CMPT 310
Assignment 3
Planning and Logic

Jiadi Luo 301354107
Haoxuan Zhao 301385109

Instructions:

- You can do this assignment individually or in a team of two. If you are doing it in a group, only one submission per group is required.
- You may submit multiple times until the deadline. Grade penalties will be imposed for late submissions (see the course outline for the details).
- Always plan before coding.
- For coding questions - Use function-level and inline comments throughout your code. We will not be specifically grading documentation. However, remember that you will not be able to comment on your code unless sufficiently documented. Take the time to document your code as you develop it properly.
- We will carefully analyze the code submitted to look for plagiarism signs, so please do not do it! If you are unsure about what is allowed, please talk to an instructor or a TA.
- Make sure that in your answers, you clearly indicate the exact section you are answering.
- Check A3 rubric available on canvas.
- You will submit one .zip file.
    - o For theoretical questions - Your submission must be formatted as a single PDF file; other types of submissions (non-pdf, multiple files, etc.) will not be graded. Your name(s) and student number(s) must appear at the top of the first page of your submission.
    - o Make sure that in your answers, you clearly indicate the exact section you are answering.
- Please put all your files (pdf + code) in one folder. Compress these into a single archive named a3.zip and submit it on Canvas before the due date listed there.

Ques 1. [25 marks] Assume we have a world with 9 different locations in a 2x4 grid defined as (x, y) as shown below. There is a game agent located at position (0,0) that wants to end up at location (4,1). We want to write an automated planner for this simple video game.

(0,0)	(1,0)	(2,0)	(3,0)	(4,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)

There is a monster at each of positions (1,0), (3,1), and (4,0). The agent cannot move into these squares when there is a monster there. However, a weapon is held by the agent that is either charged or uncharged. There are weapon charges at locations (0,0) and (1,1). The agent can perform the following actions:

1) "walkRight" increases the agent's x location by 1 (max 4).
2) "walkLeft" decreases the agent's x location by 1 (min 0).
3) "walkDown" increases the agent's y location by 1 (max 1).
4) "walkUp" decreases the agent's y location by 1 (min 0).
5) "obtain" will have the agent obtain the charge at its location if there is one. This will charge the agent's weapon (Regardless of whether it is already charged), removing the charge from its position.
6) "discharge" destroys **every** monster at locations from the **agent** that are either $(x, y\pm1)$ or $(x\pm1, y)$ and completely discharges the weapon.

We will use the STRIPS notation to write this planning problem. The actions cannot be parameterized because the effects must be constant without depending on the agent's state. Therefore, an action **cannot** be written as "walkRight" parameterized by location with "Pos = (PrevPos[x] + 1, PrevPos[y])" as an effect. Instead, the actions must be "walkRight$[i, j]$," "walkLeft $[i, j]$," "walkDown$[i, j]$," "walkUp$[i, j]$," "obtain$[i, j]$," and "discharge$[i, j]$" for $i = 0, 1, 2, 3, 4,\ \ j = 0, 1$, where each of these actions can only be applicable at position (i, j) (This is similar to the robot delivery example discussed in class for *move-clockwise* and *move-counter-clockwise* actions).

(a) **[5 marks]** Define STRIPS features and their domains for the problem. The agent needs to keep track of its location, whether monsters exist at the 3 locations, whether the weapon is charged, and whether there is an available charge at (0,0) and (1,1).
Features: atLoci_j for i ∈ {0,1,2,3,4}, j ∈ {0,1}; monsterAtLoc1_0; monsterAtLoc4_0; monsterAtLoc3_1; chargeWeaponAtLoc0_0; chargeWeaponAtLoc1_1; isWeaponCharged.
Domain = {T, F} for all features mentioned above. In other words, all features are binary.

(b) **[8 marks]** Give STRIPS representation for location (1,1) ("walkRight[1,1]," "walkLeft[1,1]," "walkUp[1,1]," "obtain[1,1]," and "discharge[1,1]")

walkRight[1,1]:
preconditions: {atLoc1_1 = T}
effects: {atLoc1_1 = F; atLoc2_1 = T}

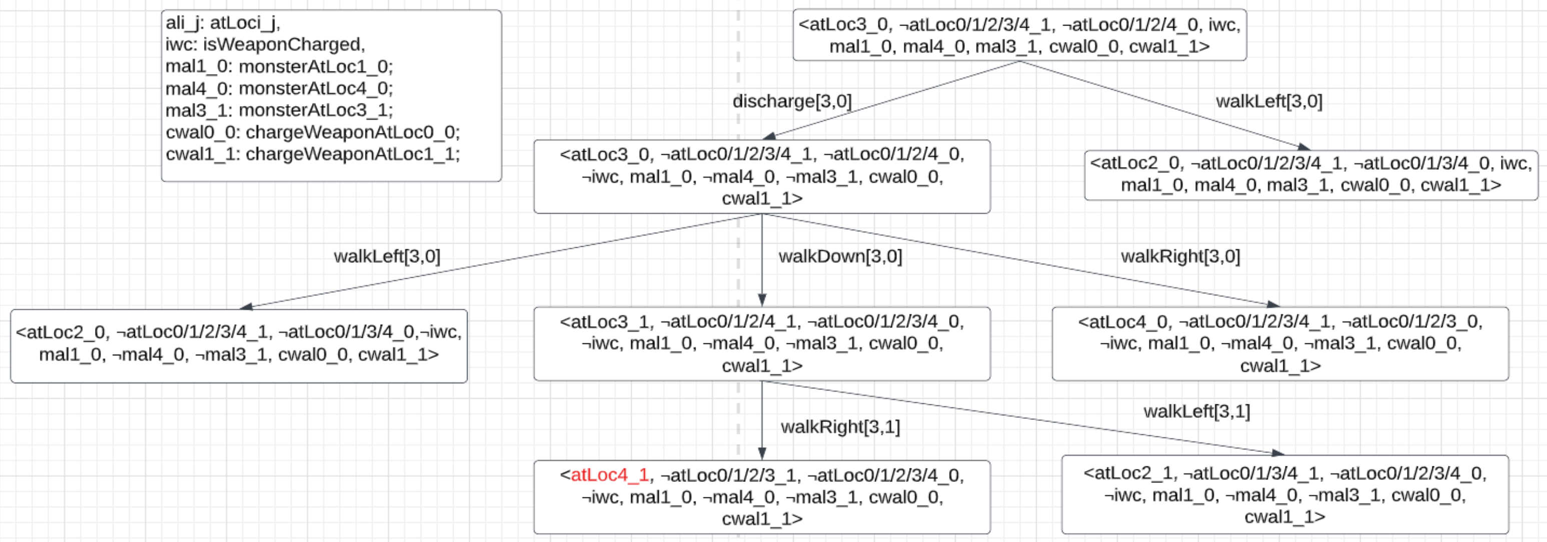
walkLeft[1,1]:
preconditions: {atLoc1_1 = T}
effects: {atLoc1_1 = F; atLoc0_1 = T}

