Projet LRC

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

Hocine Kadem 21309534

Neil Benahmed 21200977

Date : [19/12/2023]

Table des matières

1 Introduction

Le présent rapport rend compte du projet de l'UE Logique et Représentation des Connaissances. L'objectif de ce projet est de développer un démonstrateur basé sur l'algorithme des tableaux pour la logique de description ALC, en utilisant le langage Prolog.

Dans la première partie du projet, nous nous concentrons sur une étape préliminaire cruciale, qui consiste à vérifier la correction syntaxique et sémantique d'une Tbox et d'une Abox. Nous mettons également ces bases de connaissances en forme, afin de les utiliser comme référence pour les tests du démonstrateur. Cette étape garantit la fiabilité des données fournies au démonstrateur lorsqu'une proposition sera soumise à la démonstration.

La deuxième partie du projet concerne la saisie de la proposition à démontrer. Nous abordons les différents types de propositions à prouver et établissons les mécanismes pour acquérir la proposition à l'aide du démonstrateur. Cette étape est essentielle pour fournir une entrée appropriée au démonstrateur et poursuivre la démonstration.

La troisième partie constitue le cœur du démonstrateur, où nous implémentons l'algorithme de résolution basé sur la méthode des tableaux. Nous visons à démontrer l'insatisfiabilité de l'ensemble des assertions de la Abox étendue, que nous avons construite à partir de la proposition saisie précédemment. Nous construisons progressivement un arbre de résolution en appliquant les règles de l'algorithme des tableaux. Cet arbre de résolution nous permettra de déterminer la validité de la proposition soumise.

2 Description du programme

2.1 Execution du projet

Afin d'executer notre code. Veuillez charger le fichier programme.pl sur prolog (swipl) en utilisant la commande : ["programme.pl"]. Puis executer le main avec la commande main.

2.2 Découpage du code

Notre code est composé de plusieurs parties écrite dans un seul fichier (comme indiqué dans le forum du projet) qui travaillent ensemble pour implémenter le projet. Chaque partie a un objectif spécifique et contribue à la fonctionnalité globale du système.

2.2.1 partie1

Elle contient le code pour la première partie du projet. elle met en œuvre les prédicats nécessaires et la logique spécifique à cette partie, en traitant les fonctionnalités ou les exigences correspondantes.

2.2.2 partie2

Elle contient le code pour la deuxième partie du projet. elle se concentre sur la mise en œuvre des prédicats requis et de la logique spécifique à cette partie, en traitant les tâches ou les fonctionnalités pertinentes.

2.2.3 partie3

Elle contient le code pour la troisième partie du projet. Elle englobe les prédicats et la logique nécessaires pour répondre aux exigences ou fonctionnalités associées à cette partie.

2.2.4 utils

La partie utils contient le code des prédicats utilitaires auxquels nous avons fait appel dans les autres parties du projet. Elle fournit des fonctions ou des outils communs utilisés dans l'ensemble du système.

2.2.5 main

Représente le code principal qui exécute les trois parties du projet. Il orchestre l'interaction entre les différentes parties, appelant les prédicats appropriés et coordonnant la logique globale du système.

2.3 Description du code

2.3.1 Partie 1

La partie 1 consiste à vérifier la correction syntaxique et sémantique d'une Tbox et d'une Abox. Pour cela, nous avons mis en place plusieurs prédicat.

Vérification sémantique :

- Les prédicats `cname/1`, `concept/1`, `role/1`, et `instance/1` servent à définir les entités valides dans la logique ALC.
- Les prédicats `concept/1` définissent les règles pour vérifier si un concept est valide.
- Les prédicats `role/1` définissent les règles pour vérifier si un rôle est valide.

Vérification syntaxique :

- Le prédicat `verifTbox/1` vérifie la syntaxe de la Tbox (ensemble d'axiomes) en utilisant le prédicat `equiv/2` pour vérifier l'équivalence entre deux concepts.
- Le prédicat `verifAboxConcept/1` vérifie la syntaxe de l'Abox (ensemble d'assertions de concepts) en utilisant le prédicat `inst/2` pour vérifier l'instanciation d'un concept.
- Le prédicat `verifAboxRole/1` vérifie la syntaxe de l'Abox pour les assertions de rôles en utilisant
- le prédicat `instR/3` pour vérifier l'instanciation d'un rôle.

Vérification des concepts auto-référents :

- Le prédicat `nonCirclTbox/1` vérifie si une Tbox est non circulaire en utilisant le prédicat `autoref/2` pour détecter les auto-références.
- Le prédicat `autoref/2` vérifie si un concept est auto-référent en utilisant la récursion pour parcourir les expressions conceptuelles.

2.3.2 Partie 2

Dans cette partie du projet, nous nous sommes concentrés sur le traitement des propositions logiques de deux types différents. Nous avons mis en place des règles et des prédicats pour gérer ces propositions et les intégrer à notre système.

Traitement des propositions de type 1

Le premier type de proposition est représenté par la règle `acquisition_prop_type1/3`. Cette règle vise à traiter les propositions de la forme $I : C$, où I représente une instance et C un concept. Le processus de traitement comprend les étapes suivantes :

1. Lecture de l'entrée de la proposition :
   — L'utilisateur est invité à saisir l'instance I.
   — L'utilisateur est invité à saisir le concept C.
2. Transformations et vérifications :
   — Le concept C est transformé en une forme composée uniquement d'éléments atomiques à l'aide du prédicat `transAtom/2`.
   — Une mise en forme normale négative (NNF) est appliquée à la négation du concept transformé à l'aide du prédicat `nnf/2`.
   — Des vérifications sont effectuées pour s'assurer que l'instance est déclarée et que le concept est valide. Si ces vérifications échouent, un message d'erreur est affiché et l'utilisateur est invité à recommencer.
3. Ajout à l'Abox :
   — La paire (I, NCS) représentant la négation de l'instance de concept est ajoutée à la liste `Abi`, qui constitue notre Abox.

