Sorbonne Université

Rapport Final

Décembre 2022

Projet de LRC

-

Ecriture en Prolog d'un démonstrateur basé sur l'algorithme des tableaux pour la logique de description ALC

Auteurs :
Zhe Wang
Yukai Luo

Encadrants :
Colette Faucher

Table des matières

Chapitre 1

Introduction

Dans ce projet, on va faire un démonstrateur en utilisant la language *Prolog*, basé sur l'algorithme des tableaux pour la logique ALC. Ce projet a 3 partie. Chaque partie joue un rôle précise et importante. La première partie concentre sur tester la correction syntaxique et sémantique. La deuxième partie conentre sur saisir de la proposition à démontrer. Le rôle de troisième est construire la démonstrateur.

Chapitre 2

Etape Préliminaire de Vérification et de Mise en Forme de la Tbox et de la Abox

2.1 Correction syntaxique et sémantique(—concept)

Dans cette partie on va commencer par vérifier la correction sémantique et syntaxique. Pour ce faire on implémente les prédicats concept, instance, role qui va vérifiee si des objets sont de ce type. Voici les cas basique :

```
concept(C) :- cnamea(C), !.
concept(C) :- cnamena(C), !.
instance(I) :- iname(I), !.
role(R) :- rname(R), !.
```

Vu que le Tbox et Abox est réprésenté par liste, on extraite des conceptions correctes dans une liste avec la fonction setof. Puis on vérifie si tous les conceptions est le membre de la liste.

```
concept([C|_]) :- setof(X, cnamea(X), L), member(C, L), !.
concept([C|_]) :- setof(X, cnamena(X), L), member(C, L), !.
```

On vérifie aussi les conception non-atomique avec les prédicats suivants :

```
concept(and(C1,C2)) :- concept(C1), concept(C2), !.
concept(or(C1,C2)) :- concept(C1), concept(C2), !.
concept(not(C)) :- concept(C), !.
concept(some(R,C)) :- role(R), concept(C), !.
concept(all(R,C)) :- role(R), concept(C), !.

concept([and(C1,C2)|_]) :- concept([C1]), concept([C2]), !.
concept([or(C1,C2)|_]) :- concept([C1]), concept([C2]), !.
concept([not(C)|_]) :- concept([C]), !.
concept([some(R,C)|_]) :- rname(R), concept([C]), !.
concept([all(R,C)|_]) :- rname(R), concept([C]), !.
```

On va ensuite déterminer si une conception est bien celui de Tbox. Vu que le Tbox est réprésenté par liste, on commence par son première élément. On vérifie que C1 est bien une conception non-atomique, puis on vérifie que C2 est bien une conception, et on fera les même opération pour les éléments suivants.

```
concept_tbox([]).
concept_tbox([(C1,C2)|O]) :- cnamena(C1), concept(C2), concept_tbox(O), !.
```

Pareillement, on détermine si une conception est bien celui de Tbox. Vu que le Abox est réprésenté par liste et dans deux cas. (*ex.* (michelAnge,personne) et (michelAnge, david, aCree)). On écrit le prédicat comme suivant :

```
concept_abox([]).
concept_abox([(I,C)|O]) :- instance(I), concept(C), inst(I,C), concept_abox(O), !.
concept_abox([(I1,I2,R)|O]) :- instance(I1), instance(I2), role(R), instR(I1,R,I2), concept_abox(O), !.
```

2.2 Vérification de l'auto-référencement(—autoref)

On va ensuite vérifier s'il y a des auto réference dans des conception non-atomique. S'il y en a, au moment de développer les concepts pour n'avoir que des concepts atomiques il y aura une boucle infinie.

On définie tout d'abord le cas le plus basique.

```
autoref(C, C).
```

Ensuite, on le vérifie dans les conception non-atomique.

```
autoref(and(C1,C2), C) :- autoref(C1, C), autoref(C2, C), !.
autoref(or(C1,C2), C) :- autoref(C1, C), autoref(C2, C), !.
autoref(not(C), C) :- autoref(C, C), !.
autoref(some(R,C1), C) :- autoref(C1, C), !.
autoref(all(R,C1), C) :- autoref(C1, C), !.
```