walkUp[1,1]:
preconditions: {atLoc1_1 = T; monsterAtLoc1_0 = F}
effects: {atLoc1_1 = F; atLoc1_0 = T}

obtain[1,1]:
preconditions: {atLoc1_1 = T; chargeWeaponAtLoc1_1 = T}
effects: {chargeWeaponAtLoc1_1 = F; isWeaponCharged = T}

discharge[1,1]:
preconditions: {atLoc1_1 = T; isWeaponCharged = T}
effects: {isWeaponCharged = F; monsterAtLoc1_0 = F}

(c) **[4 marks]** Suppose we change the start state to (3,0) with the agent's weapon already charged, and everything else the same. Draw the search graph including the optimal plan (Optimal solution path from the start state to a goal state) like the coffee delivery example in class. Immediate successor states should be included alongside the optimal path.


ali_j: atLoci_j,
iwc: isWeaponCharged,
mal1_0: monsterAtLoc1_0;
mal4_0: monsterAtLoc4_0;
mal3_1: monsterAtLoc3_1;
cwal0_0: chargeWeaponAtLoc0_0;
cwal1_1: chargeWeaponAtLoc1_1;
<atLoc3_0, ¬atLoc0/1/2/3/4_1, ¬atLoc0/1/2/4_0, iwc, mal1_0, mal4_0, mal3_1, cwal0_0, cwal1_1>
discharge[3,0]
walkLeft[3,0]
<atLoc3_0, ¬atLoc0/1/2/3/4_1, ¬atLoc0/1/2/4_0, ¬iwc, mal1_0, ¬mal4_0, ¬mal3_1, cwal0_0, cwal1_1>
<atLoc2_0, ¬atLoc0/1/2/3/4_1, ¬atLoc0/1/3/4_0, iwc, mal1_0, mal4_0, mal3_1, cwal0_0, cwal1_1>
walkLeft[3,0]
walkDown[3,0]
walkRight[3,0]
<atLoc2_0, ¬atLoc0/1/2/3/4_1, ¬atLoc0/1/3/4_0,¬iwc, mal1_0, ¬mal4_0, ¬mal3_1, cwal0_0, cwal1_1>
<atLoc3_1, ¬atLoc0/1/2/4_1, ¬atLoc0/1/2/3/4_0, ¬iwc, mal1_0, ¬mal4_0, ¬mal3_1, cwal0_0, cwal1_1>
<atLoc4_0, ¬atLoc0/1/2/3/4_1, ¬atLoc0/1/2/3_0, ¬iwc, mal1_0, ¬mal4_0, ¬mal3_1, cwal0_0, cwal1_1>
walkRight[3,1]
walkLeft[3,1]
<atLoc4_1, ¬atLoc0/1/2/3_1, ¬atLoc0/1/2/3/4_0, ¬iwc, mal1_0, ¬mal4_0, ¬mal3_1, cwal0_0, cwal1_1>
<atLoc2_1, ¬atLoc0/1/3/4_1, ¬atLoc0/1/2/3/4_0, ¬iwc, mal1_0, ¬mal4_0, ¬mal3_1, cwal0_0, cwal1_1>

For the remaining questions, let the goal be that the agent should be in location (4,1) and have a charged weapon.

(d) **[2 marks]** What is a good admissible heuristic for this planning goal? (Note that this is a domain-dependent heuristic) To get full marks, you must also provide an explanation of why your proposed heuristic is good and is *admissible.*

A good admissible heuristic can be the shortest distance between the agent and goal since the shortest distance is always less than or equal to the least number of actions that the agent needs to perform and this heuristic will always lead the agent to the direction of goal.

Let's now consider the domain-independent heuristic "ignore preconditions", mentioned in the lectures. Recall that for a state *n* this is defined as the cost of the optimal path from n to a goal using a relaxed problem whose actions have empty preconditions.

(e) **[2 marks]** Give the value of this heuristic for an agent at (4,0), all cells free of monsters, weapon is charged, and charges are all available. Justify. Represent the state using features from (a).

State: <atLoc4_0, ¬atLoc0/1/2/3_0, ¬atLoc0/1/2/3/4_1, isWeaponCharged, ¬monsterAtLoc1_0, ¬monsterAtLoc4_0, ¬monsterAtLoc3_1, chargeWeaponAtLoc0_0, chargeWeaponAtLoc1_1>

Value of heuristic: Since we are ignoring preconditions, in this case, we can apply action (walkDown[4,0]) directly by ignoring all its preconditions. Therefore, the heuristic is 1.

