INSTITUT POLYTECHNIQUE DE PARIS
ENSTA PARIS

CSC_5RO16_TA, Planification et Contrôle

TP5, Planning Domain Definition Language

by
Guilherme NUNES TROFINO

supervised by
Philippe MORIGNOT
David FILLIAT

Confidentiality Notice
Non-confidential and publishable report

ROBOTIQUE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET COMMUNICATION

Paris, FR
21 janvier 2025

Table des matières

1. Question 1

1.1. Opérateurs

Ces opérateurs décrivent toutes les manipulations possibles dans un monde de cubes tout en respectant les contraintes physiques, par exemple : un bloc doit être dégagé pour pouvoir être manipulé.

1.1.1. `pick-up`

> **Résolution.** Permet de saisir un bloc `?x` dégagé, posé sur la table, lorsque la main du robot est vide. Après cette action, le bloc est tenu par le robot, n'est plus sur la table et n'est plus dégagé.

1.1.2. `put-down`

> **Résolution.** Permet de déposer un bloc `?x` tenu par le robot sur la table. Après cette action, le bloc est sur la table, devient dégagé, et la main du robot redevient vide.

1.1.3. `stack`

> **Résolution.** Permet de placer un bloc `?x` situé sur un autre bloc `?y`, à condition que `?x` soit dégagé et que la main du robot soit vide. Après cette action, le bloc `?x` est tenu par le robot, `?y` devient dégagé, `?x` n'est plus sur `?y`.

1.1.4. `unstack`

> **Résolution.** Permet de retirer un bloc `?x` situé sur un bloc `?y`, à condition que `?x` soit dégagé et que la main du robot soit vide. Après cette action, le bloc `?x` est tenu par le robot, `?y` devient dégagé, et `?x` n'est plus sur `?y`.

1.2. `put-down` vs `stack`

L'opérateur put-down permet de déposer un bloc sur la table sans interagir avec d'autres blocs. En revanche, l'opérateur stack implique une interaction directe entre les blocs, car il place un bloc sur un auatre bloc qui doit être dégagé.

Ces deux cas représentent des actions fondamentalement différentes dans le monde des blocs. En dissociant ces opérateurs, on met en évidence les contraintes spécifiques à chaque situation :

1. put-down n'impose pas de vérifier si un autre bloc est dégagé, car la table est toujours disponible.
2. stack, en revanche, exige que le bloc cible soit dégagé, ajoutant une contrainte supplémentaire.

Cette distinction permet une représentation précise des contraintes physiques et évite toute ambiguïtés dans les plans générés.

1.3. `holding`

Le fluent `(holding ?x)` indique que le robot tient actuellement le bloc `?x`. Il joue un rôle essentiel pour modéliser l'état de la main du robot et garantir la cohérence des actions.

Exemple 1.1. Une action comme put-down ou stack ne peut être réalisée que si le robot tient déjà un bloc. Cela empêche qu'un bloc soit placé sans qu'il ait été préalablement saisi.

Sans `(holding ?x)`, il serait nécessaire de redéfinir les opérateurs pour inclure des préconditions ou effets complexes afin de suivre indirectement l'état de la main du robot. Cela impliquerait :

1. Ajouter une variable implicite ou un état global décrivant le contenu de la main.
2. Multiplier les conditions liées aux transitions entre l'état vide et occupé de la main.

En somme, l'absence de `(holding ?x)` compliquerait la représentation, rendant les opérateurs plus difficiles à interpréter.

2. Question 2

L'algorithme a été lancé avec la commande suivante sur Windows PowerShell :

```
.\cpt.exe -o .\domain-blocksaips.pddl -f .\blocksaips01.pddl
```

