NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

FINAL ASSESSMENT FOR
Semester 2 AY2023/2024

CS3243: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

SOLUTIONS

April 30, 2024 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. Please write your Student Number only. Do not write your name.
2. This assessment contains FIVE (5) questions and comprises TWENTY (20) pages, including blank pages. All the questions are worth a total of 60 MARKS. It is set for a total duration of 120 MINUTES. You are to complete all 5 questions.
3. This is a CLOSED BOOK assessment. However, you may reference a SINGLE DOUBLE-SIDED A4 CHEAT SHEET.
4. You are allowed to use NUS APPROVED CALCULATORS.
5. If something is unclear, solve the question under reasonable assumptions. State your assumptions clearly in the answer. If you must seek clarification, the invigilators will only answer questions with Yes/No/No Comment answers.
6. You may not communicate with anyone other than the invigilators in any way.

STUDENT NUMBER: | **A** | | | | | | | | |

EXAMINER'S USE ONLY		
Question	Mark	Score
1	10	
2	10	
3	12	
4	14	
5	14	
TOTAL	60	

1a. Given an unsorted list, $\boldsymbol{L}_0$, we may perform a **local search** via the **hill-climbing** algorithm to sort $\boldsymbol{L}_0$ in ascending order.

Assume that $\boldsymbol{L}_0$ contains n unique elements. Further, assume that you are given the heuristic function ***eval***(..) and the actions function ***actions***(..), specified in each of the following parts.

In each case, ***determine the worst-case time complexity*** (i.e., the number of iterations to terminate), ***and space complexity*** (i.e., the number of neighbours per state) that is required for the hill-climbing algorithm to terminate.

You are also to assume that the hill-climbing algorithm:

- seeks to ***maximise*** the heuristic (i.e., ***eval***(..)),
- ***terminates*** when the list is sorted, and
- ***restarts*** whenever a plateau is reached.

(i) [2.0 marks] Given:

- $\boldsymbol{eval}(\boldsymbol{L}_i) \rightarrow \begin{cases} 1 & \text{if } \boldsymbol{L}_i \text{ is sorted} \\ 0 & \text{otherwise} \end{cases}$
- $\boldsymbol{actions}(\boldsymbol{L}_i) \rightarrow$ permutations of all the elements in $\boldsymbol{L}_i$

Option **A**: $\boldsymbol{O}(1)$ time complexity and $\boldsymbol{O}(1)$ space complexity.
Option **B**: $\boldsymbol{O}(1)$ time complexity and $\boldsymbol{O}(n!)$ space complexity.
Option **C**: $\boldsymbol{O}(n!)$ time complexity and $\boldsymbol{O}(1)$ space complexity.
Option **D**: $\boldsymbol{O}(n!)$ time complexity and $\boldsymbol{O}(n!)$ space complexity.
Option **E**: None of the above.

Solution: B /* Each action generates all permutations of the list; one of them is guaranteed to be sorted, hence only 1 iteration is required to terminate. However, there are $n!$ permutations of the list that are generated. */

(ii) [2.0 marks] Given:

- $\boldsymbol{eval}(\boldsymbol{L}_i) \rightarrow$ the number of indices, $j \in [0, |\boldsymbol{L}_i|]$, where $\boldsymbol{L}_i[j] < \boldsymbol{L}_i[j+1]$
- $\boldsymbol{actions}(\boldsymbol{L}_i) \rightarrow$ permutations of $\boldsymbol{L}_i$ where exactly 2 elements are swapped

Option **A**: $\boldsymbol{O}(1)$ time complexity and $\boldsymbol{O}(n^2)$ space complexity.
Option **B**: $\boldsymbol{O}(n)$ time complexity and $\boldsymbol{O}(n^2)$ space complexity.
Option **C**: $\boldsymbol{O}(n^2)$ time complexity and $\boldsymbol{O}(1)$ space complexity.
Option **D**: Does not terminate in the worst case; has $\boldsymbol{O}(n^2)$ space complexity.
Option **E**: None of the above.

Solution: D /* Consider the case where $\boldsymbol{L}_i = [3,4,1,2]$, where $\boldsymbol{eval}(\boldsymbol{L}_i) = 2$. Here, no single swap would yield a higher ***eval***(..) value – i.e., a plateau. So, the algorithm would not terminate. */

(iii) [2.0 marks] Given:

- ***eval***(***L***$_i$) → k, the length of the list slice, ***L***$_i$[0: k], where ***L***$_i$[0: k] corresponds to the longest possible list slice that is sorted in ascending order
- ***actions***(***L***$_i$) → find the first element, v = ***L***$_i$[j], where ***L***$_i$[j] < ***L***$_i$[j–1], then pop v and reinsert into ***L***$_i$ at any valid index (i.e., between 0 and |***L***$_i$|), keeping the relative order of the remaining elements in ***L***$_i$ the same

Option **A**: ***O***(n) time complexity and ***O***(n) space complexity.
Option **B**: ***O***(n) time complexity and ***O***(n^2) space complexity.
Option **C**: ***O***(n^2) time complexity and ***O***(n) space complexity.
Option **D**: ***O***(n^2) time complexity and ***O***(n^2) space complexity.
Option **E**: None of the above.

