15-887 Solutions Homework 1

Lunar Lockout Game

a) - d)

Many valid solutions to this problem. We were looking for clear idea about the predicates that a state could be composed of. We also wanted to see recognition that actions have precondition, add, and delete effects. Domain axioms should be stated but are implicit in the definition of actions.

Below is a possible solution, represented using PDDL. Adapted from the answer by Devin Schwab.

Listing 1: Domain specification

```
(define (domain lunar-lockout-b)
  (:requirements :strips :adl :typing)

  (:types spacecraft coord)

  (:predicates
    (greater-than ?s1 - coord ?s2 - coord)
    (one-greater-than ?s1 - coord ?s2 - coord)
    (at ?spacecraft - spacecraft ?row - coord ?col - coord))

  (:action move-left
    :parameters (?spacecraft1 - spacecraft ?row - coord
                 ?spacecraft2 - spacecraft ?s2-col - coord
                 ?from-col - coord ?to-col - coord)
    :precondition (and
                    (at ?spacecraft1 ?row ?from-col)
                    (at ?spacecraft2 ?row ?s2-col)
                    (one-greater-than ?to-col ?s2-col)
                    (greater-than ?from-col ?to-col)
                    (forall (?spacecraft3 - spacecraft ?col - coord)
                      (not (and
                        (at ?spacecraft3 ?row ?col)
                        (greater-than ?from-col ?col)
                        (greater-than ?col ?s2-col)))))
    :effect (and (at ?spacecraft1 ?row ?to-col)
              (not (at ?spacecraft1 ?row ?from-col))))

  (:action move-right
    :parameters (?spacecraft1 - spacecraft ?row - coord
                 ?spacecraft2 - spacecraft ?s2-col - coord
                 ?from-col - coord ?to-col - coord)
    :precondition (and
                  (at ?spacecraft1 ?row ?from-col)
                  (at ?spacecraft2 ?row ?s2-col)
                  (one-greater-than ?s2-col ?to-col)
                  (greater-than ?to-col ?from-col)
                  (forall (?spacecraft3 - spacecraft ?col - coord)
                    (not (and
                      (at ?spacecraft3 ?row ?col)
                      (greater-than ?col ?from-col)
                      (greater-than ?s2-col ?col)))))
  :effect (and (at ?spacecraft1 ?row ?to-col)
            (not (at ?spacecraft1 ?row ?from-col))))


(:action move-up
  :parameters (?spacecraft1 - spacecraft ?col - coord
               ?spacecraft2 - spacecraft ?s2-row - coord
               ?from-row - coord ?to-row - coord)
  :precondition (and
                  (at ?spacecraft1 ?from-row ?col)
                  (at ?spacecraft2 ?s2-row ?col)
                  (one-greater-than ?s2-row ?to-row)
                  (greater-than ?to-row ?from-row)
                  (forall (?spacecraft3 - spacecraft ?row - coord)
                    (not (and
                      (at ?spacecraft3 ?row ?col)
                      (greater-than ?row ?from-row)
                      (greater-than ?s2-row ?row)))))
:effect (and (at ?spacecraft1 ?to-row ?col)
          (not (at ?spacecraft1 ?from-row ?col))))

(:action move-down
  :parameters (?spacecraft1 - spacecraft ?col - coord
               ?spacecraft2 - spacecraft ?s2-row - coord
               ?from-row - coord ?to-row - coord)
  :precondition (and
                  (at ?spacecraft1 ?from-row ?col)
                  (at ?spacecraft2 ?s2-row ?col)
                  (one-greater-than ?to-row ?s2-row)
                  (greater-than ?from-row ?to-row)
                  (forall (?spacecraft3 - spacecraft ?row - coord)
                    (not (and
                      (at ?spacecraft3 ?row ?col)
                      (greater-than ?from-row ?row)
                      (greater-than ?row ?s2-row)))))
  :effect (and (at ?spacecraft1 ?to-row ?col)
            (not (at ?spacecraft1 ?from-row ?col)))))
```

