CS 188 Spring 2024 — Introduction to Artificial Intelligence — Midterm

Solutions last updated: Thursday, March 7th

- You have 110 minutes.
- The exam is closed book, no calculator, and closed notes, other than one double-sided cheat sheet that you may reference.
- Anything you write outside the answer boxes or you ~~cross out~~ will not be graded. If you write multiple answers, your answer is ambiguous, or the bubble/checkbox is not entirely filled in, we will grade the worst interpretation.

For questions with **circular bubbles**, you may select only one choice.

○ Unselected option (completely unfilled)

● Only one selected option (completely filled)

✓ Don't do this (it will be graded as incorrect)

For questions with **square checkboxes**, you may select one or more choices.

■ You can select

■ multiple squares (completely filled)

First name	
Last name	
SID	
Name of person to the right	
Name of person to the left	
Discussion TAs (or None)	

Honor code: "As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others."

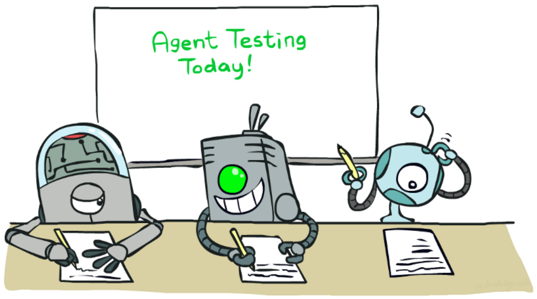
By signing below, I affirm that all work on this exam is my own work, and honestly reflects my own understanding of the course material. I have not referenced any outside materials (other than my cheat sheets), nor collaborated with any other human being on this exam. I understand that if the exam proctor catches me cheating on the exam, that I may face the penalty of an automatic "F" grade in this class and a referral to the Center for Student Conduct.

Signature: ___________________________

Point Distribution

Q1.	Potpourri	12
Q2.	Search: Half the Dots	17
Q3.	Logic: Pacdrone	15
Q4.	Logic: Transactions	21
Q5.	Games: Five Nights at Wheeler	17
Q6.	Bayes Nets: Easter Island Elections	18
	Total	100

It's testing time for our CS188 robots!
Circle your favorite robot below.
(ungraded, just for fun)



Q1. [12 pts] Potpourri

In the next two subparts, consider a search problem with maximum branching factor of b, and maximum search tree depth of m (i.e. the search tree is finite).

(a) [1 pt] What is the **space complexity** (maximum number of nodes on the fringe at any given time) for **breadth-first** tree search?

● $O(b^m)$ ○ $O(bm)$ ○ $O(m)$ ○ $O(b)$

In the worst case, breadth-first search needs to keep all the search nodes on the current layer of the search tree on the fringe (after dequeueing everything at layer $m-1$ and enqueueing everything at layer m).

At the bottom layer m, there are $O(b^m)$ nodes that could potentially all be on the queue at the same time.

(b) [1 pt] What is the **time complexity** (maximum number of nodes explored in the worst case) for **depth-first** tree search?

● $O(b^m)$ ○ $O(bm)$ ○ $O(m)$ ○ $O(b)$

In the worst case, we may need to explore the entire tree before finding a solution. The tree contains roughly $1 + b + b^2 + \ldots + b^m = O(b^m)$ search nodes.

In the next three subparts, consider a search problem with a finite state space. Suppose we have an admissible heuristic function $h_1(n)$, a consistent heuristic function $h_2(n)$, and a heuristic function $h_3(n) = \max(h_1(n), h_2(n))$.

(c) [2 pts] Select all heuristics that will guarantee that A* **graph** search (used with that heuristic) will return the optimal solution.

☐ $h_1(n)$ ■ $h_2(n)$ ☐ $h_3(n)$ ○ None of the above

For A* graph search to be optimal, we need to use a consistent heuristic: $h_2(n)$. $h_1(n)$ is admissible, but this does not imply that it is consistent. We can find a counterexample where $h_3(n)$ is admissible, but not consistent. Suppose that $h_1(n)$ is admissible but not consistent and $h_2(n) = 0$. Thus, $h_3(n) = \max(h_1(n), 0) = h_1(n)$, which means that $h_3(n)$ is not consistent.

(d) [2 pts] Select all heuristics that will guarantee that A* **tree** search (used with that heuristic) will return the optimal solution.

■ $h_1(n)$ ■ $h_2(n)$ ■ $h_3(n)$ ○ None of the above

For A* tree search to be optimal, we need to use an admissible heuristic. $h_1(n)$ is admissible, $h_2(n)$ is consistent, which implies admissibility, and we can show $h_3(n)$ is admissible. Since $h_1(n)$ and $h_2(n)$ are admissible (consistency implies admissibility), then we know that

$$0 \le h_1(n) \le h^*(n) \tag{1}$$

and,

$$0 \le h_2(n) \le h^*(n) \tag{2}$$

Where $h^*(n)$ is the true heuristic. Using equation (1) and (2), we get

$$0 \le \max(h_1(n), h_2(n)) \le \max(h^*(n), h^*(n)) = h^*(n) \tag{3}$$

Thus, $h_3(n) = \max(h_1(n), h_2(n))$ is admissible by definition.

(e) [2 pts] Select all true statements.

☐ A^* tree search is always optimal.

☐ A^* graph search is optimal if the heuristic function used is admissible.

- [✓] A^* graph search is optimal if the heuristic function used is consistent.
- ○ None of the above

Option (A): A* tree search can be suboptimal with an non-admissible heuristic.
Option (B): A* graph search is only optimal if the heuristic is consistent, which is a stricter criterion than admissibility.
Option (C): See Option (B)'s explanation.

(f) [2 pts] In the context of Monte Carlo Tree Search (MCTS), when are nodes likely to be visited more frequently during the search process? Select all that apply.

- [ ] Nodes that have a lower win rate, as the algorithm seeks to explore less successful paths more thoroughly.
- [✓] Nodes that have a higher win rate, as the algorithm prioritizes paths with a higher probability of success.
- [✓] Nodes that have fewer rollouts, as the algorithm prioritizes exploration.
- [ ] Nodes at the deepest level of the tree, as the algorithm focuses on fully exploring the end.
- ○ None of the above

In MCTS, we want to allocate rollouts to nodes that have a higher win rate (Option (B)) or more uncertainty ((Option (C)) (refer to slide 55 and 56 in lecture 6). Option (A) contradicts our strategy of maximizing our success under uncertainty. In many problems we apply MCTS, there are many ending states (like the board game Go), and it would be intractable to explore all of them. This means that many of the nodes at the deepst level end up not being searched (Option (D)). MCTS allows us to search the state space intelligently, so we do not have to look at every leaf node.

(g) [2 pts] Which of these scenarios would be most appropriate for a Hidden Markov Model (HMM) setup?

- ○ You receive daily reports of "sun" or "no sun." You know the probability of sun only depends on whether there was sun the previous day. You want to estimate the probability of sun on future days.
- ● You receive daily reports of a patient's symptoms, but cannot directly observe the state of the patient's disease. You know how the disease causes the symptoms, and you know that the state of the disease on a given day only depends on the state of the disease on the previous day. You want to estimate the probability of disease on each day.
- ○ You receive the exact location of a vehicle every minute. You control the vehicle, and you want to guide the vehicle to its destination.

Option (A) is false because there is no hidden variable. This can be represented with a Markov model, where you directly observe the variable you are trying to predict (sun or no sun).

Option (B) is true. You cannot directly observe the query variable (disease), but you can observe evidence (symptoms), and you know how the disease causes the symptoms. Also, you have the Markov assumption that the probability of disease on a given day only depends on the state of the disease on the previous day. Therefore, an HMM would be a good choice here.

Option (C) is false. HMMs are used for reasoning over an uncertain query variable. There is no notion of taking actions or reaching a goal in an HMM.