Solution: **A** /* In the worst case, the algorithm places one more element in sorted order per iteration, which requires n iterations. As the algorithm finds the first element not in sorted order and places it at ***any*** valid index, there are n possible target indices. */

1b. [4.0 marks] Suppose we are given a string, ***S***, and our goal is to find the shortest sequence of transformations that, when applied to ***S***, would bring us to a given target string, ***G***.

Assume that the only possible transformation is to correctly replace the character at a given index in ***S*** (i.e., to replace ***S***[i] such that ***S***[i] = ***G***[i]).

For simplicity, suppose that in all cases, |***S***| = |***G***|.

Assume that this problem is to be solved using the **A* search** algorithm, adopting the **Levenshtein distance** heuristic.

The Levenshtein distance between two strings, ***A*** and ***B*** (of length |***A***| and |***B***| respectively), is given by ***lev***(***A***, ***B***), which is defined as follows.

$$\textbf{\textit{lev}}(\textbf{\textit{A}}, \textbf{\textit{B}}) = \begin{cases} |\textbf{\textit{A}}| & \text{if } |\textbf{\textit{B}}| = 0, \\ |\textbf{\textit{B}}| & \text{if } |\textbf{\textit{A}}| = 0, \\ \textbf{\textit{lev}}(\textbf{\textit{tail}}(\textbf{\textit{A}}), \textbf{\textit{tail}}(\textbf{\textit{B}})) & \text{if } \textbf{\textit{head}}(\textbf{\textit{A}}) = \textbf{\textit{head}}(\textbf{\textit{B}}), \\ 1 + \textbf{\textit{min}} \begin{cases} \textbf{\textit{lev}}(\textbf{\textit{tail}}(\textbf{\textit{A}}), \textbf{\textit{B}}) \\ \textbf{\textit{lev}}(\textbf{\textit{A}}, \textbf{\textit{tail}}(\textbf{\textit{B}})) \\ \textbf{\textit{lev}}(\textbf{\textit{tail}}(\textbf{\textit{A}}), \textbf{\textit{tail}}(\textbf{\textit{B}})) \end{cases} & \text{otherwise} \end{cases}$$

Note:
- ***tail***(***X***): ***X***[1:], all but first char.
- ***head***(***X***): ***X***[0], only first char.

As the Levenshtein distance is a proper distance metric, it obeys the following rules.
- For all strings ***A***, ***lev***(***A***, ***A***) = 0.
- For all strings ***A***, ***B***, such that ***A*** != ***B***, ***lev***(***A***, ***B***) > 0 (i.e., **positivity**).
- For all strings ***A***, ***B***, ***lev***(***A***, ***B***) = ***lev***(***B***, ***A***) (i.e., **symmetry**).
- For all strings ***A***, ***B***, ***C***, ***lev***(***A***, ***B***) + ***lev***(***B***, ***C***) ≥ ***lev***(***A***, ***C***) (i.e., **triangle inequality**).

Prove or disprove the following statement.

*The Levenshtein distance is a **consistent heuristic** for this problem.*

Solution: With this problem, note that all action costs are 1. Thus, for the heuristic, ***lev***(..) to be a consistent, we require that ***lev***(X, G) ≤ ***lev***(Y, G) + 1, where X is a parent string, Y is a child string.

- By triangle inequality, we have ***lev***(X, Y) + ***lev***(Y, G) ≥ ***lev***(X, G), or rather, ***lev***(X, G) ≤ ***lev***(X, Y) + ***lev***(Y, G). Since, for consistency, we must show that ***lev***(X, G) ≤ ***lev***(Y, G) + 1, we must therefore show that ***lev***(X, G) ≤ ***lev***(X, Y) + ***lev***(Y, G) ≤ 1 + ***lev***(Y, G), or rather, that ***lev***(X, Y) ≤ 1.

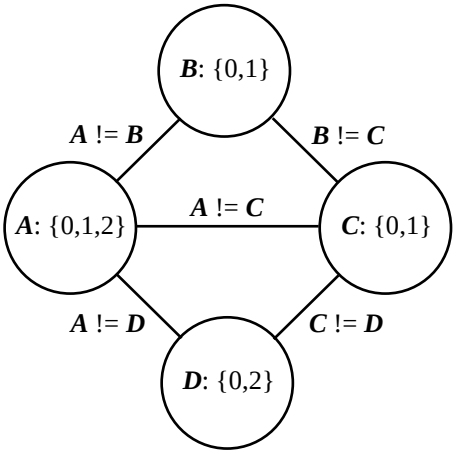
Let $n = |X|$ and suppose that for any given action, we will replace $X[i]$ with the correct character $G[i]$. This may be described as follows:

- $X = X[0{:}i] + X[i] + X[i{+}1{:}n]$ // where $X[a{:}b]$ refers to a slice from index a to b–1
- $Y = X[0{:}i] + G[i] + X[i{+}1{:}n]$ // and where $i \in [0, n) \wedge i \in \mathbb{Z}$

There are two cases that must be considered:

- Case 1: $X[i] = G[i]$ // notice that this IS a possible action
  - Here, $X = Y$, and thus have ***lev***(X, Y) = ***lev***(X, X) = 0, which is less than 1 as desired.
- Case 2: $X[i] \neq G[i]$
  - Here, ***lev***(X, Y) = ***lev***($X[0{:}i] + X[i] + X[i{+}1{:}n]$, $X[0{:}i] + G[i] + X[i{+}1{:}n]$).
  - Applying the definition of Levenshtein distance, we thus have:
    - As the first i characters of X and Y are the same, we repeatedly apply the recursive case where ***head***(A) = ***head***(B), i times, and get:
      lev(X,Y) = ***lev***(***tail***(A), ***tail***(B)) = ***lev***($X[i]{+}X[i{+}1{:}n]$, $G[i]{+}X[i{+}1{:}n]$)
    - Note that $|X[i] + X[i{+}1{:}n]| = |G[i] + X[i{+}1{:}n]| = n - i > 0$ since $i \in [0, n{-}1]$. This implies that the two base cases, $|A| = |B| = 0$ do not apply. Also, since under Case 2, we assert $X[i] \neq G[i]$, the recursive case where ***head***(A) = ***head***(B) also does not apply.
    - We must apply complex *otherwise* recursive case. So, we have:
      - ***lev***($X[i]{+}X[i{+}1{:}n]$, $G[i]{+}X[i{+}1{:}n]$) = 1 + ***min***(P, Q, R), where:
        P = ***lev***($X[i{+}1{:}n]$, $G[i]{+}X[i{+}1{:}n]$) // ***lev***(***tail***(A), B) case
        Q = ***lev***($X[i]{+}X[i{+}1{:}n]$, $X[i{+}1{:}n]$) // ***lev***(A, ***tail***(B)) case
        R = ***lev***($X[i{+}1{:}n]$, $X[i{+}1{:}n]$) // ***lev***(***tail***(A), ***tail***(B)) case
      - This is: 1 + ***min***(***lev***(S, $G[i]{+}S$), ***lev***($X[i]{+}S$, S), ***lev***(S, S)), where $S = X[i{+}1{:}n]$.
  - Since by definition, ***lev***(K, K) = 0 and ***lev***(K, J) > 0 for all strings K, J where $K \neq J$, we can conclude here that ***lev***(X,Y) ≥ 0 for all strings.
  - Thus, we have P, $Q \geq 0$, while $R = 0$, and therefore ***min***(P, Q, R) = 0.
  - Finally, we have ***lev***($X[i]{+}X[i{+}1{:}n]$, $G[i]{+}X[i{+}1{:}n]$) = 1 + ***min***(P, Q, R) = 1, which is less than or equal 1 as desired.

2a. [3.0 marks] Trace the execution of the **backtracking** algorithm to solve the following *Constraint Satisfaction Problem* (CSP).



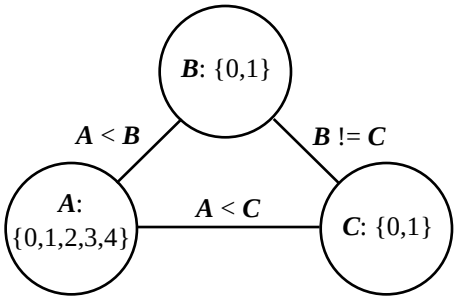
For *variable order*, you are to assume the **Minimum Remaining Values** (**MRV** – i.e., most restricted variable) heuristic, followed by the **degree** heuristic (i.e., as tiebreaker for MRV), and finally, followed by an **alphabetical order** tiebreaker. For value order, you are to assume the ordering specified within each domain value set given below. Further, you must apply **forward checking** as inference.

Complete the given table in the solution box to reflect your trace. ***Clearly show each variable assignment and every backtrack made***. Note that you do not have to show how the domain changes after each iteration. Do also note that you need ***not*** fill every row of the table.

Solution:

Assign or **Unassign** the value, *v*, to the variable, *x* (circle the one that applies)	**Variable** (*x*)	**Value** (*v*)
(Assign) / Unassign	***C***	0
(Assign) / Unassign	***B***	1
(Assign) / Unassign	***A***	2
Assign / (Unassign)	***A***	2
Assign / (Unassign)	***B***	1
Assign / (Unassign)	***C***	0
(Assign) / Unassign	***C***	1
(Assign) / Unassign	***B***	0
(Assign) / Unassign	***A***	2
(Assign) / Unassign	***D***	0
Assign / Unassign		
Assign / Unassign		

2b [4.0 marks] Trace the execution of the **AC-3** algorithm on the following CSP. Assume that the arcs are enqueued in the order: (***B***, ***C***), (***A***, ***B***), (***B***, ***A***), (***A***, ***C***), (***C***, ***A***), (***C***, ***B***).



Complete the given table in the solution box to reflect your trace. ***Clearly show the arc popped, the updated domains to achieve arc consistency, and the resultant queue***. Note that you need **not** fill every row of the table.

