

Rapport de projet

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

Charles Vin

Barthelemy Dang-Nhu

Année : 2022/2023



Table des matières

1	Introduction	2
2	Description générale et fonctionnement	2
2.1	Les fichiers	2
2.2	Utilisation du programme	2
2.2.1	Initialisation de la TBox et de la ABox	2
2.2.2	Lancement et saisie de la proposition à prouver	2
2.3	Exemple d'utilisation	2
3	Étape préliminaire de vérification et de mise en forme de la Tbox et de la Abox	2
3.1	Correction syntaxique et sémantique	3
3.2	Vérification de l'autoréférencement	4
3.3	Mise sous forme	4
4	Saisie de la proposition à montrer	5
5	Démonstration de la proposition	5

1 Introduction

2 Description générale et fonctionnement

2.1 Les fichiers

Pour une meilleure compréhension du code, nous avons séparé celui-ci en plusieurs fichiers. Chacun à sa spécialisation :

- `T-A_box.pl` : C'est ici que l'utilisateur entre la TBox et la ABox initial.
- `run.pl` : Contient le prédicat `programme/0` qui appelle les grandes étapes nécessaires à la résolution.
- `part1.pl` : Contient les fonctions liées à la première partie décrite par le sujet.
- `part2.pl` : Contient les fonctions liées à la deuxième partie décrite par le sujet.
- `part3.pl` : Contient les fonctions liées à la troisième partie décrite par le sujet.
- `helper.pl` : Contient quelques prédicats utiles

2.2 Utilisation du programme

2.2.1 Initialisation de la TBox et de la ABox

2.2.2 Lancement et saisie de la proposition à prouver

Pour lancer le programme entrez cette commande à la racine du projet : `swipl -f run.pl`. Puis dans l'interpréteur prolog utilisez le prédicat `programme` pour lancer le programme.

2.3 Exemple d'utilisation

3 Étape préliminaire de vérification et de mise en forme de la Tbox et de la Abox

À ce stade les Abox et Tbox ne sont pas altérées, il est donc inutile de les mettre en argument des prédicats car on peut y accéder avec les prédicats [cnamea](#), [cnamena](#), [iname](#), [rname](#), [equiv](#), [inst](#), [instR](#).

3.1 Correction syntaxique et sémantique

Dans cette première partie nous commençons par vérifier la correction sémantique et syntaxique des deux box. Pour ce faire nous implémentons les prédicats d'arité 1 `concept`, `instance`, `role` qui vérifie si des objets sont de ce type. Les cas de base sont les suivants :

```
concept(C) :- cnamea(C), !.  
concept(CG) :- cnamena(CG), !.  
instance(I) :- iname(I), !.  
role(R) :- rname(R), !.
```

Le prédicat `concept` nécessite de la récursivité à cause des concepts non-atomiques. Nous vérifions la grammaire avec les prédicats suivants :

```
concept(not(C)) :- concept(C), !.  
concept(and(C1, C2)) :- concept(C1), concept(C2), !.  
concept(or(C1, C2)) :- concept(C1), concept(C2), !.  
concept(some(R, C)) :- role(R), concept(C), !.  
concept(all(R, C)) :- role(R), concept(C), !.
```

Nous utilisons ces prédicats pour définir le prédicat `definition` d'arité 2 qui vérifie si la définition d'une équivalence est juste : il faut que le premier élément soit un concept non-atomique et que le deuxième soit bien la définition d'un concept :

```
definition(CA, CG) :- cnamena(CA), concept(CG), !.
```

Grâce à ce prédicat nous pouvons vérifier la Tbox avec `verif_Tbox` qui prend en argument une TBox sous forme d'une liste de d'équivalence et qui vérifie la correction syntaxique et sémantique :

```
verif_Tbox([(CA, CG) | Q]) :-  
    definition(CA, CG),  
    verif_Tbox(Q).  
verif_Tbox([]).
```

On fait de même avec la Abox

3.2 Vérification de l'autoréférencement

Nous voulons ensuite s'assurer qu'il n'y a pas d'autoréférencement dans la définition des concepts non-atomiques. S'il y en a, au moment de développer les concepts pour n'avoir que des concepts atomiques il y aura une boucle infinie. Nous implémentons le prédicat `pautoref(C, Def)` qui prend en argument un concept non-atomique C et la définition de concept Def et qui est vraie ssi C n'est pas présent récursivement dans Def. Le cas de base est le suivant :

```
pautoref(C, Def) :- cnamea(Def).
```

Nous pouvons ainsi construire le prédicat `verif_Autoref(L)` qui prend en argument la liste des concept non-atomiques et qui vérifie s'il n'y a pas autoréférencement dans leur définition :

```
verif_Autoref([]).  
verif_Autoref([C|L]) :-  
    equiv(C, Def_C),  
    pautoref(C, Def_C),  
    verif_Autoref(L).
```

3.3 Mise sous forme

Une fois que nous sommes sûr qu'il n'y a pas d'autoréférencement on peut développer les concepts non-atomiques. Nous implémentons le prédicat `developp(C,D)` qui est vraie ssi D est le developpement de C. Le cas de base est le suivant :

```
developpe(C, C) :- cnamea(C).
```

On peut ensuite écrire le prédicat `transforme(L1,L2)` qui prend en argument deux box L1 et L2 et qui est vraie ssi L2 est la box équivalent à L1 mais dans laquelle les concepts non-atomiques sont développés puis mis sous formes normales négatives :

```
transforme([], []).  
transforme([(X,C) | L], [(X,D) | M]) :-  
    developpe(C, E),  
    nnf(E, D),  
    transforme(L, M).
```

4 Saisie de la proposition à montrer

5 Démonstration de la proposition