Q2. [17 pts] Search: Half the Dots

Recall the "All-the-Dots" problem from Project 1. Here are some reminders of Pacman rules: Pacman lives in an M by N grid. He can move up, down, left, or right, but he is not allowed to take actions that move into a wall. Walls may exist throughout the grid, but the locations of the walls do not change for the entire problem. In the starting state, there are K dots throughout the grid. Pacman's goal in Project 1 was to eat all K dots.

In this question, Pacman has built a spaceship and has landed on a new world, Earth Prime (also an M by N grid), with K dots. For this entire question, you can assume that M, N, and K are fixed, and you can **assume that K is even**. Pacman's new goal is to solve the "Half-the-Dots" search problem and eat **any** $K/2$ dots!

(a) [2 pts] Select all pieces of information that should be part of a **minimal state representation**.

Note: In other words, all the options you select should, together, form the smallest valid state representation. Consider only space complexity (do not consider time complexity).

- ☐ Coordinates of Pacman's starting position.
- [✓] Coordinates of Pacman's current position.
- ☐ A boolean matrix with the dots' starting positions.
- [✓] A boolean matrix indicating which dots have not been eaten.
- ☐ A boolean matrix indicating where Pacman has visited.

In general, we require Pacman's current position, as well as a way to track where the dots are. (B) and (D) together provide a minimal state representation, by tracking these two things.

Since we don't require Pacman's starting position or the dots starting position, answer choices (A) and (C) are incorrect. Answer choice (E) is also not useful for this search problem; we only need to eat all the dots, not something like visit every square.

For the rest of the question, assume that Pacman's state representation is an M by N matrix of integers, where the integers are interpreted as follows: 0 indicates an empty square, 1 indicates a wall is present, 2 indicates that Pacman is present, and 3 indicates that a dot is present.

(b) [2 pts] Suppose Pacman implements a goal test, $G(s)$. This function takes in a state (and no other information about the problem), and returns whether this state passes the goal test.

What is the time complexity (in big O notation) of $G(s)$, in terms of M, N, and K? Assume the function is optimal (written to be as fast as possible).

$\boxed{O(\qquad\qquad)}$

$O(MN)$, because you have to iterate over the entire matrix to count how many dots remain; ie. count how many 3's are in the state representation. If this number is less than $K/2$, then the goal test returns True; otherwise, return False.

Given the state representation above, there is no other way to directly count how many dots there are without iterating through the entire matrix, which has size M by N.

(c) [2 pts] Blinky thinks that Pacman's state representation is inefficient, and suggests adding this variable to the state space: R = the number of remaining dots Pacman needs to eat to solve the problem.

Select all true statements about Blinky's modified problem.

- ☐ The original successor function, with no modifications, can output successor states for the modified problem.
- [✓] The logic of the original goal test can still be used for the modified problem.
- ☐ Running BFS (with all necessary modifications made) will explore fewer search nodes, compared to the unmodified problem.
- ☐ If we only count valid, reachable states, the size of the state space will increase by a factor of R.
- ○ None of the above

The key realization in this problem is that R is a redundant variable. R does not change the search problem; it only changes the representation of the search problem.

Option (A) is false. The successor function now needs to update R when generating successor states as well.

Option (B) is true. The original goal test of checking the $N \times M$ grid for $K/2$ (or fewer) dots remaining is still valid, even if there is an extra variable that could make computation faster (as seen in the next subpart).

Option (C) is false. Intuitively, we are solving the exact same search problem (just with a redundant variable added to the state space), so we should end up exploring the exact same set of search nodes.

Option (D) is false. The number of states is still the same; every state just has an extra counter R attached. You might be tempted to argue that the state space expands by a factor of R, but remember that each state in the original problem only has one valid corresponding state in the modified problem (the state where the R value matches the number of dots left on the grid). For example, you can't have a state where there are 5 dots left on the grid, but $R = 6$. This state is not reachable, and the question asks to only count valid, reachable states.

(d) [2 pts] With Blinky's modification, what is the time complexity of the goal test function? Assume the function is optimal (written to be as fast as possible).

$\boxed{O(\qquad\qquad)}$

$O(1)$, because you can read off R. This takes the same amount of time, no matter how large the grid is. $O(\log R)$, the number of bits in R is also acceptable.

(e) [3 pts] Suppose we have an arbitrary admissible heuristic $h(s)$ for the All-the-Dots problem. Select all heuristics that are guaranteed to be admissible for the Half-the-Dots problem.

- ☐ $h(s)$
- ☐ $\frac{1}{2}h(s)$
- ☐ $\frac{1}{2}cost(s)$, where $cost(s)$ is the cost of the optimal solution to the All-the-Dots problem from state s.
- ☐ Half of the currently remaining number of dots.
- 🟥 The number of remaining dots that need to be eaten for $K/2$ dots to be eaten in total.
- ○ None of the above

For answer choices (A) through (E), consider that the heuristic $h(s) = h^{*all}(s)$, the true cost for the All-the-Dots problem.

(A), (B), (C) will all overestimate the goal. Consider the following counterexample:

●	P			●

Here, the optimal cost for the All-the-Dots problem is 5: going left, then going right. The optimal cost of the Half-the-Dots problem is 1. We see here that $h(s) = h^{*all}(s)$ overestimates the the goal cost and thus is inadmissible.

(D) is false, because we want half of the total, starting number of dots. Suppose there are $K = 100$ dots in the beginning, and we have eaten 49, and the last dot is next to us. The true cost is 1, and this heuristic returns 25.5, grossly overestimating the goal.

(E) is correct as a problem relaxation, where you assume Pacman can teleport to any dot, and he must teleport to half the dots. It is an example of a heuristic that lower bounds the amount of steps or costs needed to reach the goal.

(f) [2 pts] Consider the following heuristic:

First, order the remaining dots in **descending** order of Manhattan distance. Examples: The 1st dot is the furthest dot from Pacman. The 2nd dot is the second-furthest dot from Pacman. In the start state, the Kth dot is the closest dot to Pacman.

If there are X or fewer dots remaining, return 0. Else, return the Manhattan distance to the Yth dot in the list.

Do there exist any values of X and Y that make this heuristic admissible? If yes, write down the smallest values of X and Y that make it admissible. Otherwise, write N/A in both squares. You may answer in terms of K, the number of dots the problem begins with. *Remember K is even.*

X: 　　　　　　　　Y:

NOTE: During the exam, we posted the clarification: Q2(f): K is the only variable you can use.

$X = K/2, Y = (K/2) + 1.$

This heuristic is admissible. It represents a problem relaxation, where Pacman only needs to travel to the farthest necessary dot, and on the way he automatically collects any other necessary dots.

For example, suppose $K = 10$. What this heuristic does is sort the dots in order of distance, and selects the $(K/2) + 1 =$ 6th dot, which is actually the fifth closest dot. This means that this heuristic prints out the Rth dot to go.

A trivial answer is $X = K, Y = 1$, which always returns 0.

For the following subparts, Pacman has a supercomputer, ORACLE, that can instantly compute the optimal solution to a given "All-the-Dots" search problem. Pacman is still attempting to find an optimal solution to the "Half-the-Dots" problem.

(g) [2 pts] Consider this strategy: At the beginning of the problem, Pacman selects the $K/2$ closest dots by Manhattan distance and runs ORACLE to find the optimal solution for an "All-the-Dots" problem with only the $K/2$ selected dots (effectively ignoring the existence of the farther dots).

Will this strategy return an optimal solution to the "Half-the-Dots" problem?

- ○ Yes, because the optimal solution always uses the closest dots to the starting position.
- ○ Yes, because planning the complete path ahead of time is always optimal.
- ○ No, because planning the complete path from the starting position can never be optimal for this problem.

[✓ No, because the optimal solution does not always use the closest dots to the starting position.]

