E3.16
ISE3.23

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2006

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

ARTIFICIAL INTELLIGENCE

Corrected Copy

Monday, 24 April 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) The job-shop scheduling problem requires that m jobs, each job requiring time t to process, are assigned to n machines.

Formulate the search space for the job-shop scheduling problem, in Prolog or other declarative notation so that it can be used with the General Graph Search (GGS) program to generate solutions.

[5]

b) Justify which of the five following `uninformed' search algorithms you would (or would not) use to generate solutions: depth-first, breadth-first, uniform-cost, iterative-deepening depth-first, beam.

[5]

c) Suppose the requirement was to find a 'good enough' solution to the job-shop scheduling problem. Explain how you would modify the search space formulation to do this; give a metric which you would use to evaluate solutions, and specify this metric (in Prolog or other declarative notation).

[8]

d) Supposing it was possible to run many instances of the General Graph Search program in parallel, suggest an alternative approach to finding a 'good enough' solution, using only 'uninformed' search and the search space formulation of part (a).

[2]

2 a) Explain, using an example, how a search space can be represented as a graph comprising nodes, edges, and an incidence relation.

[4]

b) Give an explicit definition of the paths in a graph based on the graph representation given in part (a).

[2]

c) Give an implicit definition of a graph, and use that to give an inductive definition of the nodes, incidence relation, and paths in the graph. State the condition(s) that need to be satisfied for the inductive definition of the paths to be equivalent to the explicit definition given in part (b).

[6]

d) Explain, with reference to the definition in part (c), how the General Graph Search (GGS) program computes a representation of a graph using the A* search algorithm.

[8]

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

[6]

b) Consider the search space shown in Figure 3.1. Show which branches are pruned by AlphaBeta search, and explain why.

[6]

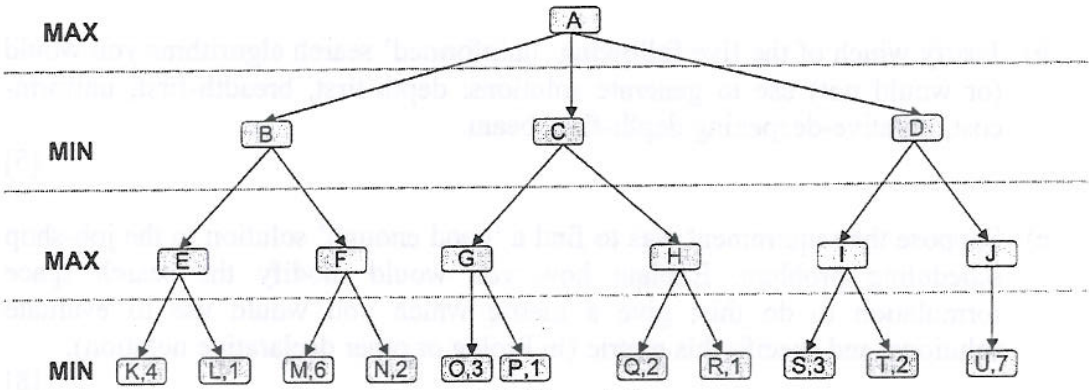


Figure 3.1: Search Space

c) Describe a Reinforcement Learning Algorithm for path-finding, e.g. in grid-like mazes.

[4]

d) Consider the grid maze in Figure 3.2. Assuming a 'follow left wall' exploration strategy, show the map constructed after the robot has found the goal location G.

[4]

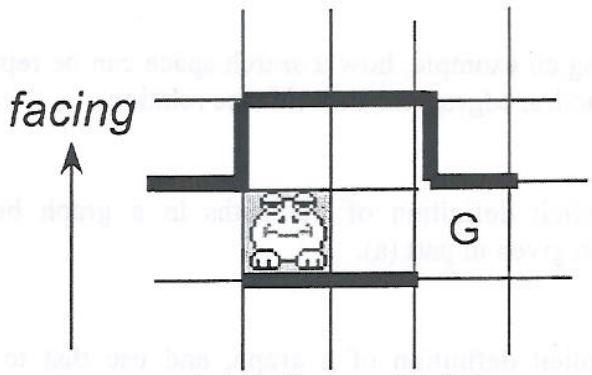


Figure 3.2: Robot in a Grid Maze

4 a) Consider the following statements for representing knowledge about recommending garden trees.

