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 démonstrateur pour la logique ALC en Prolog

Partie I.

- **premiere_etape** : Génère les 3 listes en fonctions de la *Tbox* et la *Abox* initiale.
  - **arguments de la fonction** : *Tbox* représente la *Tbox*. *Abi* contient la *Abox* des instances. *Abr* contient la *Abox* des relations.
- **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 prédicat prend en charge la requête si elle est de type 1.
  - **arguments de la fonction**: *Abi* représente la *Abox* des instances. *Tbox* représente la *Tbox* et *Abi1* représente la *Abox* des instances avec la proposition de type 1 en plus.
  - 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.
  - **arguments de la fonction**: *Abi* représente la *Abox* des instances, *Tbox* représente la *Tbox* et *Abi1* représente la *Abox* des instances avec la proposition de type 2 en plus.
  - 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*.
  - **arguments de la fonction**: C représente le concept a remplacer et CC le concept remplacé par sa définition dans la *Tbox*.
- **testinstance** : Test si une instance existe.
  - **arguments de la fonction**:I représente l'instance à tester
- **testconcept** : Test si un concept existe.
  - **arguments de la fonction**: C représente le concept à tester.

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.
  - **arguments de la fonction**: *Abi* est la liste des assertions de concepts de la **Abox** étendue. *Lie* est la liste des assertions du type (I,some(R,C)). *Lpt* est la liste des assertions du type (I,all(R,C)) . *Li* est la liste des assertions du type (I,and(C1,C2)). *Lu* est la liste des assertions du type (I,or(C1,C2)). *Ls* est la liste les assertions restantes, à savoir les assertions du type (I,C) ou (I,not(C)), C étant un concept atomique.
- **resolution** : Test s'il y a un clash dans la liste *Ls* grâce au prédicat **checkclash** et appelle notre premiere règle de résolution **complete_some**.
  - **arguments de la fonction**: Ensemble des listes générées par **tri_Abox** et *Abr* qui représente la liste des relations de la *Abox*.
- **checkclash** : Test si il y a clash dans la liste *Ls* et renvoie la variable qui cause le clash si il y en a un afin de l'afficher.
  - **arguments de la fonction**: Liste des assertions où on va tester la présence d'un clash.
- **complete_some** :
  - **arguments de la fonction**: Ensemble des listes générées par **tri_Abox** et *Abr* qui représente la liste des relations de la *Abox*.
  - 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** :
  - **arguments de la fonction**: Prend l'ensemble des listes générées par **tri_Abox** et *Abr* qui représente la liste des relations de la *Abox*.
  - 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** :
  - **arguments de la fonction**: Ensemble des listes générées **par tri_Abox** et *Abr* qui représente la liste des relations de la *Abox*.
  - 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** :
  - **arguments de la fonction**: Ensemble des listes générées par **par tri_Abox** et *Abr* qui représente la liste des relations de la *Abox*.
  - 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 renvoie faux car la résolution a échouer.
- **evolue** : Insere un élément mis en argument dans la liste qui convient.
  - **arguments de la fonction**: Ensemble des listes générées par **par tri_Abox** et *Abr* qui représente la liste des relations de la *Abox* et une assertion qu'il faut mettre dans la bonne liste.
- **affiche_evolution_Abox** : Affiche l'évolution des différentes listes lors de la résolution grâce à plusieurs prédicats.
  - **arguments de la fonction**:prend deux ensembles composés des 5 listes générées par **par tri_Abox** et renvoit la différence entre chacune des listes.
  - **diff_list** : Renvoie les éléments qui sont différents entre deux listes.
    - **arguments de la fonction** : Deux listes L1 et L2 qui sont les deux listes à comparer et renvoie les differences dans une liste R.
  - **print_diff** : Utilise le prédicat précédent pour récuperer les différents éléments puis appelle
    - **arguments de la fonction**: Prend deux listes L1 et L2 qui sont les deux listes à comparer.

trad_infix :

* **arguments de la fonction** : Une liste en argument qui sera affichée.
* Traduit la notation prefixe en notation infixe pour un meilleur affichage.

Batterie de tests.

- **acquisition_prop_type1** et **acquisition_prop_type2**

```
premiere_etape(Tbox,Abi,Abr),
deuxieme_etape(Abi,Ab1,Tbox),
%% erreur.     MichelAnge. personne.   ==> Reponse incorrecte.
%% MichelAnge. MichelAnge. personne.   ==> Reponse incorrecte.
%% 1.          MichelAnge. personne.   ==> true.
%% 1.          MichelAnge. erreur.     ==> Reponse incorrecte.
%% 1.          erreur.     personne.   ==> Reponse incorrecte.
%% 1.          MichelAnge. objet.      ==> true.
%% 1.          objet.      MichelAnge. ==> true.
%% 2.          sculpture. objet.       ==> true.
%% 2.          MichelAnge. personne.   ==> Reponse incorrecte.
```

- **tri_Abox**

```
tri_Abox([(michelAnge, personne), (david,some(aCree,sculpture)), (david,all(aCree,sculpture)), (david,or(aCree,sculpture)), (david,and(aCree,sculpture))],Lie_1,Lpt_1,Li_1,Lu_1,Ls_1),
write(Lie_1), nl,
write(Lpt_1), nl,
write(Li_1), nl,
write(Lu_1), nl,
write(Ls_1), nl,
%% ==> [(david,some(aCree,sculpture))]
%% ==> [(david,all(aCree,sculpture))]
%% ==> [(david,and(aCree,sculpture))]
%% ==> [(david,or(aCree,sculpture))]
%% ==> [(michelAnge,personne)]
```

