CSE 352	**Artificial Intelligence**	**Handout A4**
Spring 2003	**Annie Liu**	**Feb. 25, 2003**
Stony Brook	**Assignment 4**	**No due date**

This assignment gives exercise problems for Exam 1. Solutions are provided and the assignment will not be graded. The exam will be open your own handwritten notes, and open textbook, but nothing else.

Problem 1. An intelligent agent

An agent is placed in an environment containing two slot machines,each of which costs \$1 to play. The expected payoffs are p and q, and the agent can play each machine as many times as it likes. Describe, in qualitative terms, how a rational agent should behave in each of the following cases:

(a) $p = 1.5$ and $q = 0.8$.
(b) $p = 1.5$ and q is unknown.
(c) $p = 0.95$ and q is unknown.
(d) p and q are unknown.

This is a simple-looking question that can bring up a lot of interesting issues. You are not expected to have noticed and resolved all of the issues, but you should have noticed some of them. The answer given is somewhat precise, but it's fine to use purely qualitative terms as long as the main ideas of the solution are present.

The first thing is to identify a performance measure for the agent; without one, "rational" means nothing! The performance measure evaluates the environment history, which in this case is just the sequence of payoffs obtained. Let r_t be the payoff obtained at time t, and let U be the performance measure.

There are several choices for U, but you need to make one! First, we need to decide whether the agent's lifetime is finite or infinite. If it's finite, we can just use the sum of payoffs:

$$U_1(r_1, ..., r_T) = r_1 + r_2 + ... + r_T = \sum_{t=1}^{T} r_t$$

If the agent's lifetime is unbounded, we have to be more careful because comparing infinities is difficult! One way to force the total payoff to be finite is to discount future payoffs. (It's as if the agent is forgoing interest on payoffs received in the future instead of immediately.) The discount factor γ is between 0 and 1, and

$$U_2(r_1, ..., r_\infty) = \sum_{t=1}^{\infty} \gamma^t * r_t$$

If $\gamma < 1$, then payoffs far in the future are essentially worthless because γ is close to 0. Another way to deal with infinite lifetimes is to look at the average payoff per time step. This is probably the most natural measure for this problem, because an agent can maximize this measure only if it works out which lever to pull. The technical definition is as follows:

$$U_3(r_1, ..., r_T) = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} r_t$$

Strictly speaking, the limit may not exist for certain kinds of agent programs, but we'll ignore this issue. Another aspect we need to fix is the set of available actions. In the case where both machine have payoffs<1, the agent may want to have the choice of doing nothing. We'll assume this choice is available; the answers to (c) and (d) change somewhat if not.

The final aspect of the problem that needs to be pinned down is the agent's prior knowledge; rational agent always chooses the behavior that optimizes its performance measure given what it knows. In particular, when one or both payoffs are unknown, the agent's prior beliefs about the payoffs do matter. For example, the agent may believe that whenever it's presented with two slot machines with unknown payoffs, one of those machines has payoff 10 and the other has payoff 20; or, more reasonably, it may believe that all unknown slot machines have payoff less than 1. Another important set of beliefs concern how "reliable" the payoffs of a given machine are; for example, a machine with expected payoff 1.5 might pay 1.5 every time, or 3 every other time, or 150 every 100th time. Fortunately, if we're willing to take the long view (i.e., large T in U_1, γ close to 1 in U_2, or just use U_3), the agent's initial beliefs will "soon" be overcome by its actual experience.

(a) Each of the three performance measures above is maximized by always choosing p.

(b) Taking the long view, it makes sense for the agent to explore its environment, i.e., try q at least a few times. From its experience, it can obtain an estimate $\hat{q}$. Given $\hat{q}$, the agent will have a better idea of whether the real $q > p$ or not. The first point to make is that it will never be absolutely sure–even if q loses 1000 times in a row, it may still be the case that $q = 2$ and the agent is unlucky.

For U_1 and U_2, the agent cannot afford to explore for an arbitrarily long time, so it will have to decide on a stopping criterion, after which it always chooses the lever that appears best. The amount of exploration in this case will depend on how close $\hat{q}$ is to p and how reliable the payoffs are. One might think that the closer they are, the more exploration is needed to get an accurate decision as to which is better; but it's a tricky tradeoff because the closer they are, the less difference it makes which one chooses.

For U_3, an agent that correctly identifies the best lever will do better in the long run than one that does not; hence, we want a policy for exploration that will eventually settle on the best lever. One idea is to choose randomly between the two levers, such that the probability of picking lever p is proportional to the probability that p is the best lever, given the experience so far. There are various other schemes that have the same effect.

