Rapport de Projet

LRC

MU4IN800

Encadrant: Colette Faucher

Antoine TOULLALAN

Jérémy DUFOURMANTELLE

Introduction

Ce projet a été réalisé dans le cadre de l'UE de LRC par Jérémy DUFOURMANTELLE et Antoine TOULALLAN. Le but de ce projet est d'implémenter un démonstrateur basé sur la méthode des tableaux pour la logique de description ALC avec le langage Prolog. Ce rapport présente les 3 parties de développement de ce démonstrateur. Nous avons écrit tout notre projet est dans le fichier "projet.pl". La première étape aura pour but de charger les définitions de concepts et leurs assertions. Ensuite nous développerons la partie qui necessite la saisie d'une proposition qui pourra prendre deux formes. Enfin, nous présenterons le coeur du démonstrateur.

Partie I : Préliminaire

Nous avons commencé par écrire dans notre fichier "projet.pl" la ABox et la TBox fournit dans l'énoncé. On introduit dans la TBox les prédicats equiv (pour les équivalences des concepts complexes),cnmae (déclaration des conceptes atomiques),cnamena (déclaration des conceptes complexes), iname (déclaration des instances), rname (déclaration des rôles) . Puis on introduit dans la ABox les prédicats inst (déclaration des instanciations de concepts) et instR (déclaration des instanciations de rôles).
On écrit aussi le prédicat " premiere_etape " qui fournit les listes correspondant à la Tbox et à la Abox:

Nous allons présenter dans cette partie comment se déroule le chargement des connaissances des instanciations de concepts (Abi), de roles (Abr) et de définitions de concepts complexes (Tbox).

Tout d'abord nous allons définir le prédicat **premiere_etape(TBox,Abi,Abr)** qui permet d'effectuer l'opération de chargement des données. Nous allons pour cela utiliser le prédicat **setof(Motif,But,Liste)** définit par Prolog qui nous permet de récupérer tous les les concepts complexes (Ligne 49), instanciations de concept(Ligne 50) et la liste des instanciations de role (Ligne 51).

```
48  premiere_etape(Tbox,Abi,Abr) :-
49      setof((X,Y),equiv(X,Y),Tbox),
50      setof((X,Y),inst(X,Y),Abi),
51      setof((X,Y,Z),instR(X,Y,Z),Abr).
```

Serie de test sur premiere_etape(TBox,Abi,Abr) :

```
[4]  ?- premiere_etape(TBox,Abi,Abr).
TBox = [(auteur, and(personne, some(aEcrit, livre))),  (editeur, and(personne, and(not(some(aEcrit, livre)), some(aEdite, livre)))),  (parent, and(personne, s
ome(aEnfant, anything))),  (sculpteur, and(personne, some(aCree, sculpture)))],
Abi = [(david, sculpture),  (joconde, objet),  (michelAnge, personne),  (sonnets, livre),  (vinci, personne)],
Abr = [(michelAnge, david, aCree),  (michelAnge, sonnets, aEcrit),  (vinci, joconde, aCree)].
```

Nous pouvons voir que le prédicat charge correctement les valeurs des instanciations dans la TBox,Abi et Abr qui nous servirons à l'étape 2.

Partie II : Saisie de la proposition à démontrer

Pour faire le lien entre la premiere étape et la deuxième étape, nous recopions dans notre code les prédicats **programme** et **deuxieme_etape(Abi,Abi1,Tbox)** fourni dans l'énoncé du projet.

```
57  deuxieme_etape(Abi,Abi1,Tbox) :-
58   saisie_et_traitement_prop_a_demontrer(Abi,Abi1,Tbox).
59
60  saisie_et_traitement_prop_a_demontrer(Abi,Abi1,Tbox) :-
61   nl,write("Entrez le numero du type de proposition que vous voulez demontrer :"),
62   nl,write("1 : Une instance donnee appartient a un concept donne. (I : C)"),
63   nl,write("2 : Deux concepts ne possede aucun elements en commun(ils ont une intersection vide => C1 ⊓ C2 ⊑ ⊥ ) : "),
64   nl,write("3 : Pour sortir de l'invité de commande"),
65   nl,read(R),
66   suite(R,Abi,Abi1,Tbox).
67
68  suite(1,Abi,Abi1,Tbox) :- acquisition_prop_type1(Abi,Abi1,Tbox),!.
69  suite(2,Abi,Abi1,Tbox) :- acquisition_prop_type2(Abi,Abi1,Tbox),!.
70  suite(3,_,_,_) :- nl,write("Vous êtes sorti de l'invité de commande"),!.
71  suite(_,Abi,Abi1,Tbox) :-
72   nl, write("Cette reponse est incorrecte."),
73   nl, saisie_et_traitement_prop_a_demontrer(Abi,Abi1,Tbox).
```

A noter que nous avons modifié ce prédicat de façon à pouvoir annuler la saisie en selectionnant la troisième option. Nous présenterons le jeu de test associé à ce prédicat une fois qu'on aura présenté tous les prédicats sous-jacents.

Saisie de la proposition de Type 1 ($I : C$)

```
76  acquisition_prop_type1(Abi , Abi1 ,Tbox) :-
77   nl,write("Entrer linstance I :"),
78   nl,read(I),
79   nl,write("Entrer le concept C :"),
80   nl,read(C),
81   verifSemantiqueConcept(C),
82   verifSemantiqueConcept(I),
83   concept(C),
84   replace(C,RC),
85   nnf(not(RC),NRC),
86   concatList(Abi,[(I,NRC)],Abi1).
```

Pour ce prédicat,"prop_type_1", on charge d'abord l'instance I et le concept C, on vérifie qu'ils sont sémantiquement (grâce à verifSemantiqueConcept) et syntaxiquement (grâce à concept) corrects, puis on ajoute not(C) dans la ABox.
On voit avec le test suivant que ce prédicat a un fonctionnement correct:

Serie de test sur acquisition_prop_type1(Abi,Abi1,Tbox) :

```
[4]  ?- acquisition_prop_type1([],Abi1,[]).

Entrer linstance I :
|: david.

Entrer le concept C :
|: and(some(aCree,personne),or(all(aEcrit,livre),objet)).

Abi1 = [(david, or(all(aCree, not(personne)), and(some(aEcrit, not(livre)), not(objet))))].
```

En effet, on a bien rajouté dans la liste en argument la négation du concept C, on va pouvoir ajouter nnf(C) dans la Abox et vérifier sa validité.

