Projet LRC

Écriture d'un démonstrateur en Prolog

1

Superviseur:

Colette Faucher

Binôme:

Amine Kahil (28616887)
Esther Choi (3800370)

Introduction

Dans le cadre de l'UE de LRC (Logique et Représentation des Connaissances), nous devions écrire en Prolog un démonstrateur basé sur l'algorithme des tableaux dans la logique de description $\mathcal{ALC}$.
Ce document constitue le compte-rendu de ce projet. Il contient notamment la liste des prédicats importants utilisés, avec leur description et un jeu de test.

Sommaire

Partie 1: Préliminaires

Le code est intégralement donné dans l'annexe.

Partie 2: Saisie de la proposition à démontrer

concept(L)

`concept/1` vérifie que le premier élément de la liste `L` est un concept valide (sémantiquement et syntaxiquement).
Jeu de tests :

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

?- concept([animal]).
false.

?- concept([auteur]).
true.

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

correction(P)

`correction/1` vérifie que le premier élément de `P` est bien l'identificateur d'une instance et que le deuxième élément est un concept valide (en utilisant `concept`).
Jeu de tests :

```
?- correction([joconde,o]).
false.

?- correction([joconde,objet]).
true.

?- correction([david,and(parent,auteur)]).
true.
```

remplace_concepts_complexes(L,R)

`remplace_concepts_complexes/2` met dans `R` la définition constituée uniquement de concepts atomiques du concept contenu dans `L`.
Jeu de tests :

```
?- remplace_concepts_complexes([objet],R).
R = objet.

?- remplace_concepts_complexes([auteur],R).
R = and(personne, some(aEcrit, livre)).
```

```
?- remplace_concepts_complexes([or(some(aEnfant,auteur),sculpteur)],R).
R = or(some(aEnfant, and(personne, some(aEcrit, livre))),
    and(personne, some(aCree, sculpture))).
```

traitement(P,Ptraitennf)

Le premier élément de `P` est une instance I et le second, un concept C (qui peut être complexe). `traitement/2` met dans `Ptraitennf` le couple formé par I et la forme normale négative de ¬C (en utilisant `remplace_concepts_complexes/2` et `nnf/2`).
Jeu de tests :

```
?- traitement([michelAnge,personne],Ptraitennf).
Ptraitennf =  (michelAnge, not(personne)).

?- traitement([michelAnge,sculpteur],Ptraitennf).
Ptraitennf =  (michelAnge, or(not(personne), all(aCree, not(sculpture)))).
```

ajout(Ptraitennf,Abi,Abi1)

`Ptraitennf` est un couple (I,C) où I est une instance et C un concept, `Abi` est une liste, contenant en pratique les assertions de concepts de la Abox.
`ajout/3` ajoute `Ptraitennf` dans `Abi` et met le résultat dans `Abi1` (en utilisant `concat`).

correction2(P)

`correction/1` vérifie que les éléments contenus dans la liste `P` sont des concepts valides (en utilisant `concept`).
Jeu de tests :

```
?- correction2([sculpture,obj]).
false.

?- correction2([personne,objet]).
true.

?- correction2([or(some(aEcrit,livre),all(aCree,sculpture)),parent]).
true.
```

traitement2(P,Ptraite)

La liste `P` contient deux concepts C1 et C2 (qui peuvent être complexes).
`traitement2/2` met dans `Ptraite` le couple formé par les définitions n'utilisant que les concepts atomiques de C1 et C2.
Jeu de tests :

```
?- traitement2([personne,auteur],Ptraite).
Ptraite =  (personne, and(personne, some(aEcrit, livre))).

?- traitement2([personne,or(parent,sculpteur)],Ptraite).
Ptraite =  (personne, or(and(personne, some(aEnfant, anything)),
                         and(personne, some(aCree, sculpture)))).
```

ajout2(Ptraite,Abi,Abi1)

`Ptraite` est un couple (C1,C2) où C1 et C2 sont des concepts, `Abi` est une liste, contenant en pratique les assertions de concepts de la Abox.
`ajout2/3` génère un nouvel identificateur d'instance `Nom` (avec `genere`) et ajoute le couple (`Nom`,and(C1,C2)) dans `Abi` et met le résultat dans `Abi1` (avec `concat`).

