Name:____The answers_____

SSN:_____

1	2	3	4	5	6	7	8	9	total
20	*20*	*15*	*15*	*10*	*15*	*30*	*15*	*10*	*150*

UMBC CMSC471 Midterm Exam
October 10, 2001

Being forced to write comments actually improves code,
because it is easier to fix a crock than to explain it. - Guy Steele

Please write all of your answers on this exam. The exam is closed book and consists of N problems which add up to M points. You have the entire class period, seventy five minutes, to work on this exam. Good luck.

0. Warm Up Exercise (0)

Is no your answer to this question?

This of course is a variation on the classic "liar's paradox". If you like this kind of thing, check out any of the popular books by mathematician/logician Raymond Smullyan.

1. Unification (20)

Each of the following Prolog expressions invokes unification. For each, say whether or not the unification would succeed and, if it does, what values the variables in the terms would take on. Assume that the same variable name in two different terms represents the same variable.

1. loves(john,John)=loves(Mary,mary)
   yes. John=mary,Mary=john
2. f(X)=X(f)
   no. syntax error.
3. p(X,q(Y),r(Z)) = p(r(a), q(X), r(b))
   X=Y=r(a),Z=b

4. loves(john,X) = loves(Y,Y)
   yes. X=Y=john
5. 1+2+3=X+Y
   yes.~~X=1,Y=2+3~~
6. [X,Y|Z] = [1,2]
   yes.X=1,Y=2,Z=[]
7. [[1,2],1,2]=[X|X]
   yes. X=[1,2]
8. f(X,f(Y)) = f(f(~~y~~),f(X))
   yes. X=Y=f(f(f(f(f…)))))
9. X = f(X)
   yes.X=f(f(f(f(…))))
10. f(g(X),Y)=f(Y,g(a))
    X=a,Y=g(a)

2. True/False (20 points)

Instructions: Circle either T or an F in the space before each statement to indicate whether the statement is true or false. If you think the answer is simultaneously true and false, quit while you are ahead. There is no penalty for incorrect answers (but then, there are no points for incorrect answers either).

T F Alan Turing proposed his famous Turing test as a technique for deciding whether or not a problem was "Turing computable".**FALSE**

T F Even Alan Turing would not be able to pass the Turing test..**FALSE**

T F Prolog's \+ operator is not true negation but instead is "negation as failure", e.g., \+P is true if Prolog is unable to prove P given the current database of facts and rules.**TRUE**

T F In Prolog, \+ (\+ P) is necessarily true whenever P is true. **TRUE**

T F If Prolog term A unifies with term B, and term A unifies with term C, then term B unifies with term C. **FALSE**

T F The only reason that Prolog is not considered a "pure logic programming language" is how it does numerical computations. **FALSE**

T F Uninformed search algorithms are only useful in problems which have finite search spaces. **FALSE**

T F One of the advantages of hill climbing is that it requires a small, bounded amount of memory while doing the search. **TRUE**

T F It is possible for Algorithm A* to expand a non-optimal solution node. **FALSE**

T F Algorithm A will perform a breadth-first search if F(n) = G(n). **TRUE**

T F Algorithm A will perform a depth-first search if F(n) = -G(n). **TRUE**

T F Using an uninformed breadth-first search will always find an optimal solution to a problem if a finite solution exists. **TRUE**

T F Uniform cost search is a generalization of depth first search in which arcs can have non-unit costs. **FALSE**

T F Bi-directional search is always more efficient than uni-directional search. **FALSE**

T F One reason to prefer *an iterative deepening* version of depth first search over a regular depth first search is that the iterative deepening version requires less space to run. **TRUE**

T F The **maximin principle** in game theory is based on the idea that a good strategy is to plan on taking advantage of the tactical errors your opponent makes. **FALSE**

T F If graph search uses an evaluation function of the form f(n)=w1*g(n)+w2*h(n) where w1,w2, g(n), and h(n) are all greater than zero for all n, then it will always find a solution if one exists. **TRUE**

T F The **minimax algorithm** is based on the assumption that the game is a zero-sum game. **TRUE**

T F The mini-max and alpha-beta procedures will **always** back up identical values to the root node of a game tree. **TRUE**