Solution:

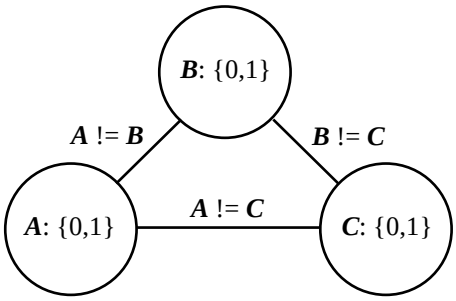
Arc Popped (X_i, X_j)	**Domain** of Xi (after updating for arc consistency)	**Resultant Queue**
(***B***, ***C***)	***B***: {0,1}	(***A***, ***B***), (***B***, ***A***), (***A***, ***C***), (***C***, ***A***), (***C***, ***B***)
(***A***, ***B***)	***A***: {0}	(***B***, ***A***), (***A***, ***C***), (***C***, ***A***), (***C***, ***B***)
(***B***, ***A***)	***B***: {1}	(***A***, ***C***), (***C***, ***A***), (***C***, ***B***)
(***A***, ***C***)	***A***: {0}	(***C***, ***A***), (***C***, ***B***)
(***C***, ***A***)	***C***: {1}	(***C***, ***B***), (***B***, ***C***)
(***C***, ***B***)	***C***: {}	AC-3 terminates as ***C*** reduced to ∅

2c [3.0 marks] Prove or disprove the following statement.

If the AC-3 algorithm is run (once) as a preprocessing step, and all the variables have one or more values in the domain after the execution of the AC-3 algorithm, then the CSP (i.e., the given set of variables, domains, and binary constraints between variables) has at least one solution.

Solution: False.

Consider the following counterexample.



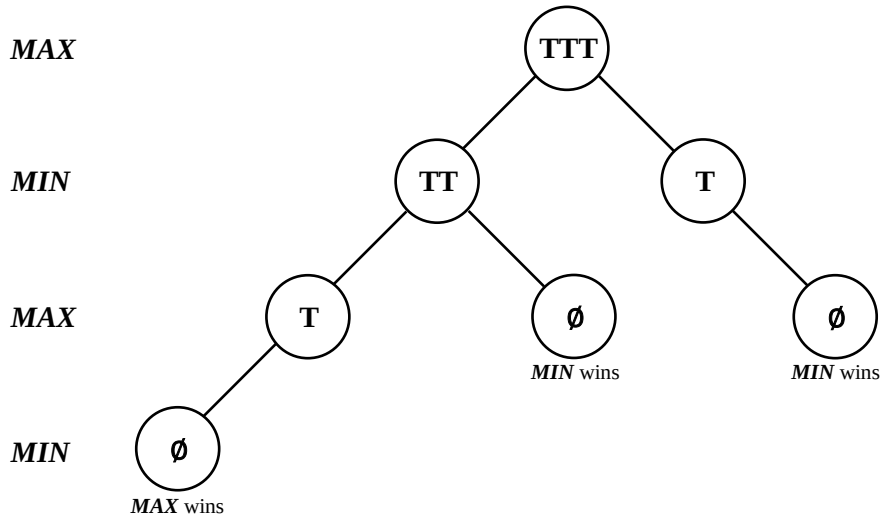
In this example, all arcs are consistent. However, this CSP has no solution.

3. The game of NIM has many different variants. In this version of NIM, *n* tokens are placed in a common pool. *Each player takes turns in removing either **one** or **two** tokens.* The first player that is unable to make a move loses. Assume that the ***MAX*** player goes first.

For each part in this question, assume that we will use the **minimax** algorithm and **alpha-beta pruning** algorithm to analyse a simplified version of the game.

(i) [2.0 marks] Draw the tree for the game containing **three** tokens. To represent a state, you must use the notation **T** to represent a state with one token, **TT** to represent a state with two tokens, and so on. To represent a state with no tokens remaining, use **Ø**.

Solution:



(ii) [1.0 marks] Hence or otherwise, determine which player (i.e., MAX or MIN) has a winning strategy in the initial position.

Solution: The ***MIN*** player has a winning strategy.

(iii) [2.0 marks] To model a ***minimax*** problem, one is required to assign a numerical value to each leaf node. This is usually done through an evaluation function that takes in a state and returns a utility value (i.e., in this case, an integer). Define the ***state representation*** and ***evaluation function*** for the above game. Your evaluation function must be a function that takes in a ***terminal state***. Hence, if a given state is non-terminal, the evaluation function must return 0. Otherwise, if the given state is terminal, the evaluation function must return +1 if the state represents a win for ***MAX***, and -1 if it represents a win for ***MIN***.

Solution:

- State representation:
  - A tuple: (*tokens*, *player*) // representing (tokens left, player's turn)
- Evaluation function:
  - ***eval***(*state*) = $\begin{cases} 0 & \text{if } state[0] \text{ != } 0 \\ +1 & \text{if } state[0] == 0 \wedge state[1] == \textbf{\textit{MIN}} \\ -1 & \text{if } state[0] == 0 \wedge state[1] == \textbf{\textit{MAX}} \end{cases}$

The main idea is to realise that a state must also contain turn information; just using the number of tokens remaining as a state representation is not descriptive enough to give unambiguous evaluations.

For the following parts of this question (i.e., parts **(iv)** to **(vi)**), we examine the **alpha-beta pruning** algorithm on a modified version of the aforementioned game.

