AI COMP SOLUTIONS

1. KNOWLEDGE REPRESENTATION [16 points]

(a) **[3 points]** Translate each of the following sentences into first-order logic. Use the following vocabulary: the predicate symbols Workstation(x), Monitor(x), Component(x,y) (x is a component of y), Belongs(x,p) (x belongs to p), and At(x,y) (x is at location y); the function symbol Office-of(p); and the constant symbols Tom and Sun.[1]

i. **[1 points]** All workstations come with a monitor.
ii. **[1 points]** Any workstation belonging to a person is in that person's office.
iii. **[1 points]** The workstation Sun belongs to Tom.

i. $\forall x(Workstation(x) \rightarrow \exists y(Monitor(y) \wedge Component(y,x)))$
ii. $\forall x,p((Workstation(x) \wedge Belongs(x,p)) \rightarrow At(x, Office\text{-}of(p)))$
iii. $Workstation(Sun) \wedge Belongs(Sun, Tom)$

(b) **[2 points]** Can you prove from these axioms that there is a monitor in Tom's office? If not add any (consistent) axioms necessary for proving this fact.

No. We also need the following axiom:

$$\forall x,y,l((At(x,l) \wedge Component(y,x)) \rightarrow At(y,l))$$

(c) **[4 points]** Translate each of the above sentences, including the negated query, into clausal form. For brevity, you may use (here and below) the first initial of the various vocabulary symbols to represent them (e.g., you can use $C(x,y)$ instead of $Component(x,y)$). You may use either disjunctive normal form or implicational normal form.

The query is $\exists z(M(z) \wedge A(z,O(T)))$, so its negation is $forall z(\neg M(z) \vee \neg A(z,O(T)))$.

(1) $\neg W(x_1) \vee M(f(x_1))$
(2) $\neg W(x_2) \vee C(f(x_2),x_2)$
(3) $\neg W(x_3) \vee \neg B(x_3,p) \vee A(x_3,O(p))$
(4) $W(S)$
(5) $B(S,T)$
(6) $\neg A(x_4,l) \vee \neg C(y,x_4) \vee A(y,l)$
(7) $\neg M(z) \vee \neg A(z,O(T))$

(d) **[7 points]** Using the clauses you have generated, use refutation resolution to prove that there exists a monitor in Tom's office.

Answer format: number the clauses in your axioms, as well as any clauses generated during your proof; for each step in your proof, specify the numbers of the clauses resolved, any variable unifications needed, and the resulting clause. For brevity, you may resolve more than two clauses in one resolution step.

(9)	$A(S,O(T))$	(3),(4),(5)	$[x_3/S,\ p/T]$
(10)	$C(f(S),S)$	(2),(4)	$[x_2/S]$
(11)	$A(f(S),O(T))$	(6),(9),(10)	$[x_4/S,\ l/O(T),\ y/f(S)]$
(12)	$M(f(S))$	(1),(4)	$[x_1/S]$
(13)	false	(7),(11),(12)	$[z/f(S)]$

[1] If we were being careful, you would also have a Person(x) predicate. However, for simplicity we'll ignore that.

2. **PROBABILITY [8 points]**

Consider the following Bayesian network (influence diagram) over three binary-valued variables:

P(smoking) = 0.2

P(emphysema | smoking) = 0.6

P(emphysema | no smoking) = 0.05

P(dyspnea | emphysema) = 0.9

P(dyspnea | no emphysema) =0.15

(a) **[2 points]** Name one conditional independence assumption which is encoded in the structure of this network.

Dyspnea is conditionally independent of smoking given emphysema.

(b) **[6 points]** Show how you would compute *P(smoking | dyspnea)* in this network. For brevity, you may use the abbreviations s, e, and d for the events smoking, emphysema, and dyspnea, and the abbreviations $\neg s$, $\neg e$, and $\neg d$ for their negations. A formula for the answer is fine; you do not have to compute the final numerical answer.

