E3.16
ISE3.23

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2004

EEE/ISE PART III/IV: MEng, BEng and ACGI

ARTIFICIAL INTELLIGENCE

Monday, 17 May 10:00 am

Time allowed: 3:00 hours

There are SIX questions on this paper.

Corrected Copy

Answer FOUR questions.

All questions carry equal marks

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible First Marker(s) : J.V. Pitt
Second Marker(s) : M.P. Shanahan

1. (a) To celebrate the end of exams, four students, Alice, Bill, Clive, and Danielle intend to eat dinner at El Prologo, a tapas restaurant in downtown South Kensington. They want to order five dishes from the menu to share. The dishes must satisfy the following requirements:

- Alice is a vegetarian, so at most one dish can use beef, chicken or pork;
- Bill likes spicy foods, so at least one dish must be spicy;
- Clive likes variety, so no two dishes can have the same main ingredient;

(i) Give an appropriate representation (in Prolog or other declarative notation) of the dishes;
(ii) Specify the initial state;
(iii) Specify the goal state;
(iv) Define the state change rule(s), in Prolog or other declarative notation;
(v) Define the constraints on the state change rules, in Prolog or other declarative notation.

[12]

(b) Danielle is poor, so the students decide to find an agreeable combination of five dishes with the least total cost.

(i) Explain what search strategy would be used;
(ii) Explain why this strategy would return the combination of dishes with the least total cost;
(iii) Explain, using definitions in Prolog or other declarative notation, the changes that need to be made to the basic General Graph Search Program to implement this strategy.

[6]

(c) Finally, the students decide to find an agreeable combination of as many dishes as they can, for less than £50. Explain how the state space is changed as a result of this decision.

[2]

2. Given an initial state and a goal state in the formulation of a problem, one search strategy is to search 'forwards' from the initial state and 'backwards' from the goal state. This search strategy is called bi-directional search.

   (a) What properties should hold of the search space and the problem formulation for bi-directional search to be a viable search strategy.

   [2]

   (b) Specify the condition that should hold to terminate bi-directional search.

   [2]

   (c) Modify the General Graph Search program so that it implements bi-directional search. Use Prolog or other declarative notation, but give an informal explanation of the algorithm in English. Include a definition of any extra clauses need to detect termination as specified in part (b). Indicate the form of a query to the modified program.

   [12]

   (d) State which search strategy should be used for the 'forwards' search, and which search strategy should be used for the 'backwards' search.

   [2]

   (e) Briefly evaluate bi-directional search with respect to the four criteria (optimality, completeness, time complexity and space complexity).

   [2]

3. The Bin-Packing problem is defined as follows. There are j bins, each of capacity c_j. There are k objects, each of volume v_k. The problem is to put the objects in the bins such that for each bin, the sum of the volumes of the objects in each bin is no more than the capacity of the bin.

For example, if $j = 3$ and $c_1 = 15$, $c_2 = 10$ and $c_3 = 20$, and there are 8 objects and their volumes are $v_1 = 10$, $v_2 = 8$, $v_3 = 6$, $v_4 = 6$, $v_5 = 4$, $v_6 = 4$, $v_7 = 3$ and $v_8 = 2$, then one solution is put v_1, v_7, and v_8 in bin1, v_3 and v_5 in bin2, and v_2, v_4, and v_6 in bin3.

(a) Given an initial state in which no objects have been put in any bins, formulate the state space for the Bin-Packing problem, in Prolog or other declarative notation, so that it could be used with the General Graph Search program. The formulation is required in two ways:
(i) by considering each bin in turn, and assigning objects to bins;
(ii) by considering each object in turn, and assigning a bin to each object.

[10]

(b) Given an initial state in which all the objects have already been randomly assigned to bins, in a possibly invalid configuration:
(i) define a relation *valid* which returns true if the assignment of objects is such that the sum of the volumes of the objects in each bin does not exceed its capacity;
(ii) use *valid* to define goal states;
(iii) state what state change rules could be used to search for a valid assignment;
(iv) propose a heuristic which could be used to guide the search using a best first search strategy;
(v) state, with justification, whether or not your heuristic would be usable in A* search.

