CS 188 Fall 2015 — Introduction to Artificial Intelligence — Midterm

- You have approximately 80 minutes.
- The exam is closed book, closed calculator, and closed notes except your one-page crib sheet.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a *brief* explanation. All short answer sections can be successfully answered in a few sentences AT MOST.

First name	
Last name	
SID	
edX username	
Name of person on your left	
Name of person on your right	

For staff use only:

Q1.	True/False and Multiple Choice	/20
Q2.	Search Problems	/30
Q3.	Heuristics	/15
Q4.	Fair Play	/15
Q5.	Propositional Logic	/20
	Total	/100

Q1. [20 pts] True/False and Multiple Choice

(a) For each True/False question, circle the correct answer. Missing choices and wrong choices with no explanation are worth zero.

(i) [2 pts] [*true* or *false*] Given any tree, it's always possible to design an admissible heuristic which makes A* search equivalent to uniform cost search.
True. They are equivalent when $h(s) = 0$ for all states s. In that case $f(s) = g(s)$, which is the measure used by uniform-cost search to order the queue.

(ii) [2 pts] [*true* or *false*] Two different search algorithms may give different results on the same constraint satisfaction problem.
True. A CSP may have many solutions, and which one is found first depends on the order in which the space of assignments is searched.

(iii) [2 pts] [*true* or *false*] Improving an agent's initial observation and providing it with a more accurate transition model will cause it to have larger belief states.
False. Generally these have the opposite effect. In the limit, a perfect initial observation and a deterministic transition model lead to a singleton belief state.

(iv) [2 pts] [*true* or *false*] A CSP can only have unary and binary constraints.
False. There are several examples in lecture and in the book of constraints with more than two variables, including cryptarithmetic and sudoku.

(v) [2 pts] [*true* or *false*] Iterative deepening search needs more memory than breadth-first search.
False. Iterative deepening uses depth-first search, which is linear-space. If, however, the iterative deepening search adds an explored set in order to perform graph search, the memory requirements may be comparable to breadth-first search.

(vi) [2 pts] [*true* or *false*] No logical agent can behave rationally in a partially observable environment.
False. As seen in lecture, sensor models can be written in logical form for partially observable environments and a logical agent can maintain a belief state.

(b) Agents and Environments

Select the **better** example of a task environment for each of the following attributes.

(i) [2 pts] **Partially Observable**
○ Image Processing System ● Medical Diagnosis System
For an image processing system, the input image is all that matters, so the environment is fully observable; for a medical diagnosis system, the patient's internal state matters but is not directly observable: tests give partial and noisy information.

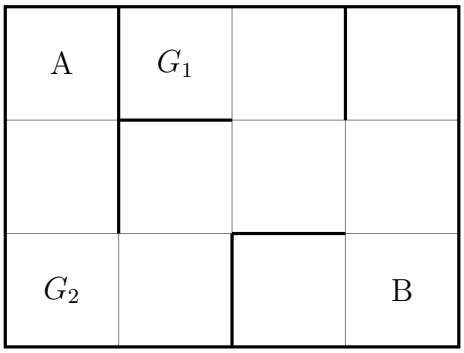
(ii) [2 pts] **Continuous**
● Self-Driving Car ○ Pacman
Driving has both continuous state and continuous actions; Pacman is of course discrete.

(iii) [2 pts] **Stochastic**
● Backgammon ○ Sudoku
The dice make backgammon stochastic.

(iv) [2 pts] **Static**
○ Chat Room Agent ● Checkers
Although technically checkers is semi-dynamic, because it has time controls, it is much less dynamic than a chat room where participants may chime in at any time and where failure to communicate may result in others leaving the room!

Q2. [30 pts] Search Problems

Alice (A) and Bob (B) are in a maze of height M and width N. They know the layout of the maze, including the size, locations of all walls, and locations of the K goal squares $G_1, \ldots, G_K$. K is between 0 and MN, and is a known quantity. Alice and Bob know their own location, as well as the other person's location, at all times. Their goal is to meet up at the *same* goal square in as few time steps as possible.