Listing 2: Problem 1 specification

```
(define (problem lunar-lockout-b-probone)
  (:domain lunar-lockout-b)
  (:objects red yellow purple orange green - spacecraft
            zero one two three four - coord)
  (:goal (at red two two))
  (:init
   (at red zero four)
   (at yellow one three)
   (at orange four four)
   (at green three two)
   (at purple two one)

   (greater-than one zero)
   (greater-than two zero)
   (greater-than three zero)
   (greater-than four zero)

   (greater-than two one)
   (greater-than three one)
   (greater-than four one)

   (greater-than three two)
   (greater-than four two)

   (greater-than four three)

   (one-greater-than one zero)
   (one-greater-than two one)
   (one-greater-than three two)
   (one-greater-than four three)))
```

Means-Ends Analysis

a)

Sussman's anomaly was a popular solution. The idea here is to point out a situation where the resulting solution requires more options than the optimal solution. This is different from a solution that requires backtracking. The idea is to show that means-ends resolves one goal at a time and may repursue a goal that it undoes in the process of achieving another goal. This results in lengthier plans than necessary.

b)

op2, op1, op3.

c)

Adapted from the answer by Devin Schwab.

Prodigy starts with an empty head and tail plan as shown in Figure 1. In other words the set of relevant operators is empty. The initial state is $g2, g3$ and the goal is $g1, g2$. Goal $g2$ is satisfied by the starting state, so according to Means-Ends Analysis, only $g1$ is in the set of pending goals G. There are no relevant operators because the tail plan is empty. Therefore, at this decision point we must choose the goal $g1$ to expand.

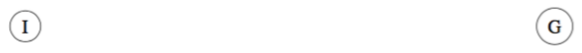


Figure 1: Initial head and tail plan

The only operator relevant to goal $g1$ is $op1$, so we add it to the tail plan. The head and tail plan now look like the one shown in Figure 2.

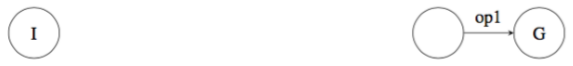


Figure 2: After expanding $g1$

Now goal $g1$ is satisfied by an operator in the tail plan, and goal $g2$ is still in the state resulting from the current head plan. So $G = \emptyset$. Of the operators in the tail plan $op1$ is applicable. So this time we will apply $op1$. This leads to the head and tail plan shown in Figure 3.

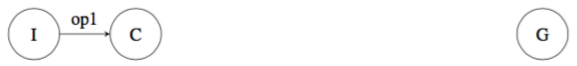


Figure 3: After applying $op1$

By applying $op1$, the current state C becomes $C = \{g1\}$. So the goal $g1$ is satisfied by the current state, but $g2$ is not. The tail plan is now empty so there are no relevant operators to consider applying. Therefore the goal $g2$ is expanded.

The only operator relevant to goal $g2$ is $op3$, which has preconditions $g4$, which is unsatisfied. The head and tail plan now look like Figure 4

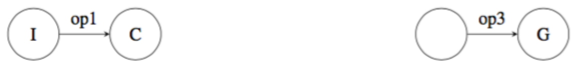


Figure 4: After adding $op3$

Now $g1$ is satisfied by the state in C and $g2$ is satisfied by the $op3$ in the tail plan. This leaves only the g4 goal unsatisfied. $op3$ is the only relevant operator, and it is currently not applicable because C does not contain $g4$. So we expand $g4$.

The only operator relevant to goal $g4$ is $op2$, which has preconditions $g3$ which is unsatisfied. The head and tail plan now look like Figure 5.

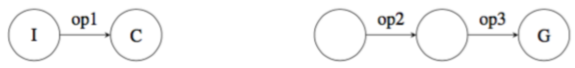


Figure 5: After expanding $g4$

Now the only goal not satisfied by the state C or an operation in the tail plan is $g3$. So $G = g3$. There are two relevant operators $op2$ and $op3$. $op3$ is not applicable because $op2$ must come before it according to the tail plan. $op2$ is not applicable because $g3$ is not in state C. Therefore, we will expand goal $g3$.