Traitement des propositions de type 2

Le deuxième type de proposition est géré par la règle `acquisition_prop_type2/3`. Cette règle traite les propositions de la forme $C1 \wedge C2 \sqsubseteq \bot$, où $C1$ et $C2$ sont des concepts. Le processus de traitement comprend les étapes suivantes :

1. Lecture de l'entrée de la proposition :
   — L'utilisateur est invité à saisir l'instance $C1$.
   — L'utilisateur est invité à saisir l'instance $C2$.
2. Transformations et vérifications :
   — Le concept $C1$ est transformé en une forme composée uniquement d'éléments atomiques à l'aide du prédicat `transAtom/2`.
   — Le concept $C2$ est également transformé en une forme composée uniquement d'éléments atomiques.
   — Une mise en forme normale négative (NNF) est appliquée à la conjonction des deux concepts transformés à l'aide du prédicat `nnf/2`.
   — Des vérifications sont effectuées pour s'assurer que les concepts $C1$ et $C2$ sont valides. Si ces vérifications échouent, un message d'erreur est affiché et l'utilisateur est invité à recommencer.
3. Ajout à l'Abox :
   — La proposition $(\_, C1 \wedge C2)$ est ajoutée à la liste `Abi`, qui constitue notre Abox.

2.3.3 Partie 3

La partie 3 se caracterise en lancant le predicat `troisieme_etape(Abi,Abr)/2`. Son but est de trier la Abox en differentes listes selon le type d'assertion pourr chaque élement, ensuite il effectue une résolution grace à l'algorithme des tableaux. si il y'a au moins une feuille ouverte, le predicat de resolution renverra true, si toute les feuilles sont fermées le prédicat renverra false.

`troisieme_etape(Abi,Abr)/2` :-

— `tri_Abox(Abi,Lie,Lpt,Li,Lu,Ls)/6`

Ce prédicat prend six arguments : `Abi`, `Lie`, `Lpt`, `Li`, `Lu` et `Ls`. Il est utilisé pour effectuer le tri d'une boîte A (`Abi`) en utilisant les listes `Lie`, `Lpt`, `Li`, `Lu` et `Ls`.

— `resolution(Lie,Lpt,Li,Lu,Ls,Abr)`

Ce prédicat prend cinq arguments : `Lie`, `Lpt`, `Li`, `Lu`, `Ls` et `Abr`. Il est utilisé pour résoudre une résolution en utilisant les listes `Lie`, `Lpt`, `Li`, `Lu`, `Ls` et l'arbre `Abr`.

— `tri_Abox(Abi,Lie,Lpt,Li,Lu,Ls)/6`

Le prédicat fonctionne de manière récursive en traitant chaque type d'assertion présent dans la boîte A. Il utilise des clauses différentes en fonction du type d'assertion. Les différentes clauses correspondent aux différentes variantes des assertions que l'on peut trouver.

Lorsque le prédicat est appelé, il examine le premier élément de la liste `Abi` et le déplace vers une des listes de sortie (`Lie`, `Lpt`, `Li`, `Lu` ou `Ls`) en fonction de son type. Ensuite, il appelle récursivement le prédicat avec le reste de la liste `Abi`. Ce processus se répète jusqu'à ce que toutes les assertions de la boîte A aient été traitées.

— `resolution(Lie,Lpt,Li,Lu,Ls,Abr)` Ce prrédicat implémente l'algorithme de résolution pour traiter différents types d'assertions dans une Abox. Le prédicat prend six arguments : `Lie`, `Lpt`, `Li`, `Lu`, `Ls` et `Abr`, qui sont des listes utilisées pour stocker des assertions ou d'autres informations.

Le prédicat fonctionne par correspondance de motifs selon les différentes clauses. Chaque clause correspond à un type d'assertion spécifique que l'on peut rencontrer dans le processus de résolution.

Voici une explication générale des différentes clauses :

— La première clause traite le cas où l'Abox est vide. Cela signifie qu'il n'y a plus d'assertions à traiter, et donc le processus de résolution est terminé.

— La deuxième clause traite les assertions de la forme `(I,some(R,C))`. Elle vérifie d'abord s'il n'y a pas de conflit (`not_clash(Ls)`), puis appelle le prédicat `complete_some/6` pour appliquer la régle correspondante.

— La troisième clause traite les assertions de la forme `(I,and(C1,C2))`. Elle vérifie également l'absence de conflit, puis appelle le prédicat `transformation_and/6` pour effectuer une transformation spécifique.

— La quatrième clause traite les assertions de la forme `(I,all(R,C))`. Elle suit le même schéma que les clauses précédentes, en vérifiant l'absence de conflit et en appelant le prédicat `deduction_all/6` pourappliquer la régle correspondante.

— La cinquième clause traite les assertions de la forme `(I,or(C1,C2))`. Elle vérifie l'absence de conflit et appelle le prédicat `transformation_or/6` pour appliquer la régle correspondante.

— La dernière clause traite les assertions de la forme `(I,C)` ou `(I,not(C))`. C'est la dernière étape de l'algorithme de résolution pour une feuille donnée. Elle vérifie s'il y a un conflit dans ce nœud.

— `not_clash(Ls)`

Le prédicat `not_clash/1` parcourt la liste d'assertions et vérifie l'absence de conflit en comparant chaque assertion avec les autres assertions de la liste. Si une paire (I, D) et $(I, \neg D)$ ou $(I, \neg D)$ et (I, D) sont présentes dans la liste, cela indique qu'il y a un clash et le prédicat renvoie `false`. Sinon, le prédicat continue de vérifier le reste de la liste jusqu'à ce qu'il atteigne la fin ou détecte un clash. En résumé, le prédicat `not_clash/1` est utilisé pour vérifier l'absence de conflit dans une liste d'assertions en comparant chaque assertion avec les autres assertions de la liste.

— `complete_some([(I, some(R, C)) | Lie], Lpt, Li, Lu, Ls, Abr)`

Ce prédicat est utilisé pour appliquer la règle d'existence ($\exists$) dans un raisonnement logique. Il prend la liste d'assertions correspondante, prend son premier élement de la forme $(I, \text{some}(R, C))$ le prédicat génère une nouvelle instance B, crée une nouvelle assertion (B, C) et effectue une évolution de l'ontologie grace au prédicat `evolue/2` en ajoutant cette nouvelle assertion. Ensuite, il affiche le résultat de cette évolution et continue la résolution sur le nouveau nœud créé.