Néanmoins nous voyons que ce prédicat n'a pas un fonctionnement correct pour "all", commoe on peut le voir ci-dessous:

```
[4]  ?- acquisition_prop_type1([],Abi1,[]).

Entrer linstance I :
|: vinci.

Entrer le concept C :
|: all(personne,objet).

false.
```

Saisie de la proposition de Type 2 ($C_1 \sqcap C_2 \sqsubseteq \bot$)

Nous avons écrit le prédicat acquisition_prop_type2(Abi,Abi1,Tbox) (voir image ci-dessous) qui permet d'entrer dans la Abox la négation de and(c1,c2). Pour cela, on charge d'abord C1 dont on vérifie qu'il est correct sémantiquement et syntaxiquement, on fait la même chose pour C2.On ajoute ensuite nnf(and(C1,C2)) dans la liste Abox entrée en argument.

```
90  acquisition_prop_type2(Abi,Abi1,Tbox) :-
91   nl,write("Entrer le premier concept de lintersection : "),
92   nl,read(C1),
93   verifSemantiqueConcept(C1),
94   concept(C1),
95   nl,write("Entrer le deuxieme concept de lintersection : "),
96   nl,read(C2),
97   verifSemantiqueConcept(C2),
98   concept(C2),
99   replace(and(C1,C2),RC),
100  nnf(not(RC),NRC),
101  genere(Nom),
102  concatList(Abi,[(Nom,NRC)],Abi1).
```

On vérifie la validité de ce prédicat en effectuant des tests comme le suivant:

```
[?- acquisition_prop_type2([],Abi,[]).

Entrer le premier concept de lintersection :
[|: personne.

Entrer le deuxieme concept de lintersection :
[|: sculpture.



[(inst1,or(not(personne),not(sculpture)))]
Abi = [(inst1, or(not(personne), not(sculpture)))] .
```

On voit bien que lors du test, on a ajouté dans la liste en entrée la négation de and(personne,sculpture) qui est or(not(personne),not(sculpture)).

Vérification syntaxique d'un concept C

On doit vérifier que les concepts en entrée sont bien corrects syntaxiquement, on utilise donc le prédicat **concept(C)** pour vérifier que C est bien un concept conformément à la grammaire suivante :

$$
\begin{aligned}
\text{concept} \quad ::= \; & \langle \text{concept atomique} \rangle \\
| \; & \top \\
| \; & \bot \\
| \; & \neg \, \langle \text{concept} \rangle \\
| \; & \langle \text{concept} \rangle \; \sqcup \; \langle \text{concept} \rangle \\
| \; & \langle \text{concept} \rangle \; \sqcap \; \langle \text{concept} \rangle \\
| \; & \exists \; \langle \text{rôle} \rangle \; . \; \langle \text{concept} \rangle \\
| \; & \forall \; \langle \text{rôle} \rangle \; . \; \langle \text{concept} \rangle
\end{aligned}
$$

Nous allons définir récursivement le prédicat concept(C) afin de vérifier les différentes règles de la grammaires (Ligne : 165 à 169). Nous avons introduit un prédicat différent **conceptRole(R)** qui va vérifier que R est bien un role (Ligne : 170). Enfin les lignes 161 à 164 permettent de vérifier que le concept C est soit un concept atomique, soit un concept complexe.

```
161   concept(nothing).
162   concept(anything).
163   concept(C) :- cnamea(C),!.
164   concept(C) :- cnamena(C),! .
165   concept(and(C1,C2)) :- concept(C1), concept(C2), !.
166   concept(or(C1,C2)) :- concept(C1), concept(C2), !.
167   concept(all(R,C)) :- concept(C) ,conceptRole(R),!.
168   concept(some(R,C)) :- concept(C),conceptRole(R) ,!.
169   concept(not(C)) :- concept(C) ,!.
170   conceptRole(R) :- rname(R),!.
```

Serie de test sur concept(C) :

```
?- concept(sculpture).
true.

?- concept(and(parent,objet)).
true.

?- concept(all(aCree,some(aEcrit,livre))).
true.

?- concept(some(personne,objet)).
false.

?- concept(and(objet,aCree)).
false.
```

Vérification sémantique d'un concept C

Dans cette partie, nous tenions à vérifier si l'entrée d'un concept, ou d'une instance faisait belle et bien partie des assertions de notre ABox. Pour cela, nous avons concu le prédicat récursif **verifSemantiqueConcept(C)**. Nous pouvons voir que les lignes 142 à 145 permettent de savoir si C appartient à une des listes fourni par **setof** qui contient les différents type d'instance et de concepts.

Les lignes 147 à 151 définissent la récursion de ce prédicat.

```
142   verifSemantiqueConcept(C) :- setof(C, iname(C),L), member(C,L) ,!.
143   verifSemantiqueConcept(C) :- setof(C, rname(C),L), member(C,L) ,!.
144   verifSemantiqueConcept(C) :- setof(C, cnamena(C),L), member(C,L) ,!.
145   verifSemantiqueConcept(C) :- setof(C, cnamea(C),L), member(C,L) ,!.
146
147   verifSemantiqueConcept(and(C1,C2)) :- verifSemantiqueConcept(C1),verifSemantiqueConcept(C2),!.
148   verifSemantiqueConcept(or(C1,C2)) :- verifSemantiqueConcept(C1),verifSemantiqueConcept(C2),!.
149   verifSemantiqueConcept(all(R,C)) :- verifSemantiqueConcept(R),verifSemantiqueConcept(C),!.
150   verifSemantiqueConcept(some(R,C)) :- verifSemantiqueConcept(R),verifSemantiqueConcept(C),!.
151   verifSemantiqueConcept(not(C)) :- verifSemantiqueConcept(C),!.
```

Serie de test sur verifSemantiqueConcept(C) :

```
?- verifSemantiqueConcept(personne).
true.

?- verifSemantiqueConcept(and(personne,objet)).
true.

?- verifSemantiqueConcept(and(all(aCree,livre),objet)).
true.

?- verifSemantiqueConcept(lrc).
false.

?- verifSemantiqueConcept(and(all(aVendu,livre),objet)).
false.
```

