Final Examination — December 7, 2018

Duration: 180 minutes
No aids allowed.
Total marks: 135.

SOLUTIONS

Name:

Student Number:

1)	/15
2)	/10
3)	/8
4)	/4
5)	/8
6)	/10
7)	/20
8)	/20
9)	/10
10)	/10
11)	/10
12)	/10
Total	/135

1. [15 points] For each of the following statements indicate whether it is *true* or *false*:

   (a) Hill climbing search may get stuck in a local maximum. T

   (b) A Horn clause has at most one positive literal. T

   (c) When we make the closed world assumption about a 2-place predicate/relation r, if the knowledge base does not entail that $r(a, b)$, then $r(a, b)$ must be false. T

   (d) Iterative deepening depth-first search uses more space than breadth-first search on problems with a large search space. F

   (e) A constraint satisfaction problem that has been made arc-consistent can always be solved without backtracking. F

   (f) In forward checking, we check constraints that have only one uninstantiated variable and delete values that violate the constraint. T

   (g) If a heuristic is admissible, it must be monotone/consistent. F

   (h) If q_1 is conditionally independent from q_2, then q_2 is conditionally independent from q_1. T

   (i) When A^* uses a heuristic that is only admissible, it may not find the optimal path. F

   (j) Markov decision processes assume that the state is fully observable. T

   (k) The value iteration algorithm can be used to solve a Markov decision process. T

   (l) In Q-learning, one learns the transition model $P(s'|s, a)$. F

   (m) Decision-tree learning produces models that are more understandable than neural network learning does. T

   (n) A single-layer feedforward neural network (perceptron) can learn any continuous function. F

   (o) A learned model that displays overfitting generalizes well. F

2. [10 points] Suppose that we have a search algorithm that keeps unexpanded nodes in a sorted list, the frontier (or fringe), and that always selects the first node from the frontier. What kind of search do we have if...

   (a) we always place the successors of the current node at the front of the frontier?

   depth-first search

   (b) we always place the successors of the current node at the back of the frontier?

   breadth-first search

   (c) we insert the successors of the current node so as to keep the frontier sorted by path cost $g(n)$ (i.e. the cost of the path from the start node to node n)?

   uniform cost search

   (d) we insert the successors of the current node so as to keep the frontier sorted by the value of the heuristic function $h(n)$?

   greedy best-first search

   (e) we insert the successors of the current node so as to keep the frontier sorted by the value of the sum of the path cost and the heuristic function $g(n) + h(n)$?

   A*

3. [8 points] Implement the following description of an *above* relation (in a blocks world) using one or more Prolog rules:

   if X is on Y, then X is above Y;

   if X on Y and Y is above Z, then X is above Z;

   nothing else is in the above relation.

   You can assume that the `on(X,Y)` predicate has already been defined.

```
above(X,Y) :- on(X,Y).
above(X,Z) :- on(X,Y), above(Y,Z).
```

4. [4 points] What is the output of the following Prolog program, when we invoke the query `?- q.`

```
q:- writeln(qa), r.
q:- writeln(qb), !, r.
q:- writeln(qc), fail.

r:- writeln(ra), !, fail.
r:- writeln(rb), fail.
```

```
qa
ra
qb
ra
false
```

5. [8 points] The predicate `bagof(T, Q, S)` constructs a list `S` of all instances of `T` for which `Q` is true. A textbook describes the computation as follows: the query `Q` is satisfied in all possible ways, and for each instantiation of the variables in the arguments of `Q`, `T` is instantiated and recorded as such in the list `S`.