The game is modified such that ***if no tokens have been removed by ANY player***, then ***either player*** may then opt to ***pass their turn*** (i.e., ***if ANY player removes a token***, ***neither player can pass their turn***). Hence, if a player removes no tokens, they may pass their turn, after which, the game proceeds to the opponent's turn. Note that this allows players to pass their turn repeatedly at the initial state, which leads to an infinite subtree under certain branches.

(iv) [1.0 marks] Show that with an initial state with **three** tokens, there exists some move ordering such that the **alpha-beta pruning** algorithm does not terminate.

Solution: Suppose the move order is such that the first action searched is always the action that skips a turn. Then the players will always skip their turn and the leaf node is never reached. With such a case, no pruning is possible.

(v) [3.0 marks] Determine if the following statement is True, or False. Justify your answer.

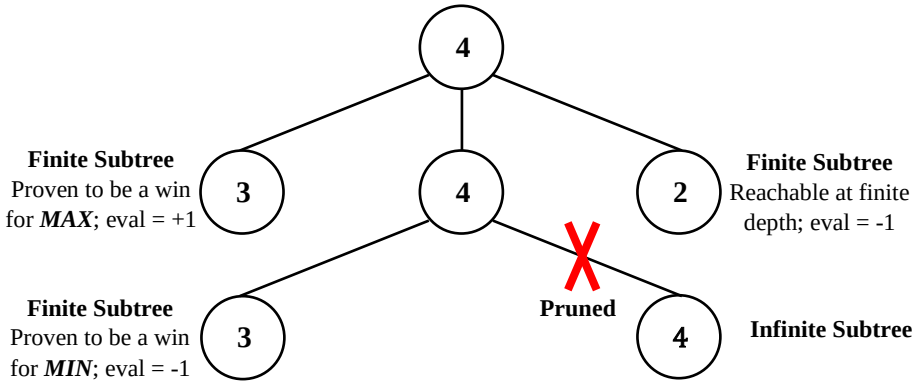
In this modified version of the game (with the option to pass), there exists no move ordering that allows the alpha-beta pruning algorithm to terminate, assuming an initial state with three tokens.

Solution: **True**. The given initial state represents a loss for ***MAX*** if ***MAX*** removes a token, or else, it represents an uncertain evaluation if ***MAX*** instead passes the turn. Assuming ***MAX*** passes, the resultant state represents a loss for ***MIN*** if ***MIN*** removes a token, or else, it represents an uncertain evaluation if ***MIN*** instead passes the turn.

The **alpha-beta pruning** algorithm will never be able to prune in the given game tree as the actions that potentially lead to winning states for either player are always the actions that pass a turn.
/* The optimal move for ***MAX*** in the starting state would be to pass since it avoids the forced loss (i.e., utility -1) and allows ***MAX*** to, potentially, obtain a better evaluation (i.e., utility +1, if the opponent takes a token). Using the same reasoning, the optimal move for ***MIN*** would also be to pass the turn, in which case we are back to the starting state. This is an infinite cycle. As alpha-beta pruning will always find the optimal move, the infinite sequence of turn passing will always be explored, and thus, never be pruned. Hence, alpha-beta pruning will not be able to terminate in finite time. */

(vi) [3.0 marks] Assume a game (once again, which ***includes the option to pass***) where the initial state contains **four** tokens. Show that for such a game, there exists some move ordering such that the **alpha-beta pruning** algorithm will terminate.

Solution:

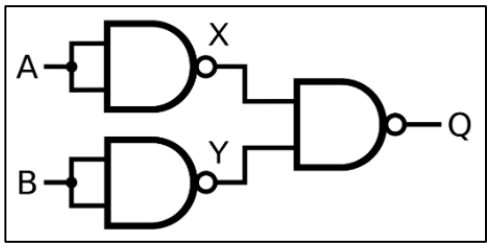


The main idea is that ***MAX*** has a winning strategy by taking one token in the given game. If ***MAX*** passes, ***MIN*** now has a winning strategy. With the correct move ordering, ***MAX*** can determine that skipping the turn would result in a loss without having to explore the infinite subtree of passing a turn. Since it is determined that the first player loses by skipping his turn, the infinite subtree can be pruned off, hence alpha-beta pruning terminates in finite time.

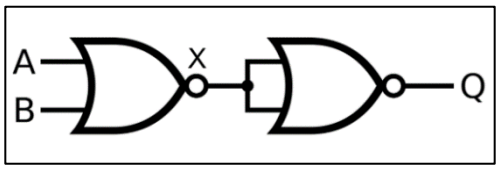
4a. You are a chief designer in a semiconductor company that has been making the following digital circuits for years now.

In this circuit (***Circuit* 1** – depicted on the right), three ***NAND*** gates are used. It is represented as follows:

- ***NAND***(A, A) = X
- ***NAND***(B, B) = Y
- ***NAND***(X, Y) = Q



One day, an intern that you personally hired suggests that it is possible to make the same digital circuit using fewer logic gates. The intern suggests that ***NOR*** gates be used instead of ***NAND*** gates.



In this circuit (***Circuit* 2** – depicted on the left), two ***NOR*** gates are used. It is represented as follows:

- ***NOR***(A, B) = X
- ***NOR***(X, X) = Q

As the chief designer, you would like to prove (or disprove) the intern's claim.

(i) [2.0 marks] Use **Model Checking** to prove or disprove the claim. You may label additional points in the circuits and use these points in your table.

