E3.16
ISE3.23

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2007

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

Corrected Copy

ARTIFICIAL INTELLIGENCE

Tuesday, 1 May 10:00 am

Time allowed: 3:00 hours

There are SIX questions on this paper.

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) : Y.K. Demiris

The Questions

1 a) In the sequent calculus, a sequent is composed of two sets of formulas, call them Delta and Gamma. To use the sequent calculus for proof by refutation, we aim to make all of the formulas in Delta true, and all of the formulas in Gamma false. Then we *search* for a valid sequent, i.e. a sequent that has the same literal formula in common, i.e. a formula p that is in both Delta and Gamma. In the construction of the proof, if every end-sequent is a valid sequent, then there is no assignment that satisfies the starting sequent (i.e. makes it false), and so every assignment must make it true.

You are required to formulate the proof rules of the sequent calculus so that they are expressed as state change rules, which in turn can be used with the General Graph Search program, to implement a proof checker for Propositional Logic.

(i) Specify a representation of logical formulas as terms.

(ii) Specify a representation of sequents (bearing in mind the need to distinguish between the formulas and the literals in Delta and Gamma).

(iii) Specify a representation of the state for the search.

(iv) Specify the initial state and the goal state.

(v) Specify the state change rules: for current purposes it is only necessary to specify rules for: finding a valid sequent, finding a literal in Delta, finding an and-formula in Delta, and finding an and-formula in Gamma. (Note: you should **not** use *append* 'backwards'.)

[16]

b) If these state change rules were to be used with a Prolog implementation of the General Graph Search program, justify which 'version' of the program you would use.

[4]

2 a) Explain the difference between uniform cost, best first, and A* search algorithms. State the difference between the three algorithms' time/space complexity.

[4]

b) Explain why A* search is optimal and complete.

[6]

c) In general, given two admissible heuristics for A* search, explain why one heuristic may be more 'efficient' ('better informed') than the other.

[4]

d) A graph G is defined by a 3-tuple $G = \langle N,E,R \rangle$, where N is a set of nodes, E is a set of edges, and R is the incidence relation.

Give an implicit definition of a Graph G, and state under what condition the implicit definition is equivalent to the explicit definition as a 3-tuple.

Give an inductive definition of the paths of a graph, and explain how the General Graph Search (GGS) program computes a (partial) representation of a graph G using the A* search algorithm.

[6]

3 a) Describe the Minimax Algorithm for two-player games.

[4]

b) The game of Abyss is played on a 5-by-5 board. To begin with, one player places 6 white stones and 6 black stones on empty cells. Then, the other player may: either, choose a colour and be the second to move; or, be the first to move and the opposing player chooses a colour.

Formulate these rules as state change rules. You may assume a relation *random*(5,*N*) which generates a random number between 1 and 5, to use in the placement of the 12 stones.

[10]

c) Play then proceeds as follows: on each turn, the player must select a stone of his/her colour, and slide it in one direction, until at least one stone of the opposite colour is pushed off the board.

When one player cannot perform a valid move, the game ends. The player with the most stones left is the winner.

Decide, with justification, if Minimax is a suitable algorithm for solving this problem. Briefly describe two alternative approaches.

[6]

4 a) Define the syntax and semantics of First Order Predicate Logic.

[8]

b) Consider the following English statements:

A traveller can go from one city to another, if the first city is connected to the second.
A traveller can go from one city to another, if the first city is connected to a third city and the traveller can go from the third city to the second.
There is a city that is connected to London and is connected to Manchester.

Express these statements as formulas of First Order Predicate Logic.

Transform these formulas into a set of Horn clauses, explaining the steps in the transformation.

Prove, using resolution and showing the unifiers, that a traveller *Bob* can *go* from *London* to *Manchester*.

[8]

c) Suppose the third statement in part (b) was replaced by the statements:

There is a city that is connected to London.
There is a city that is connected to Manchester.

