Représentation des connaissances et raisonnement
INTRODUCTION À L'ARGUMENTATION – PROJET

1 Contexte

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

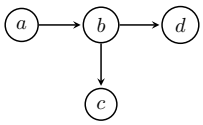

FIGURE 1 – Un exemple simple d'un système d'argumentation

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

- $S \subseteq \mathcal{A}$ est **sans conflit** ssi $\nexists a,b \in S$ tels que $a\mathcal{R}b$
- S est un **ensemble admissible** ssi S est sans conflit et S défend tous ses arguments
- S est une **extension complète** ssi S est admissible et ssi tout argument défendu par S appartient à S
- 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 :

- `SE-CO` : Étant donné un AF $\mathsf{F} = \langle \mathcal{A}, \mathcal{R} \rangle$, donner une extension complète de F.
- `DC-CO` : Étant donné un AF $\mathsf{F} = \langle \mathcal{A}, \mathcal{R} \rangle$ et un argument $a \in \mathcal{A}$, déterminer si a appartient à une extension complète de F.
- `DS-CO` : Étant donné un AF $\mathsf{F} = \langle \mathcal{A}, \mathcal{R} \rangle$ et un argument $a \in \mathcal{A}$, déterminer si a appartient à toutes les extensions complètes de F.
- `SE-ST` ; Étant donné un AF $\mathsf{F} = \langle \mathcal{A}, \mathcal{R} \rangle$, donner une extension stable de F.
- `DC-ST` ; Étant donné un AF $\mathsf{F} = \langle \mathcal{A}, \mathcal{R} \rangle$ et un argument $a \in \mathcal{A}$, déterminer si a appartient à une extension stable de F.
- `DS-ST` ; Étant donné un AF $\mathsf{F} = \langle \mathcal{A}, \mathcal{R} \rangle$ et un argument $a \in \mathcal{A}$, déterminer si a appartient à toutes les extensions stables de F.

`SE-`σ signifie "donne **S***ome* **E***xtension* dans le cadre de la sémantique σ", `DC-`σ signifie "**D***ecide the* **C***redulous acceptability* de l'argument dans le cadre de la sémantique σ", et `DS-`σ signifie "**D***ecide the* **S***keptical acceptability* de l'argument dans le cadre de la sémantique σ".

L'AF sera lu à partir d'un fichier texte 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 `SE-CO` et `SE-ST` :

  ```
  COMMAND -p SE-XX -f FILE
  ```

  où `XX` est soit `CO`, soit `ST` ; `FILE` est le chemin d'accès au fichier texte décrivant l'AF ;
- pour `DC-CO`, `DS-CO`, `DC-ST` et `DS-ST` :

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

  où `XX` est soit `DC`, soit `DS` ; `YY` est soit `CO`, soit `ST` ; `FILE` est le chemin d'accès au fichier texte 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 `SE-CO` and `SE-ST`, il affiche soit une extension complète, soit une extension stable (s'il y en a une), soit `NO` s'il n'y en a pas ;
- pour `DC-CO`, `DS-CO`, `DC-ST` et `DS-ST`, il affiche `YES` si l'argument de la requête est accepté, ou `NO` dans le cas contraire.

Pour afficher une extension, le format est le suivant :

```
[arg1,arg2,...,argN]
```

pour représenter l'ensemble des arguments $\{arg_1, arg_2, \ldots, arg_n\}$. Après avoir affiché la réponse (`YES`, `NO` ou un ensemble d'arguments), le programme affiche une nouvelle ligne.

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 SE-CO -f af.txt
[a,c,d]
~$ ./my_solver -p DC-CO -f af.txt -a a
YES
~$ ./my_solver -p DS-CO -f af.txt -a a
YES
~$ ./my_solver -p DC-CO -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 **31/12/2024** (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 qui compile le code pour produire un fichier exécutable (dans le cas de C ou C++) ou un fichier jar exécutable (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 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 ainsi que les principales structures de données.

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).