[10]

4. (a) In a two-player game, a forced win is a finite sequence of moves (according to the rules of the game) which always ends in a win for the player making the first move.

Specify, in Prolog or other declarative notation, a check for a forced win. Annotate your answer with 'English' pseudo-code.

[4]

(b) The 23 Matchstick Game is played as follows. There are 23 matchsticks in a row. Players take it in turns to remove either 1, 2, or 3 matches. The loser is the player stuck with the last match.

Formulate the 23 Matchstick game, in Prolog or other declarative notation, so that it could be used with the General Graph Search Algorithm.

[4]

(c) Describe the Minimax Procedure for exhaustively searchable graphs.

[5]

(d) From the state where there are 6 matches left and it is max to move, show the search space and the state values assigned by the minimax procedure. Indicate the forced win for max, and use this to suggest a strategy for winning the game.

[5]

(e) If exhaustive search is not feasible, briefly describe alternative strategies for determining the next best move.

[2]

5. (a) Briefly define the following terms, as they relate to knowledge representation and automated reasoning:
   (i) Conceptualisation.
   (ii) Unification.
   (iii) Resolution.

[6]

(b) Consider the following five English statements:

s1. For any party, if everyone at the party enjoys it, then the party is a success.
s2. Everyone who dances during a party, enjoys it.
s3. Everyone who sings during a party, enjoys it.
s4. Everyone at party p1 either danced or sang.
s5. Party p1 was a success.

(i) Formalise these statements as sentences of First Order Predicate Logic;
(ii) Express these sentences as implicitly quantified disjunctions, stating which logical equivalences have been used;
(iii) State, with justification, whether or not these disjunctions are Horn Clauses;
(iv) Prove, using resolution, that statement s5 is a logical consequence of the other four.

[12]

(c) Comparing propositional logic with the predicate calculus as a language of knowledge representation and reasoning, briefly state the advantages and disadvantages of each.

[2]

6. (a) Define the syntax of propositional logic.

[2]

(b) Define the semantics of propositional logic.

[2]

(c) Convert the formula $\neg((p \rightarrow (q \vee r)) \rightarrow ((p \rightarrow q) \wedge (p \rightarrow r)))$ into disjunctive normal form. State the logical equivalence that is being used at each step in the conversion.

[6]

(d) Briefly explain the relationship between disjunctive normal form and automated reasoning with the calculus KE.

[4]

(e) Using the KE calculus for propositional logic, show whether the following formulas are either theorems or non-theorems. Annotate your proof with the rules used at each step. For a non-theorem, show a counter-model which makes the formula false.

(i) $(p \rightarrow (q \vee r)) \leftrightarrow ((p \rightarrow q) \vee (p \rightarrow r)))$
(ii) $(p \rightarrow (q \vee r)) \leftrightarrow ((p \rightarrow q) \wedge (p \rightarrow r)))$

[4]

(f) Specify the additional rules required to extend the KE calculus to predicate logic.

[2]

master 27/4/04

2004

E3.16 Artificial Intelligence **MODEL ANSWERS** **Dr J V Pitt**

1.
(a) 12 marks (Application)
(b) 6 marks (Application)
(c) 2 marks (Application)