Listing 1 : Lancement `cpt.exe` pour `blocksaips01.pddl`

Qui a retourné comme résultat le suivant :

```
domain file : .\domain-blocksaips.pddl
problem file : .\blocksaips01.pddl

Parsing domain..... done : 0.00
Parsing problem..... done : 0.00
domain : blocks
problem : blocks-4-0
Instantiating operators..... done : 0.00
Creating initial structures..... done : 0.00
Computing bound..... done : 0.00
Computing e-deleters..... done : 0.00
Finalizing e-deleters..... done : 0.00
Refreshing structures..... done : 0.00
Computing distances..... done : 0.00
Finalizing structures..... done : 0.00
Variables creation..... done : 0.00
Bad supporters..... done : 0.00
Distance boosting..... done : 0.00
Initial propagations..... done : 0.00

Problem : 34 actions, 25 fluents, 79 causals
          9 init facts, 3 goals

Bound : 6  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00

0: (pick-up b) [1]
1: (stack b a) [1]
2: (pick-up c) [1]
3: (stack c b) [1]
4: (pick-up d) [1]
5: (stack d c) [1]

Makespan : 6
Length : 6
Nodes : 0
Backtracks : 0
Support choices : 0
Conflict choices : 0
Mutex choices : 0
Start time choices : 0
World size : 100K
Nodes/sec : 0.00
Search time : 0.00
Total time : 0.02
```

Listing 2 : Résultat `cpt.exe` pour `blocksaips01.pddl`

2.1. Plan-Solution

2.1.1. Longueur

Résolution. La longueur du plan-solution est de 6 actions, correspondant à une séquence minimale permettant d'atteindre l'objectif

```
Length : 6
```

2.1.2. Temps d'Exécution

Résolution. Le plan-solution a été trouvé en 0,00 seconde de temps de recherche, avec un temps total d'exécution y compris le parsing et les calculs initiaux de 0,02 seconde.

```
Search time : 0.00
Total time : 0.02
```

2.1.3. Itérations

Résolution. Le planificateur a effectué 1 itérations pour trouver la solution. Cela indique qu'aucune exploration ou backtracking n'a été nécessaire.

```
Bound : 6  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00
```

2.1.4. Durée d'Action

Résolution. Dans ce cas, chaque action est considérée comme prenant 1 unité de temps, ce qui est arbitraire et dépend de la manière dont le domaine est modélisé.

Le `Makespan` total est donc de 6 unités de temps, correspondant aux 6 actions du plan.

```
Bound : 6  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00

0: (pick-up b) [1]
1: (stack b a) [1]
2: (pick-up c) [1]
3: (stack c b) [1]
4: (pick-up d) [1]
5: (stack d c) [1]

Makespan : 6
```

2.2. Longueur Solution

Résolution. Des plans plus longs pourraient exister si des actions supplémentaires ou inutiles étaient insérées.

Exemple 2.1. Ramener un bloc sur la table avant de le replacer sur une autre position :

```
(pick-up b) (put-down b) (pick-up b) (stack b a)
```

Changer l'ordre des blocs temporairement avant de revenir à l'ordre final.

Ces plans ne sont pas proposés car le planificateur optimise pour trouver une solution minimale en termes de longueur ou de `Makespan`. Cela est dû à l'utilisation d'heuristiques qui privilégient les solutions les plus efficaces en respectant les contraintes du domaine. Fournir des plans plus longs irait à l'encontre de cette logique d'optimalité.

Le planificateur génère un plan optimal en minimisant la longueur et le temps. Les plans plus longs, bien que possibles, ne sont pas proposés car ils n'ajoutent aucune valeur à la résolution du problème et ne respectent pas les objectifs d'efficacité.

3. Question 3

3.1. Algorithme

```
(define (problem BLOCKS-4-1)
  (:domain BLOCKS)
  (:objects A B C D)
  (:init (on B C) (on C A) (on A D) (ontable D)
         (clear B) (handempty))
  (:goal (and (on D C) (on C A) (on A B)))
)
```

Listing 3 : Algorithme `blocksaips02.pddl`

3.2. Analyse

L'algorithme a été lancé avec la commande suivante sur Windows PowerShell :

```
    .\cpt.exe -o .\domain-blocksaips.pddl -f .\blocksaips02.pddl
```

