SCIENCES
SORBONNE
UNIVERSITÉ

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

Auteurs : Cherfi Farouck & Boumessaoud Abdelkader

Encadrante : Colette Faucher

Table des matières

1 Partie 1 **2**
- 1.1 Autoref . . . 2
- 1.2 Concept . . . 3
- 1.3 Traitement_Tbox . . . 4
- 1.4 Traitement_Abox . . . 4

2 Partie 2 **5**
- 2.1 Acquisition_prop_type1 . . . 5
- 2.2 Acquisition_prop_type2 . . . 6

3 Partie 3 **7**
- 3.1 Tri_Abox . . . 7
- 3.2 Resolution . . . 8
- 3.3 Complete_some(Lie,Lpt,Li,Lu,Ls,Abr) . . . 8
- 3.4 Transformation_and . . . 8
- 3.5 Deduction_all . . . 9
- 3.6 Transformation_or . . . 9
- 3.7 Evolue . . . 10
- 3.8 Affiche_evolution_Abox . . . 11

1 Partie 1

1.1 Autoref

Pour autoref, nous avons traité les différents cas possibles. Les cas : or, and , not , some, all et equiv.

```
/*Test concept complexe auto-r f rent*/
        /*cas trivial*/
autoref(Ca,Ca).
        /*not*/
autoref(Cna,not(D)) :- autoref(Cna,D),!.
        /*or*/
autoref(Cna,or(D,_)) :- autoref(Cna,D),!.
autoref(Cna,or(_,D)) :- autoref(Cna,D),!.
        /*and*/
autoref(Cna,and(D,_)) :- autoref(Cna,D),!.
autoref(Cna,and(_,D)) :- autoref(Cna,D),!.
        /*some*/
autoref(Cna,some(_,D)) :- autoref(Cna,D),!.
        /*all*/
autoref(Cna,all(_,D)) :- autoref(Cna,D),!.
        /*equiv*/
autoref(Cna,equiv(D,_)) :- autoref(Cna,D),!.
autoref(Cna,equiv(_,D)) :- autoref(Cna,D),!.
/*Test concept complexe pas auto-r f rent*/
pas-autoref(Cna,D) :- not(autoref(Cna,D)).
```

1.2 Concept

le but du prédicat concept/1 est de faire un correction sémantique et syntaxique de manière récursive, afin d'obtenir des concepts valide en vue de la mise en forme de la Tbox et de la Abox

```
/*Correction semantique et syntaxique*/
    /*Concept*/
        /*Atomique*/
concept(C) :-    setof(X,cnamea(X),L),
                 member(C,L),!.
        /*Non-Atomique*/
concept(C) :-    setof(X,cnamena(X),L),
                 member(C,L),!.
        /*not*/
concept(not(C)) :- concept(C),!.
        /*or*/
concept(or(C1, C2)) :-  concept(C1),
                        concept(C2),!.
        /*and*/
concept(and(C1, C2)) :- concept(C1),
                        concept(C2),!.
        /*some*/
concept(some(R, C)) :-  role(R),
                        concept(C),!.
        /*all*/
concept(all(R, C)) :-   role(R),
                        concept(C),!.
        /*Role*/
role(R) :-       setof(X,rname(X),L),
                 member(R,L),!.
        /*Instance*/
instance(I) :- setof(X,iname(X),L),
                 member(I,L),!.
        /*Assertions de concepts*/
instanciation(I,C) :-   instance(I),
                        concept(C),!.
        /*Assertions de roles*/
instanciationR(I1,I2,R) :-      instance(I1),
                                instance(I2),
                                role(R),!.
```

1.3 Traitement_Tbox

Pour le traitement de la Tbox, nous prenons chaque concept non atomique de la TboxNonTraité et nous appliquons les différentes modification : Traitement du couple si c'est un or, and, not , some, all et equiv jusqu'à obtenir que les concepts non atomique. On applique ensuite un nnf sur ce résultat et le rajoute dans la TboxTraitée.

```
        /*Traitement*/
                /*TBox*/
traitement_Tbox([],[]).
traitement_Tbox([(Cna,D)|Reste_Tbox_non_traitee],
[(Cna,DTnnf)|Reste_Tbox_traitee]) :-    traitement_box_couple(D,DT),
nnf(DT,DTnnf),
traitement_Tbox(Reste_Tbox_non_traitee,Reste_Tbox_traitee).
```