— `transformation_and(Lie, Lpt, [(I,and(C1,C2)) | Li], Lu, Ls, Abr)`

Ce prédicat est utilisé pour appliquer la règle d'intersection ($\cap$) dans un raisonnement logique. Il prend la liste d'assertions correspondante, prend son premier élement de la forme $(I, \text{and}(C1, C2))$. Ensuite effectue deux évolutions successives de l'ontologie en ajoutant les assertions $(I, C1)$ et $(I, C2)$ à l'aide du prédicat interne `evolue/6`. Ensuite, il affiche le résultat de ces évolutions à l'aide du prédicat interne `affiche_evolution_Abox/12` et continue la résolution sur le nouveau nœud créé.

— `deduction_all(Lie, [(I, all(R,C)) | Lpt], Li, Lu, Ls, Abr)`

Ce prédicat est utilisé pour appliquer la règle ($\forall$) dans un raisonnement logique. Il prend la liste d'assertions correspondante, prend son premier élement de la forme $(I, \text{all}(R, C))$. Si une telle assertion est trouvée, le prédicat récupére toutes les instances B existantes dans la liste Abr telles que (I, B, R) est inclus dansAbr à l'aide du prédicat interne `setof/3`. Ensuite, il effectue une évolution de l'ontologie pour chaque instance trouvée en ajoutant l'assertion (B, C) à l'aide du prédicat interne `evolue/2`. Ensuite, il affiche le résultat de ces évolutions à l'aide du prédicat interne `affiche_evolution_Abox/12` et continue la résolution sur le nouveau nœud créé.

— `transformation_or(Lie, Lpt, Li, [(I, or(C1,C2)) | Lu], Ls, Abr)`

Ce prédicat est utilisé pour appliquer la règle de l'union ($\cup$) dans un raisonnement logique. Il prend la liste d'assertions correspondante, prend son premier élement de la forme $(I, \text{or}(C1, C2))$. Si une telle assertion est trouvée, le prédicat effectue deux évolutions de l'ontologie en créant deux nouveaux nœuds avec les assertions $(I, C1)$ et $(I, C2)$ à l'aide du prédicat interne `evolue/11`. Ensuite, il affiche le résultat de ces évolutions à l'aide du prédicat interne `affiche/1` et continue la résolution sur pour les deux nouveaux nœuds créés.

— `evolue/11`

Le prédicat evolue/11 est utilisé pour insérer une nouvelle assertion dans une liste d'assertions. Il prend en entrée une assertion A et les différentes listes (Lie, Lpt, Li, Lu, Ls) qui représentent les assertions de concepts dans l'Abox étendue. Les listes NewLie, NewLpt, NewLi, NewLu, NewLs sont les nouvelles listes mises à jour après l'insertion de l'assertion.

Le prédicat traite deux cas : le cas d'une nouvelle assertion (qui n'existe pas déjà dans la liste correspondante) et le cas d'une assertion existante (qui existe déjà dans la liste correspondante).

Dans le cas d'une nouvelle assertion, le prédicat vérifie d'abord si l'assertion A n'existe pas

déjà dans la liste correspondante en utilisant le prédicat member/2. Si c'est le cas, l'assertion A est ajoutée à la liste correspondante (Lie, Lpt, Li, Lu, Ls) pour obtenir les nouvelles listes NewLie, NewLpt, NewLi, NewLu, NewLs.

Dans le cas d'une assertion existante, le prédicat vérifie si l'assertion A existe déjà dans la liste correspondante en utilisant le prédicat member/2. Si c'est le cas, les nouvelles listes NewLie, NewLpt, NewLi, NewLu, NewLs sont identiques aux listes d'origine (Lie, Lpt, Li, Lu, Ls).

Le prédicat evolue_liste/11 est une fonction auxiliaire utilisée pour appliquer le prédicat evolue/11 sur une liste d'assertions. Il prend en entrée une liste d'assertions L et les mêmes listes que evolue/11. Il applique récursivement le prédicat evolue/11 sur chaque assertion de la liste L, mettant à jour les listes à chaque étape pour obtenir les nouvelles listes NewLie, NewLpt, NewLi, NewLu, NewLs.

3 Tests

3.1 Tests validité des Boxs

3.1.1 Test d'une Tbox non circulaire

Avec les définitions suivantes, on a une Tbox circulaire :

```
% scupleur appelle auteur dans sa définition, et auteur appelle sculpteur dans sa définition --> Tbo
equiv(sculpteur, and(auteur, some(aCree, sculpture))). % Ici sculpteur appelle auteur
equiv(auteur, and(sculpteur, some(aEcrit, livre))). % Ici auteur appelle sculpteur

% Tbox
equiv(editeur, and(personne, and(not(some(aEcrit, livre)), some(aEdite, livre)))).
equiv(parent, and(personne, some(aEnfant, anything))).

% concept atomique
cnamea(personne).
cnamea(livre).
cnamea(objet).
cnamea(sculpture).
cnamea(anything).
cnamea(nothing).

% concept non atomique
cnamena(auteur).
cnamena(editeur).
cnamena(sculpteur).
cnamena(parent).

% instances
iname(michelAnge).
iname(david).
iname(sonnets).
```

```
iname(vinci).
iname(joconde).

% rôles
rname(aCree).
rname(aEcrit).
rname(aEdite).
rname(aEnfant).

% instanciation de concept
inst(michelAnge, personne).
inst(david, sculpture).
inst(sonnets, livre).
inst(vinci, personne).
inst(joconde, objet).

% instanciation de rôle
instR(michelAnge, david, aCree).
instR(michelAnge, sonnets, aEcrit).
instR(vinci, joconde, aCree).
```

L'execution du main nous donne le résultat suivant :

```
Début de la première étape

Affichage des boxs :
Tbox :
(auteur, and(sculpteur,some(aEcrit,livre)))
(editeur, and(personne,and(not(some(aEcrit,livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(auteur,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(joconde, objet)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)

Abox d'insertion de rôle :
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)

[PROG] Vérification de la TBox ............
[PROG] Vérification de la TBox réussi ............
[PROG] Vérification de la ABox ............
[PROG] Vérification de la ABox réussi ............
[ERROR] Tbox circulaire
```