The answer is (D).

It cannot be (A) or (B), because the solution is not guaranteed to be optimal. (C) is false, because it is possible for planning the complete path from the starting position to be optimal (consider infinite depth minimax).

For why it is not optimal, consider the following counterexample:

●	●	*P*	●	●

Here, $K = 4$, and ORACLE will return a solution that goes one step left and two steps right, or one step right and two steps left. The true optimal solution is to take two steps total, either left or right.

(h) [2 pts] Suppose Pacman uses the method in part (g), but instead as a heuristic to A^* search. For a given state, Pacman calculates R, the remaining number of dots he needs to eat to reach $K/2$ dots eaten. Then, Pacman runs ORACLE on the R closest dots by Manhattan distance. The length of the optimal path returned by ORACLE is used as the heuristic value.

This heuristic is not admissible. Select the counterexample that shows this.

In the grids below, $K = 4$ and $R = 2$. Assume that Pacman breaks ties arbitrarily. P marks Pacman's position.

○

P		●		
	●			
			●	●

○

P		●		
		●		
		●	●	

[✓]

P				●
				●
●			●	

○

P			●	
				●
			●	●

The correct answer is option (C), the top right grid. It is the only grid in which the ORACLE solution overestimates the cost to reach the goal.

(A), top left: ORACLE returns 4, the true cost is 4. Example path for both: Right, Down, Up, Right.

(B), bottom left: ORACLE returns 3, true cost is 3. The path for both: Right, Right, Down.

(C), top right: ORACLE returns 8, using path: Down, Down, Up, Up, Right, Right, Right, Right. However, the true cost is 5, which is path: Right, Right, Right, Right, Down.

(D), bottom right: ORACLE returns 5, true cost is 5. Example path for both: Right, Right, Right, Down, Down.

Q3. [15 pts] Logic: Pacdrone

Pacman needs your help predicting what his drone will do on Mars! Let's do some first order logic warmup.

Suppose there are many sensors: S_0, S_1, S_2, …. Each sensor is located either on Mars, on Earth, or on the Drone.

If sensor S_i is present on Mars, on Earth, or on the Drone, then OnMars(S_i), OnEarth(S_i), or OnDrone(S_i) returns True, respectively. Otherwise, the predicate returns False.

A sensor can *disrupt* another sensor. Disrupts(S_i, S_j) is True if S_i disrupts S_j. Note that if S_i disrupts S_j, that does not necessarily mean that S_j disrupts S_i.

Choose all statements that are equivalent to the given statement.

(a) [2 pts] All sensors on Mars disrupt some sensor that is either on Earth or on the Drone.

- ☐ $\exists s_a, \text{OnMars}(s_a) \implies \forall s_b, \text{Disrupts}(s_a, s_b) \vee \text{OnMars}(s_b)$
- ☐ $\forall s_a, \text{OnMars}(s_a) \implies \forall s_b, \text{Disrupts}(s_a, s_b) \wedge \text{OnMars}(s_b)$
- ☐ $\exists s_a, \text{OnMars}(s_a) \implies \exists s_b, \text{Disrupts}(s_a, s_b) \vee \left(\text{OnEarth}(s_b) \vee \text{OnDrone}(s_b)\right)$
- ■ $\forall s_a, \text{OnMars}(s_a) \implies \exists s_b, \text{Disrupts}(s_a, s_b) \wedge \left(\text{OnEarth}(s_b) \vee \text{OnDrone}(s_b)\right)$
- ○ None of the above

Half a point for every box. The correct answer is the fourth statement: For every sensor, if it is on Mars, then there is a sensor out there that it disrupts, which is on Earth or on the Drone.

The first statement says: There is a sensor out there, that if it is on Mars, then it disrupts non-Martian sensors.

The second statement says: For every sensor, if it is on Mars, they disrupt all other sensors AND all other sensors are on Mars.

The third statement says: There is a sensor, that if it is on Mars, then there is also another sensor out there that it disrupts, or that sensor is on Earth or on the Drone.

(b) [3 pts] $\forall (s_a, s_b) \left[(s_a \neq s_b) \implies \left(\neg\text{Disrupts}(s_a, s_b) \vee \left(\text{OnEarth}(s_a) \wedge \text{OnDrone}(s_b)\right)\right)\right]$

- ■ If sensor A disrupts different sensor B, then A is on Earth and B is on the Drone.
- ■ Given different sensors A and B, either: A does not disrupt B, or: A is on Earth and B is on the Drone.
- ☐ If sensor A disrupts sensor B, one of them is on Earth and one of them is on the Drone.
- ☐ All sensors that disrupt another sensor are either on Earth or on the Drone.
- ■ For every distinct pair (s_a, s_b), $\text{Disrupts}(s_a, s_b) \implies [\text{OnEarth}(s_a) \wedge \text{OnDrone}(s_b)]$
- ☐ $\exists (s_a, s_b)$, $\text{Disrupts}(s_a, s_b) \implies \neg\text{OnEarth}(s_a) \vee \neg\text{OnDrone}(s_b)$
- ○ None of the above

During the exam, we posted these clarifications:

Q3(b), second answer choice, add "or both" at the end of the sentence.

Q3(b) A distinct pair (s_a, s_b) is a pair such that $s_a \neq s_b$

Converting the right hand side makes for more interpretibility:

$\forall (s_a, s_b), s_a \neq s_b \implies \left[Disrupts(s_a, s_b) \implies [OnEarth(s_a) \wedge OnDrone(s_b)]\right].$

Options (A), (B) and (E) are correct.

Option (C) is wrong because it allows for sensor B on Earth and sensor A on the Drone, but the logical statement strictly requires A to be on Earth and B on the drone.

Option (D): note that disruption is a one way relationship. The statement given in the problem only asserts if A disrupts B, not that they both disrupt each other.

Option (F): we are not stating that there exists two sensors out there for which this is true, but that it applies to *all* distinct pairs of sensors.

The rest of the problem is independent of the previous subparts.

Pacman tells you that there are four binary environment variables on Mars: Dusty (D), Cloudy (C), Windy (W), Sunny (S). However, we have no access to these values directly. Instead, we observe the truth values of the drone's logical sensors, which are logical expressions that consist of the environment variables. For example, if D and C are true, then the sensor $D \implies C$ returns True. For the next three subparts, select the logical sentence that matches the given sensor description.

(c) [1 pt] Mars is either dusty or sunny.

○ $D \implies S$ ○ $S \implies D$ ○ $D \wedge S$ [✓ ● $D \vee S$]

This statement follows the definition of the logical OR operator ($\vee$).

(d) [1 pt] If Mars is windy, then it must be dusty or cloudy.

○ $W \vee D \vee C$ ○ $W \wedge (D \vee C)$ [✓ ● $W \implies (D \vee C)$] ○ $(D \vee C) \implies W$

The definition of an implication statement $A \implies B$ matches the English statement "If A then B". In this case, "Mars is windy" is referred to by W, and "dusty or cloudy" can be represented as $D \vee C$

(e) [1 pt] Whenever it is not sunny, if it is windy, then it is dusty.

○ $\neg S \wedge W \wedge D$ ○ $\neg S \vee W \vee D$ [✓ ● $S \vee \neg W \vee D$] ○ $\neg S \implies (W \wedge D)$

Convert the implication statements.

$\neg S \implies (W \implies D)$

$S \vee (W \implies D)$

$S \vee \neg W \vee D$

Now we can help Pacman understand what his Drone will do on Mars. Pacman tells you he outfitted the Drone with three sensors:

Solar Panel: $S \wedge \neg D$ Window: $(C \wedge W) \vee D$ Storm: $(D \vee W) \implies (C \wedge \neg S)$

We also have our own, high tech sensors on Earth:

α: $\neg S \implies W$ β: $W \implies C$ γ: $(D \wedge C) \vee W$

Suppose that our sensors currently detect the following: α = False, β = True, γ = True.