An appropriate tree for your garden should be the right size and safe for children. If your garden is small then a small tree is the right size. If you have children, then a non-poisonous tree is appropriate. Laburnum, Yew and Maple are small trees, while Cypresses and Sycamores are large. Laburnum and Yew are poisonous.

Represent this knowledge as a set of Horn Clauses.
Suppose I have a small garden and 2 children. Show how these Horn Clauses might be used to check that a Maple tree is appropriate for my circumstances. What would be required if I had a big garden, 3 children, and wanted to know which trees were appropriate?

[10]

b) The following English statement:

For any committee meeting, if everyone on the committee is present, then the committee meeting is quorate.

can be represented in First Order Predicate Logic (FOPL) as:

$\forall c.(\forall p.on(p,c) \rightarrow present(p)) \rightarrow quorate(c)$

Express this sentence as one or more implicitly quantified disjunctions, explaining the steps in the transformation.

Give a FOPL representation of the English statement:

For any committee meeting, the committee meeting is quorate only if everyone on the committee is present.

[6]

c) Assume a domain of discourse with two objects, *Tom* and *Jerry*. You are told the two following statements (are true): *Everything is a cat or a mouse. Tom is not a mouse*. Explain why neither of the two inferences *"Jerry is a cat"* and *"Jerry is a mouse"* is valid. State the valid inference.

[4]

5 a) Show the truth table for the exclusive-or operator $\otimes$.

[2]

b) Extend the KE-Calculus with rules to eliminate the $\otimes$-operator.

[4]

c) Suppose we had three options, a, b, and c, only one of which is true. We might write this: $(a \otimes b) \otimes c$.

Build a KE-tree (tableau) for $\{((a \otimes b) \otimes c), a\} \mid\!- \{\neg b \wedge \neg c\}$. Explain the result.

[4]

d) You are told: *There are three doors. Behind one (and only one) door there is a goat, behind the other two doors there is nothing*. Formalise this situation using 3 propositional symbols a, b, and c, in a single statement of propositional logic (i.e. as a disjunction of conjunctions, without using either $\leftrightarrow$ or exclusive-or).

Taking this statement as one premise, a as a second premise, show, using the KE proof procedure, that $\neg b \wedge \neg c$.

[6]

e) Each door has a message pinned on it. On Door A, it says: *"The goat is not behind Door B"*. On Door B, it says: *"The goat is behind Door A"*. On Door C, it says: *"The goat is not behind Door A"*. You are told that only one of these statements is true.

Formalise the three statements in propositional logic, and using the fact that only one statement is true, reduce the statements to their simplest form. Show, using the KE proof procedure, that the goat is behind Door B.

[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

[8]

b) Specify a cyclic Contract Net Protocol. For each 'state' of the protocol, indicate the institutional powers and obligations of each agent according to its role.

[8]

c) Briefly describe how a BDI agent from part (a) could use the specification from part (b) to inform/determine its behaviour.

[4]

E3.16/IC3.23

SOLUTIONS - ARTIFICIAL INTELLIGENCE

2006 Master Copy 10/4/06

1

(a)

State representation (jobs,machines) where
jobs is a list of integers, each integer representing time t to process the job
machines is a list of lists, each list being jobs assigned to machine

Initial state ([t1, t2, …, tm], [[], [], …, []]) %length of list of empty lists is n
Goals state ([], J2M) % all jobs assigned

```
State_change( assign, ([Job | Jobs], Machines), (Jobs, NewMachines) :-
        Append( Front, [Machine | Back], Machines ),
        NewMachine = [Job | Machine ],
        Append( Front, [NewMachine | Back ], NewMachines ).
```

(b)

Depth first – no, the way it is formulated, the first solution found will assign all the jobs to the first machine…
Breadth – no, excessive space requirements
Uniform-cost – no, no cost involved in making assignment
Iterative-deepening depth-first breadth first result with depth first algorithm, and we would no use either of those
Beam – no, risk of throwing out solutions