Listing 4 : Lancement `cpt.exe` pour `blocksaips02.pddl`

Qui a retourné comme résultat le suivant :

```
    domain file : .\domain-blocksaips.pddl
    problem file : .\blocksaips02.pddl

    Parsing domain..................... done : 0.00
    Parsing problem.................... done : 0.00
    domain : blocks
    problem : blocks-4-1
    Instantiating operators............ done : 0.00
    Creating initial structures........ done : 0.00
    Computing bound.................... done : 0.00
    Computing e-deleters............... done : 0.00
    Finalizing e-deleters.............. done : 0.00
    Refreshing structures.............. done : 0.00
    Computing distances................ done : 0.00
    Finalizing structures.............. done : 0.00
    Variables creation................. done : 0.00
    Bad supporters..................... done : 0.00
    Distance boosting.................. done : 0.00
    Initial propagations............... done : 0.00

    Problem : 34 actions, 25 fluents, 79 causals
              6 init facts, 3 goals

    Bound : 10  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00

    0: (unstack b c) [1]
    1: (put-down b) [1]
    2: (unstack c a) [1]
    3: (put-down c) [1]
    4: (unstack a d) [1]
    5: (stack a b) [1]
    6: (pick-up c) [1]
    7: (stack c a) [1]
    8: (pick-up d) [1]
    9: (stack d c) [1]

    Makespan : 10
    Length : 10
    Nodes : 0
    Backtracks : 0
    Support choices : 0
    Conflict choices : 0
    Mutex choices : 0
    Start time choices : 0
    World size : 100K
    Nodes/sec : 0.00
    Search time : 0.00
    Total time : 0.02
```

Listing 5 : Résultat `cpt.exe` pour `blocksaips02.pddl`

3.2.1. Longueur

Résolution. La longueur du plan-solution est de 10 actions, correspondant à une séquence minimale permettant d'atteindre l'objectif

```
Length : 10
```

3.2.2. Temps d'Exécution

Résolution. Le plan-solution a été trouvé en 0,00 seconde de temps de recherche, avec un temps total d'exécution y compris le parsing et les calculs initiaux de 0,02 seconde.

```
Search time : 0.00
Total time : 0.02
```

3.2.3. Itérations

Résolution. Le planificateur a effectué 1 itérations pour trouver la solution. Cela indique qu'aucune exploration ou backtracking n'a été nécessaire.

```
Bound : 10  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00
```

4. Question 4

4.1. Algorithme

```
(define (problem BLOCKS-MULTI-TOWERS)
    (:domain BLOCKS)
    (:objects A B C D E F G H I J)
    (:init
        ;; Tour 1
        (on C G) (on G E) (on E I) (on I J) (on J A) (on A B) (ontable B)
        ;; Tour 2
        (on F D) (on D H) (ontable H)
        ;; Propriétés initiales
        (clear C) (clear F) (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)
        )
    )
)
```

Listing 6 : Algorithme `blocksaips03.pddl`

4.2. Analyse

L'algorithme a été lancé avec la commande suivante sur Windows PowerShell :

```
.\cpt.exe -o .\domain-blocksaips.pddl -f .\blocksaips03.pddl
```

Listing 7 : Lancement `cpt.exe` pour `blocksaips03.pddl`

Qui a retourné comme résultat le suivant :

```
domain file : .\domain-blocksaips.pddl
problem file : .\blocksaips03.pddl

Parsing domain..... done : 0.00
Parsing problem..... done : 0.00
domain : blocks
problem : blocks-multi-towers
Instantiating operators..... done : 0.00
Creating initial structures..... done : 0.00
Computing bound..... done : 0.00
Computing e-deleters..... done : 0.00
Finalizing e-deleters..... done : 0.00
Refreshing structures..... done : 0.00
Computing distances..... done : 0.00
Finalizing structures..... done : 0.00
Variables creation..... done : 0.00
Bad supporters..... done : 0.00
Distance boosting..... done : 0.00
Initial propagations..... done : 0.00

Problem : 202 actions, 121 fluents, 499 causals
          13 init facts, 9 goals

Bound : 26  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00
Bound : 27  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00
Bound : 28  ---  Nodes : 29  ---  Backtracks : 29  ---  Iteration time : 0.01
Bound : 29  ---  Nodes : 36  ---  Backtracks : 36  ---  Iteration time : 0.01
Bound : 30  ---  Nodes : 782  ---  Backtracks : 782  ---  Iteration time : 0.24
Bound : 31  ---  Nodes : 787  ---  Backtracks : 787  ---  Iteration time : 0.23
Bound : 32  ---  Nodes : 166  ---  Backtracks : 129  ---  Iteration time : 0.04

0: (unstack c g) [1]
1: (put-down c) [1]
2: (unstack g e) [1]
3: (put-down g) [1]
4: (unstack e i) [1]
5: (put-down e) [1]
6: (unstack i j) [1]
```

```
    7: (put-down i) [1]
    8: (unstack j a) [1]
    9: (put-down j) [1]
    10: (pick-up g) [1]
    11: (stack g j) [1]
    12: (unstack f d) [1]
    13: (put-down f) [1]
    14: (unstack d h) [1]
    15: (put-down d) [1]
    16: (pick-up h) [1]
    17: (stack h g) [1]
    18: (pick-up e) [1]
    19: (stack e h) [1]
    20: (unstack a b) [1]
    21: (stack a e) [1]
    22: (pick-up i) [1]
    23: (stack i a) [1]
    24: (pick-up f) [1]
    25: (stack f i) [1]
    26: (pick-up d) [1]
    27: (stack d f) [1]
    28: (pick-up b) [1]
    29: (stack b d) [1]
    30: (pick-up c) [1]
    31: (stack c b) [1]

    Makespan : 32
    Length : 32
    Nodes : 1800
    Backtracks : 1763
    Support choices : 510
    Conflict choices : 1290
    Mutex choices : 0
    Start time choices : 0
    World size : 300K
    Nodes/sec : 3333.33
    Search time : 0.54
    Total time : 0.56
```