$$
\begin{aligned}
P(d \mid s) &= P(d \mid e, s)P(e \mid s) + P(d \mid \neg e, s)P(\neg e \mid s) && (1)\\
&= P(d \mid e)P(e \mid s) + P(d \mid \neg e)P(\neg e \mid s) && (2)\\
&= 0.9 \cdot 0.6 + 0.15 \cdot 0.4 = 0.6 && (3)\\
P(d \mid \neg s) &= P(d \mid e)P(e \mid \neg s) + P(d \mid \neg e)P(\neg e \mid \neg s) && (4)\\
&= 0.9 \cdot 0.05 + 0.15 \cdot 0.95 = 0.1875 && (5)\\
P(d) &= P(d \mid s)P(s) + P(d \mid \neg s)P(\neg s) && (6)\\
&= 0.6 \cdot 0.2 + 0.1875 * 0.8 = 0.27 && (7)\\
P(s \mid d) &= \frac{P(d \mid s)P(s)}{P(d)} && (8)\\
&= \frac{0.6 \cdot 0.2}{0.27} = 0.0324 && (9)
\end{aligned}
$$

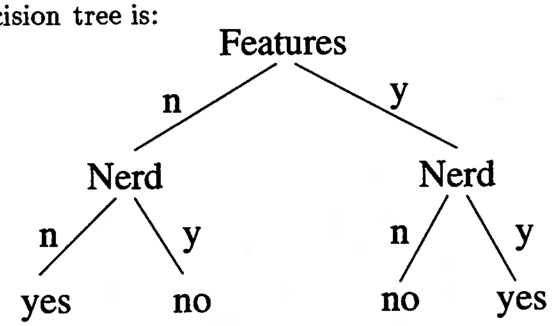
3. **LEARNING [10 points]**

The marketing department of Microsquish Corporation is trying to construct a classifier which will tell them whether a customer will like a piece of software. From responses to survey forms, they have collected the following data:

Form #	customer is computer nerd	s/w has lots of features	customer liked s/w
1	yes	yes	yes
2	yes	yes	yes
3	yes	yes	yes
4	yes	yes	yes
5	yes	yes	yes
6	yes	yes	yes
7	yes	yes	yes
8	yes	no	no
9	yes	no	no
10	no	yes	yes
11	no	yes	no
12	no	yes	no
13	no	no	yes
14	no	no	yes
15	no	no	no

(a) **[2 points] What Boolean function would do the best job of classifying these examples?**
The best Boolean function would be the negation of the exclusive or function, i.e., it would say "yes" when both variables are true or both variables are false, and "no" otherwise.

(b) **[5 points] What decision-tree, including classifications, would be output by an ID3-style decision-tree learning algorithm? Explain or show your computations.**
The optimal decision tree is:



The reason for splitting on the number of features first is as follows. If we split on the features (F), the distribution of positive and negative examples is 2:3 on the "no" branch and 8:2 on the "yes" branch. If we split on nerdiness (N), the distribution is 3:3 on the "no" branch and 7:2 on the "yes" branch. Even without going through the math, it's clear that the information gain is higher if we split on F. Formally, the entropy along each branch for the F split is lower than the entropy along the corresponding branch for the N split, and therefore the weighted average entropy is also lower, so that the information gain is higher.

(c) **[3 points] Is it possible to construct a neural network with a single thresholding element (i.e., a perceptron) which classifies these examples as well as the decision tree? If so, show the parameters of the thresholding unit. If not, explain why not.**
A single perceptron is limited to describing functions that are linearly separable. Since the NXOR function is not linearly separable, we cannot represent it using a perceptron, so we cannot achieve the optimal accuracy on this set of examples.

4. **SEARCH [14 points]**

(a) **[9 points]** Consider the problem of heuristic search in a space where our heuristic function h' is almost, but not quite, admissible. More precisely, we have that for each node n, $h'(n) \leq h(n) + \epsilon$. In this case, we are not guaranteed that the first goal node returned by the A* algorithm will be the *optimal* goal node. Explain briefly how you could extend the A^* algorithm in order to find the optimal goal node n^*. (Hint: Consider the f-value of n^*.)

We know from the assumptions and from the definitions that:

$$f(n^*) = g(n^*) + h'(n^*) \tag{10}$$

$$h'(n^*) \leq h(n^*) + \epsilon \tag{11}$$

$$h(n^*) = 0 \tag{12}$$

Therefore, we have that

$$f(n^*) \leq g(n^*) + \epsilon. \tag{13}$$

Now, consider any other goal node n'. Since n^* is a better goal node than n', we also have that

$$g(n^*) \leq g(n') \tag{14}$$

Putting (13) and (14) together, we get that

$$f(n^*) \leq g(n') + \epsilon \tag{15}$$

Based on this equation, we can extend A* as follows:

- Run A* until the first goal node n is found.
- Compute $c = g(n)$.
- Continue expanding nodes in A* order until all nodes of f-cost at most $c + \epsilon$ have been expanded.
- Output the lowest cost goal node found.

