National University of Singapore
School of Computing

Semester 1, AY2021-22

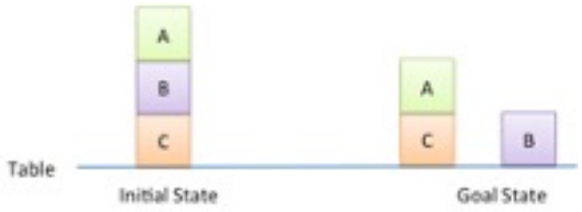
CS4246/CS5446 AI Planning and Decision Making

Assignment 1 Solution Sketches

1 Homework Assignment (LumiNUS Quiz)

1.1 Homework Problem 1: The Blocks World Reloaded

(15 marks) Consider a blocks-world as shown in the figure below. The objective of this problem, called *Middle-pop*, is to remove the block B in-between two other blocks, A and C, and put it on the Table (represented by the horizontal line).



a. Express the blocks-world problem as a planning problem in the Planning Domain Definition Language (PDDL). You may use the following assumptions, and/or make **additional** assumptions as necessary:

- There are only four objects in the world – Block A, Block B, Block C, and the Table.
- A set of predicates define the descriptions of and the relations among the objects:
  - A block can be on top of another object, i.e., **On**(*x, y*), where x is a block and *y* is another object.
  - The top of the object to be clear, i.e., **Clear**(*x*), where *x* is an object, before another object can be placed on top of it.
  - You can define additional predicates if necessary.
- A set of action schema and actions define how to change the world state:
  - A block can be re-arranged by moving it to (the top of) another object, i.e., **Move**(*x, fr, to*), where *x* is a block, and *fr, to* are other objects in the world.
  - A block can be moved only when it is not stacked upon, i.e., it is "clear" on the top, or **Clear**(*x*), where x is a block.

Please remember these are not exact solutions, but a guide. Comments and suggestions are included in the boxes.

- You cannot move the Table, but you can always move something to or from the Table, i.e., **Clear**(Table) is always assumed to be True (interpreted as: there is always "clear space" on the Table).
- You can define additional action schemas or actions as necessary.

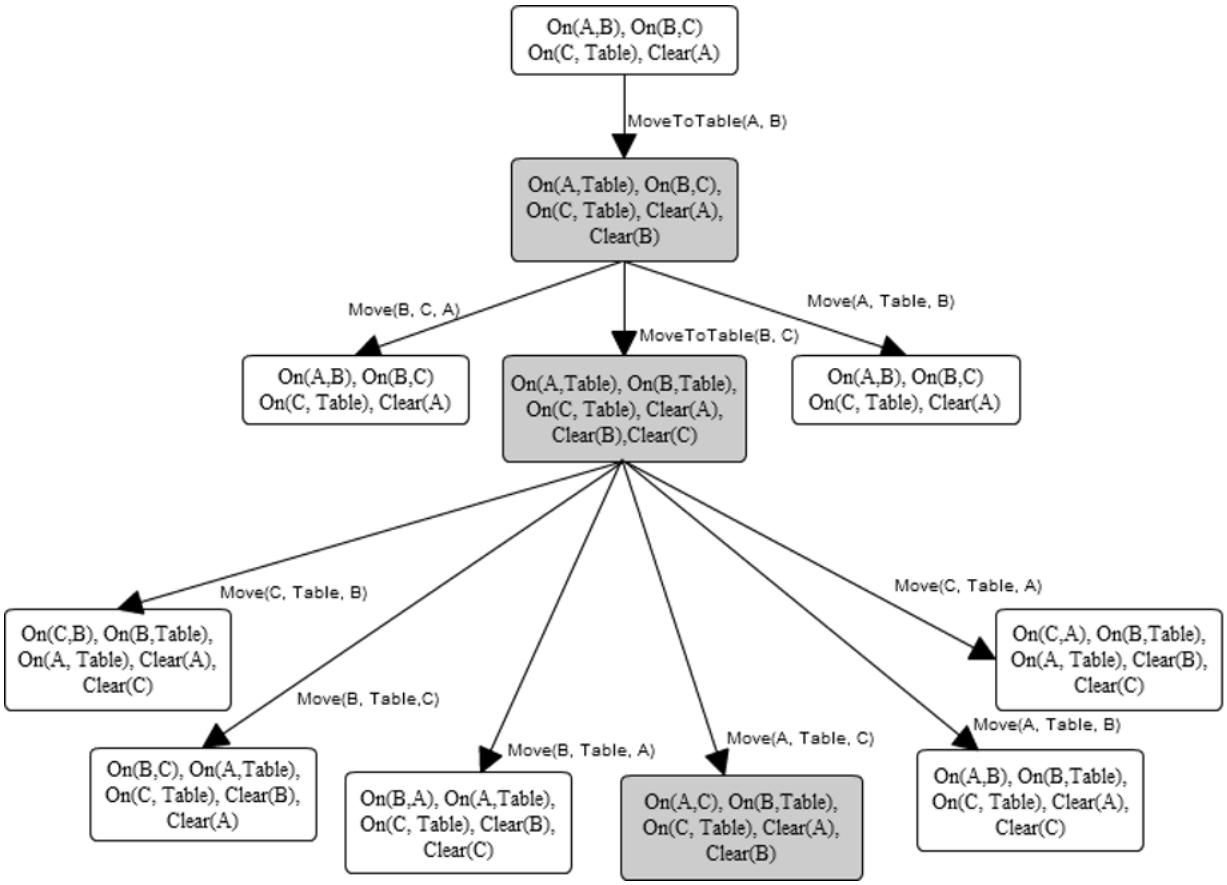
Solution:

Additional predicates:
Block(x): Object x is a block
Initial state: On(A,B) ∧ On(B,C) ∧ On(C, table) ∧
Clear(A) ∧ Block(A) ∧ Block(B) ∧ Block(C)
Action schema:
Move(x, fr, to)
Precond: Clear(x) ∧ Block(x) ∧ On(x, fr) ∧ Clear(to)
Effect: On(x, to) ∧ Clear(fr) ∧ :On(x, fr) ∧ :Clear(to) *MoveToTable(x, fr)*
Precond: Clear(x) ∧ Block(x) ∧ Block(fr) ∧ On(x, fr)
Effect: On(x, table) ∧ Clear(fr) ∧ :On(x, fr)
Goal:
On(A,C) ∧ On(C, table) ∧ On(B, table) ∧ Clear(A) ∧ Clear(B)

b. Show the search tree for planning by forward search.

Solution:

- The search tree should be a *tree* not a graph
- Depending upon the state space definition, the tree might differ
- You may exclude some states in the search tree due to lack of space, or summarize understood variations; but all the assumptions and omissions should be clearly stated.



c. Show a valid plan for the problem.

Solution:

Valid Plan:
MoveToTable(A, B)
MoveToTable(B, C)
Move(A, Table, C)

1.2 Homework Problem 2: High Level Actions

(10 marks) [RN3e 11.1] Armed with the AI planning knowledge that you have learned in CS4246/CS5446, you set up a new parcel delivery company – Swift Hoppers Delivery Services. You have a number of vans with which to deliver a set of packages. Each package starts at some location on a grid map, and has a destination somewhere else. Each van is directly controlled by moving forward and turning left or turning right.

a. Construct a hierarchy of high-level actions for Swift Hoppers Delivery Services. State clearly any assumptions you make.

Solution:

- The hierarchy can consist of high level actions (HLA) denoting the activities of a delivery service. You can define your own operative principles
- The expectation of this exercise is to observe the structure of HLAs and primitive actions and how we can use their relation to represent operative principles
- You can have HLAs to represent *navigation* and *delivery*. You can also have slightly granular HLAs including *scheduling* and *Delivering multiple parcels*
- You can include a hierarchy of HLAs as well, with a intermediate layer of HLAs above the primitive actions
- You can also include refinements of HLAs to represent variations in action schema
- We provide a sample solution below

Primitive actions:
For movement Forward (v), TurnLeft(v), and TurnRight (v) where v is a van,
For package delivery: Load(p, v) and Unload(p, v) where p is a package and v is a van.

HLAs:
HLANavigate(v, [x, y]) to take a van v to coordinates [x, y],
Deliver(v, p) to deliver package p to its destination with van v.

Assumptions:

- The fluent At(p, [x, y]) for vans and packages p records their current position [x, y]
- The predicate Destination(p, [x', y']) gives the package's destination.
- Vans can only carry one package at a time, can only drop packages off at their destinations not intermediate points, and can only have serialize deliveries (no representation for parallel actions here).
- The planner needs to choose which vans deliver which packages in what order, and vans should navigate given their destinations.

