National University of Singapore
School of Computing
CS3243 Introduction to AI

Important Instructions:

1. You CAN NOT start writing BEFORE 9 AM. Use the time until 9 AM to read the instructions carefully.
2. You have 120 minutes to attempt the exam. You must STOP writing at 11 AM. You may use the time between 11 AM and 11:20 AM only for scanning and uploading your solutions. You may use your phone for scanning only after you stop writing. You would not be allowed to write anything once you have used your phone for scanning.
3. You should name your submission file as <Student Number>.pdf. For example, if Student Number is A1234567Z, then the final name should be A1234567Z.pdf. The file must be submitted to the folder name: "Final Exam Submission" folder.
4. By 11:20 AM, there must be exactly one submitted file from your end. The submitted file must be a PDF file.
5. In case of emergency situation due to LumiNUS being unavailable, please email your solutions to dcsdlsw@comp.nus.edu.sg You must use this option only if LumiNUS is down else your marks may be deducted (the penalty will be decided on case by case basis).
6. The exam consists of FIVE questions. All questions carry equal weightage, i.e., 20 marks per question.
7. From our viewpoint, the questions are ordered in increasing order of difficulty but your experience may be different.
8. Write the following on the first page of your answer sheet:

   On my honor, I have neither given nor received any unauthorized aid on this exam.

9. Any violations will be reported to the highest levels in the university and the instructors will argue for the maximum penalty.
10. You are not allowed to have any other electronic devices (including mobile phone, tablet, notebook, smart watch, electronic dictionary, secondary computer monitor, etc.) within reach or sight or hearing from where you are sitting for the exam with the exceptions of the PC (desktop computer / laptop), the phone used for remote proctoring, and any approved calculator.

11. This is an OPEN book exam: you are allowed to refer to material on LumiNUS or any other handwritten/printed/electronic material in your possession locally. You are NOT allowed to use the internet other than LumiNUS. You are not allowed to consult with anyone.
12. Short answers are preferred over the long answers.
13. It is extremely unlikely that wrong answers will get partial credits.
14. If something is unclear, instead of asking for clarifications, it is advisable that you make a reasonable assumption and solve the question under those assumptions.
15. Have fun – do not stress out.

Question 1: Logical Agents

Part A

A country is said to be landlocked when it is not connected to the sea. In our world, there are only three countries: Arrakis, Middle-Earth, and Narnia. Suppose our agent is given the following premises

- P1. At least one of the three countries is landlocked.
- P2. If Arrakis is landlocked then at least one of Middle-Earth or Narnia is landlocked.
- P3. If Narnia is landlocked then so is Arrakis.
- P4. If Middle-Earth and Arrakis are both landlocked then so is Narnia.
- P5. If Middle-Earth is landlocked then at least one of Arrakis and Narnia are landlocked.
- P6. It is not the case that both Narnia and Middle-Earth are landlocked.

Use propositional logic to help our agent prove or disprove the following statement:

All countries are landlocked.

Part B

Suppose our all powerful rational agent knows the following statements about tennis stars Djokovic, Nadal, and Federer in the year 2023:

- P1: If Federer retires then at least one of the following happens
    1. Djokovic retires
    2. Nadal does not retire.
- P2: If Djokovic retires then at least one of the following happens:
    1. Federer does not retire
    2. Nadal retires.
- P3: If both Federer and Nadal don't retire then Djokovic does not retire either.
- P4: If Nadal retires then Federer retires.
- P5: At least one of the three stars does not retire.

Use propositional logic to help our agent prove or disprove the following statement:

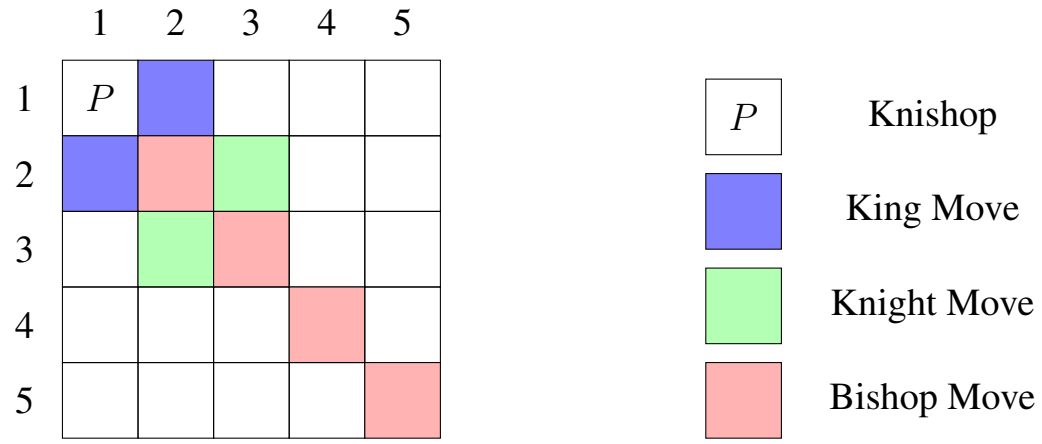
Neither Djokovic nor Nadal retires in 2023.

