16.410-13 Recitation 5 Problems

Problem 1: PDDL example

We car that has a flat tire. We are in the following situation.

- *The car has its* flat *tire on the* axle*.*
- *There is a* spare *tire in the* trunk*.*
- *The driver can* remove *any* tire *from any* place *(i.e., either axle or trunk) and put it on the* ground*.*
- *The driver can* put *any* tire *that is on the* ground *to any* place *that is* empty*.*

Formulate this problem in PDDL and draw the corresponding plan graph.

Solution: First let us define two predicates, namely `AT` and `Empty`. The predicate `AT` takes two parameters denoted by `x` and `y`, where `x` is a tire type (e.g., `flat` or `spare`) and `y` is a place (e.g., `trunk` or `axle`); if `AT x y` is true, then `x` is *at* `y`. For instance, if `x` = `flat` and `y` = `trunk`, then `AT x y` is true whenever the flat tire is inside the trunk. The predicate `Empty` takes one parameter denoted by `x`, which in this case is a place such as the `axle` or the `trunk`. If `Empty x` is true, then `x` is empty. For instance, if `x` = `axle`, then `Empty x` indicates that the axle is empty, i.e., there is no tire on the axle. In PDDL, these predicates can be written as follows:

```
(:predicates (AT ?x ?y)
             (EMPTY ?x))
```

Next, let us consider the actions available to us in fixing our car. Let us define two actions, namely `Remove` and `Put`. `Remove` action takes three parameters: `tire`, `place`, and `ground`. The action is tailored to remove the `tire` from its `place` and put the `tire` on the `ground`. Hence, a precondition is that the `tire` must be at the `place`. The effect, on the other hand, is threefold: (i) the `tire` is not at the `place` anymore, (ii) the `tire` is on the `ground`, and (iii) the `place` becomes empty (does not have a tire on it). This action can be represented in PDDL as follows:

```
(:action Remove :parameters (?tire ?place ?ground)
                :precondition (AT ?tire ?place))
                :effect       (and (not (AT ?tire ?place))
                                   (AT ?tire ?ground)
                                   (EMPTY ?place))
```

The `Put` action also takes the same three parameters, namely `tire`, `place`, and `ground`. This time, however, the action will take the `tire` off of the `ground`, and put it on the `place`. Note that the preconditions that need to be satisfied are: (i) the `tire` has to be on the `ground` and (ii) `place` has to be empty. The effects are again threefold: (i) the `tire` is not on the `ground` anymore, (ii) the `tire` is at the `place`, and (iii) the `place` is not empty. Following is a PDDL description of this action.

```
(:action Put :parameters (?tire ?place ?ground)
             :precondition (and (AT ?tire ?ground))
                                (EMPTY ?place)
             :effect      (and (not (AT ?tire ?ground))
                                (AT ?tire ?place)
                                (not (EMPTY ?place)))
```

In this problem, we have five objects in total. Two of the objects are the tires, namely `flat` and `spare`. Two others are the places, namely `axle` and `trunk`. The final object is `ground`. The PDDL description of this will be:

```
(:objects flat space axle trunk ground)
```

The initial condition is that the `flat` tire is on the `axle`, whereas the `spare` tire is in the `trunk`. The PDDL form of this is:

```
(:init  (AT flat axle)
        (AT spare trunk))
```

The goal is to have the `spare` tire on the `axle`.

```
(:goal  (AT spare axle)
```

Problem 2: More PDDL

Suppose you have a robot that moves in a house with several rooms and can pickup balls and put them down. More precisely, the robot has three actions: `Navigate` *from one room to another,* `Pickup` *a certain ball from a certain room, and* `Putdown` *a certain ball to a certain room. Your robot can carry several balls all at once. Model this problem using PDDL. Write down the* predicates *and the* actions*.*

Assume that the house has a `bedroom` *and a* `kitchen`*. Assume also that there is only one ball called the* `blueball`*. Initially, the* `blueball` *is in the* `bedroom`*. The robot starts in the* `kitchen`*. The goal is to take the* `blueball` *to the* `kitchen`*. Write down your* objects*,* initial condition*, and* goal condition *in PDDL.*

Solution: The corresponding PDDL specification is given below. Note that there are three predicates. The predicate `IN` takes two parameters `?ball` and `?room`. The predicate `IN(?ball, ?room)` is true if the ball indicated by `?ball` is in room indicated by `?room`. The predicate `AT` takes on parameter, `?room`. If `AT(?room)` is true, then the robot is *at* room indicated by `?room`. Finally, the predicate `HAS` takes one parameter, `?ball`. If `HAS(?ball)` is true, then the robot *has* the ball indicated by `?ball`.

