3-16. ARTIFICIAL INTELLIGENCE — MODEL ANSWERS

1. a)

```
eligible( Player, Country ) :-
        person( Player, Country, _, _ ).

eligible( Player, Country ) :-
        person( Player, _, Mother, _ ),
        person( Mother, Country, _, _ ).

eligible( Player, Country ) :-
        person( Player, _, _, Father ),
        person( Father, Country, _, _ ).

eligible( Player, Country ) :-
        person( Player, _, Mother, Father ),
        person( Mother, _, MatGM, MatGF ),
        person( Father, _, PatGM, PatGF ),
        countgps( [MatGM, MatGF, PatGM, PatGF], Country, 0 ).

countgps( [], Country, Sum ) :-
        Sum >= 2.

countgps( [GP|GPs], Country, SoFar ) :-
        person( GP, Country, _, _ ), !,
        SoFar1 is SoFar + 1,
        countgps( GPs, Country, SoFar1 ).

countgps( [_|GPs], Country, SoFar ) :-
        countgps( GPs, Country, SoFar ).
```

b)

```
check_team( Squad, Country, Reasons ) :-
        check_length( Squad, R1 ),
        check_differnt( Squad, R2 ),
        check_eligible( Squad, Country, R3 ),
        append( R1, R2, Rtemp ),
        append( Rtemp, R3, Reasons ).

check_length( Squad, [] ) :-
        length( Squad1, L ),
        L < 26, !.
check_length( _, [too_big] ).

check_different( [], [] ).
check_different( [H|T], [same_man] ) :-
        member( H, T ), !.
check_different( [H|T], R ) :-
        check_different( T, R ).
```

```
check_eligible( [], [] ).
check_eligible( [H|T], R ) :-
        eligible( H, C ), !,
        check_eligible( T, C, R ).
check_eligible( _, [ineligible_player] ).
```

c)

```
check_international( Squad1, Country1, Squad2, Country2, CR ) :-
        different_countries( Country1, Country2, R1 ),
        check_country( Squad1, Country1, R2 ),
        check_country( Squad2, Country2, R3 ),
        append( R1, R2, Rtemp ),
        append( Rtemp, R3, CR ).

different_countries( Country, Country, [(Country,same)] ) :- !.
different_countries( _, _, [] ).

check_country( Squad, Country, [] ) :-
        check_team( Squad, Country, [] ), !.
check_country( Country, [(Country,bad_squad)] ).
```

d)

```
points2scores( 0, [] ).

points2scores( N, [goal|L] ) :-
        N >= 7,
        NewN is N - 7,
        points2scores( NewN, L ).
points2scores( N, [try|L] ) :-
        N >= 5,
        NewN is N - 5,
        points2scores( NewN, L ).
points2scores( N, [penalty|L] ) :-
        N >= 3,
        NewN is N - 3,
        points2scores( NewN, L ).
```

2. a) state – Prolog term
Node = state plus possibly some more information
Path = list of nodes
Graph = list of Paths

- A graph G can be searched for a Solution Path SP, if
  - Pick a path P in G, AND
  - Get frontier node N of path P, AND
  - Get problem-state S of node N, AND
  - S is a goal state. [So P *is* a Solution Path SP!]
- A graph G can be searched for a Solution Path SP, if
  - Pick a path P in G, AND
  - Set $OtherPaths \leftarrow G - \{P\}$, AND
  - Get frontier node N of path P, AND
  - Compute the set N' of all the new nodes reachable from N, AND
    - * by applying all the state change rules to N
  - Set $NewPaths \leftarrow \{[n' \mid P] \mid \exists n' \in N'\}$, AND
  - Make a bigger graph G^+ from $NewPaths$ plus $OtherPaths$, AND
  - Graph G^+ can be searched for a Solution Path SP.

