
# Rapport de projet

*Logique et représentation des connaissances*

Rédigé par :

**BENAISSA Rania**

**ZHANG Zimeng**

*Année universitaire 2021/2022*

# TABLE DES MATIÈRES

# 1 Introduction

Dans ce présent document, nous décrivons les différentes parties constituant notre démonstrateur de propositions. Chaque partie est consolidée par l'explication et l'exécution des prédicats qui y sont définis. Quant à la dernière section (section **6**), elle réunit des exemples de démonstration de diverses propositions.

# 2 Lancement du programme

Dans le fichier **«.pl»**, on retrouve :

- Tous les prédicats que nous avons personnellement implémentés
- La base sur laquelle nous nous sommes reposés (Tbox et Abox de l'exercice 3 du TD4)
- Les différents utilitaires donnés en annexe (tels que concat,enleve, etc...)

Le prédicat permettant de lancer le démonstrateur est le prédicat *«program»* qui fait appel aux trois principaux prédicats de chaque partie du démonstrateur :

```
/************* PROJECT *************/

/* ----FINAL PROG---- */
program :- premiere_etape(Tbox,Abi,Abr),
          deuxieme_etape(Abi,Abi1,Tbox),
          troisieme_etape(Abi1,Abr).
```

**Figure 1:** Prédicat de lancement du programme

# 3 Partie 1 : préliminaires

La première étape revient à construire les listes de la Abox et de la Tbox qui est présentée à travers les prédicats suivants:

```
/*  ----PARTIE 1---- */

/*  TBOX = liste de ( concept Atomique, concept generique) */
tbox(L) :- setof((X,Y),equiv(X,Y),L).

/*  ABI = liste de ( instance, concept) */
abi(L) :- setof((X,Y),inst(X,Y),L).

/*  ABR = liste de ( instance,instance, role) */
abr(L) :- setof((X,Y,Z),instR(X,Y,Z),L).

premiere_etape(Tbox,Abi,Abr) :- tbox(Tbox), abi(Abi), abr(Abr).
```

**Figure 2:** Prédicats de la partie 1

A l'aide des prédicats prédéfinis «equiv», «inst» et «instR», nous avons défini des prédicats permettant d'obtenir les listes Tbox, Abi et Abr:

- **tbox :** ce prédicat retourne une liste de tuples L où chaque tuple désigne un concept non atomique et sa définition, comme le montre le test suivant:

```
?- tbox(L).
L = [(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(person
ne, some(aCree, sculpture)))].
```

**Figure 3:** Résultat de l'exécution du prédicat «tbox»

- **abi:** ce prédicat retourne une sous-liste qui contient les assertions de concepts, comme suit:

```
?- abi(L).
L = [(david, sculpture),  (joconde, objet),  (michelAnge, personne),  (sonnets, livre),  (vinci, perso
nne)].
```

**Figure 4:** Résultat de l'exécution du prédicat «abi»

- **abr :** ce prédicat retourne une sous-liste qui contient les assertions de rôles :

```
?- abr(L).
L = [(michelAnge, david, aCree),  (michelAnge, sonnets, aEcrit),  (vinci, joconde, aCree)].
```

**Figure 5:** Résultat de l'exécution du prédicat «abr»

- **premiere_etape :** retourne trois listes dont la Tbox et la Abox subdivisée en deux sous-listes : Abi et Abr

```
?- premiere_etape(Tbox,Abi,Abr).
Tbox = [(auteur, and(personne, some(aEcrit, livre))),  (editeur, and(personne, and(not(some(aEcrit, li
vre)), some(aEdite, livre)))),  (parent, and(personne, some(aEnfant, anything))),  (sculpteur, and(per
sonne, some(aCree, sculpture)))],
Abi = [(david, sculpture),  (joconde, objet),  (michelAnge, personne),  (sonnets, livre),  (vinci, per
sonne)],
Abr = [(michelAnge, david, aCree),  (michelAnge, sonnets, aEcrit),  (vinci, joconde, aCree)].
```

**Figure 6:** Résultat de l'exécution du prédicat «premiere_etape»

# 4 Partie 2 : Saisie de la proposition à démontrer

La deuxième partie du projet, qui consiste à saisir et à vérifier la proposition à démontrer, est lancée via le prédicat «deuxieme_etape» appelé après exécution de la première étape et en récupère la Tbox et Abi (voir figure 1)

Celui-ci fait appel au prédicat de saisie de propositions comme l'illustre la figure ci-dessous:

```
saisie_et_traitement_prop_a_demontrer(Abi,Abi1,Tbox) :-
nl,write('Entrez le numero du type de proposition que vous voulez demontrer :'),nl,nl,
write('1 - Une instance donnee appartient a un concept donne.'),nl,nl,
write('2 - Deux concepts n\'ont pas d\'elements en commun.'),nl,nl,read(R),nl,suite(R,Abi,Abi1,Tbox).


deuxieme_etape(Abi,Abi1,Tbox) :- saisie_et_traitement_prop_a_demontrer(Abi,Abi1,Tbox).
```

**Figure 7:** Illustration du prédicat de choix du type de propositions à démontrer

L'utilisateur est donc amené à saisir le chiffre «1» s'il souhaite démontrer une proposition du type $I : C$ et à saisir le chiffre «2» s'il souhaite démontrer une proposition du type $C_1 \sqcap C_2 \sqsubseteq \bot$.

Le prédicat *«suite»* se chargera ensuite de diriger l'acquisition de la proposition selon le type choisi.

## 4.1 Acquisition des propositions

La démonstration d'une proposition (qu'importe son type) requiert que l'on effectue quelques étapes préliminaires que l'on illustre dans les sous-sections à venir.

### 4.1.1 Acquisition de la proposition de type : I:C

```
acquisition_prop_type1(Abi,Abi1,Tbox) :- demander_proposition1(Instance,Concept),
                        is_correct_pro1(Instance,Concept),
                        transform_all(Tbox,Abi,AbiA),
                        transform_concept(Tbox,not(Concept),Prop_atomique),
                        nnf(Prop_atomique,Negation),
                        concat(AbiA,[(Instance,Negation)],Abi1).
```

**Figure 8:** Prédicat d'acquisition du $1^{er}$ type de propositions

- ***demander_proposition1*** : récupère l'instance et le concept introduits par l'utilisateur

- ***is_correct_pro1*** : vérifie que l'instance et le concept saisis par l'utilisateur sont corrects
- ***transform_all*** : fait appel à *transform_concept* pour transformer tous les concepts complexes déjà présents dans la liste des assertions de concepts par leurs définitions
- ***transform_concept*** : transforme un concept complexe par sa définition dans la Tbox

Voici un exemple d'exécution du prédicat *«acquisition_prop_type1»* sur notre base :

```
7 ?- premiere_etape(Tbox,Abi,Abr),acquisition_prop_type1(Abi,Abi1,Tbox).
Entrez l'instance :
|: sonnets.

Entrez le concept :
|: or(auteur,livre).

Tbox = [(auteur,and(personne,some(aEcrit,livre))),(editeur,and(personne,and(not(some(aEcrit,livre)),some(aEdite,livre)))),(parent,and(pers
onne,some(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)],
Abi1 = [(david,sculpture),(joconde,objet),(michelAnge,personne),(sonnets,livre),(vinci,personne),(sonnets,and(or(not(personne),all(aEcrit,
not(livre))),not(livre)))] .
```

**Figure 9:** Test du prédicat d'acquisition du $1^{eme}$ type de propositions

Et on voit bien que la liste Abi a été correctement étendue avec la forme normale négative de la négation de la proposition transformée introduite.

### 4.1.2 Acquisition de la proposition de type : $C_1 \sqcap C_2 \sqsubseteq \bot$

```
acquisition_prop_type2(Abi,Abi1,Tbox) :- demander_proposition2(Concept1,Concept2),
                        is_correct_pro2(Concept1,Concept2),
                        transform_all(Tbox,Abi,AbiA),
                        transform_concept(Tbox,Concept1,Prop_atomique1),
                        transform_concept(Tbox,Concept2,Prop_atomique2),
                        genere(Nom), /* on genere un nom car on veut une instantiation
                        dans la negation */
                        concat(AbiA,[(Nom,and(Prop_atomique1,Prop_atomique2))],Abi1).
```

**Figure 10:** Prédicat d'acquisition du $2^{eme}$ type de propositions

- ***demander_proposition2*** : demande à l'utilisateur d'entrer deux concepts
- ***is_correct_pro2*** : vérifie que les concepts saisis par l'utilisateur sont corrects

Voici un exemple d'exécution du prédicat *«acquisition_prop_type2»* sur notre base:

```
9 ?- premiere_etape(Tbox,Abi,Abr),acquisition_prop_type2(Abi,Abi1,Tbox).
Entrez le concept 1 :
|: some(aEcrit,livre).

Entrez le concept 2 :
|: sculpteur.


Tbox = [(auteur,and(personne,some(aEcrit,livre))),(editeur,and(personne,and(not(some(aEcrit,livre)),some(aEdite,livre)))),(parent,and(pers
onne,some(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)],
Abi1 = [(david,sculpture),(joconde,objet),(michelAnge,personne),(sonnets,livre),(vinci,personne),(inst1,and(some(aEcrit,livre),and(personn
e,some(aCree,sculpture))))] .
```

**Figure 11:** Test du prédicat d'acquisition du $2^{eme}$ type de propositions

Comme pour le premier type de propositions, la Abox a correctement été étendue.

