Q.1 a. Define Artificial Intelligence. State the significance of using heuristic function. (4)

Answer:
Significance of using heuristic functions are:

- The path cost from the current state to goal state is calculated, to select the minimum path cost as the next state.
- Find the shortest solution using heuristic function that never over estimate the number of steps to the goal.

b. What is the difference between informed & uninformed search techniques? (4)

Answer:

Sl No.	Informed Search (Heuristic Search)	Uninformed Search (Blind Search)
1	The path cost from the current state to goal state is calculated, to select the minimum path cost as the next state	No information about the number of steps (or) path cost from the current state to goal state
2	More effective	Less effective in search method
3	Additional information can be added as assumption to solve the problem	Problem to be solved with the given information
4	E.g. a) Best first search b) Greedy search c) A* search	E.g. a)Breadth first search b)Uniform cost search c) Depth first search d) Depth limited search e) Interactive deepening search f) Bi-directional search

c. Explain constraint satisfaction problem using a suitable example. (4)

Answer:
Constraint satisfaction problem: Constraint satisfaction problems (CSPs) are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations. CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which are solved by constraint satisfaction methods.

2 marks

To represent the resolution in predicate logic by:

- Constant symbols: a, b, c, John, … to represent primitive objects
- Variable symbols: x, y, z, … to represent unknown objects
- Predicate symbols: safe, married, love, … to represent relations married(John) love(John, Mary)

2 marks

d. Define the terms Agent & Agent function. (4)

Answer:
An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors.

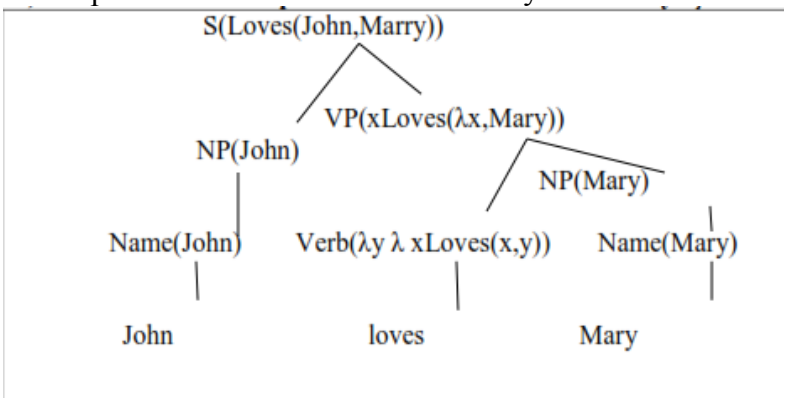
2 marks

Agent Function:
The agent function for an agent specifies the action taken by the agent in response to any percept sequence. It is an abstract mathematical description. Internally, the agent function for an artificial agent will be implemented by an agent program.

2 marks

e. Give the semantic representation of "John Loves Marry". (4)

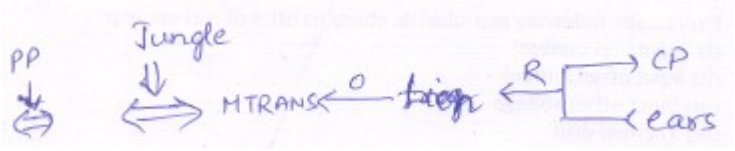
Answer:
Semantic representation of "John Loves Marry":



4 marks

f. Convert the following sentence into conceptual dependency: (4)
I heard a lion in the jungle

Answer:



g. Write short notes on Inductive & supervised learning. (4)

Answer:
Inductive Learning: Involves the process of learning by example – where a system tries to induce a general rule from a set of observed instances. The type of feedback available for learning is usually the most important factor in determining the nature of the learning problems that the agent faces. The field of machine learning distinguishes three cases: supervised, unsupervised and reinforcement learning. **Supervised Learning** Involves learning a function from examples of its inputs and

outputs. It requires that the training data set provides the correct answer for each instance. **Unsupervised Learning** Involves learning patterns in the input when no specific values are supplied. **Reinforcement Learning** Rather than being told what to do by a teacher (cf. supervised learning) a reinforcement learning agent must learn from the reinforcement/reward each action receives from the environment. **Semi-supervised Learning** We are given a few labeled examples and must make what we can of a large collection of unsupervised examples. Even the labels themselves may not be the oracular truths that we hope for. Thus both noise and lack of labels create a continuum between supervised and unsupervised learning. Inductive Machine Learning involves searching through the hypothesis space for a function that will perform well even on new examples beyond the training set. These functions can be defined using many different representation schemes: e.g., linear weighted polynomials, propositional and first-order logical sentences, probabilistic descriptions such as Bayesian networks.