At each time step, Alice and Bob act simultaneously, and they can both either move to an adjacent square, **or stay put (stop)**. Alice starts in the top left corner of the maze, and Bob starts in the bottom right corner.

(a) [4 pts] Give the state representation for the minimal state space for this problem (i.e., do not include extra information). You should answer for a **general instance of the problem**, not the specific maze shown.

The minimal state representation is two tuples (x_A, y_A) and (x_B, y_B), representing the current locations of Alice and Bob, respectively.

(b) [2 pts] What is the size of this state space?

$(MN)^2$

(c) [3 pts] What is the maximum branching factor for this search problem? Remember that stay put (stop) is also a choice of actions for Alice and Bob.

Both Alice and Bob can take a maximum of five different actions at each time step, so 25. We gave partial credit if it was clear that the branching factors for Alice and Bob were being multiplied.

(d) [6 pts] Suppose we use breadth-first graph search to solve this problem. If we now consider the maze above, of size 3 by 4, which solution will BFS produce? Specifically, which goal square will they end up at, and how many time steps would it take for both Alice and Bob to reach that goal square?

Goal state:

Time steps:

Breadth-first search will find the solution with the smallest number of steps. A step consists of simultaneous moves by Alice and Bob; the solution will be at whichever goal has the lowest maximum distance for Alice and Bob, that is, G_2, and it would take 5 time steps. G_1 is 4 away for Bob but 7 for Alice.

Now suppose that the locations of the K goal squares $G_1, \ldots, G_K$ are unknown, and K is also unknown, though it is still known to be between 0 and MN. When either Alice or Bob enters a square, he/she can determine if it is a goal square, and because the two can communicate with each other, both share that information.

(e) [4 pts] What is the size of the initial belief state, before any percepts are received (i.e., Alice and Bob don't know if they are on a goal)?

Before any percepts are received, any square could be a regular square or a goal square. So the size of the initial belief state is 2^{MN}.

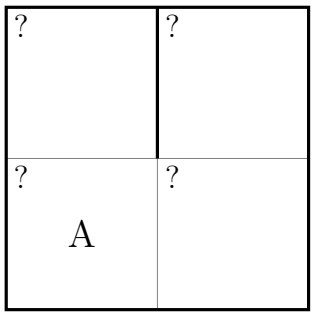
(f) [3 pts] After receiving the initial percept, but before they take any actions, Alice and Bob both determine that their starting positions are not goal squares. What is the size of the new belief state?

After this initial percept, the two starting squares are known to be regular squares, but the remaining $MN-2$ squares could still be goal squares. Therefore, the size of the new belief state is 2^{MN-2}.

(g) [3 pts] If, instead of K being unknown, Alice and Bob received prior knowledge that K was *exactly* one, what would be the size of the initial belief state, before any percepts are received?

Given this prior knowledge, we know that, out of the MN squares, exactly one of them is a goal square. So, the size of our initial belief state is MN.