Explain (using a model and an inference rule for existential elimination) why it would now be unsafe to conclude *go(Bob, London, Manchester)*.

[4]

5 a) Discuss four properties for evaluating proof systems for propositional logic.

[4]

b) Using normal forms for tautology checking, show that:

$$\{X \vee A, Y \vee \neg A\} \models X \vee Y$$

Explain the result.

[4]

c) Using the proof system KE, show that:

$$\{X \vee A, Y \vee \neg A\} \models X \vee Y$$
$$\{P \vee Q, P \rightarrow R, Q \rightarrow S\} \models R \vee S$$

Annotate the proof to show the derivation at each step.

[4]

d) Using the proof system KE, show that:

$$\models (P \rightarrow (Q \rightarrow R)) \leftrightarrow ((P \wedge Q) \rightarrow R)$$

Annotate the proof to show the derivation at each step.

[4]

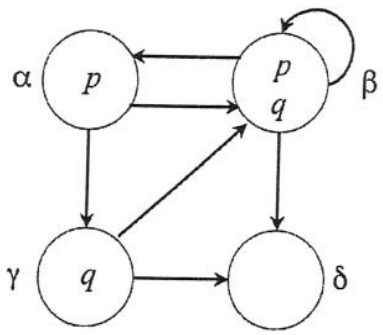
e) Explain the relationship between the proof method in parts (c) and (d), the proof in part (d) itself, and the Deduction Theorem.

[4]

6 a) Briefly describe the BDI (Belief-Desire-Intention) Agent architecture, giving a schematic representation, an explanation of the major components, and the operation of the BDI interpreter cycle. Indicate how the agent could combine reasoning and planning.

[8]

b) Consider the following diagram of a set of possible worlds.



Give the Kripke model represented by this diagram.

[4]

c) Given the Kripke model M of part (b), say, with justification, whether each of the following formulas of modal logic are true or false:

(i) $M,a \models \Box(p \vee q)$

(ii) $M,\beta \models \Diamond\Diamond\Diamond(p \wedge q)$

(iii) $M,\gamma \models \Box\Box(p \rightarrow q)$

(iv) $M,\delta \models \Box(p \wedge \neg p)$

[4]

d) Briefly justify which of the standard axiom schema are appropriate for reasoning with the modality B in a BDI agent.

[4]

Solutions - 2007 EJ-16 / Isc J.27

Marks - 20/4/07

1
(a) application

P & Q as and(P, Q) [or define with operators]

Sequent representation: 4-tuple
Each member of the tuple is a list of formulas
Formulas in Delta, Literals in Delta, Formulas in G, Literals in G.

State representation: List of sequents

Initial state
[(DeltaF,[],DeltaG,[])]

Goal State
[ ]

State change rules

```
State_change( close, [ Sequent | Rest], [NewSequent | Rest] ) :-
        Sequent = (DFor, DLit, GFor, GLit),
        Member( P, DLit ),
        Member( P, Glit ).

State_change( move_lit, [ Sequent | Rest], Rest ) :-
        Sequent = ( [Lit|DFor], DLit, GFor, GLit),
        Literal( Lit ),
        NewSequent = (For, [Lit|Dlit], Gfor, Glit ).

State_change( and_elim [ Old_sequent | Rest ], [New_sequent | Rest ] ) :-
        Old_sequent = (DFor, DLit, GFor, GLit),
        DFor = [and(P,Q)|Rest],
        DFornew = [P,Q|Rest],
        New_Sequent = (Dfornew, DLit, GFor, GLit).

State_change( and_elim [ Old_sequent | Rest ], [New_seq1, New_seq2 | Rest ] ) :-
        Old_sequent = (DFor, DLit, GFor, GLit),
        GFor = [and(P,Q)|Rest],
        DFornew1 = [P|Rest],
        GFornew2 = [Q,Rest],
        New_Seq1 = (DFor, DLit, GFornew1, GLit),
        New_Seq2 = (DFor, DLit, GFornew2, GLit).
```

(b) [understanding]

Finding a proof in sequent calculus proof is not really 'search' (in the problem solving sense) at all. Although in each state, there are many steps we can take, i.e. it is non-deterministic, all the steps end up in the same place, so long as we apply all the rules to all the formulas. So we want at each step to apply just one rule, and the only branching we allow are the branches of the proof tree, not of the search graph.