What are the truth values of the Drone's sensors?

(f) [1 pt] Solar Panel:

- ○ True
- [✓ False]

(g) [1 pt] Window:

- [✓ True]
- ○ False

(h) [1 pt] Storm:

- [✓ True]
- ○ False

The answer is Solar Panel = False, Window = True, Storm = True.

1. Knowing α is False is very useful— if an implication claim is False in a world, that must mean that the left side is True and the right side is False. Thus we can determine that S must be False, and W is false.

2. γ is true, and we know W is false. Therefore, we conclude that D and C must be true.

3. β doesn't tell us any information, because W is false, and it is vacuously true.

We have found: S false, W false, D true, C true. Then the drone's sensors must be: False, True, True.

(i) [2 pts] The Drone decides to land and go into hibernation whenever the following Prime Directive evaluates as True:

$$(\neg\text{Window} \vee \text{Storm}) \implies \neg\text{Solar Panel}$$

Which of the following is the correct Conjunctive Normal Form (CNF) of the Prime Directive?

- ○ (Window ∨ Storm) ⟹ Solar Panel
- ○ Window ∧¬Storm ∧ Solar Panel
- ○ (Window ∧¬Storm) ∨ ¬Solar Panel
- [✓ (Window ∨ ¬Solar Panel) ∧ (¬Storm ∨ ¬Solar Panel)]

The third option is Disjunctive Normal Form, not CNF. CNF is an AND of ORs, while DNF is ORs of ANDs. Here is the breakdown:

(¬Window ∨ Storm) ⟹ ¬Solar Panel

(¬Window ⟹ ¬Solar Panel) ∧ (Storm ⟹ ¬Solar Panel)

(Window ∨ ¬Solar Panel) ∧ (¬ Storm ∨ ¬Solar Panel)

(j) [2 pts] Pacman has learned some of the environment variables: S = False, D = False, C = unknown, W = True.

The Prime Directive evaluates as True. What is the value of C?

○ True ○ False [✓ Can't calculate]

Using known values of S, D, W, we can fill in some of the sensors.

Solar Panel evaluates to False, because S is False. Window, which is $(C \land W) \lor D$, simplifies to C. Storm evaluates to C as well, because the left hand side is True; Storm will only be true if the right hand side is also True, which depends on C.

Now consider the Prime Directive. We can simplify it to:

$\neg C \lor C \implies \neg False$

This statement must always be True, because the left hand side is a tautology, and the right hand side is True. Thus, we have no way to tell what value C must be– the answer is Can't Calculate.

Q4. [21 pts] Logic: Transactions

A database system has 3 disks, A, B, and C.

A transaction is a database operation (e.g. reading a file), which requires using one or more disks for one or more consecutive time steps. At a given time step, a transaction is either running, or not running.

A disk cannot be used by two transactions at the same time. One way to enforce this is to introduce a **lock** for each disk. You can think of a lock as an object that can be acquired by a transaction. When a transaction is holding a lock, all other transactions are unable to acquire that lock (and use the lock's corresponding disk). The transaction can hold on to the lock for as long as needed. Once the transaction is finished using the disk, the transaction can release the corresponding lock for other transactions to acquire.

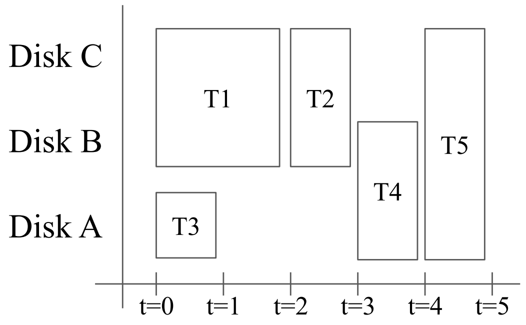
Up to 2 transactions can run at the same time, as long as they are using different disks.

There are 5 transactions we need to run, each needing to run for a certain amount of time steps and needing to use a certain set of disks. (See an example below.)

Note: You can every transaction uses all the needed disks and holds all the needed locks for the transaction's entire duration.

We would like to write logical statements to find a time for each of the 5 transactions to run, while avoiding any scheduling conflicts (e.g. ensuring transactions don't access the same disk at the same time).

Here is an example of a set of transactions, their requirements, and an example of a valid schedule. This is just an example, and is not needed to solve the problem. (Note: The height of the transaction boxes is just for indicating which disks are being used, and has no other meaning.)



Transaction	Disks Used	Time Steps Needed
T1	B, C	2
T2	B, C	1
T3	A	1
T4	A, B	1
T5	A, B, C	1

Let's define the following propositional logic symbols:

- A_1 is True if and only if Transaction 1 needs to access disk A.

  $A_2,\ A_3,\ A_4,\ A_5,\ B_1,\ B_2,\ \ldots,\ C_4,\ C_5$ are defined similarly.

- $a_1^{t=0}$ is True if and only if Transaction 1, at time $t = 0$, is holding the lock a corresponding to disk A.

  Note that we use lowercase letters to represent the locks.

  $a_2^{t=0},\ a_3^{t=0},\ a_4^{t=0},\ a_5^{t=0},\ b_1^{t=0},\ b_2^{t=0},\ \ldots,\ c_4^{t=0},\ c_5^{t=0}$ are defined similarly.

  Another similar set of variables is defined for $t = 1$, $t = 2$, etc.

- $T_1^{t=0}$ is True if and only if Transaction 1 is running at time $t = 0$.

  $T_2^{t=0},\ T_3^{t=0},\ T_4^{t=0},\ T_5^{t=0}$ are defined similarly.

  Another similar set of variables is defined for $t = 1$, $t = 2$, etc.

The symbol definitions, repeated for your convenience:

- A_1 is True if and only if Transaction 1 needs to access disk A.
- $a_1^{t=0}$ is True if and only if Transaction 1, at time $t = 0$, is holding the lock a corresponding to disk A.
- $T_1^{t=0}$ is True if and only if Transaction 1 is running at time $t = 0$.

In the next few subparts, select the logical sentence that matches the English statement.

(a) [2 pts] If Transaction 1 is holding a lock at time $t = 0$, then Transaction 1 is running at that time, and needs to access the disk corresponding to the lock.

- $a_1^{t=0} \vee b_1^{t=0} \vee c_1^{t=0}$
- $a_1^{t=0} \implies \left(\neg a_2^{t=0} \wedge \neg a_3^{t=0} \wedge \neg a_4^{t=0} \wedge \neg a_5^{t=0}\right)$
- $\left((A_1 \wedge T_1^{t=0}) \implies a_1^{t=0}\right) \wedge \left((B_1 \wedge T_1^{t=0}) \implies b_1^{t=0}\right) \wedge \left((C_1 \wedge T_1^{t=0}) \implies c_1^{t=0}\right)$
- [✓ $\left(a_1^{t=0} \implies (A_1 \wedge T_1^{t=0})\right) \wedge \left(b_1^{t=0} \implies (B_1 \wedge T_1^{t=0})\right) \wedge \left(c_1^{t=0} \implies (C_1 \wedge T_1^{t=0})\right)$]

$a_1^{t=0}$ is Transaction 1 holding lock a at time 0. If then is an implication, leading to A_1 being the fact that Transaction 1 accesses Disk A and $T_1^{t=0}$ is Transaction 1 running at time 1. Lastly, we repeat this for locks b and c since we want "a lock".

(b) [1 pt] Is the logical sentence in the previous subpart an axiom that is true of the problem we described?

In other words: Select "Yes" if the logical sentence above should be included for any possible set of 5 transactions we want to schedule.

Note: In the problem, a transaction could possibly hold a lock without using the corresponding disk.