(a)

```
dish( (name, list_of_ingredients_main_first, veggy_flag, spicy_flag, cost) )

initial_state( [ ] )

goal_state( L ) :-
        is_one_spicy( L ).

State_change( _, L, [D|L] ) :-
        dish( D ),
        veggie( [D|L] ),
        length( [D|L], Len ), Len < 6,
        variety( [D|L] ).

veggie( [D|L] ) :-
        count_veggies( [D|L], N ),
        N =< 1.


count_veggies( [], 0 ).

count_veggies( [D|L], N1 ) :-
        D = (_,_,veggy,_,_),
        count_veggies( L, N ),
        N1 is N + 1.

variety( L ) :-
        get_main_ingredients( L, M ),
        all_different( M ).


get_main_ingredient( [], [] ).

get_main_ingredient( [ (_,[M|_],_,_,_)|L1], [M|L2] ) :-
        get_main_ingredient( L1, L2 ).

all_different( [] ).

all_different( [H|T] ) :-
        not member( H, T ),
        all_different( T ).

is_one_spicy( [] ) :-
        !, fail.

is_one_spicy( [(_,_,_,spicy,_)|_] ) :-
        !.
```

```
is_one_spicy( [_|T] ) :-
        is_one_spicy( T )..
```

(b)

(i) uniform cost

(ii) breadth first search is uniform cost with cost of everything the same; breadth first search is optimal

(iii) assume still choose first path, changes required are insert in order in add to paths rather than append, compute total cost by cost function g (which is just add cost of new dish to accumulated cost of dishes, change node representation to include total cost.

(c)

Previously, the state space was constrained only to be five deep. Now can have longer chains which are only terminated when the total cost is greater than £50.

2.
(a) 2 marks (Application)
(b) 2 marks (Application)
(c) 12 marks (Application)
(d) 2 marks (Application)
(d) 2 marks (Application)

(a)
state space should be fully connected, so there is a path from the initial state to the goal for us to find
state change rules (operators) should all have inverses

(b)
when the state of a frontier node on the path chosen for expansion from the graph computed going forwards is the same state as a frontier node on the graph computed backwards, and vice versa

(c)
search(Fgraph, Bgraph, Direction, SolnPath) where
Fgraph is the graph computed from the initial state
Bgraph is the graph computed from the goal state
Direction is either forwards or backwards
SolnPath is the solution path

Informal description, on each recursive call,
If we are going forwards,
Choose a path from Fgraph
Get its state
If it is the same state as a frontier node on a path of Bgraph, found a solution path, and halt,
If it is not, compute where to go from this node, giving new paths, add these to Fgraph, change direction, and recursively call search/4

Query is search([[initial_state]], [[goal_state]], forwards, SolnPath)

```
search( Fgraph, Bgraph, forwards, SolnPath ) :-
        choose( Path, Fgraph, _ ),
        Path = [FN|RP], %Path is frontier node ++ rest of path
        State_of( FN, State ),
        is_fn_of( State, Bgraph, PathToGoal ),
        reverse( PathToGoal, RevPtoG ),
        result_is( RP, RevPtoG, SolnPath ). %append reverse of path to goal to
forward path to get solution path

search( Fgraph, Bgraph, forwards, SolnPath ) :-
        choose( Path, Fgraph, RestF ),
```

```
        one_step_extensions( Path, NewPaths ),
        add_to_paths( NewPaths, RestF, BigFgraph ),
        search( BigFgraph, Bgraph, backwards, SolnPath ) %change direction

is_fn_of( _, [], _ ) :-
        !, fail.
Is_fn_of( State, [Path|_], Path ) :-
        Path = [FN|_],
        State_of( FN, State ).
Is_fn_of( State, [_|Paths], Path ) :-
        Is_fn_of( State, Paths, Path ).
```

(d)
Of course any strategy could be used in principle, but breadth first would be the preferred strategy.

(e)
Not necessarily optimal, the best path going in one direction may connect with the worst path coming from the other direction
Complete, if search space connected as above
Time/Space complexity O(b^(d/2)) in one direction, ditto the other, so O(b^(d/2))

3.
(a) 10 marks (Application)
(b) 10 marks (Application)