1.4 Traitement_Abox

Même principe que la Tbox mais pour la Abox.

```
                /*ABox*/
traitement_Abox([],[]).
traitement_Abox([(I,C)|Reste_Abox_non_traitee],[(I,CTnnf)|Reste_Abox_traitee]) :-
traitement_box_couple(C,CT),nnf(CT,CTnnf),
traitement_Abox(Reste_Abox_non_traitee,Reste_Abox_traitee).
```

2 Partie 2

2.1 Acquisition_prop_type1

Prédicat rattaché au choix de l'utilisateur de l'option 1, c-a-d : réaliser l'acquisition d'une proposition de type : Une instance I appartient a un concept C (I : C).
On pourra alors saisir le I et le C, pour qu'après une vérification sémantique, syntaxique (instanciation\2) et d'absence d'auto-référencement (pas-autoref\2) on procédé a un traitement des expressions de concepts (traitement_box_couple\2) une mise sous forme normale négative (nnf\2) pour finir avec une insertion dans la Abox (concaten\3).

```
/*Proposition de type -> I:C*/
acquisition_prop_type1(Abi,Abi1,Tbox) :-    nl,
                                            write('L instance I = '),
                                            read(I),
                                            nl,
                                            write('Le concept C = '),
                                            read(C),
                                            instanciation(I,C),
                                            pas-autoref(I,C),
                                            traitement_box_couple(C,CT),
                                            nnf(CT, CTnnf),
                                            concat([(I,CTnnf)],Abi,Abi1),
                                            nl.
```

2.2 Acquisition_prop_type2

Prédicat rattaché au choix de l'utilisateur de l'option 2, c-a-d : réaliser l'acquisition d'une proposition de type : La negation de $C1 \sqcap C2 \sqsubseteq$.

On pourra alors saisir les deux concepts C_1 et C_2, pour qu'après une vérification sémantique, syntaxique (concept\2) on procédé a un traitement des expressions de concepts (traitement_box_couple\2) une mise sous forme normale négative (nnf\2) pour finir sur la generation du nom de la nouvelle instance avec (genere\1) et une insertion dans la Abox (concaten\3).

```
/*Proposition de type -> C1andC2 C Vide*/
acquisition_prop_type2(Abi,Abi1,Tbox) :-
                                        nl,
                                        write('Le concept C1 = '),
                                        read(C1),
                                        nl,
                                        write('Le concept C2 = '),
                                        read(C2),
                                        concept(C1),
                                        concept(C2),
                                        traitement_box_couple(and(C1,C2),CT)
                                        nnf(CT, CTnnf),
                                        genere(Nom),
                                        concat([(Nom,CTnnf)],Abi,Abi1),
                                        nl.
```

3 Partie 3

3.1 Tri_Abox

Le prédicat (Tri_Abox\6) sert à repartir les assertions de la Abox en les parcourant une par une de manière récursive tout en insérant dans la bonne catégorie avec le prédicat (concaten\3) ; la répartition se fait en 5 catégories : - Lie : assertions du type (I,some(R,C)) - Lpt : assertions du type (I,all(R,C)) - Li : assertions du type (I,and(C1,C2)) - Lu : assertions du type (I,or(C1,C2)) - Ls : assertions restantes, à savoir les assertions du type (I,C) ou (I,not(C)), C étant un concept atomique.

```
/*tri_Abox : Abe ---> Lie,Lpt,Li,Lu,Ls,Abr*/
    /*some ---> Lie*/
tri_Abox([(I,some(R,C))|Abe_rest],Lie1,Lpt,Li,Lu,Ls) :- instance(I),
concept(some(R, C)),
tri_Abox(Abe_rest,Lie0,Lpt,Li,Lu,Ls),
concat([(I,some(R,C))],Lie0,Lie1).

    /*all ---> Lpt*/
tri_Abox([(I,all(R,C))|Abe_rest],Lie,Lpt1,Li,Lu,Ls) :-  instance(I),
concept(all(R, C)),
tri_Abox(Abe_rest,Lie,Lpt0,Li,Lu,Ls),
concat([(I,all(R,C))],Lpt0,Lpt1).

    /*    ---> Li*/
tri_Abox([(I,and(C1,C2))|Abe_rest],Lie,Lpt,Li1,Lu,Ls) :-
instance(I),
concept(and(C1, C2)),
tri_Abox(Abe_rest,Lie,Lpt,Li0,Lu,Ls),
concat([(I,and(C1,C2))],Li0,Li1).

    /*    ---> Lu*/
tri_Abox([(I,or(C1,C2))|Abe_rest],Lie,Lpt,Li,Lu1,Ls) :- concept(or(C1, C2)),
tri_Abox(Abe_rest,Lie,Lpt,Li,Lu0,Ls),
concat([(I,or(C1,C2))],Lu0,Lu1).

    /*Autre ---> Ls*/
tri_Abox([(I,not(C))|Abe_rest],Lie,Lpt,Li,Lu,Ls1) :-    concept(not(C)),
tri_Abox(Abe_rest,Lie,Lpt,Li,Lu,Ls0),
concat([(I,not(C))],Ls0,Ls1).

tri_Abox([(I,C)|Abe_rest],Lie,Lpt,Li,Lu,Ls1) :- instance(I),
concept(C),tri_Abox(Abe_rest,Lie,Lpt,Li,Lu,Ls0),concat([(I,C)],Ls,Ls1).
```

```
    /*Condition d'arret*/
tri_Abox([],[],[],[],[],[]).
```

