Massachusetts Institute of Technology

16.410 Principles of Automated Reasoning and Decision Making

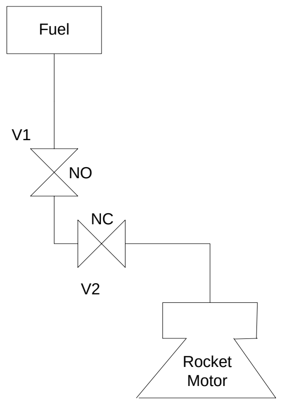
Problem Set #5

Problem 1: Planning for Space Operations (30 points)

In this problem you will construct a plan graph for a very simple spacecraft control problem.

A critical stage of many deep space probe missions is orbital insertion. One of the most ambitious simulation-based autonomy demonstrations to date has been robust mission planning, execution and failure recovery for Saturn Orbital Insertion (**SOI**). During SOI it is essential that the main engine be commanded reliably under failure. In this problem we consider the problem of automatically planning a control sequence for a simple rocket engine system. We address execution and failure recovery later in this course.

Consider an extremely simple rocket engine system, shown at the top of the next page. To fire the engine, fuel must flow to it. Fuel flow is controlled by valves V1 and V2, which are pyrotechnic valves, pyro for short. Pyro valves are initially in one particular state (open or closed). An explosive bolt can be fired that switches a pyro valve to its other state. Thus, an important disadvantage of a pyro valve (with respect to a typical, electrically activated on-off valve) is that a pyro valve can switch states only once. The advantage of using a pyro valve is that it is extremely reliable; it will stay in its initial state until fired. When the valve is fired, it will switch with high probability to its opposite state, where the valve remains. In the following diagram, valve V1 is initially open (indicated by **NO** for normally open). Firing V1 closes it. Valve V2 is initially closed (**NC** for normally closed). Firing V2 opens it.



We formulate the problem using the STRIPS plan representation (the STRIPS representation is introduced in the lecture notes and Ch. 11 of AIMA). Our problem is to generate a command sequence that fires the rocket engine, given that initially v2 is closed, v1 is open and the rocket is off. The initial and goal conditions in STRIPS are:

Initial Conditions:
```
      (preconds
        (closed-v2)
        (open-v1)
        (off-rocket))
```
Goal:
```
      (effects
        (rocket-fired)
        (fuel-flowing))
```

We define the operations of firing pyro valves v1 and v2 through the plan operators **fire-v1** and **fire-v2**, given below. Note that fire-v1 moves v1 from open to closed, and can only be executed when v2 is open. Likewise, fire-v2 moves v2 from closed to open, and can only be executed when v1 is open:

```
      (OPERATOR fire-v1
        (params)
        (preconds (open-v2)
                  (open-v1))
        (effects  (del open-v1)
                  (closed-v1)
                  (del fuel-flowing)
                  (fuel-not-flowing)))

      (OPERATOR fire-v2
        (params)
        (preconds (closed-v2)
                  (open-v1))
        (effects  (del closed-v2)
                  (open-v2)
                  (del fuel-not-flowing)
                  (fuel-flowing)))
```

Note that “del” in the effect of an action is short for “delete”. This is equivalent to “not”, e.g. (del closed-v2) is equivalent to (not closed-v2).

In addition, we introduce operators for firing and shutting down the rocket, **`fire-rocket`** and **`shut-off-rocket`**, respectively. The engine can only be fired if it is off and fuel is flowing. The engine can only be shut off if the rocket is on, but fuel is not flowing:

```
(OPERATOR fire-rocket
  (params)
  (preconds (fuel-flowing)
            (off-rocket))
  (effects  (del off-rocket)
            (on-rocket)
            (rocket-fired)))

(OPERATOR shut-off-rocket
  (params)
  (preconds (on-rocket)
            (fuel-not-flowing))
  (effects  (del on-rocket)
            (off-rocket)))
```

Part A. Draw the plan graph for this problem, beginning with the initial state, and expanding levels until a level appears that contains all goal variables. Ignore mutex relations for now.

Part B. Draw a plan graph for this problem that includes mutex relations for both actions and variables. Note that this may require adding layers to the graph, relative to Part A. The last level for this graph must include all goal variables with no mutex relations between any of them.