4 marks

h. List any four applications of GA (4)

Answer:

Q.2 a. Explain steepest hill climbing algorithm. (6)

Answer:

In computer science, hill climbing is a mathematical optimization technique which belongs to the family of local search. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.

2 marks

For example, hill climbing can be applied to the travelling salesman problem. It is easy to find an initial solution that visits all the cities but will be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which two cities are visited. Eventually, a much shorter route is likely to be obtained.

2 marks

Hill climbing is good for finding a local optimum (a solution that cannot be improved by considering a neighbouring configuration) but it is not guaranteed to find the best possible solution (the global optimum) out of all possible solutions (the search space). The characteristic that only local optima are guaranteed can be cured by using restarts (repeated local search), or more complex schemes based on iterations, like iterated local search, on memory, like reactive search optimization and tabu search, on memory-less stochastic modifications, like simulated annealing.

2 marks

The relative simplicity of the algorithm makes it a popular first choice amongst optimizing algorithms. It is used widely in artificial intelligence, for reaching a goal state from a starting node. Choice of next node and starting node can be varied to give a list of related algorithms. Although more advanced algorithms such as simulated annealing or tabu search may give better results, in some situations hill climbing works just as well. Hill climbing can often produce a better result than other algorithms when the amount of time available to perform a search is limited, such as with real-time systems.

2 marks

It is an anytime algorithm: it can return a valid solution even if it's interrupted at any time before it ends.

In simple hill climbing, the first closer node is chosen, whereas in steepest ascent hill climbing all successors are compared and the closest to the solution is chosen. Both forms fail if there is no closer node, which may happen if there are local maxima in the search space which are not solutions. Steepest ascent hill climbing is similar to best-first search, which tries all possible extensions of the current path instead of only one.

1 **mark**

b. How searching is used to provide solutions also describe some real world problems? (6)

Answer:

Searching for solutions

We need to solve the formulated problems are done by a search techniques through the state space that use an **search tree** that is generated by the initial state and the successor function that together define the state space. The root of the search tree is a **search node** corresponding to the initial state. We continue choosing, testing, and expanding until either a solution is found or there are no more states to be expanded. The choice of which state to expand is determined by the **search strategy**.

2 marks

The general tree-search algorithm is described in figure as follows:

function TREE-SEARCH(problem, strategy)
returns a solution, or failure
initialize the search tree using the initial state of problem
loop do
if there are no candidates for expansion **then return** failure
choose a leaf node for expansion according to strategy
if the node contains a goal state **then return** the corresponding solution
else expand the node and add the resulting nodes to the search tree

2 marks

The nodes in the search tree are defined using **five components** in data structure. They are

1. **STATE**: the state in the state space to which the node corresponds;
2. **PARENT-NODE**: the node in the search tree that generated this node;
3. **ACTION**: the action that was applied to the parent to generate the node;
4. **PATH-COST**: the cost, traditionally denoted by g(n), of the path from the initial state to the node, as indicated by the parent pointers.
5. **DEPTH**: the number of steps along the path from the initial state.

The **difference between** nodes and states, a **node** is bookkeeping data structure used to represent the search tree. A **state** corresponds to a configuration of the world.

To represent the collection of nodes that have been generated but not yet expanded this collection is called **fringe**. Each element of the fringe is a **leaf node**, that is, a node with no successors in the tree. The representation of the fringe would be a set of nodes. The search strategy then would be a function that selects the next node to be expanded from this set. It could be computationally

expensive, because the strategy function might have to look at every element of the set to choose the best one. Alternatively, the collection of nodes is implemented as a **queue** representation.

