

Sorbonne Université

Master DAC

Projet LRC

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

Réalisé par  
Samy NEHLIL  
Allaa BOUTALEB

Groupe 01

Paris, France  
Décembre 2022

# TABLE DES MATIÈRES

<b>1 Partie I : Préliminaires</b>	<b>6</b>
1.1 Prédicat concept(C) . . . . .	7
1.2 Prédicat replace_concept_na(C1,C2) . . . . .	7
1.3 Prédicat transform(C,NC) . . . . .	8
1.4 Prédicat retrieve_na(C1,C2, Tbox) . . . . .	9
1.5 Prédicat pas_autoref(C,E, Tbox) . . . . .	9
1.6 Prédicat traitement_Tbox(C,E, Tbox) . . . . .	11
1.7 Prédicat traitement_Abox(abi, Abi1, Abr, Abr1, Tbo) . . . . .	11
<b>2 Partie II : Saisie de la proposition à démontrer</b>	<b>13</b>
2.1 Prédicat acquisition_prop_type1(abi,Abi1,Tbox) . . . . .	13
2.2 Prédicat acquisition_prop_type2(abi,Abi1,Tbox) . . . . .	14
2.3 Exemple - Acquisition de proposition . . . . .	16
<b>3 Partie III : Démonstration de la proposition</b>	<b>17</b>
3.1 Prédicat tri_Abox . . . . .	17
3.2 Prédicat test_clash(Ls) . . . . .	18
3.3 Prédicat resolution . . . . .	19
3.4 Prédicat complete_some(Lie,Lpt,Li,Lu,Ls,Abr) . . . . .	19
3.5 Prédicat transformation_and(Lie,Lpt,Li,Lu,Ls,Abr) . . . . .	20
3.6 Prédicat deduction_all(Lie,Lpt,Li,Lu,Ls,Abr) . . . . .	20
3.7 Prédicat transformation_or(Lie,Lpt,Li,Lu,Ls,Abr) . . . . .	20
3.8 Prédicat evolve(A, Lie, Lpt, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1) . . . . .	21
3.9 Prédicat affiche_evolution_Abox(Ls1, Lie1, Lpt1, Li1, Lu1, Abr1, Ls2, Lie2, Lpt2, Li2, Lu2, Abr2) . . . . .	21
3.10 Démonstration de notre programme . . . . .	23

## TABLE DES FIGURES

1.1	concept	7
1.2	replace_concept_na	8
1.3	transform	8
1.4	retrieve_na	9
1.5	retrieve_na	10
1.6	traitement_Tbox	11
1.7	traitement_Abox	12
2.1	Premiere partie	14
2.2	Deuxième partie	15
2.3	Exemple d'acquisition de props	16
3.1	tri_abox	18
3.2	test_clash	18
3.3	resolution	19
3.4	complete_some	19
3.5	transformation_and	20
3.6	deduction_all	20
3.7	transformation_or	20
3.8	evolute	21
3.9	affiche_evolution_Abox	22
3.10	Proposition de type 1	23
3.11	Proposition de type 2	24
3.12	Proposition de type 2	25
3.13	Proposition de type 2	26

# INTRODUCTION

Ce projet s'inscrit dans le cadre du module de logique et représentation des connaissances LRC. Nous devons réaliser en langage Prolog un démonstrateur basé sur l'algorithme des tableaux pour la logique de description *ALC*. Ce document constitue le compte rendu du projet, il contient une description des prédictats utilisés avec leurs codes respectifs ainsi qu'un jeu de test de notre démonstrateur.

# CHAPTER 1

## PARTIE I : PRÉLIMINAIRES

Dans cette partie, nous allons vérifier la correction syntaxique et sémantique d'une Tbox et d'une Abox et de les mettre en forme, afin de nous en servir comme bases de tests pour le démonstrateur à réaliser, lorsque nous lui fournirons une nouvelle proposition à démontrer.

La partie 01 du projet consiste en la préparation à la démonstration qui est le but final du programme.

Afin de rassembler les informations liées à la TBox et à la ABox, on crée 03 listes allant contenir respectivement les instances (Abi), les rôles (Abr) et les équivalences (equiv). Ces dernières seront amenées à évoluer au fur et à mesure que l'on soumettra des propositions à la démonstration.

Ces listes sont créées en utilisant le prédicat *setof* allant rassembler toutes les instances d'une certaine forme au sein d'une liste dont on passe le nom en paramètre.

Par exemple, ***setof((X), chien(X), nomsChien)*** renvoie la liste contenant tous les noms de chiens, en considérant que le prédicat ***chien(X)*** représente le fait que l'instance X soit un chien.

On utilise donc le prédicat ***premiere\_etape(Tbox, Abi, Abr)*** allant renvoyer les différentes listes demandées.

## 1.1 Prédicat concept(C)

concept/1 : vérifie que C est un concept valide (sémantiquement et syntaxiquement) par rapport à la définition des concepts donnée dans la Tbox. Ce prédicats traite tout les cas possibles d'un concept comme rappelé dans le cahier des charges du projet.

Code:

```
221
222  concept(C) :- cnamea(C),!.
223  concept(C) :- cnamena(C),!.
224  concept(not(C)) :- concept(C),!.
225  concept(and(C1,C2)) :- concept(C1), concept(C2),!.
226  concept(or(C1,C2)) :- concept(C1), concept(C2),!.
227  concept(some(R,C)) :- rname(R), concept(C),!.
228  concept(all(R,C)) :- rname(R), concept(C),!.
229
```

Figure 1.1: concept

Jeu de tests :

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

```
?- concept(animal).  
false.
```

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

```
?- concept(or(some(aCree,sculpture),and(parent,all(aEcrit,livre)))).  
true.
```

## 1.2 Prédicat replace\_concept\_na(C1,C2)

replace\_concept\_na/2 : définit les règles de remplacement des concepts non atomiques par leur définition équivalente dans la Tbox. Autrement dit, met dans C2 la définition constituée uniquement par des concepts atomiques du concept C1. De même pour ce prédicat, on doit traîter tout les cas possibles de la définition d'un concept (all, some, and, or).