(c)

Modification: goal state evaluates found solution and accepts it if is better than some threshold

```
Goal_state( ( [], A ) :-
        Evaluate( A, Quality ),
        Quality =< Threshold. %whatever threshold is set to
```

Metric: minimise the difference between earliest and latest finishing times of all the machines

```
Evaluate( A, Q ) :-
        Sum_all_jobs( A, Esofar ),    %the earliest finishing time if all jobs are
                                      %assigned to one machine
        eval( A, Esofar, 0, Earliest, Latest ),
        Q is Latest – Earliest.

Eval( [], Earliest, Latest, Earliest, Latest ).

Eval( [M | Rest], Esofar, Lsofar, Earliest, Latest ) :-
        Sum_jobs( M, Time ),
        Earlier( Esofar, Time, E ),
        Later( Lsofar, Time, L ),
        Eval( Rest, E, L, Earliest, Latest ).
```

```
Earlier( E, T, T ) :-
        T < E.
Earlier( E, T, E ).

Later( L, T, L ) :-
        T > L.
Later( L, T, L ).
```

(d)
Take an agent-like approach, run multiple copies of the GGS and one evaluation process, generate random orders of jobs, assign them to each GGS-process, when they generate solutions, send them to eval-process to evaluate.

2

(a)

Search space can be considered as set of states. Each state is node of a graph. States can be transformed one into another. States related by such transformation are encoded in incidence relation R. We're not interested in edges. Therefore search space can be represented as G = <N, E, R> where n is the set of nodes, E is the set of edges and R is the incidence relation (node x edge x node).
Example can be anything reasonable…

(b)

$P_G = \bigcup_{i=0}^{\infty} P_i$

where

$P_0 = \{ \langle start \rangle \}$

$P_{i+1} = \{ p_i ++ \langle n_{i+1} \rangle \mid \exists\, p_i \in P_i \,.\, (frontier(p_i), e, n_{i+1}) \in R \}$

where ++ is append

frontier function gives last node in a sequence

(c)

G' = < start_node, Op > where

start_node is the root node

op is set of state transformers op: node -> (edge,node)

$N_G = \bigcup_{i=0}^{\infty} N_i$

where

$N_0 = \{ \langle start \rangle \}$

$N_{i+1} = \{ n_{i+1} \mid \exists\, op \in Op \,.\, \exists\, n_i \in N_i \,.\, (n_{i+1},e) = op(n_i) \}$

$R_G = \bigcup_{i=1}^{\infty} R_i$

where

$R_1 = \{ (n_0, e, n_1) \mid \exists\, op \in Op \,.\, (n_i,e) = op(n_0) \}$

$R_{i+1} = \{ (n_i, e, n_{i+1}) \mid \exists\, op \in Op \,.\, \exists\, n_i \in N_i \,.\, (n_{i+1},e) = op(n_i) \}$

$P'_G = \bigcup_{i=0}^{\infty} P'_i$

where

$P'_0 = \{ \langle start \rangle \}$

$P'_{i+1} = \{ p_i ++ \langle n_{i+1} \rangle \mid \exists\, op \in Op \,.\, \exists\, p_i \in P'_i \,.\, (n_{i+1},e) = op(frontier(p_i)) \}$

The inductive definitions gives us the same set of paths as before provided the operators compute all elements of the incidence relation.

(d)

The value of e in the incidence relation is of course g, the path cost function.

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

$P'_0 = \{ (\langle start \rangle, 0) \}$

$P'_{i+1} = \{ (p_i +\!+ \langle n_{i+1} \rangle, f) \mid \exists\, op \in Op \,.\, \exists\, (p_i, g) \in P'_i \,.\, (n_{i+1}, e) = op(frontier(p_i)) \text{ AND } f = g + e + h(n_{i+1}) \}$

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)

➢ **Opponents in a games are called MAX and MIN**

➢ **MAX is player trying to win or MAXimise advantage**

➢ **MIN is player trying to stop MAX winning, or to MINimize MAX's advantage**

➢ **Assume MIN uses same information as MAX and always moves to a state that is worst for MAX**