## 4.2 Vérification syntaxique et sémantique des propositions

Les prédicats qui se chargent de vérifier syntaxiquement et grammaticalement les données entrées par l'utilisateur sont les suivants:

```
/* verifier si les propositions sont correctes */
is_correct_pro1(Instance,Concept) :- instance(Instance), concept(Concept),!.
is_correct_pro1(Instance,_) :- not(instance(Instance)),nl,
                    write('L\'instance donnee est incorrecte'),instance(Instance),!.
is_correct_pro1(_,Concept) :- not(concept(Concept)),nl, write("Le concept donne est incorrect")
                        ,nl,concept(Concept),!.

is_correct_pro2(Concept1,Concept2) :- concept(Concept1), concept(Concept2),!.
is_correct_pro2(Concept1,_) :- not(concept(Concept1)),nl,
                       write("Le concept1 donne est incorrect"),nl,concept(Concept1),!.
is_correct_pro2(_,Concept2) :- not(concept(Concept2)),nl,
                        write("Le concept2 donne est incorrect"),nl,concept(Concept2),!.
```

**Figure 12:** Prédicats de vérification des propositions

Les concepts, les instances et les rôles sont définis par le biais de ces prédicats:

```
/* Verifier si instance existe idf */
instance(Instance) :- atom(Instance), setof(X,iname(X),L), member(Instance,L),!.

/* Verifier si un role existe*/
role(Role) :-  setof(X,rname(X),L), member(Role,L),!.

/* Verifier si un concept est correct grammaticalement et syntaxiquement*/

is_concept_atom(Concept) :- atom(Concept), setof(X,cnamea(X),L), member(Concept,L),!.
is_concept_gen(Concept) :- atom(Concept), setof(X,cnamena(X),L), member(Concept,L),!.


concept(Concept) :- is_concept_atom(Concept).
concept(Concept) :- is_concept_gen(Concept).
concept(not(Concept)) :- concept(Concept),!.
concept(and(Concept1,Concept2)) :- concept(Concept1), concept(Concept2),!.
concept(or(Concept1,Concept2)) :- concept(Concept1), concept(Concept2),!.
concept(all(R,Concept)) :- role(R),concept(Concept),!.
concept(some(R,Concept)) :- role(R),concept(Concept),!.
```

**Figure 13:** Prédicats définition d'un concept, d'une instance et d'un rôle

- ***instance*** vérifie si la donnée introduite est une variable ou un atom, si ce n'est pas une variable, elle vérifie ensuite si cet identificateur est bien un identificateur d'instance
- ***role*** : vérifie si un rôle existe
- ***is_concept_atom*** : vérifie si un concept est un concept atomique
- ***is_concept_gen*** : vérifie si un concept est un concept générique

*Il est à noter que les concepts sont bien définis selon la grammaire des concepts.*

Voici un exemple de vérification de propositions du premier type :

```
13 ?- is_correct_pro1(david,all(aEcrit,auteur)).
true.

14 ?- is_correct_pro1(david,all(aEcrit,something)).

Le concept donne est incorrect
false.

15 ?- is_correct_pro1(X,all(aEcrit,something)).

L'instance donnee est incorrecte
Le concept donne est incorrect
false.

16 ?- is_correct_pro1(livre,all(aEcrit,auteur)).

L'instance donnee est incorrecte
false.
```

**Figure 14:** Test du prédicat *is_correct_pro1*

Et voici un exemple de vérification de propositions du deuxième type :

```
5 ?- is_correct_pro2(david,personne).

Le concept1 donne est incorrect
false.

6 ?- is_correct_pro2(auteur,and(personne,objet)).
true.

7 ?- is_correct_pro2(auteur,and(personne,obj)).

Le concept2 donne est incorrect
false.

8 ?- is_correct_pro2(vinci,and(personne,some(hello,all))).

Le concept1 donne est incorrect

Le concept2 donne est incorrect
false.
```

**Figure 15:** Test du prédicat *is_correct_pro2*

## 4.3 Transformation des concepts génériques par leurs définitions

Avant d'ajouter la forme normale négative de la négation de la proposition saisie, nous devons -si nécessaire- la transformer et transformer toutes les propositions de la liste des assertions de concepts contenant des concepts complexes. Pour cela, nous utilisons les prédicats suivants:

```
/*tranformer un concept complexe par sa definition */

transform_concept(_,Concept,Concept) :-  is_concept_atom(Concept).
transform_concept(Tbox,Concept,X) :- is_concept_gen(Concept), member((Concept,X),Tbox).
transform_concept(Tbox,not(Concept),not(X)) :- transform_concept(Tbox,Concept,X),!.

transform_concept(Tbox,and(Concept1,Concept2),and(X1,X2)) :- transform_concept(Tbox,Concept1,X1),
                                                             transform_concept(Tbox,Concept2,X2),!.

transform_concept(Tbox,or(Concept1,Concept2),or(X1,X2)) :- transform_concept(Tbox,Concept1,X1),
                                                           transform_concept(Tbox,Concept2,X2),!.

transform_concept(Tbox,all(R,Concept),all(R,X)) :- role(R), transform_concept(Tbox,Concept,X),!.
transform_concept(Tbox,some(R,Concept),some(R,X)) :- role(R), transform_concept(Tbox,Concept,X),!.

/* tranformer tous les  concepts complexes de l ABI  par leurs definitions */

transform_all(_,[],[]).
transform_all(Tbox,[(I,C)|Q],[(I,Prop_atomique)|Abi1]) :- transform_concept(Tbox,C,Prop_atomique),
                                                          transform_all(Tbox,Q,Abi1).
```

**Figure 16:** Prédicat de transformation de concepts par leurs définitions

Voici un exemple d'exécution du prédicat *transform_concept*:

```
2 ?- tbox(Tbox),transform_concept(Tbox,and(auteur,parent),Concept).
Tbox = [(auteur, and(personne, some(aEcrit, livre))),  (editeur, and(personne, and(not(some(aEcrit, livre)), some(aEdite, livre)))),  (par
ent, and(personne, some(aEnfant, anything))),  (sculpteur, and(personne, some(aCree, sculpture)))],
Concept = and(and(personne, some(aEcrit, livre)), and(personne, some(aEnfant, anything))).
```

**Figure 17:** Test du prédicat *transform_concept*

Le concept « auteur ⊓ parent » devient alors « ((personne ⊓ ∃aEcrit.livre) ⊓ (personne ⊓ ∃aEnfant.⊤))».

# 5 Partie 3 : Démonstration de la proposition

Cette troisième partie étant la dernière, c'est le coeur de notre démonstrateur puisque c'est à ce niveau que la résolution a lieu. Le prédicat *troisieme_etape* permet de lancer le processus de résolution :

```
/* TROISIEME ETAPE CALL */

troisieme_etape(Abi,Abr) :-

    tri_Abox(Abi,Lie,Lpt,Li,Lu,Ls),

    not(resolution(Lie,Lpt,Li,Lu,Ls,Abr)),

    nl,write('La proposition donnee en entree est valide').
```

**Figure 18:** Illustration du prédicat de lancement de la résolution

- ***tri_Abox*** : répartit les assertions de la Abox dans une des cinq listes selon leurs types
- ***resolution*** : c'est le prédicat qui lance la résolution à partir de la Abox triée

Nous détaillerons ces prédicats dans les sous-sections à suivre.

## 5.1 Tri de la Abox

Le prédicat *tri_Abox* prend en entrée la liste Abi dont on parcourt chaque assertion que l'on répartit selon son type dans l'une des cinq listes retournées. Cela donne le prédicat suivant:

```
/* je met chaque element de ABI dans la bonne liste */

tri_Abox([],[],[],[],[],[]).

tri_Abox([(I,some(R,C))|Q],[(I,some(R,C))|Lie],Lpt,Li,Lu,Ls) :- tri_Abox(Q,Lie,Lpt,Li,Lu,Ls),!.
tri_Abox([(I,all(R,C))|Q],Lie,[(I,all(R,C))|Lpt],Li,Lu,Ls) :- tri_Abox(Q,Lie,Lpt,Li,Lu,Ls),!.
tri_Abox([(I,and(C1,C2))|Q],Lie,Lpt,[(I,and(C1,C2))|Li],Lu,Ls) :- tri_Abox(Q,Lie,Lpt,Li,Lu,Ls),!.
tri_Abox([(I,or(C1,C2))|Q],Lie,Lpt,Li,[(I,or(C1,C2))|Lu],Ls) :- tri_Abox(Q,Lie,Lpt,Li,Lu,Ls),!.
tri_Abox([(I,C)|Q],Lie,Lpt,Li,Lu,[(I,C)|Ls]) :- tri_Abox(Q,Lie,Lpt,Li,Lu,Ls),!.
tri_Abox([(I,not(C))|Q],Lie,Lpt,Li,Lu,[(I,not(C))|Ls]) :- tri_Abox(Q,Lie,Lpt,Li,Lu,Ls),!.
```

**Figure 19:** Illustration du prédicat *tri_Abox*

Voici quelques tests du prédicat *tri_Abox* :

```
13 ?- abi(Abi), tri_Abox(Abi,Lie,Lpt,Li,Lu,Ls).
Abi = Ls, Ls = [(david,sculpture),(joconde,objet),(michelAnge,personne),(sonnets,livre),(vinci,personne)],
Lie = Lpt, Lpt = Li, Li = Lu, Lu = [].

14 ?- abi(Abii),concat(Abii,[(vinci,some(aCree,sculpture))],[(joconde,or(livre,sculpture))],[(michelAnge,all(aCree,sculpture))],[(vinci,and(pers
onne,objet))],Abi), tri_Abox(Abi,Lie,Lpt,Li,Lu,Ls).
Abii = Ls, Ls = [(david,sculpture),(joconde,objet),(michelAnge,personne),(sonnets,livre),(vinci,personne)],
Abi = [(david,sculpture),(joconde,objet),(michelAnge,personne),(sonnets,livre),(vinci,personne),(vinci,some(aCree,sculpture)),(joconde,or(
livre,sculpture)),(michelAnge,all(aCree,sculpture)),(vinci,and(personne,objet))],
Lie = [(vinci,some(aCree,sculpture))],
Lpt = [(michelAnge,all(aCree,sculpture))],
Li = [(vinci,and(personne,objet))],
Lu = [(joconde,or(livre,sculpture))].
```

**Figure 20:** Test du prédicat *tri_Abox*

Le premier exemple illustre l'état initial de notre TBox (avant la saisie d'une proposition) et en fonction de notre base, les listes Lie, Lpt, Li et Lu sont vides et la liste Ls reçoit le contenu de Abi.

Dans le deuxième exemple, nous ajoutons quatre nouvelles assertions à la Abox avant de la trier. On voit bien qu'après tri, les assertions sont placées dans leurs listes respectives (en fonction de leurs types).