Principe général

L'utilisateur est invité à entrer le type de proposition à montrer puis à la saisir.
Dans le cas du type 1, il faut taper l'instance, puis le concept, puis "`fin.`". On vérifie que ce qui a été entré est correct, puis, après avoir remplacé le concept par sa définition, la négation de cette assertion est ajoutée à la Abox avec `ajout`.
Dans le cas du type 2, il faut taper successivement les deux concepts, puis "`fin.`". On vérifie que ce qui a été entré est correct, puis, après avoir remplacé les concepts par leur définition, on applique `ajout2` sur l'intersection des deux.

Partie 3: Démonstration de la proposition

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

Fonctionne comme décrit dans l'énoncé du projet.
Jeu de tests :

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

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

test_clash(L)

La liste `L` contient des couples du type (I,C) où I est une instance, et C un concept atomique ou sa négation.
`test_clash/1` vaut vrai si `L` ne contient pas de clash, et faux sinon.
Jeu de tests :

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

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

?- test_clash([(michelAnge,personne),(david,sculpteur),
                (michelAnge,auteur)]).
true.
```

`evolue(A,Lie,Lpt,Li,Lu,Ls,Lie1,Lpt1,Li1,Lu1,Ls1)`

Fonctionne comme décrit dans l'énoncé du projet.
Jeu de tests :

```
?- evolue((joconde,all(aCree,livre)),[],[],[],[],[],Lie1,Lpt1,Li1,Lu1,Ls1).
Lie1 = Li1, Li1 = Lu1, Lu1 = Ls1, Ls1 = [],
Lpt1 = [(joconde, all(aCree, livre))].

?- evolue((joconde,all(aCree,livre)),[],[(michelAnge,all(aCree,livre))],
          [],[],[],Lie1,Lpt1,Li1,Lu1,Ls1).
Lie1 = Li1, Li1 = Lu1, Lu1 = Ls1, Ls1 = [],
Lpt1 = [(joconde, all(aCree, livre)),  (michelAnge, all(aCree, livre))].

?- evolue((joconde,all(aCree,livre)),[],[(michelAnge,all(aCree,livre))],[],[],
          [(sonnets,livre)],Lie1,Lpt1,Li1,Lu1,Ls1).
Lie1 = Li1, Li1 = Lu1, Lu1 = [],
Lpt1 = [(joconde, all(aCree, livre)),  (michelAnge, all(aCree, livre))],
Ls1 = [(sonnets, livre)].

?- evolue((joconde,not(all(aCree,livre))),[],[(michelAnge,all(aCree,livre))],
          [],[],[(sonnets,livre)],Lie1,Lpt1,Li1,Lu1,Ls1).
Lie1 = [(joconde, some(aCree, not(livre)))],
Lpt1 = [(michelAnge, all(aCree, livre))],
Li1 = Lu1, Lu1 = [],
Ls1 = [(sonnets, livre)].
```

Principe général

On regarde si l'on peut appliquer une des règles (dans l'ordre $\exists, \sqcap, \forall, \sqcup$) en regardant la longueur des listes de la Abe correspondante.
Si c'est le cas, on applique la transformation adéquate, on teste s'il y a un clash dans `Ls`, et s'il n'y en a pas, alors on fait un appel récursif à `resolution`.

`resolution/6`, `complete_some/6`, `transformation_and/6`, `transformation_or/6` et `deduction_all/6` fonctionnent comme décrits dans le sujet.
Pour tester ces prédicats, nous pouvons simplement lancer le programme en entier sur des exemples.

Exemples

Voici deux exemples d'exécutions complètes du programme, avec les ABox et TBox de l'exercice 3 du TD4, l'une de type 1 et l'autre de type 2.

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

```
?- programme.

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 (ils ont une intersection vide).
|: 1.

Entrez la proposition a montrer (d'abord l'instance, puis le concept, puis tapez 'fin.') :
|: michelAnge.
|: some(aEcrit, livre).
|: fin.
Début de la résolution

Appel de deduction_all
---État de départ de la Abox---

michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
michelAnge : ∀aEcrit.¬livre
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree
---Etat d'arrivée---

sonnets : ¬livre
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree
=======FIN========

Branche fermée !!