➢ **In alpha-beta search**

➠Associate one of two values with each node

—Alpha value, associated with MAX nodes, which can never decrease
•Alpha is the 'least' MAX can get, given MIN will do its best to minimise MAX's value
-- Beta value, associated with MIN nodes, which can never increase
• Beta is the 'most' MAX can get, given MIN will do its best to minimise MAX's value

➢ **Algorithm**

➠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

—If MIN nodes, back up the maximum value
➠Offer this value to ***grandparent*** MIN node as possible beta cutoff
➠Descend to other grandchilren
➠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

(b)
search depth first to ply
h(k) := 4
e_alpha := 4
h(l) = 1
e_alpha stays 4
b_beta := 4
search depth first to ply
h(m) = 8
f_alpha := 8
MAX node with alpha value greater than beta-value of MIN ancestor, therefore right hand branch under F is pruned
a_alpha := 4

search depth first to ply
h(o) = 3
h(p) = 1
g_alpha := 3
c_beta := 3
MIN node with beta value less than alpha-value of MAX ancestor, therefore right hand branch under C is pruned
Same happens on third branch and right hand branch under D is pruned

(c)

➢ **adopt basic strategy**

➠ e.g. random walk, follow left wall, …, with some mechanism for loop checking

➢ **initialise** (starting grid location)

➠ assign a coordinate value (say (0,0))

➠ set reward to 0.0 and visited to true

➠ perceive and record the result of performing action A (go up, down, left or right)

— update pointers, set rewards to 0.0, visited to false, assign relative co-ordinate

➢ **explore**

➠ choose next direction to go in and move one grid location in that direction

➠ set visited for to 'true' for that grid location

➠ perceive and record the result of performing action A (go up, down, left or right)

— update pointers, set rewards to 0.0, visited to false, assign relative co-ordinate

➠ repeat until exit location is found

➢ **assign** (reward or credit assignment)

➠ set reward of exit location, when found, to 10

➠ set reward of each visited grid location to 0.9 * maximum reward of any grid location that can be moved to by doing action A in that location

➠ propagate values back until all visited grid locations are assigned a reward (with loop checking…)

(d)

location	north	south	east	west	visited	value
0,0	0,1	nil	1,0	-1,0	true	0.81
-1,0	nil	nil	nil	nil	false	0.0
1,0	1,1	nil	2,0	0,0	true	0.9
0,1	nil	0,0	1,1	nil	true	0.729
1,1	nil	1,0	nil	0,1	true	0.81
2,0 (G)	nil	2,-1	3,0	1,0	true	10.0
2,-1	nil	nil	nil	nil	false	0.0
3,0	nil	nil	nil	nil	false	0.0

4

(a)

$appropriate(T, P) \vee$
$\quad \neg\, rightsize(T, P) \vee$
$\quad \neg\, safe(T, P)$

$rightsize(T, P) \vee$
$\quad \neg\, small\_garden(P, G) \vee$
$\quad \neg\, small\_tree(T)$

$safe(T, P) \vee$
$\quad \neg\, children(P, N) \vee$
$\quad \neg\, N > 0 \vee$
$\quad \neg\, nonpoisonous(T)$

$small\_tree(laburnum)$
$small\_tree(yew)$
$small\_tree(maple)$

$nonpoisonous(maple)$
$nonpoisonous(cypress)$
$nonpoisonous(sycamore)$

$small\_garden(jeremy, g).$
$children(jeremy, 2).$

Use refutation proof and backward chaining

Query: $\neg\, appropriate(maple, Jeremy)$

$\neg\, appropriate(maple, jeremy)$
$\neg\, rightsize(maple, jeremy) \vee \neg\, safe(maple, jeremy)) \; [T = maple, P = jeremy]$
$\neg\, small\_garden(jeremy, G) \vee \neg\, small\_tree(maple) \vee \neg\, safe(maple, jeremy)$
$\neg\, small\_tree(maple) \vee \neg\, safe(maple, jeremy) \; [G = g]$
$\neg\, safe(maple, jeremy)$
$\neg\, children(jeremy, N) \vee \neg\, N > 0 \vee \neg\, nonpoisonous(maple) \; [N = 2]$
$\neg\, 2 > 0 \vee \neg\, nonpoisonous(maple)$
$\neg\, nonpoisonous(maple)$
contradiction

Express rules to add and delete facts, then use forward chaining.