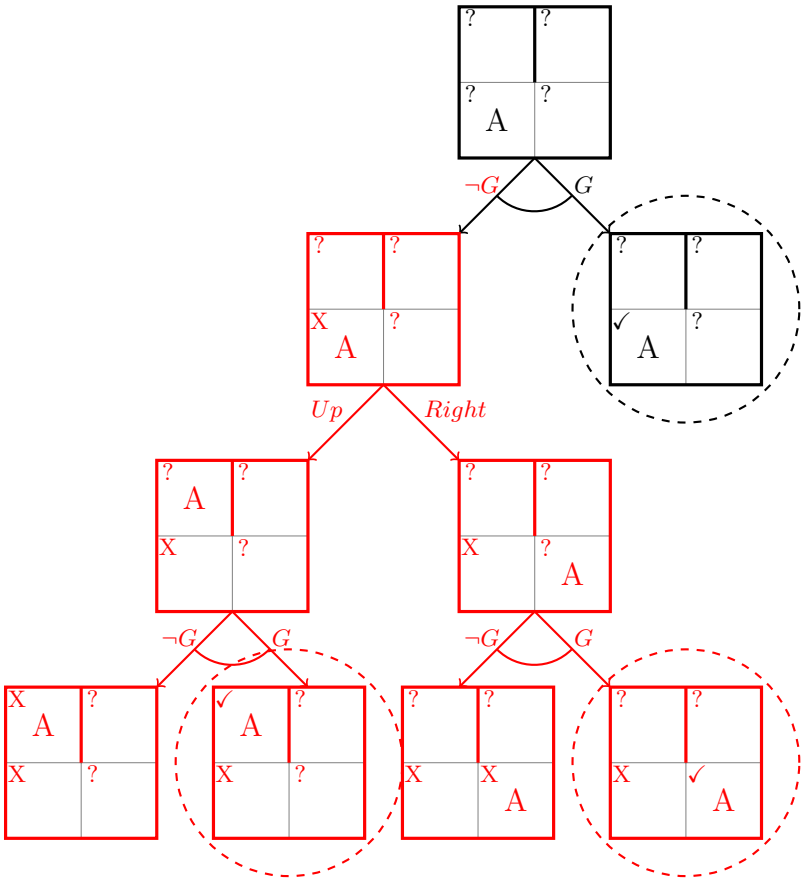
Let's now consider a simplified setting, where Alice is alone in a smaller maze. The locations of the goal squares are still unknown, and Alice has to find a goal square. **Staying in the same square is no longer a valid action.** Assume that this is our starting state, before Alice receives any percepts:



(h) [5 pts] Below is an incomplete And-Or search tree for this new problem. The first layer is complete, and the second layer is partially done. Fill in the missing parts **up to and including the fourth layer of the tree**.

- For the nodes of the tree, draw out the board configurations. A complete board configuration will include the position of Alice, and a "✓", "?", or "X" for every square, denoting whether that square is known to be a goal, still unknown, or known not to be a goal, respectively.
- Label the edges one of [G, $\neg G$, Up, $Down$, $Left$, $Right$] depending on the edge type, and the percept received or the action taken. G indicates a Goal percept and $\neg G$ indicates a No-goal percept.
- Mark AND with an arc across the branches, as shown in the beginning of the tree below. Conversely, you will indicate OR by **not** including an arc across the branches.
- Circle all terminal nodes. One has been circled for you.
- Hint: there will be nine total boxes, including the two we gave you.

This question was generally done well. The biggest problem was people forgetting to mark the actions, observations, and AND-node arcs.


?
?
?
A
?
¬G
G
?
?
X
A
?
?
?
✓
A
?
Up
Right
?
A
?
X
?
?
?
X
?
A
¬G
G
¬G
G
X
A
?
X
?
✓
A
?
X
?
?
?
X
X
A
?
?
X
✓
A

Q3. [15 pts] Heuristics

For parts a, b, c below, consider now the CornersProblem from Project 1: there is a food pellet located at each corner, and Pacman must navigate the maze to find each one.

(a) [5 pts] For each of the following heuristics, say whether or not it is admissible. If a heuristic is inadmissible, give a concrete counterexample (i.e., draw a maze configuration in which the value of the heuristic exceeds the true cost).

- h_1 is the maze distance to the nearest food pellet (if no food pellets remain, $h_1 = 0$).

  [*Admissible* or *Not-Admissible*]
  Eating pellets requires reaching them, which means moving at least this far.

- h_2 is the number of uneaten food pellets remaining.

  [*Admissible* or *Not-Admissible*]
  Eating pellets requires reaching them all, which means at least this many moves.

