Problem assignment 7

Due: Wednesday, October 23, 2019

Problem 1

Part a.

-Action: **put-on**(x,y)

-Preconditions: Conjunctions of literals with variables

On(x,z), Clear(x), Clear(y), z≠y

-Effects: Two lists:

Add list: On(x,y), Clear(z)

Delete list: Clear(y), On(x,z)

Everything else remain untouched.

-Action: **put-table**(x)

-Preconditions: Conjunctions of literals with variables

On(x,z), Clear(x), z≠Table

-Effects: Two lists:

Add list: On(x,Table), Clear(z)

Delete list: On(x,z)

Everything else remain untouched.

Part b.

The state we obtain after **put-table**(B):

On(B, Table), Clear(A), On(A, C), On(C, Table), On(D, Table), Clear(B), Clear(D).

(Highlighted is new added.)

Part c.

Our final goal state is:

On(C,B), On(B,A), On(A,D), On(D,Table), Clear(C)

Before reaching the final goal, we applied put-on(C,B), according to part a, the action would affect some states:

Add: On(C,B) (Clear(Table) is always be true)

Delete: On(C, Table), Clear(B)

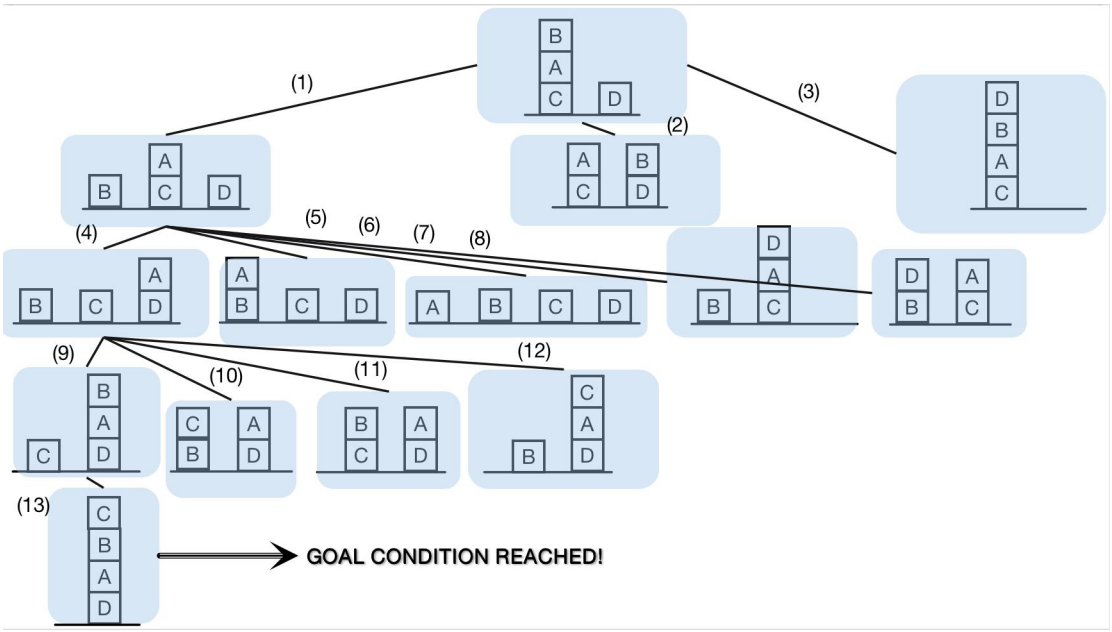
Thus, the new goal results from the selection of put-on(C,B) would be:

On(C,Table), Clear(B), On(B,A), On(A,D), On(D,Table), Clear(C)

(Highlighted is new added.)

Part d.

Depth First Search would be more applicable for solving the planning problem, because we know the depth of the solution, the depth equals to the steps left to the goal state. If we set the depth and eliminate the repetition, the DFS would perform better than other uninformed search methods. And by using DFS, we don't have to keep large number of depth in queues.



Operator (1): put-table(B)
Operator (2): put-on(B,D)
Operator (3): put-on(D,B)
Operator (4): put-on(A,D)
Operator (5): put-on(A,B)
Operator (6): put-table(A)
Operator (7): put-on(D,A)
Operator (8): put-on(D,B)
Operator (9): put-on(B,A)
Operator (10): put-on(C,B)
Operator (11): put-on(B,C)
Operator (12): put-on(C,A)
Operator (13): put-on(C,B)

So the operators lead to the goal conditions are (1),(4),(9),(13).

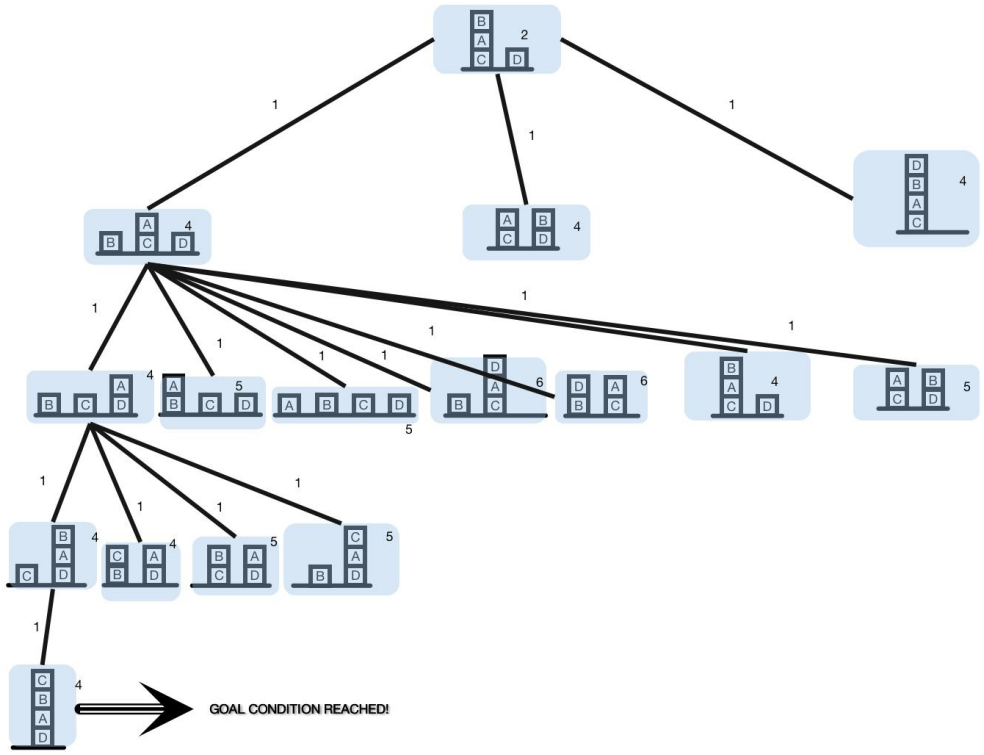
Part e.

