

---

## Représentation des connaissances et raisonnement

### INTRODUCTION À L'ARGUMENTATION – PROJET

---

## 1 Contexte

Un **système d'argumentation (AS)** est défini par  $F = \langle \mathcal{A}, \mathcal{R} \rangle$  avec

- $\mathcal{A}$  un ensemble d'arguments abstraits
- $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$  l'ensemble des attaques entre ces arguments

Par exemple, la Figure 1 est la représentation graphique du système  $F = \langle \mathcal{A}, \mathcal{R} \rangle$  avec  $\mathcal{A} = \{a, b, c, d\}$  et  $\mathcal{R} = \{(a, b), (b, c), (b, d)\}$ .

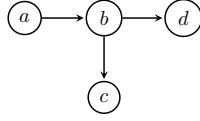


FIGURE 1 – Un exemple simple d'un système d'argumentation

On rappelle également que, étant donné  $F = \langle \mathcal{A}, \mathcal{R} \rangle$  un système d'argumentation,

- $S \subseteq \mathcal{A}$  est **sans conflit** ssi  $\nexists a, b \in S$  tels que  $aRb$
- $S$  est un **ensemble admissible** ssi  $S$  est sans conflit et  $S$  défend tous ses arguments
- $S$  est une **extension préférée** ssi  $S$  est maximal pour l'inclusion parmi les extensions admissibles
- $S$  est une **extension stable** ssi  $S$  est sans conflit et  $S$  attaque tout argument de  $\mathcal{A}$  qui n'appartient pas à  $S$

## 2 Instructions

L'objectif de ce projet est de développer un outil permettant de résoudre les problèmes suivants :

- **VE-PR** : Étant donné un AF  $F = \langle \mathcal{A}, \mathcal{R} \rangle$  et un ensemble d'arguments  $S \subseteq \mathcal{A}$ , déterminer si  $S$  est une **extension préférée** de  $F$ .
- **DC-PR** : Étant donné un AF  $F = \langle \mathcal{A}, \mathcal{R} \rangle$  et un argument  $a \in \mathcal{A}$ , déterminer si  $a$  appartient à une **extension préférée** de  $F$ .
- **DS-PR** : Étant donné un AF  $F = \langle \mathcal{A}, \mathcal{R} \rangle$  et un argument  $a \in \mathcal{A}$ , déterminer si  $a$  appartient à **toutes les extensions préférées** de  $F$ .
- **VE-ST** : Étant donné un AF  $F = \langle \mathcal{A}, \mathcal{R} \rangle$  et un ensemble d'arguments  $S \subseteq \mathcal{A}$ , déterminer si  $S$  est une **extension stable** de  $F$ .
- **DC-ST** : Étant donné un AF  $F = \langle \mathcal{A}, \mathcal{R} \rangle$  et un argument  $a \in \mathcal{A}$ , déterminer si  $a$  appartient à une **extension stable** de  $F$ .
- **DS-ST** : Étant donné un AF  $F = \langle \mathcal{A}, \mathcal{R} \rangle$  et un argument  $a \in \mathcal{A}$ , déterminer si  $a$  appartient à **toutes les extensions stables** de  $F$ .

VE- $\sigma$  signifie “**V**erify **E**xtension dans le cadre de la sémantique  $\sigma$ ”, DC- $\sigma$  signifie “**D**ecide the **C**redulous **a**ccceptability de l’argument dans le cadre de la sémantique  $\sigma$ ”, et DS- $\sigma$  signifie “**D**ecide the **S**keptical **a**ccceptability de l’argument dans le cadre de la sémantique  $\sigma$ ”.

L’AF sera lu à partir d’un fichier `.apx` en utilisant le format suivant. Chaque argument est défini sur une ligne de la forme :

```
arg(name_argument).
```

et chaque attaque est définie sur une ligne de la forme :

```
att(name_argument_1,name_argument_2).
```

Pour chaque ligne, nous supposons qu’il n’y a pas d’espace blanc. Tous les arguments doivent être définis avant d’être utilisés dans une ligne d’attaque. Enfin, le nom d’un argument peut être n’importe quelle séquence de lettres (majuscules ou minuscules), de chiffres ou le symbole de soulignement `_`, à l’exception des mots `arg` et `att` qui sont réservés à la définition des lignes.