**Code:**

```
236 % Règles de remplacement des concepts non atomiques par leur définition (replace_concept_na)
237 replace_concept_na([not(C)|_],not(Ctraite)) :- replace_concept_na([C],Ctraite).
238 replace_concept_na([and(C1,C2)|_],and(C1traite,C2traite)) :- replace_concept_na([C1],C1traite), replace_concept_na([C2],C2traite), !.
239 replace_concept_na([or(C1,C2)|_],or(C1traite,C2traite)) :- replace_concept_na([C1],C1traite), replace_concept_na([C2],C2traite), !.
240 replace_concept_na([some(R,C)|_],some(R,Ctraite)) :- replace_concept_na([C],Ctraite), !.
241 replace_concept_na([all(R,C)|_],all(R,Ctraite)) :- replace_concept_na([C],Ctraite), !.
242 replace_concept_na([cNamea(C)|_],cNamea(C)) :- !.
243 replace_concept_na([equiv(C)|_],equiv(C,Ctraite)) :- !.
244
```

Figure 1.2: replace\_concept\_na

**Jeu de tests :**

```
?- replace_concept_na([objet],R).
```

R = objet.

```
?- replace_concept_na([auteur],R).
```

R = and(personne, some(aEcrit, livre)).

```
?- replace_concept_na([or(some(aEnfant,auteur),sculpteur)],R).
```

R = or(some(aEnfant, and(personne, some(aEcrit, livre))), and(personne, some(aCree, sculpture))).

### 1.3 Prédicat transform(C,NC)

transform/2 : remplace les concepts non atomiques par leur définition dans la Tbox puis met l'expression sous sa forme NNF. Utilise les deux prédicats replace\_concept\_na et nnf.

**Code:**

```
245 % _____
246 % remplace les concepts non atomiques par leur définition, met l'expression sous nnf
247 transform(C, NC):-_
248     replace_concept_na(C, CA),
249     nnf(CA, NC),!.
```

Figure 1.3: transform

**Jeu de tests :**

```
?- transform([auteur],R).
```

R = and(personne, some(aEcrit, livre)).

```
?- transform([or(some(aEnfant,auteur),sculpteur)],R).
```

R = or(some(aEnfant, and(personne, some(aEcrit, livre))), and(personne, some(aCree, sculpture))).

## 1.4 Prédicat retrieve\_na(C1,C2, Tbox)

retrieve\_na/3 : retourne dans C2 le concept non atomique utilisé dans la définition de C1 conformément à la définition de concepts données dans la Tbox, ce prédicat sera utile dans la définition du prédicat pas\_autoref.

**Code:**

```
281 % Prédicat qui retourne dans Y le concept non atomique utilisé dans la définition de X
282 % utile dans pas_autoref
283 retrieve_na(X, X, _) :- 
284     cnamea(X).
285 retrieve_na(X, D, [(X, D)|_]) .
286 retrieve_na(X, Y, [(C, _)|L]) :- 
287     X \= C,
288     retrieve_na(X, Y, L).
```

Figure 1.4: retrieve\_na

## 1.5 Prédicat pas\_autoref(C,E, Tbox)

pas\_autoref/3 : Permet de s'assurer que les définitions de concepts données pour la Tbox ne sont pas circulaires, c'est-à-dire auto-référentes, sinon nous risquons le bouclage dans le traitement des expressions de concepts.

**Code:**

**Jeu de tests :**

```
Tbox = [(auteur, and(personne, some(aEcrit, livre))),  

(editeur, and(personne, and(not(some(aEcrit, livre)), some(aEdite, livre)))),  

(parent, and(personne, some(aEnfant, anything))),  

(sculpteur, and(personne, some(aCree, sculpture))),  

(sculpture, and(objet, all(cree_par, sculpteur)))]  

Abi = [(david, sculpture), (joconde, objet), (michelAnge, personne),  

(socrate, personne), (sonnets, livre), (vinci, personne)]  

Abr = [(michelAnge, david, aCree), (michelAnge, sonnets, aEcrit), (vinci, joconde, aCree)]
```

false.

```
Tbox = [(auteur, and(personne, some(aEcrit, livre))),  

(editeur, and(personne, and(not(some(aEcrit, livre)), some(aEdite, livre)))),  

(parent, and(personne, some(aEnfant, anything))),  

(sculpteur, and(personne, some(aCree, sculpture)))]
```

```
Abi = [(david, sculpture), (joconde, objet), (michelAnge, personne),  

(socrate, personne), (sonnets, livre), (vinci, personne)]  

Abr = [(michelAnge, david, aCree), (michelAnge, sonnets, aEcrit), (vinci, joconde, aCree)]
```

