Projet d'argumentation

Description du projet

On appelle "Argumentation Framework" (AF) un graphe ¡A,R¿ avec les arguments A et les attaques R. L'objectif de ce projet est de donner les différentes extensions et sémantiques des graphes fournis dans des fichiers. Nous vonlons ainsi :

- Soit F un graphe ¡A, R¿, donner une extension complète de F.
- Soit F un graphe ¡A, R¿ et un argument a, déterminer si a fait partie de toutes les extensions complètes de F. Ici, a est crédulement accepté.
- Soit F un graphe ¡A, R¿ et un argument a, déterminer si a fait partie d'une certaine extension complète de F. Ici, a est sceptiquement accepté.
- Soit F un graphe ¡A, R¿, donner une extension stable de F.
- Soit F un graphe ¡A, R¿ et un argument a, déterminer si a fait partie de toutes les extensions stables de F. Ici, a est crédulement accepté.
- Soit F un graphe ¡A, R¿ et un argument a, déterminer si a fait partie d'une certaine extension stable de F. Ici, a est sceptiquement accepté.

D'autres fonctionnalités ont été ajoutées au fur et à mesure de la réalisation de ce projet. Nous verrons dans la partie suivante comment nous avons implémenté ces différentes notions. Pour retrouver la liste des fonctionnalités, nous vons invitons à lire le README.md.

Découpage modulaire

Pour implémenter ce projet, il faut commencer par transformer les données qu'un fichier contient en graphe numérique. C'est pour cette raison que nous avons décidé de créer une classe objet **ArgF** (dans le fichier "class-af.py") qui représentera le graphe F. Cette classe objet possède deux attributs : une liste **args** pour les arguments et une liste **attacks** pour les attaques entre les arguments. Une fois un fichier lu, s'il est correctement écrit, le programme va créer une instance **ArgF** avec le bon contenu. Le programme peut ainsi lire des fichiers et comprendre si son contenu représente un grpahe F.

Ensuite, pour déterminer l'extension complète, il faut tout d'abord passer par l'extension fondée (*grounded()*). Pour déterminer cette dernière, nous avons implémenté plusieurs fonctions qui vont prendre les arguments qui ne sont pas attaqués (*get_valide(), defense(), et defense_recursive()*) et ceux qui sont défendus par ces arguments-là. Après avoir déterminer l'extension fondée, nous avons calculé toutes les combinaisons possibles des différents arguments et avec les mêmes fonctions, déterminé les ensembles qui se protègent eux-même (*completed()*).

Après avoir déterminer l'extension complète, nous allons déterminer l'extension préférée (*preferred()*), pour avoir l'extension stable (*stable()*). Pour déterminer l'extension préférée, nous partons de l'extension complète et nous enlevons tous les ensembles qui sont inclus dans les autres. Et après avoir déterminé l'extension préférée, nous partons de cette dernière et nous regardons si tous les arguments présents attaquent bien les autres.

À partir d'ici, nous savons comment déterminer les différentes extensions, il reste juste à déterminer quels sont les arguments qui sont acceptés crédulement et sceptiquement (*give_sem()*). Pour savoir si un argument est accepté, il suffit de regarder si l'argument est présent dans la liste des arguments acceptés.