Planification par ASP

Projet 02 d'Intelligence Artificielle et Manipulation Symbolique de l'Information

13 mars 2024

LUONG Ethan 28627904
PHAM Louis-Antoine 21304385

Introduction

L'objectif de ce second projet est de résoudre des problèmes PDDL avec ASP-STRIPS. Pour cela, on étudie dans un premier temps une étude de cas sur le monde des blocs. Après avoir écrit un convertisseur PDDL vers ASP-STRIPS en Python, nous écrirons finalement un script permettant de trouver un plan minimal à partir des fichiers PDDL.

1 Planificateur STRIPS

Cette étape consistait à traduire le domaine et le problème PDDL par des règles ASP. Dans la suite, le prédicat `holds(F,T)` assure que le fluent `F` est vrai à l'instant T, et le prédicat `perform(A,T)` indique que l'action `A` est réalisée à l'instant T. De plus, la constante n est le temps maximal pour atteindre l'objectif. Le fichier `blockworld.lp` est un exemple de traduction PDDL vers ASP, implémente le premier exercice du TD 5, et fournir un plan correct en n actions.

1.1 Domaine PDDL en ASP

Pour le domaine, nous avons déclaré chaque prédicat, chaque action, et avons associé à ces dernières leurs préconditions et effets. La prémisse permet de s'assurer du bon typage.

Pour traduire les préconditions, nous avons rajouté une contrainte d'intégrité empêchant une action d'être accomplie à l'instant T si ces préconditions n'étaient pas remplies à l'instant T.

```
:- perform(A,T), pre(A,C), not holds(C,T), time(T).
```

Dans le cas des effets positifs, il fallait simplement ajouter les fluents à l'instant $T+1$ après avoir réalisé l'action à l'instant T :

```
holds(F,T+1) :- perform(A,T), add(A,F).
```

Enfin, pour les effets négatifs, il s'agissait de propager les fluents vrais à l'instant T, à l'instant $T+1$ à condition que l'action réalisée à l'instant T ne les supprime pas :

```
holds(F,T+1) :- holds(F,T), perform(A,T), not del(A,F), time(T).
```

1.2 Problème PDDL en ASP

Concernant la partie problème de PDDL, il s'agissait de définir les objets utilisés, l'état initial et l'objectif à atteindre.

Les conditions initiales sont spécifiées par le prédicat `init` et correspondent au fait que les fluents concernés sont vrais à l'instant 0 :

```
holds(F,0) :- init(F).
```

Le but à atteindre se traduit par le prédicat `but`. Il faut que tous les fluents à l'intérieur de ce prédicat soient vrais au dernier instant n :

```
:- but(F), not holds(F,n).
```

La dernière étape était de ne pas oublier qu'une et une seule action devait être choisie à un instant donné (excepté à l'instant n durant lequel on n'accomplit aucune action):

```
1{perform(A,T): action(A)}1 :- time(T), time(T+1).
```

2 Convertisseur PDDL vers ASP-STRIPS

2.1 Généralités

Pour récupérer les prédicats, types et objets, nous avons utilisé le parser `pddl` conseillé dans l'énoncé. Autrement, nous avons seulement généralisé ce qui avait été fait dans la section 1.

A noter que comme le domaine définit les prédicats, types et actions, nous avons uniquement utilisé des variables (en majuscules) pour la traduction du domaine, contrairement au problème, qui définit l'état initial, le but et les objets, pour lequel nous avons utilisé des minuscules.

Pour tester notre classe `PDDLWriter`, nous avons utilisé deux jeux domaine / problème PDDL. Le premier est un exemple tiré de la documentation du parser `pddl`, tandis que le second correspond à l'exercice 1 du TD 5.

2.2 Cas particulier : prédicats négatifs

L'inconvénient majeur de notre code est qu'il ne gère pas les préconditions négatives `pre(A, not F)`. Nous avons pensé dans un premier temps à ajouter des prédicats négatifs : `pred(not F)`, cependant cela posait également des problèmes pour les effets positifs qu'il aurait été difficiles de distinguer des effets négatifs.

3 Combinaison

Dans cette dernière partie, nous avons simplement écrit une fonction `find_minimal_plan` qui itérait sur tous les plans possibles en $1, ..., n_{max}$ instants. Par défaut, n_{max} est fixé à 1000.

Les résultats de chaque exécution de clingo sont enregistrés dans le dossier `outputs`, chaque sous-dossier correspondant à une exécution reconnaissable à son timestamp écrit dans le nom du sous-dossier.

Pour tester le code, nous avons à nouveau utilisé le premier exercice du TD 5 en exécutant la ligne de code suivante :

```
python3 aspplan.py ./pddl_examples:blockWorld-domain.pddl ./pddl_examples:blockWorld-
    problem.pddl
```

Nous avons ainsi obtenu un plan en 4 étapes :

1. unstack(b,a)
2. stack(b,c)
3. pickup(a)
4. stack(a,b)

NB : il peut être nécessaire de changer le répertoire dans lequel se situe `clingo` dans la fonction `find_minimal_plan` à la ligne suivante :

```
os.system(f"clingo {asp_filename} > {results_filename}")
```

Conclusion

Pour résumer, au cours de ce projet, nous avons tout d'abord traduit un cas particulier de problème PDDL en ASP-STRIPS. En utilisant le parser `pddl`, nous avons généralisé cette approche à n'importe quel ensemble de fichiers PDDL ne contenant pas de prédicats négatifs. Cela nous aura finalement permis d'établir un plan minimal pour un problème PDDL donné.