(f) **[2 marks]** Give the value of this heuristic for an agent at (3,0), monsters all exist, weapon is charged, and all charges are available. Justify. Represent the state using features from (a).
State: <atLoc3_0, ¬atLoc0/1/2/3/4_1, ¬atLoc0/1/2/4_0, isWeaponCharged, monsterAtLoc1_0, monsterAtLoc4_0, monsterAtLoc3_1, chargeWeaponAtLoc0_0, chargeWeaponAtLoc1_1>
Value of heuristic: Since we are ignoring preconditions, in this case, we can apply action (walkRight[3,1]) directly by ignoring all its preconditions. Therefore, the heuristic is 1.

(g) **[2 marks]** Why is this heuristic not that useful?

As we can see from (e) & (f), the heuristics will always be 1 no matter where the agent is. This is because we are ignoring preconditions, thus we can apply either walkDown[4,0] or walkRight[3,1] directly from every location resulting in a heuristic of 1. This heuristic supplies no useful information and this is also a shortcoming of "ignore preconditions" heuristics: admissible but not very informative.

Ques 2. [25 marks] Bob wants to plan a journey and he has two options. The first option is traveling by car and the second one is to travel by airplane. He has to repair his old car if he wants to use it for his travel or he has to buy a ticket if he chooses the second option and Both of these actions need money, which can be obtained by going to work.

Figure 1 shows the STRIPS representation for this problem.

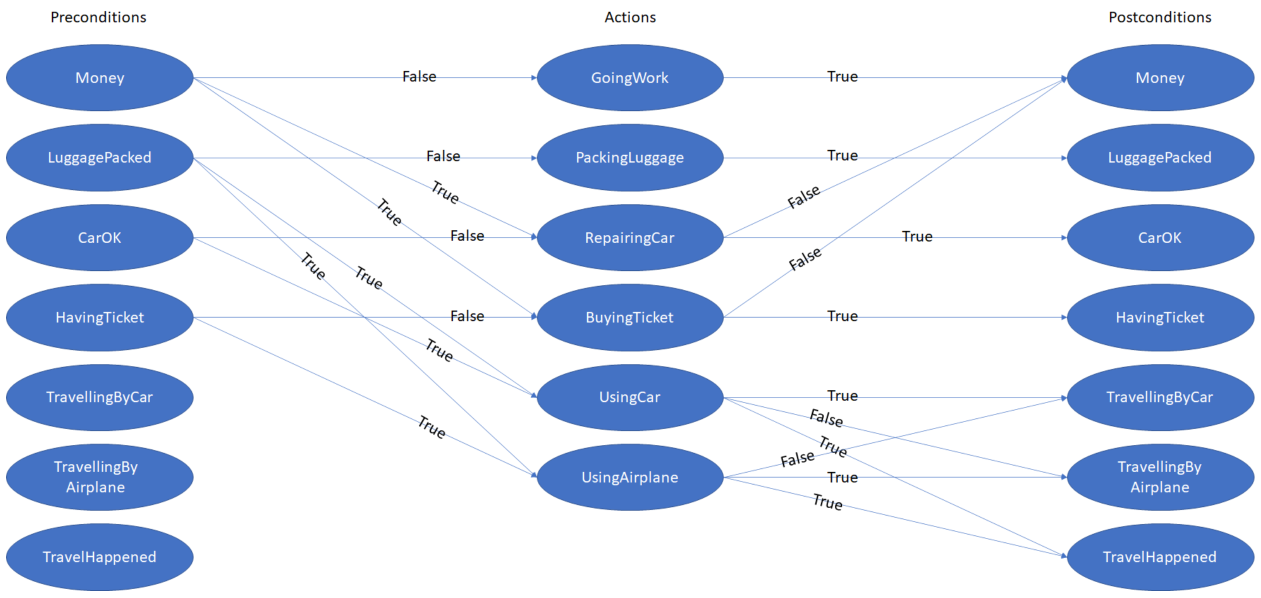


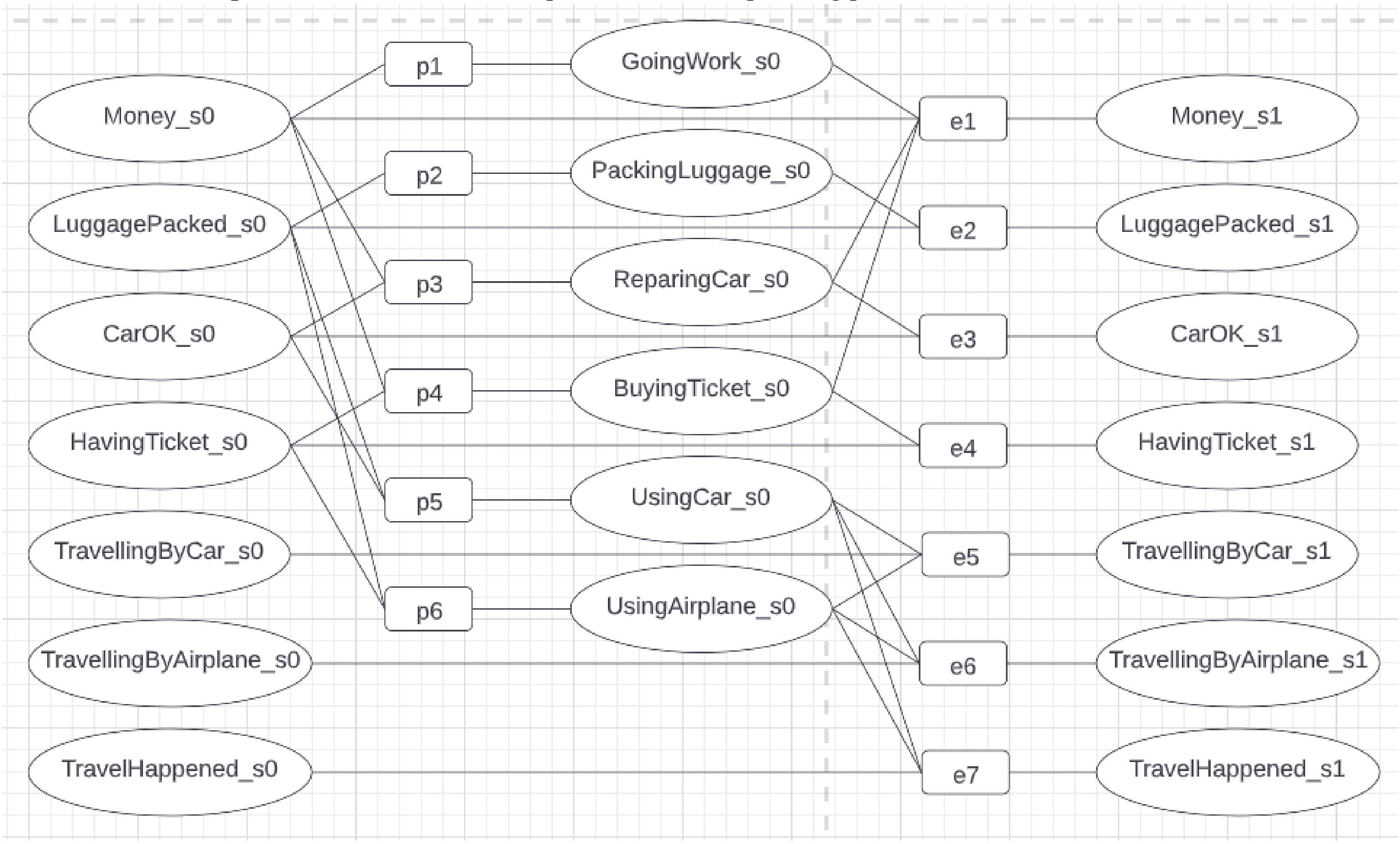
Figure 1 - STRIPS representation for the travel planning problem

