Projet IAMSI

Planification ASP

Hocine Kadem 21309534

Neil Benahmed 21200977

Date : [19/03/2024]

Table des matières

1 Introduction

Dans le cadre de ce projet, nous nous sommes intéressés à la planification basé sur l'encodage en ASP (Answer Set Programming). Notre objectif était de créer un planificateur simple en utilisant le langage de planification PDDL (Planning Domain Definition Language). Pour cela, nous avons réalisé plusieurs exercices pratiques. Dans un premier temps, nous avons étudié la syntaxe du PDDL et créé des fichiers de domaine et de problème pour définir des scénarios de planification. En utilisant une interface Solveur, nous avons généré des plans correspondants à ces problèmes. Nous avons également exploré une variante simplifiée du monde des blocs, où nous avons réduit le nombre d'actions et généralisé les préconditions. Ensuite, nous avons abordé la planification par ASP en développant un moteur de planification basé sur le planificateur STRIPS. Nous avons défini les étapes nécessaires, telles que la représentation des états, des actions et des contraintes de planification. Enfin, nous avons créé un programme convertisseur qui traduit un domaine PDDL en un programme ASP-STRIPS en définissant les types, les prédicats, les actions, les préconditions, les effets et les objectifs. Ces exercices nous ont permis de mieux comprendre les concepts de base de la planification et de mettre en pratique notre compréhension en développant des solutions concrètes.

2 Présentation des fichiers du projet et execution du programme

2.1 Présentation des fichiers et des répertoires du projet

Le répertoire du projet comprend les éléments suivants :

- Répertoire 'solveur/' : Ce répertoire contient le solveur ASP clingo, qui est utilisé pour résoudre les problèmes de planification. Le solveur clingo génère des ensembles de réponses (answer sets) conformes aux contraintes spécifiées dans les fichiers ASP.

- Répertoire 'asp_manuel/' : Ce répertoire contient des fichiers ASP créés manuellement. Ces fichiers sont utilisés pour résoudre des problèmes de planification spécifiques en décrivant les contraintes et les règles logiques correspondantes.

- Répertoire 'asp/' : Ce répertoire contient les fichiers ASP générés à partir du programme principal en utilisant des fichiers PDDL en entrée. Ces fichiers ASP sont générés automatiquement en appliquant une logique spécifique pour la transformation des données.

- Fichier 'parserpddl.py' : Ce fichier Python contient une fonction de conversion de PDDL en ASP. Cette fonction permet de transformer les fichiers PDDL en fichiers ASP en utilisant une logique prédéfinie. Cette étape est essentielle pour préparer les données de planification pour le solveur ASP.

- Fichier 'main.py' : Ce fichier Python contient le programme principal du projet. Il gère le processus de transformation des fichiers PDDL en fichiers ASP en utilisant le module 'parserpddl.py'.

Ensuite, il appelle le solveur ASP pour résoudre les problèmes de planification en utilisant les fichiers ASP générés.

2.2 Execution du programme

Avant d'utiliser le programme principal, assurez-vous que le fichier `clingo` dans le répertoire `solveur/clingo-4.4.0-x86_64-linux/` a les permissions d'exécution. Si ce n'est pas le cas, vous pouvez accorder les permissions d'exécution en exécutant la commande suivante dans le terminal : `chmod 777 ./solveur/clingo-4.4.0-x86_64-linux/clingo`.

Pour résoudre un problème de planification dans le monde des blocs, exécutez la commande suivante dans le terminal :

```
python3 main.py ./plans/blockWorld-domain.pddl ./plans/blockWorld-problem.pddl ./asp/blockWorld
```

Pour résoudre un problème de planification dans le domaine des avions, exécutez la commande suivante dans le terminal :

```
python3 main.py ./plans/avion-domain.pddl ./plans/avion-problem.pddl ./asp/avion
```

Pour exécuter un autre fichier PDDL, utilisez la commande suivante dans le terminal :

```
python3 main.py [nom_fichier_domain.pddl] [nom_fichier_problem.pddl] [path_fichier_asp_généré]
```

Assurez-vous d'avoir installé les dépendances nécessaires pour exécuter le programme principal.