(c) If the agent has the option of doing nothing, then the information it needs is whether $q > 1$. There is no reason ever to play p. So the analysis is essentially identical to part (b), except that we are comparing q to the option of doing nothing (payoff 1) instead of doing p. If the agent must play one machine or the other, then the analysis is identical to (b).

(d) This is like (b), except that now we have to try both machines to obtain estimates $\hat{p}$ and $\hat{q}$; the amount of exploration again depends on how different the estimates are and their variances. With the option of doing nothing, we also need to compare the estimates to 1.0 and eventually stop exploring if both seem to be worse than 1.

Problem 2. Agent performance, environment, actuators, and sensors

For each of the following agents, develop a PEAS description of the task environment and characterize the environment along the six dimensions discussed in lecture:

(a) Robot soccer player.
(b) Spelling and grammar assistant for Word.
(c) Autonomous Mars rover.
(d) Mathematician's theorem-proving assistant.

Some variations in answers are also acceptable, but you need to provide good justifications.

(a) Performance measure: A natural measure is the team's goal differential: a positive score indicates that the player's team is winning, and a negative score indicates that the player's team is losing.

Environment: A soccer-playing agent plays on a field with many opponent-agents and many teammate agents for the duration of a "game". The environment is partially observable (one does not know the other's strategy, even if all players have perfect vision of the entire field), non-deterministic, sequential at the level of actions (episodic at the level of a game), dynamic, continuous, and multi-agent.

Actuators: A robot soccer player probably has wheels, tracks, or legs for locomotion, and some sort of mechanism (a bumper or a plunger) for kicking the ball. Agents may also have transmitters (radio or infrared, for example) for communicating with their teammates.

Sensors: Agents should have some means of sensing the ball and other players; for example, cameras, sonar or infrared range finders. They may also have radio or infrared receivers for communication.

(b) Performance measure: A spelling and grammar assistant should be measured by the number of misspelled words or ungrammatical sentences it identifies and/or corrects. A misidentification penalty may also be appropriate.

Environment: The assistant may operate either offline or online. An offline checker, which processes an entire document in a known language as a batch, has a fully observable, deterministic, episodic, static, discrete, single-agent environment.

An online checker, which processes new keystrokes or words as they arrive, has either a partially observable (if it does not wait for the user to finish a word or sentence) or fully observable, nondeterministic (because it does not know what the user-agent will do next), episodic (one correction counting as an episode), static (the document does not change between keystrokes, or "moves" by the user-agent), discrete, multi-agent environment.

Actuators: These depend on the formulation. An agent may be able to highlight or underline suspicious text, to change suspicious text, or to alert the user by other means. Most of these actuators will take the form of function calls on other programs (rendering or sound libraries, or Word itself).

Sensors: Again, these depend on the formulation. Some assistants may sense keystrokes directly; others may only have facilities for scanning an existing file.

(c) Performance measure: A Mars rover may be tasked with maximizing the distance or variety of terrain it traverses, or with collecting as many samples as possible, or with finding life (for which it receives 1 point if it succeeds, and 0 points if it fails.) Criteria such as maximizing lifetime or minimizing power consumption are (at best) derived from more fundamental goals; e.g., if it crashes or runs out of power in the field, then it can't explore.

Environment: The environment (the Martian surface) is partially observable, non-deterministic, sequential, dynamic, continuous, and (most likely) single-agent. If a rover must cooperate with its mother ship or other rovers, or if mischievous Martians tamper with its progress, then the environment gains additional agents.

Actuators: The Mars rover Sojourner had motor-driven wheels for locomotion, a scoop for collecting samples, and a radio transmitter for communication.

Sensors: Sojourner also had chemical sensors for analyzing the geo- and (exo)bio-chemistry of samples, cameras, and a radio receiver.

(d) Here again, a variety of answers are possible, depending on how one interprets "assistant". For example, an automated assistant might verify proofs generated by its owner or might generate new proofs on its own.

Performance measure: A theorem-proving assistant may be measured by its owner's success, with 1 point per publication, 10 points for achieving tenure, and 1000 points for a Fields Medal. A more useful approach would be to design an agent that maximizes the numbers of errors it finds in a submitted proof, or maximizes the number of results it finds given some set of axioms.

