissant les fonctions internes de l'algorithme générique.

- $CondVrai(R_{att}, R_{app}, I, O)$ renvoie *vrai* si l'ensemble I de la feuille courante est un ensemble faiblement s-admissible, c'est-à-dire un ensemble directement sûr qui défend tous ses éléments. Nous posons donc :
  $Incoherent(x, R_{att}, R_{app}) = NonSur(x, R_{att}, R_{app})$, $NonSur$ est la fonction définie dans l'algorithme de la fonction $EnumPartiesSurSABP$ et,
  $CondVrai(R_{att}, R_{app}, I, O) = R^-_{att}(I) \subseteq R^+_{att}(I)$, pour les mêmes raisons que dans le cas des ensembles faiblement d-admissibles.

- $CondFaux(R_{att}, R_{app}, I, O, U)$ renvoie *vrai* si aucun sous-ensemble faiblement s-admissible ne peut être trouvé à partir du nœud courant. Les ensembles faiblement s-admissibles sont des ensembles faiblement d-admissibles, donc la condition sous laquelle aucun sous-ensemble faiblement d-admissible ne peut être trouvée à partir du nœud courant reste identique à celle de l'algorithme précédent puisque qu'elle ne repose que sur le concept de défense.

- La fonction $Filtre(A, R_{att}, R_{app})$ est identique à celle proposée dans l'algorithme de la fonction $EnumPartiesSurSABP$ puisque l'algorithme repose sur le même concept de cohérence, la fonction $Filtre$ trouve les éléments non directement sûrs.

Exemple 14 page 36 – Suite *Voici l'arbre binaire d'énumération des sous-ensembles faiblement s-admissibles :*



Le résultat est :
$\emptyset \cup \{\{d, e\}\} \cup \{\{e\}\} \cup \{\{d\}\} \cup \{\emptyset\} = \{\{d, e\}, \{e\}, \{d\}, \emptyset\}$.

6.2.5 Algorithme d'énumération des extensions de la sémantique faiblement préférée

6.2.5.1 Énumération des extensions faiblement d-préférées

Par définition, une extension faiblement d-préférée est un ensemble d-admissible maximal pour l'inclusion. Par conséquent, nous pouvons reprendre l'algorithme d'énumération des sous-ensembles

d-admissibles et étendre la condition $CondVrai$ afin qu'elle vérifie si l'ensemble passé en argument est maximal pour l'inclusion.
I est maximal pour l'inclusion ssi, pour tout sous-ensemble X non vide de O, $X \cup I$ n'est pas d-admissible, c'est-à-dire :

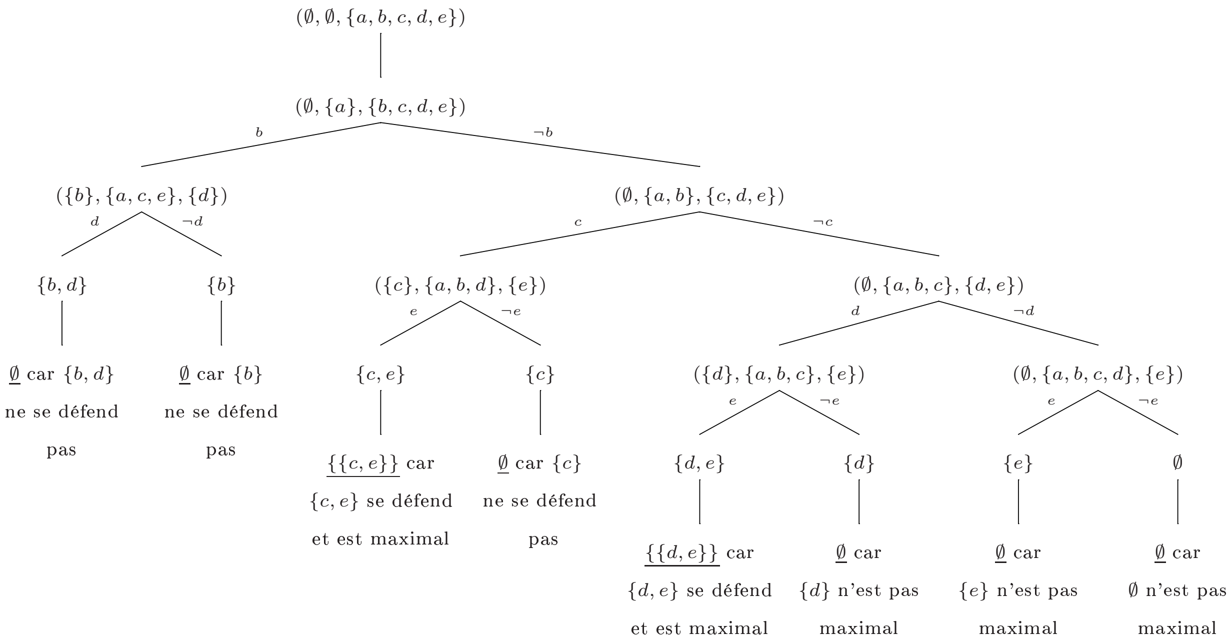
- soit $X \cup I$ n'est pas *sans conflit-direct* : $(X \cup I) \cap R^{+}_{att}(X \cup I) \neq \emptyset$,
- soit X n'est pas défendu par $X \cup I$: $R^{-}_{att}(X) \not\subseteq R^{+}_{att}(X \cup I)$.