- $h_3 = h_1 + h_2$

  [*Admissible* or *Not-Admissible*]
  Doesn't quite work because if the first pellet is one step away this heuristic counts 2. Counterexample: In a 2x2 board with pellets in 3 corners, $h_3 = 4$ but $h^* = 3$.

- $h_4 = |h_1 - h_2|$

  [*Admissible* or *Not-Admissible*]
  Since $h_2 \geq 0$, this is never bigger than h_5.

- $h_5 = \max\{h_1, h_2\}$

  [*Admissible* or *Not-Admissible*]
  See textbook. Since both h_1 and h_2 are lower bounds on h^*, their maximum is also.

(b) [2 pts] Pick one heuristic from part (a) that you said was inadmissible; call it h_k. Give the smallest constant $\epsilon > 0$ such that $h' = h_k - \epsilon$ is an admissible heuristic. Briefly justify your answer.

$\epsilon = 1$ works. It must be at least 1, to fix the counterexample given above. And 1 works, because the total cost must be at least the cost to get to any pellet and the cost of eating the remaining $n - 1$ pellets.

(c) [3 pts] We will say that for two heuristics h_a and h_b, $h_a \leq h_b$ if for all possible states x, $h_a(x) \leq h_b(x)$. In this case, we say h_b *dominates* h_a. Fill in the boxes below with all of the heuristics that you said were admissible in part (a) so that all heuristics are to the right of any heuristics they dominate. If two heuristics share no dominance relationship, put them in the same box. You may not need to use all the boxes.

h_1, h_2, and h_4 belong in the leftmost box because they are all incomparable (for each pair, there is some state where one has a higher value than the other and vice versa; h_5 in the next box because it dominates all three.

(d) [2 pts] Let h_i and h_j be two admissible heuristics and let $h_\alpha = \alpha h_i + (1 - \alpha)h_j$. Give the range of values for α for which h_α is guaranteed to be admissible.

[0,1] works; this heuristic is diminated by h_5 and is therefore admissible.

(e) [3 pts] Consider an arbitrary search problem in a graph with start state S and goal state G. Let h^* be the "perfect heuristic," so $h^*(x)$ is the optimal distance from from x to the goal G. Let ϵ be some positive number.

For each of the heuristics h_A, h_B, and h_C, give the range of possible values for the cost of a path that A* tree search could return when using that heuristic. You may write your answers in terms of $h^*(S)$, and ϵ.

Let h_0 be an arbitrary admissible heuristic.

(i) [1 pt] h_A: $h_A = h_0$ for all states except at one unspecified state y, where $h_A(y) = h_0(y) + \epsilon$
$[h^*(S), h^*(S) + \epsilon]$; if y is on the optimal path, it makes the optimal path look ϵ worse than it is, allowing nodes on some suboptimal path to be selected for expansion if they are no more than ϵ worse.

(ii) [1 pt] h_B: $h_B(x) = h_0(x) + \epsilon$ for all states x.
$[h^*(S), h^*(S)]$: as A^* expands states in order of $f = g + h$, adding a constant to h for all nodes doesn't change the order of expansion, so it still returns an optimal solution as before. This might seem counter-intuitive because h_B is "worse" than h_A almost everywhere!

