Rapport
Bases de Connaissances

Selove OKE CODJO
Solène EHOLIE

04/06/2014

Table des matières

1 Introduction

2 Rendu des programmes réalisés en TP

2.1 TP1 : generation de plan

Le but du TP1 était de prendre en main la programmation en prolog à travers un problème simple mettant en scène un agent et un objet. L'agent est capable de se déplacer d'un point a à un point b, de prendre l'objet et de le poser. Il va donc falloir générer le plan à suivre par l'agent pour aller d'une certaine situation à une autre

2.1.1 Résolution du problème

La première chose à faire était de définir les différentes actions dont notre agent est capable. Une action étant définie par une specification, une liste de conditions nécessaire à son accomplissement, une liste de propriétés qui ne seront plus vraies et une liste de nouvelles propriétés.

Ainsi, on définit l'action aller demandant à un robot de se deplacer d'un point X à un point Y. il faut donc que le robot soit en X, propriété qui deviendra fausse après le déplacement où il se retrouve en Y. La définition est donnée ci-dessous.

```
action(aller(robot,X,Y),
       [lieu(robot) = X],
       [lieu(robot) = X], [lieu(robot) = Y]) :-
           member(X,[a,b]), member(Y,[a,b]), X \= Y.
```

Ensuite on définit prendre, il faut que le robot et la boite soient au même endroit et que la main du robot soit libre, bien sûr la main n'est plus libre aprè l'action et l'objet n'est plus à l'endroit où il était mais dans la main du robot.

```
action(prendre(robot,O),
       [lieu(robot) = L, lieu(O) = L, libre(main(robot))],
       [libre(main(robot)), lieu(O)=L],
       [lieu(O)=main(robot)]) :-
           member(L,[a,b]), member(O,[boite]).
```

Enfin, on définit poser, l'objet doit être dans la main du robot avant de la quitter et se retrouver à la même position que le robot.

```
action(poser(robot,O),
       [lieu(robot) = L, lieu(O) = main(robot)],
       [lieu(O)=main(robot)],
       [lieu(O) = L, libre(main(robot))]) :-
           member(L, [a,b]), member(O, [boite]).
```

Il faut maintenant decrire ce que c'est qu'une transition entre un état E et un autre F, cele consiste juste en la réalisation d'une action dans E, il faut donc que les conditions soient vérifiées en E, il faut supprimer les propriétés qui ne seront plus vérifiées et ajouter les nouvelles. On a donc :

```
transition(A,E,F) :- action(A, C, S, AJ), verifcond(C,E),
                     suppress(S, E, EI), ajouter(AJ,EI,F).
```

La fonction verifcond(C,E) vérifiée l'inclusion de C dans E, suppress(S, E, EI) supprime S de E pour donner EI et ajouter(AJ,EI,F) ajoute AJ à EI pour obtenir F.

```
%verifcond(C,E) est l inclusion de C dans E
verifcond([], _). %la liste vide est toujours incluse
verifcond([C|LC], L) :- member(C,L), verifcond(LC,L).
```

```
%suppression
suppress([],L,L).
suppress([X|Y], Z, T) :- delete(Z,X,U), suppress(Y,U,T).
```

```
%ajout
ajouter(AJ,E,F) :- union(AJ,E,F).
```

Pour finir, il nous faut generer un plan d'une certaine profondeur, et une autre qui nous permet d'entrer la profondeur que l'on veut sans avoir à modifier le code ces deux fonctions sont données ci-dessous :

```
%generation de plan
genere(E,F,[A],1):-
    transition(A,E,F).

genere(EI,EF,[ACT|PLAN],M):-
   M > 1, transition(ACT,EI,E), N is M-1,
    between(1,N,P), genere(E,EF,PLAN,P).
```

```
planifier(Plan) :-
    init(E),
    but(B),
    nl,
    write('␣Profondeur␣limite␣:␣'),
    read(Prof),
    nl,
    genere(E,F,Plan,Prof),
    verifcond(B,F).
```

2.1.2 Tests

2.2 TP2 : planification multi-agents

L'objectif ici était de faire de la planification comme au TP1 à la seule différence qu'il y a ici plusieurs agents différents. Ce qui offre un nombre de situations plus élevé et des cas plus complexes à traiter.