Nous posons donc :
$Max(R_{att}, I, O) \equiv \forall X \subseteq O,\ X \neq \emptyset,\ ((X \cup I) \cap R^{+}_{att}(X \cup I) \neq \emptyset) \vee\ (R^{-}_{att}(X) \not\subseteq R^{+}_{att}(X \cup I))$,
et :
$CondVrai(R_{att}, R_{app}, I, O) = R^{-}_{att}(I) \subseteq R^{+}_{att}(I) \wedge Max(R_{att}, I, O)$.

$CondFaux(R_{att}, R_{app}, I, O, U)$ renvoie *vrai* si aucun ensemble de la sémantique faiblement d-préférée ne peut être trouvé à partir du nœud courant. Puisque les extensions faiblement d-préférées sont des ensembles faiblement d-admissibles, la condition pour laquelle aucune extension ne peut être trouvée à partir du nœud courant reste identique à celle de l'algorithme d'énumération des sous-ensembles d-admissibles.

Les fonctions $Incoherent$ et $Filtre$ restent identiques à celles de l'algorithme d'énumération des sous-ensembles d-admissibles puisque l'algorithme repose sur le même concept de cohérence.

Exemple 14 page 36 – Suite *Voici l'arbre binaire d'énumération des extensions faiblement d-préférées :*



Le résultat est :
$\emptyset \cup \emptyset \cup \{\{c, e\}\} \cup \emptyset \cup \{\{d, e\}\} \cup \emptyset \cup \emptyset \cup \emptyset = \{\{c, e\}, \{d, e\}\}$.

6.2.5.2 Énumération des extensions faiblement s-préférées

De façon analogue aux extensions faiblement d-préférées, les extensions faiblement s-préférées sont des ensembles faiblement s-admissibles. Il suffit donc de prendre l'algorithme d'énumération des sous-ensembles faiblement s-admissibles et d'étendre la fonction $CondVrai$ afin qu'elle puisse prendre en compte la notion d'ensemble maximal pour l'inclusion :
I est maximal pour l'inclusion ssi, pour tout sous-ensemble X non vide de O, $X \cup I$ n'est pas s-admissible, c'est-à-dire :

1. soit $X \cup I$ n'est pas *directement sûr* :

   $$\begin{array}{lll} \text{non directement sûr} & \Leftrightarrow & (X \cup I) \cap NonSur(X \cup I) \neq \emptyset \\ & \Leftrightarrow & (X \cup I) \cap (NonSur(X) \cup NonSur(I)) \neq \emptyset \\ & \text{or} & (I \cap NonSur(I)) = \emptyset \\ & \Leftrightarrow & ((X \cup I) \cap NonSur(X)) \cup (X \cap NonSur(I)) \neq \emptyset \\ & \Leftrightarrow & ((X \cup I) \cap NonSur(X)) \neq \emptyset \text{ ou} \\ & & (X \cap NonSur(I)) \neq \emptyset \end{array}$$

   Ou, de façon équivalente, en fonction des relations R_{att} et R_{app} :
   Sachant que I est directement sûr, $(X \cup I)$ est non directement sûr ssi

   – soit $\exists b \in I$ et $\exists c \in X$ tels que $c \not\rightarrow b$, c'est-à-dire $R^{+}_{att}(X) \cap I \neq \emptyset$,
   – soit $\exists b \in X$ et $\exists c \in I$ tels que $c \not\rightarrow b$, c'est-à-dire $R^{+}_{att}(I) \cap X \neq \emptyset$,
   – soit $\exists b \in A$, $\exists c \in X$ et $\exists d \in I$ tels que $c \not\rightarrow b \leftarrow d$, c'est à dire $R^{+}_{att}(X) \cap R^{+}_{app}(I) \neq \emptyset$,
   – soit $\exists b \in A$, $\exists c \in I$ et $\exists d \in X$ tels que $c \not\rightarrow b \leftarrow d$, c'est à dire $R^{+}_{att}(I) \cap R^{+}_{app}(X) \neq \emptyset$.