There is no operator which can satisfy goal g3. This means that this branch of the prodigy search has failed to find a plan. However, there are no other possible branches (at no point did we make a choice between applying or expanding and when expanding we did not make any choices between applicable operators). Therefore, prodigy will return that no plan exists. When in reality the plan shown in part b is a valid plan for the given starting state and goal.

d)

Adapted from the answer by Devin Schwab.

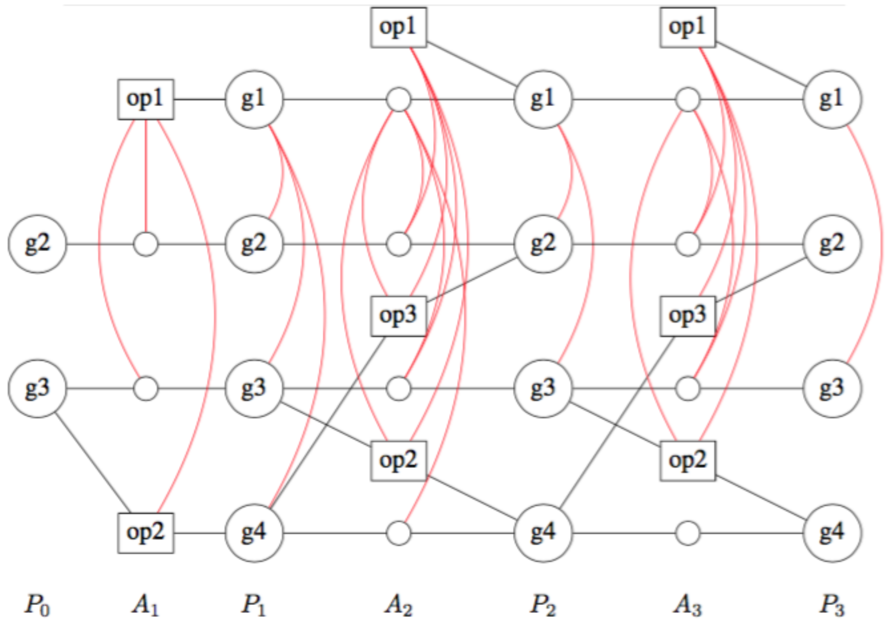


Figure 6: Expansions of the graphplan algorithm

e)

The incompleteness is due to Means-Ends Analysis. By only considering unachieved goals, Prodigy failed to explore the possibility of $op2$ coming before $op1$. A possible solution would be to allow Prodigy to consider all goals, including those already satisfied by the current state.

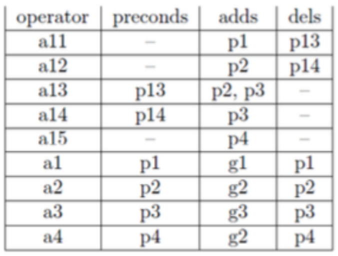
In that case, on the first decision point of the execution of the contrived example, Prodigy would be able to expand either $g1$ or $g2$. By expanding $g2$ we would arrived at a solution.

This solution however, may require some sort of loop detection, to prevent Prodigy from expanding the same goals over and over.

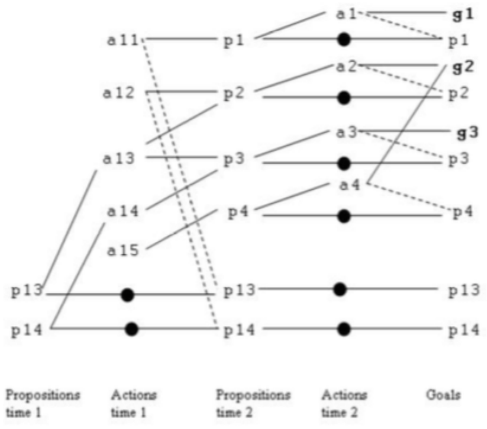
GraphPlan

a)

An example of one reason why Graphplan can backtrack in its backward search is due to the fact that it only considers pairwise mutex exclusion. Consider the following domain and the corresponding planning graph for goals g1, g2, and g3:



operator	preconds	adds	dels
a11	–	p1	p13
a12	–	p2	p14
a13	p13	p2, p3	–
a14	p14	p3	–
a15	–	p4	–
a1	p1	g1	p1
a2	p2	g2	p2
a3	p3	g3	p3
a4	p4	g2	p4



Propositions time 1: No exclusive relations: p13 and p14 are true.