Traitement de la Tbox réalisé avec succès

```

289 | % pas_autoref/3 : verifie si la Tbox ne contient pas des concepts autoreferences
290 | pas_autoref(_, E, _) :- 
291 |   % Si prédicat utilisé dans la définition est atomique, retourne vrai car cycle impossible
292 |   cnamena(E), !.
293 | pas_autoref(C, E, Tbox) :- 
294 |   % Si prédicat E utilisé non atomique, vérifier qu il contient pas C
295 |   cnamena(E),
296 |   retrieve_na(E, D, Tbox),
297 |   pas_autoref(C, D, Tbox).
298 | % Différents cas de concepts
299 | pas_autoref(C, not(C1), Tbox) :- 
300 |   C \= C1,
301 |   pas_autoref(C, C1, Tbox).
302 | pas_autoref(C, and(C1, C2), Tbox) :- 
303 |   C \= C1,
304 |   C \= C2,
305 |   pas_autoref(C, C1, Tbox),
306 |   pas_autoref(C, C2, Tbox).
307 | pas_autoref(C, or(C1, C2), Tbox) :- 
308 |   C \= C1,
309 |   C \= C2,
310 |   pas_autoref(C, C1, Tbox),
311 |   pas_autoref(C, C2, Tbox).
312 | pas_autoref(C, some(_, C1), Tbox) :- 
313 |   C \= C1,
314 |   pas_autoref(C, C1, Tbox).
315 | pas_autoref(C, all(_, C1), Tbox) :- 
316 |   C \= C1,
317 |   pas_autoref(C, C1, Tbox).
318 |
319

```

Figure 1.5: retrieve\_na

Traitement Abi avec succes

Traitement Abr avec succes

Traitement de la Abox realisé avec succes

Traitement de la Tbox et de la Abox realisé avec succes

FIN DE LA PREMIERE PARTIE AVEC SUCCES

## 1.6 Prédicat traitement\_Tbox(C,E, Tbox)

traitement\_Tbox/3 : Permet de vérifier la cohérence d'une Tbox. Une fois vérifiée la correction syntaxique et sémantique d'une expression conceptuelle équivalente à un concept complexe défini dans la Tbox, cette expression est remplacée par une expression où ne figurent plus que des identificateurs de concepts atomiques et qui a été mise sous forme normale négative.

Ce prédicat réalise un certain nombre de traitements sur la Tbox que nous allons résumer dans les points suivants :

1. Pour chaque couple (C,E) présent dans la Tbox, C n'est pas un concept atomique et E doit être un concept.
2. On vérifie que le couple (C,E) n'est pas autoréférencé.
3. On remplace E par sa NNF et on met à jour la Tbox.

```
251
252  traitement_Tbox([], [], _).
253  traitement_Tbox([(C, E)|L], [(C, E1)|L1], Tbox) :- 
254      cnamen(C),
255      concept(E),
256      pas_autoref(C, E, Tbox),
257      transform([E], E1),
258      traitement_Tbox(L, L1, Tbox), !.
```

Figure 1.6: traitement\_Tbox

## 1.7 Prédicat traitement\_Abox(Abi, Abi1, Abr, Abr1, Tbo)

traitement\_Abox/5 : Permet de vérifier la cohérence d'une Abox. Une fois vérifiée la correction syntaxique et sémantique de l'expression conceptuelle d'appartenance d'une instance de la Abox, il s'agit de remplacer tous les identificateurs de concepts complexes d'une telle expression conceptuelle par leurs définitions respectives ne comportant que des concepts atomiques (que l'on peut récupérer dans la Tbox simplifiée que l'on vient de générer). Cette expression est enfin mise sous forme normale négative.

Ce prédicat réalise un certain nombre de traitements sur la Abox que nous allons résumer dans les points suivants :

1. traitement\_Abr : traite les assertions de type (I1,I2,R).
2. traitement\_Abi : traite les assertions de type (I,C).

```

259 |     traitement_Abi([], [], _).
260  traitement_Abi([(I, C)|L], [(I, C1)|L1], Tbox) :- 
261    concept(C),
262    transform([C], C1),
263    traitement_Abi(L, L1, Tbox), !.
264
265
266  traitement_Abr([], [], _).
267  traitement_Abr([(I1, I2, R)|L], [(I1, I2, R)|L1], Tbox) :- 
268    iname(I1),
269    iname(I2),
270    rname(R),
271    traitement_Abr(L, L1, Tbox), !.
272
273  traitement_Abox(Abi, Abi1, Abr, Abr1, Tbox) :- 
274    traitement_Abi(Abi, Abi1, Tbox),
275    nl, write("Traitement Abi avec succes"), nl,
276    traitement_Abr(Abr, Abr1, Tbox),
277    nl, write("Traitement Abr avec succes").
278

```

Figure 1.7: traitement\_Abox

### *Exemple d'exécution de la première partie du programme*

Tbox = [(auteur, and(personne, some(aEcrit, livre))),  
 (editeur, and(personne, and(not(some(aEcrit, livre)), some(aEdite, livre)))),  
 (parent, and(personne, some(aEnfant, anything))),  
 (sculpteur, and(personne, some(aCree, sculpture)))]

Abi = [(david, sculpture), (joconde, objet), (michelAnge, personne), (socrate, personne),  
 (sonnets, livre), (vinci, personne)]

Abr = [(michelAnge, david, aCree), (michelAnge, sonnets, aEcrit), (vinci, joconde, aCree)]

Traitement de la Tbox realise avec succes

Traitement Abi avec succes

Traitement Abr avec succes

Traitement de la Abox realise avec succes

Traitement de la Tbox et de la Abox realise avec succes

FIN DE LA PREMIERE PARTIE AVEC SUCCES

# CHAPTER 2

## PARTIE II : SAISIE DE LA PROPOSITION À DÉMONTRER

Cette partie a pour but d'acquérir la proposition à prouver à l'aide du démonstrateur. L'utilisateur est donc invité à choisir le type de la proposition qu'il souhaite démontrer ainsi que les concepts et les instances.

### 2.1 Prédicat acquisition\_prop\_type1(Abi,Abi1,Tbox)

acquisition\_prop\_type1/3 : Permet à l'utilisateur de notre programme d'entrer une proposition de type 1 ie L'instance I appartient au concept C ( I : C).