(a) [5 points] List the variables/features that define the states in the problem, as well as their domains.
Variables: Money, LuggagePacked, CarOK, HavingTicket, TravellingByCar, TravellingByAirplane, TravelHappened

Domain = {T, F} for all variables mentioned above. In other words, all variables are binary.

(b) [8 points] List the actions in this problem, with their preconditions and effects.
GoingWork:
Preconditions: {Money = F}
Effects: {Money = T}
PackingLuggage:
Preconditions: {LuggagePacked = F}
Effects: {LuggagePacked = T}
ReparingCar:
Preconditions: {Money = T; CarOK = F}
Effects: {Money = F; CarOK = T}
BuyingTicket:
Preconditions: {Money = T; HavingTicket = F}
Effects: {Money = F; HavingTicket = T}
UsingCar:
Preconditions: {LuggagePacked = T; CarOK = T}
Effects: {TravellingByCar = T; TravellingByAirplane = F; TravelHappened = T}
UsingAirplane:
Preconditions: {LuggagePacked = T; HavingTicket = T}
Effects: {TravellingByCar = F; TravellingByAirplane = T; TravelHappened = T}

(c) [2 point] Convert this STRIPS problem to a CSP planning problem with a horizon of 1.



(d) [3 points] Show the truth table representing the constraint Pre_RepairingCar_0 and explain what the different rows mean.

Money_ s0	CarOK_ s0	ReparingCar_ s0	Explanation
T	T	F	Have money but the car is OK, we can't fix it
T	F	T	Have money and the car is not OK, we can fix it
T	F	F	Have money and the car is not OK, but choose not to fix the car
F	T	F	Don't have money and the car is OK, we can't fix it
F	F	F	The car is not OK but don't have money, we can't fix it

(e) [3 points] Show the truth table representing the constraint Effect_travelHappened_1 and explain what the different rows mean.

TravelHappened_s0	Using_Car_s0	Using_Airplane_s0	TravelHappened_s1	Explanation
T	T	T	T	Travel happened at s0 already
T	T	F	T	Travel happened at s0 already
T	F	T	T	Travel happened at s0 already
T	F	F	T	Travel happened at s0 already
F	T	T	T	TravelHappened_s0 is F, this row is not important since there are other constraints restricting that using car and using plane cannot happen together
F	T	F	T	TravelHappened_s0 is F, if either Using_Car_s0 or Using_Airplane_s0 is T then TravelHappened_s1 is T
F	F	T	T	TravelHappened_s0 is F, if either Using_Car_s0 or Using_Airplane_s0 is T then TravelHappened_s1 is T
F	F	F	F	TravelHappened_s0 is F, don't use car or plane, TravelHappened_s1 is F

(f) [4 points] Suppose Bob does not have money and he doesn't have a ticket and his car needs to be fixed. At what minimum horizon do we need to unroll the CSP if the goal is *TravelHappened* = T? Give a plan to reach the goal, listing the action(s) taken at each step.
a minimum horizon of 4 is needed.

Step 1: GoingWork to gain money

Step 2: RepairingCar to make Car OK

Step 3: Pack luggage

Step 4: UsingCar to reach, then final goal TravelHappend = T

Logic

Ques 3. [25 marks] Consider the following knowledge base KB.

1. $a \leftarrow c \land j$
2. $b \leftarrow c \land g \land i$
3. $c \leftarrow a \land i$
4. $c \leftarrow h$
5. $d \leftarrow h$
6. f
7. $g \leftarrow e \land h$
8. h
9. $i \leftarrow a \land f$
10. $i \leftarrow d \land e$
11. $i \leftarrow j$
12. $j \leftarrow g$
13. $j \leftarrow h$

a) **[6 marks]** Using the bottom-up proof procedure, list all of the atoms v such that KB $\vdash$ v. Show your work, using the numbers provided above to indicate which clauses are involved in each step of your work.
clause 8: {h}
clause 6: {h, f}
clause 4: {h, f, c}
clause 5: {h, f, c, d}
clause 13: {h, f, c, d, j}
clause 11: {h, f, c, d, j, i}
clause 1: {h, f, c, d, j, i, a}

b) **[2 points]** Is it true that for all these atoms *v*, KB $\models$ *v*? Why or why not?
Yes, because the bottom-up proof is sound and everything derived from a sound proof procedure is entailed by the KB.