Remplacement des concepts complexes par leur définitions dans la liste de la ABox

```
114  replace(and(C1,C2),and(RC1,RC2)) :- replace(C1,RC1), replace(C2,RC2), !.
115  replace(or(C1,C2),or(RC1,RC2)) :- replace(C1,RC1), replace(C2,RC2), !.
116  replace(all(R,C),all(R,RC)) :- replace(C,RC) ,!.
117  replace(some(R,C),some(R,RC)) :- replace(C,RC) ,!.
118  replace(not(C),not(RC)) :- replace(C,RC) ,!.
119  replace(C,RC) :- setof(X,cnamena(X),L) , member(C,L) , equiv(C,RC),!.
120  replace(C,C) :- setof(X,cnamea(X),L) , member(C,L),!.
```

Ce prédicat permet de remplacer un concept complexe par son équivalence d'après notre TBox. Il permet de simplifier au mieux un concept jusqu'à que ce dernier puisse s'exprimer uniquement avec des concepts atomiques.

Pour cela, nous avond défini les cas d'arret de ce prédicat aux lignes 119 et 120. La ligne 120 permet de garder notre concept tel qu'il est car c'est un concept atomique. La ligne 119 permet de remplacer le concept complexe par sa définition grâce à **equiv**. Les lignes 114 à 118 permettent juste de faire le cas de récursion en gardant toutefois le symbole.

Serie de test sur replace(C1,C2) :

```
[1]  ?- replace(personne,C2).
C2 = personne.

[1]  ?- replace(parent,C2).
C2 = and(personne, some(aEnfant, anything)).

[1]  ?- replace(and(or(parent,sculpteur),all(aCree,editeur)),C2).
C2 = and(or(and(personne, some(aEnfant, anything)), and(personne, some(aCree, sculpture))), all(aCree, and(personne, and(not(some(aEcrit, livre)), some(aEdite
, livre))))).
```

Forme Normale négative d'un concept

Pour pouvoir démontrer que notre proposition à vérifier est valide, nous aurons alors besoin de la transformer en forme normale négative afin de pouvoir appliquer la méthode des tableaux que nous allons voir dans la partie 3. Voici ci-dessous le code de la transformation d'une formule en forme normale négative. Ce code étant fourni dans l'énoncé du projet, nous n'allons faire aucun test dessus.

```
126   nnf(not(and(C1,C2)),or(NC1,NC2)):- nnf(not(C1),NC1),
127   nnf(not(C2),NC2),!.
128   nnf(not(or(C1,C2)),and(NC1,NC2)):- nnf(not(C1),NC1),
129   nnf(not(C2),NC2),!.
130   nnf(not(all(R,C)),some(R,NC)) :- nnf(not(C),NC),!.
131   nnf(not(some(R,C)),all(R,NC)):- nnf(not(C),NC),!.
132   nnf(not(not(X)),X):-!.
133   nnf(not(X),not(X)):-!.
134   nnf(and(C1,C2),and(NC1,NC2)):- nnf(C1,NC1),nnf(C2,NC2),!.
135   nnf(or(C1,C2),or(NC1,NC2)):- nnf(C1,NC1), nnf(C2,NC2),!.
136   nnf(some(R,C),some(R,NC)):- nnf(C,NC),!.
137   nnf(all(R,C),all(R,NC)) :- nnf(C,NC),!.
138   nnf(X,X).
```

Serie de test global sur deuxieme_etape(Abi,Abi1,Tbox) :

Dans ce test, on peut voir que les deux types de proposition ont réussi à être ajouté à la ABox étendu qui se nomme Abi1.

```
[1]  ?- deuxieme_etape([(michelAnge,personne), (david,sculpture), (sonnets,livre), (vinci,personne), (joconde,objet)],Abi1,[(sculpteur,and(personne,some(aCree
,sculpture))), (auteur,and(personne,some(aEcrit,livre))), (editeur,and(personne,and(not(some(aEcrit,livre)),some(aEdite,livre)))),(parent,and(personne,some(a
nfant,anything)))]).

Entrez le numero du type de proposition que vous voulez demontrer :
1 : Une instance donnee appartient a un concept donne. (I : C)
2 : Deux concepts ne possede aucun elements en commun(ils ont une intersection vide => C1 ⊓ C2 ⊑ ⊥ ) :
3 : Pour sortir de l'invité de commande
|: 1.

Entrer linstance I :
|: david.

Entrer le concept C :
|: personne.

Abi1 = [(michelAnge, personne),  (david, sculpture),  (sonnets, livre),  (vinci, personne),  (joconde, objet),  (david, not(personne))].

[1]  ?- deuxieme_etape([(michelAnge,personne), (david,sculpture), (sonnets,livre), (vinci,personne), (joconde,objet)],Abi1,[(sculpteur,and(personne,some(aCree
,sculpture))), (auteur,and(personne,some(aEcrit,livre))), (editeur,and(personne,and(not(some(aEcrit,livre)),some(aEdite,livre)))),(parent,and(personne,some(aE
nfant,anything)))]).

Entrez le numero du type de proposition que vous voulez demontrer :
1 : Une instance donnee appartient a un concept donne. (I : C)
2 : Deux concepts ne possede aucun elements en commun(ils ont une intersection vide => C1 ⊓ C2 ⊑ ⊥ ) :
3 : Pour sortir de l'invité de commande
|: 2.

Entrer le premier concept de lintersection :
|: objet.

Entrer le deuxieme concept de lintersection :
|: parent.



Abi1 = [(michelAnge, personne),  (david, sculpture),  (sonnets, livre),  (vinci, personne),  (joconde, objet),  (inst4, or(not(objet), or(not(...), all(..., .
..))))].
```

Partie III : Démonstration de la proposition

Nous allons dans cette partie expliquer les différentes parties du démonstrateur.