Solution:

***Circuit* 1**

A	B	X	Y	Q					
T	T	F	F	T					
T	F	F	T	T					
F	T	T	F	T					
F	F	T	T	F					

***Circuit* 2**

A	B	X	Q						
T	T	F	T						
T	F	F	T						
F	T	F	T						
F	F	T	F						

Comparing the truth tables, we see that the models for both circuits are the same.

(ii) [5.0 marks] Populate a **knowledge base** (**KB**) with the rules that describe ***Circuit* 1** and ***Circuit* 2**. Make sure that each rule is in **Conjunctive Normal Form** (**CNF**). You may suffix the variables with 1 or 2 to reference ***Circuit* 1** and ***Circuit* 2**.

Solution:

The KB includes the following rules (you must extend upon the two rules defined).

- $\boldsymbol{R}_{01}$: $A_1 \vee X_1$
- $\boldsymbol{R}_{02}$: $\neg A_1 \vee \neg X_1$
- $\boldsymbol{R}_{03}$: $B_1 \vee Y_1$
- $\boldsymbol{R}_{04}$: $\neg B_1 \vee \neg Y_1$
- $\boldsymbol{R}_{05}$: $X_1 \vee Q_1$
- $\boldsymbol{R}_{06}$: $Y_1 \vee Q_1$
- $\boldsymbol{R}_{07}$: $\neg X_1 \vee \neg Y_1 \vee \neg Q_1$
- $\boldsymbol{R}_{08}$: $A_2 \vee B_2 \vee X_2$
- $\boldsymbol{R}_{09}$: $\neg A_2 \vee \neg X_2$
- $\boldsymbol{R}_{10}$: $\neg B_2 \vee \neg X_2$
- $\boldsymbol{R}_{11}$: $X_2 \vee Q_2$
- $\boldsymbol{R}_{12}$: $\neg X_2 \vee \neg Q_2$

/* *Circuit* 1

- ***NAND***(A, A) → X
- ≡ ***AND***(A, A) ∨ X
- ≡ (A ∨ X) ∧ (A ∨ X)
- ≡ A ∨ X

- X → ***NAND***(A, A)
- ≡ ¬X ∨ ***NAND***(A, A)
- ≡ ¬X ∨ ¬(A ∧ A)
- ≡ ¬X ∨ ¬A ∨ ¬A
- ≡ ¬X ∨ ¬A

- ***NAND***(B, B) → Y
- ≡ B ∨ Y

- Y → ***NAND***(B, B)
- ≡ ¬Y ∨ ¬B

- ***NAND***(X, Y) → Q
- ≡ (X ∨ Q) ∧ (Y ∨ Q)

- Q → ***NAND***(X, Y)
- ≡ ¬Q ∨ ¬(X ∧ Y)
- ≡ ¬Q ∨ ¬X ∨ ¬Y

***/**

/* *Circuit* 2

- ***NOR***(A, B) → X
- ≡ A ∨ B ∨ X

- X → ***NOR***(A, B)
- ≡ ¬X ∨ ¬(A ∨ B)
- ≡ ¬X ∨ (¬A ∧ ¬B)
- ≡ (¬X ∨ ¬A) ∧ (¬X ∨ ¬B)

- ***NOR***(X, X) → Q
- ≡ X ∨ Q

- Q → ***NOR***(X, X)
- ≡ ¬X ∨ ¬Q

***/**

(iii) [3.0 marks] Recall that our objective is to show that ***Circuit* 1** and ***Circuit* 2** are equivalent. State the **query** to the **KB** that would prove this. You may use the following Boolean functions: ***AND***, ***OR***, ***NOT***, ***XOR***, and ***XNOR***.

Solution:

We wish to show that "$(A_1 = A_2 \wedge B_1 = B_2) \Rightarrow Q_1 = Q_2$".

Thus, the query, α, is:

$((A_1 \Leftrightarrow A_2) \wedge (B_1 \Leftrightarrow B_2)) \Rightarrow (Q_1 \Leftrightarrow Q_2)$

$\equiv (\boldsymbol{XNOR}(A_1, A_2) \wedge \boldsymbol{XNOR}(B_1, B_2)) \Rightarrow \boldsymbol{XNOR}(Q_1, Q_2)$ // $\boldsymbol{IFF} \equiv \boldsymbol{XNOR}$

$\equiv \neg(\boldsymbol{XNOR}(A_1, A_2) \wedge \boldsymbol{XNOR}(B_1, B_2)) \vee \boldsymbol{XNOR}(Q_1, Q_2)$ // $A \Rightarrow B \equiv \neg A \vee B$

$\equiv \boldsymbol{XOR}(A_1, A_2) \vee \boldsymbol{XOR}(B_1, B_2) \vee \boldsymbol{XNOR}(Q_1, Q_2)$ // De Morgan's

4b. The following is a **knowledge base** (**KB**) given to an agent.

- $\boldsymbol{R}_1$: swimming $\Rightarrow$ wet
- $\boldsymbol{R}_2$: (rain $\wedge$ outside) $\Rightarrow$ wet
- $\boldsymbol{R}_3$: (warm $\wedge \neg$rain) $\Rightarrow$ pleasant
- $\boldsymbol{R}_4$: $\neg$wet
- $\boldsymbol{R}_5$: outside
- $\boldsymbol{R}_6$: warm

(i) [2.0 marks] Express the above rules in **Conjunctive Normal Form** (**CNF**).