Yes [✓ No]

Although it's intuitive for this is to be true in real life and this doesn't reduce the set of solutions (schedules) that are available because the SAT solver has the option to not give extraneous locks, this is not required by the problem statement.

(c) [2 pts] If Transaction 3 needs to use Disk A and is running at time $t = 2$, then no other transaction that uses Disk A should be running at the same time step.

- [✓ $(A_3 \wedge T_3^{t=2}) \implies \left((\neg A_1 \vee \neg T_1^{t=2}) \wedge (\neg A_2 \vee \neg T_2^{t=2}) \wedge (\neg A_4 \vee \neg T_4^{t=2}) \wedge (\neg A_5 \vee \neg T_5^{t=2})\right)$]
- $a_3^{t=2} \implies \left(\neg a_1^{t=2} \vee \neg a_3^{t=2} \vee \neg a_4^{t=2} \vee \neg a_5^{t=2}\right)$
- $(A_3 \wedge T_3^{t=2}) \implies a_3^{t=2}$
- $a_3^{t=2} \implies (A_3 \wedge T_3^{t=2})$

The left side is logically equivalent to the English "if" for all options because the $(A_3 \wedge T_3^{t=2})$ implies $a_3^{t=2}$. Since we want no other transaction, the right side needs to have something for every transaction, so only options 1 and 2 remain. The first option matches the then part and it joins the conditions for each transaction with and, which is correct because we want all of the transaction to be prevented from using the Disk, not just at least one.

Interestingly, option 2 would have been logically equivalent (but not direct translation) if it had $\wedge$ instead of $\vee$. This is because $(A_1 \wedge T_1^{t=2}) \implies a_1^{t=2}$. When we negate this implication, we get $\neg a_1^{t=2} \implies \neg(A_1 \wedge T_1^{t=2})$. This means that we are able to establish a chain: option 1 left implies option 2 left (known from rest of setup), and option 2 left implies option 2 right (via selecting it), and option 2 right implies option 1 right (known from rest of setup).

(d) [1 pt] Is the logical sentence in the previous subpart an axiom that is true of the problem we described?

[✓ Yes] ○ No

Yes, the problem specifies that any actively accessed disk is exclusively accessed by only that transaction.

(e) [2 pts] We never want the database system to idle (do nothing). Select the logical sentence that matches the statement below:

"At least one transaction is running at $t = 4$."

- [✓ $T_1^{t=4} \vee T_2^{t=4} \vee T_3^{t=4} \vee T_4^{t=4} \vee T_5^{t=4}$]
- ○ $T_1^{t=4} \wedge T_2^{t=4} \wedge T_3^{t=4} \wedge T_4^{t=4} \wedge T_5^{t=4}$
- ○ $\left(a_1^{t=4} \vee a_2^{t=4} \vee \ldots \vee a_5^{t=4}\right) \wedge \left(b_1^{t=4} \vee b_2^{t=4} \vee \ldots \vee b_5^{t=4}\right) \wedge \left(c_1^{t=4} \vee c_2^{t=4} \vee \ldots \vee c_5^{t=4}\right)$
- ○ $\left(T_1^{t=0} \implies T_1^{t=1}\right) \vee \left(T_1^{t=1} \implies T_1^{t=2}\right) \vee \ldots \vee \left(T_1^{t=3} \implies T_1^{t=4}\right)$

This is satisfied by Transaction 1 running, or 2, etc. This is also the implementation of the *atLeastOne* function from Project 3.

The symbol definitions, repeated for your convenience:

- A_1 is True if and only if Transaction 1 needs to access disk A.
- $a_1^{t=0}$ is True if and only if Transaction 1, at time $t = 0$, is holding the lock a corresponding to disk A.
- $T_1^{t=0}$ is True if and only if Transaction 1 is running at time $t = 0$.

(f) [2 pts] Select the English sentence that matches the sentence below:

$$\begin{aligned}
&\left((T_1^{t=0} \wedge T_2^{t=0}) \implies (\neg T_3^{t=0} \wedge \neg T_4^{t=0} \wedge \neg T_5^{t=0})\right) \wedge \\
&\left((T_1^{t=0} \wedge T_3^{t=0}) \implies (\neg T_2^{t=0} \wedge \neg T_4^{t=0} \wedge \neg T_5^{t=0})\right) \wedge \\
&\left((T_1^{t=0} \wedge T_4^{t=0}) \implies (\neg T_2^{t=0} \wedge \neg T_3^{t=0} \wedge \neg T_5^{t=0})\right) \wedge \\
&\left((T_1^{t=0} \wedge T_5^{t=0}) \implies (\neg T_2^{t=0} \wedge \neg T_3^{t=0} \wedge \neg T_4^{t=0})\right) \wedge \\
&\left((T_2^{t=0} \wedge T_3^{t=0}) \implies (\neg T_1^{t=0} \wedge \neg T_4^{t=0} \wedge \neg T_5^{t=0})\right) \wedge \\
&\ldots \\
&\left((T_4^{t=0} \wedge T_5^{t=0}) \implies (\neg T_1^{t=0} \wedge \neg T_2^{t=0} \wedge \neg T_3^{t=0})\right)
\end{aligned}$$

[✓ We can run at most 2 transactions at time $t = 0$.]

○ We can run at most 2 transactions consecutively (without a time step in between transactions).

○ We can run any 2 transactions consecutively (without a time step in between transactions).

○ None of the above.

If any pair of two transaction is already known to be running, we know that none of the other transactions are running.

(g) [3 pts] For this subpart only, suppose there are X transactions, each accessing exactly K disks each, D total disks, and T time steps. How many different propositional logic symbols are used to specify this problem?

You can answer in big-O notation, i.e. you can drop lower-order terms in the summation.
Unrelated example: $O(MN + 3N)$ can be simplified to $O(MN)$.

$O(\qquad\qquad)$

$O(DTX)$

There are DX symbols of type "transaction X needs to access disk Y," one per transaction per disk.

There are DTX symbols of type "transaction X holds lock Y at time Z", one per transaction per disk per time step.

There are TX symbols of type "transaction X is running at time Y", one per transaction per time step.

In total, we have $DTX + DX + TX$ symbols. Dropping lower-order terms (as shown in the unrelated example) gives us $O(DTX)$.

(h) [2 pts] Is it possible to represent the same problem using propositional logic, without using the lock symbols like $a_1^{t=0}$?
In other words, can you write a logical problem that always outputs a solution (a set of satisfying assignments for all other symbols) if one exists, without using the lock symbols?

○ Yes, because different transactions can access different items.

[✓ Yes, because we can write a goal test that ensures that transactions running at the same time do not access the same disk.]

○ No, because it is no longer possible for a transaction to acquire a lock it does not need.

○ No, because the symbols encoding which transactions need which disks (e.g. A_1) cannot specify a time when the transaction needs the disk.

Yes, locks are redundant logically since we can express exclusivity without them, which is what they are used for here.

The relevant symbol definitions, repeated for your convenience:

- A_1 is True if and only if Transaction 1 needs to access disk A.

In the rest of the question, we want to express the same problem in first-order logic. Let's add the following constants:

- Transactions: T1, T2, T3, T4, T5
- Locks: LA, LB, LC
- Disks: A, B, C
- Time steps: 0, 1, 2, 3, 4, …

(i) [2 pts] Suppose we want to convert the symbol B_3 into first-order logic.

We define a new predicate named Accesses to help represent this symbol. What is the minimum set of constants that Accesses needs in order to compute a true/false value corresponding to B_3?

Your answer may only use constants from the list above and commas.

$$\boxed{\text{Accesses}\left(\qquad\qquad\qquad\qquad\right)}$$

Accesses(T3, B)

B_3 is true when transaction 3 (T3) accesses disk B.

In order to represent this symbol in first-order logic, we should provide the corresponding constants, T3 and B. Any other constants passed in would be extraneous.

