Ecole des Mines de Saint-Etienne
Cycle I.C.M. / Toolbox IA / Planning
O. Boissier (boissier@emse.fr)

Planning

Documents containing the answers to QUESTIONS should be returned in an archive named with your family name, by November 14th, 2016 by email to boissier@emse.fr

1. Initialization and configuration

- Download the file: *lab-planning-toolbox-ia.zip available on the Toolbox Web side*
- Unzip it
- You should have the following structure:
  - *TP_26octobre2016_ToolboxIA.pdf // this document*
  - *javagp // graphplan planner used in section 4*
  - *JPlan // graphplan planner used in section 3*

2. Modeling Plans in PDDL

Understand PDDL

This short introduction comes from: http://users.cecs.anu.edu.au/~patrik/pddlman/writing.html

A PDDL definition consists of two parts: The *domain* and the *problem* definition.
Note: Although not required by the PDDL standard, most planners require that the two parts are in separate files.
Comments: comments in a PDDL file start with a semicolon (";") and last to the end of the line.
Requirements
Because PDDL is a very general language and most planners support only a subset, domains may declare requirements. The most commonly used requirements are:

:**strips** The most basic subset of PDDL, consisting of STRIPS only.
:**equality** This requirement means that the domain uses the predicate =, interpreted as equality.
:**typing** This requirement means that the domain uses types (see **Typing** below).
:**adl** Means that the domain uses some or all of ADL (*i.e.* disjunctions and quantifiers in preconditions and goals, quantified and conditional effects).

The Domain Definition: The domain definition contains the domain predicates and operators (called *actions* in PDDL). It may also contain types (see **Typing**, below), constants, static facts and many other things, but, again, the majority of planners don't support them.

The format of a (simple) domain definition is:

```
(define (domain DOMAIN_NAME)   (:requirements [:strips] [:equality] [:typing]
[:adl])   (:predicates (PREDICATE_1_NAME ?A1 ?A2 ... ?AN)
(PREDICATE_2_NAME ?A1 ?A2 ... ?AN)           ...)
(:action ACTION_1_NAME
        [:parameters (?P1 ?P2 ... ?PN)]
        [:precondition PRECOND_FORMULA]
        [:effect EFFECT_FORMULA]
)
(:action ACTION_2_NAME     ...)     ...)
```

Elements in []'s are optional, for those not familiar with formal grammars.

Names (domain, predicate, action, *et c.*) are usually made up of alphanumeric characters, hyphens ("-") and underscores ("_"), but there may be some planners that allow less.

Parameters of predicates and actions are distinguished by their beginning with a question mark ("?").

The parameters used in predicate declarations (the :predicates part) have no other function than to specify the number of arguments that the predicate should have, *i.e.* the parameter names do not matter (as long as they are distinct). Predicates can have zero parameters (but in this case, the predicate name still has to be written within parenthesizes).

The Problem Definition: The problem definition contains the objects present in the problem instance, the initial state description and the goal.
The format of a (simple) problem definition is:

```
(define (problem PROBLEM_NAME)
        (:domain DOMAIN_NAME)
        (:objects OBJ1 OBJ2 ... OBJ_N)
        (:init ATOM1 ATOM2 ... ATOM_N)
        (:goal CONDITION_FORMULA)
)
```

Practice

1. Open the "*javagp/examples/pddl"* directory
2. Open the domain definition file "*gripper.pddl*" as well as the problem definition file "pb1.pddl"

QUESTION

   a. **Draw or describe the initial state as well as the final state of the problem.**
   b. **What are the possible actions?**

3. You can explore the other examples that are provided in this directory.

3. Understanding GraphPlan Algorithm

JPlan is an implementation of graphplan that generates both a plan and the complete trace of the planning.
However JPlan doesn't use the PDDL Language.
This section aims at using and understanding the way graphplan works. In the next section we will use another implementation of graphplan using domain and problem defined in PDDL.

How to launch JPlan
JPlan uses two files for generating a plan of N levels: a file describing the domain (operators_file) and a file describing the problem (facts_file).
In order to launch JPlan, you should:

- use the following command in the JPlan directory

  java –classpath ./lib/jplan.jar JPlan operators_file facts_file [max_level]

  - operators_file: text file containing the STRIPS-like definitions of operators
  - facts_file: is a text file containing the planning problem (set of objects, initial state, and goal state).
  - max_level: maximum number of planning graph level (default is 10)

If a valid plan has been generated then you should see it in the file "output.pln".
The created planning graph information is written in the file "output.txt".