   (a) Give the value computed for `L` by the query

   ```
   ?- bagof(X, (member(X, [0, 8, 5, 4]), not(X >= 7)), L).
   ```

   [0, 5, 4]

   (b) The following query will not work:

   ```
   ?- bagof(X, member(0, X), L).
   ```

   Explain briefly why.

   because member(0, X) succeeds with an infinite number of values for X, lists of different sizes

6. [10 points] Convert the following formulas into clausal form (recall that the 8 steps are: eliminate implications, move negations inward, standardize variables, skolemize, convert to prenix, distribute $\vee$ over $\wedge$, flatten conjunctions and disjunctions, and convert to clauses):

   (a) $((\exists X.p(X)) \vee (\exists X.q(X))) \rightarrow (\exists X.p(X) \vee q(X))$

see attached

   (b) $\forall X.(p(X) \rightarrow \exists Y.(\forall Z.(q(X,Y,Z)) \wedge r(X,Y,Z)))$

see attached

6a) as intended.

$$((\exists x.p(x)) \vee (\exists x.q(x))) \rightarrow (\exists x.(p(x) \vee q(x)))$$

$$\neg((\exists x.p(x)) \vee (\exists x.q(x))) \vee (\exists x(p(x) \vee q(x)))$$

$$((\forall x.\neg p(x)) \wedge (\forall x.\neg q(x))) \vee (\exists x(p(x) \vee q(x)))$$

$$((\forall x.\neg p(x)) \wedge (\forall y.\neg q(y))) \vee (\exists z.(p(z) \vee q(z)))$$

$$((\forall x.\neg p(x)) \wedge (\forall y.\neg q(y))) \vee (p(a) \vee q(a))$$

$$\forall x \forall y.(\neg p(x) \wedge \neg q(y)) \vee (p(a) \vee q(a))$$

$$\forall x \forall y.(\neg p(x) \vee p(a) \vee q(a)) \wedge$$
$$(\neg q(y) \vee p(a) \vee q(a))$$

$$[\neg p(x), p(a), q(a)]$$

$$[\neg q(y), p(a), q(a)]$$

6a) alternative interpretation

$((\exists x.p(x)) \vee (\exists x,q(x))) \rightarrow (\exists x.p(x) \vee q(x))$

$\neg((\exists x.p(x)) \vee (\exists x,q(x))) \vee (\exists x.p(x) \vee q(x))$

$((\forall x.\neg p(x)) \wedge (\forall x.\neg q(x))) \vee (\exists x.p(x) \vee q(x))$

$((\forall x.\neg p(x)) \wedge (\forall y.\neg q(y))) \vee (\exists z.p(z) \vee q(w))$

$(\forall x,\neg p(x) \wedge (\forall y.\neg q(y))) \vee p(a) \vee q(w)$

$\forall x \forall y.(\neg p(x) \wedge \neg q(y)) \vee (p(a) \vee q(w))$

$\forall x \forall y.(\neg p(x) \vee p(a) \vee q(w)) \wedge$
$(\neg q(y) \vee p(a) \vee q(w))$

$[\neg p(x), p(a), q(w)]$

$[\neg q(y), p(a), q(w)]$

6b) as it should be

$$\forall x.(p(x) \rightarrow \exists y.(\forall z.(q(x,y,z) \wedge r(x,y,z))))$$

$$\forall x.(\neg p(x) \vee \exists y.(\forall z.(q(x,y,z) \wedge r(x,y,z))))$$

$$\forall x.(\neg p(x) \vee \forall z.(q(x,f(x),z) \wedge r(x,f(x),z$$

$$\forall x \forall z.(\neg p(x) \vee (q(x,f(x),z) \wedge r(x,f(x),z)))$$

$$\forall x \forall z.(\neg p(x) \vee q(x,f(x),z)) \wedge (\neg p(x) \vee r(x,f(x),z))$$