c) **[2 points]** Are there any other atom v satisfying KB $\vdash$ *v* but is not found by the bottom-up proof procedure? Why or why not?
No, because the bottom-up proof is complete and everything entailed by the KB can be derived from a complete proof procedure.

d) **[2 points]** How many possible interpretations of the KB are there?
2^10 since there are in total of 10 atoms and each has a binary property.

e) **[3 points]** Provide an interpretation of the KB that is not a model (justify your answer).
I: {a = T, b = T, c = F, d = T, e = T, f = T, g = T, h = F, i = F, j = F}
In this case, we have several clauses are false, for example
    j (F) $\leftarrow$ i (T)
    j (F) $\leftarrow$ g (T)
    h (F)

Using the top-down resolution proof method, answer if the following queries are logical consequences of KB. If the answer is "yes" show the successful derivation; and if the answer is "no", show one of the failing derivations.

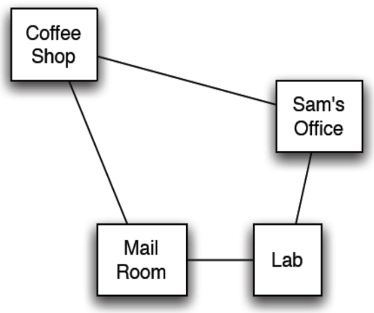
f) **[5 marks]** ? a ∧ i.
γ0: yes ← a ∧ i
γ1: yes ← c ∧ j ∧ i => resolving against clause 1
γ2: yes ← h ∧ j ∧ i => resolving against clause 4
γ3: yes ← h ∧ j ∧ j => resolving against clause 11
γ4: yes ← h ∧ h ∧ h => resolving against clause 13
γ5: yes ← . => resolving against 8
Therefore, a ∧ i is a logical consequence of KB.

g) **[5 marks]** ? a ∧ b.
γ0: yes ← a ∧ b
γ1: yes ← c ∧ j ∧ b => resolving against clause 1
γ2: yes ← c ∧ j ∧ c ∧ g ∧ i => resolving against clause 2
γ3: yes ← h ∧ j ∧ g ∧ i => resolving against clause 4
γ4: yes ← h ∧ j ∧ g ∧ j => resolving against clause 11
γ5: yes ← h ∧ g => resolving against clause 13
γ6: yes ← h ∧ e => resolving against clause 7
γ7: yes ← e => resolving against clause 8
Failing derivations.
Note that, clause 2 is the only clause that derives b, the derivation of b must require g and the derivation of g must require e which is not derivable in this KB, thus a ∧ b cannot be derived from this KB in any order of applying top-down proof procedure.

Coding

You can use whatever programming language you like.

Ques 4.[25 marks] We want to solve the delivery robot example using planning as CSP. Remember that Rob is a delivery robot who navigates in the following environment:



The state is defined by the following variables/features:

- **RLoc** - Rob's location
  - Domain: Coffee Shop (*cs*), Sam's office (*off*), Mail Room (*mr*), or Laboratory (*lab*)
- **RHC** - Rob has coffee.
  - Domain: True or False
- **SWC** - Sam wants coffee.
  - Domain: True or False
- **MW** - Mail is waiting.
  - Domain: True or False
- **RHM** - Rob has mail.
  - Domain: True or False

The actions are defined by the following preconditions and results:

Action	**Preconditions**	**Effects/ Results**
Move Rob's move action: Move clockwise (***mc***), Move counter-clockwise (***mcc***)		
PUC: Rob picks up coffee	Rob must be at the coffee shop	**Rob has coffee**
DelC: Rob delivers coffee	Rob must be at the office and must have coffee	**Rob does not have coffee, Sam does not want coffee**
PUM: Rob picks up mail	Rob must be in the mail room, and mail must be waiting	**Rob has mail, mail is not waiting**
DelM: Rob delivers mail	Rob must be at the office and have mail	**Rob does not have mail**

(a) Describe all actions using the STRIPS representation.

Move:

mc_cs:
- Preconditions: {cs}
- Effects: {¬cs, off}

mc_off:
- Preconditions: {off}
- Effects: {¬off, lab}

mc_lab:
- Preconditions: {lab}
- Effects: {¬lab, mr}

mc_mr:
- Preconditions: {mr}
- Effects: {¬mr, cs}

mcc_cs:
Preconditions: {cs}
Effects: {¬cs, mr}

mcc_mr:
Preconditions: {mr}
Effects: {¬mr, lab}

mcc_lab:
Preconditions: {lab}
Effects: {¬lab, off}

mcc_off:
Preconditions: {off}
Effects: {¬off, cs}

PUC:
Preconditions: {cs}
Effects: {rhc}

DelC:
Preconditions: {off, rhc}
Effects: {¬rhc, ¬swc}

PUM:
Preconditions: {mr, mw}
Effects: {rhm, ¬mw}

DelM:
Preconditions: {off, rhm}
Effects: {¬rhm}

(b) Now you need to reformulate the STRIPS model as CSPs with different horizons, and then solve them by implementing the Arc Consistency + Domain Splitting algorithm. The pseudocode is provided in the below:

```
solved = false
horizon = 0
while solved = false
     map STRIPS into CSP with horizon
     solve CSP → solution
     if solution then
        solved = T
     else
        horizon = horizon + 1
Return solution
```

Remember that if the horizon is *k*, you need to define a CSP variable for each STRIPS variable at time step 0 to *k*, and for each STRIPS action at time step 0 to *k* - 1. You also need CSP constraints for start and goal values, as well as preconditions and effects of actions. Another constraint that you have to consider is that at each time step only one action can be done (mutual exclusion.)

The initial state is **RLoc**: *off*, **RHC**: *F*, **SWC**: *T*, **MW**: *F*, and **RHM**: *T*.
The goal state is **RLoc**: *off* and **SWC**: *F*.