(a)
(i)
state representation
each bin a 2-tuple (bin-x, list-of-objects-in-the-bin)
all bins is a list of these 2-tuples
objects is just a list
state is a 2 tuple (list of bin 2-tuples, list of objects)

initial state
([(bin1,[]), (bin2,[]), ..., (binj,[])], [o1,o2, ..., ok]

goal_state
(Assignment, []).

```
state_change( ( Bins, [O|Bjects] ), (NewBins, Bjects) ) :-
        append( Front, [(Bin, Asgnd)|Back], Bins ),
        capacityof( Bin, Cbin ),
        volumeof( O, Vobj ),
        sum_assigned( Asgnd, Vasgnd ),
        Vall is Vasgnd + Vobj,
        Cbin >= Vall,
        append( Front, [ (Bin, [O|Asgnd]) | Back], NewBins ).

Sum_assigned( [], 0 ).
sum_assigned( [H|T], Vall ) :-
        volumeof( H, Vh ),
        sum_assigned( T, Vt ),
        Vall is Vh + Vt.
```

(ii)
state representation
each object a 2-tuple (obj-x, bin)
all objects is a list of these 2-tuples
assigned objects is another list
state is a 2 tuple (list of obj 2-tuples, list of assigned objects)

initial state
([(obj1,B1), (obj2,B2), ..., (objk,Bk)], []

goal_state
([], Assignment).

```
state_change( ([(Obj,Bin)|Objs],Assigned), (Objs, [(Obj,Bin)|Assigned]) ) :-
        bin( Bin, Cbin )
        volumeof( O, Vobj ),
        sum_assigned_v2( Assigned, Bin, Vasgnd ),
        Vall is Vasgnd + Vobj,
        Cbin >= Vall.

sum_assigned_v2( [], _, 0 ).
sum_assigned_v2( [(Obj,Bin)|T], Bin, Vall ) :-
        volumeof( Obj, Vobj ),
        sum_assigned_v2( T, Bin, Vt ),
        Vall is Vobj + Vt.
sum_assigned_v2( [ _ |T], Bin, Vt ) :-
        sum_assigned_v2( T, Bin, Vt ).


Sum_assigned( [], 0 ).
sum_assigned( [H|T], Vall ) :-
        volumeof( H, Vh ),
        sum_assigned( T, Vt ),
        Vall is Vh + Vt.
```

(b)
(i)

```
valid( [] ).
valid( [(Bin,Objs)|T] ) :-
        sum_assigned( Objs, Vobjs ),
        capacityof( Bin, Cbin ),
        Cbin >= Vobjs,
        valid( T ).
```

(ii)

```
goal_state( A ) :- valid( A ).
```

(iil)
state change rules to either move an obj from one bin to another or swap two objs

(iv) count the number of bind over capacity

(v) no because it is not admissible.

4.
(a) 4 marks (Bookwork/Application)
(b) 4 marks (Application)
(c) 5 marks (Bookwork)
(c) 5 marks (Application)
(c) 2 marks (Bookwork/Application)

(a)
A move from state A to B is a forced win, if
A => B is a valid move
for every move B => C,
C =>* goal

```
forced_win( S, NewS ) :-
        state_change( S, NewS ),
        find_all( NextS, state_change( _, NewS, NextS ), OppM ),
        check_every( OppM ).

check_every( [] ).
check_every( [Move|Moves] ) :-
        search( Move, _ ),
        check_every( Moves ).
```

(b)
initial state (23, max)
goal state (1, min)

```
state_change( _, (N,M), (N1,O) ) :-
        N1 is N – 1,
        opponent( M, O ).
state_change( _, (N,M), (N2,O) ) :-
        N2 is N – 2,
        opponent( M, O ).
state_change( _, (N,M), (N3,O) ) :-
        N3 is N – 3,
        opponent( M, O ).

opponent( max, min ).
opponent( min, max ).
```