Listing 8 : Résultat `cpt.exe` pour `blocksaips03.pddl`

4.2.1. Longueur

Résolution. La longueur du plan-solution est de 32 actions, correspondant à une séquence minimale permettant d'atteindre l'objectif

```
    Length : 32
```

4.2.2. Temps d'Exécution

Résolution. Le plan-solution a été trouvé en 0,54 seconde de temps de recherche, avec un temps total d'exécution y compris le parsing et les calculs initiaux de 0,56 seconde.

```
    Search time : 0.54
    Total time : 0.56
```

4.2.3. Itérations

Résolution. Le planificateur a effectué 7 itérations pour trouver la solution. Cela indique qu'aucune exploration ou backtracking n'a été nécessaire.

```
    Bound : 26  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00
    Bound : 27  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00
    Bound : 28  ---  Nodes : 29  ---  Backtracks : 29  ---  Iteration time : 0.01
    Bound : 29  ---  Nodes : 36  ---  Backtracks : 36  ---  Iteration time : 0.01
    Bound : 30  ---  Nodes : 782  ---  Backtracks : 782  ---  Iteration time : 0.24
    Bound : 31  ---  Nodes : 787  ---  Backtracks : 787  ---  Iteration time : 0.23
    Bound : 32  ---  Nodes : 166  ---  Backtracks : 129  ---  Iteration time : 0.04
```

5. Question 5

5.1. Algorithme

5.1.1. `domain-graph`

```
(define (domain GRAPH)
    (:requirements :strips)
    (:predicates
        (agent-at ?node)
        (arc ?from ?to)
    )
    (:action move
        :parameters (?from ?to)
        :precondition (and (agent-at ?from) (arc ?from ?to))
        :effect (and
            (not (agent-at ?from))
            (agent-at ?to)
        )
    )
)
```

Listing 9 : Algorithme `domain-graph.pddl`

5.1.2. `problem-graph`

```
(define (problem SMALL-GRAPH)
    (:domain GRAPH)
    (:objects A B C D E)
    (:init
        (agent-at A)

        (arc A B)
        (arc B C)
        (arc C D)
        (arc D E)
    )
    (:goal (agent-at E))
)
```

Listing 10 : Algorithme `problem-graph.pddl`

5.2. Analyse

L'algorithme a été lancé avec la commande suivante sur Windows PowerShell :

```
    .\cpt.exe -o .\domain-graph.pddl -f .\problem-graph.pddl
```