T F One drawback of game playing strategies based simply on the use the mini-max and alpha-beta algorithms is that one can not make sacrifices where an immediate loss is permitted in order to obtain a subsequent advantage. **FALSE**

3. Oldest Person (15)

Suppose a Prolog database contains assertions of the form *person(X)* which are true if X is a person and *age(X,N*), true if the age of thing X is N. Write a Prolog query which will find the oldest person in the database. The query should generate multiple solutions if there are several people who are the oldest (i.e., there are no people are in then database who are older than any of them). You may define any auxiliary predicates you need. Hint: try finding a person P with age A such that it's impossible to find a person who has a larger age. This will require the use of the unprovable operator (\+).

person(P),agent(P,A),\+(person(P2),\+(P2=P),agent(P2,A2),A2>A).

4. Prolog (15 points)

Consider two possible definitions of the member predicate, member1 and member2.

```
member1(X,[X|Tail]).
member1(X,[Y|Tail]) :-
   member1(X,Tail).
```

```
member2(X,[X|Tail]) :- !.
member2(X,[Y|Tail]) :-
    member2(X,Tail).
```

Both of these definitions are, in fact, useful. Describe in high level terms (a) [5] how they differ; (b)[5] why one would chose to use one definition rather than the other and (c) [5] show all possible solutions to the two goals *member1(X,[1,2,3])* and *member2(X,[1,2,3])*

(a) member1/2 can generate all possible answers to the query. Member2/2 will only generate one answer.

(b) Member2/2 is useful if you just want to check if a given term is in a list. It's preferred to using member1/2 for such a case because it will not succeed twice if the term is in the list more than once and it is more efficient, in that the cut allows the interpreter to remove choice points it would otherwise have to keep on the stack.

(c) member1(X,[1,2,3]) succeeds with X=1, X=2 and X=3 whereas member2(X,[1,2,3]) only succeeds with X=1.

5. Mystery predicate (10)

For the following mystery predicate, (a) [5] Describe what it does in a simple sentence; and (b) [5] Show what each of these two queries would result in: *p([1,2,3],X*) and *p(X,[1,2,3]).*

```
p(A,B) :- p(A,[],B).
p([],A,A).
p([A|B],C,D) :- p(B,[A|C],D).
```

(a) p/2 is reverse. That is p(A,B) is true if A and B are lists and B is equal to A with its top-level elements reversed. Note that this is written ina tail recursive style.

(b) | ?- p([1,2,3],X).
X = [3,2,1] ?
yes
| ?- p(X,[1,2,3]).
X = [3,2,1] ?

6. Heuristic functions (15)

Suppose we have two heuristic functions, h1 and h2, both of which are known to be admissible (e.g. underestimates of the optimal heuristic function h*). (a) [5] What constraint has to hold for h1 to be considered as a stronger heuristic than h2? (b)[5] which of the following heuristic is also admissible? (c) [5] which are guaranteed to be no weaker than either h1 or h2?

(1) h3(n) = (h1(n) + h2(n)) / 2
(2) h4(n) = min(h1(n),h2(n))
(3) h5(n) = max(h1(n), h2(n))
(4) h6(n) = w1*h1(n) + w2*h2(n) where 0 =< w1,w2 =<1 and w1+w2=1.0
(5) h7(n) = h1(n) *h2(n)

(a) h1(x) >= h2(x) for all x and h1(x) > h2(x) for some value of x.
(b) 1,2,3,and 4 are admissible
(c) only 3 is guarnteed not to be weaker than either h1 or h2.

7. Three Puzzle Search (30)

We are given a sliding-tile puzzle with three tiles and a blank and are given a problem with the following initial and final states where a "o" represents a blank spot. Assume that we always try moving the blank first right, then left, then up and then down.