3 Représentation de probl‘eme de planification en PDDL

3.1 Exercice 1 : Prise en main de PDDL

Nous avons commencé par écrire les fichiers PDDL du problème et du domaine en utilisant le langage de description standardié PDDL.

Dans le fichier "blockWorld-domain.pddl", nous avons défini notre domaine de planification. Nous avons identifié les types de blocs et déclaré les prédicats nécessaires pour représenter l'état du monde, tels que "on", "ontable", "clear", "handempty" et "holding". De plus, nous avons défini les actions spécifiques telles que "pickup", "putdown", "stack" et "unstack", avec leurs préconditions et effets correspondants.

Ensuite, nous avons créé le fichier "blockWorld-problem.pddl" pour décrire notre problème spécifique. Nous avons déclaré les objets blocs (A, B, C, D) et défini l'état initial du problème, qui

comprend des informations sur la position des blocs, leur disponibilité et l'état de la main. Nous avons également spécifié l'état final souhaité, qui implique que les blocs doivent être clairs, empilés dans un ordre spécifique et que la main doit être vide.

Une fois les fichiers PDDL écrits, nous les avons soumis au solveur ENSHP (version 2020) pour générer un plan. Le solveur a effectué le processus de grounding, convertissant notre problème de planification en un problème de recherche bien défini. Ensuite, il a simplifié le problème en réduisant le nombre de fluentes et d'axiomes.

En utilisant l'algorithme de recherche heuristique "WA-star", le solveur a évalué les états et les actions pour trouver une solution au problème. Il a finalement généré le plan de solution suivant :

0.0 : (unstack b a)

1.0 : (stack b c)

2.0 : (pickup a)

3.0 : (stack a b)

On a aussi utilisé le solveur ASPPLAN qui nous a généré le plan suivant :

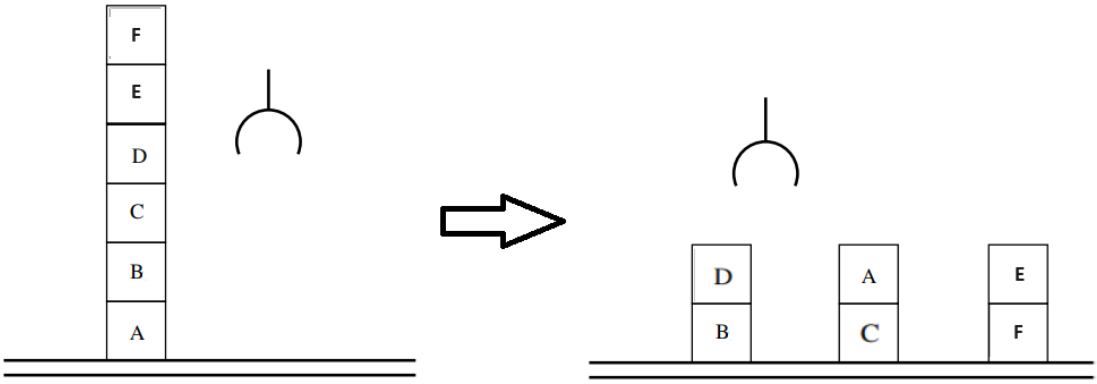
0 : (UNSTACK B A) [1]

1 : (STACK B C) [1]

2 : (PICK-UP A) [1]

3 : (STACK A B) [1]

Nous avons ensuite créé un problème plus complexe, comme le montre l'image suivante :



Le plan généré par le solveur ENSHP pour ce problème plus complexe est le suivant :

0.0 : (unstack f e)

1.0 : (put-down f)

2.0 : (unstack e d)

3.0 : (stack e f)

4.0 : (unstack d c)

5.0 : (stack d e)

6.0 : (unstack c b)

7.0 : (put-down c)

8.0 : (unstack d e)

9.0 : (stack d c)

10.0 : (unstack b a)

11.0 : (put-down b)

12.0 : (unstack d c)

13.0 : (stack d b)

14.0 : (pick-up a)

15.0 : (stack a c)

En suivant ce plan, nous atteignons l'état final souhaité pour le problème plus complexe, en respectant toutes les contraintes spécifiées.