2.3 Determiner le Tbox et Abox(—traitement_Tbox —traitement_Abox)

Avant finir cette partie, on veut d'abord simplifier les expression compliqué en implémentant le prédicat remplace.

```
remplace(C, C) :- cnamea(C), !.
remplace(C, E) :- equiv(C, D), remplace(D, E) ,!.
remplace(not(C1), not(C2)) :-         remplace(C1, C2), !.
remplace(or(C1, C2), or(D1, D2)) :- remplace(C1, D1), remplace(C2, D2), !.
remplace(and(C1, C2), and(D1, D2)) :- remplace(C1, D1), remplace(C2, D2), !.
remplace(some(R, C1), some(R, C2)) :-         remplace(C1, C2), !.
remplace(all(R, C1), all(R, C2)) :- remplace(C1, C2), !.
```

Ensuite, on détermine le Tbox et Abox.

```
traitement_Tbox([],[]).
traitement_Tbox([(C1,C2)|O],[(C1,C3)|O1]) :- remplace(C2,C4), nnf(C4,C3), traitement_Tbox(O,O1), !.

traitement_Abox([],[]).
traitement_Abox([(I,C)|O],[(I,C1)|O1]) :- remplace(C,C2), nnf(C2,C1), traitement_Abox(O,O1), !.
traitement_Abox([(I1,I2,R)|O],[(I1,I2,R)|O1]) :- traitement_Abox(O,O1), !.
```

Chapitre 3

Saisie de la Proposition à Démontrer

3.1 acquisition_prop_type1

On utilise le prédicat *acquisition_prop_type1(Abi,Abi1,Tbox)* qui permet d'ajouter une proposition pour un test du type "I : C".On la simplifie, et la met en forme normale négative et la concatène dans Abox.

```
acquisition_prop_type1(Abi,Abi1,Tbox):- nl,
    write('Entrez le nom de l"instance :'),nl,
    read(I),
    write('Entrez le nom du concept :'),nl,
    read(C),
    remplace(C, C1), nnf(not(C1),C2), concat(Abi,[I,C2],Abi1), !.
```

3.2 acquisition_prop_type2

Pour acquisition_prop_type2, on commence par demander à l'utilisateur d'entrer $C1$ et $C2$ par le biais du prédicat input_prop_type2 .

```
acquisition_prop_type2(Abi,Abi1,Tbox):- nl,
    write('Entrez le nom du premier concept :'),nl,
        read(C1), concept(C1), remplace(C1, CA1),
    write('Entrez le nom du second concept :'),nl,
         read(C2), concept(C2), remplace(C2, CA2),
        nnf(and(CA1, CA2), NCA),
        genere(Nom),
        concat(Abi, [(Nom, NCA)], Abi1), !.
```

Chapitre 4

Démonstration de la Proposition

4.1 tri_Abox

```
tri_Abox([], Lie, Lpt, Li, Lu, Ls).

tri_Abox([ (I,some(R,C)) | Abi], [ (I,some(R,C)) | Lie], Lpt, Li, Lu, Ls) :-
tri_Abox(Abi, Lie, Lpt, Li, Lu, Ls), !.

tri_Abox([ (I,all(R,C)) | Abi], Lie, [ (I,all(R,C)) | Lpt], Li, Lu, Ls) :-
tri_Abox(Abi, Lie, Lpt, Li, Lu, Ls), !.

tri_Abox([ (I,and(C1,C2)) | Abi], Lie, Lpt, [ (I,and(C1,C2)) | Li], Lu, Ls) :-
tri_Abox(Abi, Lie, Lpt, Li, Lu, Ls), !.

tri_Abox([ (I,or(C1,C2)) | Abi], Lie, Lpt, Li, [ (I,or(C1,C2)) | Lu], Ls) :-
tri_Abox(Abi, Lie, Lpt, Li, Lu, Ls), !.

tri_Abox([ (I,C) | Abi], Lie, Lpt, Li, Lu, [ (I,C) | Ls]) :-
tri_Abox(Abi, Lie, Lpt, Li, Lu, Ls), !.

tri_Abox([ (I,not(C)) | Abi], Lie, Lpt, Li, Lu, [ (I,not(C)) | Ls]) :-
tri_Abox(Abi, Lie, Lpt, Li, Lu, Ls), !.
```

On fait un prédicat *clash* pour tester s'il y a un clash dans l'arbre

```
clash([]).
clash([(I,C) | Ls]) :-        not(member((I,not(C)), Ls)), clash(Ls), !.
```