3.1.2 Test d'une Tbox avec une erreur sytaxique

Avec les définitions suivantes, on a une erreur syntaxique dans la Tbox :

```
% sculpteur appelle nom_qui_nexiste_pas dans sa définition --> Erreur syntaxique dans la TBox
equiv(sculpteur,and(nom_qui_nexiste_pas,some(aCree,sculpture))).

% Tbox
equiv(auteur,and(personne,some(aEcrit,livre))).
equiv(editeur,and(personne,and(not(some(aEcrit,livre)),some(aEdite,livre)))).
equiv(parent,and(personne,some(aEnfant,anything))).

% concept atomique
cnamea(personne).
cnamea(livre).
cnamea(objet).
cnamea(sculpture).
cnamea(anything).
```

```
cnamea(nothing).

% concept non atomique
cnamena(auteur).
cnamena(editeur).
cnamena(sculpteur).
cnamena(parent).

% instances
iname(michelAnge).
iname(david).
iname(sonnets).
iname(vinci).
iname(joconde).

% rôles
rname(aCree).
rname(aEcrit).
rname(aEdite).
rname(aEnfant).

% instanciation de concept
inst(michelAnge,personne).
inst(david,sculpture).
inst(sonnets,livre).
inst(vinci,personne).
inst(joconde,objet).

% instanciation de rôle
instR(michelAnge, david, aCree).
instR(michelAnge, sonnets, aEcrit).
instR(vinci, joconde, aCree).
```

L'execution du main nous donne le résultat suivant :

```
Début de la première étape

Affichage des boxs :
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(nom_qui_nexiste_pas,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(joconde, objet)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)

Abox d'insertion de rôle :
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)

[PROG] Vérification de la TBox .............
[ERROR] Erreur syntaxique dans la TBox
```

3.1.3 Test d'une Abox avec une erreur sytaxique

Avec les définitions suivantes, on a une erreur syntaxique dans la Abox :

```
% On a une instanciation de role ou michealAnge est un concept_qui_nexiste_pas  --> Erreur syntaxiqu
inst(michelAnge,concept_qui_nexiste_pas).

% Tbox
equiv(sculpteur,and(personne,some(aCree,sculpture))).
equiv(auteur,and(personne,some(aEcrit,livre))).
equiv(editeur,and(personne,and(not(some(aEcrit,livre)),some(aEdite,livre)))).
equiv(parent,and(personne,some(aEnfant,anything))).

% concept atomique
cnamea(personne).
cnamea(livre).
cnamea(objet).
cnamea(sculpture).
cnamea(anything).
cnamea(nothing).
```

```
% concept non atomique
cnamena(auteur).
cnamena(editeur).
cnamena(sculpteur).
cnamena(parent).

% instances
iname(michelAnge).
iname(david).
iname(sonnets).
iname(vinci).
iname(joconde).

% rôles
rname(aCree).
rname(aEcrit).
rname(aEdite).
rname(aEnfant).

% instanciation de concept
inst(david,sculpture).
inst(sonnets,livre).
inst(vinci,personne).
inst(joconde,objet).

% instanciation de rôle
instR(michelAnge, david, aCree).
instR(michelAnge, sonnets, aEcrit).
instR(vinci, joconde, aCree).
```

L'execution du main nous donne le résultat suivant :

```
?- main.
╭──────────────────────────────────╮
│     Début de la première étape   │
╰──────────────────────────────────╯

Affichage des boxs :
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)))

Abox d'insertion de concept :
(david, sculpture)
(joconde, objet)
(michelAnge, concept_qui_nexiste_pas)
(sonnets, livre)
(vinci, personne)

Abox d'insertion de rôle :
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)

[PROG] Vérification de la TBox .............
[PROG] Vérification de la TBox réussi .............
[PROG] Vérification de la ABox .............
[ERROR] Erreur syntaxique dans la ABox
```

3.2 Tests Global

Afin de tester notre code : on a utilisé le test donné dans l'ennoncé du projet en ajoutant quelques instances et concepts comme suit :

```
% Tbox
equiv(sculpteur,and(personne,some(aCree,sculpture))).
equiv(auteur,and(personne,some(aEcrit,livre))).
equiv(editeur,and(personne,and(not(some(aEcrit,livre)),some(aEdite,livre)))).
equiv(parent,and(personne,some(aEnfant,anything))).
equiv(animal,not(personne)).

% concept atomique
cnamea(personne).
cnamea(livre).
cnamea(objet).
```

```
cnamea(sculpture).
cnamea(anything).
cnamea(nothing).

% concept non atomique
cnamena(auteur).
cnamena(editeur).
cnamena(sculpteur).
cnamena(parent).
cnamena(animal).

% instances
iname(michelAnge).
iname(david).
iname(sonnets).
iname(vinci).
iname(joconde).
iname(garfield).
iname(jon).

% rôles
rname(aCree).
rname(aEcrit).
rname(aEdite).
rname(aEnfant).
rname(aDessine).

% instanciation de concept
inst(michelAnge,personne).
inst(david,sculpture).
inst(sonnets,livre).
inst(vinci,personne).
inst(joconde,objet).
inst(garfield,animal).
inst(jon,personne).

% instanciation de rôle
instR(michelAnge, david, aCree).
instR(michelAnge, sonnets, aEcrit).
instR(vinci, joconde, aCree).
instR(jon, garfield, aEnfant). % Propriètere de Garfield (son papa)
```