(c) Your program should output the horizon of the CSP that it has solved. It should also print all the variable assignments in the solution.

For example:

Horizon: 4
RLoc0 = off
RHC0 = F
etc…
Move0 = T
PUC0 = F
etc…
RLoc1 = cs
RHC1 = F
etc…

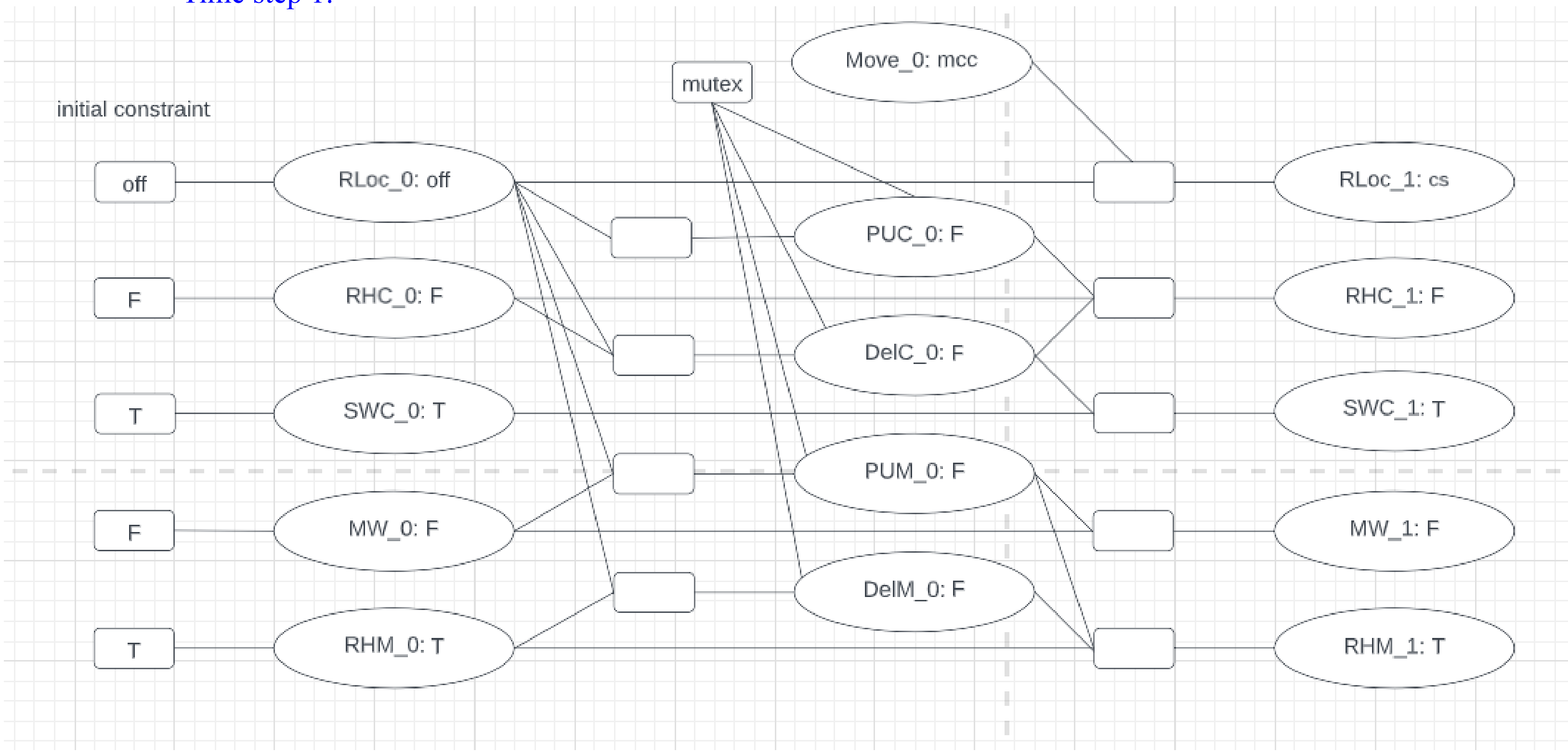
Output: (“nm” for Not Move)

```
Horizen: 4
RLoc_0 {'off'}
RHC_0 {False}
SWC_0 {True}        Initial variable assignments (initial)
MW_0 {False}
RHM_0 {True}
RLoc_1 {'cs'}
RHC_1 {False}
SWC_1 {True}        Step 1
MW_1 {False}
RHM_1 {True}
RLoc_2 {'cs'}
RHC_2 {True}
SWC_2 {True}        Setp 2
MW_2 {False}
RHM_2 {True}
RLoc_3 {'off'}
RHC_3 {True}
SWC_3 {True}        Step 3
MW_3 {False}
RHM_3 {True}
RLoc_4 {'off'}
RHC_4 {False}
SWC_4 {False}       Step 4 (goal)
MW_4 {False}
RHM_4 {True}
Move_0 {'mcc'}
PUC_0 {False}
DelC_0 {False}      Step 1 action
PUM_0 {False}
DelM_0 {False}
Move_1 {'nm'}
PUC_1 {True}
DelC_1 {False}      Step 2 action
PUM_1 {False}
DelM_1 {False}
Move_2 {'mc'}
PUC_2 {False}
DelC_2 {False}      Step 3 action
PUM_2 {False}
DelM_2 {False}
Move_3 {'nm'}
PUC_3 {False}
DelC_3 {True}       Step 4 action
PUM_3 {False}
DelM_3 {False}
```