$$[\neg p(x), q(x,f(x),z)]$$

$$[\neg p(x), r(x,f(x),z)]$$

6 b) as in question w. parentheses is error

$\forall x.(p(x) \Rightarrow \exists y.(\forall z.(q(x,y,z)) \wedge r(x,y,z))$

$\forall x.(\neg p(x) \vee \exists y.(\forall z.(q(x,y,z)) \wedge r(x,y,z)$

$\forall x.(\neg p(x) \vee \exists y.(\forall z(q(x,y,z)) \wedge r(x,y,w)$

$\forall x.(\neg p(x) \vee (\forall z.(q(x,f(x),z)) \wedge r(x,f(x),w$

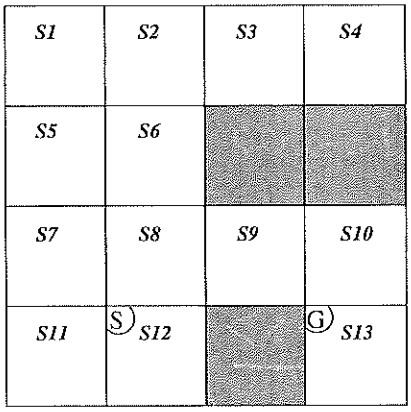
$\forall x.\forall z.(\neg p(x) \vee (q(x,f(x).z) \wedge r(x,f(x),w,$

$\forall x.\forall z.(\neg p(x) \vee q(x,f(x),z)) \wedge$
$(\neg p(x) \vee r(x,f(x),w))$

$[\neg p(x), q(x,f(x),z)]$

$[\neg p(x), r(x,f(x),w)]$

7. [20 points] Consider the problem of finding a path in a grid using *heuristic search*. The problem is to find a path from square $S = s12$ to square $G = s13$, given that you can move only horizontally and vertically, one square at a time, and no step may be made into a shaded square. Each step *has cost one*.



Suppose that we want to use A* *search with cycle-checking* and the *Manhattan distance* as the heuristic function h to solve this problem. The Manhattan distance between two cells is the sum of the horizontal and vertical distances (counting shaded squares as well). Note that this heuristic is *monotonic*.

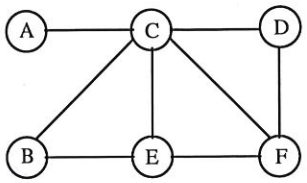
(a) Write down the h values of squares $s1$ through $s13$ in the grid below (the values for the start node $s12$ and the goal node $s13$ have already been given):

S1 h= 6	S2 h= 5	S3 h= 4	S4 h= 3
S5 h= 5	S6 h= 4		
S7 h= 4	S8 h= 3	S9 h= 2	S10 h= 1
S11 h= 3	S S12 h=2		G S13 h=0

(b) Now, apply the A* search: continue writing down the frontier for each step of the algorithm in the format shown below (it must be sorted on f values, and f written as $f = g + h$). In the case of equal f values, break ties as follows: the square with the *larger number* comes first (e.g. $s11$ before $s10$ and $s5$). Underline the node selected for expansion at each step. *Remember to enforce cycle-checking.*

1. $\{\underline{\langle s12, 2 = 0 + 2\rangle}\}$
2. $\{\underline{\langle s11, 4 = 1 + 3\rangle}, \langle s8, 4 = 1 + 3\rangle \}$
3. $\{\underline{\langle s8, 4 = 1 + 3\rangle}, \langle s7, 6 = 2 + 4\rangle\}$
4. $\{\underline{\langle s9, 4 = 2 + 2\rangle}, \langle s7, 6 = 2 + 4\rangle, \langle s6, 6 = 2 + 4\rangle\}$
5. $\{\underline{\langle s10, 4 = 3 + 1\rangle}, \langle s7, 6 = 2 + 4\rangle, \langle s6, 6 = 2 + 4\rangle\}$
6. $\{\underline{\langle s13, 4 = 4 + 0\rangle}, \langle s7, 6 = 2 + 4\rangle, \langle s6, 6 = 2 + 4\rangle\}$
7. $\{\langle s7, 6 = 2 + 4\rangle, \langle s6, 6 = 2 + 4\rangle\}$ Goal is reached!

8. [20 points] Consider the following *constraint satisfaction problem.* We have the graph shown below with 6 nodes A, B, C, D, E, and F. We want to color each node such that no two nodes connected by an edge get the same color. There are only three colors b, g and r (i.e. blue, green, red) available for nodes C, D, E, F. Moreover, node A must be colored red and node B must be colored blue.