Listing 11 : Lancement `cpt.exe` pour `problem-graph.pddl`

Qui a retourné comme résultat le suivant :

```
    domain file : .\CSC_5RO16_TA_TP5_src_domain-graph.pddl
    problem file : .\CSC_5RO16_TA_TP5_src_problem-graph.pddl

    Parsing domain..... done : 0.00
    Parsing problem..... done : 0.00
    domain : graph
    problem : small-graph
    Instantiating operators..... done : 0.00
    Creating initial structures..... done : 0.00
    Computing bound..... done : 0.00
    Computing e-deleters..... done : 0.00
    Finalizing e-deleters..... done : 0.00
    Refreshing structures..... done : 0.00
    Computing distances..... done : 0.00
    Finalizing structures..... done : 0.00
    Variables creation..... done : 0.00
    Bad supporters..... done : 0.00
    Distance boosting..... done : 0.00
```

```
    Initial propagations.............. done : 0.00

    Problem : 6 actions, 5 fluents, 5 causals
              1 init facts, 1 goals

    Bound : 4  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00

    0: (move a b) [1]
    1: (move b c) [1]
    2: (move c d) [1]
    3: (move d e) [1]

    Makespan : 4
    Length : 4
    Nodes : 0
    Backtracks : 0
    Support choices : 0
    Conflict choices : 0
    Mutex choices : 0
    Start time choices : 0
    World size : 100K
    Nodes/sec : 0.00
    Search time : 0.00
    Total time : 0.02
```

Listing 12 : Résultat `cpt.exe` pour `problem-graph.pddl`

La modélisation sous forme d'actions PDDL est intuitive pour les graphes orientés avec un chemin calculé souvent minimal en termes de longueur.

Par contre,pour des graphes de grande taille, cette méthode devient inefficace car les planificateurs doivent explorer un espace d'état potentiellement exponentiel.

> **Remarque.** Si les arcs ont des poids, cette méthode nécessite des adaptations pour inclure une minimisation des coûts.

La méthode est efficace pour des petits graphes avec des objectifs simples, mais elle est peu adaptée aux grands graphes ou aux problèmes avec des coûts d'arcs. Dans ces cas, des algorithmes spécialisés comme Dijkstra ou A* sont plus performants.

6. Question 6

6.1. Algorithme

6.1.1. domain-monkey

```
(define (domain MONKEY)
    (:requirements :strips)
    (:predicates
        (vide)
        (singe ?from)
        (singe-bas)
        (singe-caisse ?from)
        (caisse ?from)
        (bananes ?from)
        (attrape ?objet)
    )
    (:action aller
        :parameters (?from ?to)
        :precondition (and
            (singe ?from)
            (singe-bas)
        )
        :effect (and
            (not (singe ?from))
            (singe ?to)
        )
    )
    (:action pousser
        :parameters (?objet ?from ?to)
        :precondition (and
            (singe ?from)
            (caisse ?from)
            (singe-bas)
        )
        :effect (and
            (not (caisse ?from))
            (caisse ?to)
            (not (singe ?from))
            (singe ?to)
        )
    )
    (:action monter
        :parameters (?lieu)
        :precondition (and
            (singe ?lieu)
            (caisse ?lieu)
            (singe-bas)
        )
        :effect (and
            (not (singe-bas))
            (singe-caisse ?lieu)
        )
    )
    (:action descendre
        :parameters (?objet ?lieu)
        :precondition (and
            (singe ?lieu)
            (caisse ?lieu)
            (singe-caisse ?lieu)
        )
        :effect (and
            (singe-bas)
            (not (singe-caisse ?lieu))
        )
    )
    (:action attraper
        :parameters (?objet ?lieu)
        :precondition (and
            (singe ?lieu)
            (bananes ?lieu)
            (singe-caisse ?lieu)
            (vide)
        )
        :effect (and
            (not (vide))
            (attrape ?objet)
```

```
        )
    )
    (:action lacher
        :parameters (?objet ?lieu)
        :precondition (and
            (attrape ?objet)
            (singe ?lieu)
        )
        :effect (and
            (not (attrape ?objet))
            (vide)
        )
    )
)
```

Listing 13 : Algorithme `domain-monkey.pddl`

6.1.2. problem-monkey

```
(define (problem MONKEY-BANANA)
    (:domain MONKEY)
    (:objects A B C bananes)
    (:init
        (singe A)
        (caisse B)
        (bananes C)
        (singe-bas)
        (vide)
    )
    (:goal
        (and
            (singe C)
            (caisse C)
            (attrape bananes)
        )
    )
)
```

Listing 14 : Algorithme `problem-monkey.pddl`

6.2. Analyse

L'algorithme a été lancé avec la commande suivante sur Windows PowerShell :

```
    .\cpt.exe -o .\domain-monkey.pddl -f .\problem-monkey.pddl
```