Troisième étape

```
183  troisieme_etape(Abi,Abr) :-
184      tri_Abox(Abi,Lie,Lpt,Li,Lu,Ls),
185      not(resolution(Lie,Lpt,Li,Lu,Ls,Abr)),
186      nl,write('Youpiiiiii, on a demontre la proposition initiale !!!').
```

Nous allons redefinir le prédicat **troisieme_etape(Abi,Abr)** afin qu'il fasse la négation de la resolution car par la suite nous allons retourner faux si il y a une présence de contradiction. Cela est dû au prédicat qui test si un clash est présent dans la ABox et si oui alors il retourne faux. Par conséquent on doit appliquer la négation car on veut écrire "Youpi" si justement il y a une contradiction.

Tri de la ABox

Le prédicat Tri_Abox prend en argument une liste L de concepts complexes ou atomiques et des listes contenant les concepts triés. On renvoie les listes triées avec les concepts de L dans les bonnes listes:

```
188  tri_Abox([],[],[],[],[],[]).
189  tri_Abox([(I,some(R,C))|Abi],[(I,some(R,C))|Lie],Lpt,Li,Lu,Ls)
190    :- tri_Abox(Abi,Lie,Lpt,Li,Lu,Ls),!.
191  tri_Abox([(I,all(R,C))|Abi],Lie,[(I,all(R,C))|Lpt],Li,Lu,Ls)
192    :-tri_Abox(Abi,Lie,Lpt,Li,Lu,Ls),!.
193  tri_Abox([(I,and(C1,C2))|Abi],Lie,Lpt,[(I,and(C1,C2))|Li],Lu,Ls)
194    :- tri_Abox(Abi,Lie,Lpt,Li,Lu,Ls),!.
195  tri_Abox([(I,or(C1,C2))|Abi],Lie,Lpt,Li,[(I,or(C1,C2))|Lu],Ls)
196    :- tri_Abox(Abi,Lie,Lpt,Li,Lu,Ls),!.
197  tri_Abox([(I,C)|Abi],Lie,Lpt,Li,Lu,[(I,C)|Ls])
198    :- setof(X,cnamea(X),L), member(C,L) , tri_Abox(Abi,Lie,Lpt,Li,Lu,Ls),!.
199    tri_Abox([(I,not(C))|Abi],Lie,Lpt,Li,Lu,[(I,not(C))|Ls])
200    :- setof(X,cnamea(X),L), member(C,L) , tri_Abox(Abi,Lie,Lpt,Li,Lu,Ls),!.
```

Ainsi pour chaque type de concept (or,and,some... ou atomique), on associe une liste.

Serie de test sur tri_Abox(Abi,Lie, Lpt, Li,Lu, Ls) :

```
?- tri_Abox([  (david,personne) , (michelAnge,and(personne,livre)) , (personne,or(livre,objet)) , (livre,all(aEcrit,michelAnge)) , (pe
rsonne,some(aCree,personne)) ], Lie , Lpt , Li , Lu , Ls).
Lie = [(personne, some(aCree, personne))],
Lpt = [(livre, all(aEcrit, michelAnge))],
Li = [(michelAnge, and(personne, livre))],
Lu = [(personne, or(livre, objet))],
Ls = [(david, personne)].
```

On voit que pour chaque concept de la liste en argument, on a un tri correct en sortie avec un tri en fonction de some, all, and ,or et les concepts atomiques.

Application des régles de résolution : le prédicat resolution(Lie,Lpt,Li,Lu,Ls,Abr)

```
202  resolution(Lie,Lpt,Li,Lu,Ls,Abr) :-
203      verif_contradiction(Ls),
204      complete_some(Lie,Lpt,Li,Lu,Ls,Abr),
205      transformation_and(Lie,Lpt,Li,Lu,Ls,Abr),
206      deduction_all(Lie,Lpt,Li,Lu,Ls,Abr),
207      transformation_or(Lie,Lpt,Li,Lu,Ls,Abr).
```

Ce prédicat va être appelé dans le prédicat de la troisième partie pour appliquer les différentes règles de résolution. Comme on peut le voir dans la capture ci-dessus, on applique tout d'abord une vérification de test de clash (que l'on présentera juste après) qui vérifie si il y a une contradiction déja présente dans la ABox. On est obligé de faire cette vérification car si les 4 règles réussissent alors on retournera vrai alors que l'on doit retourner faux conformément à la redefinition que l'on a fait pour de l'appel à resolution dans le prédicat troisieme_etape.

Test sur la présence d'un clash dans une liste de la Abox

Dans le démonstrateur, nous aurons besoin de vérifier si dans chaque noeud il y a la présence d'un clash autrement dit une contradiction tel que $I : C$ et $I : \neg C$

```
275  verif_contradiction_elem([],(_,_)).
276
277  verif_contradiction_elem([(U,A)|L],(I,E)) :-
278      setof(X,iname(X),L3) , member(I,L3), member(U,L3),
279      (U \== I ), verif_contradiction_elem(L,(I,E)),!.
280
281
282  verif_contradiction_elem([(U,A)|L],(I,E)) :-
283      setof(X,iname(X),L3) , member(I,L3), member(U,L3),
284      U == I,
285      A \== not(E),
286      E \== not(A),
287      setof(X,iname(X),L3) , member(I,L3),
288      verif_contradiction_elem(L,(I,E)),!.
289
290  verif_contradiction_elem([(U,A)|L],(I,E)) :-
291      setof(X,iname(X),L3) , not(member(I,L3)),
292      A \== not(E),
293      E \== not(A),
294      verif_contradiction_elem(L,(I,E)),!.
295
296  verif_contradiction([]).
297  verif_contradiction([(I,E)|L]) :-
298      verif_contradiction_elem(L,(I,E)),
299      verif_contradiction(L).
```

Le but de ce prédicat est d'itérer sur les couples instance/concept de la liste donné en paramètre et de vérifier si le reste de la liste possède son complémentaire. Les Lignes 296 à 299 constituent le point d'entrée du prédicat. La ligne 298 va itérer sur les éléments de la liste tant qu'il y aura aucune contradiction et sortira à la ligne 296 si aucune contradiction n'est détécté. Le prédicat sera vrai si tel est le cas et donc faux si il y a une contradiction.
Ce prédicat prend en compte les instances générique, c'est à dire des instances que nous aurons générer avec le prédicat **genere(B)**. Cela permet de vérifier une contradiction qui pourrait arriver dynamiquement lors de l'application de la règle ∃. On vérifie un tel cas avec le bout de code Ligne 290 à 294 grâce au prédicat **member**, on aura alors juste besoin de vérifier A et E ne soient pas complémentaire. Dans le cas ou A et E appartiendraient à des instances existantes, on doit alors vérifier si ils sont égaux (Ligne 279 et 284). Si ils ne le sont pas alors c'est trivial il n'y a pas de contradiction (ligne 279). Si ils sont egaux alors on est obligé de faire une vérification sur A et E.