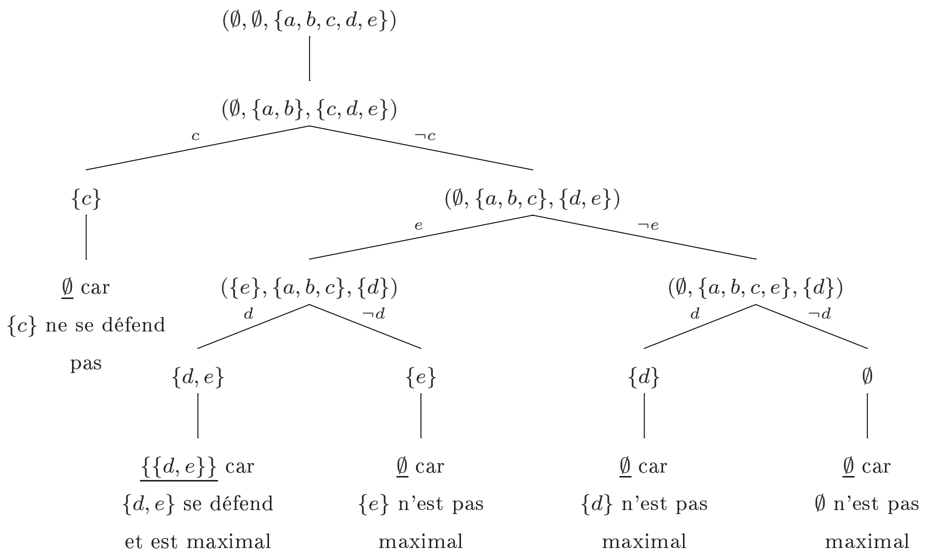
   Donc, sachant que I est directement sûr, $(X \cup I)$ est non directement sûr ssi :
   $R^{+}_{att}(X \cup I) \cap (X \cup I) \neq \emptyset$, ou
   $R^{+}_{att}(X \cup I) \cap R^{+}_{app}(X \cup I) \neq \emptyset$.

2. soit X n'est pas défendu par $X \cup I$: $R^{-}_{att}(X) \not\subseteq R^{+}_{att}(X \cup I)$.

Ainsi, nous posons :
$Max(R_{att}, R_{app}, I, O) \equiv \forall X \subseteq O,\ X \neq \emptyset,\ (((X \cup I) \cap NonSur(X)) \neq \emptyset) \vee\ ((X \cap NonSur(I)) \neq \emptyset) \vee (R^{-}_{att}(X) \not\subseteq R^{+}_{att}(X \cup I))$, ou
$Max(R_{att}, R_{app}, I, O) \equiv \forall X \subseteq O,\ X \neq \emptyset,\ (R^{+}_{att}(X \cup I) \cap (X \cup I) \neq \emptyset) \vee (R^{+}_{att}(X \cup I) \cap R^{+}_{app}(X \cup I) \neq \emptyset) \vee (R^{-}_{att}(X) \not\subseteq R^{+}_{att}(X \cup I))$.
Nous avons donc :
$CondVrai(R_{att}, R_{app}, I, O) = R^{-}_{att}(I) \subseteq R^{+}_{att}(I)\ \wedge\ Max(R_{att}, R_{app}, I, O)$.

La fonction $CondFaux$ ne change pas des algorithmes précédents puisqu'elle ne dépend que du concept de défense. Les fonctions $Incoherent$ et $Filtre$ restent identiques à celles de l'algorithme d'énumération des sous-ensembles faiblement s-admissibles puisqu'elles dépendent du même concept de cohérence.

Exemple 14 page 36 – Suite *Voici l'arbre binaire d'énumération des extensions faiblement s-préférées :*



Le résultat est :
$\emptyset \cup \{\{d,e\}\} \cup \emptyset \cup \emptyset \cup \emptyset = \{\{d,e\}\}$.

6.2.6 Correction et complétude des algorithmes

Les algorithmes proposés dans les sections précédentes sont corrects et complets. En effet, nous ne souhaitons pas que les algorithmes fournissent des solutions incorrectes ou qu'ils ne soient pas capables de retrouver toutes les solutions attendues. Il faut donc que les fonctions respectives des algorithmes renvoient des ensembles d'arguments corrects et que tous les ensembles d'arguments corrects soient renvoyés par ces fonctions.

6.2.6.1 Correction

Pour la correction de l'algorithme permettant l'énumération des sous-ensembles sans conflit-direct, la preuve est identique à celle effectuée par S. Doutre [Dou02] pour sa fonction *EnumParties-Sansconflit* puisque les fonctions *EnumPartiesSansconflit* et *EnumPartiesSansconflitSABP* portent sur le même algorithme. La preuve de la correction de l'algorithme permettant l'énumération des sous-ensembles directement sûrs a été faite dans la section 6.2.2 page 37.
Un sous-ensemble cohérent (sans conflit-direct ou directement sûr) appartient à l'ensemble de sortie d'une fonction d'énumération des sous-ensembles d'une sémantique s'il respecte la condition *CondVrai*. Ainsi, les algorithmes d'énumération des sous-ensembles d'une sémantique sont corrects si les fonctions *CondVrai* respectives sont correctes. Les fonctions *CondVrai* sont trivialement correctes, donc les algorithmes qui énumèrent les sous-ensembles d'une sémantique sont corrects.