So if this were Prolog, we would use the cut down version of the depth-first search, and use Prolog to save any backtracking points. But we know there won't be any

2

(a)

Uniform cost expands the frontier node on the path the least accumulated cost from the start node, as given by the path cost function g

Best first expands the frontier node on the path with the least estimated cost to a/the goal node, as given by a heuristic cost function h

A* expands the frontier node on the path, the least estimated path cost through the node from the start node to the goal node, given by summing f = g + h

There is no difference, the number of nodes expanded is still exponential in the length of the path, although it is possible to get sub-exponential growth in A* search for certain types of heuristic

(b)

Optimality:

Optimal solution has cost f* to get to optimal goal G

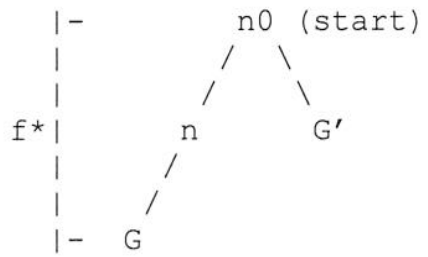
Suppose A* search returns path to sub-optimal goal G'

We show that this is impossible

f(G')	=	g(G') + h(G')	
	=	g(G') + 0	G' is a goal state, we require h to be 0
	=	g(G')	

If G' is sub-optimal then g(G') > f*

Now consider a node n on path to optimal solution G



Then:	f*	≥	f(n)	monotonicity
	f(n)	≥	f(G')	otherwise A* expands n first
	f*	≥	f(G')	transitivity of [3]
	f*	≥	g(G')	a contradiction

So either G' was optimal or A* does not return a sub-optimal solution.

Completeness:

A* expands nodes in order of increasing f-cost

Each expansion has lower bound > 0

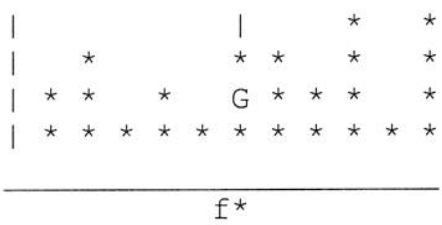
So A* must eventually expand all nodes n with f(n) less than or equal to f*, one of which must be a goal state

(unless there are an infinite number of nodes with f(n) < f*, or infinitie number of noides with finite total cost

(c)

Let f* be cost of optimal node.

A* expands all nodes with f-cost less than f*.
A* expands some nodes with f-cost = f*.
A* expands no nodes with f-cost > f*.
Since f(n) = g(n) + h(n), this means that A* expands all those nodes such that h(n) < f* - g(n).
In other words, the more nodes for which this relation holds, the more nodes will be expanded by A* using this heuristic, and the less efficiently will the search space be explored.
Alternatively, consider histogram of nodes according to actual f-cost, whereby f-actual(n) = g(n) + h-actual(n).



Only those nodes to the left of the f* bar will be expanded. In practice of course, we don't have h-actual we just have h. Thus we could have a redistribution of the histogram which pushes more of the nodes to the left of the f* bar.
In other words, we want to ensure f* < f(n) + h(n) < f-actual(n)
With the two heuristics given, the second is more informed.

(d)
G' = < start_node, Op > where
start_node is the root node
op is set of state transformers op: node -> (edge,node)

Op has to compute every element of the incidence relation.

What we then need to store with each path we create is f, the path cost function g plus the estimated path cost to the goal given by the heuristic function h.