## 5.2 Processus de résolution

La résolution est effectuée à travers le prédicat *«resolution»* qui prend en entrée toutes les listes formant notre Abox à savoir Lie,Lpt,Li,Lu,Ls et Abr :

```
/* Partie resolution */
resolution([],[],[],[],Ls,_):- member(_,Ls),test_clash(Ls),nl,
                               write('La proposition donnee en entree n\'est pas valide').

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

**Figure 21:** Illustration du prédicat *resolution*

- Si l'une des listes contient au moins une assertion, on cherchera à faire évoluer l'arbre de résolution en appliquant une des quatre règles définies (la règle $\sqcap$ , la règle $\sqcup$ , la règle $\forall$ et la règle $\exists$) aux assertions de la liste correspondante (par exemple si la liste Lie contient au moins une assertion, on cherchera à appliquer la règle $\exists$)
- si toutes les listes pouvant nécessiter l'application d'une règle ( les listes Lie, Lpt,Li et Lu ) sont vides et s'il n'y pas de clash alors la résolution a échoué (puisque l'on trouve une feuille ouverte)

**Afin d'éviter une redondance dans les tests, le test du prédicat *«resolution»* se fera dans la section 6 à travers divers exemples mais aussi lors de la définition et du test des prédicats auxquels il fait appel.**

### 5.2.1 Règle $\sqcap$

Comme l'indique la figure ci-dessous, le prédicat *«transformation_and»* enlève l'assertion en tête de la liste Li à laquelle il applique la Règle $\sqcap$ à travers le prédicat *«evolue»* et affiche

l'évolution du noeud en conséquence. Les assertions de concepts sont ensuite toutes regroupées en une liste pour vérifier si l'application de la règle ⊓ a généré un clash.

Dans le cas où il y a un clash, cette branche de l'arbre est abandonnée. Sinon, on continue la résolution en rappelant le prédicat *«resolution»*.

```
/* transformation ET */
transformation_and(Lie,Lpt,Li,Lu,Ls,Abr) :-

    /* enleve la tete de liste de Li et retourne le reste de la liste dans Q*/
    enleve((I,and(C1,C2)),Li,Q),

    /* ajout de des nouvelles assertions SI elles existent */
    evolue((I,C1), Lie, Lpt, Q, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1),
    evolue((I,C2), Lie1, Lpt1, Li1, Lu1, Ls1, Lie2, Lpt2, Li2, Lu2, Ls2),


    affiche_evolution_Abox(Ls, Lie, Lpt, Li, Lu, Abr, Ls2, Lie2,
    Lpt2, Li2, Lu2, Abr),

    /* test de clash */
    flatten([Lie2, Lpt2, Li2, Lu2, Ls2],Y),

    test_clash(Y),

    resolution(Lie2,Lpt2,Li2,Lu2,Ls2,Abr)
.
```

**Figure 22:** Illustration du prédicat *«transformation_and»*

Voici un test de la règle ⊓

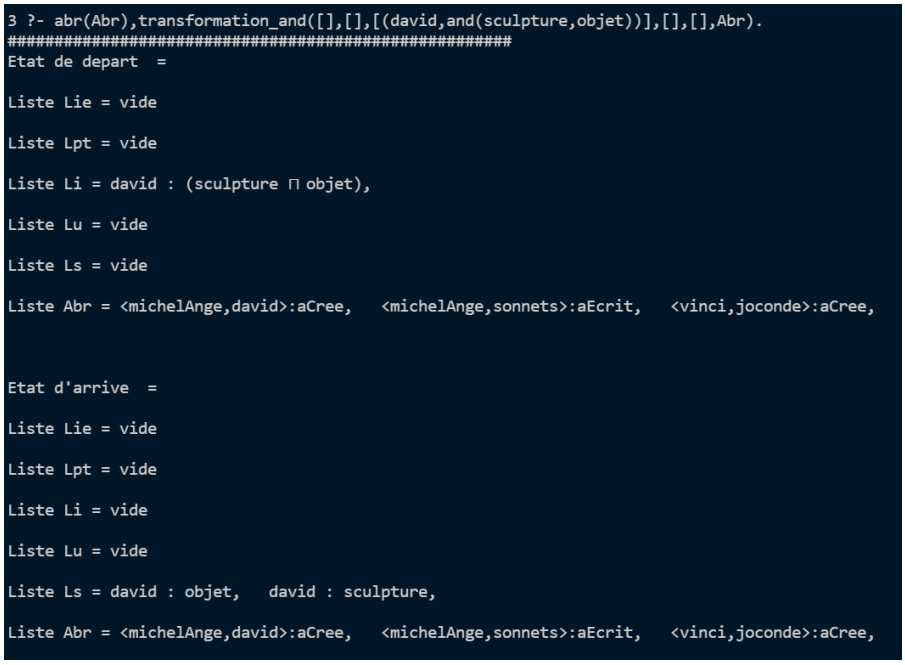


**Figure 23:** Test du prédicat *«transformation_and»*

L'assertion «david : (sculpture ⊓ objet)» donne lieu à deux nouvelles assertions « david : sculpture» , «david,objet» que l'on ajoute au nouveau noeud généré.

### 5.2.2 Règle ⊔

La règle ⊔ suit le même procédé au détail près que le prédicat *«transformation_or»* permet de générer deux noeuds et pour chacun d'eux on ajoute différentes assertions.

Lorsque l'on génère le premier noeud, on explore sa branche jusqu'à ce que l'on rencontre un clash puis on passe à la deuxième branche où l'on génère le deuxième noeud.

Si la branche du premier noeud donne une feuille ouverte, la résolution s'arrête et on n'explore pas la branche contenant le deuxième noeud généré par la règle.

```
/* tranformation or */
transformation_or(Lie,Lpt,Li,Lu,Ls,Abr) :-

    /* enleve la tete de liste de Lu et retourne le reste de la liste dans Q*/
    enleve((I,or(C1,_)),Lu,Q),
    /* creation de la 1ere branche */
    evolue((I,C1),Lie, Lpt, Li,Q, Ls, Lie1, Lpt1, Li1, Lu1, Ls1),
    affiche_evolution_Abox(Ls, Lie, Lpt, Li, Lu, Abr, Ls1, Lie1,
    Lpt1, Li1, Lu1, Abr),
    /* test de clash */
    flatten([Lie1, Lpt1, Li1, Lu1, Ls1],Y),
    test_clash(Y),
    resolution(Lie1,Lpt1,Li1,Lu1,Ls1,Abr).

transformation_or(Lie,Lpt,Li,Lu,Ls,Abr) :-
    enleve((I,or(_,C2)),Lu,Q),
    /* Creation de la seconde branche */
    evolue((I,C2),Lie, Lpt, Li,Q,  Ls, Lie2, Lpt2, Li2, Lu2, Ls2),
    affiche_evolution_Abox(Ls, Lie, Lpt, Li, Lu, Abr, Ls2, Lie2,
    Lpt2, Li2, Lu2, Abr),
    /* test de clash */
    flatten([Lie2, Lpt2, Li2, Lu2, Ls2],Y1),
    test_clash(Y1),
    resolution(Lie2,Lpt2,Li2,Lu2,Ls2,Abr)
.
```

**Figure 24:** Illustration du prédicat *«transformation_or»*

Voici un test de la règle ⊔ :