6.2.6.2 Complétude

Les algorithmes des fonctions *EnumPartiesSansconflitSABP* et *EnumPartiesSurSABP* sont complets car ils reposent sur l'algorithme d'énumération des parties d'un ensemble qui est complet. Quant aux algorithmes d'énumération des sous-ensembles d'une sémantique, ils sont complets si et seulement si aucune solution ne peut être trouvée à partir d'un nœud vérifiant la condition *CondFaux*. La fonction *CondFaux* est indépendante du système d'argumentation puisqu'elle repose sur le concept de défense. Or, la preuve qu'aucune solution ne peut être trouvée lorsque la fonction *CondFaux* renvoie *vrai* a été donnée par S. Doutre [Dou02] dans le cadre de systèmes d'argumentation unipolaires et nous pouvons réutiliser cette preuve puisque notre fonction

$CondFaux$ et la fonction $StopCondFaux$ de S. Doutre [Dou02] sont identiques. Les algorithmes sont donc complets.

Chapitre 7

Conclusion et perspectives

7.1 Conclusion

Ce travail a été basé sur l'extension du cadre abstrait de Dung [Dun95] proposé par C. Cayrol et M.C. Lagasquie-Schiex [CLS04]. Dans ce cadre, nous avons étudié les sémantiques classiques et avons constaté que l'apparition d'une nouvelle relation avait engendré de nouvelles notions de cohérence en particulier la notion *sûr*. En effet, la combinaison des deux relations, l'attaque et l'appui, a provoqué la création de nouvelles interactions : les interactions « complexes ou complexes+ », composées d'au moins deux relations, comme l'attaque appuyée, l'attaque détournée ou la séquence d'appui et les interactions « directes », composées d'une seule relation. Nous avons donc proposé différentes notions de cohérence basées sur les interactions, puis, comme les sémantiques d'acceptabilité sont toutes construites sur le principe *concept de cohérence + concept de défense*, nous avons défini plusieurs sémantiques d'acceptabilité reposant sur ces différentes notions de cohérence, ainsi nous trouvons les sémantiques « fortes » associées aux interactions « complexes+ », les sémantiques « moyennes » associées aux interactions « complexes » et les sémantiques « faibles » associées aux interactions « directes ». Le concept de défense étant indépendant de la relation d'appui, nous avons alors utilisé les notions déjà définies dans les cadres unipolaires (ex : préférée, stable).
Puis nous avons testé la validité des notions ainsi définies dans une instanciation du cadre abstrait, où la structure des arguments apparaît concrètement, et avons remarqué que les interprétations différaient selon les cas considérés.
Finalement, nous avons proposé des algorithmes sans heuristiques permettant l'énumération d'ensembles respectant les quatre sémantiques suivantes : d-admissible, s-admissible, d-préférée et s-préférée et donné, pour chaque algorithme, les preuves de sa correction et de sa complétude.

7.2 Perspectives de recherche

7.2.1 Particularité des coalitions

Nous avons vu qu'une coalition était la faculté des arguments qui s'appuient à se rassembler. Cet aspect pourrait être utilisé pour une éventuelle conversion d'un système d'argumentation bipolaire en un système d'argumentation au sens de Dung [Dun95]. En effet, une coalition peut être vue comme un ensemble d'arguments indépendant pour la relation R_{app} et maximal pour l'inclusion, car l'exploitation de la notion d'indépendance pour la relation R_{app} « rassemble » tous les arguments qui s'appuient. On peut ainsi transformer un SABP en système d'argumentation avec uniquement des attaques entre ensembles d'arguments.

Propriété 11
Soit $S \subseteq A$ un ensemble indépendant pour la relation R_{app} et maximal pour l'inclusion. Si S attaque un argument a alors S attaque directement tout ensemble indépendant pour la relation R_{app} et maximal pour l'inclusion contenant a.

Preuve :
1er cas : S attaque directement l'argument a. S attaque alors tout ensemble contenant a (en particulier ceux indépendants pour la relation R_{app} et maximal pour l'inclusion),
2e cas : l'attaque de S sur a est une attaque appuyée. Soit $b = b_1 \rightarrow b_2 \rightarrow \ldots \rightarrow b_n \not\rightarrow a$ cette attaque avec $b_1 \in S$. Or b_1 appuie b_n donc $b_n \in S$, puisque S est indépendant pour la relation R_{app}. Finalement, S attaque directement a, donc S attaque directement tout ensemble contenant a (en particulier ceux indépendants pour la relation R_{app} et maximal pour l'inclusion),
3e cas : l'attaque de S sur a est une attaque détournée. Soit $b \not\rightarrow a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_n$ cette attaque avec $b \in S$ et $a_n = a$. Or a_1 appuie a_n donc a_1 appartient à tout ensemble indépendant pour la relation R_{app} et maximal pour l'inclusion contenant $a_n = a$. Donc, comme S attaque directement a_1, S attaque directement tout ensemble indépendant pour la relation R_{app} et maximal pour l'inclusion contenant a. $\square$

Un système d'argumentation bipolaire peut être vu comme un système composé uniquement d'*attaques directes* entre ensembles indépendants pour la relation R_{app} et maximal pour l'inclusion, comme on peut le constater sur la figure 7.1. C'est une conséquence de la propriété précédente. On se retrouve alors dans la situation étudiée par Dung [Dun95].

