Comp3620/Comp6320 Artificial Intelligence
Tutorial 5: Planning Representations and Graph-Based Approaches

May 7-11, 2018

Exercise 1 (STRIPS problem formulation)

This is part of exercise 10.3 from the book. The monkey-and-bananas domain features a monkey in a laboratory with some bananas hanging out of reach from the ceiling. A box is available that will enable the monkey to reach the bananas by pushing the box underneath them and climbing onto it. Initially, the monkey is at location A, the bananas are at location B, and the box is at location C. When the monkey climbs onto the box, the monkey's height changes from *floor* to *ceiling*. The actions available are *Go* from one location to another, *Push* a pushable object from a location to another, *ClimbUp* onto or *ClimbDown* from a climbable object, and *Grasping* a graspable object. Grasping results in having the object if the monkey and object are at the same place and same height. Suppose that the goal of the monkey is to fool the scientists by getting the bananas but returning the box to its original place and then going back to its original location. Recall that STRIPS does not have negative preconditions nor negative goals.

Questions:

1. **Describe the problem using the STRIPS representation (operators, initial state, goal).**
2. **Give the shortest sequential plan for this problem.**
3. **What is the STRIPS rule? Apply it to determine the description of the state resulting from applying the first action of the plan in the initial state.**

Question 1:

Note that below we're omitting brakets {} for clarity, but technically, preconditions, effects, state and goal descriptions are sets.

- Predicates:
  - climbable(o): o can be climbed on
  - pushable(o): o can be pushed
  - have(o): the monkey has o
  - graspable(o): o can be grasped
  - at(o,l,h): o is at location l and at height h
- Initial state s_0:
  {climbable(box), pushable(box), graspable(bananas), at(monkey,A,floor), at(bananas,B,ceiling), at(box,C,floor)}
- Goal g:
  {have(bananas), at(box,C,floor), at(monkey,A,floor)}

- Operators:
  - go(l_1,l_2)
    - PRE: {at(monkey,l_1,floor)}
    - EFF$^-$: {at(monkey,l_1,floor)}
    - EFF$^+$: {at(monkey,l_2,floor)}

    Note, we can also write EFF: $\neg$ at(monkey,l_1,floor), at(monkey,l_2,floor). We can also write add-list and del-list instead of EFF$^+$ and EFF$^-$
  - push(o,l_1,l_2)
    - PRE: {pushable(o), at(monkey,l_1,floor), at(o,l_1,floor)}
    - EFF$^-$: {at(monkey,l_1,floor), at(o,l_1,floor)}
    - EFF$^+$: {at(monkey,l_2,floor), at(o,l_2,floor)}
  - climbUp(o,l)
    - PRE: {climbable(o), at(o,l,floor), at(monkey,l,floor)}
    - EFF$^-$: {at(monkey,l,floor)}
    - EFF$^+$: {at(monkey,l,ceiling)}
  - climbDown(o,l)
    - PRE: {climbable(o), at(o,l,floor), at(monkey,l,ceiling)}
    - EFF$^-$: {at(monkey,l,ceiling)}
    - EFF$^+$: {at(monkey,l,floor)}
  - grasp(o,l,h)
    - PRE: {graspable(o), at(o,l,h), at(monkey,l,h)}
    - EFF$^-$: {at(o,l,h)}
    - EFF$^+$: {have(o)}
- Shortest plan:
  ⟨go(A,C), push(box,C,B), climbUp(box,B), grasp(bananas,B,ceiling), climbDown(box,B), push(box,B,C), go(B,A)⟩

Question 2:

The strips rule for a state s and an action $a = \langle \text{PRE}(a), \text{EFF}^-(a) : \text{EFF}^+(a) \rangle$ is

$$s' = \gamma(s,a) = \begin{cases} (s \setminus \text{EFF}^-(a)) \cup \text{EFF}^+(a) & \text{if } \text{PRE}(a) \subseteq s \\ \text{undefined otherwise} & \text{(action not executable)} \end{cases}$$

We want to compute the result s' of applying go(A,C) in s_0:

- the precondition is verified: {at(monkey,A,floor)} $\subseteq s_0$ hence the action is applicable
- the resulting state is:
  {climbable(box), pushable(box), graspable(bananas), at(monkey,C,floor), at(bananas,B,ceiling), at(box,C,floor)}

Exercise 2 (ADL to STRIPS)

Consider the following action description, written in the ADL fragment of PDDL.

```
(:action A
:precondition (p)
:effect (and (not (p))
             (when (q) (r))
             (q))
)
```

Questions:

1. **Write down a set of equivalent plain STRIPS actions (i.e., without conditional effects, and without disjunction or negation in the preconditions)**
2. **Describe the changes that would need to be made to other actions in the domain description, the initial state and/or the goal in order to make the domain fall only within the STRIPS language**

Question 1:

We need to get rid of the conditional effect `(when (q) (r))`. This can be achieved by creating two actions, one in which the effect condition is true and one in which it is false. Note that in the first action, the effect `(q)` is not needed since the precondition guarantees that `(q)` it was already true before the action is applied.

```
(:action A1
:precondition (and (p) (q))
:effect (and (not (p))
             (r))
)

(:action A2
:precondition (and (p) (not (q)))
:effect (and (not (p))
             (q))
)
```