3.2 Resolution

Le prédicat (resolution\6) qui prend comme paramètre les 5 catégories cités auparavant après tri, en plus de la Abox dans le but de réaliser la méthode de résolution par tableaux et cela en passants par les 4 predicats (complete_some\6), (transformation_and\6), (deduction_all\6), (transformation_or\6), dont on détaillera les fonctionnalités de chacune d'entre elle dans les sections suivantes.

```
resolution(Lie,Lpt,Li,Lu,Ls,Abr) :- complete_some(Lie,Lpt,Li,Lu,Ls,Abr),
transformation_and(Lie,Lpt,Li,Lu,Ls,Abr),
deduction_all(Lie,Lpt,Li,Lu,Ls,Abr),
transformation_or(Lie,Lpt,Li,Lu,Ls,Abr).
```

3.3 Complete_some(Lie,Lpt,Li,Lu,Ls,Abr)

Sert dans le cas d'une instanciation de type R.C.
On génère une nouvelle instance avec le prédicat (generer\2) pour après y introduire le concept C avec le predicat (evolue\11) qu'on détaillera après dans ce rapport, on finira avec une (resolution\6) dont on a deja détaillé les faits.

```
complete_some([],_,_,_,_,_).(resolution\textbackslash6)
complete_some([(I,some(R,C))|Lie0_reste],Lpt0,Li0,Lu0,Ls0,Abr0):-
genere(N),
evolue((N,C), Lie0_reste, Lpt0, Li0, Lu0, Ls0, Lie1, Lpt1, Li1, Lu1, Ls1),
concat([(I,N,R)],Abr,Abr1),
affiche_evolution_Abox(Ls0, Lie0_reste, Lpt0, Li0, Lu0,
Abr0, Ls1, Lie1, Lpt1, Li1, Lu1, Abr1),
resolution(Lie1, Lpt1 ,Li1 ,Lu1 ,Ls1 ,Abr1).
```

3.4 Transformation_and

Sert dans le cas d'une instanciation de type $C_1 \sqcap C_2$
Dans ce cas il faudra introduire les deux concepts $C_1 et C_2$ avec l'appel de (evolue\11) pour dinir sur une (resolution\6).

```
transformation_and(_,_,[],_,_,_).
transformation_and(Lie0;Lpt0,[(I,and(C1,C2))|Li0_reste],Lu0,Ls0,Abr0):-
evolue((I,C1), Lie0, Lpt0, Li0_reste, Lu0, Ls0, Lie1, Lpt1, Li1, Lu1, Ls1),
evolue((I,C2),
Lie0, Lpt0, Li0_reste, Lu0, Ls0, Lie1, Lpt1, Li1, Lu1, Ls1),
affiche_evolution_Abox(Ls0, Lie0, Lpt0, Li0_reste, Lu0,
Abr0, Ls1, Lie1, Lpt1, Li1, Lu1, Abr1),
resolution(Lie1, Lpt1 ,Li1 ,Lu1 ,Ls1 ,Abr1).
```

3.5 Deduction_all

Sert dans le cas d'une instanciation de type : R.C
Après recherche dans l'Abox de X tel que < I1, X > : R et cela a l'aide du prédicat (member\2) pour après enchaîner sur une insertion avec (evolue\11).

```
deduction_all(_,[],_,_,_,_).
deduction_all(Lie0,[(I,all(R,C))|Lpt0_reste],Li0,Lu0,Ls0, Abr0) :-
member((I,X,R),Abr0),
evolue((X,C), Lie0, Lpt0_reste, Li0, Lu0, Ls0,
Lie1, Lpt1, Li1, Lu1, Ls1),
affiche_evolution_Abox(Ls0, Lie0, Lpt0_reste, Li0
, Lu0, Abr0, Ls1, Lie1, Lpt1, Li1, Lu1, Abr1),
resolution(Lie1, Lpt1, Li1, Lu1, Ls1, Abr1).
```