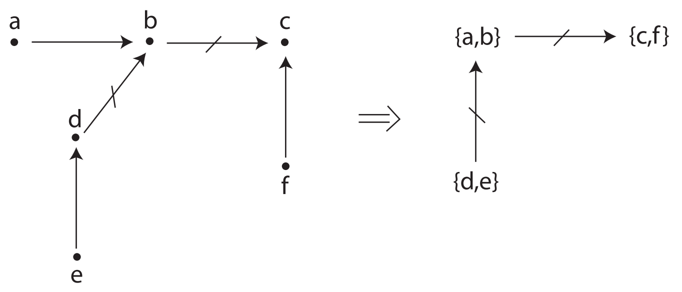


Fig. 7.1 – Transformation d'un SABP en système d'argumentation au sens de Dung [Dun95]

7.2.2 Algorithmique

Les algorithmes que nous proposons ne possèdent pas d'heuristiques, en inclure permettrait de rendre le calcul plus efficace, comme par exemple améliorer la fonction $Select$ de chaque algorithme ou améliorer la fonction $CondFaux$ pour que la fonction générale détecte le plus rapidement possible les nœuds inutiles.
La partie algorithmique ne concernait que les sémantiques d-admissibles, s-admissibles, d-préférées et s-préférées. Il reste encore à réaliser celle concernant les notions d'ensemble clos et indépendant pour la relation R_{app}.

7.2.3 Un système d'argumentation hybride

Nous avons vu que les notions « fortes » (c'est-à-dire les sémantiques fortement admissibles, préférées, etc...) étaient utilisées lorsque l'interprétation d'un système d'argumentation prêtait à une interprétation de coalitions, et les notions « faibles » (c'est-à-dire les sémantiques faiblement admissibles, préférées, etc...) dans le cas contraire. Mais ces notions ne sont pas très flexibles car elles interprètent tous les appuis de la même façon : soit tous les appuis sont des appuis de coalition (des appuis inconditionnels), soit tous les appuis sont des appuis simple (des appuis modérés). Or, nous pourrions avoir des systèmes d'argumentation qui possèdent des appuis de coalition et des appuis simples. L'application de ces notions sur un tel système d'argumentation ne donnerait pas des résultats satisfaisants. Il faudrait alors fusionner les sémantiques « faibles » et « fortes » en définissant des sémantiques qui prennent en compte les deux sortes de relation d'appui.

Bibliographie

[CLS04] Cayrol (Claudette) et Lagasquie-Schiex (Marie-Christine). – *Bipolarité en argumentation.* – Rapport de recherche n 2004-07-R, France, Institut de Recherche en Informatique de Toulouse (I.R.I.T.), février 2004.

[CLS05] Cayrol (Claudette) et Lagasquie-Schiex (Marie-Christine). – On the acceptability of arguments in bipolar argumentation frameworks. *In : Proc. of the eighth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU) – LNAI 3571*, éd. par Godo (Luis). pp. 378–389. – Barcelone, Spain, 2005.

[Dou02] Doutre (Sylvie). – *Autour de la sémantique préférée des systèmes d'argumentation.* – Thèse, Université Paul Sabatier, IRIT, 2002.

[DS04] Delgrande (J.) et Schaub (T.) (édité par). – *Proceedings of the 10^{th} NMR workshop (Non Monotonic Reasoning), Uncertainty Framework subworkshop.* – Whistler, BC, Canada, 2004.

[Dun95] Dung (Phan Minh). – On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, vol. 77, 1995, pp. 321–357.

[KP01] Karacapilidis (Nikos) et Papadias (Dimitris). – Computer supported argumentation and collaborative decision making : the HERMES system. *Information systems*, vol. 26, n 4, 2001, pp. 259–277.

[Ver02] Verheij (Bart). – On the existence and multiplicity of extension in dialectical argumentation. *In : Proceedings of the 9th International Workshop on Non-Monotonic Reasoning (NMR'2002)*, éd. par Benferhat (Salem) et Giunchiglia (Enrico), pp. 416–425.

Annexe A

Implémentation des algorithmes d'énumération

Voici l'implémentation des algorithmes d'énumération des ensembles respectant les sémantiques faiblement d-admissible, s-admissible, d-préférée et s-préférée.