3.2.1 Test d'insertions valides

- michelAnge fait partie du concept or(personne, some(aCree, sculpture)).

```
?- main.

     Début de la première étape

Affichage des boxs :
Tbox :
(animal, not(personne))
(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)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, animal)
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)

[PROG] Vérification de la TBox .....
[PROG] Vérification de la TBox réussi .....
[PROG] Vérification de la ABox .....
[PROG] Vérification de la ABox réussi .....
[PROG] Tbox non circulaire .....
[PROG] Simplification des boxs .....
[PROG] Traitement de la Tbox et des Abox terminé avec succès .....

Affichage des boxs transformés :
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))
```



```
Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)
```

```
Début de la deuxième étape

Entrez le numéro du type de proposition que vous voulez démontrer :
1 - type I : C (Vérifié qu'une instance I appartient a un concept C)
2 - type C1 ⊓ C2 ∈ ⊥ (Vérifier si deux concepts C1 et C2 sont disjoints)
|: 1.
Vérification de la proposition : I : C
Saisir l'instance I :
|: michelAnge.
Saisir le concept C :
|: or(personne, some(aCree, sculpture)).

Proposition à vérifier : [michelAnge : or(personne,some(aCree,sculpture))]
Ajout de la négation de cette proposition : [michelAnge : ¬(or(personne,some(aCree,sculpture)))]

Affichage des boxs avec instertions de la négation de la proposition a vérifier:
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)
(michelAnge, and(not(personne),all(aCree,not(sculpture))))

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)
```

```
Début de la troisième étape

[PROG] Trie de la Abox terminé avec succès .....
Application de la regle ⊓ pour michelAnge : ¬ personne ⊓  ∀ aCree. ¬ sculpture

Etat de départ :

michelAnge :  ¬ personne ⊓  ∀ aCree. ¬ sculpture
david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Etat d'arrivée :

michelAnge :  ¬ personne
david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
michelAnge :  ∀ aCree. ¬ sculpture
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Clash dans ce noeud

============================

[RES] Toutes les feuilles sont fermées : la proposition initiale est VALIDE :)

[PROG] Programme terminé avec succès ....."),
true .
```

- michelAnge fait partie du concept some(aCree, sculpture))

```
?- main.

     Début de la première étape


Affichage des boxs :
Tbox :
(animal, not(personne))
(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)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, animal)
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)

[PROG] Vérification de la TBox .............
[PROG] Vérification de la TBox réussi .............
[PROG] Vérification de la ABox .............
[PROG] Vérification de la ABox réussi .............
[PROG] Tbox non circulaire .............
[PROG] Simplification des boxs .............
[PROG] Traitement de la Tbox et des Abox terminé avec succès .............

Affichage des boxs transformés :
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))
```



```
Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)
```

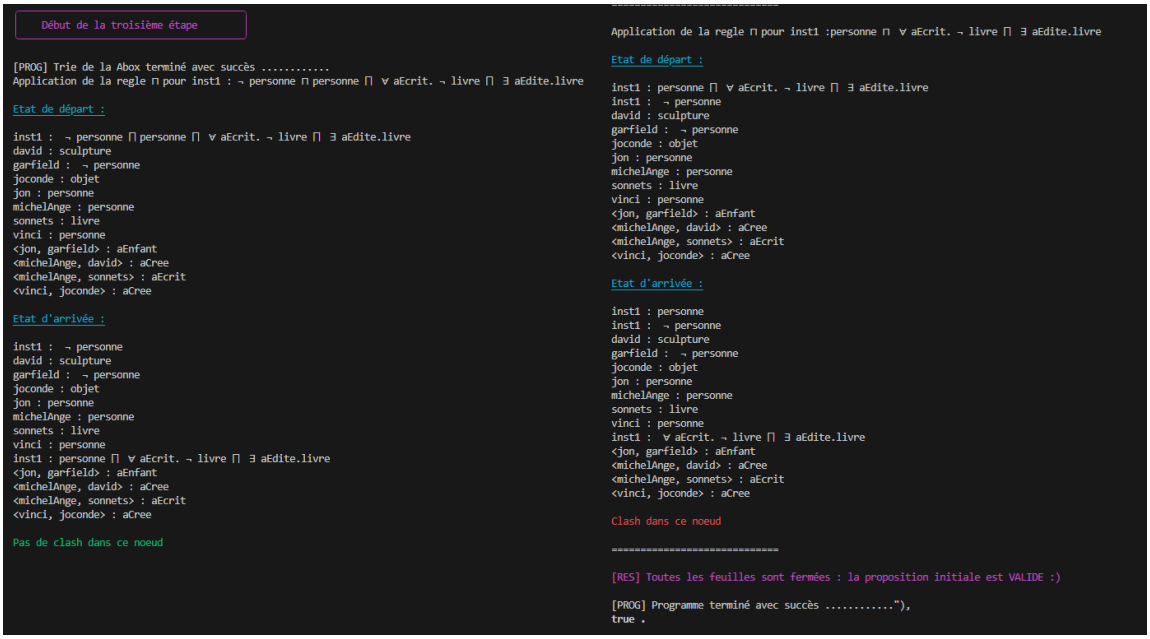
```
Début de la deuxième étape

Entrez le numéro du type de proposition que vous voulez démontrer :
1 - type I : C (Vérifié qu'une instance I appartient a un concept C)
2 - type C1 ⊓ C2 ⋐ ⊥ (Vérifier si deux concepts C1 et C2 sont disjoints)
|: 1.
Vérification de la proposition : I : C
Saisir l'instance I :
|: michelAnge.
Saisir le concept C :
|: some(aCree, sculpture).

Proposition à vérifier : [michelAnge : some(aCree,sculpture)]
Ajout de la négation de cette proposition : [michelAnge : ¬(some(aCree,sculpture))]

Affichage des boxs avec instertions de la négation de la proposition a vérifier:
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)
(michelAnge, all(aCree,not(sculpture)))

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)
```

```
Début de la troisième étape

[PROG] Trie de la Abox terminé avec succès .....
Application de la regle ∀ pour michelAnge : aCree ∀  ¬ sculpture

Application de la regle ∀ trouvée
Etat de départ :

michelAnge :  ∀ aCree. ¬ sculpture
david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Etat d'arrivée :

david :  ¬ sculpture
david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Clash dans ce noeud

============================

Traitement d'une feuille

Clash dans ce noeud


============================

[RES] Toutes les feuilles sont fermées : la proposition initiale est VALIDE :)

[PROG] Programme terminé avec succès .....");
true .
```