Question 2: Constraints

Instead of Queens, we are going to deal with a fictional Chess piece called Knishop. Knishop has the combined power of Bishop, Knight and King. You want to place 5 Knishops in a 5×5 gird such that:

1. No two Knishops attack each other, and
2. Every column has exactly one Knishop.

The figure below illustrates how a Knishop placed at P can attack at all the grid's color-filled positions.



The set of variables, denoted by SetOfVariables, is set to $\{p_1, p_2, p_3, p_4, p_5\}$, such that each p_i represents Knishop for i^{th} column. Note that the domain of all p_i is $\{1, 2, 3, 4, 5\}$.

The Algorithm 1 is the same as discussed in the class, except we have introduced a variable *counter* at line 2 and a loop that simply invokes the procedure ComputeDomain. The variable counter is introduced purely for the purpose of ensuring questions below are unambiguous. Recall, ComputeDomain(p_i, *assign*, *inference*) returns the set of values of p_i that p_i can take for the given *assign* and *inference*.

Assume that variables are assigned in the order p_1, p_2, p_3, p_4, p_5, i.e., variable p_1 will be assigned first, p_2 afterwards, and so on. Furthermore, the domain values of p_i are tried in the order $\langle 1, 2, 3, 4, 5 \rangle$, i.e., value 1 is tried before 2, and so on.

For the following scenarios, compute the value of $D(p_i)$ for each p_i when the value of the variable *counter* is 3 at line 3.

Part A: Algorithm 1 BacktrackingSearch(*prob*, *assign* $= \emptyset$, *counter*$= 0$) is invoked where only forward-checking is performed in the function Infer .

Part B: Algorithm 1 BacktrackingSearch(*prob*, *assign* $= \emptyset$, *counter*$= 0$) is invoked where full inference (as taught in the lecture) is performed in the function Infer.

(The Appendix contains additional information about the moves and also the Infer function as taught in the lectures)

Algorithm 1 BacktrackingSearch(*prob, assign,counter*)

```
1: if AllVariablesAssigned(prob, assign) then return assign
2: counter ← counter + 1
3: for p in SetOfVariables do
4:     D(p) ←ComputeDomain(p, assign, inference)
5: var ← PickUnassignedVar(prob, assign)
6: for value in OrderDomainValue(var,prob,assign) do
7:     if Val_Is_ConsistentWithAssignment(value,assign) then
8:         assign ← assign ∪ (var=value)
9:         inference ← Infer(prob, var, assign)
10:        assign ← assign ∪ inference
11:        if inference!=failure then
12:            result ← BacktrackingSearch(prob,assign,counter)
13:            if result ! = failure then return result
14:        assign ← assign \ {(var=value) ∪ inference}
15: return failure
```

Question 3: Bayesian Networks

You decide to create a Bayesian Network to distinguish between three professions; inspection, jockeying and kickboxing. You survey some inspectors, jockeys and kickboxers based on the following true and false questions:

1. S: Do you wear safety gear at work?
2. W: Do you work from home?
3. C: Do you like Coke more than Pepsi?
4. R: Do you play Racquetball?

Your survey of 10 people returned the following data set of answers to the above questions:

ID	S	W	C	R	Profession
1	Yes	Yes	No	No	Kickboxer
2	Yes	No	Yes	No	Kickboxer
3	No	No	Yes	Yes	Kickboxer
4	Yes	No	No	Yes	Jockey
5	No	Yes	No	Yes	Jockey
6	Yes	Yes	Yes	No	Jockey
7	Yes	No	No	Yes	Jockey
8	Yes	No	Yes	No	Inspector
9	No	No	No	Yes	Inspector
10	Yes	Yes	Yes	No	Inspector

Suppose we have two Bayesian network models M_A and M_B: Figure 1, and Figure 2.