3.2 Exercice 2 : Variante du monde des blocs

Nous avons proposé une version simplifiée du monde des blocs qui utilise uniquement deux actions : "moveTo(X,Y,Z)" et "moveToTable(X,Y)". Ces actions permettent de déplacer un bloc X qui est sur Y et de le mettre sur Z, ou de déplacer un bloc X qui est sur Y pour le mettre sur la table. Dans cette version simplifiée, les prédicats "handempty", "holding" et "ontable" ne sont plus nécessaires. Cependant, les prédicats "clear(Y)" et "on(X,Y)" doivent être généralisés pour s'appliquer également lorsque Y est la constante "table".

Pour adapter les fichiers PDDL existants à cette version simplifiée, nous avons créé les fichiers "blocksSimp-domain.pddl" et "blocksSimp-problemTD.pddl". Dans le fichier "blocksSimp-domain.pddl", nous avons modifié les actions pour utiliser les nouvelles actions simplifiées "moveTo" et "moveToTable". Nous avons également généralisé les prédicats "clear(Y)" et "on(X,Y)" pour qu'ils s'appliquent à la constante "table" en plus des blocs.

Dans le fichier "blocksSimp-problemTD.pddl", nous avons adapté l'état initial et l'état final en utilisant les prédicats et les actions de la version simplifiée. Nous avons déclaré les objets blocs (A, B, C) et spécifié l'état initial, qui comprend des informations sur la position des blocs. Nous avons également défini l'état final souhaité, indiquant que les blocs doivent être empilés dans un certain ordre.

4 Planification par ASP

4.1 Exercice 3 : Planificateur STRIPS

Dans le cadre de cet exercice, nous avons travaillé sur le développement d'un planificateur ASP-STRIPS capable de résoudre des problèmes de planification. Pour ce faire, nous avons créé des fichiers spécifiques aux deux problèmes donnés, ainsi qu'un fichier plus général appelé "planificateur_asp.lp".

4.1.1 Fichiers spécifiques aux problèmes

Afin de pour bien comprendre le problème, nous avons commencé par créé des fichiers spécifiques pour chaque problème donné. Ces fichiers contenaient des informations sur le domaine du problème

en utilisant des prédicats, des actions et des règles décrivant les préconditions et les effets des actions. Ces informations nous ont permis de représenter les états initiaux, les buts à atteindre, ainsi que les actions disponibles pour résoudre chaque problème.

4.1.2 Fichier "planificateur_asp.lp" général :

En plus des fichiers spécifiques aux problèmes donnés qu'on a fait dans le but de mieux comprnedre, nous avons surtout créé un fichier plus général appelé "planificateur_asp.lp". Ce fichier avait pour objectif de généraliser le planificateur pour qu'il puisse résoudre différents problèmes de planification.

Le fichier "planificateur_asp.lp" contenait des déclarations et des règles générales pour la gestion du temps, la spécification des actions disponibles, les préconditions et les effets des actions, ainsi que la définition des états initiaux et des buts à atteindre. Il utilisait des prédicats et des contraintes pour représenter les informations du problème de manière générique.

Nous avons utilisé des règles ASP pour définir les conditions de validité des actions à chaque étape de temps, les effets des actions sur les prédicats et les contraintes pour spécifier les choix d'actions à effectuer. Le planificateur a utilisé la programmation par ensembles de réponses pour générer un ensemble de réponses représentant un plan valide pour atteindre les buts spécifiés.

En utilisant ce fichier "planificateur_asp.lp" plus général, nous avons pu résoudre différents problèmes de planification en spécifiant simplement les détails spécifiques du problème dans des fichiers dédiés.