Serie de test sur verif_contradiction(L) :

```
?- verif_contradiction([(david,personne) , (michelAnge,personne)]).
true.

?- verif_contradiction([(david,personne) , (michelAnge,personne) , (david,not(personne))]).
false.

?- verif_contradiction([(david,personne) , (michelAnge,livre) , (inst,not(personne))]).
false.

?- verif_contradiction([]).
true.
```

Intégration d'une nouvelle assertion de concept à intégrer dans une des listes de la ABox : Le prédicat evolue((I,C), Lie, Lpt, Li,Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1)

```
303  evolue((B,not(C)), Lie, Lpt, Li, Lu, Ls, Lie, Lpt, Li, Lu,[(B,not(C))| Ls]) :-
304      setof(C, cnamena(C),L), member(C,L),!.
305  evolue((B,not(C)), Lie, Lpt, Li, Lu, Ls, Lie, Lpt, Li, Lu,[(B,not(C))| Ls]) :-
306      setof(C, cnamea(C),L), member(C,L) ,!.
307  evolue((B,C), Lie, Lpt, Li, Lu, Ls, Lie, Lpt, Li, Lu,[(B,C)| Ls]) :-
308      setof(C, cnamea(C),L), member(C,L),!.
309  evolue((B,C), Lie, Lpt, Li, Lu, Ls, Lie, Lpt, Li, Lu,[(B,C)| Ls]) :-
310      setof(C, cnamena(C),L), member(C,L) ,!.
311  evolue((I,some(R,C)), Lie, Lpt, Li, Lu, Ls, [(I,some(R,C))|Lie], Lpt, Li, Lu, Ls):-!.
312  evolue((I,and(C1,C2)), Lie, Lpt,Li, Lu, Ls, Lie, Lpt,  [(I,and(C1,C2))|Li], Lu, Ls):-!.
313  evolue((I,all(R,C)), Lie, Lpt, Li, Lu, Ls, Lie, [(I,all(R,C))|Lpt], Li, Lu, Ls):-!.
314  evolue((I,or(C1,C2)), Lie, Lpt, Li,Lu, Ls, Lie, Lpt, Li, [(I , or(C1,C2))| Lu], Ls):-!.
```

Le prédicat évolue permet d'associer une assertion de concept dans la bonne liste de la Abox. Ce prédicat n'est pas récursif et on doit juste vérifier la facon dont C est écrit. Les règles 311 à 314 sont assez explicites, en revanche pour les lignes 303 à 310 on doit vérifier si C appartient bien à la liste des concept(atomique ou complexe) et on doit prendre en compte la négation de concept avant de l'inserer à la liste correspondante.

Serie de test sur evolue((I,C), Lie, Lpt, Li,Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1) :

```
?- evolue((david,not(personne)),[],[],[],[],[],Lie1,Lpt1,Li1,Lu1,Ls1).
Lie1 = Lpt1, Lpt1 = Li1, Li1 = Lu1, Lu1 = [],
Ls1 = [(david, not(personne))].

?- evolue((david,or(not(personne),livre)),[],[],[],[],[],Lie1,Lpt1,Li1,Lu1,Ls1).
Lie1 = Lpt1, Lpt1 = Li1, Li1 = Ls1, Ls1 = [],
Lu1 = [(david, or(not(personne), livre))].

?- evolue((david,and(not(personne),livre)),[],[],[],[],[],Lie1,Lpt1,Li1,Lu1,Ls1).
Lie1 = Lpt1, Lpt1 = Lu1, Lu1 = Ls1, Ls1 = [],
Li1 = [(david, and(not(personne), livre))].

?- evolue((david,all(aEcrit,and(not(personne),livre))),[],[],[],[],[],Lie1,Lpt1,Li1,Lu1,Ls1).
Lie1 = Li1, Li1 = Lu1, Lu1 = Ls1, Ls1 = [],
Lpt1 = [(david, all(aEcrit, and(not(personne), livre)))].

?- evolue((david,some(aEcrit,and(not(personne),livre))),[],[],[],[],[],Lie1,Lpt1,Li1,Lu1,Ls1).
Lie1 = [(david, some(aEcrit, and(not(personne), livre)))],
Lpt1 = Li1, Li1 = Lu1, Lu1 = Ls1, Ls1 = [].
```

Application de la règle ∃ : Le prédicat complete_some(Lie,Lpt,Li,Lu,Ls,Abr)

```
225  complete_some([],Lpt,Li,Lu,Ls,Abr).
226  complete_some([(A,some(R,C))|Lie],Lpt,Li,Lu,Ls,Abr):-
227      genere(B),
228      evolue((B,C), Lie, Lpt, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1),
229      affiche_evolution_Abox(Ls, Lie, Lpt, Li, Lu, Abr, Ls1, Lie1, Lpt1, Li1, Lu1, Abr),
230      concat([(A,B,R)],Abr,Abr),
231      verif_contradiction(Ls1),
232      resolution(Lie1, Lpt1,Li1,Lu1,Ls1,Abr).
```

Cette règle est la première règle à appliquer lors de la résolution. Comme rappele à la page 10 de l'énoncé du projet, nous générer une instanciation de concept ou B sera l'instance générer comme on peut le voir ligne 227. Nous appliquerons ensuite le prédicat **evolue** (ligne 228) qui permet d'inserer ce nouvel objet dans la liste correspondante, soit Ls. Comme nous savons que ce nouvel élément sera ajouter dans Ls , on peut vérifier uniquement les clashs dans la nouvelle liste générer par **evolue** , c'est à dire Ls1 (Ligne 231). Nous devons aussi ajouter à la liste des assertions de rôle : $< A, B >: R$, ligne 230. Enfin on rapelle resolution avec les listes modifiés. Remarque : on applique un prédicat affiche qui permettra de suivre les traces de l'execution et que l'on présentera en fin de rapport.