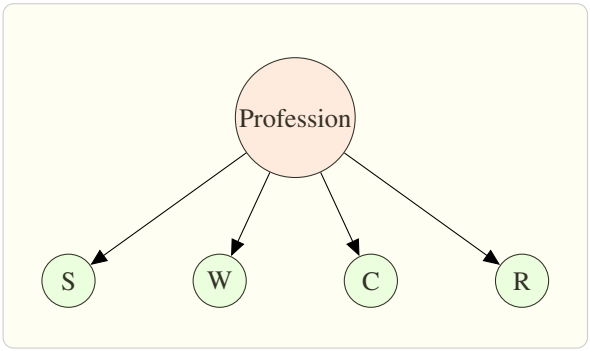


Figure 1: Model M_A

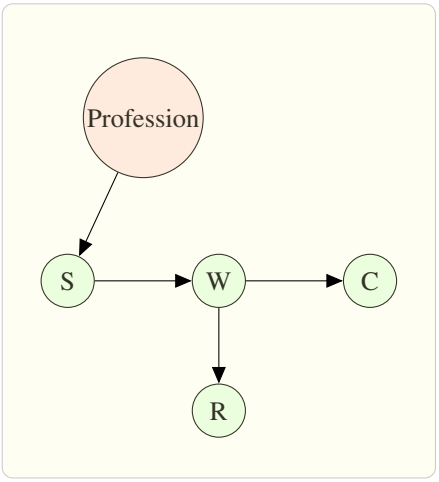


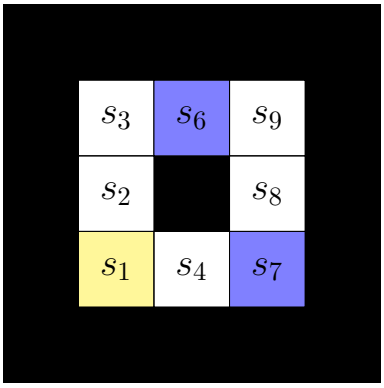
Figure 2: Model M_B

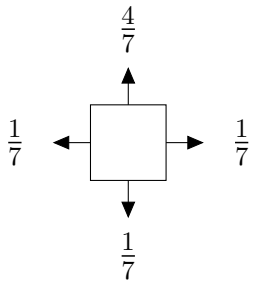
1. Fill in the CPT tables for each of these Bayesian networks.
2. Find out the probability of John being a kickboxer for both Bayesian network models M_A and M_B. He has the following characteristics:
   (a) He never works from home.
   (b) He does not like Coke over Pepsi.
   (c) He does not play Racquetball.

   Show your calculations.
3. Suppose John is a kickboxer. Then, which of the Bayesian network models would you choose to keep for future? Provide a rationale for your choice.

Question 4: Markov Decision Processes

In the following figure, we have a 3 by 3 grid world. The black colored squares are obstacles. We are initially at s_1, the yellow square in the diagram. States s_6 and s_7 are terminal states, colored in blue. The reward for non-terminal states is -1. The reward for s_6 is +4 and the reward for s_7 is -4. The discount factor γ is 0.5.





Action = Up

Following the same model as in the lectures, the set of actions for an agent is $A = \{$Up, Left, Right, Down $\}$. When the agent takes actions to move in a direction, the probability that the agent actually move in the intended direction is $\frac{4}{7}$, and the probability of moving in each of the other three directions is $\frac{1}{7}$. For example, if the agent takes the action "Up" diagram on the right, then there is a probability of $\frac{4}{7}$ that the agent indeed moves up, a probability of $\frac{1}{7}$ that the agent moves down, a probability of $\frac{1}{7}$ that the agent moves left and a probability of $\frac{1}{7}$ that the agent moves right.

Furthermore, if the agent moves in the direction of an obstacle, then the agent will remain at same state. For example, if the agent takes action "Left" at state s_2, then with probability $\frac{5}{7}$ the agent will end up at state s_2. (Observe that with probability $\frac{4}{7}$, the agent will move in left direction and with probability $\frac{1}{7}$, the agent will move right direction; and there is an obstacle in both the directions).

Given the above information, and following the notation in class, calculate $U(s)$ for every state s. You must show your working and provide rationale for your answer.