```
18 ?- abi(Abii),concat(Abii,[(david,or(not(sculpture),livre))],Abi), tri_Abox(Abi,Lie,Lpt,Li,Lu,Ls),transformation_or(Lie,Lpt,Lie,Lu,Ls,Ab
r).
```

**Figure 25:** Test du prédicat *«transformation_or»*

Nous avons récupéré la Abox de notre base à laquelle nous avons ajouté l'assertion « david: ( ¬ sculpture ⊔ livre)»

A l'exécution du prédicat *«transformation_or»* nous obtenons les résultats suivants:

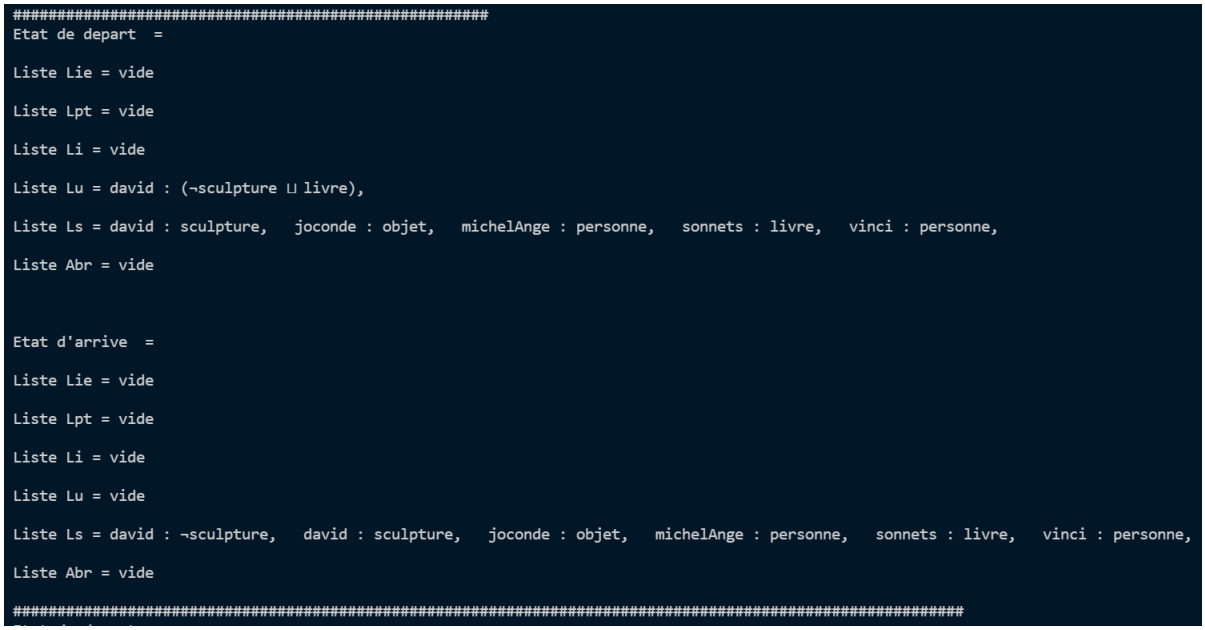


**Figure 26:** Premier noeud résultant du test du prédicat *«transformation_or»*

- L'état de départ étant le noeud racine de l'arbre de résolution, il contient l'assertion « david: ( ¬ sculpture ⊔ livre)» ajoutée.
- En appliquant la règle ⊔, un premier noeud est généré (état d'arrivée dans la figure) contenant la nouvelle assertion « david: ( ¬ sculpture)»
- L'assertion ajoutée donne lieu à un clash, la branche est donc abandonnée et l'on passe à la branche contenant le deuxième noeud généré par la règle ⊔ :

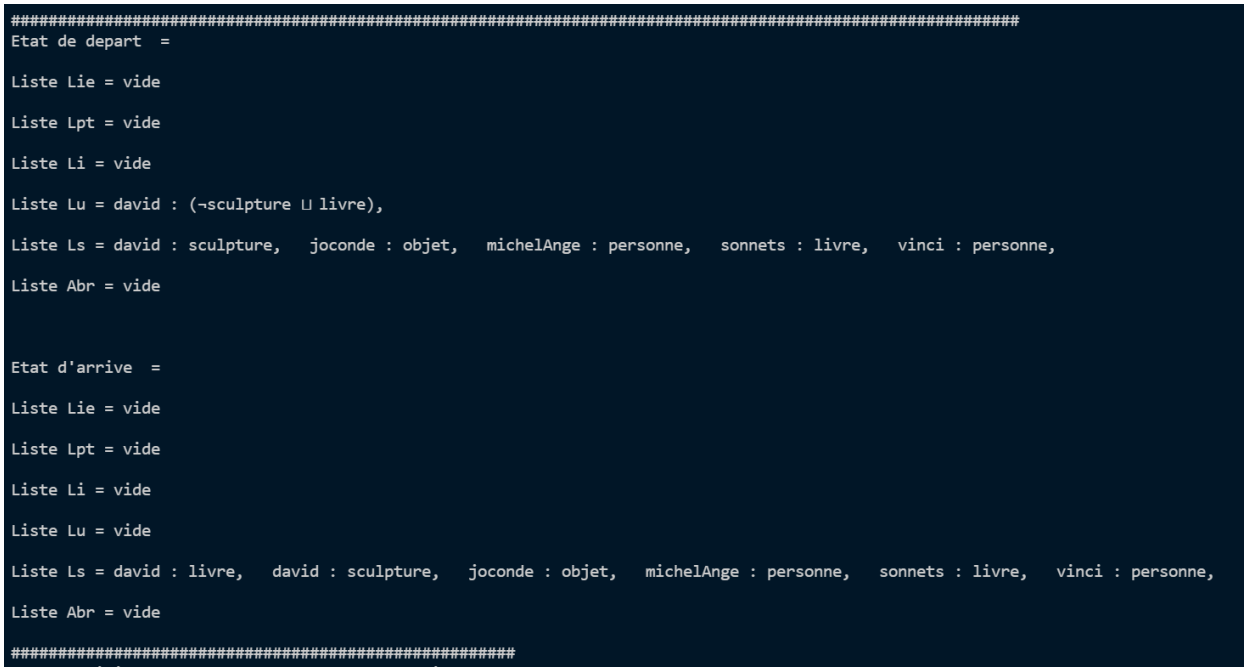


**Figure 27:** Deuxième noeud résultant du test du prédicat *«transformation_or»*

La nouvelle assertion ajoutée est bien évidemment l'assertion « david: livre».

### 5.2.3 Règle ∀

Le prédicat *«deduction_all»* enlève l'assertion «∀R.C» en tête de la liste Lpt et récupère toutes déductions possibles à partir de la liste des assertions de rôle Abr à travers le prédicat *«get_all_-new_elts»*. Ces nouvelles assertions seront ajoutées une à une dans le nouveau noeud généré grâce au prédicat *«evolue_all»*.

La suite de la résolution est la même que pour les deux précédentes règles.

```
/* Application de la regle de qlq soit */
deduction_all(Lie,Lpt,Li,Lu,Ls,Abr) :-

    /* enleve la tete de liste de Lpt */
    enleve((I,all(R,C)),Lpt,Q),

    /* reecupere toutes les assertions I2 : C possibles */
    get_all_new_elts((I,all(R,C)),Abr,New_assertions),

    evolue_all(New_assertions, Lie, Q, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1),


    affiche_evolution_Abox(Ls, Lie, Lpt, Li, Lu, Abr, Ls1, Lie1,
    Lpt1, Li1, Lu1, Abr),

     /* je met toutes les listes ensemble */
    flatten([Lie1, Lpt1, Li1, Lu1, Ls1],Y),


    /* test de clash */
    test_clash(Y),

    resolution(Lie1,Lpt1,Li1,Lu1,Ls1,Abr)
.
```

**Figure 28:** Illustration du prédicat *«deduction_all»*

Voici un test de la règle ∀

```
2 ?- deduction_all([],[(vinci,all(aCree,sculpture))],[],[],[],[(vinci,joconde,aCree),(michelAnge,david,aCree),(vinci,monalisa,aCree)]).
######################################################
Etat de depart  =

Liste Lie = vide

Liste Lpt = vinci : ∀aCree.sculpture,

Liste Li = vide

Liste Lu = vide

Liste Ls = vide

Liste Abr = <vinci,joconde>:aCree,   <michelAnge,david>:aCree,   <vinci,monalisa>:aCree,


Etat d'arrive  =

Liste Lie = vide

Liste Lpt = vide

Liste Li = vide

Liste Lu = vide

Liste Ls = monalisa : sculpture,   joconde : sculpture,

Liste Abr = <vinci,joconde>:aCree,   <michelAnge,david>:aCree,   <vinci,monalisa>:aCree,
```

**Figure 29:** Test du prédicat *«deduction_all»*

Au départ la liste Lpt contient l'assertion «vinci : ∀aCree.sculpture» et à l'execution du prédicat nous déduisons les deux nouvelles assertions attendues «monalisa : sculpture» et «joconde : sculpture».

### 5.2.4 Règle ∃

Comme pour les autres prédicats, le prédicat *«complete_some»* enlève la tête de la liste correspondante aux assertions existentielles (la liste Lie) et rajoute une nouvelle assertion dans la bonne liste, affiche l'évolution de la branche et teste s'il y a un clash afin de soit abandonner la branche ou bien de continuer la résolution dessus.

```
/* Application de la regle d'il existe */
complete_some(Lie,Lpt,Li,Lu,Ls,Abr) :-

    /*enleve la tete de liste de Lie*/
    enleve((I,some(R,C)),Lie,Q),

    /* ajout de des nouvelles assertions de concepts*/
    genere(B),

    evolue((B,C), Q, Lpt, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1),

    /* ajout de des nouvelles assertions de roles */
    concat([(I,B,R)],Abr,Abr1),

    affiche_evolution_Abox(Ls, Lie, Lpt, Li, Lu, Abr, Ls1, Lie1,
    Lpt1, Li1, Lu1, Abr1),

    /* je met toutes les listes ensemble */
    flatten([Lie1, Lpt1, Li1, Lu1, Ls1],Y),