The solution found is optimal. It is actually the A* search.

Our goal state is :

On(C,B), On(B,A), On(A,D), On(D,Table).

So there are totally 4 sub-goals to meet.



Part f.

The f(s)=g(s)+h(s) search is actually the so called “A* search” algorithm. According to what we’ve learned before at informed search part, as long as the h(s) is admissible, the evaluation-function should be optimal. So what we have to prove is that h(s) in arbitrary STRIPS planning problem is always admissible, which is $h(s) \leq h^*(s)$ always be true. The reason is that for all the subgoals, they are not always independent. There would sometimes interactions among sub-goals. What’s more, even no interaction, the h(s) is not very applicable. Three reasons:

Uninformative: the range of heuristic values in a given task is small, which means for most successors have the same estimate.

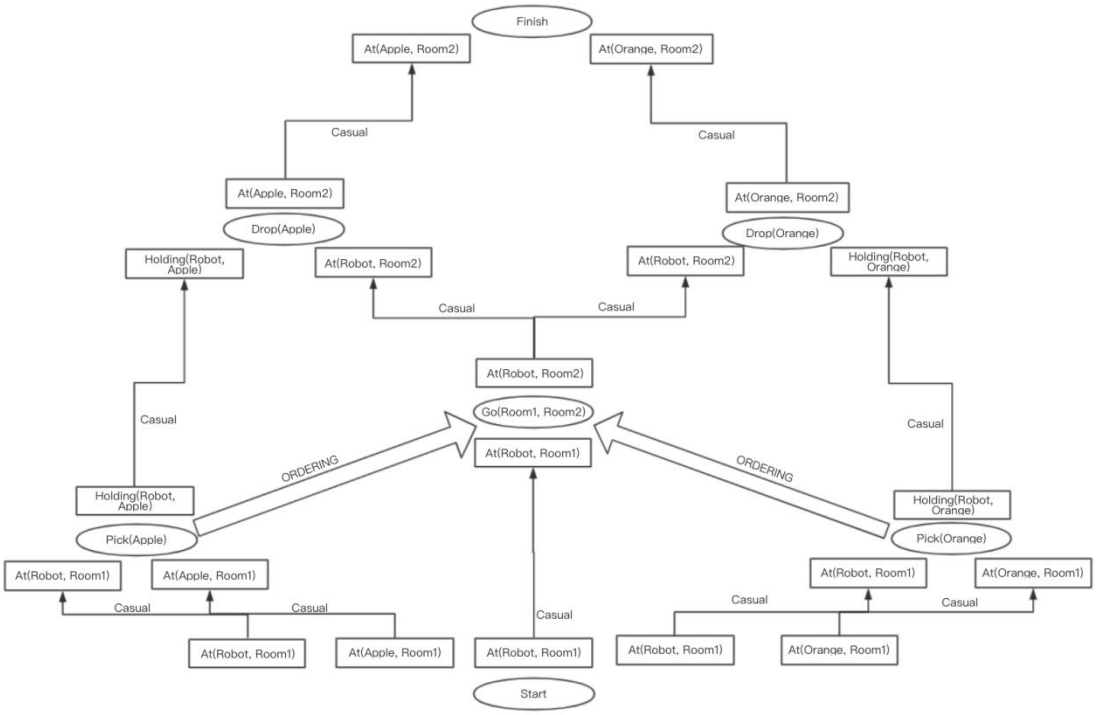
Sensitive to reformulation: if we have a non-goal states when processing, we can easily transform any planning task into an equivalent one where h(s) = 1, then the heuristic could be very misleading.

No problem structure: since the heuristic only count in the sub-goal, we don’t think of the operators, so the heuristic can’t reflect the structure of the problem.

So the evaluation-function search with f(s) = g(s) + h(s) in combination with such a heuristic is not always optimal.

Problem 2.

Part a.



Through the two orderings, the threats between At(Robot, room1) and Go(Room1, Room2) is resolved. If we go(Room1, Room2) first, the state： At(Robot,Room1) would be deleted, then the two Pick(o) actions would not be satisfied. So we set the ordering Pick(o)-Go(x,y).

Part b.

All plan consistent with my partial order plan:

> Pick(Apple), Pick(Orange), Go(Room1, Room2), Drop(Apple), Drop(Orange)
> Pick(Apple), Pick(Orange), Go(Room1, Room2), Drop(Orange), Drop(Apple)
> Pick(Orange), Pick(Apple), Go(Room1, Room2), Drop(Apple), Drop(Orange)
> Pick(Orange), Pick(Apple), Go(Room1, Room2), Drop(Orange), Drop(Apple)