Environment: One environment for a theorem-proving assistant might consist of the "givens" (axioms and theorems that the mathematician provides to the assistant, or rules to be applied at each step of a proof the assistant checks) and the results that the assistant might find. This environment is fully observable (the mathematician won't expect his assistant to use a result that the assistant hasn't been told about), deterministic, either episodic or sequential (depending on the goal the mathematician chooses; a proof-checking assistant sees episodes, while a more general theorem-prover might prove results forever from an initial set of axioms), static, discrete (each theorem is a finite list of symbols), and single-agent.

Actuators: A theorem-proving assistant has means of generating output (true or false for a proof checker, or lists of theorems for a theorem-prover) and manipulating symbols to create new results from old.

Sensors: The assistant must be able to read statements provided by the mathematician.

Problem 3. Problem formulation

Give the initial state, goal test, successor function, and cost function for each of the following. Choose a formulation that is precise enough to be implemented.

(a) You are to color a planar map using four colors, where two adjacent regions must have different colors.

(b) A 3-foot-tall monkey is in a room where some bananas are suspended from the 8-foot ceiling. He would like to get the bananas. The room contains two stackable, movable, climbable 3-foot high crates.

(c) You have a standard phone book, and you want to know the phone number of a given person.You can turn pages, you can scan a page for a name, and you know the alphabet.

(d) You have a standard phone book, and you want to know the name of a person who has a given phone number. You can turn pages, and you can scan a page for a number.

(e) You have a program that outputs the message "illegal input line" when fed a certain input file.You want to determine what line in the file is causing the problem.

(f) You have three jugs measuring 12 gallons, 8 gallons, and 3 gallons, and a water faucet. You need to measure out exactly one gallon.

(a) This is a version of the graph-coloring problem (a standard CSP) discussed in lecture.

Initial state: An uncolored map.

Goal test: Each region on the map is shaded in one of (at most) four colors, and no region shares a color with its adjacent regions.

Successor function: For a partially-colored map, the successor of a particular partially-colored map, on which an agent colors a new region, is a copy of the map with the new region colored in.

Cost function: As discussed in lecture, if the cost of coloring a region is independent of the color used, all paths to the goal will have the same cost. No solution is preferred over any other, so let the cost of coloring a state be 1.

(b) Initial state: Two crates are sitting on the floor at particular locations, the bananas are suspended from the ceiling, and the monkey is hopping around on the floor at a particular location (as monkeys do).

Goal test: The monkey is holding the bananas.

Successor function: The monkey can move from one place on the floor to another. If a monkey is holding nothing and is at a crate, it can pick it up. If the monkey is holding a crate it can place the crate on the floor, or on top of another crate at that place. If a monkey is standing on the same level as a crate, then it can climb that crate. If the monkey is standing on two stacked crates at the location of the bananas, then it can grab the bananas.

Cost function: Assign unit cost to each step, so the monkey minimizes the number of actions he has to perform to attain the bananas. A more specialized cost function may assign different costs to different actions, depending on the monkey's exertion.

(c) Initial state: The phone book is open to a random page n, and the target lies on some page j. Denote the state by a pair indicating the page number and whether or not the page contains the target (true, false, or unknown). The initial state is (n, unknown).

Goal test: The second component of the state is true.

Successor function: scan(n, $\cdot$) returns (n, true) if the page contains the target and (n, false) otherwise. Turn$^+(n)$ returns (min($n+1$, n_{max}, unknown), and Turn$^-(n)$ returns (max(1, $n-1$, unknown). Would also be good to allow the agent to turn k pages at once, allowing for a form of binary search (if we augment the state to include the previous page). A more efficient search results if we make it illegal to go in the "wrong" direction.

Cost function: If the phone-book agent can turn many pages at once, one can assign the same cost for turning k pages at once as for turning one page at a time. Otherwise, one may assign a cost to each Turn$^\pm$. In either case each scan operation should incur some cost.

(d) This is more or less identical to the previous problem. However, a phone book is arranged by names rather than numbers, so there's no way to decide which is the "wrong" direction or how many pages to turn.

(e) Initial state: The agent has a buggy file f and does not know the broken line l.

Goal test: The broken line l is known.

Successor function: The agent can insert and delete lines from its current file, resulting in a new file f+, or can submit f to the verifier program, which returns either nothing or "illegal input line".

Cost function: One approach is to charge the agent 1 point for each submission of a file.

(f) Initial state: All three jugs are empty: $G_1 = G_2 = G_3 = 0$.

Goal test: Some jug contains exactly one gallon.