Listing 15 : Lancement `cpt.exe` pour `problem-monkey.pddl`

Qui a retourné comme résultat le suivant :

```
    domain file : .\CSC_5RO16_TA_TP5_src_domain-monkey.pddl
    problem file : .\CSC_5RO16_TA_TP5_src_problem-monkey.pddl

    Parsing domain..................... done : 0.00
    Parsing problem.................... done : 0.00
    domain : monkey
    problem : monkey-banana
    Instantiating operators........... done : 0.00
    Creating initial structures....... done : 0.00
    Computing bound.................... done : 0.00
    Computing e-deleters............... done : 0.00
    Finalizing e-deleters.............. done : 0.00
    Refreshing structures.............. done : 0.00
    Computing distances................ done : 0.00
    Finalizing structures.............. done : 0.00
    Variables creation................. done : 0.00
    Bad supporters..................... done : 0.00
    Distance boosting.................. done : 0.00
    Initial propagations............... done : 0.00

    Problem : 102 actions, 18 fluents, 275 causals
              4 init facts, 3 goals

    Bound : 4  ---  Nodes : 1  ---  Backtracks : 0  ---  Iteration time : 0.00

    0: (aller a b) [1]
    1: (pousser a b c) [1]
    2: (monter c) [1]
```

```
    3: (attraper bananes c) [1]

    Makespan : 4
    Length : 4
    Nodes : 1
    Backtracks : 0
    Support choices : 1
    Conflict choices : 0
    Mutex choices : 0
    Start time choices : 0
    World size : 100K
    Nodes/sec : 1000.00
    Search time : 0.00
    Total time : 0.02
```

Listing 16 : Résultat `cpt.exe` pour `problem-monkey.pddl`

6.2.1. Longueur

Résolution. La longueur du plan-solution est de 4 actions, correspondant à une séquence minimale permettant d'atteindre l'objectif

```
    Length : 4
```

6.2.2. Temps d'Exécution

Résolution. Le plan-solution a été trouvé en 0,00 seconde de temps de recherche, avec un temps total d'exécution y compris le parsing et les calculs initiaux de 0,02 seconde.

```
    Search time : 0.00
    Total time : 0.02
```

6.2.3. Itérations

Résolution. Le planificateur a effectué 1 itérations pour trouver la solution. Cela indique qu'aucune exploration ou backtracking n'a été nécessaire.

```
    Bound : 4  ---  Nodes : 1  ---  Backtracks : 0  ---  Iteration time : 0.00
```

6.2.4. Ramification

En effet, les préconditions du modèle ne spécifient pas toutes les conditions nécessaires pour que l'opérateur réussisse (comme le poids de l'objet ou la force du singe).

Ajouter de telles conditions rendrait le modèle très complexe, et certaines limitations doivent être implicitement comprises par l'utilisateur du modèle.

Exemple 6.1. Si l'objet à pousser est trop lourd, l'opérateur "pousser" n'aurait pas d'effet, car la précondition n'est pas satisfaite.

7. Question 7

7.1. Algorithme

7.1.1. domain-hanoi

```
(define (domain HANOI)
    (:requirements :strips)
    (:predicates
        (disk ?disk)
        (tower ?tower)

        (clear ?object)
        (on ?object_A ?object_B)
        (on_tower ?tower)
        (smaller ?object_A ?object_B)

        (tower_empty)
        (tower_full)
    )
    (:action remove_from_tower
        :parameters (?disk ?tower)
        :precondition (and
            (disk ?disk)
            (tower ?tower)
            (on ?disk ?tower)
            (clear ?disk)
            (tower_empty)
        )
        :effect (and
            (not (tower_empty))
            (on_tower ?disk)
            (tower_full)
            (clear ?tower)
            (not (on ?disk ?tower))
        )
    )
    (:action put_on_tower
        :parameters (?disk ?tower)
        :precondition (and
            (disk ?disk)
            (tower ?tower)
            (on_tower ?disk)
            (clear ?disk)
            (clear ?tower)
            (smaller ?disk ?tower)
        )
        :effect (and
            (tower_empty)
            (not (on_tower ?disk))
            (not (tower_full))
            (on ?disk ?tower)
            (not (clear ?tower))
        )
    )
    (:action remove_from_disk
        :parameters (?disk_A ?disk_B)
        :precondition (and
            (disk ?disk_A)
            (disk ?disk_B)
            (on ?disk_A ?disk_B)
            (clear ?disk_A)
            (tower_empty)
        )
        :effect (and
            (not (tower_empty))
            (on_tower ?disk_A)
            (tower_full)
            (clear ?disk_B)
            (not(on ?disk_A ?disk_A))
        )
    )
    (:action put_on_disk
        :parameters (?disk_A ?disk_B)
        :precondition (and
            (disk ?disk_A)
            (disk ?disk_B)
```

```
            (on_tower ?disk_A)
            (clear ?disk_A)
            (clear ?disk_B)
            (smaller ?disk_A ?disk_B)
        )
        :effect (and
            (tower_empty)
            (not (on_tower ?disk_A))
            (not (tower_full))
            (on ?disk_A ?disk_B)
            (not(clear ?disk_B))
        )
    )
)
```