    /* test de clash */
    test_clash(Y),

    resolution(Lie1,Lpt1,Li1,Lu1,Ls1,Abr1)
.
```

**Figure 30:** Illustration du prédicat *«complete_some»*

Voici un test du prédicat *«complete_some»*:

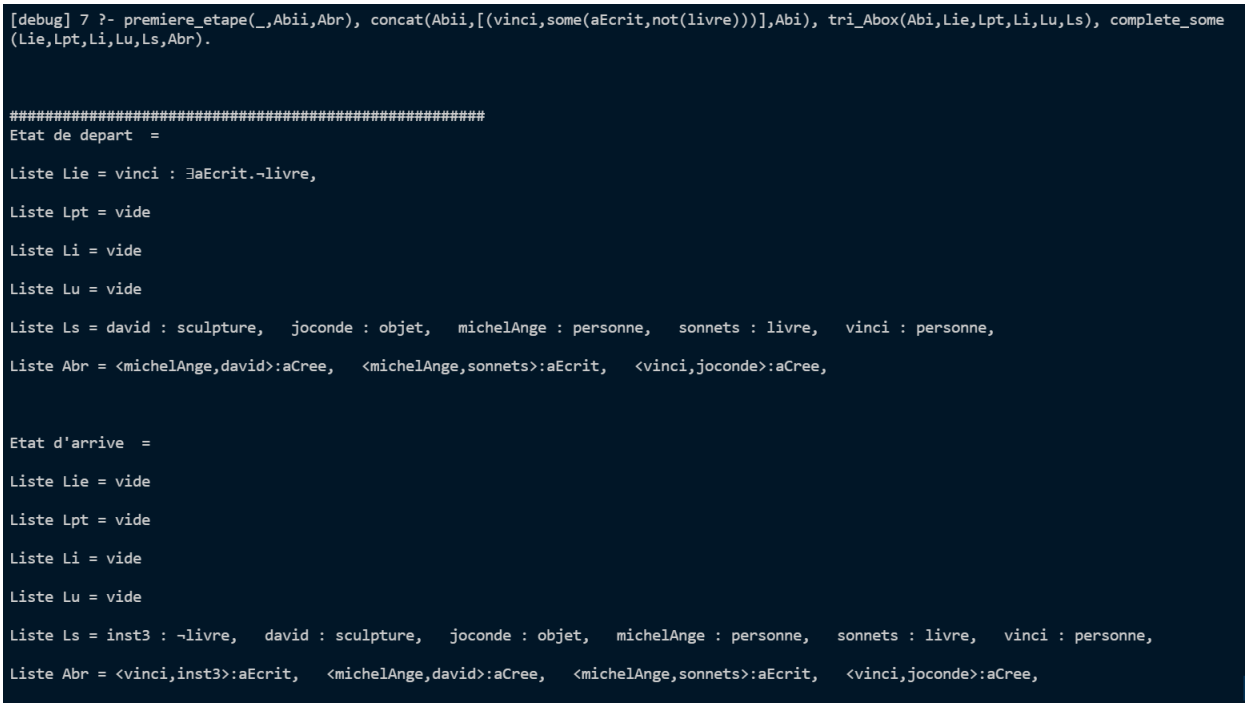


**Figure 31:** Test du prédicat *«complete_some»*

Nous avons réecupéré la Abox de la base à laquelle nous avons ajouté l'assertion «∃aEcrit.¬livre». Nous avons ensuite trié la Abox puis fait appel au prédicat *«complete_some»*.

Ce prédicat crée un nouveau noeud auquel il ajoute la nouvelle assertion de concepts «inst3:¬livre» et la nouvelle assertion de rôles «<vinci,inst3>: aEcrit».

*«inst3»* étant l'identificateur d'un nouvel objet.

### 5.2.5 Évolution de la Abox

Le prédicat *«evolue»* permet d'ajouter la nouvelle insertion issue d'une des règles dans l'une des liste des assertions de concepts, comme le montre la figure suivante:

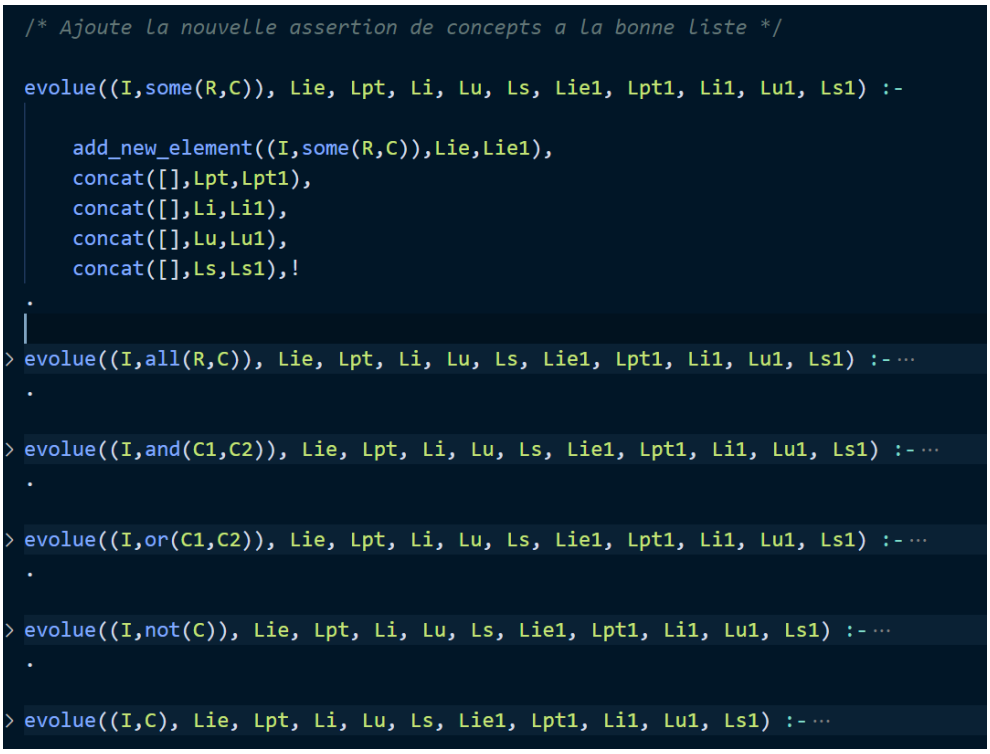


**Figure 32:** Illustration du prédicat *«evolue»*

Si l'on considère un ensemble de listes Lie,Li,Ls,Lpt,Lu que l'on pré-remplit, à chaque fois que l'on souhaitera ajouter une nouvelle assertion (déduite des règles données), elle sera ajoutée a la liste adéquate, comme le montrent nos tests :

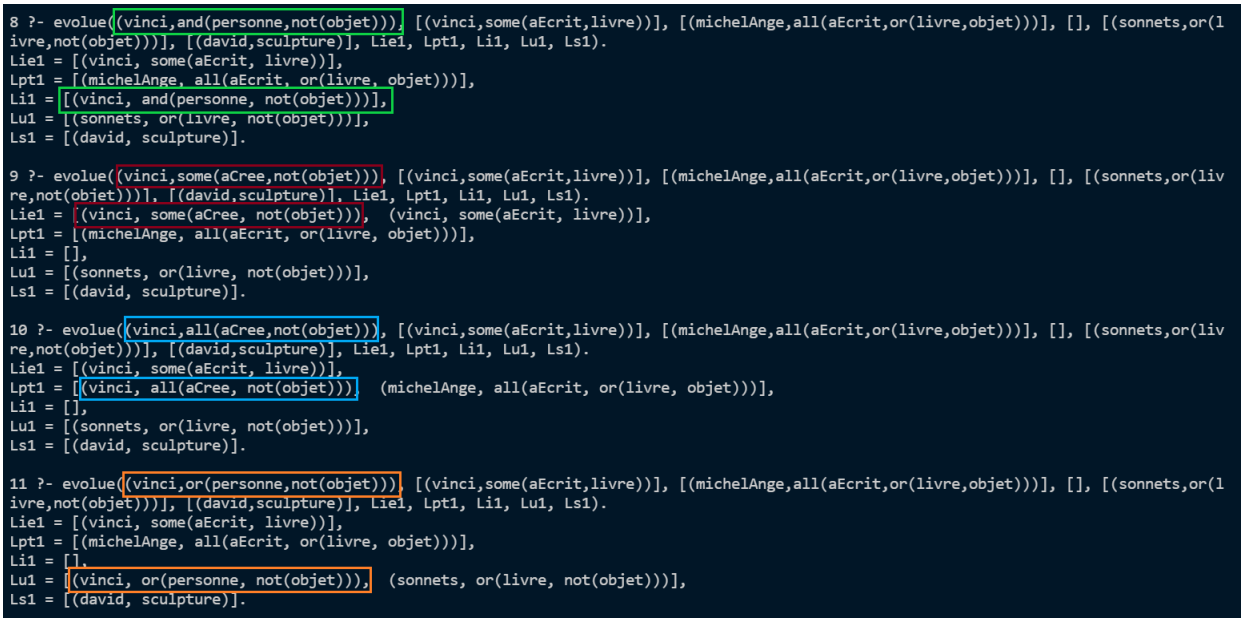


**Figure 33:** Test du prédicat *«evolue»*

### 5.2.6 Affichage de l'évolution de la Abox

Le prédicat *«affiche_evolution_Abox»* affiche toutes les listes d'assertions de concepts et la liste d'assertion de rôles d'un état de départ vers un état d'arrivée.

```
affiche_evolution_Abox(Ls1, Lie1, Lpt1, Li1, Lu1, Abr1, Ls2, Lie2,Lpt2, Li2, Lu2, Abr2) :-
    write('#####################################################'),
    nl,write('Etat de depart  = '),nl,nl,

    write('Liste Lie = '),
    afficheList(Lie1),nl,nl,

    write('Liste Lpt = '),
    afficheList(Lpt1),nl,nl,

    write('Liste Li = '),
    afficheList(Li1),nl,nl,

    write('Liste Lu = '),
    afficheList(Lu1),nl,nl,

    write('Liste Ls = '),
    afficheList(Ls1),nl,nl,

    write('Liste Abr = '),
    afficheList(Abr1),nl,nl,

    nl,nl,write('Etat d\'arrive  = '),nl,nl,
```

**Figure 34:** Illustration du prédicat *«affiche_evolution_Abox»*

Le prédicat *«afficheList»* sélectionne le type d'affichage selon que l'on affiche une liste d'assertion de rôles ou une liste d'assertion de concepts. Les prédicats appelés à ce niveau font, à leur tour, appel au principal prédicat d'affichage, le prédicat *«affiche»*, qui permet d'afficher une assertion tel que demandé dans l'énoncé du projet :

```
/* affiche un element en infixé -> ici il faut ajouter le !*/

affiche(not(Concept),Res) :- affiche(Concept,Res1),
                             atom_concat('\u00AC',Res1,Res),!.

affiche(and(Concept1,Concept2),Res) :- affiche(Concept1,Res1),
                                       affiche(Concept2,Res2),
                                       atom_concat(Res1,' \u2293 ',Res3),
                                       atom_concat('(',Res3,Res4),
                                       atom_concat(Res2,')',Res5),
                                       atom_concat(Res4,Res5,Res),!.

affiche(or(Concept1,Concept2),Res) :- affiche(Concept1,Res1),
                                      affiche(Concept2,Res2),
                                      atom_concat(Res1,' \u2294 ',Res3),
                                      atom_concat('(',Res3,Res4),
                                      atom_concat(Res2,')',Res5),
                                      atom_concat(Res4,Res5,Res),!.

affiche(all(R,Concept),Res) :- affiche(R,Res1),
                               affiche(Concept,Res2),
                               atom_concat('\u2200',Res1,Res3),
                               atom_concat('.',Res2,Res4),
                               atom_concat(Res3,Res4,Res),!.

affiche(some(R,Concept),Res) :- affiche(R,Res1),
                                affiche(Concept,Res2),
                                atom_concat('\u2203',Res1,Res3),
                                atom_concat('.',Res2,Res4),
                                atom_concat(Res3,Res4,Res),!.