Successor function: Applying the action transfer to jugs i and j with capacities C_i and C_j and containing G_i and G_j gallons of water, respectively, leaves jug i with $max(0, G_i - (C_2 - G_2))$ gallons of water and jug j with $min(C_2, G_1 + G_2)$ gallons of water. Applying the action fill to jug i leaves it with C_i gallons of water.

Cost function: Charge one point for each gallon of water transferred and each gallon of water filled.

Problem 4. Exhaustive and other kinds of search

Consider a state space where the start state is number 1, and the successor function for state n returns two states labeled $2n$ and $2n + 1$.

(a) Draw the portion of the state space for states 1 through 15.

(b) Suppose that the goal state is 11. List the order in which nodes will be visited for breadth-first search, depth-limited search with depth limit 3, and iterative deepening search.

(c) Can you apply best-first search to this problem? What would be a good heuristic? List the order in which nodes are visited in searching for the goal of 11 using your heuristic.

(d) Would bidirectional search be appropriate for this problem? If so, describe how it would work. What is the branching factor in each direction of the bidirectional search? Does this suggest a reformulation of the problem that would allow you to solve the problem of getting from state 1 to a given goal state with almost no search at all?

(a) The state space is illustrated by a full binary tree (with implicit edges from each node to the two nodes below it on the next level) of depth 3 with nodes at level i numbered from 2^i to $2^{i+1} - 1$.

```
                  1
          2               3
      4       5       6       7
    8   9   10  11  12  13  14  15
```

(b) Breadth-first search: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.
Depth-limited search with depth limit 3: 1, 2, 4, 8, 9, 5, 10, 11.
Iterative deepening search:1; 1, 2, 3; 1, 2, 4, 5, 3, 6, 7; 1, 2, 4, 8, 9, 5, 10, 11.

(c) Represent the current path by a binary representation that assigns 1 to the initial state, and appends 0 for every left branch followed and 1 for every right branch followed in the search tree. Let $h(n)$, where n is at depth k, denote the sum of the negated bitwise XOR of n in this representation and the k most significant bits of $1011_2 = 11$. (The XOR of bits $b1$ and $b2$ is 0 if $b1 = b2$, and 1 otherwise. 1011 is the binary representation for 11.) Thus, for $5 = 101_2$, $h(n) = 0$. One is easily satisfied that this heuristic is admissible–the estimated cost-to-goal is 0 for all n on the path to the goal, and greater than 0 (but not infinite) otherwise. This results in a search path of 1, 2, 5, 11, as we would hope.

(d) The forward direction of bidirectional search has a branching factor of 2, while the backward direction has a branching factor of 1, and the predecessor of n is given by $\lceil n/2 \rceil$. This suggests that backward search alone can give the path to a goal state g in time $\log_2 g$.

Problem 5. Heuristics

Come up with heuristics for the following problems. Explain whether they are admissible, and whether the state spaces contain local maxima with your heuristic:

(a) Algebraic equation solving (e.g., "solve $x^2y^3 = 3 - xy$ for x").

(b) Path planning in the plane with rectangular obstacles.

(a) The goal is to isolate the unknown, x, by itself on the left-hand side of the equation. A good heuristic is thus the number of terms other than x on the left plus the number of xs on the right hand side. For example, $x^2y^3 = 3 - xy$ gets a score of 4: one each for the 2, y and 3 on the left, and one for the x on the right. Similarly, $x = (-y + \sqrt{y^2 + 12y^3})/(2y^3)$ gets a score of 0 and is in fact a solution to this problem.

This is admissible because each operation on an equation (adding a term to both sides; multiplying both sides by a term, etc.) reduces at most one of these.

There are local maxima because, for example, you may do a lot of work to eliminate terms on the left side, but end up with a lone x on the right side that is deeply embedded and hard to eliminate. To deal with this, you may want to assign xs on the right a score equal to their embedding depth. This is admissible because any operation on the right-hand side will reduce the embedding depth of an x by at most one.

(b) A reasonable heuristic is the straight-line distance to the goal point. This is admissible because the straight-line distance is always less than or equal to the best path. One can construct local maxima in this domain: consider a long and thin rectangular "wall" standing between the agent and the goal, where the agent must move further away from the goal in order to travel around the wall.

Problem 6. A constraint satisfaction problem and an alternative

Consider the problem of constructing crossword puzzles: fitting words into a grid of intersecting horizontal and vertical squares. Assume that a list of words (i.e., a dictionary) is provided, and that the task is to fill in the squares using any subset of this list. Go through a complete goal and problem formulation for this domain, and choose a search strategy to solve it. Specify the heuristic function, if you think one is needed.