Listing 17 : Algorithme `domain-hanoi.pddl`

7.1.2. problem-hanoi-1

```
(define (problem HANOI-1)
  (:domain HANOI)
  (:objects T1 T2 T3 D1)
  (:init

    (tower T1)  (tower T2)  (tower T3)
    (disk D1)
    (smaller D1 T1)
    (smaller D1 T2)
    (smaller D1 T3)

    (on D1 T1)
    (clear D1)  (clear T2)  (clear T3)

    (tower_empty)
  )
  (:goal  (and (on D1 T3))
  )
)
```

Listing 18 : Algorithme `problem-hanoi-1.pddl`

7.1.3. problem-hanoi-2

```
(define (problem HANOI-2)
  (:domain HANOI)
  (:objects T1 T2 T3 D1 D2)
  (:init

    (tower T1)  (tower T2)  (tower T3)
    (disk D1) (disk D2)
    (smaller D1 T1) (smaller D2 T1)
    (smaller D1 T2) (smaller D2 T2)
    (smaller D1 T3) (smaller D2 T3)
    (smaller D1 D2)

    (on D1 D2)  (on D2 T1)
    (clear D1)  (clear T2)  (clear T3)

    (tower_empty)
  )
  (:goal  (and (on D1 D2) (on D2 T3))
  )
)
```

Listing 19 : Algorithme `problem-hanoi-2.pddl`

7.1.4. problem-hanoi-3

```
(define (problem HANOI-3)
  (:domain HANOI)
  (:objects T1 T2 T3 D1 D2 D3)
  (:init

    (tower T1)  (tower T2)  (tower T3)
    (disk D1) (disk D2) (disk D3)
    (smaller D1 T1) (smaller D2 T1) (smaller D3 T1)
    (smaller D1 T2) (smaller D2 T2) (smaller D3 T2)
```

```
    (smaller D1 T3) (smaller D2 T3) (smaller D3 T3)
    (smaller D1 D2) (smaller D1 D3)
    (smaller D2 D3)

    (on D1 D2)  (on D2 D3)  (on D3 T1)
    (clear D1)  (clear T2)  (clear T3)

      (tower_empty)
  )
  (:goal  (and (on D1 D2) (on D2 D3) (on D3 T3))
  )
)
```

Listing 20 : Algorithme `problem-hanoi-3.pddl`

7.1.5. problem-hanoi-4

```
(define (problem HANOI-4)
  (:domain HANOI)
  (:objects T1 T2 T3 D1 D2 D3 D4)
  (:init

    (tower T1)  (tower T2)  (tower T3)
    (disk D1) (disk D2) (disk D3) (disk D4)
    (smaller D1 T1) (smaller D2 T1) (smaller D3 T1) (smaller D4 T1)
    (smaller D1 T2) (smaller D2 T2) (smaller D3 T2) (smaller D4 T2)
    (smaller D1 T3) (smaller D2 T3) (smaller D3 T3) (smaller D4 T3)
    (smaller D1 D2) (smaller D1 D3) (smaller D1 D4)
    (smaller D2 D3) (smaller D2 D4)
    (smaller D3 D4)

    (on D1 D2)  (on D2 D3)  (on D3 D4)  (on D4 T1)
    (clear D1)  (clear T2)  (clear T3)

    (tower_empty)
  )
  (:goal  (and (on D1 D2) (on D2 D3) (on D3 D4) (on D4 T3))
  )
)
```