#### Fonctionnement :

Le prédicat lit en premier l'instance I, vérifie qu'il s'agit bien d'un nom d'instance, puis lit le concept C et vérifie qu'il s'agit bien d'un concept (à l'aide du prédicat concept).

Après lecture de I et de C, on remplace C par sa définition donnée dans la Tbox s'il s'agit d'un concept non atomique, puis on le transforme en sa forme normale négative.

Finalement, on ajoute (I, Cnnf) à la Abox des instances (en l'occurrence Abi) et on obtient une nouvelle Abi1.

Code :

```
132 % concat/3 : concatene les deux listes L1 et L2 dans L3
133 concatene([],L1,L1).
134 concatene([X|Y],L1,[X|L2]) :- concatene(Y,L1,L2).
135
136 rev([I|C],[I,Ctraiteenf]) :- replace_concept_na(C,Ctraite), nnf(not(Ctraite),Ctraiteenf).
137
138 % Méthode permettant l acquisition de propositions du type 1 (I : C)
139 acquisition_prop_type1(Abi,Abi1,Tbox) :-
140     % On entre le type de l instance
141     nl, write('Veuillez entrer l"instance :'),
142     nl, read(I),
143
144     % On réalise une vérification sur l instance
145     iname(T),
146
147     % On entre le concept
148     nl, write('Veuillez entrer le concept :'),
149     nl, read(C),
150
151     % On effectue une vérification sur le concept
152     % concept(C)
153
154     % On effectue les manipulations sur le concept
155     % On le remplace (RC) puis on effectue sa négation (NRC)
156     rev([I,C],Ctraiteenf),
157
158     % On ajoute l élément (I, Ctraiteenf) à la ABox
159     concatene([Ctraiteenf], Abi,Abi1).
160
161
```

Figure 2.1: Première partie

## 2.2 Prédicat acquisition\_prop\_type2(Abi,Abi1,Tbox)

acquisition\_prop\_type2/3 : Permet à l'utilisateur de notre démonstrateur d'entrer une proposition de type 2 ie les deux concepts C1 et C2 ont une intersection vide.

### Fonctionnement :

Le prédicat lit en premier les deux concepts et vérifie qu'il s'agit bien de concepts (en utilisant le prédicat concept/1).

Après lecture de C1 et C2, on les remplace par leurs définitions respectives dans la Tbox puis on les transforme en NNF.

Finalement, on génère une instance Inst et on ajoute l'élément (I : C1 and C2) à la Abox des instances (en l'occurrence Abi) et on obtient une nouvelle Abi1.

Code :

```
102 %
163 % Méthode permettant l acquisition de proposition du type 2 :
164 acquisition_prop_type2(Abi,Abi1,Tbox) :-
165     % on entre le concept 1
166     nl, write('veuillez entrer le concept 1 :'),
167     nl, read(C1),
168     concept(C1),
169
170
171     % on entre le concept 2
172     nl, write('veuillez entrer le concept 2 :'),
173     nl, read(C2),
174
175     % on effectue une vérification sur les deux concepts
176     concept(C2),
177     nl, write('Etape concept passée'),
178
179     % on effectue les manipulations sur les concepts
180     % on effectue le remplacement (RC) puis on on effectue sa négation (NRC)
181
182 traitement([C1,C2],(C1traite,C2traite)),
183
184
185     % On génère une instance et on ajoute l élément (I : C1 and C2) à la ABox
186     generer(Inst),
187     concatene([(Inst, and(C1traite,C2traite))],Abi,Abi1).
188
189
190 % Traitement sémantique de la Tbox (prédictat traitement)
191 traitement([],(_,_)).
192 traitement([C2],(_,C2traite)) :- replace_concept_na([C2],C2traite), !.
193 traitement([C1|C2],(C1traite_n,C2traite_n)) :- replace_concept_na([C1],C1traite),
194     traitement(C2,(C1traite,C2traite)),
195     nnf(C1traite, C1traite_n),
196     nnf(C2traite, C2traite_n).
```

Figure 2.2: Deuxième partie

## 2.3 Exemple - Acquisition de proposition

```
M1 DAC - PROJET LRC - Samy NEHLIL et Alla BOUTALEB
_____  
Premiere partie _____  
Tbox = [(auteur, and(personne, some(aEcrit, livre))), (editeur, and(personne, and(not(some(aEcrit, livre)), some(aEdite, livre)))), (parent, and(personne, some(aCree, sculpture))))]  
Abi = [(david, sculpture), (joconde, objet), (michelAnge, personne), (socrate, personne), (sonnets, livre), (vinci, personne)]  
Abr = [(michelAnge, david, aCree), (michelAnge, sonnets, aEcrit), (vinci, joconde, aCree)]  
Traitement de la Tbox realise avec succes  
Traitement Abi avec succes  
Traitement Abr avec succes  
Traitement de la Abx realisé avec succès  
Traitement de la Tbox et de la Abx realisé avec succès  
FIN DE LA PREMIERE PARTIE AVEC SUCCES
_____  
Deuxieme partie _____  
Entrer le numero du type de proposition que l'on souhaite démontrer :  
    Type 1 - Une instance donnée appartient à un concept donné.  
    Type 2 - Deux concepts n-éléments en commun dont l'intersection est vide (négation).  
|: 1.  
Veuillez entrer l'instance :  
|: michelAnge.  
Veuillez entrer le concept :  
|: some(aEcrit, livre).
_____  
Traitement de la Tbox realisé avec succès  
Traitement de la Abx realisé avec succès  
Traitement de la Tbox et de la Abx realisé avec succès
```

Figure 2.3: Exemple d'acquisition de props

# CHAPTER 3

## PARTIE III : DÉMONSTRATION DE LA PROPOSITION

Cette partie a pour but d'acquérir la proposition à prouver à l'aide du démonstrateur.