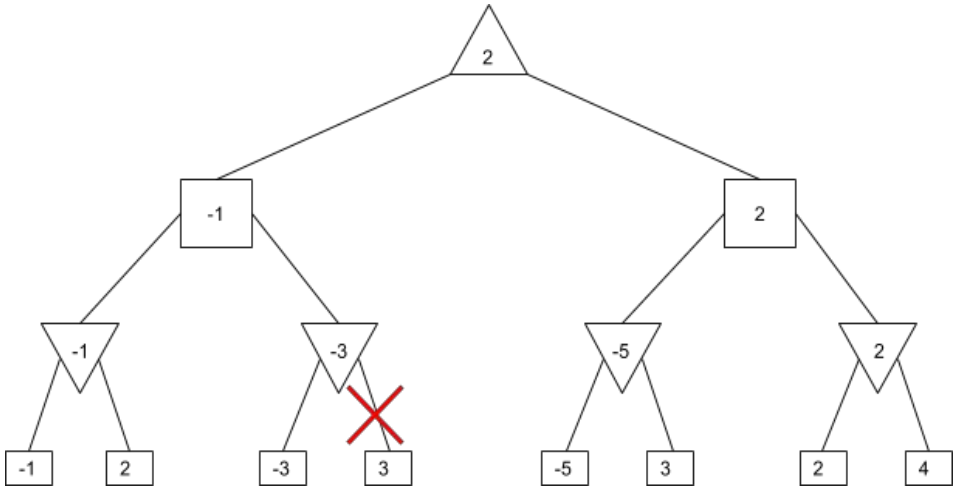
(iii) [1 pt] h_C: $h_C(x) = h_0(x) + \epsilon$ for all states x except the goal state G, where $h_C(G) = h_0(G) = 0$.
$[h^*(S), h^*(S) + \epsilon]$; this one is very tricky and most people missed it—but fortunately it's only worth one point. The reason a suboptimal path can be returned is that the goal can get onto the queue via a suboptimal path first, and it can look *epsilon* better than the last node on the optimal path. For example, consider the graph with S-A = 1, A-G = 1, and S-G = 3, and let $\epsilon = 1.01$.

Q4. [15 pts] Fair Play

Consider a game tree with three agents: a maximizer, a minimizer, and an equalizer. The maximizer chooses the highest score, the minimizer chooses the lowest score, and the equalizer chooses tries to *minimize the absolute value* (i.e. equalizer wants to make the game as close as possible, so it chooses whichever value is closest to zero).

We use an upward-facing triangle to represent a max node, a downward-facing triangle to represent a min node, and a square to represent an equalizer node. the values in the leaves are given from max's point of view.

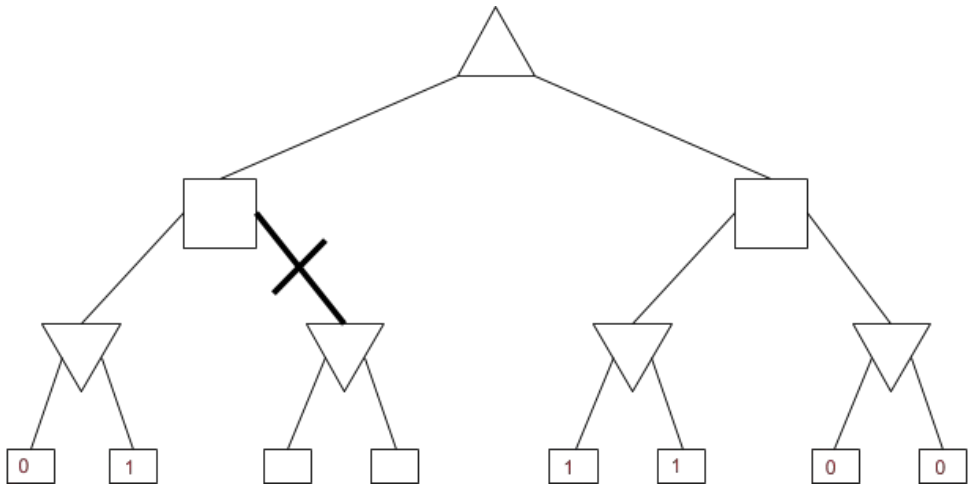
(a) [2 pts] Fill in all values in the game tree below:



(b) [5 pts] In the same game tree above, put an X on the line of all branches that can be pruned, or write "No pruning possible." Assume that branches are explored from left to right.
We can prune this node because once min sees a -3, the min node is worth -3 or lower, so equalizer prefers -1, and the fourth leaf is irrelevant. Note that we cannot prune the sixth leaf: even though the min node is -5 or worse, suppose the seventh and eighth leaves were +10; in that case, if the sixth leaf were -20, the equalizer would prefer the +10 to the -20, causing max to prefer moving right, whereas if in the same case the sixth leaf were -5, the equalizer would prefer the -5 to the +10, causing max to prefer moving left. Hence in some situations the value of the sixth leaf matters. (Note that in min/max trees, pruning can be decided *without* considering the values that subsequent leaves might have.) This reasoning is tricky, so we took off only 1 point for pruning the sixth leaf.