Unfortunately, this still isn't in the STRIPS formalism, which has no negative precondition. We need to get rid of the negation `(not (q))`. We can achieve this by creating a new proposition `(not-q)` in addition to `(q)` (note that the name of the new proposition is "not-q"), and ensuring that every time we add `(q)`, we delete `(not-q)`, and vice versa. This leads to the second action becoming:

```
(:action A2
:precondition (and (p) (not-q))
:effect (and (not (p))
             (q)
             (not (not-q)))
)
```

Observe that the negative precondition `(not (q))` has been replaced with `(not-q)`, which needs to be deleted in the effect.

Question 2:

Since we have introduced a new proposition in the domain we need to ensure that:

- the initial state description contains either `(q)` or `(not (q))`, since one of them will need to be true given the closed-world assumption
- any other action adding `(q)` also deletes `(not (q))`
- any other action deleting `(q)` also adds `(not (q))`

This assumes of course that the other actions were already STRIPS compliant, otherwise they need to be transformed in the same way as `A`. If the goal wasn't STRIPS compliant and mentioned `(not q)`, then this needs to be transformed into `(not-q)`

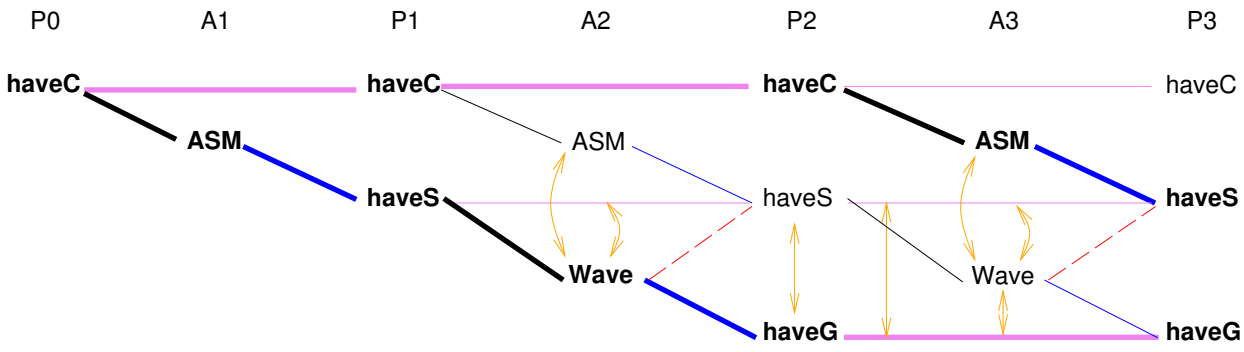
Exercise 3 (Graph-Based Planning)

Consider the following planning problem. You want a sheep and a goat. Using your credit card, you can buy a sheep from the Automatic Sheep Machine (ASM). Waving your magic wand turns a sheep into a goat. The planning operators are as follows, where a box represents an operator whose pre-conditions are on the left-hand side and effects are on the right-hand side of the box.

Show how graphplan would solve the problem:

1. **Draw the planning graph until the first level leading to a plan. Include all mutex relations.**
2. **State at which levels Graphplan would attempt extraction.**
3. **Highlight in the graph the plan extracted by Graphplan (include the maintenance actions chosen).**
4. **Given the built planning graph structure, provide at least two heuristics that can be computed from the graph and then used in an informed state space heuristic search framework (e.g., A*). Report the estimated distance to the goal in the initial state for each heuristic.**

1. The planning graph is shown below. The first level leading to a plan is level 3.



The mutexes are as follows:

- at level 2, $Wave$ deletes $haveS$ and is mutex with both ASM (inconsistence) and the maintenance action for $haveS$ (interference + inconsistence) which have $haveS$ as precondition (and effect for the maintenance action).
- at level 2, $haveS$ is mutex with $haveG$ (inconsistent support caused by the action mutexes)
- at level 3, $Wave$ raemains mutex with ASM and the maintenance action for $haveS$ for the same reasons (interference and inconsistence are static properties so these mutexes cannot disappear).
- at level 3, the maintenance action for $haveG$ is mutex with both $Wave$ and the maintenance action for $haveS$ because of competing needs (due to the mutex of $haveS$ and $haveG$ at level 2).
- at level 3, the mutex between $haveS$ and $haveG$ has disappeared because $haveG$ can now be produced by its maintenance action, which is not mutex with ASM.

2. Extraction would only be attempted at level 3 because the goal ($\{haveS, haveG\}$) is mutex at level 2.

3. The actions, propositions and links that are part of the final plan are in bold face.

4. A first heuristic can be computed by taking the index of the smallest level in which all the goal propositions have all appeared (h_1). Another, more informed heuristic is the index of the first level in which all the goal propositions appear and are not mutex (h_2). Therefore $h_1(s_0, G) = 2$ and $h_2(s_0, G) = 3$.