Solution:

KB (in **CNF**):

- $\boldsymbol{R}_1$: $\neg$swimming $\vee$ wet
- $\boldsymbol{R}_2$: $\neg$rain $\vee \neg$outside $\vee$ wet
- $\boldsymbol{R}_3$: $\neg$warm $\vee$ rain $\vee$ pleasant
- $\boldsymbol{R}_4$: $\neg$wet
- $\boldsymbol{R}_5$: outside
- $\boldsymbol{R}_6$: warm

(ii) [2.0 marks] Use **resolution** to prove that "pleasant" is true.

Solution:

KB (in **CNF**):

- $\boldsymbol{R}_{01}$: ¬swimming ∨ wet
- $\boldsymbol{R}_{02}$: ¬rain ∨ ¬outside ∨ wet
- $\boldsymbol{R}_{03}$: ¬warm ∨ rain ∨ pleasant
- $\boldsymbol{R}_{04}$: ¬wet
- $\boldsymbol{R}_{05}$: outside
- $\boldsymbol{R}_{06}$: warm

We are given the **query**, α: pleasant. Add ¬α to the **KB** and prove **unsatisfiable**.

- $\boldsymbol{R}_{07}$: ¬pleasant

The **resolution** is as follows.

- $\boldsymbol{R}_{08}$: ¬warm ∨ rain ($\boldsymbol{R}_{03}$, $\boldsymbol{R}_{07}$)
- $\boldsymbol{R}_{09}$: rain ($\boldsymbol{R}_{06}$, $\boldsymbol{R}_{08}$)
- $\boldsymbol{R}_{10}$: ¬outside ∨ wet ($\boldsymbol{R}_{02}$, $\boldsymbol{R}_{09}$)
- $\boldsymbol{R}_{11}$: wet ($\boldsymbol{R}_{05}$, $\boldsymbol{R}_{10}$)
- $\boldsymbol{R}_{12}$: ∅ ($\boldsymbol{R}_{04}$, $\boldsymbol{R}_{11}$)

Since (**KB** ∧ ¬α) is **unsatisfiable**, we may infer that **KB** ⊨ α, i.e., α is true given the **KB**.

5a. [5.0 marks] A student is working on an application to determine the probability that a person boarding the NUS shuttle bus is a tourist. The student has managed to gather the following data.

Wearing Hat/Cap	Appears Lost	Age Group	Dressing	Carrying Laptop	Tourist (Label)
True	False	Young	Casual	False	No
False	True	Middle-Aged	Formal	True	No
True	True	Old	Smart-Casual	False	Yes
False	False	Young	Casual	True	No
True	False	Young	Casual	False	Yes
False	True	Middle-Aged	Formal	True	Yes
False	False	Old	Smart-Casual	True	No
True	True	Middle-Aged	Formal	True	No
False	False	Old	Smart-Casual	False	Yes

Apply the **naive Bayes** algorithm to ***determine the probability that a new passenger on the NUS shuttle bus is a tourist*** given that the new passenger: (1) is not wearing a hat/cap, (2) does not appear lost, (3) is middle-aged, (4) is dressed smart-casual, and (5) carries a laptop.

Solution:

Denote:

- **S**: {Cap = F, Lost = F, Age = Middle, Dress = Smart-Casual, Laptop = True}

We wish to determine:

- P(Tourist | **S**) = P(**S** | Tourist) * P(Tourist) / P(**S**) // Bayes' Theorem

From the data, we have:

- P(Tourist) = 4/9; P(Cap = F) = 5/9; P(Lost = F) = 5/9; P(Age = Middle) = 3/9; P(Dress = Smart-Casual) = 3/9; P(Laptop = True) = 5/9
- P(Cap = F | Tourist) = 2/4; P(Cap = F | Not Tourist) = 3/5
- P(Lost = F | Tourist) = 2/4; P(Lost = F | Not Tourist) = 3/5
- P(Age = Middle | Tourist) = 1/4; P(Age = Middle | Not Tourist) = 2/5
- P(Dress = Smart-Casual | Tourist) = 2/4; P(Dress = Smart-Casual | Not Tourist) = 1/5
- P(Laptop = True | Tourist) = 1/4; P(Laptop = True | Not Tourist) = 4/5

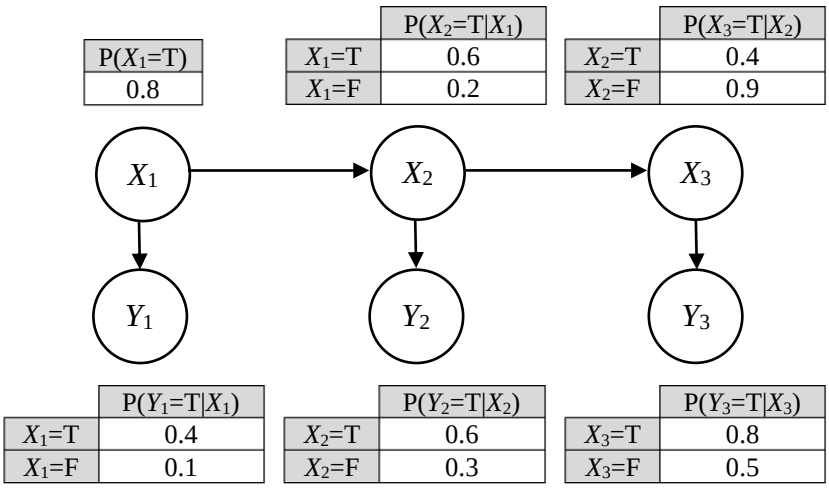
Thus, we have:

- P(**S** | Tourist) = 2/4 * 2/4 * 1/4 * 2/4 * 1/4 = 0.0078125
- P(**S** | Not Tourist) = 3/5 * 3/5 * 2/5 * 1/5 * 4/5 = 0.02304
- P(**S**) = P(**S** | Tourist)·P(Tourist) + P(**S** | Not Tourist)·P(Not Tourist)
  = (0.0078125)(4/9) + (0.02304)(1 - 4/9) = 0.016272222222222223

And finally:

- P(Tourist | **S**) = P(**S** | Tourist) * P(Tourist) / P(**S**)
  = 0.0078125 * 4/9 / 0.016272222222222223 = **0.21338340730624786**

5b. Consider the following Bayesian Network.



(i) [1.0 marks] State the formula for the joint probability distribution induced by the above Bayesian Network – i.e., define the expression for P(X_1, X_2, X_3, Y_1, Y_2, Y_3).

Solution:

$P(X_1)\cdot P(X_2|X_1)\cdot P(X_3|X_2)\cdot P(Y_1|X_1)\cdot P(Y_2|X_2)\cdot P(Y_3|X_3)$

(ii) [2.0 marks] Calculate the probability of P(X_1 = F, X_2 = T, X_3 = F, Y_1 = F, Y_2 = T, Y_3 = F).

Solution:

$P(X_1=F)\cdot P(X_2=T|X_1=F)\cdot P(X_3=F|X_2=T)\cdot P(Y_1=F|X_1=F)\cdot P(Y_2=T|X_2=T)\cdot P(Y_3=F|X_3=F)$
$= (1 - 0.8)(0.2)(1 - 0.4)(1 - 0.1)(0.6)(1 - 0.5)$
$=$ **0.00648**

(iii) [1.0 marks] Given P(Y_1 = T) = 0.5, calculate the probability of P(X_1 = T | Y_1 = T).

Solution:

$P(X_1=T|Y_1=T)$
$= P(Y_1=T|X_1=T)\cdot P(X_1=T)/P(Y_1=T)$
$= (0.4)(0.8) / (0.5)$
$=$ **0.64**

/* Note that in this question, we assert a newly observed value for the prior P(Y_1) and use the Bayesian Belief Network determine updated posteriors */

(iv) [1.0 marks] Given P(Y_1 = T) = 0.5, calculate the probability of P(X_1 = T | Y_1 = F).

Solution:

$P(X_1=T|Y_1=F)$
$= P(Y_1=F|X_1=T)\cdot P(X_1=T)/P(Y_1=F)$
$= (1 - 0.4)(0.8)/(0.5)$
$=$ **0.96**

/* Note that in this question, we assert a newly observed value for the prior P(Y_1) and use the Bayesian Belief Network determine updated posteriors */

(v) [2.0 marks] Given P(Y_1 = T) = 0.5, calculate the probability of P(X_2 = T | Y_1 = T).

Solution:

P(X_2=T|Y_1=T)
= P(X_2=T,Y_1=T) / P(Y_1=T)
= [P(X_2=T,Y_1=T|X_1=T)·P(X_1=T)+P(X_2=T,Y_1=T|X_1=F)·P(X_1=F)] / P(Y_1=T)
= [P(X_2=T|X_1=T)·P(Y_1=T|X_1=T)·P(X_1=T) +
P(X_2=T|X_1=F)·P(Y_1=T|X_1=F)·P(X_1=F)] / P(Y_1=T)
= [(0.6)(0.4)(0.8) + (0.2)(0.1)(0.2)] / 0.5
= **0.392**

/* Note that in this question, we assert a newly observed value for the prior P(Y_1) and use the Bayesian Belief Network determine updated posteriors */

(vi) [2.0 marks] Suppose that for a given problem we have two different **Bayesian Networks** (**BN**), $\boldsymbol{G}_1$ = ($\boldsymbol{V}$, $\boldsymbol{E}_1$) and $\boldsymbol{G}_2$ = ($\boldsymbol{V}$, $\boldsymbol{E}_2$). Describe an approach that can be used to decide which **BN**, $\boldsymbol{G}_1$ or $\boldsymbol{G}_2$, is a better model of the problem.

Solution:

Use ***real data*** to ***evaluate*** both $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$. This requires a definition of some ***performance metric*** based on model accuracy to evaluate the correctness of $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$.

/* Examples of such a performance metrics are Bayesian information criterion (BIC), or Akaike information criterion (AIC) */

END OF PAPER

BLANK PAGE

BLANK PAGE

BLANK PAGE

BLANK PAGE