(b)

$\forall c.(\forall p.on(p,c) \rightarrow present(p)) \rightarrow quorate(c)$
$\forall c.\neg(\forall p.on(p,c) \rightarrow present(p)) \vee quorate(c)$
$\forall c.\neg(\forall p.\neg on(p,c) \vee present(p)) \vee quorate(c)$
$\forall c.(\exists p.\neg(\neg on(p,c) \vee present(p)) \vee quorate(c)$
$\forall c.(\exists p.(on(p,c) \wedge \neg present(p)) \vee quorate(c)$
$\exists p.(on(p,C) \wedge \neg present(p)) \vee quorate(C)$
$on(sk(C),C) \wedge \neg present(sk(C)) \vee quorate(C)$
$on(sk(C),C) \vee quorate(C)$
$\neg present(sk(C)) \vee quorate(C)$

In other words, if everyone on the committee is present, then it can't NOT be quorate, i.e you can't have, for any C, on(sk(C),C), present(sk(C)) AND -quorate(C)

$\forall c.\ quorate(c) \rightarrow (\forall p.on(p,c) \rightarrow present(p))$

(c)
$\forall x.\ cat(x) \vee mouse(x)$
$\neg mouse(tom)$

$cat(jerry)$ is not valid because model could be $\{cat(tom), mouse(jerry)\}$
$mouse(jerry)$ is not valid because the model could be $\{cat(tom), cat(jerry)\}$

The correct inference is $cat(tom)$ with the substitution $x = tom$

5

(a)

p	$\otimes$	q
0	0	0
0	1	1
1	1	0
1	0	1

(b)

$p \quad \otimes \quad q$
p

$\neg q$

$p \quad \otimes \quad q$
$\neg p$
---
q

$\neg(p \otimes q)$
p
---
q

And five others.

(c)

premise	$(a \otimes b) \otimes c$	1
premise	a	2
$\neg$ *conc*	$\neg(\neg b \wedge \neg c)$	3
*PB*1	$\neg c$	4
β,3,4	$\neg\neg b$	5
$\neg\neg$,5	b	6
$\otimes$,1,4	$(a \otimes b)$	7
$\otimes$,6,7	$\neg a$	8
close,2,8	x	
*PB*2	c	9
$\otimes$,1,9	$\neg(a \otimes b)$	10
$\otimes$,2,11	b	11
open		

All formulas have been analysed on the open branch, even 3 which beta-simplified by formula 9.

The reason is, that, as truth tables will show, $(a \otimes b) \otimes c$ is true if just one is true, AND if all three of them are true.

(d)
$(a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c)$

premise	$(a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c)$	1
premise	a	2
$\neg$ *conc*	$\neg(\neg b \wedge \neg c)$	3
*PB*1	$(a \wedge \neg b \wedge \neg c)$	4
a,4	a	5
a,4	$\neg b$	6
a,4	$\neg c$	7
β,3,7	$\neg\neg b$	8
$\neg\neg$,8	b	9
close,6,9	x	
*PB*2	$\neg(a \wedge \neg b \wedge \neg c)$	10
β,1,10	$(\neg a \wedge b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c)$	11
*PB*2.1	$(\neg a \wedge b \wedge \neg c)$	12
a,12	$\neg a$	13
a,12	b	14
a,12	$\neg c$	15
close,2,13		
*PB*2.2	$\neg(\neg a \wedge b \wedge \neg c)$	16
β,11,16	$(\neg a \wedge \neg b \wedge c)$	17
a,12	$\neg a$	18
a,12	b	19
a,12	$\neg c$	20
close,2,18		

(e)

$(\neg b \wedge \neg a \wedge a)$
$\vee$ $(b \wedge a \wedge a)$
$\vee$ $(b \wedge \neg a \wedge \neg a)$