/* affiche les concepts atomic en chaines de cars */
affiche(anything,'\u22A4').
affiche(nothing,'\u22A5').
affiche(Concept,Res) :- atom_string(Concept,Res).
```

**Figure 35:** Illustration du prédicat *«affiche»*

Et voici un exemple d'exécution du prédicat *«affiche»*:

```
11 ?- affiche(all(aEcrit,some(aCree,or(personne,objet))),X).
X = 'VaEcrit.EaCree.(personne ⊔ objet)'.
```

**Figure 36:** Test du prédicat *«affiche»*

Ainsi qu'un exemple d'exécution du prédicat *«affiche_evolution_Abox»*:

```
[debug] 10 ?- abr(Abr1),affiche_evolution_Abox([(david,not(sculpture))],[(vinci,all(aCree,sculpture))], [],[(sonnets,and(livre,objet))], [
], Abr1, [],[(michelAnge,some(aCree,joconde))],[], [], [(vinci,or(personne,objet)),(joconde,or(livre,sculpture))], Abr1).
######################################################
Etat de depart  =

Liste Lie = vinci : VaCree.sculpture,

Liste Lpt = vide

Liste Li = sonnets : (livre ⊓ objet),

Liste Lu = vide

Liste Ls = david : ¬sculpture,

Liste Abr = <michelAnge,david>:aCree,   <michelAnge,sonnets>:aEcrit,   <vinci,joconde>:aCree,


Etat d'arrive  =

Liste Lie = michelAnge : EaCree.joconde,

Liste Lpt = vide

Liste Li = vide

Liste Lu = vinci : (personne ⊔ objet),   joconde : (livre ⊔ sculpture),

Liste Ls = vide

Liste Abr = <michelAnge,david>:aCree,   <michelAnge,sonnets>:aEcrit,   <vinci,joconde>:aCree,

######################################################
```

**Figure 37:** Test du prédicat *«affiche_evolution_Abox»*

- Chaque état est représenté par un ensemble de listes
- Chaque liste est affichée en fonction de son contenu
- Pour indiquer qu'une liste est vide, on affiche le mot «vide»

# 6 Exemples de démonstration de propositions

Dans cette section, nous allons tester notre démonstrateur à travers plusieurs exemples.

Nous testons le démonstrateur sur la base de la Abox et de la Tbox de l'exercice 3 du TD4. Initialement, notre base contient les listes suivantes :

```
?- premiere_etape(Tbox,Abi,Abr).
Tbox = [(auteur, and(personne, some(aEcrit, livre))),  (editeur, and(personne, and(not(some(aEcrit, li
vre)), some(aEdite, livre)))),  (parent, and(personne, some(aEnfant, anything))),  (sculpteur, and(per
sonne, some(aCree, sculpture)))],
Abi = [(david, sculpture),  (joconde, objet),  (michelAnge, personne),  (sonnets, livre),  (vinci, per
sonne)],
Abr = [(michelAnge, david, aCree),  (michelAnge, sonnets, aEcrit),  (vinci, joconde, aCree)].
```

**Figure 38:** État des listes avant démonstration

## 6.1 Exemples de démonstration de la proposition de type : I:C

Nous démontrerons d'abord deux propositions simples puis nous passerons à des propositions plus complexes.

### 6.1.1 Proposition *«david : sculpture»*

```
4 ?- program.

Entrez le numero du type de proposition que vous voulez demontrer :

1 - Une instance donnee appartient a un concept donne.

2 - Deux concepts n'ont pas d'elements en commun.

|: 1.

Entrez l'instance :
|: david.

Entrez le concept :
|: sculpture.

La proposition donnee en entree est valide
true.
```

**Figure 39:** Démonstration de la proposition *«david : sculpture»*

La racine de l'arbre de résolution regroupe la Abox initiale (voir la figure 38) ainsi que la négation de la proposition introduite «david : ¬ sculpture» ( qui ne sera pas transformée vu que sculpture est un concept atomique) créant dés le départ un clash avec «david : sculpture» (se trouvant dans la Abox initiale).

L'arbre de résolution ne contient donc que la racine qui est une feuille fermée, donc la proposition est valide.

### 6.1.2 Proposition *«vinci : ¬ personne»*

Comme dans le premier exemple, nous avons dans la Abox initiale la proposition «vinci est une instance de personne». La négation de la proposition introduite donne également que «vinci est une instance de personne» ce qui ne génère aucun clash avec les propositions déjà présentes dans la Abox. La racine de l'arbre de résolution est donc une feuille non fermée. On en conclut que la proposition *«vinci : ¬ personne»* n'est pas valide, comme l'illustre l'exécution de notre démonstrateur:

```
5 ?- program.

Entrez le numero du type de proposition que vous voulez demontrer :

1 - Une instance donnee appartient a un concept donne.

2 - Deux concepts n'ont pas d'elements en commun.

|: 1.

Entrez l'instance :
|: vinci.

Entrez le concept :
|: not(personne).

La proposition donnee en entree n'est pas valide
false.
```

**Figure 40:** Démonstration de la proposition *«vinci : ¬ personne»*

### 6.1.3 Proposition *« david : ∀ aCree.(sculpteur ⊓ auteur)»*

Après exécution de notre démonstrateur:

```
2 ?- program.

Entrez le numero du type de proposition que vous voulez demontrer :

1 - Une instance donnee appartient a un concept donne.

2 - Deux concepts n'ont pas d'elements en commun.

|: 1.

Entrez l'instance :
|: david.

Entrez le concept :
|: all(aCree,and(sculpteur,auteur)).



