ROB316-TP5

Mengyu Pan

January 2021

1 Exercise 1

1.1 Que signifient les quatre opérateurs du fichier de domaine?

Les quarte opérateurs sont:

- pick-up: enlaver un cube
- put-down: mettre un cube sur l'autre cube
- stack: enlaver un cube et mettre ce cube en stockage
- unstack: mettre un cube de stockage

Ils sont les actions qu'on peut faire dans cette domaine.

1.2 Quelle est la différence entre l'opérateur put-down et l'opérateur stack ? Pourquoi faut-il dissocier ces deux cas ?

L'opérateur de put-down s'agit de déplacer l'objet sur l'autre cube et l'opérateur stack s'agit de déplacer l'objet en stockage.

Les deux cas dovient être dissociés car ils sont deux opérateurs différents et ils ont des besoins des parameters différents pour éxecuter.

1.3 Que veut dire le fluent (holding ?x) ? A quoi sert-il ?

Le fluent (holding ?x) s'agit un état de robot: ce robot est en train de maintenir quelque cube. Si ce fluent n'était pas là, il faudrait ajouter la destination de l'opérateur. Donc, c'est à dire, il faudrait combiner les opérateurs d'enlaver un cube et les opérateurs de déplacer un cube.

2 Exercise 2

J'éxecute ce command: $cpt.exe - odomain - blocksaips.pddl - fblocksaips01.pddl$
Les résultats obtenus sont:

- la longueur du plan-solution est 6
- le temps total est 0.04
- l'itération est 1
- le temps dure chaque action du plan-solution est 1

Le plan-solution le plus longue qu'on peut imaginer est le plan-solution qui évite tous les bon choix directes. Il n'est pas fourni par ce planificateur car quand ce planificateur trouve le bon plan il le rapporte et se temine.

3 Exercise 3

J'éxecute ce command: $cpt.exe - odomain - blocksaips.pddl - fex3.pddl$ Le code est:

```
(define (problem BLOCKS-4-0)
(:domain BLOCKS)
(:objects D B A C )
(:INIT (CLEAR B)(ON B C) (ON C A) (ON A D) (ONTABLE D) (HANDEMPTY))
(:goal (AND (ON D C) (ON C B) (ON B A)))
)
```

Les résultats obtenus sont:

- la longueur du plan-solution est 12
- le temps total est 0.06
- l'itération est 3

4 Exercise 4

J'éxecute ce command: $cpt.exe - odomain - blocksaips.pddl - fex4.pddl – t60$
Le code est:

```
(define (problem BLOCKS-10-0)
(:domain BLOCKS)
(:objects A B C D E F G H I J)
(:INIT (CLEAR C) (ON C G) (ON G E) (ON E I) (ON I J) (ON J A) (ON A B) (ONTABLE B)
 (CLEAR F) (ON F D) (ON D H) (ONTABLE H) (HANDEMPTY))
```

```
    (:goal (AND (ON C B) (ON B D) (ON D F) (ON F I)
        (ON I A) (ON A E) (ON E H) (ON H G) (ON G J)))
    )
```

Les résultats obtenus sont:

- la longueur du plan-solution est 32
- le temps total est 1.12
- l'itération est 7

5 Exercise 5

J'éxecute ce command: $cpt.exe - odomain - ex5.pddl - fex5.pddl$ Le code de définir ce question est:

```
    (define (problem TREE-9-0)
    (:domain TREE)
    (:objects A B C D)
    (:INIT (FROM A B) (FROM A C) (FROM C D) (FROM B C)
        (CLEAR B) (CLEAR C) (CLEAR D) (CHOOSING A))
    (:goal (AND (TO A B) (TO B C) (TO C D)))
    )
```

Je définis deux relations 'from' et 'to'. La relation 'from' est pour construire un arabe et la relation 'to' est de choisir un chemin. Le code de définir la domaine est:

```
    (define (domain TREE)
      (:requirements :strips)
      (:predicates (from ?x ?y)
                   (to ?x ?y)
                   (clear ?x)
                   (choosing ?x)
                   )

      (:action pick
                 :parameters (?x ?y)
                 :precondition (and (clear ?y) (choosing ?x) (from ?x ?y))
                 :effect
                 (and  (not (clear ?y))
                       (choosing ?y)
                       (to ?x ?y))))
```

A mon avis, cette méthode de planification de chemin n'est pas efficace car il va choisir le noeud au hasard.

6 Exercise 6

J'éxecute ce command: $cpt.exe - odomain - ex6.pddl - fex6.pddl$ Le code de définir ce question est:

```
(define (problem SIGNE-3-0)
(:domain SIGNE)
(:objects A B C)
(:INIT (PosSinge A) (PosCaisse B) (PosBanane C) (Bas) (Handempty)
    (not (Haut)) (not (HadBanane)))
(:goal (AND (PosSinge C) (Haut) (HadBanane)))
)
```

Il y a six opérateurs:

- 'Aller': changer la position de signe
- 'Pousser': changer la position de caisse
- 'Monter' et 'Descendre': changer l'altitude de signe
- 'Attraper' et 'Lacher': obtenir ou déplacer la banane