4.2 resolution

On implémente le prédicat *resolution* qui prend en argument un ABox triée et qui applique la méthode des tableaux, elle renvoie vraie si et seulement si une feuille ouverte est trouvée.

```
resolution(Lie, Lpt, Li, Lu, Ls, Abr) :-
pas_clash(Ls), complete_some(Lie, Lpt, Li, Lu, Ls, Abr), !.

resolution([], Lpt, Li, Lu, Ls, Abr) :-
pas_clash(Ls), transformation_and([], Lpt, Li, Lu, Ls, Abr), !.

resolution([], Lpt, [], Lu, Ls, Abr) :-
pas_clash(Ls), deduction_all([], Lpt, [], Lu, Ls, Abr), !.

resolution([], [], [], Lu, Ls, Abr):-
pas_clash(Ls), transformation_or([], [], [], Lu, Ls, Abr), !.

resolution([], [], [], [], Ls, Abr):- pas_clash(Ls), !.
```

4.3 complete_some

On souhaite traiter une instanciation $I_1 : \exists R.C$.

```
complete_some(Lie,Lpt,Li,Lu,Ls,Abr).
complete_some([],Lpt,Li,Lu,Ls,Abr) :- transformation_and([],Lpt,Li,Lu,Ls,Abr),!.
complete_some([(I1,some(R,C))|Lie],Lpt,Li,Lu,Ls,Abr) :-
    genere(I2),evolue((I2,C),Lie,Lpt,Li,Lu,Ls,Lie1,Lpt1,Li1,Lu1,Ls1),
    complete_some(Lie,Lpt1,Li1,Lu1,Ls1,[(I1,I2,R)|Abr]),!.
```

4.4 transformation_and

On souhaite traiter une instanciation $I : C_1 \sqcap C_2$.

```
transformation_and(Lie,Lpt,[(I,and(A,B))|Li],Lu,Ls,Abr) :-
    evolue((I,A),Lie,Lpt,Li,Lu,Ls,Lie1,Lpt1,Li1,Lu1,Ls1),
    evolue((I,B),Lie1,Lpt1,Li1,Lu1,Ls1,Lie2,Lpt2,Li2,Lu2,Ls2),
    resolution(Lie2,Lpt2,Li2,Lu2,Ls2,Abr).
```

4.5 deduction_all

On souhaite traiter une instanciation $I_1 : \forall R.C$.

```
deduction_all(Lie,[],Li,Lu,Ls,Abr) :- transformation_or(Lie,[],Li,Lu,Ls,Abr).
deduction_all(Lie,[(I,all(R,C))|Lpt],Li,Lu,Ls,Abr) :-
    member((I,B,R),Abr),
    evolue((B,C),Lie,Lpt,Li,Lu,Ls,Lie1,Lpt1,Li1,Lu1,Ls1),
    resolution(Lie1,Lpt1,Li1,Lu1,Ls1,Abr).
```

4.6 transformation_or

On souhaite traiter une instanciation $I : C_1 \sqcup C_2$.

```
transformation_or(Lie,Lpt,Li,[(I,or(C1,C2))|Tu],Ls,Abr):-
    evolue((I,C1),Lie,Lpt,Li,Tu,Ls,Lie1g,Lpt1g,Li1g,Lu1g,Ls1g),
    affiche_evolution_Abox(Ls,Lie,Lpt,Li,[(I,or(C1,C2))|Tu],Abr,Ls1g,Lie1g,Lpt1g,Li1g,Lu1g,Abr),
    evolue((I,C2),Lie,Lpt,Li,Tu,Ls,Lie1d,Lpt1d,Li1d,Lu1d,Ls1d),
    affiche_evolution_Abox(Ls,Lie,Lpt,Li,[(I,or(C1,C2))|Tu],Abr,Ls1d,Lie1d,Lpt1d,Li1d,Lu1d,Abr),
    resolution(Lie1g,Lpt1g,Li1g,Lu1g,Ls1g,Abr),
    resolution(Lie1d,Lpt1d,Li1d,Lu1d,Ls1d,Abr).
```