Problem formulation: Initial State: A grid of all empty squares, and a dictionary with all the words.

Goal: Every square (except the black ones) contains a letter, and every word slot (string of consecutive horizontal or vertical squares) contains a word from the dictionary. We might also want the constraint that no word is used more than once.

Successor function: There are many choices here. They break down roughly into word-at-a-time and letter at-a-time. We will go with word at a time: you can choose any word from the dictionary and place it in any word slot of the same length, as long as there are no contradictory letters already in place there. The resulting state has one more word slot filled in and (if we have the word-used-only-once constraint) one less word in the dictionary.

Path cost: No preference here, so each operator costs 1.

Search strategy: A constraint satisfaction algorithm is quite natural. The variables are the word slots, the domain of each variable is the list of words of appropriate length in the dictionary, and the constraints are that no word may be used twice, and that if two word slots intersect, they must have the same letter in the intersecting square.

A* is also a reasonable approach for adding words to the puzzle.

Heuristic: There are many choices here; the most-constrained-variable heuristic is an important one, but it is expensive to calculate exactly when the dictionary is large. That is, we don't want to spend a lot of time going through 50,000 words to determine exactly how many fit into each slot. We can get an estimate by choosing k words at random and seeing how many of them fit into a slot. We can do the same with the least-constraining-value heuristic: of the first k candidate words to fill a slot, choose the one that imposes the least constraints on the intersecting slots.

A* search requires a different sort of heuristic. If we know in advance where the black squares will be, then we can assign an estimated cost of 1 to every unfilled word slot. This is admissible because a finished puzzle has no unfilled word slots, and a puzzle with k unfilled word slots requires at least k operations before achieving the goal state.

Problem 7. Improving heuristics

A knight moves on a chess board two squares up, down, left, or right followed by one square in one of the two directions perpendicular to the first part of the move. (I.e., the move is L-shaped.) Suppose the knight is on an unbounded board at square (a, b) and we wish to move it to square (c, d) in the smallest number of moves. (For example, to move from (0, 0) to(1, 1) requires two moves.)

(a) Explain how to decide whether the required number of moves is even or odd without constructing a solution.

(b) Design an admissible heuristic function for estimating the minimum number of moves-required; it should be as accurate as you can make it. Prove rigorously that your heuristic is admissible.

(a) If you imagine moving a knight around on a chess board, you will quickly see that each move changes the color of the square the knight is on. This parity property is easy to prove. Hence, if the goal square is a different color from the start square (i.e., $|c-a|+|d-b|$ is odd) then an odd number of moves are required, otherwise an even number of moves are required.

(b) Without loss of generality, we will assume the knight starts at square (0, 0) and the goal square is (x, y). The following three heuristics are all admissible for the knight problem:

* Euclidean Heuristic: $\lceil\sqrt{x^2+y^2}/\sqrt{5}\rceil$. This is very similar to the straight-line distance in path planning, except that we take into account that the knight's "step size" is $\sqrt{5}$. This heuristic is th e exact cost-to-goal in the relaxed problem where the knight can only move in one direction.

* Manhattan Distance: $\lceil(x+y)/3\rceil$. Here we use the intuition that the knight must move x + y squares, and that it can move 3 squares per move. This heuristic is the exact cost-to-goal in the relaxed problem where the knight is no longer restricted to moving in an L-shape, but can move to any square three squares away from the starting square.

* x or y Distance: $\lceil x/2\rceil$ or $\lceil y/2\rceil$. Here we use the intuition that in any one knight move, it can move at most 2 squares in one dimension. This is the exact solution to the relaxed problem where the goal specifies only the x or the y coordinate.

In addition, we can make even better (and still admissible) heuristics by noting two things:

* Max Heuristic: We can take two admissible heuristics and create another admissible heuristic by taking the max of the two input heuristics. This new heuristic will perform at least as well as the input heuristics, and possibly better. A better heuristic than those above is the one which outputs the max over all of them.

* Parity Correction: Any answer obtained by any admissible heuristic can be "rounded up" to the correct parity, according to the rule in part (a). This preserves admissibility.

Problem 8. Game-playing

The following diagram shows a binary game tree with chance nodes and with evaluations as the leaves. There are implicit edges from each node to the two nodes below it on the next level, and each of the 4 edges below chance nodes has a chance of 1/2.

```
max                  ?
chance      ?                  ?
min     6       ?          ?       8
             4  7     -2  -10
```

(a) Copy the diagram and fill in the values of all internal nodes including the root, and indicate the best move at the root.