Question 5: Search

Suppose you want to find a simple path with maximum cost from source to goal. (Recall from CS1231, a simple path is a path without cycles, i.e., no vertex appears more than once.) To do so, let us assume you have modified the UCS algorithm as described in Algorithm 2, essentially, there is one change introduced in the standard UCS algorithm:

1. At line 13, $\hat{g}[v] = max(\hat{g}[v], \hat{g}[u] + c(u, v))$ instead of $\hat{g}[v] = min(\hat{g}[v], \hat{g}[u] + c(u, v))$.

(Do not worry about the algorithm description, you can simply assume that in UCS, we are using max instead of min for the update of $\hat{g}[v]$). Lets call the modified algorithm as UCS^{max}.

Algorithm 2 UCS^{max} Algorithm

```
1:  function FindPathToGoal(u)
2:      F(Frontier)← PriorityQueue(u)
3:      E(Explored)← ∅
4:      ĝ[u] ← 0
5:      while F is not empty do
6:          u ← F.pop()
7:          if GoalTest(u) then
8:              return path(u)
9:          E.add(u)
10:         for all children v of u do
11:             if v not in E then
12:                 if v in F then
13:                     ĝ[v] = max(ĝ[v], ĝ[u] + c(u, v))
14:                 else
15:                     F.push(v)
16:                     ĝ[v] = ĝ[u] + c(u, v)
17:     return Failure
```

Prove or Disprove the following statement:

> The algorithm UCS^{max} will always return the simple path with maximum cost from source to goal if such a path exists.

You can assume all the conditions under which we proved UCS to be optimal, i.e., the graph is finite and for some $\varepsilon > 0$, every edge has a cost $\geq \varepsilon$.

END OF THE QUESTIONS
Additional Optional Information on the next page about Question 1

Appendix: Additional Information

Question1:Constraints

Algorithm 3 Infer*(prob,var,assign)* function of Algorithm 1

```
1: inference ← ∅
2: varQueue ← [var]
3: while varQueue is not empty do
4:     y ← varQueue.pop()
5:     for each constraint C in prob where y ∈ Vars(C) do
6:         for all x ∈ Vars(C) \ y do
7:             S ← ComputeDomain(x, assign, inference)
8:             for each value v in S do
9:                 if no valid value exists for all var ∈ Var(C) \ x s.t. C[x ⊢ v] is satisfied then
10:                    inference ← inference ∪ (x ∉ {v})
11:            T ← ComputeDomain(x, assign, inference)
12:            if T= ∅ then return failure
13:            if S ≠ T then
14:                varQueue.add(x)
15: return inference
```

Moves of Different Pieces of Chess

- A bishop moves any number of squares diagonally.
- The king moves exactly one square horizontally, vertically, or diagonally.
- A knight moves two squares horizontally then one square vertically, or one square horizontally then two squares vertically, that is in an "L" pattern.

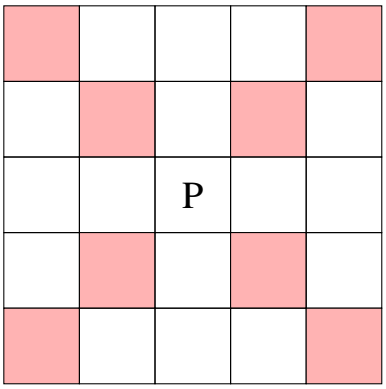


Figure 3: Moves of a Bishop

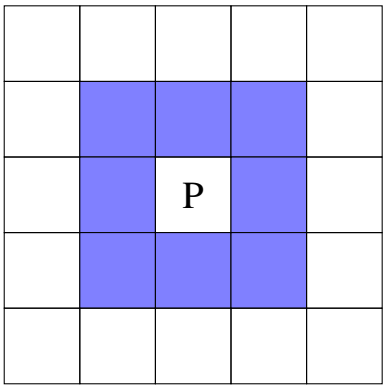


Figure 4: Moves of the King

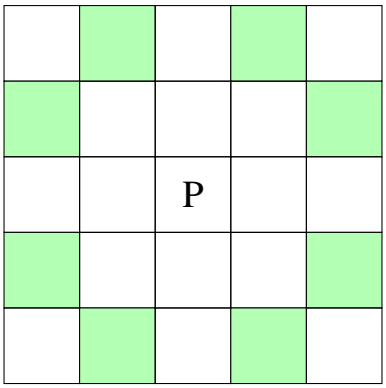


Figure 5: Moves of a Knight