b)

```
ibbs_search( Graph, SolutionPath, Beam ) :-
     search( Graph, SolutionPath, Beam ).
ibbs_search( Graph, SolutionPath, Beam ) :-
     Beam1 is Beam + 1,
     ibbs_search( Graph, SolutionPath, Beam1 ).

search( Graph, [Node|Path], _ ) :-
     choose( [Node|Path], Graph, _ ),
     state_of( Node, State ),
     goal_state( State ).
search( Graph, SolnPath, Beam ) :-
     choose( Path, Graph, OtherPaths ),
     one_step_extensions( Path, NewPaths ),
     add_to_paths( NewPaths, OtherPaths, GraphPlus, Beam ),
     search( GraphPlus, SolnPath, Beam ).

choose( Path, [Path|Graph], Graph ).

add_to_paths( NewPaths, OtherPaths, GraphPlus, Beam ) :-
     insert_in_order( NewPaths, OtherPaths, AllGraph ),
     prune( AllGraph, GraphPlus, Beam ).

insert_in_order( [], Graph, Graph ).

insert_in_order( [Path|Paths], Graph, AllGraph ) :-
     insert_one( Path, Graph, GraphPlus ),
     insert_in_order( Paths, Graph, AllGraph ).

insert_one( [(F1,Cost1)|Path1], [ [(F2,Cost2)|Path2] | Paths ], Graph ) :-
     Graph = [ [(F1,Cost1)|Path1], [(F2,Cost2)|Path2] | Paths ], !.

insert_one( Path, [CheaperPath|Graph1], [CheaperPath|Graph2] ) :-
```

```
    insert_one( Path, Graph1, Graph2 ).

insert_one( Path, [], [Path] ).

prune( [], _, [] ) :- !.

prune( _, 0, [] ) :- !.

prune( [H|T1], Beam1, [H|T2] ) :-
    Beam is Beam1 - 1,
    prune( T1, Beam, T2 ).
```

c) assumption: that every path eventually terminates, so search can fail and try another beam width

complete – yes, exhaustive search, unless infinite number of branches from node
optimal – no, might still discard the optimal path while a sub-optimal path stays in the beam
complexity: $\mathcal{O}(b^d)$, because the width of the beam might have to be the branching factor to the depth of the solution

3. a)

$$P_G = \bigcup_{i=0}^{\infty} P_i$$
$$P_0 = \{[(S,0)]\}$$
$$P_{i+1} = \{[(n,c+e) \mid p] \mid \exists p \in P_i.(frontier(p),e,n) \in R \wedge gcost(p) = c\}$$

b)

$$P_{G'} = \bigcup_{i=0}^{\infty} P'_i$$
$$P'_0 = \{[(S,0)]\}$$
$$P'_{i+1} = \{[(n,c+e) \mid p] \mid \exists p \in P'_i.\exists op \in Op.op(frontier(p)) = (n,e) \wedge gcost(p) = c\}$$

provided

$$(n,e,n') \in R \leftrightarrow \exists op_i \in Op.op_i(n) = (n',e)$$

c) uniform cost: expand path whose frontier node has lowest g-cost (from start to node)
best first: expand path whose frontier node has lowest h-cost (estimated cost from node to goal)
A*: expand path whose frontier node has lowest f-cost f = g + h

d) Admissible heuristic: never over estimate

straight-line, go this distance, and maybe more, therefore never over-estimate

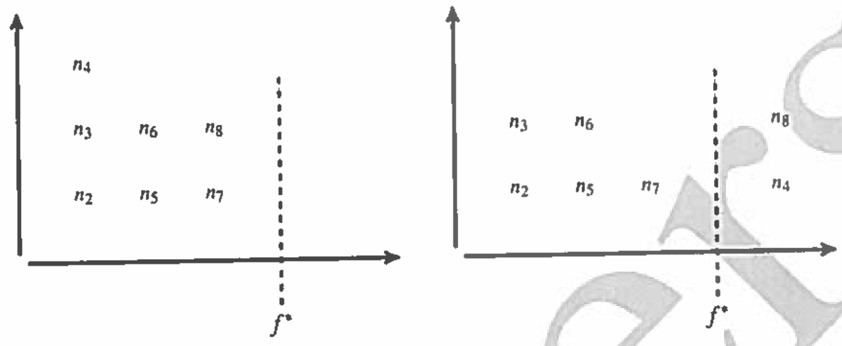
manhattan, go at least X1-X2 in X direction and Y1-Y2 in y-direction, and maybe more, therefore never over-estimate