3.6 Transformation_or

Sert dans le cas d'une instanciation de type : $C_1 \sqcup C_2$
On passera par les mêmes étapes que (Transformation_and\6) :
(evolue\11) puis (resolution\6).

```
    /*    */
transformation_or(Lie0 ,Lpt0 ,Li0 ,[(I,or(C1,_))|Lu0_reste], Ls0, Abr0) :-
evolue((I,C1), Lie0, Lpt0, Li0, Lu0_reste, Ls0, Lie1, Lpt1, Li1, Lu1, Ls1),
affiche_evolution_Abox(Ls0, Lie0, Lpt0, Li0, Lu0_reste,
Abr0, Ls1, Lie1, Lpt1, Li1, Lu1, Abr1),
resolution(Lie1, Lpt1, Li1, Lu1, Ls1, Abr1).

transformation_or(Lie0 ,Lpt0 ,Li0 ,[(I,or(_,C2))|Lu0_reste], Ls0, Abr0) :-
evolue((I,C2), Lie0, Lpt0, Li0, Lu0_reste, Ls0, Lie1, Lpt1, Li1, Lu1, Ls1),
affiche_evolution_Abox(Ls0, Lie0, Lpt0, Li0, Lu0_reste,
Abr0, Ls1, Lie1, Lpt1, Li1, Lu1, Abr1),
resolution(Lie1, Lpt1, Li1, Lu1, Ls1, Abr1).
```

3.7 Evolue

Sert lors des 4 predicats listés précédemment ; et cela pour but d'ajouter la nouvelle insertion issue d'une des règles dans la l'une des listes adéquat : Lie,Li,Ls,Lpt,Lu

```
/*Ajout d'un nouvelle assertion dans les listes  Lie, Lpt, Li, Lu ou Ls*/
    /*     --> Lie*/
evolue((I,some(R,C)),Lie,Lpt,Li,Lu,Ls,Lie1,Lpt,Li,Lu,Ls) :-
concept(some(R, C)), concat([(I,some(R,C))],Lie,Lie1).

    /*     --> Lpt*/
evolue((I,all(R,C)),Lie,Lpt,Li,Lu,Ls,Lie,Lpt1,Li,Lu,Ls) :-
instance(I),concept(all(R, C)),
concat([(I,all(R,C))],Lpt,Lpt1).

    /*     --> Li*/
evolue((I,and(C1,C2)),Lie,Lpt,Li,Lu,Ls,Lie,Lpt,Li1,Lu,Ls) :-
instance(I), concept(and(C1, C2)),
    concat([(I,and(C1,C2))],Li,Li1).

    /*     --> Lu*/
evolue((I,or(C1,C2)),Lie,Lpt,Li,Lu,Ls,Lie,Lpt,Li,Lu1,Ls) :-
instance(I),
    concept(or(C1, C2)),
    concat([(I,or(C1,C2))],Lu,Lu1).

    /*Autre --> Ls*/
evolue((I,not(C)),Lie,Lpt,Li,Lu,Ls,Lie,Lpt,Li,Lu,Ls1) :-
instance(I),
    concept(not(C)),
    concat([(I,not(C))],Ls,Ls1).
evolue((I,C),Lie,Lpt,Li,Lu,Ls,Lie,Lpt,Li,Lu,Ls1):-  instance(I),
    concept(C),
    concat([(I,C)],Ls,Ls1).
```

3.8 Affiche_evolution_Abox

Seul but de ce predicat est la vérification visuelle des différences apportés aux listes d'assertions de concepts et d'assertion de rôles, sous forme d'un avant/apres (before/after dans notre code).

```
/*Predicats d'affichage*/
    /*Evolution Abox*/
affiche_evolution_Abox(Ls0, Lie0, Lpt0, Li0, Lu0, Abr0, Ls1, Lie1, Lpt1, Li1,
Lu1, Abr1) :- write("before"),nl,
    write("Assertions de concept"),
    nl,
    affiche_liste(Ls0),
    affiche_liste(Lie0),
    affiche_liste(Lpt0),
    affiche_liste(Li0),
    affiche_liste(Lu0),
    write("Assertions de roles"),
    nl,
    affiche_listeAbr(Abr0),
    nl,
    write("after"),
    nl,
    write("Assertions de concept"),
    nl,
    affiche_liste(Ls1),
    affiche_liste(Lie1),
    affiche_liste(Lpt1),
    affiche_liste(Li1),
    affiche_liste(Lu1),
    write("Assertions de roles"),
    nl,
    affiche_listeAbr(Abr1),
    nl.
```