### 3.1 Prédicat tri\_Abox

#### Fonctionnement :

Le prédicat tri\_Abox, à partir de la liste des assertions de concepts de la Abox étendue après soumission d'une proposition à démontrer, génère 5 listes :

- la liste Lie des assertions du type (I,some(R,C))
- la liste Lpt des assertions du type (I,all(R,C))
- la liste Li des assertions du type (I, and(C1,C2))
- la liste Lu des assertions du type (I, or(C1,C2))
- la liste Ls des assertions restantes, à savoir les assertions du type (I,C) ou (I,not(C)), C étant un concept atomique.

Pour simplifier, tri\_Abox permet de classer une assertion selon son type dans une liste différente afin d'exploiter cette dernière par la suite dans la résolution.

Code :

```

388  % Le prédictat tri_Abox, à partir de la liste des assertions de concepts de la Abox étendue
389  % Cas d'arrêt
390  tri_Abox([],[],[],[],[],[]).
391  % some -> Lie
392  tri_Abox([(I,some(R,C))|T],LieNew,Lpt,Li,Lu,Ls) :- concatene([(I,some(R,C))],Lie,LieNew), tri_Abox(T,Lie,Lpt,Li,Lu,Ls),!.
393  % all -> lpt
394  tri_Abox([(I,all(R,C))|T],Lie,LptNew,Li,Lu,Ls) :- concatene([(I,all(R,C))],Lpt,LptNew), tri_Abox(T,Lie,Lpt,Li,Lu,Ls),!.
395  % and -> li
396  tri_Abox([(I, and(C1,C2))|T],Lie,Lpt,Li,Lu,Ls) :- concatene([(I, and(C1,C2))],Li,LiNew), tri_Abox(T,Lie,Lpt,Li,Lu,Ls),!.
397  % or -> Lu
398  tri_Abox([(I,or(C1,C2))|T],Lie,Lpt,Li,LuNew,Ls) :- concatene([(I,or(C1,C2))],Lu,LuNew), tri_Abox(T,Lie,Lpt,Li,Lu,Ls),!.
399  % not(concept) -> Ls
400  tri_Abox([(I,not(C))|T],Lie,Lpt,Li,Lu,LsNew) :- concatene([(I,not(C))],Ls,LsNew), tri_Abox(T,Lie,Lpt,Li,Lu,Ls),!.
401  % concept -> Ls
402  tri_Abox([(I,C)|T],Lie,Lpt,Li,Lu,LsNew) :- concatene([(I,C)],Ls,LsNew), tri_Abox(T,Lie,Lpt,Li,Lu,Ls),!.
403

```

Figure 3.1: tri\_abox

### Test :

```

?- tri_Abox([(michelAnge,personne), (david,sculpture), (sonnets,livre),(vinci, personne),(joconde,objet)],Lie,Lpt,Li,Le,Ls).
Lie = Lpt, Lpt = Li, Li = Le, Le = [],
Ls = [(michelAnge, personne), (david, sculpture), (sonnets, livre), (vinci, personne), (joconde, objet)].

```

```

?- tri_Abox([(michelAnge,personne), (david,sculpture), (sonnets,livre), (vinci, and(parent,editeur))
Lie = [],
Lpt = [(michelAnge, all(aEcrit, livre))],
Li = [(vinci, and(parent, editeur))],
Le = [(joconde, or(objet, livre))],
Ls = [(michelAnge, personne), (david, sculpture), (sonnets, livre)].

```

## 3.2 Prédicat test\_clash(Ls)

### Fonctionnement :

Retourne vrai si pas de clash dans la liste, faux sinon. Ce prédictat vérifie si il existe dans Ls une assertion et sa négation.

### Code :

```

437  |
438  % test_clash/1 : retourne vrai si pas de clash dans la liste, faux sinon
439  % Idée : Si un concept C est dans le tableau qu'on développe, vérifier si nnf(C) y est aussi
440  test_clash([]).
441  test_clash([(I,C)|T]) :- nnf(not(C),Cnnf), not(member((I,Cnnf),T)), test_clash(T).

```

Figure 3.2: test\_clash

### Test :

```
?- test_clash([(michelAnge,personne),(david,sculpteur),(michelAnge,not(personne))]).  
false.  
?- test_clash([(michelAnge,personne),(david,sculpteur),(michelAnge,auteur)]).  
true.
```

## 3.3 Prédicat resolution

### Fonctionnement :

Permet de développer l'arbre de la méthodes des tableaux sémantiques suivant le type d'expressions que l'on a dans le tableau actuel. Ce prédicat passe par toutes les listes retournées par tri\_Abox, pour chacune des listes vérifie si elle contient des assertions. Dans le cas où la liste n'est pas vide, alors on applique le traitement correspondant au type des assertions que contient cette liste.

### Code :

```
438 % test_clash/1 : retourne vrai si pas de clash dans la liste, faux sinon  
439 % idée : si un concept C est dans le tableau qu on développe, verifier si nnf(C) y est aussi  
440 test_clash([]).  
441 test_clash([(I,C)|T]) :- nnf(not(C),Cnnf), not(member((I,Cnnf),T)), test_clash(T).  
442  
443 % Prédicat résolution  
444 % Règle IL EXISTE  
445 resolution(Lie,Lpt,Li,Lu,Ls,Abr) :- not(length(Lie,0)), test_clash(Ls), write("\nAppel de complete_some\n"), complete_some(Lie,Lpt,Li,Lu,Ls,Abr).  
446 % règle ET  
447 resolution(Lie,Lpt,Li,Lu,Ls,Abr) :- not(length(Li,0)), test_clash(Ls), write("\nAppel de transformation_and\n"), transformation_and(Lie,Lpt,Li,Lu,Ls,Abr).  
448 % règle POUR TOUT  
449 resolution(Lie,Lpt,Li,Lu,Ls,Abr) :- not(length(Lu,0)), test_clash(Ls), write("\nAppel de deduction_all\n"), deduction_all(Lie,Lpt,Li,Lu,Ls,Abr).  
450 % règle OU  
451 resolution(Lie,Lpt,Li,Lu,Ls,Abr) :- not(length(Ls,0)), test_clash(Ls), write("\nAppel de transformation_or\n"), transformation_or(Lie,Lpt,Li,Lu,Ls,Abr).  
452  
453 resolution([],[],[],[],[],Abr) :- not(test_clash(Ls)), write("\nBranche fermée !!\n").
```