A.1 Opérations ensemblistes

Le fichier source suivant, appelé `operations.ml`, contient les fonctions représentant les opérations ensemblistes. On y trouve celles permettant l'union, l'intersection, la concaténation, ...

```
(********************************************************************************)
(* Ce fichier contient les fonctions utilisees pour les operations ensemblistes *)
(********************************************************************************)



(*                                                                              *)
(* La fonction estDans prend un element et une liste en argument, elle renvoie *)
(* true si l'element est dans la liste, false sinon.                           *)
(*     val estDans : 'a -> 'a list -> bool = <fun>                              *)

let rec estDans element liste=match (element,liste) with
_,[]->false
|a,e::reste->if (compare a e)==0 then true
                                else (estDans a reste);;



(*                                                                              *)
(* La fonction enleve prend un element et une liste en argument, elle renvoie  *)
(* la liste dans laquelle toutes les occurrences de element ont ete supprime.  *)
(*     val enleve : 'a -> 'a list -> 'a list = <fun>                            *)

let rec enleve element liste=match (element,liste) with
_,[]->[]
|e,a::reste->if (compare e a)==0 then enleve e reste
                                else a::(enleve e reste);;



(*                                                                              *)
```

```
(* La fonction insere prend un element et une liste de listes en argument,     *)
(* elle insere element en tete de chaque liste de la liste des listes.         *)
(*     val insere : 'a -> 'a list list -> 'a list list = <fun>                 *)

let rec insere element= function
[]->[]
|liste::reste->(element::liste)::(insere element reste);;



(*                                                                             *)
(* La fonction intersection prend deux listes en argument, elle renvoie la     *)
(* liste resultante de l'intersection des deux listes donnees en argument.     *)
(*     val intersection : 'a list -> 'a list -> 'a list = <fun>                *)

let rec intersection liste1 liste2=if (List.length liste1)>(List.length liste2)
                                   then intersection liste2 liste1
                                   else match (liste1,liste2) with
              [],_->[]
              |e::reste, liste2->if estDans e liste2
                                 then e::(intersection reste liste2)
                                 else intersection reste liste2;;



(*                                                                             *)
(* La fonction inclusion  prend deux listes en argument, elle renvoie true si  *)
(* la premiere liste est incluse dans la deuxieme et                           *)
(* false sinon.                                                                *)
(*     val inclusion : 'a list -> 'a list -> bool = <fun>                      *)

let rec inclusion liste1 liste2=match (liste1,liste2) with
[],_->true
|a::reste,list->if (estDans a list) then (inclusion reste list)
                                    else false;;




(*                                                                             *)
(* La fonction concat  prend deux listes en argument, elle renvoie la          *)
(* concatenation des deux listes, la fonction concat3 prend trois listes en    *)
(* argument et renvoie la concatenetion des trois listes.                      *)
(*     val concat : 'a list -> 'a list -> 'a list = <fun>                      *)

let rec concat liste1 liste2=match (liste1,liste2) with
[],liste2-> liste2
|e::reste,list2->if (estDans e list2) then (concat reste list2)
                                      else (concat reste (e::liste2));;

let concat3 liste1 liste2 liste3=concat liste1 (concat liste2 liste3);;


(*                                                                             *)
(* La fonction prive prend deux listes en argument, elle supprime les elements *)
(* appartenant a la deuxieme liste dans la premiere.                           *)
(*     val prive : 'a list -> 'a list -> 'a list = <fun>                       *)

let rec prive liste1 liste2=match (liste1,liste2) with
```

```
liste1,[]-> liste1
|liste1,e::reste->prive (enleve e liste1) reste;;
```