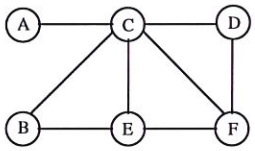


(a) Write down the domain of each of the 6 variables (do not apply forward checking yet).

$Dom(A) = \{ r \}$, $Dom(B) = \{ b \}$, $Dom(C) = \{ b, g, r \}$,
$Dom(D) = \{ b, g, r \}$, $Dom(E) = \{ b, g, r \}$, $Dom(F) = \{ b, g, r \}$.

(b) Complete the search tree on the next page to show the tree that would be explored by the *backtracking search with forward checking* algorithm. Remember that in this algorithm, constraints are checked when all but one of their variables have been assigned and values that violate the constraint are removed from the domain of the unassigned variable. You are to use a static variable ordering whereby each branch considers assignments to the variables in the sequence A, B, C, D, E, F. The values are also tried in the sequence b, g, and r when applicable. Annotate each edge with what variable is assigned what value and the updated domains of the remaining variables. Use the symbol DWO to mark whenever a domain wipe-out happens.

Q2b: Complete the search tree. Remember to try values in the order b, g, and then r. The graph on the left is shown for your convenience.



A=r
d(B) = {b}
d(C) = {b, g}
d(D, E, F) = {b, g, r}

B=b
d(C) = {g}
d(E) = {g, r}
d(D, F) = {b, g, r}

C=g
d(D) = {b, r}
d(E) = {r}
d(F) = {b, r}

D=b
d(E) = {r}
d(F) = {r}

E=r
d(F) = ∅
Dead
Backtrack!

D=r
d(E) = {r}
d(F) = {b}

E=r
d(F) = {b}

F=b

Success

changes underlined

9. [10 points] Consider the following dynamic domain, to be specified in the situation calculus. Suppose that we have a fluent $isOpen(X, S)$ that is true if and only if door X is open in situation S and another fluent $isLocked(X, S)$ that is true if and only if door X is locked in situation S. Suppose also that we have an action $open(X)$ whose only effect is to open a closed (i.e. not open) door X provided that X is not locked, as well as an action $close(X)$ whose only effect is to close an open door X. Finally, assume that these are the only actions that affect the fluent $isOpen$.

a) Write effect axioms for the actions *open* and *close*; your axioms should capture all the effects of these actions.

$\neg isLocked(X,S) \rightarrow isOpen(X, do(open(X), S))$

$\neg isOpen(X, do(close(X), S))$

b) Write a frame axiom for the action *open* and the fluent *isOpen*.

$\neg isOpen(X,S) \wedge isLocked(X,S) \rightarrow \neg isOpen(X, do(open(X),S))$

c) Write a successor state axiom for the fluent *isOpen*.

$isOpen(X, do(A,S)) \Leftrightarrow$

$A = open(X) \wedge \neg isLocked(X,S) \vee$

$isOpen(X,S) \wedge A \neq close(X)$

10. [10 points] Consider the following STRIPS actions:

Action Name	Preconditions	Add Effects	Delete Effects
A	-	p	-
B	o	-	o
C	p	q	r
D	p	r	-

Suppose that the start state is $S_0 = \{o\}$ and the goal is $G = \{p, q, r, o\}$.

(a) Which actions are applicable in the start state S_0? What is the new state for each of the applicable actions?

applicable	new state
A	{o, p}
B	{ }

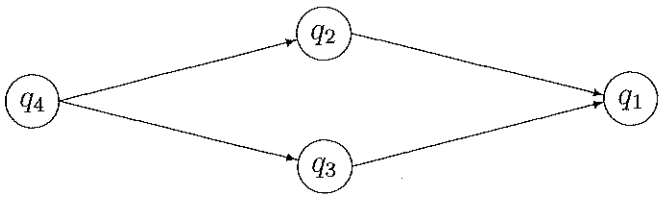
(b) Suppose we want to do backward/regression planning. Which actions are consistent with the goal G? What is the regressed goal for all the consistent actions?

consistent	regressed goal
A	{q, r, o}
D	{p, q, o}

(c) Give a sequence of actions that achieves the goal G when executed in the starting state S_0.