4.7 evolue

Le prédicat ajout une nouvelle assertion A du concept dans l'une des listes Lie, Lpt, Li, Lu, Ls qui décrivent les assertions de concepts de la Abox étendue et Lie1, Lpt1, Li1, Lu1, Ls1 représentent les nouvelles listes mises à jour. Selon le type de l'élément dans A on fait l'ajout de cet élément dans une nouvelle liste évoluée concaténée avec sa liste initiale corresponde.

```
evolue((I,some(R,C)),Lie,Lpt,Li,Lu,Ls,Lie1,Lpt,Li,Lu,Ls) :- concat([(I,some(R,C))],Lie,Lie1),!.
evolue((I,and(C1,C2)),Lie,Lpt,Li,Lu,Ls,Lie,Lpt,Li1,Lu,Ls) :- concat([(I,and(C1,C2))],Li,Li1),!.
evolue((I,or(C1,C2)),Lie,Lpt,Li,Lu,Ls,Lie,Lpt,Li,Lu1,Ls) :- concat([(I,or(C1,C2))],Lu,Lu1),!.
evolue((I,all(R,C)),Lie,Lpt,Li,Lu,Ls,Lie,Lpt1,Li,Lu,Ls) :- concat([(I,all(R,C))],Lpt,Lpt1),!.
evolue((I,not(C)),Lie,Lpt,Li,Lu,Ls,Lie,Lpt,Li,Lu,Ls1) :- cnamea(C), concat([(I,not(C))],Ls,Ls1),!.
evolue((I,not(C)),Lie,Lpt,Li,Lu,Ls,Lie1,Lpt1,Li1,Lu1,Ls1) :-
    not(cnamea(C)), nnf(not(C),NotCnnf), evolue((I,NotCnnf),Lie,Lpt,Li,Lu,Ls,Lie1,Lpt1,Li1,Lu1,Ls1),!.
evolue((I,C),Lie,Lpt,Li,Lu,Ls,Lie,Lpt,Li,Lu,Ls1):- concat([(I,C)],Ls,Ls1),!.
```

4.8 affiche_evolution_Abox

On commence par écrire le prédicat *affiche* pour construir *affiche_evolution_Abox*.

```
affiche([]).
affiche([A|L]):- affiche(A),affiche(L).
affiche((A,B,R)) :- nl,write("<"),write(A),write(","),write(B),write("> : "),write(R).
affiche((I,or(C1,C2))) :- nl,write(I),write(" : "), affiche(C1),write("  "),affiche(C2).
affiche((I,and(C1,C2))) :- nl,write(I),write(" : "), affiche(C1),write("  "),affiche(C2).
affiche((I,C)) :- nl,write(I), write(" : "), affiche(C).
affiche(or(C1,C2)) :- write("("),affiche(C1),write("  "),affiche(C2),write(")").
affiche(and(C1,C2)) :- write("("),affiche(C1),write("  "),affiche(C2),write(")").
affiche(all(R,C)) :- write(""),write(R),write("."),affiche(C).
affiche(some(R,C)) :- write(""), write(R), write("."), affiche(C).
affiche(not(C)) :- write("¬"),affiche(C).
affiche(C) :- write(C).
```

Implémentation de *affiche* à *affiche_evolution_Abox*.

```
affiche_evolution_Abox(Ls1, Lie1, Lpt1, Li1, Lu1, Abr1, Ls2, Lie2, Lpt2, Li2, Lu2, Abr2).
affiche_evolution_Abox(Ls,Lie,Lpt,Li,Lu,Abr,Ls1,Lie1,Lpt1,Li1,Lu1,Abr1) :- nl,
    affiche(Ls1),
    affiche(Lie1),
    affiche(Lpt1),
    affiche(Li1),
    affiche(Lu1),
    affiche(Abr1),nl,
    affiche(Ls2),
    affiche(Lie2),
    affiche(Lpt2),
    affiche(Li2),
    affiche(Lu2),
    affiche(Abr2),!.

affiche([]).
affiche([A|L]):- affiche(A),affiche(L).
affiche((A,B,R)) :- nl,write("<"),write(A),write(","),write(B),write("> : "),write(R).
affiche((I,or(C1,C2))) :- nl,write(I),write(" : "), affiche(C1),write("  "),affiche(C2).
affiche((I,and(C1,C2))) :- nl,write(I),write(" : "), affiche(C1),write("  "),affiche(C2).
affiche((I,C)) :- nl,write(I), write(" : "), affiche(C).
affiche(or(C1,C2)) :- write("("),affiche(C1),write("  "),affiche(C2),write(")").
affiche(and(C1,C2)) :- write("("),affiche(C1),write("  "),affiche(C2),write(")").
affiche(all(R,C)) :- write(""),write(R),write("."),affiche(C).
affiche(some(R,C)) :- write(""), write(R), write("."), affiche(C).
affiche(not(C)) :- write("¬"),affiche(C).
affiche(C) :- write(C).
```