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

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

```
?- programme.

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 (ils ont une inters
ection vide).
|: 2.

Entrez la proposition a montrer (d'abord le premier concept, puis le
 second, puis tapez 'fin.') :
|: editeur.
|: auteur.
|: fin.
```

```
Début de la résolution

Appel de transformation_and
---État de départ de la Abox---
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
inst1 : (personne ⊓ (¬∃aEcrit.livre ⊓ ∃aEdite.livre)) ⊓ (personne ⊓
∃aEcrit.livre)
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

---Etat d'arrivée---
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
inst1 : personne ⊓ ∃aEcrit.livre
inst1 : personne ⊓ (¬∃aEcrit.livre ⊓ ∃aEdite.livre)
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree
=======FIN========

Appel de transformation_and
---État de départ de la Abox---
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
inst1 : personne ⊓ ∃aEcrit.livre
inst1 : personne ⊓ (¬∃aEcrit.livre ⊓ ∃aEdite.livre)
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

---Etat d'arrivée---
inst1 : personne
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
inst1 : ∃aEcrit.livre
inst1 : personne ⊓ (¬∃aEcrit.livre ⊓ ∃aEdite.livre)
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree
=======FIN========
```

```
Appel de complete_some
---État de départ de la Abox---
inst1 : personne
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
inst1 : ∃aEcrit.livre
inst1 : personne ⊓ (¬∃aEcrit.livre ⊓ ∃aEdite.livre)
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

---Etat d'arrivée---
58308 : livre
inst1 : personne
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
inst1 : personne ⊓ (¬∃aEcrit.livre ⊓ ∃aEdite.livre)
<inst1,58308> : aEcrit
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree
=======FIN========
```

```
Appel de transformation_and
---État de départ de la Abox---
58308 : livre
inst1 : personne
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
inst1 : personne ⊓ (¬∃aEcrit.livre ⊓ ∃aEdite.livre)
<inst1,58308> : aEcrit
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

---Etat d'arrivée---
inst1 : personne
58308 : livre
inst1 : personne
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
inst1 : ¬∃aEcrit.livre ⊓ ∃aEdite.livre
<inst1,58308> : aEcrit
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree
=======FIN========
```

```
Appel de transformation_and
---État de départ de la Abox---
inst1 : personne
58308 : livre
inst1 : personne
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
inst1 : ¬∃aEcrit.livre ⊓ ∃aEdite.livre
<inst1,58308> : aEcrit
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

---Etat d'arrivée---
inst1 : personne
58308 : livre
inst1 : personne
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
inst1 : ∃aEdite.livre
inst1 : ∀aEcrit.¬livre
<inst1,58308> : aEcrit
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree
=======FIN========
```

```
Appel de complete_some
---État de départ de la Abox---
inst1 : personne
58308 : livre
inst1 : personne
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
inst1 : ∃aEdite.livre
inst1 : ∀aEcrit.¬livre
<inst1,58308> : aEcrit
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

---Etat d'arrivée---
11496 : livre
inst1 : personne
58308 : livre
inst1 : personne
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
inst1 : ∀aEcrit.¬livre
<inst1,11496> : aEdite
<inst1,58308> : aEcrit
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree
=======FIN========
```

```
Appel de deduction_all
---État de départ de la Abox---
11496 : livre
inst1 : personne
58308 : livre
inst1 : personne
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
inst1 : ∀aEcrit.¬livre
<inst1,11496> : aEdite
<inst1,58308> : aEcrit
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree

---Etat d'arrivée---
58308 : ¬livre
11496 : livre
inst1 : personne
58308 : livre
inst1 : personne
michelAnge : personne
david : sculpture
sonnets : livre
vinci : personne
joconde : objet
<inst1,11496> : aEdite
<inst1,58308> : aEcrit
<michelAnge,david> : aCree
<michelAnge,sonnets> : aEcrit
<vinci,joconde> : aCree
=======FIN========

Branche fermée !!

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