In fact, since (15) holds for any goal node found, we can do even better by reducing our bound on f if a cheaper goal node is found after the first goal node. I.e., we have c be the cost of the cheapest goal node found so far, and stop the algorithm as soon as all unexpanded nodes have f-value greater than $c + \epsilon$. Under this modification the algorithm is guaranteed to expand only nodes whose f-value is $\leq c^* + \epsilon$, where c^* is the (true) cost of the optimal goal node. Intuitively, this performance is the best we can expect given our guarantees on the heuristic function.

(b) **[5 points]** Consider the problem of search in the (familiar) blocks world domain. The domain consists of several square blocks of equal size on a (very large) table. Each block can be on the table or on top of exactly one other block. (Locations on the table are not labelled, so we don't care where on the table a block is.) A block can have at most one block on top of it. Operators in this space consist of taking one clear block (one which is not under any other block) and moving it to any other legal location (on top of a different clear block or to the table). Each application of an operator has cost 1.

Assuming your state space consists of complete blocks world configurations, and that the goal is a single fully specified state (e.g., a specific configuration of towers), define a nontrivial *admissible* heuristic function for this domain. Try to make your heuristic as powerful as possible while still making it admissible and relatively easy to compute. Explain why your heuristic is admissible.

One simple and nontrivial heuristic function is the one that counts the number of blocks in the current state that are not in the desired location according to the goal state. It's admissible because we need at least one move to get each block into the right place, and therefore this heuristic underestimate the required number of moves.

5. **SHORT ANSWERS [12 points]** Each of the following questions requires at most one sentence in response. Do not write more.

   (a) **[1 points]** True or false: α-β pruning, although typically more efficient than minimax, can occasionally result in a less desirable move. (No explanation is required.)

   False; *alpha*-β pruning only eliminates nodes that are clearly suboptimal, and therefore it cannot result in a less desirable move.

   (b) **[3 points]** Consider the action of toggling a light switch, whose effect is to turn the light on if it's currently off, and to turn the light off if it's on. Can you provide a pure STRIPS description (as in Section 14.5 of Ginsberg's book) of the *toggle* action, assuming that the predicates in our language are *On(x)* and *Off(x)*? Either show the description, or explain (in one sentence) why one is impossible.

   No. Standard STRIPS does not allow the conditional effects, i.e., where the effects depend on the preconditions.

   (c) **[3 points]** R2D1 is an office cleaning robot which vacuums the floor as it moves around. The following is a STRIPS description of its *move(x,y)* action (where *at(x)* is true if the robot is at location x):

   Preconditions: *at(x), adjacent(x,y).*

   Add list: *at(y), clean(x).*

   Delete list: *at(x).*[2]

   Is the following situation calculus axiom an equivalent description of the effects of this action (yes/no)?

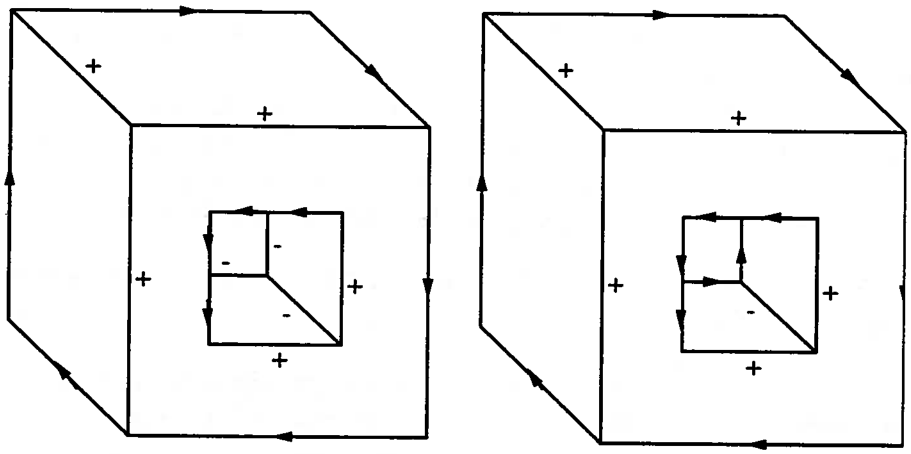
$$\forall s, x, y((at(x,s) \wedge adjacent(x,y)) \rightarrow \begin{array}{l}(at(y, result(move(x,y),s)) \wedge \\ clean(x, result(move(x,y),s)) \wedge \\ \neg at(x, result(move(x,y),s))))\end{array}$$

   If not, why not (one sentence)?