Figure 3.3: resolution

## 3.4 Prédicat complete\_some(Lie,Lpt,Li,Lu,Ls,Abr)

### Fonctionnement :

Ce prédicat traite une assertion de concept de la forme  $(I, \text{some}(R, C))$  dans la liste Lie. L'appel de ce prédicat consiste à appliquer la règle  $\exists$  comme décrit dans le sujet du projet.

### Code :

```
471 complete_some([(I,some(R,C))|Lie1],Lpt,Li,Lu,Ls,Abr) :- generer(B), /*on cree un nouvel objet B*/  
472 concatene([(I,B,R)],Abr,AbrNew), /*on ajoute (I,B,R) dans Abr*/  
473 evolve((B,C),Lie1,Lpt,Li,Lu,Ls,Lie1,Lpt1,Li1,Lu1,Ls1), /*I ajoute de (B,C) depend de la nature de C*/  
474 affiche_evolution_Abox(Ls,[(I,some(R,C))|Lie1],Lpt,Li,Lu,Abr,Ls1,lie1,lpt1,li1,lu1,AbrNew). /*on boucle*/  
475  
476  
477
```

Figure 3.4: complete\_some

### 3.5 Prédicat transformation\_and(Lie,Lpt,Li,Lu,Ls,Abr)

**Fonctionnement :**

Ce prédicat traite une assertion de concept de la forme  $(I, \text{and}(C1, C2))$  dans la liste Li. L'appel de ce prédicat consiste à appliquer la règle  $\sqcap$  comme décrit dans le sujet du projet.

**Code :**

```

471  complete_some([(I,some(R,C))|Lie],Lpt,Li,Lu,Ls,Abr) :- generer(B), /*on cree un nouvel objet B*/
472      concatene([(I,B,R)],Abr,AbrNew), /*on ajoute (I,B,R) dans Abr*/
473      evolue((B,C),Lie,Lpt,Li,Lu,Ls,Lie1,Lpt1,Li1,Lu1,Ls1), /*l ajout de (B,C) depend de la nature de C*/
474      affiche_evolution_Abox(Ls,[|(I,some(R,C))|Lie], Lpt, Lie, Lu , Abr, Ls1, Lie1, Lpt1, Li1, Lu1, AbrNew)
475      resolution(Lie1,Lpt1,Li1,Lu1,Ls1,AbrNew). /*on boucle*/
476

```

Figure 3.5: transformation\_and

### 3.6 Prédicat deduction\_all(Lie,Lpt,Li,Lu,Ls,Abr)

**Fonctionnement :**

Ce prédicat traite une assertion de concept de la forme  $(I, \text{all}(R,C))$  dans la liste Lpt. L'appel de ce prédicat consiste à appliquer la règle  $\forall$  comme décrit dans le sujet du projet.

**Code :**

```

487
488  deduction_all(Lie,[|(I,all(R,C))|Lpt],Li,Lu,Ls,Abr) :- member((I,B,R),Abr), /*on a du I : all(R,C) donc pour utiliser la règle ∀, on cherche une relation
489      evolue((B,C),Lie,Lpt,Li,Lu,Ls,Lie1,Lpt1,Li1,Lu1,Ls1), /* ajout de (B,C)*/
490      affiche_evolution_Abox(Ls,Lie,[|(I,all(R,C))|Lpt], Lie, Lu , Abr, Ls1, Lie1, Lpt1, Li1, Lu1, Abr),
491      resolution(Lie1,Lpt1,Li1,Lu1,Ls1,Abr).
492

```

Figure 3.6: deduction\_all

### 3.7 Prédicat transformation\_or(Lie,Lpt,Li,Lu,Ls,Abr)

**Fonctionnement :**

Ce prédicat traite une assertion de concept de la forme  $(I, \text{or}(C1, C2))$  dans la liste Lu. L'appel de ce prédicat consiste à appliquer la règle  $\cup$  comme décrit dans le sujet du projet.

**Code :**

```

494  transformation_or(Lie,Lpt,Li,[|(I,or(C1,C2))|Lu],Ls,Abr) :- evolue((I,C1),Lie,Lpt,Li,Lu,Ls,Lie1,Lpt1,Li1,Lu1,Ls1), /*ajout fils gauche*/
495      affiche_evolution_Abox(Ls,Lie,Lpt, Lie, [|(I,or(C1,C2))|Lu] , Abr, Ls1, Lie1,Lpt1,Li1,Lu1,Ls1),
496      evolue((I,C2),Lie,Lpt,Li,Lu,Ls,Lie2,Lpt2,Li2,Lu2,Ls2), /*ajout fils droit*/
497      affiche_evolution_Abox(Ls,Lie,Lpt, Lie, [|(I,or(C1,C2))|Lu] , Abr, Ls2, Lie2,Lpt2,Li2,Lu2,Ls2),
498      resolution(Lie1,Lpt1,Li1,Lu1,Ls1,Abr), /*fils gauche*/
499      resolution(Lie2,Lpt2,Li2,Lu2,Ls2,Abr). /*fils droit*/

```

Figure 3.7: transformation\_or

### 3.8 Prédicat evolve(A, Lie, Lpt, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1)

#### Fonctionnement :

Ce prédicat permet de mettre à jour les listes contenant les assertions, il permet simplement de rajouter une nouvelle assertion A à la liste correspondante à son type de concept. A représente une nouvelle assertion de concepts à intégrer dans l'une des listes Lie, Lpt, Li, Lu ou Ls qui décrivent les assertions de concepts de la Abox étendue et Lie1, Lpt1, Li1,Lu1 et Ls1 représentent les nouvelles listes mises à jour.