In addition to Accesses, let's add the following predicates:

- Transaction(x), Lock(x), Disk(x), and Time(x) are True if and only if the constant x is a transaction, lock, disk, or time step, respectively.
- Locked(x, l, t) is True if and only if transaction x is holding lock l at time t.
- Running(x, t) is True if and only if transaction x is running at time t.

(j) [2 pts] Select the English sentence that matches the sentence below:

$$\left(\forall x \left(\text{Transaction}(x) \implies \neg\text{Running}(x,t)\right)\right) \implies \left(\forall x, l \left(\text{Transaction}(x) \wedge \text{Lock}(l)\right) \implies \neg\text{Locked}(x,l,t)\right)$$

- ○ Every transaction at time t needs to use at least one disk.
- [✓ If no transaction is running at time t, then none of the locks are being held.]
- ○ If a transaction is holding all locks at time t, then no other transaction can be running.
- ○ If a transaction is running at time t, that transaction must be holding at least one lock.

The Transaction(x) $\implies$ … and Transaction(x) $\wedge$ Lock(l) $\implies$ … constraint the variables that we are using as inputs to the correct object type, as required for FOL statements.

So, if for every existing transaction we know that not one of them is running, we also know that for every Locked output is false.

(k) [2 pts] Select the logical sentence that goes in the blank to match the statement below:

When a transaction is holding a lock, no other transaction can be holding the lock at that time.

$$\forall x, l, t \left(\left(\text{Transaction}(x) \wedge \text{Lock}(l) \wedge \text{Time}(t)\right) \implies (\underline{\qquad\qquad})\right)$$

- ○ $\text{Locked}(x,l,t) \implies \exists y \left(\text{Locked}(y,l,t) \implies (x \neq y)\right)$
- ○ $\text{Locked}(x,l,t) \implies \forall y \left(\text{Locked}(y,l,t) \implies (x \neq y)\right)$

○ $\text{Locked}(x, l, t) \implies \exists y \left(\text{Locked}(y, l, t) \implies (x = y)\right)$

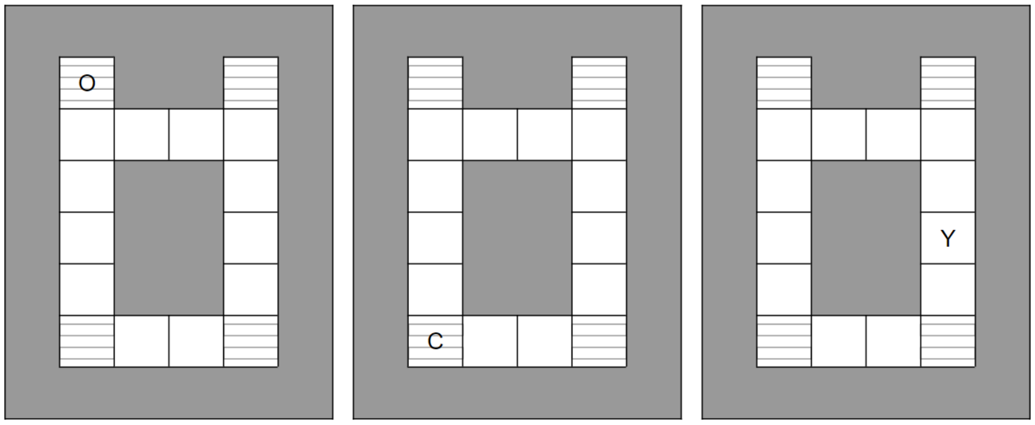
[✓ ● $\text{Locked}(x, l, t) \implies \forall y \left(\text{Locked}(y, l, t) \implies (x = y)\right)$]

For every possible Locked output, if it is true we know that looking over all transactions for the same lock and time, the only way that this is allowed is if it guarantees that it's for the same transaction.

Q5. [17 pts] Games: Five Nights at Wheeler

You and your friend Cam are being chased by Oski, a homework-eating bear. Oski has chased you and Cam inside Wheeler Hall. You decide that your homework is more important than Cam's homework and intend on getting Oski to eat Cam's homework instead of yours.

You choose to represent Wheeler Hall as a 3-dimensional $6 \times 4 \times 3$ grid, with stairs at each of the corners allowing agents to move up or down one floor (into the same corner on the new floor). Shaded squares represent walls, and squares with horizontal bars represent stairs. O is Oski, C is Cam, and Y is You.



Floor 1 Floor 2 Floor 3

At each time step, each agent can **either** move to any valid adjacent grid square **or** choose to stay in place. After choosing to stay in place, an agent can choose to continue moving on the next time step.

You choose to model the situation as a **depth-limited multi-agent game tree**. Similar to Project 2, each depth level corresponds to one action from You, followed by one action from Cam, followed by one action from Oski. The evaluation function is called on a state when the maximum depth is reached.

(a) [2 pts] Which of the following is a **valid and minimal** state representation for the specific grid shown above?

- ○ Three boolean values for each grid square, representing whether that square has You, Cam, or Oski, respectively.
- ● An (x, y, z) integer tuple for each agent, representing the agent's position.
- ○ An (x, y, z) integer tuple for Oski's position, and the Euclidean distance between You and Oski, and the Euclidean distance between Cam and Oski.
- ○ An (x, y, z) integer tuple for your position only.
- ○ None of the above.

(a) requires 216 boolean values, each can be optimally stored as a bit, which can be stored in 27 bytes.
(b) is the most minimal valid option, as it requires only 9 integers, each can be stored as a single byte (8 bits) for this specific grid, totalling 9 bytes.
(c) is not valid as there are multiple grid locations with the same distance from Oski.
(d) is not valid as it does not contain Oski's nor Cam's location.

(b) [2 pts] For the specific grid shown above, what is the **maximum branching factor** of the game tree for **any** state (not necessarily the state shown)?

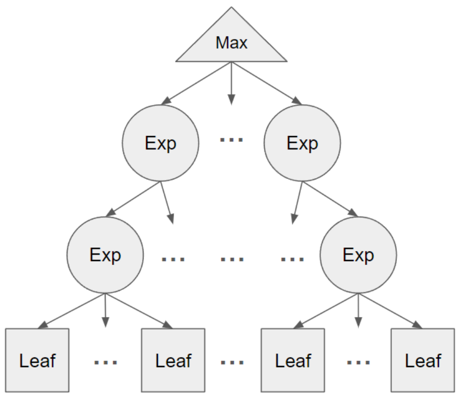
5. This branching factor is achievable at the corner stairwells of floor 2. For example, at the bottom left corner of floor 2, an agent can move north, east, up, down, or choose to stop.

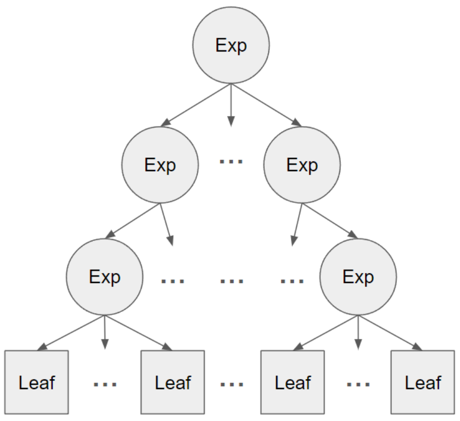
(c) [2 pts] Suppose your goal is to get Cam's homework eaten, Cam's goal is to avoid Oski, and Oski's goal is to eat either Cam's homework or your homework.