######################################################
```

**Figure 41:** Démonstration de la proposition *« david : ∀ aCree.(sculpteur ⊓ auteur)»*

Nous obtenons les résultats suivants:

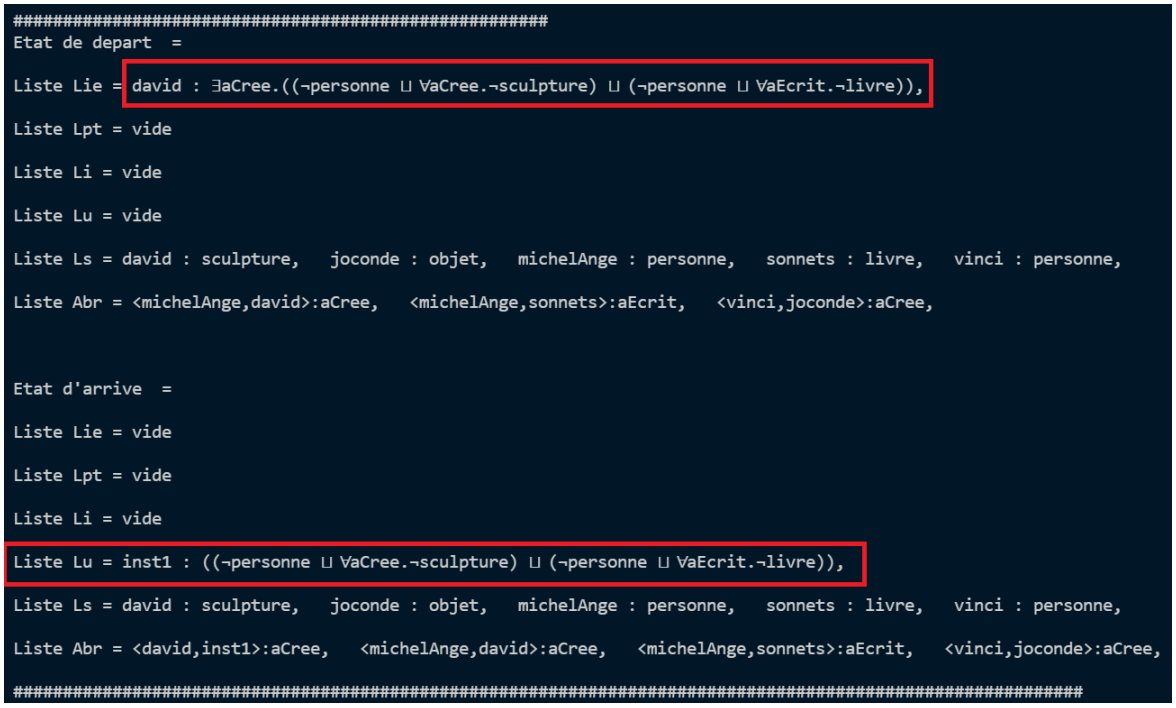


**Figure 42:** Évolution 1 de la résolution

- Le premier état de départ affiché est la racine de l'arbre de résolution
- On voit clairement qu'à cette première étape, la négation de la proposition a été correctement transformée et ajoutée à la liste adéquate (la liste Lie)
- L'arbre de résolution évolue en appliquant la règle ∃ (en exécutant le prédicat *«complete_some»*) et génère le nouveau noeud qui est l'état d'arrivée où la Abox à été mise à jour.

A la prochaine étape, l'état d'arrivée devient l'état de départ et l'on cherche à appliquer à nouveau une des règles définies. La règle ⊔ est appliquée et génère d'abord un premier noeud que le démonstrateur explore à la recherche de clashs **avant** de générer et d'explorer le deuxième noeud :

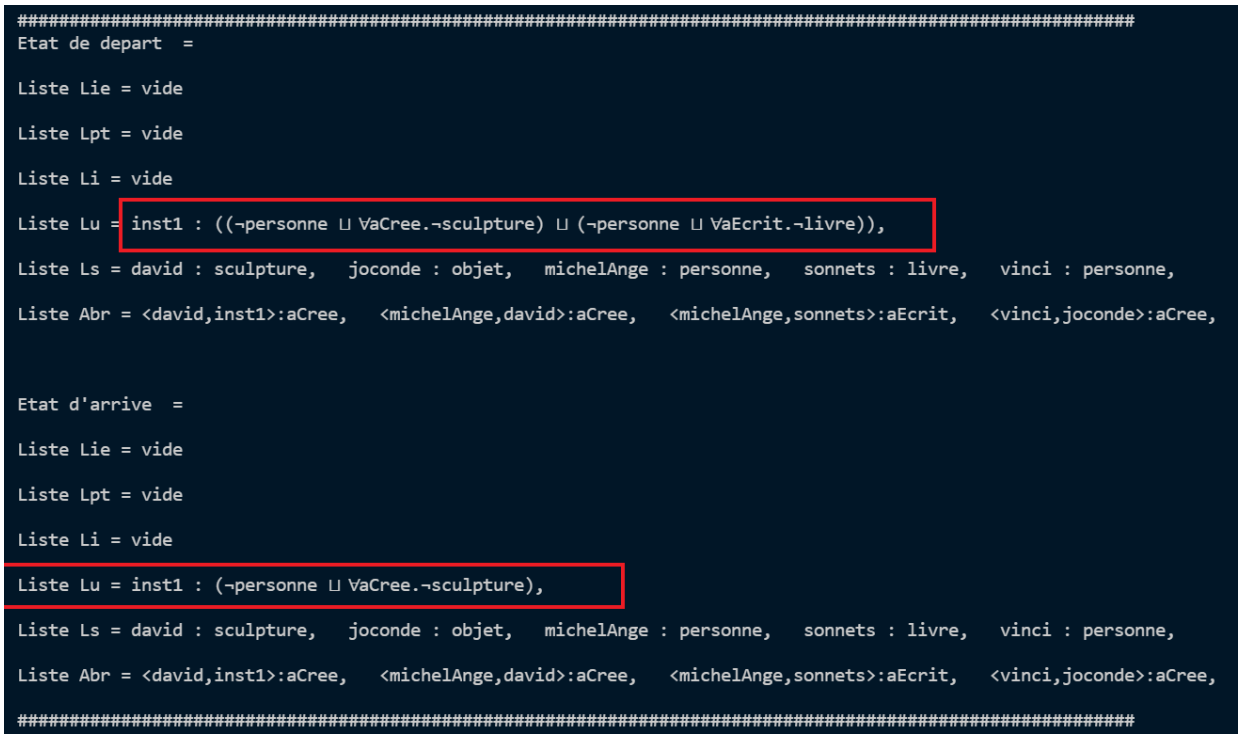


**Figure 43:** Évolution 2 de la résolution

A l'issue de cette étape, on ré-applique de la même manière la règle ⊔ qui donne lieu à un premier noeud :

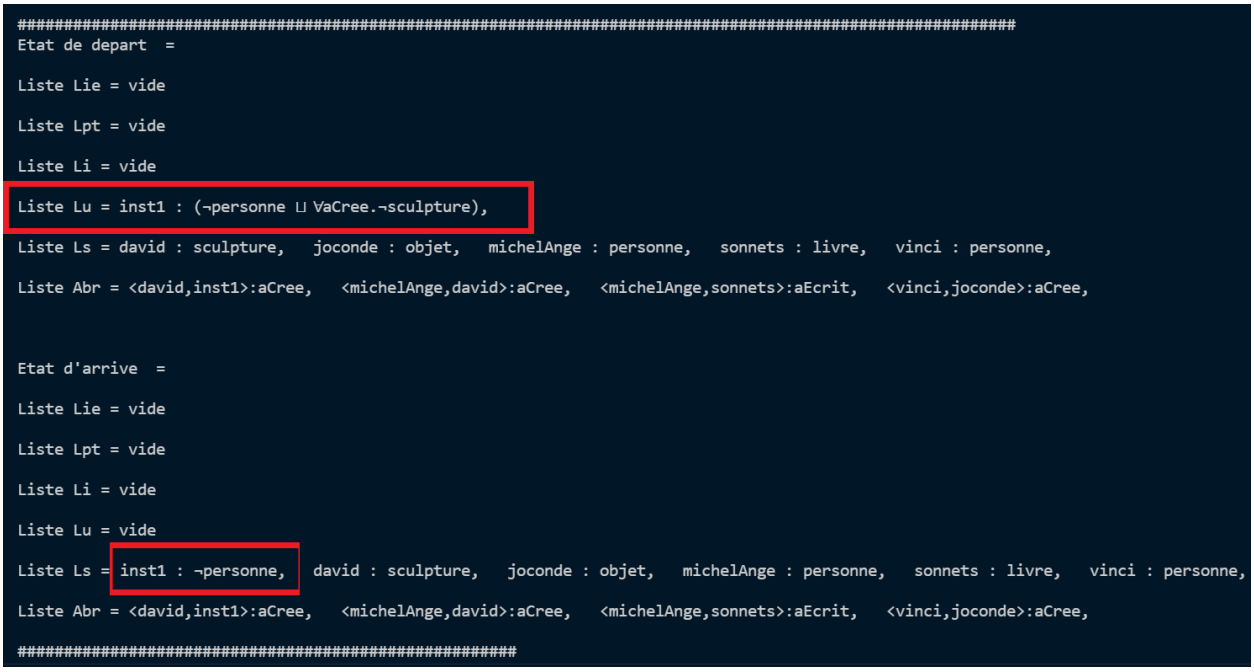


**Figure 44:** Évolution 3 de la résolution

Comme on le voit sur la figure ci-dessus, le nouveau noeud généré est une feuille ouverte, il est donc inutile de parcourir les deux noeuds restants de l'application des deux règles ⊔ précédentes, car on peut directement arrêter la résolution et conclure que la proposition donnée en entrée **n'est pas valide**.

L'arbre de résolution peut être illustré de la manière suivante (les noeuds explorés sont en rouge) :

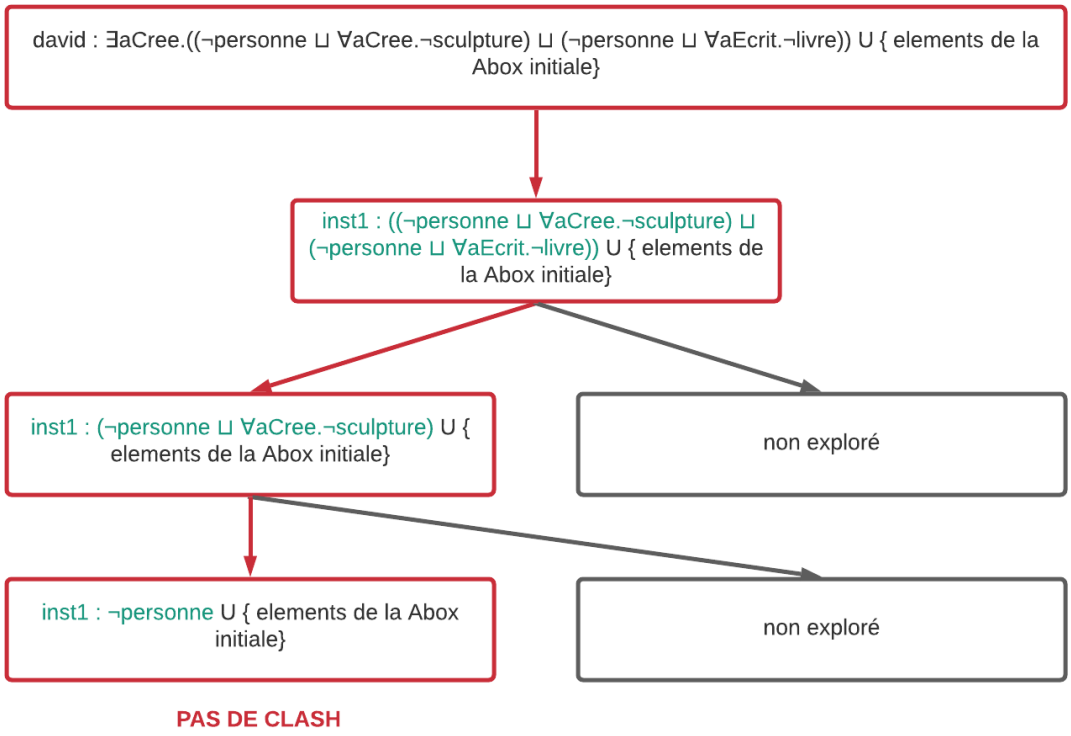


**Figure 45:** Illustration de l'évolution de l'arbre de résolution

### 6.1.4 Proposition « *michelAnge :(personne* ⊓ ∃*aCree.sculpture)* »

De la même manière nous allons procéder à la démonstration de la proposition « *michelAnge :(personne* ⊓ ∃*aCree.sculpture)* »:

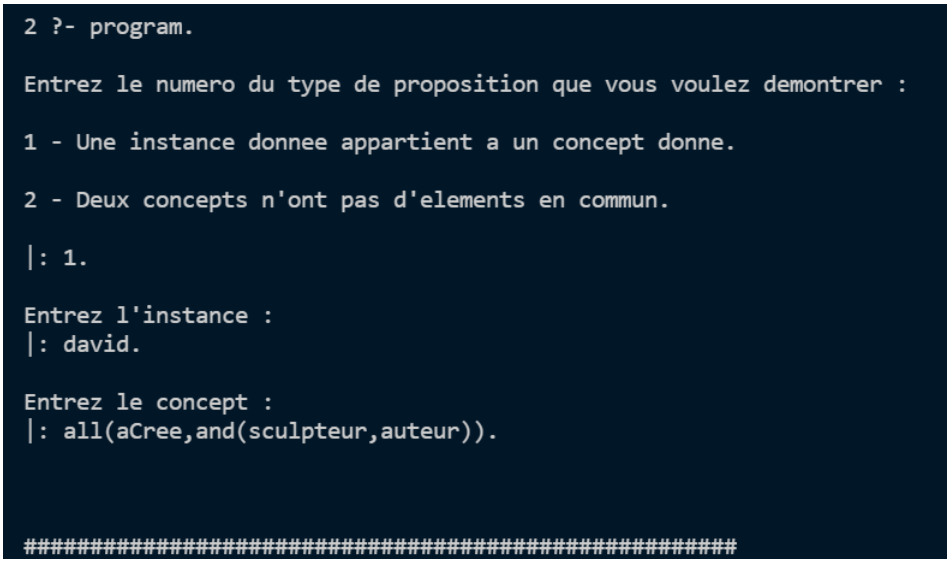


**Figure 46:** Démonstration de la proposition *« michelAnge :(personne ⊓ ∃aCree.sculpture) »*

Aprés que la négation de la proposition ait été transformée et ajoutée à la racine de l'arbre de résolution, on applique la règle ⊔ et en génère un premier noeud:

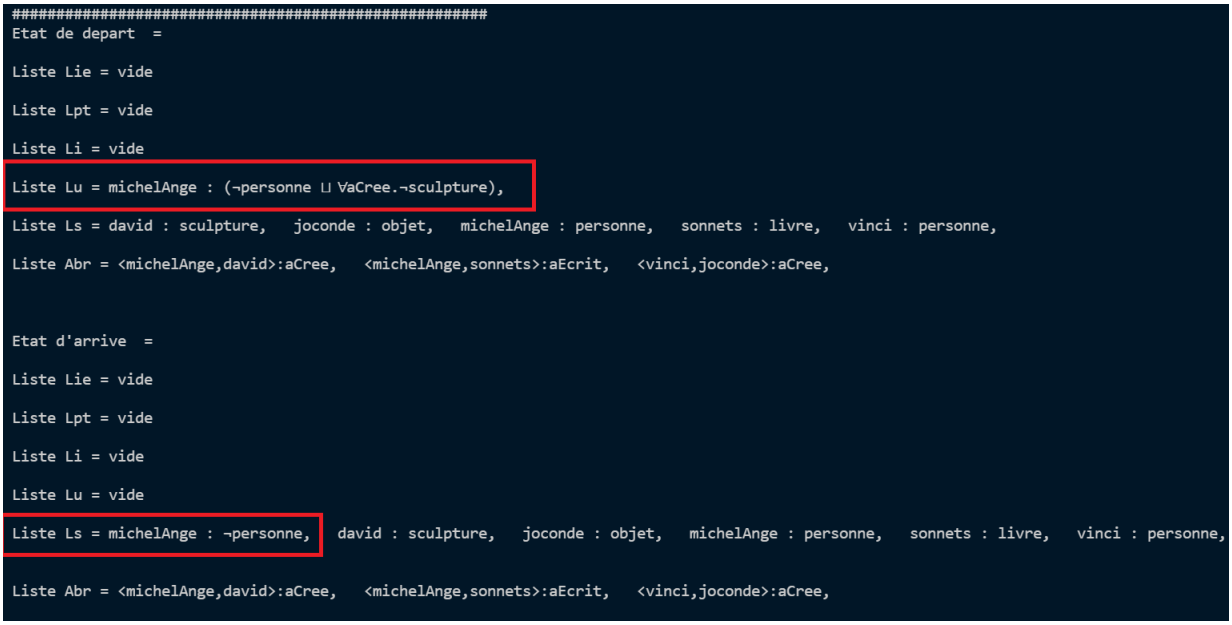


**Figure 47:** Évolution 1 de la résolution

Le noeud obtenu est une feuille fermée (car il y un clash), cette branche de l'arbre est donc abandonnée et l'on revient à l'exploration du deuxième noeud issu de la règle ⊔ :