Problem 2: Proving termination (40 points)

In this problem you will prove that GRAPHPLAN plan graph generation in fact terminates. That is, the graph reaches a fixed point after a finite number of levels.

Recall that a plan graph is said to have reached a ***fixed point*** when two consequent layers are the same. When the graph reaches a fixed point, extending the graph with more layers is unnecessary since the new layer will be same as the last layer.

A property in a graph is said to **monotonically increase/decrease** at each layer, if the property at layer n+1 is a superset/subset of the same property at layer n, for all layers n. Prove the following theorem:

Theorem: The planning graph reaches a fixed point after finitely many iterations.

To prove this theorem, state and prove three lemmas. The first lemma should state that the number literals in a layer increases monotonically at each layer. The second lemma should state that the number of actions in a layer increases monotonically at each layer. Finally, the third lemma should state that the mutexes decrease monotonically. You should prove the first two lemmas. You should only sketch the proof of the third lemma (i.e., informally explain how the proof should be done). From these three lemmas you should prove the theorem.

Problem 3: Fruitcake problem (30 points)

In this problem you will experiment with the graphplan software to understand the complexity of problems it can handle.

Graphplan software is provided to you for this exercise.
The software is also available at:
http://www.cs.cmu.edu/~avrim/graphplan.html

Consider the problem of a robot trying to stack a lettered toy blocks to form a word. At each step, if the robot's gripper is empty, the robot may pick up a block provided that the block does not have a block on top of it. If the robot's gripper holds a block, it may place the block on top of any block that doesn't already have a block on top of it, or it may place the block on the table. In this problem, you will experimentally explore and then discuss the complexity of this problem.

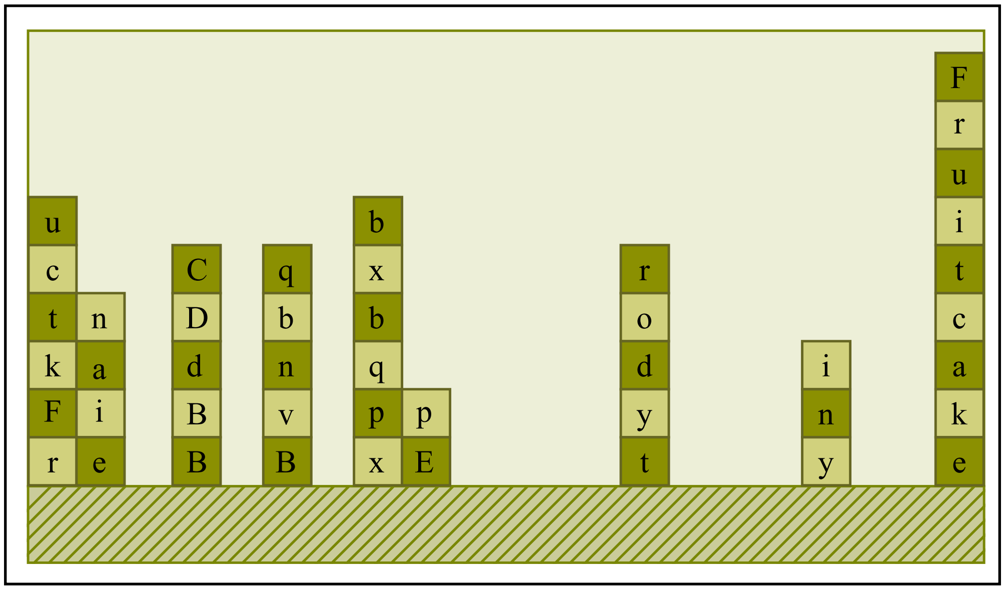


Image by MIT OpenCourseWare.

Formulate this problem in PDDL and solve using the Graphplan software. Start with simple conditions, a three-letter word with only three blocks all initially on the table. Experiment with the software and be sure to include the following changes.

- Add additional blocks to the table.
- Move the blocks so that some are initially on top of some others.
- Try to stack longer words.
- Give the robot additional grippers.

Present and discuss the result of your experiments.

Please indicate the time you have spent for each problem.