Un exemple de fichier complet, correspondant à l’AF de la Figure 1, est donné ici :

```
arg(a).
arg(b).
arg(c).
arg(d).
att(a,b).
att(b,c).
att(b,d).
```

Votre programme doit être implémenté dans l’un des langages suivants : C, C++, Java, Python.

Dans ce qui suit, nous supposons que `COMMAND` est soit :

- `./executable_file` si vous avez utilisé C ou C++,
- `java -jar runnable_jar_file.jar` si vous avez utilisé Java,
- `python program.py` si vous avez utilisé Python.

Le programme sera exécuté avec l’interface de ligne de commande suivante :

- pour VE-PR et VE-ST

```
COMMAND -p VE-XX -f FILE -a ARG1,ARG2,...,ARGn
```

où `XX` est soit PR, soit ST ; `FILE` est le chemin d’accès au fichier `.apx` décrivant l’AF ; `ARG1`, ..., `ARGn` sont les noms des arguments de l’ensemble  $S$  de la requête ;

- pour DC-PR, DS-PR, DC-ST et DS-ST :

```
COMMAND -p XX-YY -f FILE -a ARG
```

où `XX` est soit DC, soit DS ; `YY` est soit PR, soit ST ; `FILE` est le chemin d’accès au fichier `.apx` décrivant l’AF ; `ARG` est le nom de l’argument de la requête.

Le programme affiche alors sur la sortie standard le résultat suivant :

- pour VE-PR and VE-ST, il affiche YES si l'ensemble d'arguments donnés dans la requête est une extension, ou NO dans le cas contraire. ;
- pour DC-PR, DS-PR, DC-ST et DS-ST, il affiche YES si l'argument de la requête est accepté, ou NO dans le cas contraire.

Par exemple, supposons que le fichier `af.txt` décrive l'AF de la Figure 1. Les lignes de commande suivantes doivent afficher le résultat correspondant :

```
~$ ./my_solver -p VE-PR -f af.txt -a a,c,d
YES
~$ ./my_solver -p VE-PR -f af.txt -a a
NO
~$ ./my_solver -p DS-PR -f af.txt -a a
YES
~$ ./my_solver -p DC-PR -f af.txt -a b
NO
```

### 3 Livraison du projet

Ce projet doit être réalisé par **groupe de deux étudiants**.

Votre code source, **correctement documenté**, sera déposé sur Moodle avant le **31/12/2025** (23h59, heure de Paris), sous la forme d'un **fichier zip** nommé avec les noms des étudiants, par exemple `Toto_Titi.zip` si le projet a été développé par les étudiants Toto et Titi.

Ce fichier zip doit contenir :

- Un répertoire avec le **code source**.
- Un **makefile** ou un **script shell** pour produire un exécutable (dans le cas de C ou C++) ou un fichier jar (dans le cas de Java). (Bien entendu, un tel makefile ou script shell n'est pas nécessaire si le programme est un script Python)
- Un fichier **README** contenant les instructions nécessaires pour exécuter votre programme.
- un **rapport en pdf** (nommé `Toto_Titi.pdf`). Ce fichier doit indiquer les noms de chaque membre du binôme de projet, et doit expliquer
  - ◊ tous les algorithmes implémentés
  - ◊ les principales structures de données
  - ◊ les tests réalisés et leur résultats

Vous êtes libre d'utiliser des algorithmes trouvés dans la littérature sur l'argumentation. Dans ce cas vous devez expliquer l'algorithme avec vos propres mots, et citer correctement le travail qui a proposé cet algorithme à l'origine (par exemple, dans quel article scientifique cet algorithme a été introduit).