A.2 Propriété d'un SABP

Le fichier source suivant, appelé `proprietes.ml`, contient les fonctions représentant les propriétés d'un système d'argumentation bipolaire. On y trouve les fonctions permettant de trouver les prédécesseurs et les successeurs selon une relation, les fonctions *Refl* et *NonSur*, ...

```
open Operations;;
(****************************************************************************)
(* Ce fichier contient les fonctions representant les proprietes d'un systeme  *)
(* d'argumentation bipolaire                                                   *)
(****************************************************************************)

(*                                                                           *)
(* La fonction success prend un element et une relation comme argument, elle   *)
(* renvoie tous les successeurs de l'element selon la relation donnee en       *)
(* argument.                                                                  *)
(*     val success : 'a -> ('a * 'b) list -> 'b list = <fun>                  *)

let rec success element relation=match (element,relation) with
e,[]->[]
|e,rel::reste->if (compare (fst rel) e)==0 then (snd rel)::(success e reste)
                                           else success e reste;;




(*                                                                           *)
(* La fonction predec prend un element et une relation comme argument, elle    *)
(* renvoie tous les predecesseurs de l'element selon la relation donnee en     *)
(* argument.                                                                  *)
(*     val predec : 'a -> ('a * 'b) list -> 'b list = <fun>                   *)

let rec predec element relation=match (element,relation) with
e,[]->[]
|e,rel::reste->if (compare (snd rel) e)==0 then (fst rel)::(predec e reste)
                                           else predec e reste;;



(*                                                                           *)
(* La fonction successeurs prend une liste d'elements et une relation comme    *)
(* argument, elle renvoie la liste de tous les successeurs de chaque element   *)
(* de la liste selon la relation donnee en argument.                           *)
(*     val successeurs : 'a list -> ('a * 'b) list -> 'b list = <fun>         *)

let rec successeurs liste relation=match (liste,relation) with
[],rel->[]
|a::reste,rel->(success a rel)@(successeurs reste rel);;




(*                                                                           *)
(* La fonction predecesseurs prend une liste d'elements et une relation comme  *)
(* argument, elle renvoie la liste de tous les predecesseurs de chaque element *)
(* de la liste selon la relation donnee en argument.                           *)
(*     val predecesseurs : 'a list -> ('b * 'a) list -> 'b list = <fun>       *)
```

```
let rec predecesseurs liste relation=match (liste,relation) with
[],rel->[]
|a::reste,rel->(predec a rel)@(predecesseurs reste rel);;



(*                                                                        *)
(* La fonction sucpred prend un element et une relation comme argument, elle   *)
(* renvoie la liste de tous les predecesseurs et successeurs de l'element      *)
(* donne en argument selon la relation relation donnee en argument.           *)
(*     val sucpred : 'a list -> ('a * 'a) list -> 'a list = <fun>             *)

let sucpred list rel=(successeurs list rel)@(predecesseurs list rel);;



(*                                                                        *)
(* La fonction sansconflit prend une liste d'elements et une relation comme   *)
(* argument, elle renvoie true si la liste d'elements est sans-conflit, elle  *)
(* renvoie false sinon.                                                       *)
(*     val sansconflit : 'a list -> ('a * 'a) list -> bool = <fun>            *)

let rec sansconflit ensemble relationAttaque=
(intersection ensemble (successeurs ensemble relationAttaque))==[];;



(*                                                                        *)
(* La fonction tousLesEnsembles prend un ensemble d'elements et renvoie la    *)
(* liste de tous les sous-ensembles.                                          *)
(*     val tousLesEnsembles : 'a list -> 'a list list = <fun>                 *)
let rec tousLesEnsembles=function
[]->[[]]
|element::reste->(insere element (tousLesEnsembles reste))@
                 (tousLesEnsembles reste);;


(*                                                                        *)
(* La fonction maximum prend les ensembles i,o et les deux relations d'appui  *)
(* et d'attaque comme argument, elle renvoie true si i est maximal pour       *)
(* l'inclusion en fonction des autres arguments, elle renvoie false sinon.    *)
(*    val estMax :  'a -> 'b -> 'c -> 'd list ->                              *)
(*                  ('a -> 'b -> 'c -> 'd -> bool) -> bool = <fun>            *)
let rec estMax rAtt rApp i tousLesensemblesDeO condMax=
match (tousLesensemblesDeO, i) with
        [],_->true
        |ens::reste,i->if (condMax rAtt rApp i ens)
                        then (estMax rAtt rApp i reste condMax)
                        else false;;

(* Fonction principale.                                                   *)
(*     val maximum :  'a -> 'b -> 'c -> 'd list ->                            *)
(*                    ('a -> 'b -> 'c -> 'd list -> bool) -> bool = <fun>     *)
let maximum rAtt rApp i o condMax=let ens_o=tousLesEnsembles o in
                             estMax rAtt rApp i ens_o condMax;;
```

```
(*                                                                          *)
(* Fonction qui détermine tous les elements reflexifs, selon la relation rAtt, *)
(* de a.                                                                    *)
(*     val refl : 'a list -> ('a * 'a) list -> 'a list = <fun>              *)

let rec refl a rAtt=match a with
[]->[]
|element::reste->if (inclusion [element] (successeurs [element] rAtt))
                   then element::(refl reste rAtt)
                   else (refl reste rAtt);;


(*                                                                          *)
(* Fonction qui détermine tous les elements de a qui ataquent et appuient un   *)
(* meme argument.                                                           *)
(*     val nonSur : 'a list -> ('a * 'b) list ->                            *)
(*                  ('a * 'b) list -> 'a list = <fun>                       *)

let rec nonSur a rAtt rApp=match a with
[]->[]
|element::reste->if (intersection (successeurs [element] rAtt)
                                  (successeurs [element] rApp) !=[])
                   then element::(nonSur reste rAtt rApp)
                   else (nonSur reste rAtt rApp);;
```