- animal et editeur sont deux concepts disjoints

```
Début de la première étape

Affichage des boxs :
Tbox :
(animal, not(personne))
(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)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, animal)
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)

[PROG] Vérification de la TBox .....
[PROG] Vérification de la TBox réussi .....
[PROG] Vérification de la ABox .....
[PROG] Vérification de la ABox réussi .....
[PROG] Tbox non circulaire .....
[PROG] Simplification des boxs .....
[PROG] Traitement de la Tbox et des Abox terminé avec succès .....

Affichage des boxs transformés :
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)
Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)
```

```
Début de la deuxième étape

Entrez le numéro du type de proposition que vous voulez démontrer :
1 - type I : C (Vérifié qu'une instance I appartient a un concept C)
2 - type C1 ⊓ C2 ∈ ⊥ (Vérifier si deux concepts C1 et C2 sont disjoints)
|: 2.
Vérification de la proposition C1 ⊓ C2 ∈ ⊥
Saisir l'instance C1 :
|: animal.
Saisir l'instance C2 :
|: editeur.

Proposition à vérifier : [animal ⊓ editeur ∈ ⊥]



Ajout de la négation de cette proposition : [∃ inst1, inst1 : animal ⊓ editeur]
Affichage des boxs avec instertions de la négation de la proposition a vérifier:
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)
(inst1, and(not(personne),and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre)))))

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)
```



3.2.2 Test d'insertions non valides

- michelAnge appartient au concept and(personne, livre)

```
Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)

    Début de la troisième étape

[PROG] Trie de la Abox terminé avec succès .............
Application de la regle ⊔ pour michelAnge : ¬ personne ⊔  ¬ livre
Avec création du noeud contenant michelAnge : ¬ personne

Etat de départ :

david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
michelAnge :  ¬ personne ⊔  ¬ livre
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Etat d'arrivée :

michelAnge :  ¬ personne
david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Clash dans ce noeud

============================

Traitement d'une feuille
```

```
Clash dans ce noeud


============================

Application de la regle ⊔ pour michelAnge : ¬ personne ⊔  ¬ livre
Avec création du noeud contenant michelAnge : ¬ livre

Etat de départ :

david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
michelAnge :  ¬ personne ⊔  ¬ livre
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Etat d'arrivée :

michelAnge :  ¬ livre
david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Pas de clash dans ce noeud

============================

Traitement d'une feuille

Pas de clash dans ce noeud

============================
```

```
[RES] Il existe une feuille ouverte : la proposition initiale est NON VALIDE :(

[PROG] Programme terminé avec succès ............"),
true .
```

- michelAnge appartient au concept some(aCree, personne)

```
?- main.

     Début de la première étape


Affichage des boxs :
Tbox :
(animal, not(personne))
(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)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, animal)
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)

[PROG] Vérification de la TBox .....
[PROG] Vérification de la TBox réussi .....
[PROG] Vérification de la ABox .....
[PROG] Vérification de la ABox réussi .....
[PROG] Tbox non circulaire .....
[PROG] Simplification des boxs .....
[PROG] Traitement de la Tbox et des Abox terminé avec succès .....

Affichage des boxs transformés :
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)
```

```
Début de la deuxième étape

Entrez le numéro du type de proposition que vous voulez démontrer :
1 - type I : C (Vérifié qu'une instance I appartient a un concept C)
2 - type C1 ⊓ C2 ◀ ⊥ (Vérifier si deux concepts C1 et C2 sont disjoints)
|: 1.
Vérification de la proposition : I : C
Saisir l'instance I :
|: michelAnge.
Saisir le concept C :
|: some(aCree, personne).

Proposition à vérifier : [michelAnge : some(aCree,personne)]
Ajout de la négation de cette proposition : [michelAnge : ¬(some(aCree,personne))]

Affichage des boxs avec instertions de la négation de la proposition a vérifier:
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)
(michelAnge, all(aCree,not(personne)))

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)
```

```
    Début de la troisième étape

[PROG] Trie de la Abox terminé avec succès .............
Application de la regle ∀ pour michelAnge : aCree ∀  ¬ personne

Application de la regle ∀ trouvée
Etat de départ :

michelAnge :  ∀ aCree. ¬ personne
david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Etat d'arrivée :

david :  ¬ personne
david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Pas de clash dans ce noeud

============================

Traitement d'une feuille

Pas de clash dans ce noeud

============================

[RES] Il existe une feuille ouverte : la proposition initiale est NON VALIDE :(

[PROG] Programme terminé avec succès .............),
true .
```

- jon appartient au concept all(aEnfant, personne).

```
?- main.

     Début de la première étape

Affichage des boxs :
Tbox :
(animal, not(personne))
(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)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, animal)
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)

[PROG] Vérification de la TBox ............
[PROG] Vérification de la TBox réussi ............
[PROG] Vérification de la ABox ............
[PROG] Vérification de la ABox réussi ............
[PROG] Tbox non circulaire ............
[PROG] Simplification des boxs ............
[PROG] Traitement de la Tbox et des Abox terminé avec succès ............

Affichage des boxs transformés :
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)
```