[A, C, D]

11. [10 points] Suppose that we have the following belief network (or Bayes net):



(a) True or false, given this belief network, is it the case that $Pr(q_1|q_2, q_3, q_4) = Pr(q_1|q_2, q_3)$?

True

$Pr(q_1 | q_2, q_3, q_4) = Pr(q_1 | q_2, q_3)$

(b) What are the conditional probabilities that need to be specified to fully determine the joint probability distribution?

$Pr(q_4)$

$Pr(q_2|q_4)$

$Pr(q_2|\overline{q_4})$

$Pr(q_3|q_4)$

$Pr(q_3|\overline{q_4})$

$Pr(q_1|q_2,q_3)$

$Pr(q_1|q_2,\overline{q_3})$

$Pr(q_1|\overline{q_2},q_3)$

$Pr(q_1|\overline{q_2},\overline{q_3})$

(c) Express $Pr(q_3|q_2)$ in terms of the conditional probabilities given in your answer to the previous question.

$Pr(q_3|q_2) = Pr(q_2 \wedge q_3) / Pr(q_2)$

$Pr(q_2) = Pr(q_2|q_4) \cdot Pr(q_4) + Pr(q_2|\overline{q_4}) \cdot (1 - Pr(q_4))$

$Pr(q_2 \wedge q_3) = Pr(q_2|q_4) \cdot Pr(q_4) \cdot Pr(q_3|q_4) + Pr(q_2|\overline{q_4}) \cdot Pr(q_3|\overline{q_4}) \cdot (1 - Pr(q_4))$

or

$Pr(q_2 \wedge q_3) = \Sigma J(Q_1, q_2, q_3, Q_4)$

$Pr(q_3) = \Sigma J(Q_1, Q_2, q_3, Q_4)$

12. [10 points] In this question, you must write a Prolog program that solves the N queens problem, that is, figures out how to place N queens on a N by N rectangular board so that no queen attacks another, i.e., no pair of queens is on the same row, column, or diagonal. We can assume that each row contains exactly one queen. So we can represent a potential solution, a placement of the queens, as a list $[c_1, c_2, \ldots, c_N]$ where c_1 is the number of the column that the queen on row 1 is placed; c_2 is the number of the column that the queen on row 2 is placed, ..., and c_N is the number of the column that the queen on row N is placed. We can assume that columns are numbered from 1 to N. Since we can't have more than one queen in any given column, we can represent a potential solution as a permutation of the list of the positive integers from 1 to N. For example $[2, 4, 1, 3]$ is a solution to the 4 queens problem. We want to mplement a solution to this problem using a "generate and test" approach.

(a) Implement a Prolog predicate `safe(L)` that succeeds if and only if the list L representing a placement of the queens (a permutation of the first N positive integers as discussed above) is *safe*, i.e., no pair of queens is on the same row, column, or diagonal. You may introduce auxiliary predicates, with suitable documentation. You must provide definitions for all non-built-in predicates that you use.

```
safe([]).
safe([Q|R]) :- safe(R), not attack(Q,R).

attack(X,L) :- attack(X,1,L).
attack(X,N,[Y|R]) :- X =:= Y+N;
                     X =:= Y-N.
attack(X,N,[Y|R]) :- N1 is N+1,
                     attack(X,N1,R).
```

(b) Implement a Prolog predicate `queens(N,L)` that holds if and only if the list `L` represents a placement of N queens (as discussed earlier) that is a solution to the N queens problem. You may use the predicate `safe(L)` defined in (a) in your solution. You may also use (without defining it) a predicate `permutation(L,P)` that holds if and only if list `P` is a permutation of the list `L`. You may introduce auxiliary predicates, with suitable documentation. You must provide definitions for all non-built-in predicates that you use.

```
queens(N,L) :- range(1,N,L1),
               permutation(L1,L),
               safe(L).

range(N,N,[N]).
range(N,M,[N|L]) :-
        N<M, N1 is N+1, range(N1,M,L).
```