- **resolution**

```
tri_Abox([(michelAnge, personne), (michelAnge, not(personne))],Lie_2,Lpt_2,Li_2,Lu_2,Ls_2),
resolution(Lie_2,Lpt_2,Li_2,Lu_2,Ls_2,Abr),
%% ==> !!!!!!!! Clash sur michelAnge !!!!!!!!
tri_Abox([(michelAnge, personne)],Lie_3,Lpt_3,Li_3,Lu_3,Ls_3),
resolution(Lie_3,Lpt_3,Li_3,Lu_3,Ls_3,Abr),
%% ==> false.
```

- **complete_some**
- **transformation_and**
- **deduction_all**
- **transformation_or**
- **evolue**

```
tri_Abox([(david,and(aCree,sculpture))],Lie,Lpt,Li,Lu,Ls),
write(Lie), nl,
write(Lpt), nl,
write(Li), nl,
write(Lu), nl,
write(Ls), nl,
evolue((I,and(A,B)),  Lie, Lpt, Li, Lu, Ls, Lie, Lpt, Li, Lu, Ls),
write("Evolution"), nl,
write(Lie), nl,
write(Lpt), nl,
write(Li), nl,
write(Lu), nl,
write(Ls), nl, nl, nl,
%% ==> []
%% ==> []
%% ==> [(david,and(aCree,sculpture))]
%% ==> []
%% ==> []
%% ==> []
%% ==> []
%% ==> []
%% ==> []
%% ==> [(david,and(aCree,sculpture))]
tri_Abox([(david,all(aCree,sculpture))],Lie,Lpt,Li,Lu,Ls),
write(Lie), nl,
write(Lpt), nl,
write(Li), nl,
write(Lu), nl,
write(Ls), nl,
evolue((I,all(R,C)),  Lie, Lpt, Li, Lu, Ls, Lie, Lpt, Li, Lu, Ls),
write("Evolution"), nl,
write(Lie), nl,
write(Lpt), nl,
write(Li), nl,
write(Lu), nl,
write(Ls), nl, nl, nl,
%% ==> []
%% ==> [(david,all(aCree,sculpture))]
%% ==> []
%% ==> []
%% ==> []
%% ==> []
%% ==> []
%% ==> []
%% ==> []
%% ==> [(david,all(aCree,sculpture))]
```

- **affiche_evolution_Abox**

```
tri_Abox([(michelAnge, personne), (david,some(aCree,sculpture)), (david,all(aCree,sculpture)), (david,or(aCree,sculpture)), (david,and(aCree,sculpture))],Lie1,Lpt1,Li1,Lu1,Ls1),
tri_Abox([(michelAnge, personne), (michelAnge, sculpture), (david,some(aCree,sculpture)), (david,all(aCree,sculpture)), (david,or(aCree,sculpture)), (david,and(aCree,sculpture))],Lie2,Lpt2,Li2,Lu2,Ls2),
affiche_evolution_Abox(Ls1,Lie1,Lpt1,Li1,Lu1,Abr1,Ls2,Lie2,Lpt2,Li2,Lu2,Abr2),
%% ==> --> Ls :[(michelAnge,sculpture)]
%% ==> --> Lie :
%% ==> --> Lpt :
%% ==> --> Li :
%% ==> --> Lu :
%% ==> --> Abr :
tri_Abox([(michelAnge, personne), (david,some(aCree,sculpture)), (david,all(aCree,sculpture)), (david,or(aCree,sculpture)), (david,and(aCree,sculpture))],Lie1,Lpt1,Li1,Lu1,Ls1),
tri_Abox([(michelAnge, personne), (david,some(aCree,sculpture)), (michelAnge,some(aCree,objet)), (david,all(aCree,sculpture)), (david,or(aCree,sculpture)), (david,and(aCree,sculpture))],Lie2,Lpt2,Li2,Lu2,Ls2),
affiche_evolution_Abox(Ls1,Lie1,Lpt1,Li1,Lu1,Abr1,Ls2,Lie2,Lpt2,Li2,Lu2,Abr2),
%% ==> --> Ls :
%% ==> --> Lie : michelAnge:ie(aCree.objet)
%% ==> --> Lpt :
%% ==> --> Li :
%% ==> --> Lu :
%% ==> --> Abr :
tri_Abox([(michelAnge, personne), (david,some(aCree,sculpture)), (david,all(aCree,sculpture)), (david,or(aCree,sculpture)), (david,and(aCree,sculpture))],Lie1,Lpt1,Li1,Lu1,Ls1),
tri_Abox([(michelAnge, personne), (david,some(aCree,sculpture)), (david,all(aCree,sculpture)), (michelAnge,all(aCree,objet)), (david,or(aCree,sculpture)), (david,and(aCree,sculpture))],Lie2,Lpt2,Li2,Lu2,Ls2),
affiche_evolution_Abox(Ls1,Lie1,Lpt1,Li1,Lu1,Abr1,Ls2,Lie2,Lpt2,Li2,Lu2,Abr2),
%% ==> --> Ls :
%% ==> --> Lie :
%% ==> --> Lpt : michelAnge:qq(aCree.objet)
%% ==> --> Li :
%% ==> --> Lu :
%% ==> --> Abr :
```