Then our inductive definition of the graph is given by:
$P'_G = \bigcup_{i=0}^{\infty} P'_i$
where
$P'_0 = \{ (<start>, 0) \}$
$P'_{i+1} = \{ (p_i \mathbin{++} <n_{i+1}>, f) \mid \exists\, op \in Op \,.\, \exists\, (p_i,g) \in P'_i \,.\, (n_{i+1},e) = op(frontier(p_i)) \text{ AND } f = g + e + h(n_{i+1}) \}$

Let f* be the actual cost of getting from the start state to the optimal goal state. Then the A* algorithm constructs:

All the paths (p,f) in whichever P'_i such that f < f*
Some of the paths (p,f) in whichever P'_i such that f = f*
None of the paths (p,f) in any P'_i such that f > f*
At each step, A* selects the path in (p,f) from whichever P'_i such that this f is least so far, and expands that.

3
(a) [bookwork]
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
exhaustively 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

(b)[application]
State: (whose turn, which colour, Board, state)

Board list of 2 lists, location of white stones and black stones respectively as 2-tuples.

```
State_change( minplace, (_,_,[ [], [] ], minplacing), (_,_, [White,Black],
maxchoosing) ) :-
        Generate_12_locations( 0, [], Locations, ),
        Append( White, Black, Locations ),
        Length( White, 6 ).

Generate_12_locations( 13, Locations, Locations ).
Generate_12_locations( N, LSofar, Result ) :-
        N < 13,
        Random( 5, X ), random( 5, Y),
        Not member( (X,Y), LSofar ),
        N1 is N + 1,
        Generate_12_locations( N1, [ (X,Y) | LSoFar ], Result ).

%max chooses black, min to move (as white), then we're playing game
State_change( maxchoose, (_,_,Board,maxchoosing), (min,white,Board,playing) ).

%max chooses white, min to move (as black), then we're playing
State_change( maxchoose, (_,_,Board,maxchoosing), (min,black,Board,playing) ).

%max choose to go second, make minchoose a colour
State_change( maxchoose, (_,_,Board,maxchosing), (max,_,Board,minchoosing) ).

%min chooses white
State_change( minchoose, (max,_,Board,minchoosing), (max,black,Board,playing) )

%min chooses black
State_change( minchoose, (max,_,Board,minchoosing), (max,white,Board,playing) )
```

(c) [application/bookwork]
For the first move, let's say by max, there are 6 stones to choose, each of which may go 4 directions. Thus there may be up to 24 new states (although some of the moves may not be valid). When a stone is pushed off, the opponent has 5 stones each with 4 directions, after which max has 5 stones, each with 4 directions, and so on.

This gives in the limit number of states to check is 24 x 20 x 20 x 16 x 16 x …

It's a big number, and we have not considered the number of possible outcomes for placing the stones (although we can reasonably consider starting the analysis after the stones have been placed) and picking the colour…

Therefore exhaustive search is not practical.

Alternatives are:
Minimax to fixed ply, and heuristic function to evaluate states
Alphabeta to prune the search tree and double the 'lookahead' (although the game can only last 11 moves at most…).

4
(a) [bookwork]

Define the terms:
* variables, x, y, z are terms
* constant symbols are terms
* if f is a function of arity n, and t1, t2, …, tn are terms, then f(t1, t2, … tn) is a term

Define the formulas:
* if p is a relation symbol (predicate) of arity n and t1, t2, …, tn are terms, then p(t1, t2, … tn) is a formula
* if P is a formula and Q is is formula, then –P, P & Q, P + Q, P > Q, P < Q are formulas
* if x is a variable and P is a formula, then $\forall x.P$ and $\exists x.P$ are formulas.

Define the semantics: truth with respect to an interpretation.