Serie de test sur complete_some(Lie,Lpt,Li,Lu,Ls,Abr) :

```
[7]  ?- complete_some([(david,some(aCree,livre))] , [] , [] , [] , [(david,objet)] , []).



---------------------
Avant modification :
---------------------
david : objet
---------------------
Apres modification :
---------------------
inst1 : livre
david : objet
true .

[7]  ?- complete_some([(david,some(aCree,livre))] , [] , [] , [] , [(david,objet) , (david,not(livre))] , []).



---------------------
Avant modification :
---------------------
david : objet
david :  ¬livre
---------------------
Apres modification :
---------------------
inst2 : livre
david : objet
david :  ¬livre
```

Application de la règle ∀ : Le prédicat deduction_all(Lie,Lpt,Li,Lu,Ls,Abr)

```
259  deduction_all(_,[],_,_,_,_).
260  deduction_all(Lie,[(A,all(R,C))|Lpt],Li,Lu,Ls, Abr) :-
261      member((A,B,R),Abr),
262      evolue((B,C), Lie, Lpt, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1),
263      affiche_evolution_Abox(Ls, Lie, Lpt, Li, Lu, Abr, Ls1, Lie1, Lpt1, Li1, Lu1, Abr),
264      verif_contradiction(Ls1),
265      resolution(Lie1, Lpt1, Li1, Lu1, Ls1,Abr).
266
267  deduction_all(Lie,[(A,all(R,C))|Lpt],Li,Lu,Ls, Abr) :-
268      affiche_evolution_Abox(Ls, Lie, Lpt, Li, Lu, Abr, Ls1, Lie1, Lpt1, Li1, Lu1, Abr),
269      resolution(Lie, Lpt, Li, Lu, Ls,Abr).
```

Cette règle est divisé en deux car il y a un cas ou l'assertion de rôle $< A, B >: R$ appartient à la liste des assertions de role Abr et un cas ou elle n'y est pas. Le second cas est plus facile car on a juste a ignoré la règle est renvoyé vrai pour pas induire en erreur notre résolution (Ligne 267 a 269). Pour le premier cas nous appliquons la règle qui ajoute $B : C$ à la liste Ls (Ligne 262). Ensuite nous avons juste à faire un test que les clashs (Ligne 264)et l'appel recursif(Ligne 265).

Serie de test sur deduction_all(Lie,Lpt,Li,Lu,Ls,Abr) :

```
[1]  ?- deduction_all([] , [(michelAnge,all(aEcrit,personne))] , [] , [] , [(david,objet) , (david,not(livre))] , [(michelAnge, david
 aCree), (michelAnge, sonnet, aEcrit),(vinci, joconde, aCree)]).
---------------------
Avant modification :
---------------------
david : objet
david :  ¬livre
---------------------
Apres modification :
---------------------
sonnet : personne
david : objet
david :  ¬livre
true .
```

```
[2]  ?- deduction_all([] , [(michelAnge,all(aEcrit,personne))] , [] , [] , [(david,objet) , (david,not(livre)) , (sonnet,not(personne)
)] , [(michelAnge, david, aCree), (michelAnge, sonnet, aEcrit),(vinci, joconde, aCree)]).
---------------------
Avant modification :
---------------------
david : objet
david :  ¬livre
sonnet :  ¬personne
---------------------
Apres modification :
---------------------
sonnet : personne
david : objet
david :  ¬livre
sonnet :  ¬personne
false.
```

Application de la règle ⊓ : Le prédicat transformation_and(Lie,Lpt,Li,Lu,Ls,Abr)

```
293  transformation_and(_,_,[],_,_,_).
294
295  transformation_and(Lie, Lpt,[(I,(and(C1,C2)))|Li],Lu, Ls,Abr) :-
296      evolue((I,C1), Lie, Lpt, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1),
297      affiche_evolution_Abox(Ls, Lie, Lpt, Li, Lu, Abr, Ls1, Lie1, Lpt1, Li1, Lu1, Abr),
298      verif_contradiction(Ls1),
299      evolue((I,C2), Lie1, Lpt1, Li1, Lu1, Ls1, Lie2, Lpt2, Li2, Lu2, Ls2),
300      affiche_evolution_Abox(Ls1, Lie1, Lpt1, Li1, Lu1, Abr, Ls2, Lie2, Lpt2, Li2, Lu2, Abr),
301      verif_contradiction(Ls2),
302      resolution(Lie2, Lpt2, Li2, Lu2, Ls2,Abr).
```

Cette règle consiste à ajouter $I : C_1$ et $I : C_2$ à la liste Ls. Nous allons donc faire cela aux lignes 296 et 299. Puis suivre ces opérations des vérifications de clash et d'affichage puir suivre l'évolution. Nous mettons tous dans une règle car la règle ⊓ ne génére qu'un seul noeud et non deux. Donc si un clash est déclanché pour $I : C_1$ ou $I : C_2$, c'est toute la règle qui est fausse. C'est la représentation même de la méthode des tableaux!

Serie de test sur transformation_and(Lie,Lpt,Li,Lu,Ls,Abr) :

```
[2]  ?- transformation_and([], [],[(david,and(objet,livre))],[], [(michelAnge,objet)] ,[]).
----------------------
Avant modification :
----------------------
michelAnge : objet
----------------------
Apres modification :
----------------------
david : objet
michelAnge : objet
----------------------
Avant modification :
----------------------
david : objet
michelAnge : objet
----------------------
Apres modification :
----------------------
david : livre
david : objet
michelAnge : objet
true.

[2]  ?- transformation_and([], [],[(david,and(objet,livre))],[], [(david,not(objet))] ,[]).
----------------------
Avant modification :
----------------------
david :  ¬objet
----------------------
Apres modification :
----------------------
david : objet
david :  ¬objet
false.
```