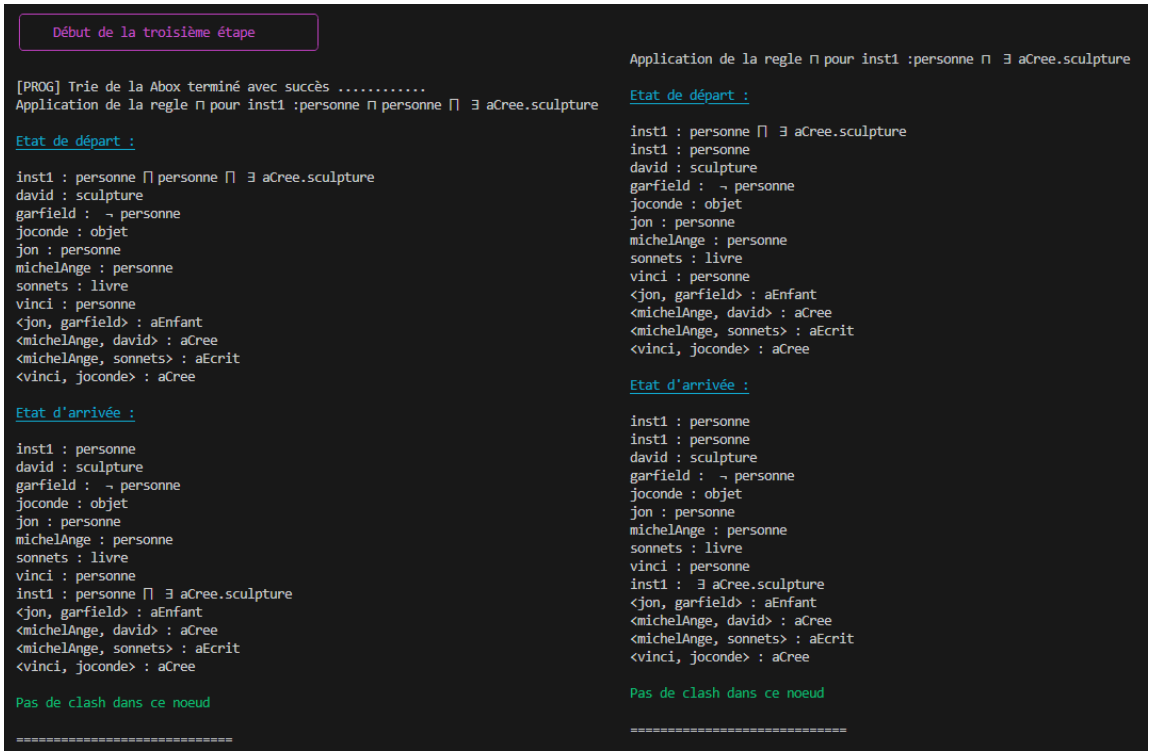
```
    Début de la deuxième étape

Entrez le numéro du type de proposition que vous voulez démontrer :
1 - type I : C (Vérifié qu'une instance I appartient a un concept C)
2 - type C1 ⊓ C2 ⊑ ⊥ (Vérifier si deux concepts C1 et C2 sont disjoints)
|: 1.
Vérification de la proposition : I : C
Saisir l'instance I :
|: jon.
Saisir le concept C :
|: all(aEnfant, personne).

Proposition à vérifier : [jon : all(aEnfant,personne)]
Ajout de la négation de cette proposition : [jon : ¬(all(aEnfant,personne))]

Affichage des boxs avec instertions de la négation de la proposition a vérifier:
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)
(jon, some(aEnfant,not(personne)))

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)
```

```
    Début de la troisième étape

[PROG] Trie de la Abox terminé avec succès .....
Application de la regle ∃ pour jon : aEnfant ∃  ¬ personne



Etat de départ :

jon :  ∃ aEnfant. ¬ personne
david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Etat d'arrivée :

inst1 :  ¬ personne
david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
<jon, inst1> : aEnfant
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Pas de clash dans ce noeud

============================

Traitement d'une feuille

Pas de clash dans ce noeud

============================

[RES] Il existe une feuille ouverte : la proposition initiale est NON VALIDE :(

[PROG] Programme terminé avec succès .....");
true .
```

- not(animal) et sculpture sont disjoint

```
Début de la première étape

Affichage des boxs :
Tbox :
(animal, not(personne))
(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)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, animal)
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)

[PROG] Vérification de la TBox .....
[PROG] Vérification de la TBox réussi .....
[PROG] Vérification de la ABox .....
[PROG] Vérification de la ABox réussi .....
[PROG] Tbox non circulaire .....
[PROG] Simplification des boxs .....
[PROG] Traitement de la Tbox et des Abox terminé avec succès .....

Affichage des boxs transformés :
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)
```

```
Début de la deuxième étape

Entrez le numéro du type de proposition que vous voulez démontrer :
1 - type I : C (Vérifié qu'une instance I appartient a un concept C)
2 - type C1 ⊓ C2 ∈ ⊥ (Vérifier si deux concepts C1 et C2 sont disjoints)
|: 2.
Vérification de la proposition C1 ⊓ C2 ∈ ⊥
Saisir l'instance C1 :
|: not(animal).
Saisir l'instance C2 :
|: sculpteur.

Proposition à vérifier : [not(animal) ⊓ sculpteur ∈ ⊥]



Ajout de la négation de cette proposition : [∃ inst1, inst1 : not(animal) ⊓ sculpteur]
Affichage des boxs avec instertions de la négation de la proposition a vérifier:
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)
(inst1, and(personne,and(personne,some(aCree,sculpture))))

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)
```