2.2.1 Résolution du problème

La différence principale avec le TP1 réside dans la définition des actions. On a plusieurs agents qui ont des capacités différentes, il faudra donc spécifier les agents capables d'effectuer telle ou telle autre action. De plus, concernant les lieux, un objet peut se retrouver dans la 'main' de tel ou tel autre agent, il faut donc un moyen de distinguer les 'mains'. Les objets peuvent aussi être empilés les uns sur les autres. La définition des action est donc un peu plus complexe comme nous allons le voir dans ce qui suit.

La première action définie est $aller\_a\_vide(R, Ld, La)$. C'est la possibilité pour un agent de se déplacer de la position Ld vers la position La sans transporter d'objet, ce qui constitue la seule différence avec l'action aller du TP1. Seul le robot1 sait le faire.

```
action( aller_a_vide(R,Ld,La),
      [position(R) = Ld, libre(main(R))],
      [position(R) = Ld],
      [position(R) = La] ) :-
      member(Ld,[a,b,c]), member(La,[a,b,c]),
      member(R,[robo1]), Ld \= La.
```

Ensuite il fallait définir $transporter(R, Ld, La, O)$ qui permet à un agent R d'aller de la position Ld à la position La en transportant l'objet O. Il faut donc que l'agent et l'objet soient en Ld et qu'il tienne l'objet O. Après, ils se retrouvent tout les deux en La. Le code est donné dans l'encadré ci-dessous.

```
action( transporter(R,Ld,La,O),
      [position(R) = Ld, position(O) = Ld, lieu(O) = main(R)],
      [position(R) = Ld, position(O) = Ld],
      [position(R) = La, position(O) = La] ) :-
      member(Ld,[a,b,c]), member(La,[a,b,c]), member(R,[robo1]),
      member(O,[cube1,cube2]), Ld \= La.
```

L'action $attraper(R, O, L)$ décrit le fait, pour un agent R, de prendre un objetO sur la table de la position L. Il faut que l'objet soit accessible, c'est à dire qu'il n'y ait pas d'objet au dessus, l'objet n'est donc plus sur la table, n'est plus accessible et la main du robot n'est plus libre. L'objet se trouve maintenant dans la main du robot R.

```
action( attraper(R,O,L),
      [position(R) = L, position(O) = L, sur(O,table(L)),
      accessible(O), libre(main(R))],
```

```
        [sur(O,table(L)), accessible(O), libre(main(R))],
        [lieu(O) = main(R)] ) :-
        member(L,[a,b,c]), member(R,[robo1, robo2, robo3]),
        member(O,[cube1,cube2]).
```

```
action( saisir(R,O,L),
        [position(R) = L, position(O) = L, sur(O,Osous),
        accessible(O), libre(main(R))],
        [sur(O,Osous), accessible(O), libre(main(R))],
        [lieu(O) = main(R), accessible(Osous)] ) :-
        member(L,[a,b,c]), member(R,[robo2, robo3]),
        member(O,[cube1,cube2]), member(Osous,[cube1,cube2]), O \= Osous.
```

```
action( deposer(R,O,L),
        [position(R) = L, position(O) = L,
        lieu(O) = main(R)],
        [lieu(O) = main(R)],
        [sur(O,table(L)), accessible(O), libre(main(R))] ) :-
        member(L,[a,b,c]), member(R,[robo1, robo2, robo3]),
        member(O,[cube1,cube2]).
```

```
action( empiler(R,Osur,Osous,L),
        [position(R) = L, position(Osur) = L, lieu(Osur) = main(R),
        accessible(Osous)],
        [lieu(Osur) = main(R), accessible(Osous)],
        [sur(Osur,Osous), accessible(Osur), libre(main(R))] ) :-
        member(L,[a,b,c]), member(R,[robo2, robo3]),
        member(Osur,[cube1,cube2]), member(Osous,[cube1,cube2]), Osur \= Osous.
```

2.2.2 Tests

2.3 TP3 : générateur-démonstrateur en logique modale

2.3.1 Résolution du problème

2.3.2 Tests

2.4 TP4 : générateur de plan en logique modale

2.4.1 Résolution du problème

2.4.2 Tests

3 Travaux réalisés lors du bureau d'étude

3.1 Etude de Lm{p}

3.2 Etude de Lm{E0}

3.3 Etude d'autres logiques

4 Conclusion