```
#########################################################################################################
Etat de depart  =

Liste Lie = vide

Liste Lpt = vide

Liste Li = vide

Liste Lu = michelAnge : (¬personne ⊔ ∀aCree.¬sculpture),

Liste Ls = david : sculpture,   joconde : objet,   michelAnge : personne,   sonnets : livre,   vinci : personne,

Liste Abr = <michelAnge,david>:aCree,   <michelAnge,sonnets>:aEcrit,   <vinci,joconde>:aCree,


Etat d'arrive  =

Liste Lie = vide

Liste Lpt = michelAnge : ∀aCree.¬sculpture,

Liste Li = vide

Liste Lu = vide

Liste Ls = david : sculpture,   joconde : objet,   michelAnge : personne,   sonnets : livre,   vinci : personne,

Liste Abr = <michelAnge,david>:aCree,   <michelAnge,sonnets>:aEcrit,   <vinci,joconde>:aCree,
```

**Figure 48:** Évolution 2 de la résolution

On applique ensuite la règle ∀ comme le montre la figure ci-dessous :

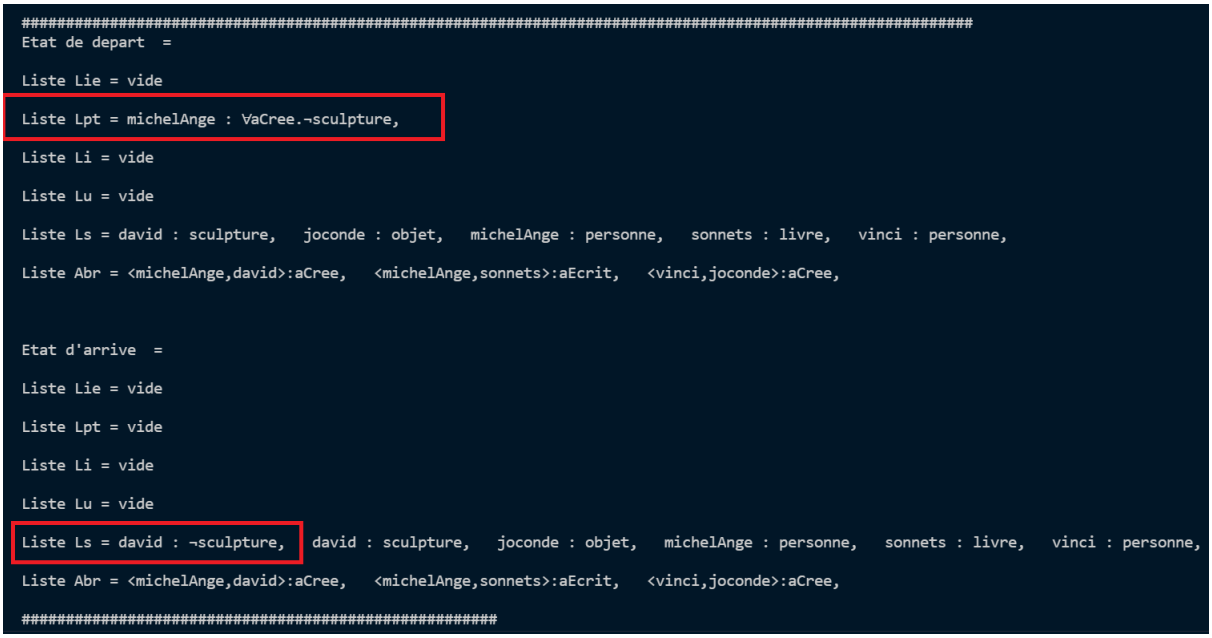


**Figure 49:** Évolution 3 de la résolution

Le nouveau noeud obtenu est une feuille générant un clash, la branche est donc abandonnée.

Toutes les feuilles de l'arbre sont fermées, le démonstrateur affiche que la proposition est valide:


**Figure 50:** Affichage du résultat de la démonstration

## 6.2 Exemple de démonstration de la proposition de type : $C_1 \sqcap C_2 \sqsubseteq \bot$

L'exemple le plus intuitif pour que ce type de propositions soient valides est de donner au démonstrateur un concept et sa négation qui est forcement toujours subsumée par le vide. Soit

donc le concept suivant : (∀aEcrit.personne ⊔ auteur) et sa négation et voici les résultats de la démonstration de la proposition de type 2 correspondante:

```
6 ?- program.

Entrez le numero du type de proposition que vous voulez demontrer :

1 - Une instance donnee appartient a un concept donne.

2 - Deux concepts n'ont pas d'elements en commun.

|: 2
.

Entrez le concept 1 :
|: or(all(aEcrit,personne),auteur).

Entrez le concept 2 :
|: not(or(all(aEcrit,personne),auteur)).
```

**Figure 51:** Exemple de démonstration de la proposition de type $C_1 \sqcap C_2 \sqsubseteq \bot$

En appliquant la règle ⊓, on génère un nouveau noeud qui est une feuille fermée :

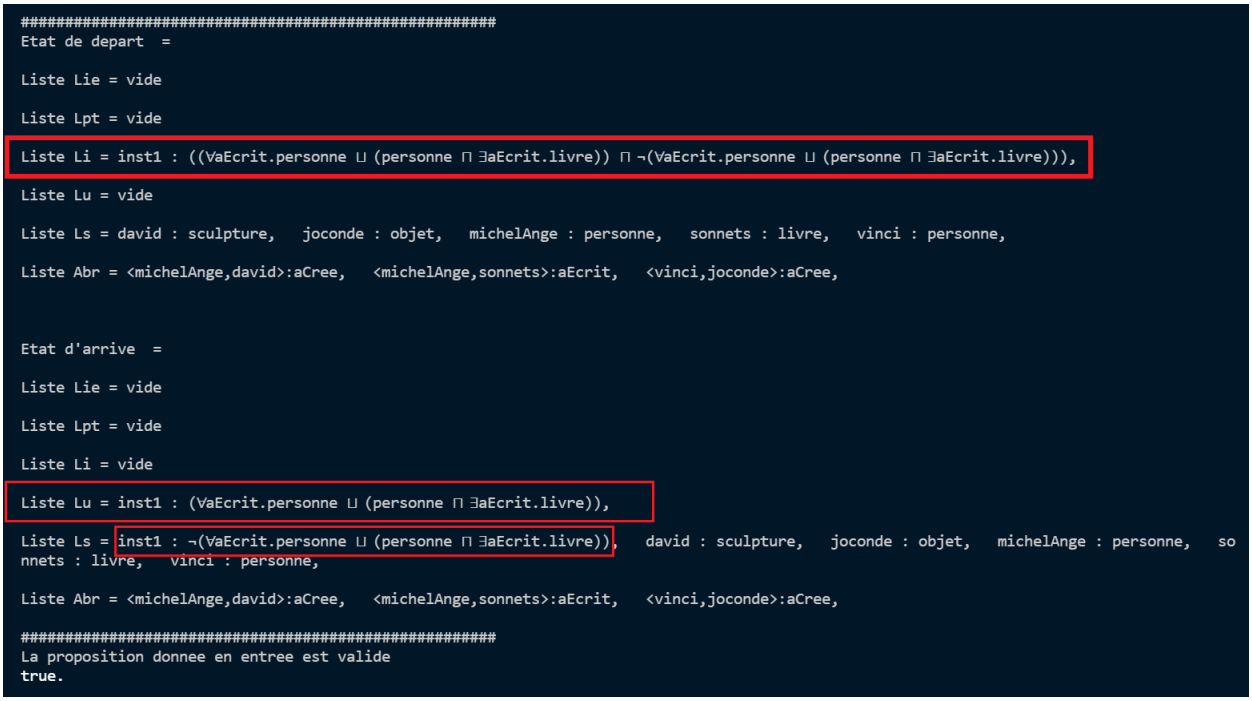


**Figure 52:** Évolution 1 de la démonstration $C_1 \sqcap C_2 \sqsubseteq \bot$

La seule branche existante étant abandonnée, la proposition est valide.