#### Code :

```

455 % evolve/11 : maj des listes de Abox
456 evolve((I,some(R,C)),lie,lpt,li,lu,ls,Lie1,lpt,Li,lu,ls) :- concatene([(I,some(R,C))],lie,lie1),!.
457 evolve((I, and(C1,C2)),lie,lpt,li,lu,ls,lie,lpt,li,lu,ls) :- concatene([(I, and(C1,C2))],li,li1),!.
458 evolve((I, or(C1,C2)),Lie,Lpt,li,lu,ls,lie,lpt,li,lu1,ls) :- concatene([(I, or(C1,C2))],lu,lu1),!.
459 evolve((I,all(R,C)),lie,lpt,li,lu,ls,lie,lpt1,li,lu,ls) :- concatene([(I,all(R,C))],lpt,lpt1),!.
460 evolve((I,not(C)),lie,lpt,li,lu,ls,lie,lpt,li,lu,ls) :- cnamea(C), concatene([(I,not(C))],ls,ls1),!.. /*si C est un concept atomique, on ajoute dans ls*/
461 evolve((I,not(C)),lie,lpt,li,lu,ls,lie1,lpt1,li1,lu1,ls1) :- not(cnamea(C)), nnf(not(C),NotCnnf), evolve((I,NotCnnf),lie,lpt,li,lu,ls,lie1,lpt1,li1,lu1,ls
462 evolve((I,C),lie,lpt,li,lu,ls,lie,lpt,li,lu,ls) :- concatene([(I,C)],ls,ls1),!.
463

```

Figure 3.8: evolve

### 3.9 Prédicat affiche\_evolution\_Abox(Ls1, Lie1, Lpt1, Li1, Lu1, Abr1, Ls2, Lie2, Lpt2, Li2, Lu2, Abr2)

#### Fonctionnement :

Ce prédicat affiche l'évolution d'un état de la Abox étendue (en ce qui concerne les listes des assertions de concepts et la liste des assertions de rôles) vers un état suivant, les 6 premiers paramètres décrivant l'état de départ, les 6 suivants l'état d'arrivée. Abr1 et Abr2 représentent respectivement les états de départ et d'arrivée de l'ensemble des assertions de rôles de la Abox étendue. L'affichage doit être agréable pour permettre à l'utilisateur de suivre facilement le développement de l'arbre de démonstration. On affichera donc les différentes assertions en utilisant les symboles mathématiques  $\exists$ ,  $\cup$ ,  $\neg$ ,  $\forall$ ,  $\sqcap$ , et une notation infixée et non préfixée.

Code :

```

44074
405 % affiche/1: prédict qui affiche une liste d assertions
406 | % Utile pour le prédict affiche_evolution_Abox demandé
407 affiche([[]).
408 affiche([A|L]) :- affiche(A),affiche(L).
409 affiche((A,B,R)) :- nl,write("<"),write(A),write(","),write(B),write("> : "),write(R).
410 affiche((I,or(C1,C2))) :- nl,write(I),write(" : "), affiche(C1),write(" ∣ "),affiche(C2).
411 affiche((I, and(C1,C2))) :- nl,write(I),write(" : "), affiche(C1),write(" ∩ "),affiche(C2).
412 affiche((I,C)) :- nl,write(I), write(" : "), affiche(C).
413 affiche(or(C1,C2)) :- write("("),affiche(C1),write(" ∣ "),affiche(C2),write(")")..
414 affiche(and(C1,C2)) :- write("("),affiche(C1),write(" ∩ "),affiche(C2),write(")")..
415 affiche(all(R,C)) :- write("∀"),write(R),write("."),affiche(C).
416 affiche(some(R,C)) :- write("∃"), write(R), write("."), affiche(C).
417 affiche(not(C)) :- write("¬"),affiche(C).
418 affiche(C) :- write(C).
419
420 % affiche_evolution_Abox/12 : Affiche l évolution de la Abox étendue
421 affiche_evolution_Abox(Ls, Lie, Lpt, Li, Lu, Abr, Ls1, Lie1, Lpt1, Lii, Lui, Abr1) :- write("---Etat de départ de la Abox---"),
422 | affiche(Ls),
423 | affiche(Lie),
424 | affiche(Lpt),
425 | affiche(Li),
426 | affiche(Lu),
427 | affiche(Abr),
428 | nl,nl,write("---Etat d'arrivée---"),
429 | affiche(Ls1),
430 | affiche(Lie1),
431 | affiche(Lpt1),
432 | affiche(Lii),
433 | affiche(Lui),
434 | affiche(Abr1),
435 | nl,write("=====FIN====="),nl,!.
436