Three actions are defined. The `Navigate` action takes two parameters: `?roomfrom` and `?roomto`. By executing `Navigate(?roomfrom, ?roomto)` the robot moves from the room indicated by `?roomfrom` to the room indicated by `?roomto`. Notice that the precondition ensures that the robot is currently *at* `?roomfrom` and not `?roomto`. The `Pickup` action takes two parameters, namely `?ball` and `?room`. By executing the action `Pickup(?ball, ?room)`, the robot picks up the `?ball` in `?room`. Of course, the robot has to be *at* the `?room` to pickup the `?ball`. Moreover, `?ball` has to be *in* the `?room`. The action `Putdown` reverses the action `Pickup`. It takes the same parameters, but by executing `Putdown (?ball, ?room)`, the robot puts the `?ball` into the `?room`. Again, the robot has to be *at* the `?room` to execute this action and *have* the `ball`.

```
(:predicates (IN ?ball ?room)
             (AT ?room)
             (HAS ?ball)

(:action Navigate :parameters (?roomfrom ?roomto)
                  :precondition (and (AT ?roomfrom)
                                     (not (AT ?roomto)))
                  :effect       (and (not (AT ?roomfrom))
                                     (AT ?roomto)))

(:action Pickup :parameters (?ball ?room)
                :precondition (and (AT ?room)
                                   (not (HAS ?ball))
                                   (IN ?ball ?room))
                :effect       (and (HAS ?ball)
                                   (not (IN ?ball ?room))))

(:action Putdown :parameters (?ball ?room)
                 :precondition (and (AT ?room)
                                    (HAS ?ball))
                 :effect       (and (not (HAS ?ball))
                                    (IN ?ball ?room)))
```

Extra exercise: You must have noticed that some of the conditions on the preconditions and effects include some redundancies. Can you point those out? That is, give a PDDL description of the actions with fewer clauses.

The list of objects along with the initial conditions and goal specification are given below.

```
(:objects blueball kitchen bedroom)

(:init (IN blueball bedroom)
       (AT kitchen))
(:goal (IN ?blueball ?kitchen)
```

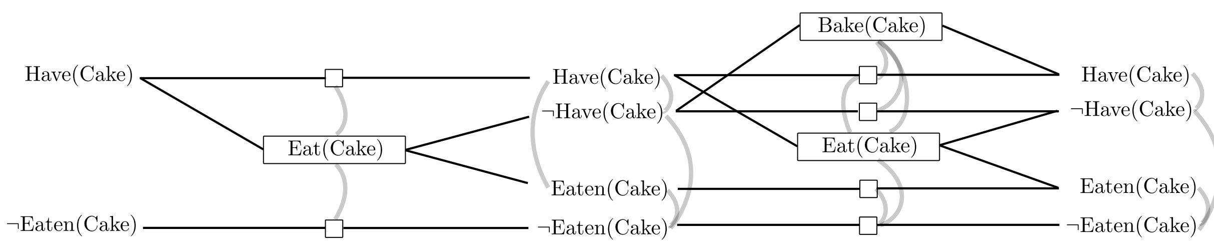
Extra exercise: Try to add more balls and rooms into the house. What if the house had a certain *topology*. That is only certain rooms are connected to one another directly. To navigate from one room to another you need to go through a different room. For example, imagine having an entrance which is connected to a hallway. Then the hallway is connected to three bedrooms. Inside one of these bedrooms there is a bathroom. So, if you would like to go from entrance to the bathroom. You need to be in four different rooms sequentially, i.e., (i) start with the entrance, (ii) go through the hallway, (iii) go through the bedroom with the bathroom, and (iv) go inside the bathroom. But there is no direct passage from the entrance to the bathroom. And assume that your robot can pick up or put down balls along the way. For instance, you can pick a ball up from the hallway as you are going through the hallway. How would you model such a problem in PDDL? What would be difference?