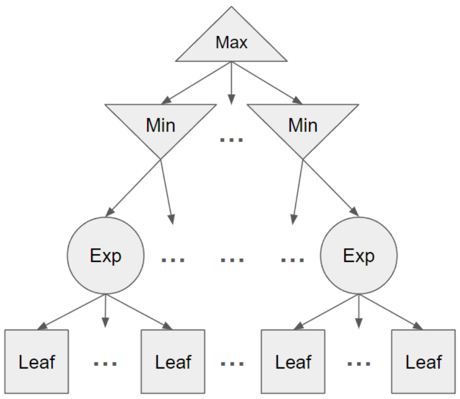
In this subpart, suppose all agents (You, Cam, and Oski) are playing optimally with respect to their own utility, and you know that Cam and Oski are playing optimally.

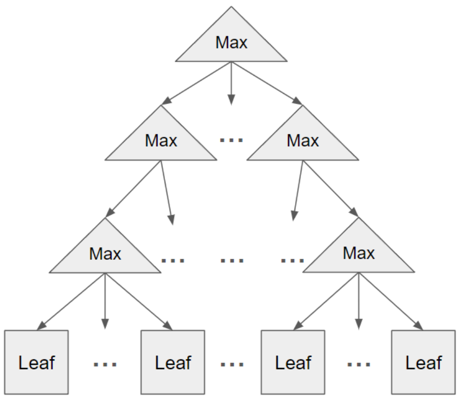
Which of the following game trees represents your model accurately with depth 1 and You as the root node?

Note: the ellipsis (...) represents omitted nodes in the tree.

○ 

○ 

○ 

[✓] 

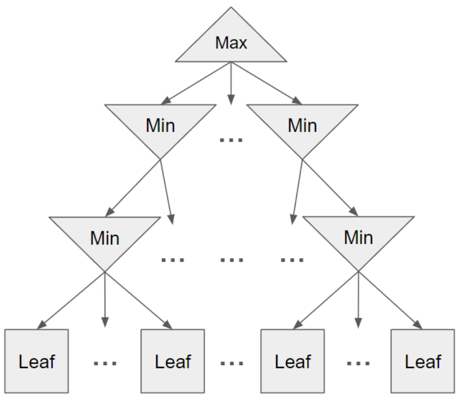
○ None of the above.

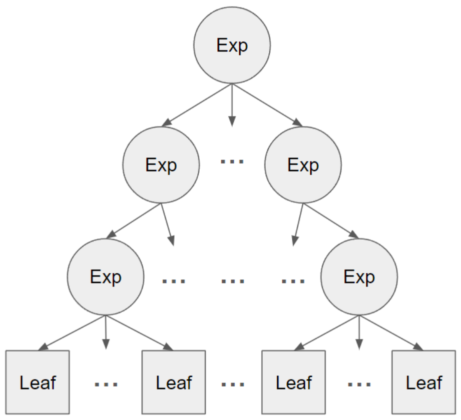
(d) All three agents are optimizing their own evaluation function separately, this can be represented as each being a maximizer node. None of the other options can represent this scenario accurately.

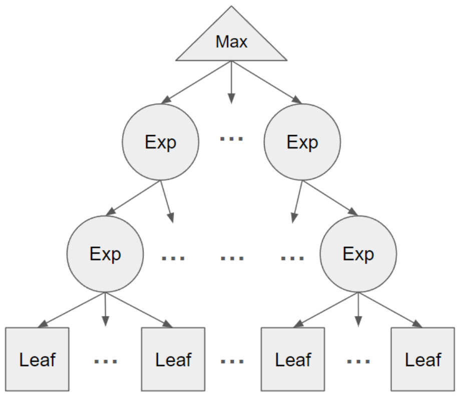
In the next two subparts, suppose Cam and Oski are selecting actions at random from some known distribution, and you know that Cam and Oski are doing this.

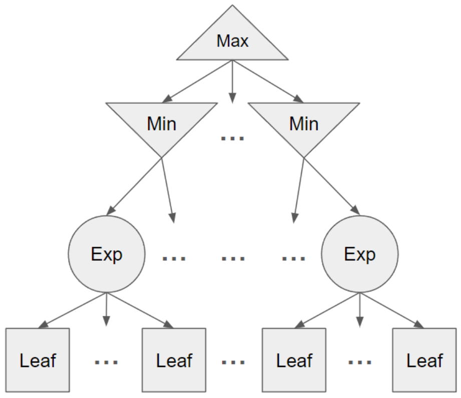
(d) [2 pts] Which of the following game trees represents your new model accurately with depth 1 and You as the root node?

Note: the ellipsis (...) represents omitted nodes in the tree.

○ 

○ 

[✓] 

○ 

○ None of the above.

Option (c) correctly represents You as a maximizer node, and both Oski and Cam as expectation nodes, as they are taking random actions.

(e) [4 pts] Write an expression representing what action you should take, according to the depth-1 game tree.

Notation:

- a_{oski}, a_{cam}, and a_{you} represent the actions available to Oski, Cam, and You, respectively.
- f_{oski}, f_{cam}, and f_{you} represent the evaluation functions used by Oski, Cam, and You, respectively.
- s' represents the successor state after taking action a from state s.

Fill in the blanks to write the expression:

$$\text{(i)} \left[\text{(ii)} \left[\text{(iii)} \left[\text{(iv)} \right] \right] \right]$$

Select one option from each column.

(i)	(ii)	(iii)	(iv)
[✓ $\operatorname{argmax}_{a_{\text{you}}}$]	[✓ $\mathbb{E}_{a_{\text{cam}}}$]	[✓ $\mathbb{E}_{a_{\text{oski}}}$]	○ $f_{\text{oski}}(s')$
○ $\max_{a_{\text{you}}}$	○ $\max_{a_{\text{cam}}}$	○ $\max_{a_{\text{oski}}}$	[✓ $f_{\text{you}}(s')$]
○ $\mathbb{E}_{a_{\text{you}}}$	○ $\min_{a_{\text{cam}}}$	○ $\min_{a_{\text{oski}}}$	○ $f_{\text{cam}}(s')$
○ $\sum_{a_{\text{you}}}$	○ $\sum_{a_{\text{cam}}}$	○ $\sum_{a_{\text{oski}}}$	○ s'

Since we want an expression that returns an action, (i) must be option 1.
(ii) and (iii) are both option 1, as Oski and Cam are represented as taking random actions.
(iv) is option 2 as you are the one taking the action at the root node, so you run your evaluation function.

The next two subparts are independent from the rest of the question.

(f) [3 pts] Consider a reflex agent who uses an evaluation function to compute a value for each successor state. However, instead of always moving to the successor state with the highest evaluation, we want the agent to probabilistically select a successor state to move to.

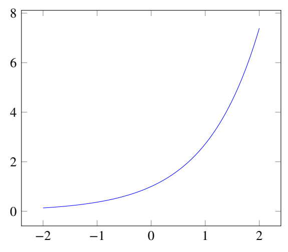
Notation:

- f is the evaluation function.
- s is the current state.
- s' is the successor state being considered.
- S' is the set of all successor states.

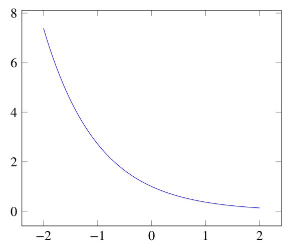
We want an expression for converting an evaluation score to a probability, satisfying the following two properties:

- The resulting probability distribution is **valid**, i.e. the probabilities of moving to the successor states must sum to 1.
- Successor states with **higher** evaluations must have a **higher** probability of being chosen.

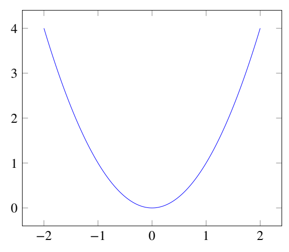
Here are some graphs to help you answer this question:



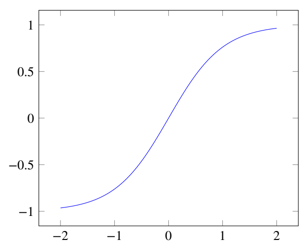
$y = \exp(x)$



$y = \exp(-x)$



$y = x^2$



$y = \tanh(x)$

Which of the following expressions satisfies the desired two properties? Select all that apply.