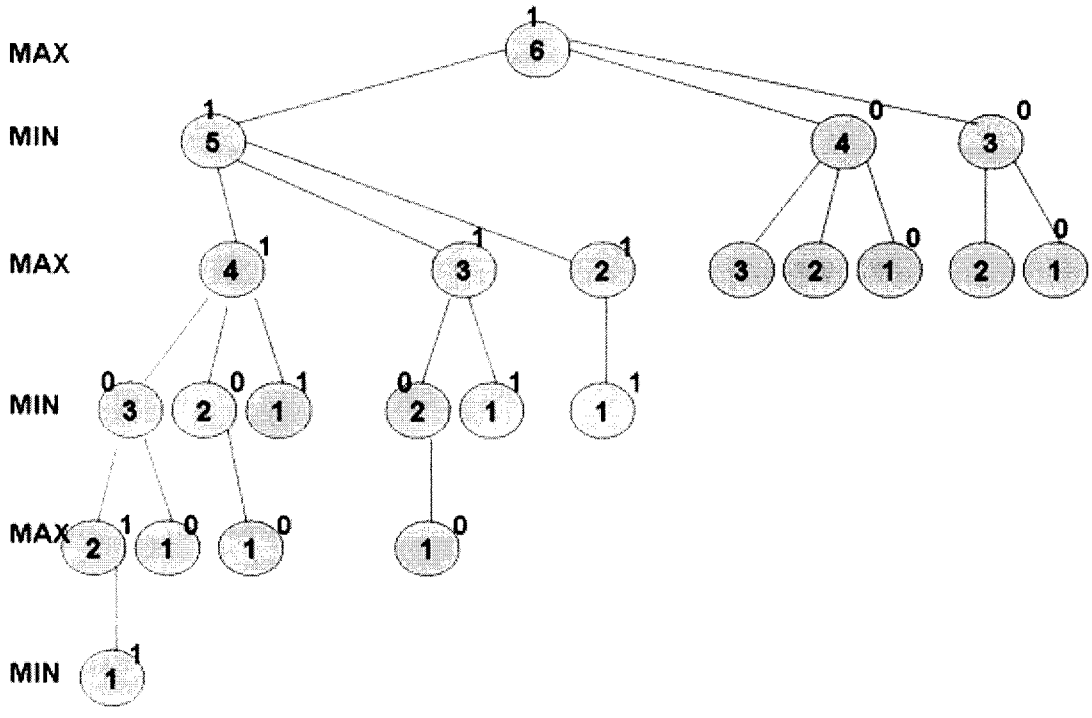
(c)
max is player trying to win, or MAXimize advantage
min is opponent who attempts to MINimize max's score. Assume that min uses the same information as max and attempts always to move to a state that is worst for max

each leaf node is given a score of 1 or 0, depending on whether the state is a win for max or min respectively

generate the graph

propagate leaf values up the graph according to the rules:
if the parent is a MAX, give it the minimum value of its children
if the parent is a MIN, give it the maximum value of its children

(d)



Winning strategy is to go for N MOD 4 matches, + 1

(e)
use minimax to fixed ply
use alphabeta search and heuristice

5
(a) 6 marks, Bookwork
(b) 14 marks, Application

(a)

conceptualization: process on concept formalisation in knowledge representation, identifying objects in domain of discourse, relations between and functions on them.

Unification: substitution of values for variable that makes two logical terms equivalent, algorithm...

Resolution: inference rule, give example

(b)

(i) in fopl

$\forall p\ .\ (\forall x\ .\ at(x,p) \rightarrow enj(x,p)) \rightarrow succ(p)$

$\forall x\ .\ \forall p\ dance(x,p) \rightarrow enj(x,p)$

$\forall x\ .\ \forall p\ sing(x,p) \rightarrow enj(x,p)$

$\forall x\ .\ at(x,p1) \rightarrow (dance(x,p1) \vee sing(x,p1))$

$succ(p1)$

(ii) as implicitly quantified disjunctions

the first sentence is:

$(\forall x\ .\ at(x,P) \rightarrow enj(x,P)) \rightarrow succ(P)$

$\neg(\forall x\ .\ at(x,P) \rightarrow enj(x,P)) \vee succ(P)$

$\exists x\ .\ \neg\ (at(x,P) \rightarrow enj(x,P)) \vee succ(P)$

skolemizing gives

$\neg\ (at(sk(P),P) \rightarrow enj(sk(P),P)) \vee succ(P)$

$\neg\ (\neg at(sk(P),P) \vee enj(sk(P),P)) \vee succ(P)$

$(at(sk(P),P) \wedge \neg enj(sk(P),P)) \vee succ(P)$

$(at(sk(P),P) \vee succ(P)) \wedge (\neg enj(sk(P),P)) \vee succ(P))$

giving

1 $at(sk(P),P) \vee succ(P)$

2 $\neg enj(sk(P),P)) \vee succ(P)$

the rest are straightforward

3 $\neg dance(X, P) \vee enj(X, P)$

4 $\neg sing(X, P) \vee enj(X, P)$

5 $\neg at(x,p1) \vee (dance(x,p1) \vee sing(x,p1))$

6 $\neg succ(p1)$

(iii)

the first four sentences are horn clauses, the last one isn't (more than one positive literal)

(iv)

the resolutions go:

7 $at(sk(p1),p1)$ *from* 6 *and* 1

8 $\neg enj(sk(p1),p1)$ *from* 6 *and* 2

9 $\neg dance(sk(p1),p1)$ *from* 8 *and* 3

10 $\neg sing(sk(p1),p1)$ *from* 8 *and* 4

11 $\neg at(sk(p1),p1) \vee dance(sk(p1),p1)$ *from* 5 *and* 10

12 $\neg at(sk(p1),p1)$ *from* 11 *and* 9

contradiction, from 7 *and* 12

(c)

Predicate logic is more expressive (structure and quantifiers), propositional logic more tractable (decidable rather then semi-decidable)

6.
(a) (b) 4 marks, Bookwork (c) 6 marks, Application
(d) 4 marks, Bookwork (e) (f) 6 marks, Application

(a)
F ::= atom | F & F | F + F | F -> F | F <-> F | ¬F

(b)
semantics given by assignments of true/false to atomic symbols, truth tables

(c)

$\neg((p \rightarrow (q \vee r)) \rightarrow ((p \rightarrow q) \wedge (p \rightarrow r)))$	
$\neg(\neg(p \rightarrow (q \vee r)) \vee ((p \rightarrow q) \wedge (p \rightarrow r)))$	*elim* $\rightarrow$
$\neg(\neg(\neg p \vee (q \vee r)) \vee ((p \rightarrow q) \wedge (p \rightarrow r)))$	*elim* $\rightarrow$
$\neg(\neg(\neg p \vee (q \vee r)) \vee ((\neg p \vee q) \wedge (\neg p \vee r)))$	*elim* $\rightarrow$
$\neg\neg(\neg p \vee (q \vee r)) \wedge \neg((\neg p \vee q) \wedge (\neg p \vee r))$	*push* $\neg$ *in*
$(\neg p \vee q \vee r) \wedge \neg((\neg p \vee q) \wedge (\neg p \vee r))$	$\neg\neg$ *elim*
$(\neg p \vee q \vee r) \wedge \neg(\neg p \vee q) \vee \neg(\neg p \vee r))$	*push* $\neg$ *in*
$(\neg p \vee q \vee r) \wedge (p \wedge \neg q) \vee (p \wedge \neg r)$	*push* $\neg$ *in*
$((\neg p \vee q \vee r) \wedge (p \wedge \neg q)) \vee ((\neg p \vee q \vee r) \wedge (p \wedge \neg r))$	*distr* $\wedge$ *over* $\vee$
$((\neg p \wedge p \wedge \neg q) \vee (q \wedge p \wedge \neg q) \vee (p \wedge \neg q \wedge r)) \vee ((\neg p \vee q \vee r) \wedge (p \wedge \neg r))$	*distr* $\wedge$ *over* $\vee$
$(p \wedge \neg q \wedge r) \vee ((\neg p \vee q \vee r) \wedge (p \wedge \neg r))$	$p \wedge \neg p = ff, ff \vee X = X$
$(p \wedge \neg q \wedge r) \vee ((\neg p \wedge p \wedge \neg r) \vee (q \wedge p \wedge \neg r) \vee (r \wedge p \wedge \neg r))$	*distr* $\wedge$ *over* $\wedge$
$(p \wedge \neg q \wedge r) \vee (q \wedge p \wedge \neg r)$	$p \wedge \neg p = ff, ff \vee X = X$