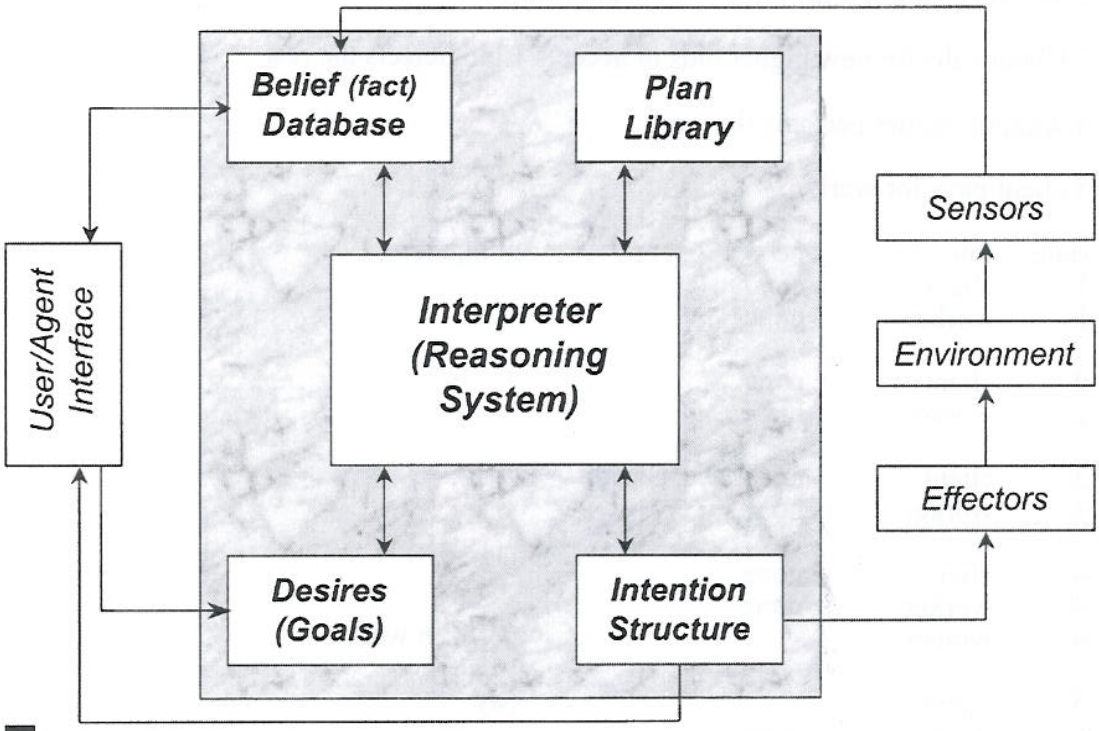
$(\neg b \wedge \neg a \wedge a) \leftrightarrow \neg b \wedge false \leftrightarrow false$, $false \vee X \leftrightarrow X$
$A \wedge A \leftrightarrow A$, *for all* A

Therefore this simplifies to $(\neg a \wedge b) \vee (a \wedge b)$

premise	$(\neg a \wedge b) \vee (a \wedge b)$	1
$\neg$*conc*	$\neg b$	2
*PB*1	$\neg a \wedge b$	3
a,3	$\neg a$	4
a,3	b	5
close,2,5	x	
*PB*2	$\neg(\neg a \wedge b)$	6
β,1,6	$a \wedge b$	7
a,3	a	8
a,3	b	9
close,2,9	x	

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

(b)
1 Client announces to (n) workers call for proposals

2 Workers submit bids, reject, or bid after timeout

3 Client calls for new higher bids or accepts 1 bid, rejects the rest

4 winning bidder perfoms the work

5 client pays for work

state	role	power	obligation
1	client	announce	
1	worker	none	
2	client	none	
2	worker	bid	
3	client	announce, accept	
3	worker	none	
4	client	none	
4	worker	none	
4	winner		perform work
5	client		pay
5	worker	none	
5	winner	none	

note assertive force of announce (client can pay) and bid (worker can do)

(c)
require objective reasoning module incorporated e.g. in C+ or event calculus
detect communicative act
calculate new normative positions using orm
use updated beliefs to formulate new action (plans)