Le code de définir la domaine avec six opérateurs est:

```
(define (domain SIGNE)
  (:requirements :strips)
  (:predicates (PosSinge ?x)
               (PosCaisse ?x)
               (PosBanane ?x)
               (Haut)
               (Bas)
               (Handempty)
               (HadBanane)
               )

  (:action Aller
            :parameters (?x ?y)
            :precondition (and (PosSinge ?x) (bas))
            :effect
            (and (not (PosSinge ?x))
                  (PosSinge ?y)))
  (:action Pousser
            :parameters (?x ?y)
            :precondition (and (PosSinge ?x) (PosCaisse ?x) (bas))
            :effect
            (and (not (PosCaisse ?x))
                  (PosCaisse ?y)
```

```
                        (bas)))
        (:action Monter
                   :parameters (?x)
                   :precondition (and (PosSinge ?x) (PosCaisse ?x) (bas))
                   :effect
                   (and (not (Bas))
                         (Haut)))
        (:action Descendre
                   :parameters (?x)
                   :precondition (and (PosSinge ?x) (PosCaisse ?x) (Haut))
                   :effect
                   (and (not (Haut))
                         (Bas)))
        (:action Attraper
                   :parameters (?x)
                   :precondition (and (PosSinge ?x) (PosBanane ?x) (Haut) (Handempty))
                   :effect
                   (and (not (Handempty))
                         (HadBanane)))
        (:action Lacher
                   :parameters (?x)
                   :precondition (and (PosSinge ?x) (PosBanane ?x) (Haut) (not(Handempty)))
                   :effect
                   (and (not (HadBanane))
                         (Handempty)))
)
```

Le plan-solution obtenu est:

- 0: (aller a b) [1]
- 1: (pousser b c) [1]
- 2: (aller b c) [1]
- 3: (monter c) [1]
- 4: (attraper c) [1]

Les résultats obtenus sont:

- la longueur du plan-solution est 32
- le temps total est 1.12
- l'itération est 7

7 Exercise 7

7.1 Code pour exercise 7

J'éxecute le command comme ça: $cpt.exe - odomain - ex7.pddl - fex7 - hanoi2.pddl$ Par exemple, le code de définir ce question avec deux disques est:

```
(define (problem HANOI-2-0)
(:domain HANOI)
(:objects P1 P2 P3 D1 D2)
(:INIT (clear D1) (on D1 D2) (onPic D2 P1) (picEmpty P2) (picEmpty P3)
(smaller D1 P1) (smaller D1 P2) (smaller D1 P3)
(smaller D2 P1) (smaller D2 P2) (smaller D2 P3)
(smaller D1 D2) (handempty)
)
(:goal (AND (on D1 D2) (onPic D2 P3)))
)
```

Il y a quatre opérateurs:

- 'pick-up': mettre le disque que tient la pince sur un disque plus grand et libre
- 'pick-up-pic': enlever avec la pince le seul disque d'un pic
- 'put-down': enlever avec la pince un disque libre du disque du dessous
- 'put-down-pic': mettre avec la pince le premier disque sur un pic vide

Le code de définir la domaine avec quatre opérateurs est:

```
(define (domain HANOI)
  (:requirements :strips)
  (:predicates (on ?x ?y)
               (onPic ?x ?y)
               (picEmpty ?x)
               (smaller ?x ?y)
               (clear ?x)
               (holding ?x)
               (handempty)
               )

  (:action pick-up
            :parameters (?x ?y)
            :precondition (and (clear ?x) (on ?x ?y) (handempty))
            :effect
            (and (not (on ?x ?y))
                  (holding ?x)
```

```
                        (clear ?y)
                        (not (clear ?x))
                        (not (handempty))
                        ))
      (:action pick-up-pic
                  :parameters (?x ?y)
                  :precondition (and (clear ?x) (onPic ?x ?y) (handempty))
                  :effect
                  (and (not (onPic ?x ?y))
                        (picEmpty ?y)
                        (not (clear ?x))
                        (not (handempty))
                        (holding ?x)
                        ))

      (:action put-down
                  :parameters (?x ?y)
                  :precondition (and (clear ?y) (smaller ?x ?y) (holding ?x))
                  :effect
                  (and  (not (holding ?x))
                        (clear ?x)
                        (not (clear ?y))
                        (handempty)
                        (on ?x ?y)))
      (:action put-down-pic
                  :parameters (?x ?y)
                  :precondition (and (picEmpty ?y) (holding ?x))
                  :effect
                  (and  (not (holding ?x))
                        (not (picEmpty ?y))
                        (clear ?x)
                        (handempty)
                        (onPic ?x ?y)))

)
```

7.2 Résultat pour exercise 7

Les résultats obtenus sont dans la table 1. J'ai essayé la commande $cpt.exe - odomain - ex7.pddl - fex7 - hanoi4.pddl - t10000$, mais il n'a pas réussi de trouver une bonne résolution. Pour résoudre les tours de Hanoi pour 5 disques, il faut prendre beaucoup plus de temps.

7.3 la fonction de code en pseudo

Dans le fonction, si le nombre de disque est plus grand que zéro, d'abord il déplace les moin grands disques au pic intermediaire et ne change pas la position

nombre disques	longueur	temps total	itération
1	2	0.03	1
2	6	0.04	1
3	14	0.13	5
4		plus que 100	plus que 35

Table 1: Résultat de hanoi

de le plus grand disque. Ensuite, il déplace le plus grand disque au pic arrivée et les autres disques au pic arrivée par la même manière qu'il a fait au première étape. Pour trouver la manière de déplacer les autre disques, il fait l'iteration. Je choisis deux disques comme un exemple et le code en pseudo:

7.4 la complexité algorithmique de la fonction

La complexité algorithmique de la fonction est $2^n - 1$ parce que cette fonction appelle deux fois lui-même. Ce ne correspond pas le résultat de l'exercise 7. Notre planificateur essaie de trouver le meilleur solution parmi les solutions disponibles donc il va faire plusieur itération

7.5 la différence entre l'algorithmique Hanoi et le CPT

La différence, comme déjà remarqué, est que le planificateur va trouver le meilleur solution parmi les solutions disponibles. Dans le code récusif, on a déjà connu la problème et on a choisi une méthode récusive pour resoudre la problème.

References