Problem 3: Planning graphs – baking the cake

Consider the following PDLL specification. Draw the corresponding plangraph until it levels off, i.e., reaches a fixed point. Indicate the mutexes.

```
(:predicates (AT ?ball ?room)
             (IN ?room)

(:action Eat :parameters (?cake)
             :precondition (HAVE ?cake)
             :effect       (and (not (HAVE ?cake))
                                (EATEN ?cake)))

(:action Bake :parameters (?cake)
              :precondition (not (HAVE ?cake))
              :effect       (HAVE ?cake))

(:objects cake)

(:init (HAVE cake))
(:goal (and (HAVE ?cake)
           (EATEN ?cake)))
```

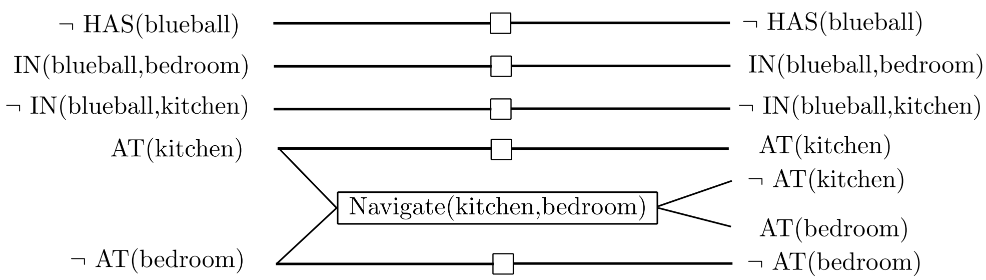
Solution: The resulting plan graph is shown in the figure below.

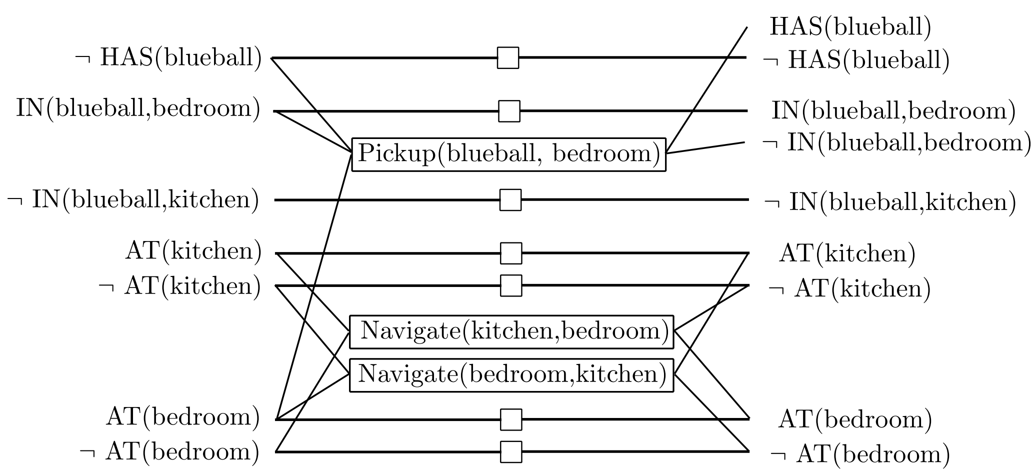


Problem 4: More planning graphs – Robot navigation

Recall the PDDL specification you had worked with in Problem 2. Draw the first two levels of the corresponding plan graph. Explain the execution of the GRAPHPLAN algorithm on the first two layers.

Solution: First two layers of the corresponding planning graph is shown below.





MIT OpenCourseWare
http://ocw.mit.edu

16.410 / 16.413 Principles of Autonomy and Decision Making
Fall 2010

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.