(b) In this question we will consider a version of the alpha-beta for game trees with chance nodes. Pruning in such trees is possible if there are known bounds on the values of the leaf nodes. Suppose the tree above is evaluated left-to-right order, and that all leaf values fall in the range [-10,+10]. Explain in words how these bounds might enable some leaf nodes to be pruned. On your diagram, circle those leaves that need not be evaluated.

(a) This just requires maxing at the max nodes, mining at the min nodes, and averaging at the chance node. The MAX player will take the left-hand branch.

(b) Alpha-beta pruning works whenever it is possible to show that the node about to be evaluated or expanded cannot possibly affect the top-level move choice. Without bounds on leaf values, the value of a chance node is also unbounded until all its children have know values, so pruning cannot occur. With bounds, it can be the case that even if the remaining unevaluated leaves have the worst/best possible value, the chance node's value would not be high/low enough to cause a change in move. In the given tree, once the 2-leaf is found, then main node is worth at most -2, so the chance node is worth at most 4 (because it's right child cannot be worth more than 10), so the top-level choice can not be affected by the values of either of the leaf nodes labeled -10 and 8.

Problem 9. Propositional logic

Given the following, can you prove that the unicorn is mythical? How about magical? Horned? Base your answers on the formal definition of entailment in propositional logic, rather than energetic handwaving.

> If the unicorn is magical, then it is immortal, but if it is not magical, then it is a mortal mammal.
> If the unicorn is either immortal or a mammal, then it is horned.

Recall that a knowledge base entails a proposition if that proposition is true in all models of the knowledge base. One reliable way to determine this is to construct a truth table. Let G denote "magical", H denote "horned", M denote "mammal", R denote "mortal", and Y denote "mythical".

H	M	R	G	$G \Rightarrow \neg R$	$\neg G \Rightarrow R \wedge M$	$\neg R \vee M \Rightarrow H$	KB
F	F	F	F	T	F	F	F
F	F	F	T	T	T	F	F
F	F	T	F	T	F	T	F
F	F	T	T	F	T	T	F
F	T	F	F	T	F	F	F
F	T	F	T	T	T	F	F
F	T	T	F	T	T	F	F
F	T	T	T	F	T	F	F
T	F	F	F	T	F	T	F
T	F	F	T	T	T	T	T
T	F	T	F	T	F	T	F
T	F	T	T	F	T	T	F
T	T	F	F	T	F	T	F
T	T	F	T	T	T	T	T
T	T	T	F	T	T	T	T
T	T	T	T	F	T	T	F

We see that the KB has 3 models, and those are the rows we need to check.

Magical: the penultimate row of the table has KB true but G false, so "magical" is not entailed.

Mythical: this is not mentioned in the KB, so could have any truth value (including false) in the models of the KB, and hence its truth cannot be determined.

Horned: this is true in all three rows where KB is true, so it is entailed.

Problem 10. Propositional models

Consider a world in which there are only five propositions, A, B, C, D, and E. How many models are there for the following sentences?

(a) $A \Rightarrow B$
(b) $A \vee B \vee C \vee D \vee E$
(c) $A \wedge B \wedge C$

(a) There are eight models corresponding to the eight possible truth values for C, D and E. There are three models corresponding to the possible truth values for A and B. Thus, there are twenty-four models.

(b) There are 2^5 possible assignments to the five variables, only one of which causes the statement to be false. Thus, there are $2^5 - 1$, or 31, models.

(c) There are four models possible corresponding to the four possible truth values for D and E.

Problem 11. Propositional backward chaining

Consider the following algorithm for backward chaining in propositional logic (remember that definite clauses include atoms as a special case, since these are definite clauses with 0 negative literals, i.e., an empty antecedent).

```
function PL-BC-Entails?(KB, goals) returns true or false
   inputs:  KB, a knowledge base
            goals, a list of symbols, initially containing only the query
   if goals is empty then return true
   q ← First(goals)
   for each clause (p1 ∧ ... ∧ pn ⇒ q) in KB
      if PL-BC-Entails?(KB, [p1, ..., pn[Rest(goals)]) then return true
   return false
```

(a) Show the sequence of values of the goals variable in the recursive calls to PL-BC-Entails? for the following KB, given Q as the original query. For example, the value on the initial call is $[Q]$, and on the next call is $[P]$.

$$\begin{aligned} P &\Rightarrow Q \\ K \wedge M &\Rightarrow P \\ B \wedge K &\Rightarrow M \\ A \wedge P &\Rightarrow K \\ A \wedge B &\Rightarrow K \\ &A \\ &B \end{aligned}$$

Continue up to the point where looping occurs. You should assume that clauses $(p_1 \wedge ... \wedge p_n \Rightarrow q)$ are processed in the order given in the KB.