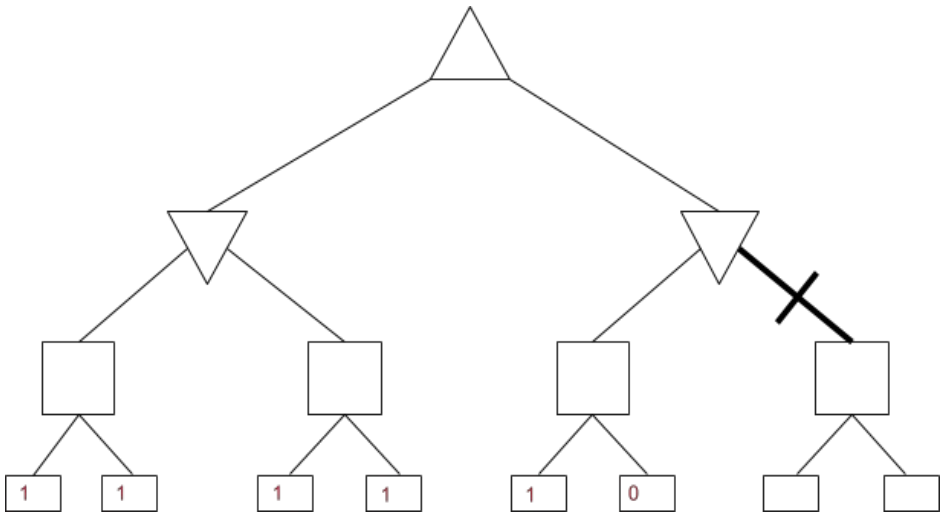
(c) [8 pts] For each of the following game trees, fill in values in the leaf nodes such that only the marked, bold branches can be pruned. Assume that branches are explored from left to right. If no values will allow the indicated nodes to be pruned, write "Not possible." **Be very clear**: if you write "Not possible," we will not look at the values you filled in.

(i) [4 pts] [Hint: what is the *best possible value* from the equalizer's viewpoint?]



0 is the optimal value for the equalizer, so the marked subtree is irrelevant. Note that you also have to put in values for the other leaves so that *no* pruning takes place!

(ii) [4 pts] Note that the order of the players has changed in the game tree below.



Here the pruning is essentially standard alpha-beta pruning because the top two levels of the tree are max and min.

Q5. [20 pts] Propositional Logic

(a) [4 pts] Pacman has lost the meanings for the symbols in his knowledge base! Luckily he still has the list of sentences in the KB and the English description he used to create his KB.

For each English sentence on the left, there is a corresponding logical sentence in the knowledge base on the right (**not necessarily the one across from it**). Your task is to recover this matching. Once you have, please fill in the blanks with the English sentence that matches each symbol.

English	**Knowledge Base**
There is a ghost at (0, 1).	$(C \vee B) \wedge (\neg C \vee \neg B)$
If Pacman is at (0, 1) and there is a ghost at (0, 1), then Pacman is not alive.	$C \wedge \neg D$
Pacman is at (0, 0) and there is no ghost at (0, 1).	$\neg A \vee \neg (B \wedge D)$
Pacman is at (0, 0) or (0, 1), but not both.	D

$A =$ Pacman is alive

$B =$ Pacman is at (0, 1)

$C =$ Pacman is at (0, 0)

$D =$ There is a ghost at (0, 1)

(b) [8 pts] Consider a generic propositional model with 4 symbols: A, B, C, D. For the following sentences, mark how many models containing these 4 symbols will satisfy it.