Listing 21 : Algorithme `problem-hanoi-4.pddl`

7.1.6. problem-hanoi-5

```
(define (problem HANOI-5)
  (:domain HANOI)
  (:objects T1 T2 T3 D1 D2 D3 D4 D5)
  (:init

    (tower T1)  (tower T2)  (tower T3)
    (disk D1) (disk D2) (disk D3) (disk D4) (disk D5)
    (smaller D1 T1) (smaller D2 T1) (smaller D3 T1) (smaller D4 T1) (smaller D5 T1)
    (smaller D1 T2) (smaller D2 T2) (smaller D3 T2) (smaller D4 T2) (smaller D5 T2)
    (smaller D1 T3) (smaller D2 T3) (smaller D3 T3) (smaller D4 T3) (smaller D5 T3)
    (smaller D1 D2) (smaller D1 D3) (smaller D1 D4) (smaller D1 D5)
    (smaller D2 D3) (smaller D2 D4) (smaller D2 D5)
    (smaller D3 D4) (smaller D3 D5)
    (smaller D4 D5)

    (on D1 D2)  (on D2 D3)  (on D3 D4)  (on D4 D5)  (on D5 T1)
    (clear D1)  (clear T2)  (clear T3)

    (tower_empty)
  )
  (:goal  (and (on D1 D2) (on D2 D3) (on D3 D4) (on D4 D5) (on D5 T3))
  )
)
```

Listing 22 : Algorithme `problem-hanoi-5.pddl`

7.2. Analyse

Ci-dessous les résultats de chaque problème sont présentées :

algorithme	longueur	total time	itérations
problem-hanoi-1	2	0.01	1
problem-hanoi-2	4	0.01	1
problem-hanoi-3	8	0.02	3
problem-hanoi-4	16	0.93	9
problem-hanoi-5	-	-	-

TABLE 7.1 : Résultats Exécution Tour de Hanoi

Remarque. Ici l'aspect exponentiel du problème se fait présent car après quelques itérations le temps d'exécution croit exponentiellement ce qui rend impossible de trouver une réponse dans un temps raisonnable :

```
    Bound : 10  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00
    Bound : 11  ---  Nodes : 0  ---  Backtracks : 0  ---  Iteration time : 0.00
    Bound : 12  ---  Nodes : 17  ---  Backtracks : 17  ---  Iteration time : 0.00
    Bound : 13  ---  Nodes : 17  ---  Backtracks : 17  ---  Iteration time : 0.00
    Bound : 14  ---  Nodes : 217  ---  Backtracks : 217  ---  Iteration time : 0.02
    Bound : 15  ---  Nodes : 217  ---  Backtracks : 217  ---  Iteration time : 0.02
    Bound : 16  ---  Nodes : 3151  ---  Backtracks : 3151  ---  Iteration time : 0.27
    Bound : 17  ---  Nodes : 3156  ---  Backtracks : 3156  ---  Iteration time : 0.27
    Bound : 18  ---  Nodes : 48516  ---  Backtracks : 48516  ---  Iteration time : 4.44
    Bound : 19  ---  Nodes : 47390  ---  Backtracks : 47390  ---  Iteration time : 4.51
    Bound : 20  ---  Nodes : 779730  ---  Backtracks : 779730  ---  Iteration time : 83.69
    Bound : 21  ---  Nodes : 780813  ---  Backtracks : 780813  ---  Iteration time : 72.38
    Bound : 22  ---  Nodes : 13522955  ---  Backtracks : 13522955  ---  Iteration time : 1610.64
    Bound : 23  ---
```

Listing 23 : Résultat `cpt.exe` pour `problem-hanoi-5.pddl`

7.2.1. Pseudo Code

Cet algorithme proposé une solution du problème des tours de Hanoi en utilisant une approche récursive avec une número de mouvements de $2^N - 1$ où N est le nombre de disques.

Remarque. La complexité de cet algorithme est de $\emptyset(2^N)$.

Ce résultat peut-être observé sur l'exercice précedent car c'est une solution algorithme du problème des tours de Hanoi et le résultat proposé avant suit cet approache.

La proposition recursive exploite une structure mathématique spécifique au problème et génère directement la solution optimale. Par contre, la proposition CPT, explore un espace d'états plus général en appliquant des opérateurs et peut nécessiter plus d'itérations pour identifier la solution optimale.