An intepretation of a language L is a pair <D,I> such that
- D is a non-empty set of individuals
- I is a mapping such that:
  - If c is a constant symbol, then I[c] e D
  - If f is a function symbol, then I[f] e Dn > D
  - If p is a predicate symbol, the I[p] e Dn

A valuation V of L over D is a function from variable symbols of L into D.

Then define satisfaction (truth of a formula relative to an interpretation) by:
- atomic formulas
- logical formulas
- quantified formulas

(b) [application]

$\forall x \forall a \forall b.\ connected(a, b) \rightarrow go(w, a, b)$
$\forall w \forall a \forall b \forall c.\ connected(a, b) \wedge go(w, b, c) \rightarrow go(w, a, c)$
$\exists x.\ connected(London, x) \wedge connected(x, Manchester)$

$f1$ $\quad \neg connected(a, b) \vee go(w, a, b)$
$f2$ $\quad \neg connected(a, b) \vee \neg go(w, b, c) \vee go(w, a, c)$
$f3$ $\quad connected(London, Sko)$
$f4$ $\quad connected(Sko, Manchester)$

$\quad \neg go(bob, London, Manchester)$

resolve with $f2$, *unifier* $\{w = Bob, a = London, c = Manchester\}$
$\quad \neg connected(London, b) \vee \neg go(Bob, b, Manchester)$
resolve with $f3$, *unifier* $\{b = Sko\}$
$\quad \neg go(Bob, Sko, Manchester)$
resolve with $f1$, *unifier* $\{w = Bob, a = Sko, b = Manchester\}$
$\quad \neg connected(Sko, Manchester\}$
resolve with $f4$,
$\quad contradiction$

(c) [understanding]
Some of the models which make the premises true do not necessarily make the conclusion true, specifically those models where it is a different 'x' that makes each of the existential statements true.

So, for example, a model with Sko1 and Sko2 will satisfy the two statements

$\exists x.\ connected(London, x)$
$\exists x.\ connected(x, Manchester)$

with $connected(London, Sko1)$ and $connected(Sko2, Manchester)$
but go(Bob, London, Manchester) is not a true statement in this model.

This is why the inference rule for existential elimination must introduce a new constant:

$\exists x.\ connected(London, x)$
$\exists x.\ connected(x, Manchester)$
$connected(London, Sko1)$
$connected(Sko2, Manchester)$

not

$\exists x.\ connected(London, x)$
$\exists x.\ connected(x, Manchester)$
$connected(London, Sko1)$
$connected(Sko1, Manchester)$

5
(a)
sound: |- implies |=
a proof in the system is an entailment in the language
essential

complete |= implies |-
an entailment in the language can be proved by the system
desirable

decidable
theorem algorithm terminates with yes
non-theorem algorithm terminates with no
desirable but can't do much about the langaueg (e.g. predicate logic semi-decidable)

efficient
deal with exponential complexity

(b)

$$
\begin{aligned}
& (X \vee A) \wedge (Y \vee \neg A) \wedge (X \vee Y) \\
= \quad & \neg((X \vee A) \wedge (Y \vee \neg A)) \wedge (X \vee Y) \\
= \quad & \neg(X \vee A) \vee \neg(Y \vee \neg A) \wedge (X \vee Y) \\
= \quad & (\neg X \wedge \neg A) \vee (\neg Y \vee \neg\neg A) \wedge (X \vee Y) \\
= \quad & (\neg X \wedge \neg A) \vee (\neg Y \vee A) \wedge (X \vee Y) \\
= \quad & (\neg X \wedge \neg A) \vee (\neg Y \vee X \vee Y) \wedge (X \vee Y \vee A) \\
= \quad & (\neg X \wedge \neg A) \vee \mathit{true} \wedge (X \vee Y \vee A) \\
= \quad & (\neg X \wedge \neg A) \vee (X \vee Y \vee A) \\
= \quad & (\neg X \vee X \vee Y \vee A) \wedge (\neg A \vee X \vee Y \vee A) \\
= \quad & \mathit{true} \wedge \mathit{true} \\
= \quad & \mathit{true}
\end{aligned}
$$