(i) [2 pts] $\alpha_1 = A$: 8

There are 16 total models; A is true in half of them.

(ii) [3 pts] $\alpha_2 = (C \wedge D) \vee (A \wedge B)$: 7

There are 4 possible worlds where $C \wedge D$ is true, 4 where $A \wedge B$ is true, and one where they are both true, so the total is 7.

(iii) [3 pts] $\alpha_3 = (A \vee B) \Rightarrow C$: 10

There are 4 possible worlds where $A \vee B$ is false so the implication is satisfied for all values of C. In the remaining 12 possible worlds, C is true is 6 of them. So there are 10 possible worlds

(c) The DPLL satisfiability algorithm is a backtracking search algorithm with 3 improvements: PURE-SYMBOLS, UNIT-CLAUSES, and EARLY TERMINATION. In this question, we'll ask you to relate these improvements (if possible) to more general CSP techniques used with backtracking search.

(i) [2 pts] The PURE-SYMBOL technique finds a propositional symbol that only occurs with the same "sign" throughout the expression and assigns it the corresponding value. Such symbols are called pure. In the following CNF expression, which symbols are pure and what value does this process assign to those symbols?

$$(C \vee D) \wedge (C \vee \neg A) \wedge (\neg D \vee A) \wedge (\neg B \vee A)$$

pure symbols(s): B, C

value(s) assigned: B = False, C = True

(ii) [2 pts] Which of the following CSP techniques is equivalent to the PURE-SYMBOLS technique when applied to SAT for CNF sentences:

- ○ MINIMUM-REMAINING-VALUES
- ○ FORWARD-CHECKING
- ● LEAST-CONSTRAINING-VALUE
- ○ BACKTRACKING
- ○ No equivalent CSP technique

Assigning a value to a pure literal can only cause clauses to become true, so it will impose 0 additional constraints on the remaining literals. Notice that a pure symbol (except in a unit clause) can still take on either value, so this is not an MRV choice.

(iii) [2 pts] The UNIT-CLAUSE technique finds all clauses that contain a single literal and assigns values to those literals. Which of the following CSP techniques is equivalent to the UNIT-CLAUSE technique when applied to SAT for CNF sentences:

- ● MINIMUM-REMAINING-VALUES
- ○ FORWARD-CHECKING
- ○ LEAST-CONSTRAINING-VALUE
- ○ BACKTRACKING
- ○ No equivalent CSP technique

A variable in a unit clause can only be a assigned to a single value in a satisfying assignment, otherwise that clause would be false. Thus, there is only 1 remaining value for such literals.

(iv) [2 pts] DPLL performs early termination in two steps: SUCCESS-DETECTION and FAILURE-DETECTION. First, SUCCESS-DETECTION checks to see if *all* clauses evaluate to true. If so, the sentence is satisfiable and any unassigned propositional symbols can be assigned arbitrarily. Next, FAILURE-DETECTION checks if *any* clause evaluates to false. In this case the sentence is unsatisfiable and this branch of the search can be pruned. How does this strategy relate to the early termination strategy used by the general BACKTRACKING algorithm?

- ○ BACKTRACKING does both SUCCESS-DETECTION and FAILURE-DETECTION
- ○ BACKTRACKING does only SUCCESS-DETECTION
- ● BACKTRACKING does only FAILURE-DETECTION
- ○ BACKTRACKING does neither

In the backtracking CSP algorithm from the book, early termination is triggered whenever the domain for a variable is empty. This is exactly the case when a clause has been reduced to false, so FAILURE-DETECTION is an application of this technique to SAT. Early detection of success is possible in SAT because if *any* literal in a clause is true, the clause is satisfied, and no further assignments to the other variables can unsatisfy it. This is *not* true in general CSPs: a partial assignment may not violate any constraints, but it is never (in general) certain to become a solution because further assignments may violate some constraints. Hence early success detection is not possible in general CSPs.