Practice

1. Open the "*JPlan*" directory. We will use JPlan on the BlocksWorld.

2. Open blocksworld.txt and pb1.txt.

QUESTION

   a. **What are they representing?**
   b. **What is the meaning of each of the three lines between [,] in blocksworld.txt?**

3. Execute JPlan on these descriptions with the default depth.

QUESTION

   a. **Is the plan produced in output.pln correct?**
   b. **What is the depth of this plan?**

4. Open and interpret the graphplan trace presented in “output.gp”:

QUESTION

   a. **What is the “proposition” layer?**
   b. **What is the “action” layer?**
   c. **What are the Mutex expressions that are written?**
   d. **What does mean the noop term that appears in some expressions? Why do we need it?**
   e. **Is the number of Action Layers consistent with the depth of the plan? Is it always the case?**
   f. **Why do we have only the operator “Pickup” in the first Action Layer?**
   g. **Why do we have a Mutex relation between “noopClear(a1)” and “Pickup(a1)” in the first Action Layer? Why the other Mutex?**
   h. **If necessary to help you understand the whole process, you can run several times the planning algorithm using different depths.**

As soon as you have understood the way graphplan works, you can test and experiment it on another example.

4. Modeling Domains and Problems, Planning with graphplan

In this part, you will use javagp, which is a more efficient implemented version of GraphPlan. Moreover, it uses different heuristics and you can use it on domains and problems written in PDDL compliant form.
https://github.com/pucrs-automated-planning/javagp

4.1. Using the GraphPlan Algorithm

How to use JavaGP on a STRIPS Language:

- Run in the javagp directory: "java -jar javagp.jar -nopddl -d examples/strips/ma-prodcell/domain.txt -p examples/strips/ma-prodcell/problem.txt"

How to use JavaGP on a PDDL Language

- Run in the javagp directory "java -jar javagp.jar -d examples/pddl/blocksworld/blockworlds.pddl -p examples/pddl/blocksworld/pb1.pddl"

4.2. Modeling 1D Rubik’s Cube

QUESTION

Model and represent in PDDL the domain and problem of 1D Rubik’s Cube from the description given below.

1D Rubik's Cube is a line of 6 numbers with original position: 1 2 3 4 5 6 which can be rotated in

3 different ways in groups of four:

operator 0 $\overline{(1\ 2\ 3\ 4)}\ 5\ 6 \rightarrow \overline{(4\ 3\ 2\ 1)}\ 5\ 6$

operator 1 $1\ \overline{(2\ 3\ 4\ 5)}\ 6 \rightarrow 1\ \overline{(5\ 4\ 3\ 2)}\ 6$

operator 2 $1\ 2\ \overline{(3\ 4\ 5\ 6)} \rightarrow 1\ 2\ \overline{(6\ 5\ 4\ 3)}$

Given a scrambled line, return the shortest sequence of rotations to restore the original position.

Examples:
solve 1 3 2 6 5 4
result is: 1 2 1
solve 5 6 2 1 4 3
result is: 0 2
solve 6 5 4 1 2 3
result is: 0 1 2

QUESTION

Test and validate your domain and problems representation by launching the javagp graphplan planner on them. What is the resulting plan?

4.2. Modeling the busy and organized student (optional)

QUESTION

Model and represent in PDDL the domain and problem corresponding to the description given below.

You're a busy student. You need to complete a project for a class. You would also like to check out and read a novel that was just heartily recommended to you. Being a good student, you prioritize your tasks wisely and read only after you completed the homework. Therefore, assume the following STRIPS operators are defined:

Action: CheckOut(b,t) "Check out book b from the library at time t"
Preconditions: Now(t), TimeAfter(t,n), LibraryOpen(t)
Add: HaveBook(b), Now(n)
Delete: Now(t)

Action: Read(b, t) "Read book b at time t"
Preconditions: HaveBook(b), Now(t), TimeAfter(t,n), HomeworkDone(c)
Add: BookRead(b), Now(n)
Delete: Now(t)

Action: DoHomework(c,t) "Do homework for class c at time t"
Preconditions: Now(t), TimeAfter(t,n), Assigned(c)
Add: HomeworkDone(c), Now(n)
Delete: Now(t)

Where the state predicates are interpreted as follows:
Now(t) "The time is now t"
TimeAfter(t,n) "The next time after t is n"
LibraryOpen(t) "The library is open at time t"
HaveBook(b) "You have book b"

Assigned(c) "Homework was assigned for class c"
HomeworkDone(c) "Homework for class c is done"
BookRead(b) "Book b was read"

Consider the initial state:
Now(7:30PM)
TimeAfter(7:30PM, 8:30PM)
TimeAfter(8:30PM, 9:30PM)
TimeAfter(9:30PM, 10:30PM)
LibraryOpen(7:30PM)
Assigned("CS101")

And the conjunctive goal (given in this order):
BookRead("One Hundred Years of Solitude") & HomeworkDone("CS101")

Solve it using javagp