(b) Assuming that a looping subgoal is detected and fails, continue with the rest of the process until the algorithm terminates. Mark with an asterisk those calls that perform unnecessary work redoing past successful or failing calls.

(a) For an initial call of Q, the goals are

$$\begin{gathered} [Q] \\ [P] \\ [K, M] \\ [A, P, M] \\ [P, M] \end{gathered}$$

At this point, P is still pending on the stack and is now the current goal, so we have hit a loop.

(b) From the top,

$[Q]$	
$[P]$	
$[K, M]$	
$[A, P, M]$	
$[P, M]$	Loop (P on stack), fails, backtrack to second K-rule
$[A, B, M]$	*(A already known)
$[B, M]$	
$[M]$	
$[B, K]$	*(B already known)
$[K]$	*(K already known)
$[A, P]$	*
$[P]$	*Loop (P on stack, fails, backtrack to second K-rule
$[A, B]$	*
$[B]$	*
$[]$	Success!

Note that the "known" facts A and B do require extra work every time they are "re-proved", so we have marked them with an asterisk, but the extra work is not huge.

Problem 12. First-order models

Answer the following questions and explain your answer in terms of models.

(a) Does a knowledge base containing just two sentences, $P(A)$ and $P(B)$, entail $\forall x P(x)$?
(b) Is the sentence $\exists x, y \; x = y$ valid?
(c) Is the sentence $\forall x, y \; \neg(x = y)$ satisfiable?

(a) Entailment in first-order logic requires that all models for the knowledge base also entail the query under all interpretations. One can easily construct interpretations of the knowledge base (even in a two-object universe) in which $\forall x P(x)$ does not hold. For example, if A and B both refer to the first object (for which P is true) and P is false for the second object, then $P(A)$ and $P(B)$ are true but $\forall x P(x)$ is false. Therefore, the knowledge base does not entail $\forall x P(x)$.

(b) First-order logic requires that all interpretations have at least one object. Since every object is equal to itself, we know that the sentence is true in every possible interpretation. We conclude that the sentence is valid.

(c) To be satisfiable, there must be some model in which the above statement is true. If we consider x and y to be the same object, then there is no model in which the statement is true.

Problem 13. First-order logic

Represent the following sentences in first-order logic, using a consistent vocabulary (which you must define):

(a) Some students took French in Spring 2001.
(b) More than one student took Greek in Spring 2001.
(c) Every person who buys a policy is smart.
(d) No person buys an expensive policy.
(e) There is an agent who sells policies only to people who are not insured.
(f) There is a barber who shaves all men in town who do not shave themselves.
(g) A person born in the UK, both of whose parents are UK citizens or UK residents, is a UK citizen by birth.
(h) A person born outside the UK, one of whose parents is a UK citizen by birth, is a UK citizen by descent.

We choose the following vocabulary:

$Student(x)$	:	"x is a student."
$Person(x)$	:	"x is a person."
$Agent(x)$	:	"x is an agent."
$Barber(b)$	:	"b is a barber."
$Man(m)$	:	"m is a man."
$Takes(x, c, s)$	:	"Student x takes course c in term s."
$Policy(p)$	:	"p is a policy."
$Expensive(x)$	:	"x is expensive."
$Buys(x, y)$	:	"x buys y."
$Sells(x, y, z)$	:	"x sells y to z."
$Insured(x)$	:	"x is insured."
$Shaves(x, y)$	:	"x shaves y."
$Citizen(x, s, c)$	:	"x is a citizen of s by justification c."
$Resident(x, s)$	:	"x is a resident of s."
$Parent(x, y)$	:	"x is a parent of y."
$BornIn(x, s)$	:	"x was born in s."
F_{101}	:	A specific French class.
G_{101}	:	A specific Greek class.
$Sp01$	:	Spring 2001.
UK	:	The UK.