e)

```
h( straightline, C1, C2, H ) :-
  sld( C1, C2, H ).

h( manhattan, C1, C2, H ) :-
  manhattan( C1, C2, H ).

sld( (X1,Y1), (X2,Y2), H )  :-
  Dx is (X1 - X2) * (X1 - X2),
  Dy is (Y1 - Y2) * (Y1 - Y2),
  H is sqrt( Dx + Dy ).

manhattan( (X1,Y1), (X2,Y2), H )  :-
  (X1 > X2 -> Dx is X1 - X2 ; Dx is X2 - X1),
  (Y1 > Y2 -> Dy is Y1 - Y2 ; Dx is Y2 - Y1),
  H is Dx + Dy.
```

f) • Imagine drawing a histogram with increasing f-cost along the x-axis
  - Then map all the nodes onto the histogram
  - A* will expand all those nodes to the left of the f^* bar
  - Therefore the 'trick' is to get as many nodes to the right of the f^* bar ...
  - ... without ever over-estimating the h-cost

Suppose with h_1 But with h_2



So straight-line distance is likely to be more efficient, since $h_{straight-line} \leq h_{manhattan}$

g)
- Search to full ply using depth first
- Apply heuristic evaluation to all siblings at ply (assume these are MIN nodes)
- Propagate value of siblings to parent using Minimax rules
- Offer this value to **grandparent** MIN node as possible *beta* cutoff
- Descend to other grandchildren
- Terminate (prune) exploration of parent if any of their values is greater than or equal to the *beta* cutoff
- Do the "same" for MAX nodes
- Two rules for terminating search
  - Search stopped below any MIN node having a *beta* value less than or equal to *alpha* value of any of its MAX ancestors
  - Search stopped below any MAX node having an *alpha* value greater than or equal to *beta* value of any of its MIN ancestors

Example: any sensible correct example will do

4\. a) Unification is a process of attempting to identify two symbolic expressions by the matching of terms and the replacement of certain sub-expressions (variables) by other expressions

resolution is an inference rule, general case:

$$\frac{\begin{array}{l}\neg q_1 \vee \neg q_2 \vee \ldots \vee \neg q_m \vee p_i \\ \neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_{i-1} \vee \neg p_i \vee \neg p_{i+1} \vee \ldots \vee \neg p_m\end{array}}{\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_{i-1} \vee (\neg q_1 \vee \neg q_2 \vee \ldots \vee \neg q_m) \vee \neg p_{i+1} \vee \ldots \vee \neg p_m}$$

skolemisation eliminate existential quantifiers in such a way that any interpretation that satisfies the original formula true also satisfies the skolemised formula and vice versa

b) yes X bound to a, Y bound to a

yes X bound to Z, Y bound to Z

yes X bound to A, Y bound to a

no Z bound to b and Y bound to a; then try to bind Y to X, and X to Z

c) $\forall x.\forall y.relation(x,y,c)$

$\forall x.\forall y.relation(x,y,sk(x,y))$

In the first case the existential is not in the scope of the universals so there is one c for every x and y so a skolem constant will do; in the second case, the particular individual that makes the formula true depends on the x and y so we require a skolem function of the universally quantified variables whose scope includes the existential quantifier (i.e. both x and y)

d)
$\neg property1(X1) \vee \neg property2(Y1) \vee \neg relation1(X1,Y1) \vee state1(X1)$
$\neg property1(X2) \vee \neg property3(Y2) \vee \neg relation2(X2,Y2) \vee state2(X2)$
$\neg state1(X3) \vee \neg state2(X3) \vee result(X3)$
$\neg prerelation1(X4,Y4) \vee relation1(X4,Y4)$
$\neg prerelation2(X5,Y5) \vee relation2(X5,Y5)$
$property1(a)$
$property2(b)$
$property3(c)$
$prerelation1(a,b)$
$prerelation2(a,c)$

