

Rapport Bases de Connaissances

Selove OKE CODJO
Solène EHOLIE

04/06/2014

Table des matières

1	Introduction	3
2	Rendu des programmes réalisés en TP	3
2.1	TP1 : generation de plan	3
2.1.1	Résolution du problème	3
2.1.2	Tests	5
2.2	TP2 : planification multi-agents	5
2.2.1	Résolution du problème	5
2.2.2	Tests	5
2.3	TP3 : générateur-démonstrateur en logique modale	5
2.3.1	Résolution du problème	5
2.3.2	Tests	5
2.4	TP4 : générateur de plan en logique modale	5
2.4.1	Résolution du problème	5
2.4.2	Tests	5
3	Travaux réalisés lors du bureau d'étude	5
3.1	Etude de $Lm\{p\}$	5
3.2	Etude de $Lm\{E0\}$	5
3.3	Etude d'autres logiques	5
4	Conclusion	5

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 spécification, 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 déplacer 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 boîte soient au même endroit et que la main du robot soit libre, bien sûr la main n'est plus libre après 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 dire ce que c'est qu'une transition entre un état E et un autre F, cela 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) vérifie 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

2.2.1 Résolution du problème

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