Application de la règle ⊔ : Le prédicat transformation_or(Lie,Lpt,Li,Lu,Ls,Abr)

```
288  transformation_or(_,_,_,[],_,_).
289  transformation_or(Lie,Lpt,Li,[(I,or(C1,C2))|Lu],Ls,Abr) :-
290      evolue((I,C1), Lie, Lpt, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1),
291      affiche_evolution_Abox(Ls, Lie, Lpt, Li, Lu, Abr, Ls1, Lie1, Lpt1, Li1, Lu1, Abr1),
292      verif_contradiction(Ls1),
293      resolution(Lie1, Lpt1, Li1, Lu1, Ls1,Abr).
294
295  transformation_or(Lie,Lpt,Li,[(I,or(C1,C2))|Lu],Ls,Abr) :-
296      evolue((I,C2), Lie, Lpt, Li, Lu, Ls, Lie2, Lpt2, Li2, Lu2, Ls2),
297      affiche_evolution_Abox(Ls, Lie, Lpt, Li, Lu, Abr, Ls2, Lie2, Lpt2, Li2, Lu2, Abr2),
298      verif_contradiction(Ls2),
299      resolution(Lie2, Lpt2, Li2, Lu2, Ls2,Abr).
```

Cette règle fait l'analogie avec la règle ⊓ expliqué plus haut. Sauf que nous allons générer deux noeuds au lieu d'un. Nous faisons ca dédoublant la règle ⊓ en deux sous prédicats et en ne générant qu'une seule instance

$I : C$ par noeud. Ainsi, si une règle échoue l'autre sera alors vérifier en prenant soit $I : C_1$ soit $I : C_2$ pour vérifier la présence ou non d'un clash.

Serie de test sur transformation_or(Lie,Lpt,Li,Lu,Ls,Abr :

```
[2]  ?- transformation_or([], [],[] ,[(david,or(objet,livre))], [(michelAnge,not(objet))] ,[]).
----------------------
Avant modification :
----------------------
michelAnge :  ¬objet
----------------------
Apres modification :
----------------------
david : objet
michelAnge :  ¬objet
true .

[2]  ?- transformation_or([], [],[] ,[(david,or(objet,livre))], [(david,not(objet))] ,[]).
----------------------
Avant modification :
----------------------
david :  ¬objet
----------------------
Apres modification :
----------------------
david : objet
david :  ¬objet
----------------------
Avant modification :
----------------------
david :  ¬objet
----------------------
Apres modification :
----------------------
david : livre
david :  ¬objet
true .
```

```
[2]  ?- transformation_or([], [],[] ,[(david,or(objet,livre))], [(david,not(objet)) , (david,not(livre))] ,[]).
----------------------
Avant modification :
----------------------
david :  ¬objet
david :  ¬livre
----------------------
Apres modification :
----------------------
david : objet
david :  ¬objet
david :  ¬livre
----------------------
Avant modification :
----------------------
david :  ¬objet
david :  ¬livre
----------------------
Apres modification :
----------------------
david : livre
david :  ¬objet
david :  ¬livre
false.
```

Affichage de l'évolution de la ABox : Le prédicat : affiche_evolution_Abox(Ls1, Lie1, Lpt1, Li1, Lu1, Abr1, Ls2, Lie2, Lpt2, Li2, Lu2, Abr2)

```
406  /* Affichage de l'évolution des noeuds de l'arbre*/
407
408  /*
409      Affichage recursif sur les listes d'instanciations de concepts sur un avant et un apres.
410      Permet de comprendre et debugger un arbre de regle sur une proposition.
411      principe : recursion sur les elements dune liste puis sur les composants d'une formule
412  */
413
414  affiche_evolution_Abox(Ls1, Lie1, Lpt1, Li1, Lu1, Abr1, Ls2, Lie2, Lpt2, Li2, Lu2, Abr2) :-
415      write("----------------------"),nl,
416      write("Avant modification : "),nl,
417      write("----------------------"),nl,
418       write("Assertions de roles (<a,b> : R) : "),nl,
419      affiche_listeAbr(Abr1),nl,
420      write("Assertions de concept (I : C) : "),nl,
421      affiche_liste(Ls1),
422      affiche_liste(Lie1),
423      affiche_liste(Lpt1),
424      affiche_liste(Li1),
425      affiche_liste(Lu1),
426      write("----------------------"),nl,
427      write("Apres modification : "),nl,
428      write("----------------------"),nl,
429      write("Assertions de roles (<a,b> : R) : "),nl,
430      affiche_listeAbr(Abr2),nl,
431      write("Assertions de concept (I : C) : "),nl,
432      affiche_liste(Ls2),
433      affiche_liste(Lie2),
434      affiche_liste(Lpt2),
435      affiche_liste(Li2),
436      affiche_liste(Lu2),!.
437
438  affiche_liste([]).
439  affiche_liste([(I , F)| L2]) :-
440      write(I),write(" : "),affiche_formule(F),nl,
441      affiche_liste(L2),!.
442
443  affiche_listeAbr([]).
444  affiche_listeAbr([(A , B , R)| L2]) :-
445      write("<"),write(A),write(","),write(B),write("> : "),write(R),nl,
446      affiche_listeAbr(L2),!.
447
448  affiche_formule(F) :-setof(C, cnamena(C),L), member(F,L),write(F).
449  affiche_formule(F) :-setof(C, cnamea(C),L), member(F,L),write(F).
450  affiche_formule(or(F1,F2)) :- write("(") , affiche_formule(F1) , write(") ⊔ (") , affiche_formule(F2), write(")").
451  affiche_formule(and(F1,F2)) :- write("(") , affiche_formule(F1) , write(") ⊓ (") , affiche_formule(F2), write(")").
452  affiche_formule(all(R,F)) :- write("∀."),write(R) , write("(") , affiche_formule(F),write(")").
453  affiche_formule(some(R,F)) :- write("∃."),write(R) , write("(") , affiche_formule(F),write(")").
454  affiche_formule(not(F)) :- write(" ¬"),affiche_formule(F).
```

Ce prédicat permet de pouvoir suivre l'évolution de la résolution d'une proposition à chaque noeud. Le point d'entrée de ce prédicat est **affiche_evolution_Abox(Ls1, Lie1, Lpt1, Li1, Lu1, Abr1, Ls2, Lie2, Lpt2, Li2, Lu2, Abr2)**. Ce prédicat va déléguer son travail à **afficher_liste(L)** qui va itérer sur les différents éléments de la liste. Il va lui-même déléguer son travail d'affichage de formule à **affiche_formule(F)** qui va pouvoir afficher convenablement la formule en fonction des connecteurs et quantificateurs rencontrés.