Van Hierarchy:

Refinement(Deliver (t, p),
PRECOND:Van(v) $\wedge$ Package(p) $\wedge$ At(p, [x, y]) $\wedge$ Destination(p, [x', y'])
STEPS: [Navigate(t, [x, y]), Load(p, t), Navigate(t, [x', y']), Unload(p, t)])

Refinement(Navigate(t, [x, y]),
PRECOND:Van(v) ∧ At(t, [x, y])
STEPS: []) Refinement(Navigate(t, [x, y]),
PRECOND:Van(v)
STEPS: [Forward(v),Navigate(t, [x, y])]) Refinement(Navigate(t, [x, y]),
PRECOND:Van(v)
STEPS: [TurnLeft(v),Navigate(t, [x, y])]) Refinement(Navigate(t, [x, y]),
PRECOND:Van(v) STEPS: [TurnRight(v),Navigate(t, [x, y])])

b. What knowledge about the solution does your hierarchy encode?
Solution:

- A ”human” understandable description or summary of what the hierarchy is about. A brief description of what the HLAs and primitive actions mean.
- This can include knowledge about how to deliver a package through navigation, and guide the vehicle to turn to different directions.
- This can also include the actions that the primitive actions assume, e.g. the driver gets in the van, the driver picks up the parcel, etc.
- These are different from assumptions, one can think of it as the pseudo-code of the HLAs or primitive actions. We need to implement it, not assume it.

2 Programming Assignment (aiVLE submission)

2.1 Programming problem 1: Parking Task

(15 marks) You are employed as a valet at a parking lot. You have to drive the car given to you and park it at a parking spot assigned to the car (identified as goal state in the environment). But, you also have to plan your way from your current position to the spot while wasting the least fuel possible (your tip depends on it!).

By virtue of being a Computing student, you decide to put your *planning* skills to use. You first retrieve an old python script written for a similar task by the TAs of CS4246/CS5446, parts of which have been lost. The script used to generate the PDDL files and fed them to the fast-downward solver to generate a solution for this planning problem. It also runs the plan on the simulator to see if it does the job.

The script `python __init__.py` is missing 3 code snippets (marked with "FILL ME" in the file) .

1. Code which generates the action schemas of the three actions available namely, UP, DOWN and FORWARD. These action schemas should reflect the 3 actions available in the environment simulator. The agent's speed range in the environment is restricted to [-1, -1] (negative speed moves towards the left, see the environment for more details).

2. Code which generates the initial state condition.

3. Code which generates the goal description.

You can test your code with the test configurations given in the script by running `python __init__.py parking N` (present in the .zip file) , where N is an int value between 0 to 5, on the docker (using the `docker run ...` command). **It might be helpful to look at the generated PDDL files for debugging.**

Solution:

- In initial states, you need to represent all relationships between the grid cells using xxx_next, and list all blocked cells.
- The goal state is a description, you only need to specify the location of the agent.
- To implement the actions, check the relationship between pt1 and pt2 as well, as the pt2 is not blocked.

2.2 Programming problem 2: Crossing the road

(10 marks) A customer forgets directions to the parking lot, and ends up on the other side of the road. The customer is not skilled enough to cross a busy multi-lane road with **moving cars**, where the speed of cars can differ across lanes, but cars in the same lane move with the same speed. To reach the entrance of the parking lot (identified as goal state in the environment), he calls you up and asks you to drive the car from his spot (the initial state) to the destination. Unlike the previous task, the agent's speed range in the environment is restricted to [-3, -1].

You decide to modify the script you wrote in Task 1 to handle this situation. You can test your code by running `python __init__.py crossing 0` in command with docker

Hint: Instead of modeling the parking lot/road as a 2 dimensional grid, it might be useful to include time as an additional dimension.

Solution:

- As the hint in the assignment handout mentioned, you need to introduce a new dimension *time*, because the cars are moving, so they are not only blocked at some grid cells, but also at a particular time step.
- However, you cannot represent the time in xxx_next predicates, but only in blocked predicates. That's because if you list all possible combinations of coordinates and time points, it will make the search space extremely large. For e.g., if we have M coordinates and N time points, using a variable like ptXptYtT will need around $M \times N$ different ground states, however, if we separate them, we only need around $M + N$ ground states, (one part to represent the coordinates and one part to represent the time relationship). Note that the M and N are very large when the environment gets much more complex. This is a main reason why some submissions got *TimeoutException* error.
- Note: The agent cannot "fly" over the cars in front of it, especially when it moves 2 or 3 cells, so only checking the pt2 3-forward-next or 2-forward-next to pt1 is not enough.