SORBONNE UNIVERSITÉ

Rapport de Projet

Student name: *Durand Enzo et Kordon Djeser*

Course: *Logique et représentation des connaissances (LRC)*
Due date: *December 13th, 2021*

Sujet : Développement d'un demonstrateur pour la logique ALC en Prolog

Partie I.

- **premiere_etape** : génère les 3 listes en fonctions de la *Tbox* et la *Abox* initiale.
- **setof** : repere la forme qui nous intéresse.
  - Pour la *Tbox*, il s'agit de la forme **equiv(C,D)**.
  - Pour *Abi*, il s'agit de **inst(I,C)**.
  - Pour *Abr*, il s'agit de la forme **instR(A,B,R)**.

Partie II.

- **acquisition_prop_type1** : Ce predicat prend en charge la requête si elle est de type 1.
  - Demande à l'utilisateur de rentrer une instance.
  - Vérifie si celle-ci existe bien grâce au prédicat **testinstance**.
  - Demande a l'utilisateur d'entrer le concept.
  - Test le concept et le remplace par sa définition dans la *Tbox*.
  - Applique la négation sur le concept et le met sous forme normale négative.
  - Ajoute à l'*Abi*
- **acquisition_prop_type2** : Ce predicat va prendre en charge la requête si elle est de type 2.
  - Demande à l'utilisateur de rentrer successivement les deux concepts.
  - Test les deux concepts.
  - Remplace l'expression par sa forme normale négative sans appliquer la négation.
  - Ajoute l'ensemble dans *Abi*.
- **remplace** : remplace un concept par sa définition dans la *Tbox*.
- **testinstance** : test si une instance existe.
- **testconcept** : test si un concept existe.

Partie III.

- **tri_Abox** : prend en entrée la liste *Abi* puis la divise en 5 listes en fonction des différentes formes d'expressions. Ce prédicat s'arrete quand *Abi* est vide.
- **resolution** : resolution va tester s'il y a un clash dans la liste *Ls* grâce au prédicat **checkclash** et appelle notre premiere règle de résolution **complete_some**.
- **complete_some** :
  - Si la liste *Lie* contient une forme (A,some(R,C)). Si c'est le cas, il créer une instance grâce à **genere** et l'insere dans *Ls* via le prédicat **evolue**. Il modifie ensuite *Abr* en ajoutant un élément de la forme (A,B,R). Après cela, il appelle **affiche_evolution_Abox** pour l'affichage puis il appelle **resolution**.
  - Si *Lie* ne contient pas d'élément on appelle alors **transformation_and**.
- **transformation_and** :
  - Si la liste *Li* contient une forme (I,and(A,B)).Si c'est le cas il insere deux (I,A) et (I,B) dans la liste *ls* grâce au prédicat **evolue**. Après cela, il appelle **affiche_evolution_Abox** pour l'affichage puis il appelle **resolution**.
  - Si la liste *Li* ne contient pas d'élément, il appelle **deduction_all**.
- **deduction_all** :
  - Si la liste *Lpt* contient une forme (I,all(R,C)). Si c'est le cas il test si il existe une relation de type (I,B,R) dans *Abr*. Si c'est le cas, il insere un élément de la forme (B,C). Ensuite, il appelle **affiche_evolution_Abox** pour l'affichage puis il appelle **resolution**.
  - Si la liste *Lpt* ne contient pas d'élément, il appelle **transformation_or**.
- **transformation_or** :
  - Si la liste *Lu* contient une forme (I,or(C,D)). Si c'est le cas, il créér deux nouvelles listes *Ls* et met dans chacune d'elle un élément different ((I,C) ou (I,D)). Ensuite, il appelle **affiche_evolution_Abox** pour l'affichage puis il appelle **resolution** deux fois pour les deux branches.
  - Si la liste *Lu* ne contient pas d'élément, il va renvoyer faux car la résolution aura échouer.
- **evolue** : insere un élément mis en argument dans la liste qui convient.
- **affiche_evolution_Abox** :