En conclusion, nous avons développé un planificateur ASP-STRIPS capable de résoudre des problèmes de planification en utilisant des fichiers spécifiques aux problèmes donnés, ainsi qu'un fichier plus général pour généraliser le planificateur. Ce planificateur utilise la programmation par ensembles de réponses pour générer des plans valides pour atteindre les buts spécifiés.

4.2 Exercice 4 : Parser PDDL à ASP

Nous avons implementer un traducteur de fichier PDDL en fichier ASP pour cela le parseur effectue les étapes suivante :

4.2.1 Traduction du Domain en ASP

1. Déclaration des prédicats : Les prédicats sont déclarés de la manière suivante : pred(nom_-prédicat(paramètres)) : type(paramètre) pour chaque paramètre.
2. Déclaration des actions :
   (a) Déclaration de l'action :

       Chaque action est déclarée de la manière suivante :

       action(nom_ action(paramètres)) : type_ paramètre(paramètre) pour chaque paramètre.

(b) Déclaration des préconditions de l'action :

pre(prédicat_ préconditions(paramètres) pour chaque précondition) : action(nom_ action(paramètres)).

(c) Déclaration des effets :

i. Ajout (Add) : Les ajouts sont déclarés avec :

add(prédicats à ajouter) :- action(nom_ action(paramètres)).

ii. Suppression (Del) : Les suppressions sont déclarées avec

del(prédicats à supprimer) :- action(nom_ action(paramètres)).

4.2.2 Traduction du Problem en ASP

1. Traitement des objets : On commence par instancier tous les objets présents, catégorisés par leur type, de la forme :

   type(objet).

2. Traitement des inits :

   Une fois les objets définis, nous décrivons l'état initial du problème. Les inits représentent la configuration de départ dans laquelle se trouvent les objets. Ils s'écrivent de la manière suivante :

   init(prédicat(paramètres)).

3. Traitement des buts :

   Nous définissons ensuite les objectifs qui décrivent l'état final souhaité des objets, de la manière suivante :

   but(prédicat(paramètres)) pour chaque élément présent dans la liste des buts.

4.3 Exercice 5

L'objectif de cet exercice est d'écrire un programme appelé ASPPLAN qui génère un plan minimal à partir de deux fichiers PDDL définissant le domaine et le problème. Le programme doit tester des valeurs de n croissantes jusqu'à trouver un plan, et mettre en forme l'answer set pour afficher lisiblement le plan obtenu.

On utilise un planificateur ASP (Answer Set Programming) et un solveur Clingo pour générer le plan. On effectue des itérations en augmentant progressivement la valeur de n jusqu'à ce qu'un plan soit trouvé. On modifie la cinquième ligne du fichier planificateur_asp.lp en y insérant la valeur de n correspondante à l'itération. Ensuite, on construit la commande en utilisant le solveur Clingo, les fichiers du domaine et du problème, ainsi que le fichier planificateur_asp.

Si le code de retour de la commande est 10 ou 30, cela indique que le planificateur a trouvé une solution satisfaisante. Dans ce cas, le résultat est formaté, trié et affiché, ainsi que le temps utilisé pour le parser et pour le générateur. Sinon, on continue d'itérer en augmentant la valeur de n jusqu'à ce qu'un plan soit trouvé ou que la valeur maximale n soit atteinte.

Nous avons réfléchi à implémenter une méthode de recherche par dichotomie, mais il n'est pas possible de le faire car l'insatisfiabilité n'est pas régulière par rapport à n.