Semantically equivalent to true

c)

1	$X \vee A$	premise
2	$Y \vee \neg A$	*premise*
3	$\neg(X \vee Y)$	*¬ conclusion*
4	$\neg X$	*a* 3
5	$\neg Y$	*a* 3
6	A	*b* 1 4
7	$\neg A$	*b* 2 5
8	*close*	6 7

1	$P \vee Q$	*prem*
2	$P \rightarrow R$	*prem*
3	$Q \rightarrow S$	*prem*
4	$\neg(R \vee S)$	*¬ conc*
5	$\neg R$	*a* 4
6	$\neg S$	*a* 4
7	$\neg P$	*b* 2 5
8	$\neg Q$	*b* 3 6
9	Q	*b* 1 7
10	*close*	8 9

d) $\vdash ((P \rightarrow (Q \rightarrow R)) \leftrightarrow ((P \wedge Q) \rightarrow R)$ $\neg$ *conc*

branch 1

2	$P \rightarrow (Q \rightarrow R)$	*PB* 1
3	$\neg((P \wedge Q) \rightarrow R)$	*e* 1 2
4	$P \wedge Q$	*a* 3
5	$\neg R$	*a* 3
6	P	*a* 4
7	Q	*a* 4
8	$Q \rightarrow R$	*b* 2 6
9	R	*b* 8 7
10	*Close*	5 9

Branch 2

11	$\neg(P \rightarrow (Q \rightarrow R))$	*PB* 1
12	$(P \wedge Q) \rightarrow R$	*e* 1 11
13	P	*a* 11
14	$\neg(Q \rightarrow R)$	*a* 11
15	Q	*a* 14
16	$\neg R$	*a* 14
17	$\neg(P \wedge Q)$	*b* 12 14
18	-Q	b 13 17
19	close	15 18

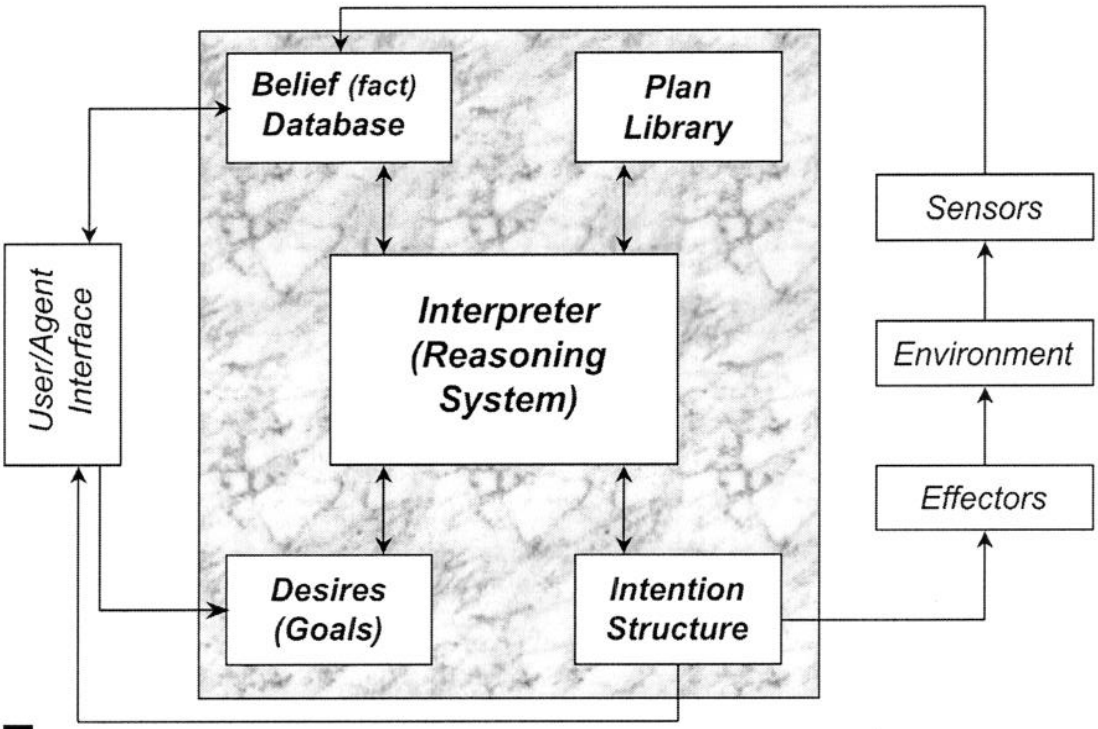
(*d*)
Proof method of KE
Write down premises at root of tree
Write down negated conclusion