Ce prédicat est testé dans chacun des prédicats de la partie 3 de ce rapport.

Serie de test sur programme :

Voici ci-dessous une série de test finale qui présentera la méthode de démonstration de notre programme sur les deux types de proposition.

Test de proposition de type I

Exemple 1 : michelAnge : auteur
Resultat attendu : True

```
[1]  ?- programme.

Entrez le numero du type de proposition que vous voulez demontrer :
1 : Une instance donnee appartient a un concept donne. (I : C)
2 : Deux concepts ne possede aucun elements en commun(ils ont une intersection vide => C1 ⊓ C2 ⊑ ⊥ ) :
3 : Pour sortir de l'invité de commande
|: 1.

Entrer linstance I :
|: michelAnge.

Entrer le concept C :
|: auteur.
----------------------
Avant modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
michelAnge :  ¬personne
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Avant modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
michelAnge : ∀.aEcrit( ¬livre)
----------------------
Avant modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
sonnets :  ¬livre
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne

Youpiiiiii, on a demontre la proposition initiale !!!
true.
```

Exemple 2 : michelAnge : sculpteur
Resultat attendu : True

```
[1]  ?- programme.

Entrez le numero du type de proposition que vous voulez demontrer :
1 : Une instance donnee appartient a un concept donne. (I : C)
2 : Deux concepts ne possede aucun elements en commun(ils ont une intersection vide => C1 ⊓ C2 ⊑ ⊥ ) :
3 : Pour sortir de l'invité de commande
|: 1.

Entrer linstance I :
|: michelAnge.

Entrer le concept C :
|: sculpteur.
----------------------
Avant modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
michelAnge :  ¬personne
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Avant modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
michelAnge : ∀.aCree( ¬sculpture)
----------------------
Avant modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
david :  ¬sculpture
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne

Youpiiiiii, on a demontre la proposition initiale !!!
true.
```

Exemple 3 : vinci : auteur
Resultat attendu : False

```
[1]  ?- programme.

Entrez le numero du type de proposition que vous voulez demontrer :
1 : Une instance donnee appartient a un concept donne. (I : C)
2 : Deux concepts ne possede aucun elements en commun(ils ont une intersection vide => C1 ⊓ C2 ⊑ ⊥ ) :
3 : Pour sortir de l'invité de commande
|: 1.

Entrer linstance I :
|: vinci.

Entrer le concept C :
|: auteur.
----------------------
Avant modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
vinci :  ¬personne
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Avant modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
vinci : ∀.aEcrit( ¬livre)
----------------------
Avant modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
false.
```

Exemple 4 : michelAnge : editeur
Resultat attendu : False

```
[1]  ?- programme.

Entrez le numero du type de proposition que vous voulez demontrer :
1 : Une instance donnee appartient a un concept donne. (I : C)
2 : Deux concepts ne possede aucun elements en commun(ils ont une intersection vide => C1 ⊓ C2 ⊑ ⊥ ) :
3 : Pour sortir de l'invité de commande
|: 1.

Entrer linstance I :
|: michelAnge.

Entrer le concept C :
|: editeur.
```

On a du tronqué la capture pour cause d'affichage.

```
----------------------
Avant modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
inst2 : livre
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
false.
```

Test de proposition de type II

Exemple 5 : objet $\sqcap$ livre $\sqsubseteq \bot$
Resultat attendu : True

```
[1]  ?- programme.

Entrez le numero du type de proposition que vous voulez demontrer :
1 : Une instance donnee appartient a un concept donne. (I : C)
2 : Deux concepts ne possede aucun elements en commun(ils ont une intersection vide => C1 ⊓ C2 ⊑ ⊥ ) :
3 : Pour sortir de l'invité de commande
|: 2.

Entrer le premier concept de lintersection :
|: objet.

Entrer le deuxieme concept de lintersection :
|: livre.



----------------------
Avant modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
inst3 :  ¬objet
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Avant modification :
----------------------
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
inst3 :  ¬livre
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne

Youpiiiiii, on a demontre la proposition initiale !!!
true.
```

Exemple 6 : sculpteur ⊓ auteur ⊑ ⊥
Resultat attendu : False

```
?- programme.

Entrez le numero du type de proposition que vous voulez demontrer :
1 : Une instance donnee appartient a un concept donne. (I : C)
2 : Deux concepts ne possede aucun elements en commun(ils ont une intersection vide => C1 ⊓ C2 ⊑ ⊥ ) :
3 : Pour sortir du programme
|: 2.

Entrer le premier concept de lintersection :
|: sculpteur.

Entrer le deuxieme concept de lintersection :
|: auteur.



----------------------
Avant modification :
----------------------
Assertions de roles (<a,b> : R) :
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

Assertions de concept (I : C) :
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
Assertions de roles (<a,b> : R) :

Assertions de concept (I : C) :
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
inst1 : ( ¬personne) ⊔ (∀.aCree( ¬sculpture))
----------------------
Avant modification :
----------------------
Assertions de roles (<a,b> : R) :
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

Assertions de concept (I : C) :
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
Assertions de roles (<a,b> : R) :

Assertions de concept (I : C) :
inst1 :  ¬personne
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
```

On a du tronqué la capture pour cause d'affichage.

```
----------------------
Avant modification :
----------------------
Assertions de roles (<a,b> : R) :
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

Assertions de concept (I : C) :
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
Assertions de roles (<a,b> : R) :

Assertions de concept (I : C) :
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
inst1 : ∀.aCree( ¬sculpture)
----------------------
Avant modification :
----------------------
Assertions de roles (<a,b> : R) :
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

Assertions de concept (I : C) :
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
----------------------
Apres modification :
----------------------
Assertions de roles (<a,b> : R) :
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

Assertions de concept (I : C) :
david : sculpture
joconde : objet
michelAnge : personne
sonnets : livre
vinci : personne
false.
```

Conclusion

Ce projet nous a permis de pouvoir nous familiariser avec le langage Prolog à travers la construction de ce démonstrateur. Nous avons pu mieux assimilier les notions de cours qui abordent la méthode des tableaux sur

la logique de description ALC. Le code fourni (**projetFinal.pl**) avec ce rapport contient aussi des explications sur le fonctionnement des prédicats.