4.3.1 Examples :

— Example avec blockWorld en lançant la commande :
python3 main.py ./pddl/blockWorld-domain.pddl ./pddl/blockWorld-problem.pddl ./asp/-blockWorld

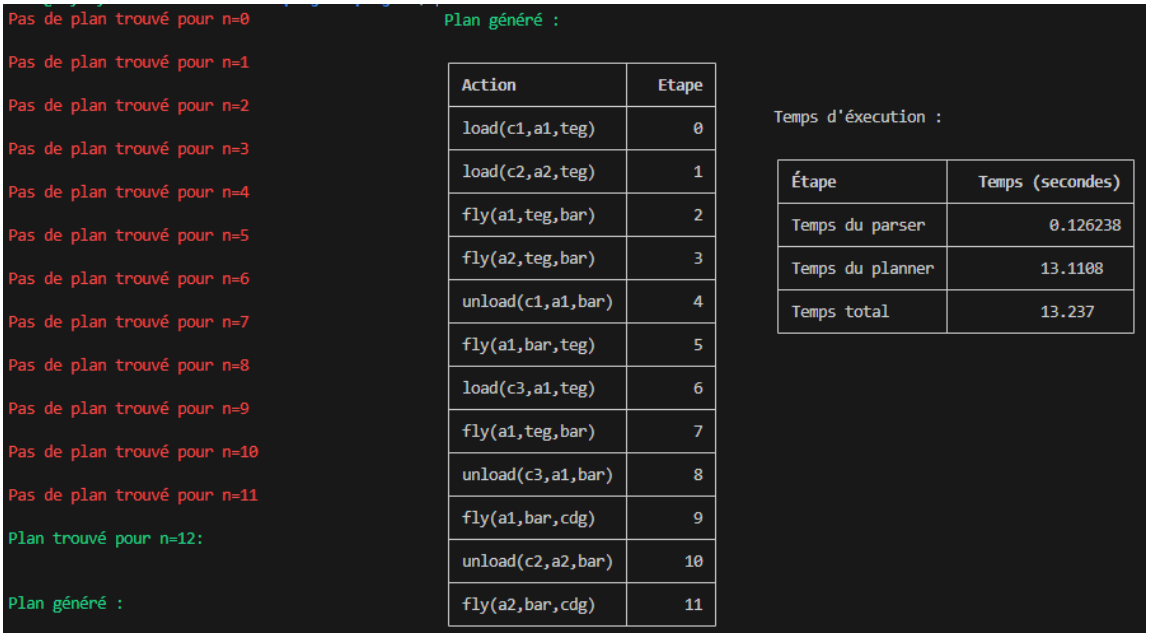
```
Pas de plan trouvé pour n=0

Pas de plan trouvé pour n=1

Pas de plan trouvé pour n=2

Pas de plan trouvé pour n=3

Plan trouvé pour n=4:


Plan généré :
```

Action	Etape
unstack(b,a)	0
stack(b,c)	1
pickup(a)	2
stack(a,b)	3

Temps d'éxecution :

Étape	Temps (secondes)
Temps du parser	0.10854
Temps du planner	0.0302105
Temps total	0.13875

— Example avec planes en lançant la commande :
python3 main.py ./pddl/avion-domain.pddl ./pddl/avion-problem.pddl ./asp/avion



4.3.2 Comparaison des temps d'exécution

Afin de comparer les performances de notre programme, nous avons comparé les temps d'exécution avec différents solveurs :

Pour le monde des blocs, notre programme a mis 0,10854 seconde pour générer le fichier ASP et 0,03 seconde pour trouver la solution. Pour le solveur SATPLAN, cela a pris 0,01 seconde. Enfin, nous avons aussi testé avec le solveur ENSHP qui nous a donné les résultats suivants :

plan-length : 4

metric (search) : 4,0

planning time : 298

heuristic time : 3

search time : 17

expanded nodes : 5

states evaluated : 6

fixed constraint violations during search (zero-crossing) : 0

number of dead-ends detected : 0

number of duplicates detected : 3

Nous avons supposé que les résultats suivants sont comptés en millisecondes (ms), donc cela a dû prendre 298 ms.

Pour le problème des avions, avec notre programme, nous avons obtenu 0,12 seconde pour générer le fichier ASP et 13,11 secondes pour générer la solution. Nous avons également utilisé le solveur ENSHP qui nous a donné les résultats suivants :

planning time : 373

heuristic time : 52

search time : 88

expanded nodes : 25

states evaluated : 145