This is because we are trying to prove S |= p
Where S is a set of premises and p is a conclusion
Let S' be distributed-& over S
Then this is S' |= p
Which by deduction theorem is |= S' > p
and to make false –(S' > P)
is false only if S' is true and p is false
applying and rule to S' is writing down premises and negating conclusion

proof of part d shows deduction theorem holds for all valuations for one formula….

6

Beliefs-Desires-Intentions (BDI) architecture
has its origins in the study of mental attitudes
beliefs: represent the agent's informational state
desires: represent the agent's motivational state
intentions: represent the agent's deliberative state



beliefs
current 'knowledge' about state of the 'world', some aspects of internal state
include facts about static properties of application domain
others acquired by agent as it executes plans
may need to represent meta-level beliefs and beliefs of other agents
desires (goals)
conditions over some interval of time (or sequence of world states)
can then have goals to achieve, test, maintain and wait for a condition
plans
how to act when certain facts added to belief db, or new goals acquired
consist of: invocation, context and maintenance conditions, & a body
used to create instances of the plan to be executed
intentions
intention structure contains all those tasks the system has chosen for execution
single intention is a top level plan instance, plus sub-plans
intention structure can contain a number of such intentions (partial order)
agent is committed to achieve goals, but may reconsider commitments

At a particular time *t*
Certain goals are established
Certain beliefs are held
An event occurs
New goals established
New beliefs held
Combination of beliefs and desires
Invoke (trigger) various plans

One or more applicable plans placed o Intention Structures
One executable intention selected and executed
Cycle repeats

Reasoning on the 'left', planning on the 'right'. Use reasoning to establish goals, use planning to device sequence of actions to achieve goals.

(b)

$M = \langle W, R, \| \rangle$

$W = \{\alpha,\beta,\gamma,\delta\}$
$R = \{ \alpha R \beta, \alpha R \gamma, \beta R \alpha, \beta R \beta, \beta R \delta, \gamma R \beta, \gamma R \delta \}$
$|p| = \{ \alpha, \beta \}$
$|q| = \{ \beta, \gamma \}$

(c)
(i) true
all the worlds accessible from a are b and g
p or q is true is both

(ii) true
p and q is true at b
b is accessible from a, so $\Diamond(p \wedge q)$is true at a
a is accessible from b, so $\Diamond\Diamond(p \wedge q)$ is true at b
b is accessible from b, so $\Diamond\Diamond\Diamond(p \wedge q)$ is true at b
(or could go a to g to b, or b to b to b)

(iii) false
b is accessible from y
a, b, and g are accessible from b
p > q is true in b and g but not in a

(iv) true
anything is necessarily true in a world with no successor

(d)
agent believes p, if p is a tautology (way of making agent 'rational')
K axiom: agent accepts all logical implications of its beliefs (believe p > q) and believe p then believe q
T axiom (reflexivity): agent's beliefs are always true in the world it finds itself in (believe p > p)
4 axiom (transitivity): positive introspection, agent believe p implies agent believes "agent believes p"; basis of doxastic logic
T and 4 together give epistemic logic (logic of knowledge)