initial

o	1
3	2

goa

1	2
o	3

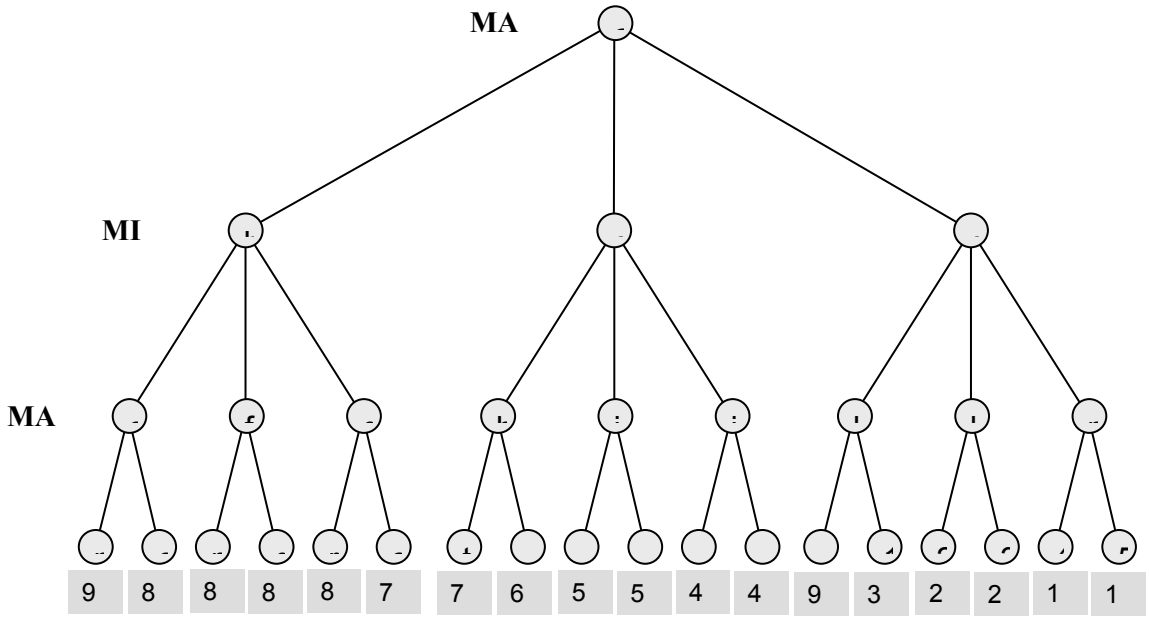
(a) [5] How many possible states are there, either reachable or unreachable, from the given start state.
(b) [10] Draw the complete search space reachable from the given start state as a graph, making sure that a given state occurs only once.
(c) [5] If we treated the search space as a tree, what would the result of a breadth-first search be? Of a depth-first search?
(d) [5] If we treat the search space as a graph (as in graph search or algorithm A), what would the result of a breadth-first search be? Of a depth-first search?
(e) [5] Suppose we use a hill climbing search algorithm on this problem with an evaluation function which counts the number of tiles "out of place" with respect to the goal. Will this lead to a solution?

(a) there are 4! Or 24 possible state. (Twelve are reachable from the initial state)
(b) see separate page
(c) A breadth first search will quickly find a solution which is of length three steps. A depth first search will loop.
(d) A breadth first search will even more quickly find the solution of length three steps. A depth first search will also find a solution of length three steps, since it can avod looping.
(e) Hill climbing will quickly find the solution in this case.

8. AlphaBeta Pruning (15)

Consider the following game tree in which the static evaluation function values are shown below each leaf node. Assume that the root node corresponds to the maximizing player. Assume the search always visits children left-to-right.

(a) [5] Compute the (final) backed-up values computed by the **alpha-beta algorithm**. Show your answer by writing values at the appropriate nodes in the above tree.
(b) [8] Circle the nodes that are *not* examined by alpha-beta.
(c) [2] What move should Max choose?



9. Minimax Algorithm Applied to 3-Person Games (10)

Consider a 3-person, perfect information game in which players A, B and C play in turn. Based on information specific to each player, we have three independent static evaluation functions, f_A, f_B and f_C that are associated with players A, B and C, respectively. Each function indicates the estimated value of a board position with respect to that player. For example, $f_A(p) = -5$ means that position p is not good for player A, whereas $f_A(p) = 100$ means that position p looks very good for player A. Say we want to modify the Minimax algorithm to work for this game. At each leaf node, n, we compute a triple of values $(f_A(n), f_B(n), f_C(n))$. What are the **backed-up values** that should be computed for the following game tree, where the root corresponds to player A's turn, nodes at depth 1 are positions where it's player B's turn, etc.