Action time 1: a11 is exclusive of a13, because a11 deletes p13, precondition of a13; a12 is exclusive of a14, because a12 deletes p14, precondition of a14.

Propostions time 2: No pair of propositions is mutually exclusive (mutex): (i) p1 and p2 can be added by actions (a11, a13) mutex, but also by (a11, a12) not mutex; (ii) p1 and p3 can be added by (a11, a13) mutex, but also by (a11, a14) not mutex; and (iii) p2 and p3 can be added by (a12, a14) mutex, but also by (a13, a14) or (a12, a13) not mutex. Note, however, that p1, p2 and p3 cannot all be true simultaneously, because there is no valid action set that will add the three propositions. For example, a11, a12, a13 cannot occur simultaneously because a11 is exclusive of a13. Similarly, a11, a12, a14 cannot occur simultaneously because a12 is exclusive of a14. By looking only at pairwise mutual exclusivity, Graphplan does not detect this.

Actions time 2: No pair of actions are mutex, but (a1, a2, a3) cannot occur simultaneously because p1, p2, p3 cannot be true simultaneously. Graphplan stops its expansion when it finds the goals g1, g2, and g3 at the proposition time level 3 and non-exclusive. The goals can be the result of two sets of actions: a1, a2, a3 or a1, a4, a3. Since there are no mutex relationships in actions in the set, Graphplan may initially pick the set a1, a2, a3 and begin a 1 backwards search for a plan. When it reaches the actions at time step 1 however, it realizes that there is no action set it can pick that will make the propositions p1, p2, p3 true simultaneously. It then needs to backtrack back to the goal level and pick a different set of actions to achieve the goal conditions. It selects a1, a4, a3 whose preconditions are p1, p4, p3 which are added by the valid action set a11, a15, a14.

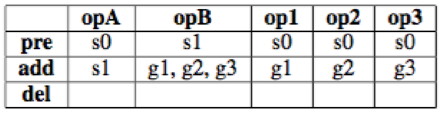
b)

In the backwards search phase, Graphplan either finds a valid plan or proves that no plan can have fewer time steps. Thus, if Graphplan does not find a valid plan during this backwards phase, it does not mean that one does not exist. It only means that there is not a valid solution plan that has the number of time steps expanded. Graphplan proceeds by returning to the forward expansion until it can repeat the backwards search again, now with more time steps.

c)

Yes, it is possible. Graphplan is only optimal in the number of timesteps, not in the number of operators.

The following example was adapted from the answer by Devin Schwab. Start state $s0$ and goal state $g1, g2, g3$.



	opA	**opB**	**op1**	**op2**	**op3**
pre	s0	s1	s0	s0	s0
add	s1	g1, g2, g3	g1	g2	g3
del					

Figure 7: Actions, preconditions and effects

As seen in Figure 8 only one expansion level is needed before all goals propositions are present and non-exclusive. So graphplan will search and return the plan $\{op1, op2, op3\}$.

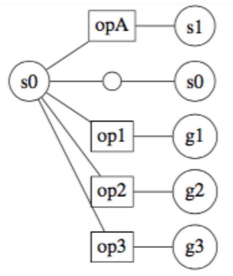


Figure 8: Graph after one expansion

However, if Graphplan expands once more, two possible plans exist: the original and $opA \rightarrow opB$, as seen in Figure 9

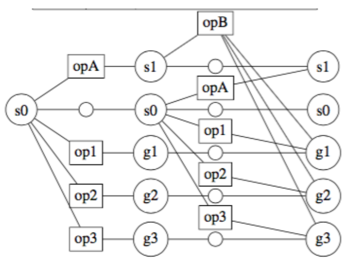


Figure 9: Graph after two expansions