(d)
KE builds tree. Each branch of tree is conjunction of formulas. Tree is disjunction of conjunctions, hence DNF. If every branch has a contradiction, then the DNF is logically equivalent to false. If the DNF of a formula is logically equivalent to false, it must have 0 in every row of the truth table. If we try to construct the KE tree for –F and find it does not have a DNF, then every row of the truth table for _F must be 0, so then every row of the truth table for F must have a 1. Therefore F is a theorem (tautology, true under every assignment, etc.)

(e)
(i)

1	$\neg((p \rightarrow (q \vee r)) \leftrightarrow ((p \rightarrow q) \vee (p \rightarrow r)))$	*premise*
2	$p \rightarrow (q \vee r)$	*PB*1.1
3	$\neg((p \rightarrow q) \vee (p \rightarrow r))$	$\leftrightarrow$, 1, 2
4	$\neg(p \rightarrow q)$	*a*, 3
5	$\neg(p \rightarrow r)$	*a*, 3
6	p	*a*, 4
7	$\neg q$	*a*, 4
8	$\neg r$	*a*,5
9	$q \vee r$	*b*, 2, 6
10	r	*b*, 7, 9

close 8, 10

11	$\neg(p \rightarrow (q \vee r))$	*PB*1.2
12	$(p \rightarrow q) \vee (p \rightarrow r)$	$\leftrightarrow$, 1, 11
13	p	*a*, 11
14	$\neg(q \vee r)$	*a*, 11
15	$\neg q$	*a*, 14
16	$\neg r$	*a*, 14
17	$p \rightarrow q$	*PB*2.1
18	$\neg p$	*b*, 15, 17
close 13, 18		
19	$\neg(p \rightarrow q)$	*PB*2.2
20	$p \rightarrow r$	*b*, 12, 19
21	$\neg p$	*b*, 16, 20
close 13, 21		

(ii)

1	$\neg((p \rightarrow (q \vee r)) \leftrightarrow ((p \rightarrow q) \wedge (p \rightarrow r)))$	*premise*
2	$p \rightarrow (q \vee r)$	*PB*1.1
3	$\neg((p \rightarrow q) \wedge (p \rightarrow r))$	$\leftrightarrow$, 1, 2
4	$p \rightarrow q$	*PB*2.1
5	$\neg(p \rightarrow r)$	*b*, 3, 4
6	p	*a*, 5
7	$\neg r$	*a*, 5
8	q	*b*, 4, 6
9	$q \vee r$	*b*, 2, 6
open branch (*note* 9 *is beta simplified*) *model* = {p=*tt*, q=*tt*, r=*ff*}		
10	$\neg(p \rightarrow q)$	*PB*2.2
11	p	*a*, 10
12	$\neg q$	*a*, 10
13	$(q \vee r)$	*b*, 11, 2
14	r	*b*, 12, 13
open branch model = {p=*tt*, q=*ff*, r=*tt*}		
15	$\neg(p \rightarrow (q \vee r))$	*PB*1.1
16	$(p \rightarrow q) \wedge (p \rightarrow r)$	$\leftrightarrow$, 1, 2
17	p	*a*, 15
18	$\neg(q \vee r)$	*a*, 15
19	$\neg q$	*a*, 18
20	$\neg r$	*a*, 18
21	$p \rightarrow q$	*a*, 16
22	$p \rightarrow r$	*a*, 16
23	q	*b*, 17, 21
close 19, 23		

Note that the two open branches are the same as the DNF of part (c)

(f)

$\forall x.P \,/\, P[x = c]$, *where c can be any constant*

$\exists x.P \,/\, P[x = c]$, *where c must be a new constant on the branch*