e)
$\neg result(X3)[X3 \mapsto a]$
$\neg state1(a) \vee \neg state2(a)[X1 \mapsto a]$
$\neg property1(a) \vee \neg property2(Y1) \vee \neg relation1(a,Y1) \vee \neg state2(a)$
$\neg property2(Y1) \vee \neg relation1(a,Y1) \vee \neg state2(a)[y1 \mapsto b]$
$\neg relation1(a,b) \vee \neg state2(a)$
$\neg prerelation1(a,b) \vee \neg state2(a)[X4 \mapsto a, Y4 \mapsto b]$
$\neg state2(a)[X2 \mapsto a]$
$\neg property1(a) \vee \neg property3(Y2) \vee \neg relation2(a,Y2)$
$\neg property3(Y2) \vee \neg relation2(a,Y2)[Y2 \mapsto c]$
$\neg relation2(a,c)[X5 \mapsto a, Y5 \mapsto c]$
$\neg prerelation2(a,c)$
$\perp$

5. a) entailment between set of formulas and formula, follows form semantics, whenever all member of set are true, so is formula

proves relation is a relation between a set of formulas and a formula computed by a proof method

soundness – $\vdash \rightarrow \models$ – if prove a theorem then it is an entailment

completeness – $\models \rightarrow \vdash$ – if there is an entailment then it is provable

b) Translate and prove:

1	$\neg lockdoor \rightarrow (forget \vee lostkey)$	premise 1
2	$lostkey \rightarrow trouble$	premise 2
3	$memory \rightarrow \neg forget$	premise 3
4	$memory$	premise 4
5	$\neg(lockdoor \vee trouble)$	negated conclusion
6	$\neg lockdoor$	α, 5
7	$\neg trouble$	α, 5
8	$\neg forget$	β, 3, 4
9	$(forget \vee lostkey)$	β, 1, 6
10	$lostkey$	β, 9, 8
11	$trouble$	β, 2, 10
	$\times$	7, 11

c) The valid sequences are:

yes, yes, yes or no

no, yes or no, no

To see this, suppose we say yes to $q \vee \neg(p \vee r)$. Then we must say yes to $p \rightarrow q$. Try building a model with $\neg(p \rightarrow q)$:

$q \vee \neg(p \vee r)$
$\neg(p \rightarrow q)$
p
$\neg q$
$\neg(p \vee r)$
$\neg p$
$\neg r$
$\times$

Try building a model with $p \rightarrow q$ then first 'move' is to apply PB on q. This gives open branches with $q\bar{p}\bar{r}, qp\bar{r}, q\bar{p}r, qpr, \bar{q}\bar{p}\bar{r}$. This is why the answer to q is yes or no because there is still at least one branch which will make it q or $\neg q$ consistent.

If the candidate says no to the first formula, then adding the second is independent, but she has to say no the third:

$\neg(q \vee \neg(p \vee r))$
q
$\neg q$
$\neg\neg(p \vee r)$
$\times$

d) Proofs

1	$1 : \neg(\Diamond\Box p \rightarrow \Box p)$	negated conclusion
2	$1 : \Diamond\Box p$	α, 1
3	$1 : \neg\Box p$	α, 1
4	$2 : \Box p$	poss, 2
5	$3 : \neg p$	poss, 3
6	$3 : p$	ness, 4
	$\times$	5, 6

1	$1 : \neg(\Diamond\Diamond p \rightarrow \Diamond p)$	negated conclusion
2	$1 : \Diamond\Diamond p$	α, 1
3	$1 : \neg\Diamond p$	α, 1
4	$2 : \Diamond p$	poss, 2
5	$3 : p$	poss, 4
6	$3 : \neg p$	ness, 3
	$\times$	5, 6