(a) $\exists x\ Student(x) \land Takes(x, F_{101}, Sp01)$

(b) $\exists x\ Student(x) \land Takes(x, G_{101}, Sp01) \land \exists y\ Student(y) \land Takes(y, G_{101}, Sp01)$

(c) $\forall x\ \forall y\ Person(x) \land Policy(y) \land Buys(x, y) \Rightarrow Smart(x)$

(d) $\forall x\ Person(x) \Rightarrow \neg\exists y\ [Policy(y) \land Expensive(y) \land Buys(x, y)]$

(e) $\exists x\ Agent(x) \land \forall y\ [Person(y) \land [\exists p\ Policy(p) \land Sells(x, p, y)] \Rightarrow \neg Insured(y)]$

(f) $\exists x\ Barber(x) \land \forall y\ [Man(y) \land \neg Shaves(y, y) \Rightarrow Shaves(x, y)]$

(g) $\forall x\ Person(x) \land BornIn(x, UK) \land [\forall y\ Parent(y, x) \Rightarrow [\exists c\ Citizen(y, UK, c)] \lor Resident(y, UK)] \Rightarrow Citizen(x, UK, Birth)$

(h) $\forall x\ Person(x) \land \neg BornIn(x, UK) \land [\exists y\ Parent(y, x) \land Citizen(x, UK, Birth)] \Rightarrow Citizen(x, UK, Descent)$

Problem 14. Resolution proof

(a) A set is normal if it is not an element of itself. Using $n(x)$ to mean that x is normal and $e(x, y)$ to mean that x is an element of y, write an axiom defining normality.

(b) Write down in first-order logic the claim that there is a set s such that the elements of s are exactly all the normal sets.

(c) Prove using resolution that the claim in (b) is false. Show the proof graphically, as discussed in textbook and in lecture, with the required substitution for each resolution step.

(d) (Extra credit, for fun) Relate this result to part (f) of Problem 13. Is that assertion also necessarily false? Explain.

(a) $\forall x\ n(x) \Leftrightarrow \neg e(x, x)$

(b) $\exists s\ \forall x\ e(x, s) \Leftrightarrow n(x)$

(c) Converting to CNF, the axiom defining normality in part (a) becomes

$$\forall x\ n(x) \Leftrightarrow \neg e(x, x)$$
$$(\neg n(x) \lor \neg e(x, x)) \land (n(x) \lor e(x, x))$$

which we label as clauses 1 and 2. The claim in part (b) becomes

$$\exists s\ \forall x\ e(x,s) \Leftrightarrow n(x)$$
$$\exists s\ \forall x\ (\neg e(x,s) \vee n(x)) \wedge (\neg n(x) \vee e(x,s))$$
$$(\neg e(x,S) \vee n(x)) \wedge (\neg n(x) \vee e(x,S))$$

which we label as clauses 3 and 4.

To show by resolution that the claim in (b) is false, we need to show that the negation of the claim is entailed by the KB, i.e., by the definition in (a). This is the same as showing that (a) and (b) lead to a contradiction.

Resolving 1 and 4, we obtain 5: $\neg n(S) \vee \neg n(S)$ which reduces to $\neg n(S)$ after factoring.

Resolving 2 and 3, we obtain 6: $n(S) \vee n(S)$ which reduces to $n(S)$ after factoring.

Resolving 5 and 6, we obtain the empty clause.

(d) The sentence in 13(f) is similar to the example used by Bertrand Russell to illustrate what became known as Russell's Paradox—namely that if set theory is constructed to allow sets to be defined by "the set of all x such that $\phi(x)$ is true" where ϕ is any first-order formula in x, then logical contradictions can arise.

A good discussion appears at `http://plato.stanford.edu/entries/russell-paradox/`

In the barber example, the contradiction is that the barber must shave himself or not. If he doesn't shave himself, then the sentence says that he does shave himself; and in Russell's example, if he does shave himself, then he doesn't (because he shaves only those who don't shave themselves).

Now, 13(f) doesn't quite amount to a contradiction. First, it says "all men" but not "all and only," so the barber could still shave himself. Second, the barber could be from out of town. Third, the barber could be a woman.[1]

Problem 15. Basic concepts.

Below are just a few examples.

(a) What is the soundness of an inference procedure?
(b) Is forward chaining complete for first-order definite clauses?
(c) What are two strategies for local search?
(d) Can local search strategies guarantee that the optimal solution can be found?

(a) An inference procedure is sound if its conclusions are logically entailed by its premises.

(b) Yes.

(c) Hill-climbing and simulated annealing.

(d) Hill-climbing can't unless local maxima is always global maxima. Simulated annealing can escape local maxima if an solution exists and one chooses an appropriate annealing schedule.

[1] Thanks to Prof. Russell for the problems and answers.