🟥 $P(s'|s) = \dfrac{\exp(f(s'))}{\sum_{\tilde{s}\in S'} \exp(f(\tilde{s}))}$

☐ $P(s'|s) = \dfrac{\exp(-f(s'))}{\sum_{\tilde{s}\in S'} \exp(-f(\tilde{s}))}$

☐ $P(s'|s) = \dfrac{f(s')^2}{\sum_{\tilde{s}\in S'} f(\tilde{s})^2}$

🟥 $P(s'|s) = \dfrac{\tanh(f(s'))+1}{\sum_{\tilde{s}\in S'}(\tanh(f(\tilde{s}))+1)}$

○ None of the above.

(a) and (d) satisfy both properties, (b) gives more weight to smaller evaluation values, and (c) gives equal weight to large negative and positive values and is not valid if all utilities are 0.

(g) [2 pts] Select all true statements about alpha-beta pruning in game trees.

🟥 It is possible to prune an expectimax game tree with bounded utilities at the leaf nodes.

🟥 It is possible to prune a game tree with three or more agents.

☐ It is always possible to prune a non-zero-sum game tree with three or more agents, all separately maximizing their own utility, as long as the tree contains both maximizer and minimizer nodes.

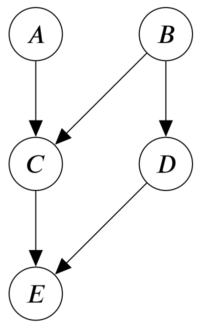
☐ It is only possible to prune a game tree if it contains both maximizer and minimizer nodes.

○ None of the above.

(a) is correct as if we bound the utilities, we can predict what the largest/smallest values are for that node and compare with out alpha value accordingly.
(b) we did this in Project 2.
(c) is incorrect as each maximizer could have increasing utilities left-to-right, while each minimizer could have decreasing utilities left-to-right, thus no nodes can be pruned.
(d) see explanation for option (a)

Q6. [18 pts] Bayes Nets: Easter Island Elections

Matei is an elf on Easter Island who is discontent with the current leadership of Pietru the Rotund. To model the island's politics in preparation for the Easter Elections, Matei has decided to use the following Bayes net.



Each letter represents a vote cast by members of the electoral college: Abby, Brad, Charles, Datsu, and Ershawn. For this question, $+v$ indicates that voter V votes in favor of Pietru the Rotund, while $-v$ indicates they have voted against. All voters must vote (in other words, A, B, C, D, E are binary variables).

Note (1): For the entire question, when computing the size of a factor or table, do not use the sum-to-one constraint to optimize rows out of the table. For example, if we had a table with the two values $P(+x) = 0.7$ and $P(-x) = 0.3$, this table has two rows (even though we could optimize and only store one of the rows, and derive the other row from the fact that both rows sum to 1).

Note (2): For the entire question, when computing the size of a factor or table, assume there is one row for each setting of the variables, i.e. one row for each probability value in the table. For example, if X and Y are binary variables, $P(X, Y)$ has four rows and $P(X \mid Y)$ also has four rows.

(a) [2 pts] Which of these probability tables can be found directly in the Bayes net, without performing any computation? Select all that apply.

[✓ $P(A)$] [✓ $P(C \mid A, B)$] ☐ $P(E \mid C)$

[✓ $P(B)$] ☐ $P(E)$

In the following subparts, Matei wants to compute the probability distribution $P(A \mid +e)$ using variable elimination.

(b) [2 pts] How many factors must Matei consider at the beginning of this process?

5 factors. $P(A)$, $P(B)$, $P(C|A, B)$, $P(D|B)$, $P(+e|C, D)$

(c) [2 pts] Matei joins on B and eliminates B. Which variables are included in the new factor generated after eliminating B? Select all that apply.

For example, if you think the factor generated is $P(A, B, C)$ or $f(A, B, C)$, select options A, B, and C (and nothing else).

[✓ A] ☐ B [✓ C] [✓ D] ☐ $+e$

$P(C, D|A)$ or $f(A, C, D)$

(d) [2 pts] Next, Matei joins on C and eliminates C. Which variables are included in the new factor generated after eliminating C? Select all that apply.

[✓ A] ☐ B ☐ C [✓ D] [✓ $+e$]

$P(+e|A, D)$ or $f(A, +e, D)$

Following both of the above eliminations, we are left with 2 factors.

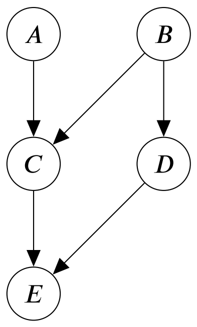
(e) [1 pt] Which variables are included in the **smaller** of the two remaining factors? Select all that apply.

■ A ☐ B ☐ C ☐ D ☐ $+e$

(f) [1 pt] How many rows are in the **smaller** of the two remaining factors?

2 rows. Following both of the above eliminations the two factors we are left with are $P(A)$ and $P(+e|A, D)$, with the former being the smaller. This factor represents the distribution of a binary variable, and thus would have 2 rows.

The Bayes Net, reproduced for your convenience:



(g) [1 pt] What is the **larger** of the two remaining factors?

- ○ One of the original factors from subpart (b).
- ○ The factor generated after eliminating B, in subpart (c).
- ● The factor generated after eliminating C, in subpart (d).
- ○ None of the above.

$f(+e, A, D)$, which we generated after eliminating C.

(h) [1 pt] How many rows are in the **larger** of the two remaining factors?

4 rows. This factor represents the distribution of a fixed evidence variable dependent on two binary variables. As a result, there would be $1 * 2 * 2 = 4$ rows.

(i) [2 pts] If Matei had instead decided to use inference by enumeration to compute $P(A \mid +e)$, how many rows would be in the joint probability table he generates?

Note: We are looking for the size of the table before any hidden variables are marginalized (eliminated).

Note: You can assume Matei deletes all rows inconsistent with the evidence. In other words, we are looking for the size of a table where all rows are consistent with the evidence.

○ 2 ○ 3 ● 16 ○ 32

To perform inference by enumeration, we would create a table representing every combination of our variables. Since we have 5 binary variables, of which one is fixed as our evidence variable ($+e$), we compute $2^4 = 16$.

In the rest of the question, Matei now decides to use prior sampling to estimate the probability that Brad will vote in favor given that Ershawn has voted against, or $P(+b \mid -e)$. The samples he generates are in the table below:

Sample #	A	B	C	D	E
1	$+a$	$-b$	$+c$	$-d$	$-e$
2	$-a$	$+b$	$-c$	$+d$	$-e$
3	$+a$	$+b$	$+c$	$-d$	$+e$
4	$-a$	$-b$	$+c$	$-d$	$-e$
5	$+a$	$-b$	$-c$	$+d$	$+e$

(j) [2 pts] Using prior sampling, what is the estimated value of $P(+b \mid -e)$ that Matei computes?

● 1/3 ○ 2/3 ○ 1/4 ○ 1/5

Similar to rejection sampling, in prior sampling when we consider the evidence $-e$, we estimate $P(+b \mid -e)$ to be the number of samples that have both $+b$ and $-e$ divided by the number of samples with $-e$.

(k) [2 pts] Now, Matei generates two additional samples: $(+a, +b, -d, -c, +e)$ and $(-a, -b, +c, -d, -e)$

Using prior sampling, what is the new estimated value of $P(+b \mid -e)$ that Matei calculates?

○ 2/7 ○ 1/6 ○ 3/5 [✓ 1/4]

The first sample does not contribute to our new estimate since it doesn't have the right evidence observation $-e$. The second does, however it sampled a value of $-b$, which is an outcome we are not interested in.