```

Figure 3.9: affiche\_evolution\_Abox

## 3.10 Démonstration de notre programme

1. (Type 1) Montrer que Michel-Ange a écrit un livre.

```
?- programme.
M1 DAC - PROJET LRC - Saxy NEHIL et Alles BOUTALEB
_____  

Premiere partie  

Tbox = [(auteur, and(personne, some(aEcrit,livre)),(editeur, and(personne, and(not(some(aEcrit,livre)),some(aEdi:e,livre)))),(parent, and(personne, some(aEnfant,anything)),(sculpteur, and(personne, some(aCree,sculpture))))]  

Abi = [(david,sculpture),(jocconde,objet),(michelange,personne),(socrate,personne),(sonnets,livre),(vinci,personne)]  

Ahr = [(michelange,david,aCree),(michelange,sonnets,aEcrit),(vinci,jocconde,aCree)]  

Traitement de la Tbox realise avec succes  

Traitement Abi avec succes  

Traitement Ahr avec succes  

Traitement de la Tbox realise avec succes  

Traitement de la Tbox de la Abi realise avec succes  

FIN DE LA PREMIERE PARTIE AVEC SUCCES  

_____  

Deuxieme partie  

_____  

Entrer le numero du type de proposition que l'on souhaite demontrer :  

Type 1 - Une instance donnee appartient à un concept donne  

Type 2 - Deux concepts n-elements en common dont l'intersection est vide (negation).  

|: 1.  

Veuillez entrer l'instance :  

|: michelange  

Veuillez entrer le concept :  

|: some(aEcrit, livre).
_____  

Troisieme partie  

_____  

Appel de deduction_all  

---  

david : sculpture  

jocconde : objet  

michelange : personne  

socrate : personne  

sonnets : livre  

vinci : personne  

michelange,david : aCree  

michelange,sonnets : aEcrit  

vinci,jocconde : aCree  

.....  

...Frot d'instance  

sonnets : Not livre  

david : sculpture  

jocconde : objet  

michelange : personne  

socrate : personne  

sonnets : livre  

vinci : personne  

michelange,david : aCree  

michelange,sonnets : aEcrit  

vinci,jocconde : aCree  

*****FIN*****  

Branche fermee !!  

Youspiaiii, on a demonstre la proposition initiale !!!  

true .  

?-
```

Figure 3.10: Proposition de type 1

## 2. (Type 2) Montrer que Editeur $\sqcap$ Auteur $\sqsubseteq \perp$

```
% Execution Aborted
?- program.
M1 DAC - PROJET LRC - Samy NEHLIL et Aissa BOUTALEB
_____  

Premiere partie _____  

Tbox = [(auteur, and(personne, some(aEcrit, livre)), (editeur, and(personne, and(not(some(aEcrit, livre)), some(aEdite, livre)))), (parent, and(personne, some(aEnfant, anything))), (sculpteur, and(personne, some(aCree, sculpture))))]  

Abi = [(david, sculpture), (joconde, objet), (michelange, personne), (socrete, personne), (sonnets, livre), (vinci, personne)]  

Abx = [(michelange, david, aCree), (michelange, sonnets, aEcrit), (vinci, joconde, aCree)]  

Traitement de la Tbox realise avec succes  

Traitement Abi avec succes  

Traitement Abx avec succes  

Traitement de la Tbox avec succes  

Traitement de la Abox realise avec succes  

Traitement de la Tbox et de la Abox realise avec succes  

FIN DE LA PREMIERE PARTIE AVEC SUCCES  

Deuxieme partie _____  

Entrer le numero du type de proposition que l'on souhaite demontrer :  

| Type 1 - Une instance donnee appartient a un concept donne.  

| Type 2 - Deux concepts n-elements en commun dont l'intersection est vide (negation).  

| 2.  

Veuillez entrer le concept 1 :  

| auteur  

Veuillez entrer le concept 2 :  

| editeur.
```

Figure 3.11: Proposition de type 2

```

---Etat d'arrivee---
inst1 : personne
david : sculpture
joconde : objet
michelAnge : personne
socrate : personne
sonnets : livre
vinci : personne
inst1 : All.aEcrit.Not.livre and Some.aEdite.livre
inst1 : personne and Some.aEcrit.livre
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree
=====FIN=====

Appel de transformation_and
---Etat de depart de la Abox---
inst1 : personne
david : sculpture
joconde : objet
michelAnge : personne
socrate : personne
sonnets : livre
vinci : personne
inst1 : All.aEcrit.Not.livre and Some.aEdite.livre
inst1 : personne and Some.aEcrit.livre
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

---Etat d'arrivee---
inst1 : personne
david : sculpture
joconde : objet
michelAnge : personne
socrate : personne
sonnets : livre
vinci : personne
inst1 : Some.aEdite.livre
inst1 : All.aEcrit.Not.livre
inst1 : personne and Some.aEcrit.livre
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree
=====FIN=====

Appel de complete_some
---Etat de depart de la Abox---
inst1 : personne
david : sculpture

```

Figure 3.12: Proposition de type 2

```

michelAnge : personne
socrate : personne
sonnets : livre
vinci : personne
inst1 : All.aEcrit.Not.livre
<inst1,80589> : aEcrit
<inst1,77058> : aEdite
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree
=====FIN=====

Appel de deduction_all
---Etat de depart de la Abox---
80589 : livre
inst1 : personne
77058 : livre
inst1 : personne
david : sculpture
joconde : objet
michelAnge : personne
socrate : personne
sonnets : livre
vinci : personne
inst1 : All.aEcrit.Not.livre
<inst1,80589> : aEcrit
<inst1,77058> : aEdite
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

---Etat d'arrivee---
80589 : Not.livre
80589 : livre
inst1 : personne
77058 : livre
inst1 : personne
david : sculpture
joconde : objet
michelAnge : personne
socrate : personne
sonnets : livre
vinci : personne
<inst1,80589> : aEcrit
<inst1,77058> : aEdite
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree
=====FIN=====

Branche fermee !!

Youpiiiiii, on a demonstre la proposition initiale !!!
true .

```

Figure 3.13: Proposition de type 2