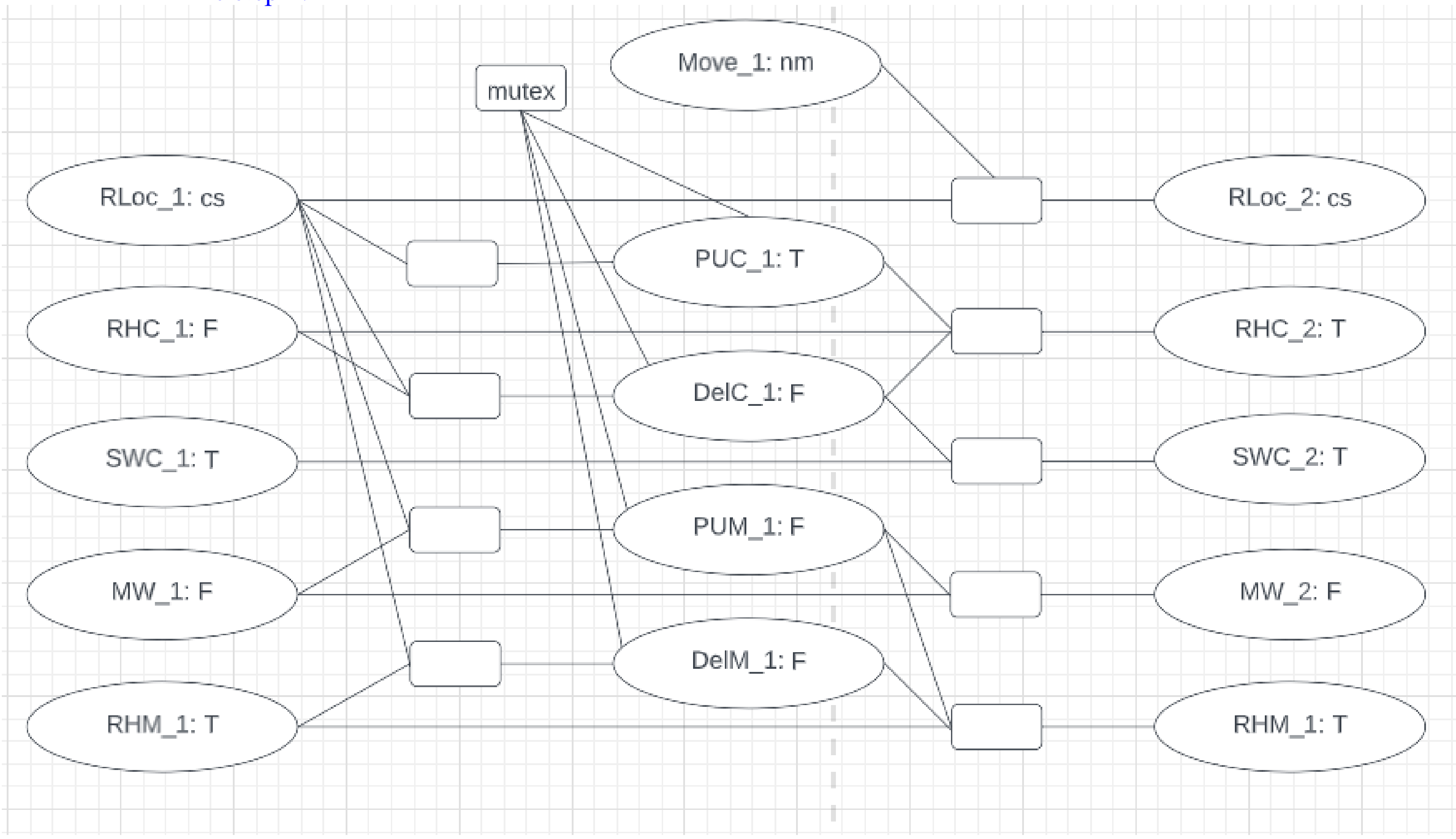
(d) Draw the constraint network of the solution that your program has found. Specify the variable assignments in your graph.

Graph is too wide, divided it into 4 pieces for 4 time steps

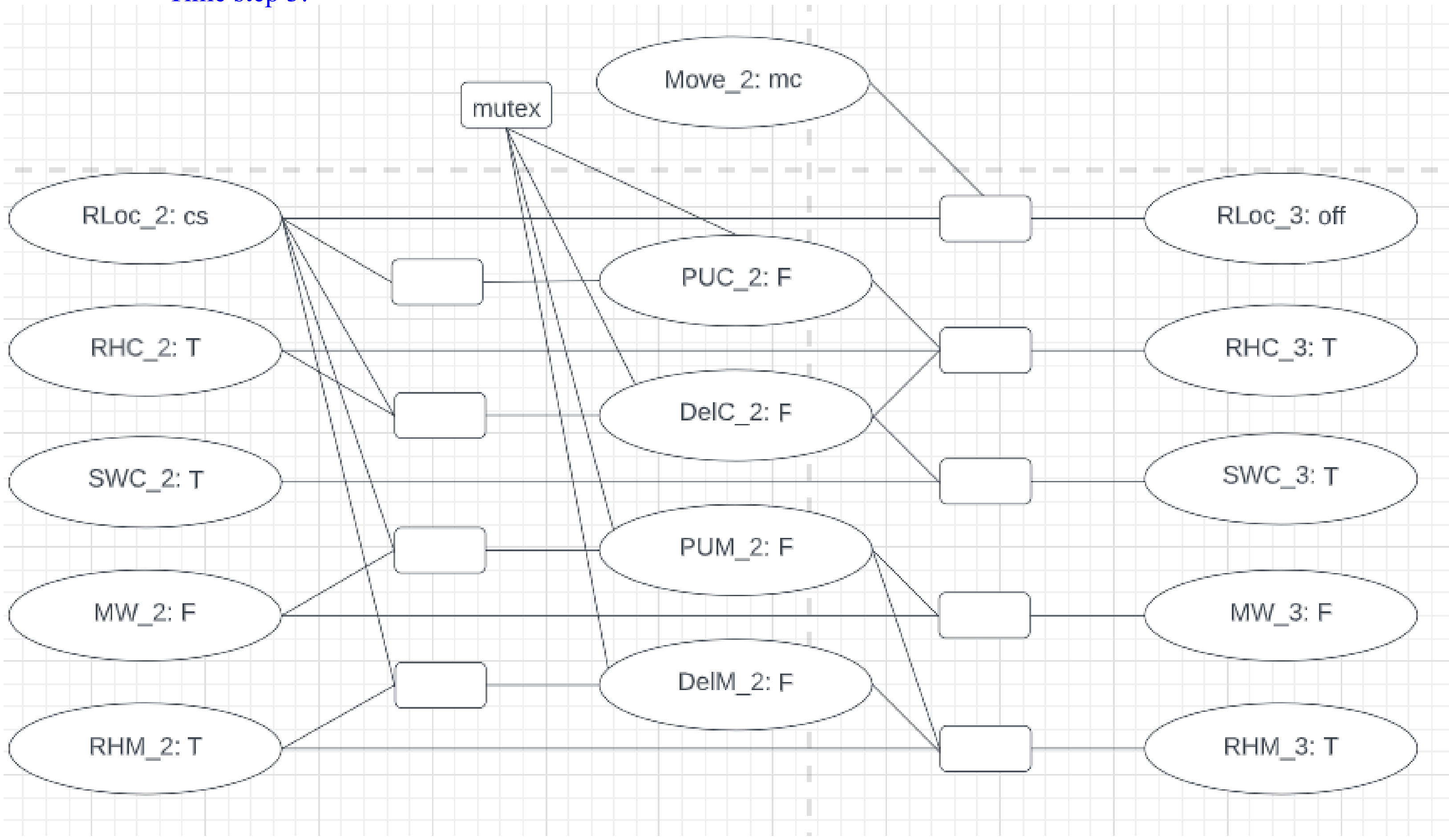
Time step 1:



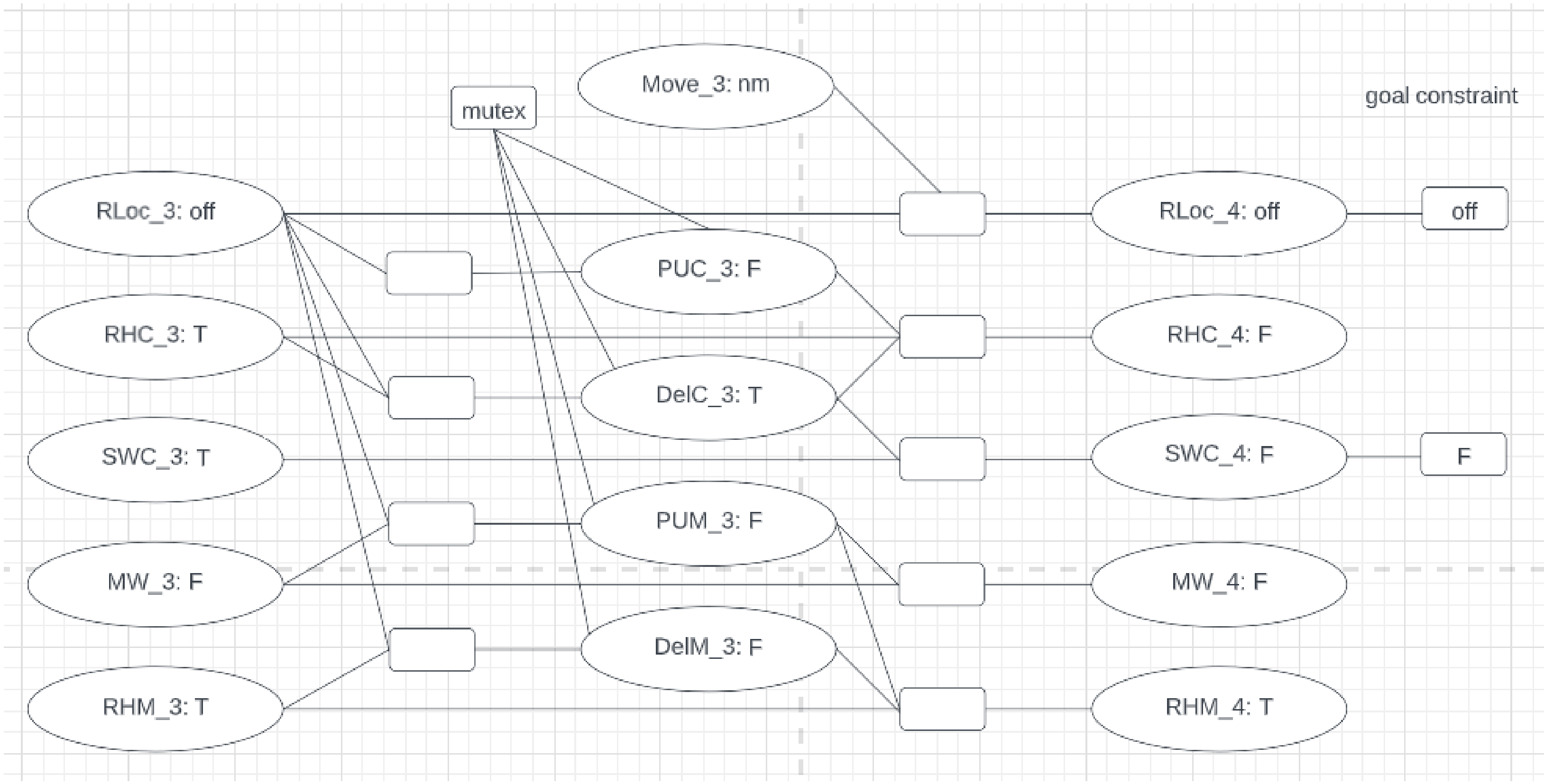
Time step 2:



Time step 3:



Time step 4:


Move_3: nm
mutex
goal constraint
RLoc_3: off
PUC_3: F
RLoc_4: off
off
RHC_3: T
RHC_4: F
DelC_3: T
SWC_3: T
SWC_4: F
F
PUM_3: F
MW_3: F
MW_4: F
DelM_3: F
RHM_3: T
RHM_4: T