A.3 Fonctions principales

Le fichier source suivant, appelé `enumeration.ml`, contient les fonctions décrites dans les sections précédentes : *Max*, *CondVrai*, *CondFaux*, *Incoherent*, ...

```
open Operations;;
open Proprietes;;
(****************************************************************************)
(* Ce fichier contient toutes les fonctions decrites dans le rapport.         *)
(****************************************************************************)



(*                                                                          *)
(* Fonction select de l'algorithme.                                         *)
(*                                                                          *)

let selection list=List.hd list;;


(*                                                                          *)
(* Fonctions Max de l'algorithme pour les cohérences de sans-conflit direct et *)
(* de directement sur, respectivement.                                      *)
(*                                                                          *)

let condMaxD rAtt rApp i ens=let iEns=concat i ens in
                                  let ensAttaque=successeurs ens rAtt
   (* ensemble des elements attaques directement par ens. *)
                                  and attaqueEns=predecesseurs ens rAtt
   (* ensemble des element attaquant directement ens. *)
                                  and iEnsAttaque=successeurs iEns rAtt
   (* ensemble des elements attaques directement par l'union des ensembles i   *)
```

```
   (* et ens.                                                                    *)
                                   and iEnsAppuie=successeurs iEns rApp in
   (* ensemble des elements appuyes directement par l'union des ensembles i et *)
   (* ens.                                                                       *)
((ens==[])or
 (* ens est vide. *)
 (intersection iEns iEnsAttaque!=[])or
 (* ens n'est pas sans-conflit direct. *)
 (not (inclusion attaqueEns iEnsAttaque))
 (* l'union des ensembles i et ens ne defend pas ens. *)
);;


let condMaxS rAtt rApp i ens=let iEns=concat i ens in
                                   let ensAttaque=successeurs ens rAtt
                                   and attaqueEns=predecesseurs ens rAtt
                                   and iEnsAttaque=successeurs iEns rAtt
                                   and iEnsAppuie=successeurs iEns rApp in
((ens==[])or
 (intersection iEns iEnsAttaque!=[])or
 (not (inclusion attaqueEns iEnsAttaque))or
 (intersection iEnsAttaque iEnsAppuie!=[])
 (* l'union des ensembles i, u et ens n'est pas directement sur. *)
);;



(*                                                                               *)
(* Fonctions CondVrai de l'algorithme pour les ensembles faiblement              *)
(* d-admissibles, s-admissibles, d-preferes et s-preferes respectivement.        *)
(*                                                                               *)

let condVraiDAdm rAtt rApp i o=( (inclusion (predecesseurs i rAtt)
                                            (successeurs i rAtt))
                               );;

let condVraiSAdm rAtt rApp i o=( (inclusion (predecesseurs i rAtt)
                                            (successeurs i rAtt))&&
                                 (intersection (successeurs i rApp)
                                               (successeurs i rAtt)==[])
                               );;

let condVraiDExt rAtt rApp i o=( (inclusion (predecesseurs i rAtt)
                                            (successeurs i rAtt))&&
                                 (maximum rAtt rApp i o condMaxD)
                               );;

let condVraiSExt rAtt rApp i o=( (inclusion (predecesseurs i rAtt)
                                            (successeurs i rAtt))&&
                                 (intersection (successeurs i rApp)
                                               (successeurs i rAtt)==[])&&
                                 (maximum rAtt rApp i o condMaxS)
                               );;



(*                                                                               *)
(* Fonction CondFaux de l'algorithme, elle est identique pour toutes les         *)
```

```
(* semantiques.                                                            *)
(*                                                                         *)

let condFaux rAtt rApp i o u= let attaquant=prive (predecesseurs i rAtt)
                                                  (successeurs i rAtt) in
if (attaquant==[]) then false
                   else inclusion (predecesseurs attaquant rAtt)
                                   o;;



(*                                                                         *)
(* Fonctions Filtre de l'algorithme pour l'enumeration des ensembles       *)
(* sans-conflit direct et directement surs respectivement.                 *)
(*                                                                         *)

let filtreD a rAtt rApp=refl a rAtt;;

let filtreS a rAtt rApp=(refl a rAtt)@(nonSur a rAtt rApp);;



(*                                                                         *)
(* Fonctions Incoherent de l'algorithme pour l'enumeration des ensembles   *)
(* sans-conflit direct et directement surs respectivement.                 *)
(*                                                                         *)

let incoherentD x rAtt rApp=sucpred [x] rAtt;;

let incoherentS x rAtt rApp=concat3 (sucpred [x] rAtt)
                                    (predecesseurs (successeurs [x] rAtt) rApp)
                                    (predecesseurs (successeurs [x] rApp) rAtt);;




(*                                                                         *)
(* Fonctions enumGenericSABPRec de l'algorithme.                           *)
(*                                                                         *)

let rec enumGenericSABPRec rAtt rApp i o u condVrai condFaux incoherent=
if (u==[]) then if (condVrai rAtt rApp i o)
                then [i]
                else []
           else if (condFaux rAtt rApp i o u)
                then []
                else let x=selection u in
                         let ensInco=incoherent x rAtt rApp in
(enumGenericSABPRec rAtt rApp (x::i)
                              (concat o ensInco)
                              (prive u (x::ensInco))
                              condVrai condFaux incoherent) @
(enumGenericSABPRec  rAtt rApp i
                              (x::o)
                              (enleve x u)
                              condVrai condFaux incoherent);;
```

```
(*                                                                        *)
(* Fonctions enumGenericSABP de l'algorithme.                             *)
(*                                                                        *)

let enumGenericSABP a rAtt rApp condVrai condFaux filtre incoherent=
let ens=filtre a rAtt rApp in
       enumGenericSABPRec rAtt rApp []
                                  ens
                                  (prive a ens)
                                  condVrai condFaux incoherent;;
```