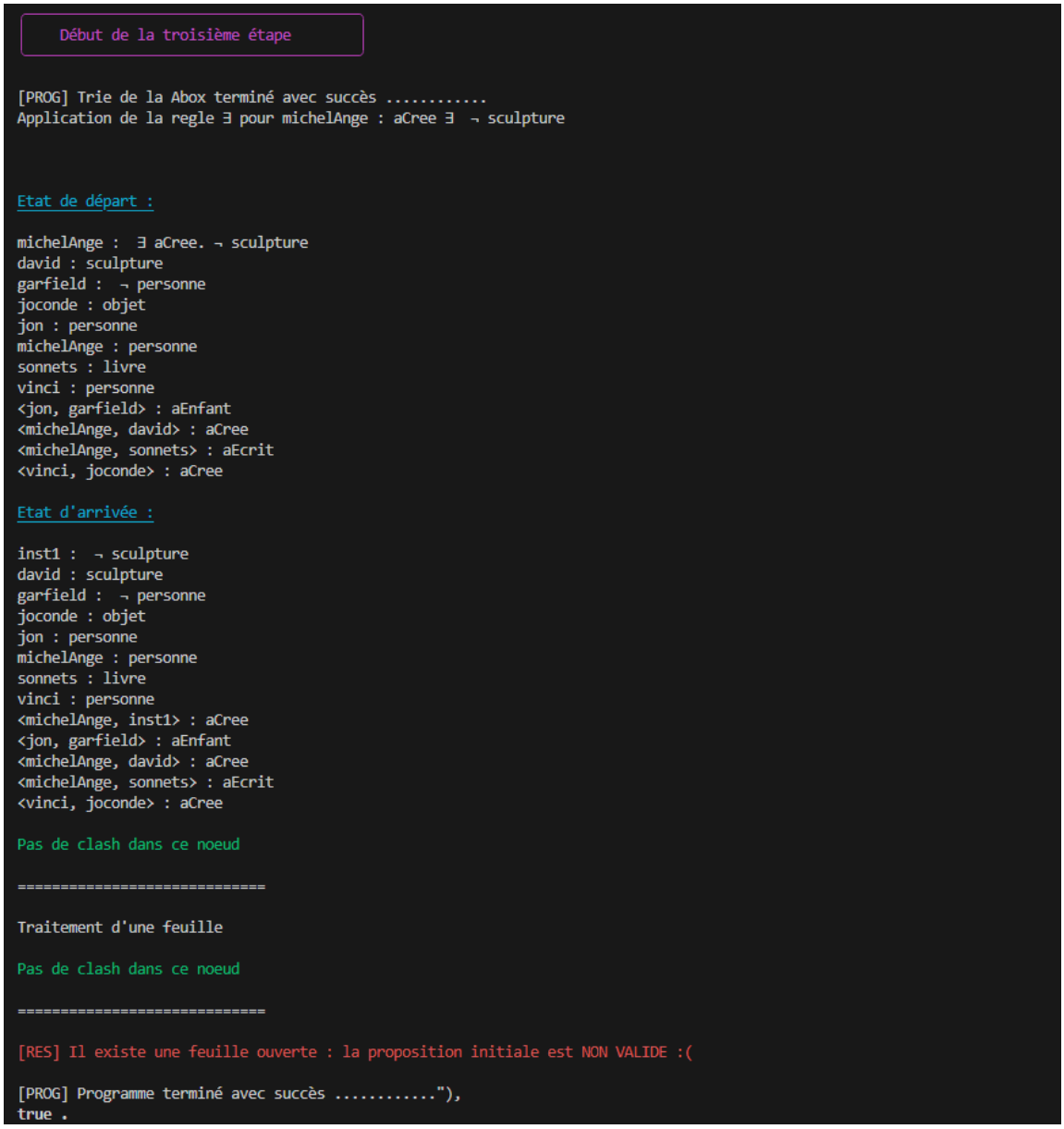

Début de la troisième étape
[PROG] Trie de la Abox terminé avec succès .....
Application de la regle ⊓ pour inst1 :personne ⊓ personne ⊓ ∃ aCree.sculpture
Etat de départ :
inst1 : personne ⊓ personne ⊓ ∃ aCree.sculpture
david : sculpture
garfield : ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree
Etat d'arrivée :
inst1 : personne
david : sculpture
garfield : ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
inst1 : personne ⊓ ∃ aCree.sculpture
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree
Pas de clash dans ce noeud
==========
Application de la regle ⊓ pour inst1 :personne ⊓ ∃ aCree.sculpture
Etat de départ :
inst1 : personne ⊓ ∃ aCree.sculpture
inst1 : personne
david : sculpture
garfield : ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree
Etat d'arrivée :
inst1 : personne
inst1 : personne
david : sculpture
garfield : ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
inst1 : ∃ aCree.sculpture
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree
Pas de clash dans ce noeud
==========

```
Application de la regle ∃ pour inst1 : aCree ∃ sculpture



Etat de départ :

inst1 :  ∃ aCree.sculpture
inst1 : personne
inst1 : personne
david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Etat d'arrivée :

inst2 : sculpture
inst1 : personne
inst1 : personne
david : sculpture
garfield :  ¬ personne
joconde : objet
jon : personne
michelAnge : personne
sonnets : livre
vinci : personne
<inst1, inst2> : aCree
<jon, garfield> : aEnfant
<michelAnge, david> : aCree
<michelAnge, sonnets> : aEcrit
<vinci, joconde> : aCree

Pas de clash dans ce noeud

============================

Traitement d'une feuille

Pas de clash dans ce noeud

============================

[RES] Il existe une feuille ouverte : la proposition initiale est NON VALIDE :(
```

- michel ange apratient au concept all(ACree, sculpture)

```
Début de la deuxième étape

Entrez le numéro du type de proposition que vous voulez démontrer :
1 - type I : C (Vérifié qu'une instance I appartient a un concept C)
2 - type C1 ⊓ C2 ∈ ⊥ (Vérifier si deux concepts C1 et C2 sont disjoints)
|: 1.
Vérification de la proposition : I : C
Saisir l'instance I :
|: michelAnge.
Saisir le concept C :
|:  all(aCree, sculpture).

Proposition à vérifier : [michelAnge : all(aCree,sculpture)]
Ajout de la négation de cette proposition : [michelAnge : ¬(all(aCree,sculpture))]

Affichage des boxs avec instertions de la négation de la proposition a vérifier:
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)
(michelAnge, some(aCree,not(sculpture)))
```

```
Début de la première étape

Affichage des boxs :
Tbox :
(animal, not(personne))
(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)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, animal)
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)

Abox d'insertion de rôle :
(jon, garfield,aEnfant)
(michelAnge, david,aCree)
(michelAnge, sonnets,aEcrit)
(vinci, joconde,aCree)

[PROG] Vérification de la TBox .....
[PROG] Vérification de la TBox réussi .....
[PROG] Vérification de la ABox .....
[PROG] Vérification de la ABox réussi .....
[PROG] Tbox non circulaire .....
[PROG] Simplification des boxs .....
[PROG] Traitement de la Tbox et des Abox terminé avec succès .....

Affichage des boxs transformés :
Tbox :
(animal, not(personne))
(auteur, and(personne,some(aEcrit,livre)))
(editeur, and(personne,and(all(aEcrit,not(livre)),some(aEdite,livre))))
(parent, and(personne,some(aEnfant,anything)))
(sculpteur, and(personne,some(aCree,sculpture)))

Abox d'insertion de concept :
(david, sculpture)
(garfield, not(personne))
(joconde, objet)
(jon, personne)
(michelAnge, personne)
(sonnets, livre)
(vinci, personne)
(...)
Abox d'insertion de rôle :
(jon, garfield,aEnfant)
```



4 Conclusion

Ce projet de recherche sur l'UE Logique et Représentation des Connaissances nous a permis de mettre en œuvre un démonstrateur basé sur l'algorithme des tableaux pour la logique de description ALC. Les étapes préliminaires de vérification et de mise en forme de la Tbox et de la Abox

garantissent la qualité des données d'entrée. La saisie de la proposition à démontrer et l'application de l'algorithme de résolution nous permettent de vérifier la validité des propositions dans le contexte de la logique de description ALC. Ce projet a renforcé notre compréhension des concepts clés de la logique de description et nous a préparés à relever de nouveaux défis dans le domaine de la représentation des connaissances.