   No. In order to capture the STRIPS description completely, we must also specify frame axioms, i.e., the axiom must also state the properties of the world that *do not* change (e.g., the cleanliness of locations other than x).

   (d) **[3 points]** The following figure contains two consistent Waltz labellings for the same line drawing. The first (a) is essentially the one given by Ginsberg, only slightly simplified. Is the second labelling (b) illegal (yes/no only)? If not, what differences in the physical object do the differences in the labellings represent (one sentence)? (Note: the differences are in the labelling of the construction at the center of the object.)

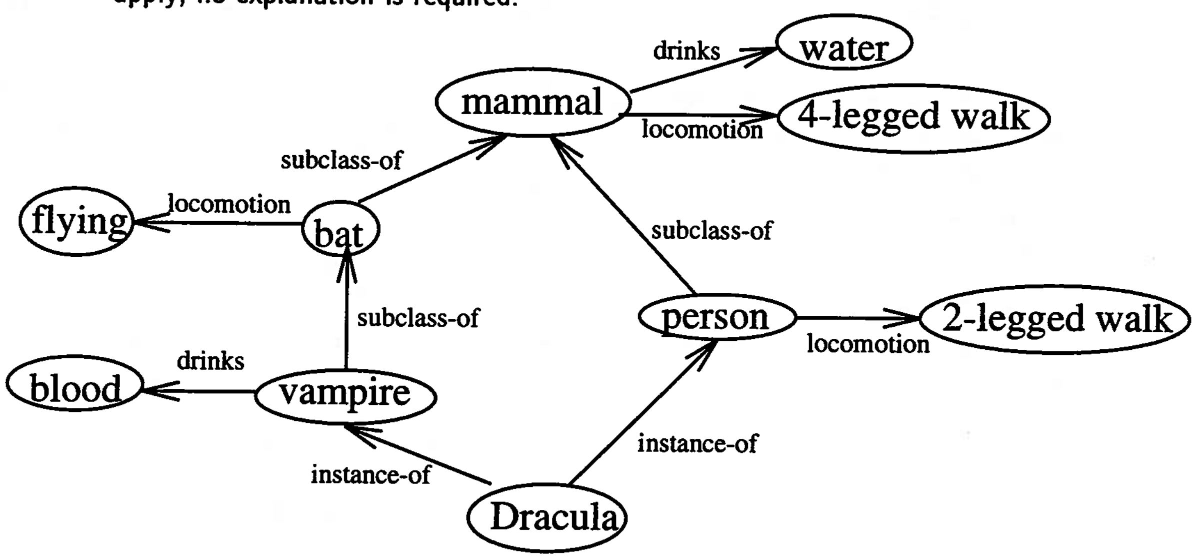
[2] In the notation of Russell & Norvig, the effects of this action are *at(y), clean(x), ¬at(x).*



(a) (b)

Both figures describe to legal physical objects. (a) corresponds to a cube with a hole, where the hole only goes partway through the cube. (b) corresponds to a cube where the hold goes all the way through to the back.

(e) **[2 points]** In the following nonmonotonic semantic network, which of the following conclusions can we make (using brave extensions, as described in Ginbserg's book)? Mark all that apply; no explanation is required.



(a) Dracula's locomotion is by walking on two legs.
(b) Dracula's locomotion is by walking on four legs.
(c) Dracula's locomotion is by flying.
(d) Dracula drinks blood.
(e) Dracula drinks water.

We can reach all of these conclusions except for (b). The reason (b) is blocked is because, along any path, there is a more specific default for locomotion that takes precedence: Along the "vampire" path, locomotion by flying takes precedence, and along the "person" path, locomotion by two-legged walking takes precedence. By contrast, the drinking water default is blocked along the "vampire" path, but can be reached along the "person" path.