The **queue operations** as follows:

MAKE-QUEUE (element…) – creates a queue with the given elements.
EMPTY? (queue) – returns true only if there are no more elements in the queue.
FIRST (queue) – returns the first element of the queue.
REMOVE-FIRST (queue) returns **FIRST** (queue) and removes it from the queue.
INSERT (element, queue) – inserts an element into the queue and the resulting queue.
INSERT-ALL (elements, queue) – inserts a set of elements into the queue and returns the
resulting queue.
The formal version of the general tree-search algorithm shown in figure.

function TREE-SEARCH(*problem*, *fringe*) **returns** a solution, or failure
fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]),*fringe*)
loop do
if EMPTY?(*fringe*) **then return** failure
node ← REMOVE-FIRST(*fringe*)
if GOAL-TEST[*problem*] applied to STATE[*node*] succeeds
then return SOLUTION(*node*)
fringe ← INSERT-ALL(EXPAND(*node*, *problem*),*fringe*)
function EXPAND(*node*, *problem*) **returns** a set of nodes
successors ← the empty set
for each <action, result> **in** SUCCESSOR-FN[*problem*](STATE[*node*]) **do**
s ← a new NODE
STATE[*s*] ← *result*
PARENT-NODE[*s*] ← *node*
ACTION[*s*] ← *action*
PATH-COST[*s*] ← PATH-COST [*node*] + STEP-COST (*node*, *action*, *s*)
DEPTH[*s*] ← DEPTH [*node*] + 1
add *s* to *successors*
return *successors*

Measuring problem-solving performance **4 marks**

The output of a problem-solving is either failure or a solution. We will evaluate an algorithm‘s performance in **four** ways:

Completeness: The strategy guaranteed to find a solution when there is one.

Optimality: If more than one way exists to derive the solution then the best one is selected.

Time complexity: Time taken to run a solution.

Space complexity: Memory needed to perform the search.

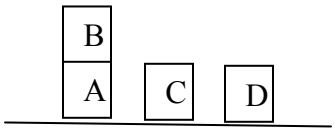
In AI, where the graph is represented implicitly by the initial state and successor function and is frequently infinite, **complexity** is expressed in terms of three quantities: **b**, the branching factor or maximum number of successors of any node; **d**, the depth of the shallowest goal node; and **m**, the maximum length of any path in the state space.

Definition of branching factor (b): The number of nodes which is connected to each of the node in the search tree. It is used to find space and time complexity of the search strategy.

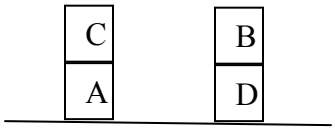
Some of the real world problems are:
Route finding
Touring & Travelling 1 **mark**
VLSI Layout etc..

c. Explain how STRIPS would solve the following problem: (6)

Start :



Goal :



Answer: Refer pages 255-56 of Text Book

Q.3 a. Describe Alpha-Beta pruning using a suitable example and give the other modifications to minmax procedure to improve its performance. (9)

Answer:
Alpha-Beta pruning Pruning: The process of eliminating a branch of the search tree from consideration without examining is called pruning.

The two parameters of pruning technique are:
1. **Alpha (a):** Best choice for the value of MAX along the path or lower bound on the value that on maximizing node may be ultimately assigned. 1 **mark**
2. **Beta (ß)**: Best choice for the value of MIN along the path or upper bound on the value that a minimizing node may be ultimately assigned. 1 **mark**

Alpha-Beta Pruning: The alpha and beta values are applied to a minimax tree, it returns the same move as minimax, but prunes away branches that cannot possibly influence the final decision is called **AlphaBeta pruning** or **Cutoff**. Consider the two ply game tree from figure.

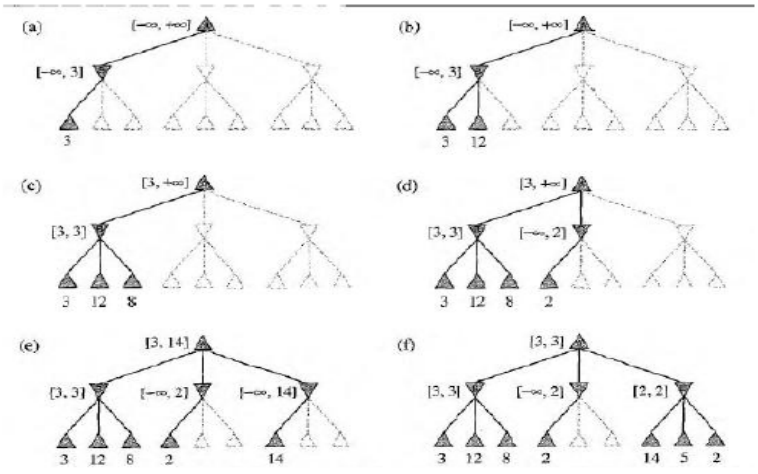
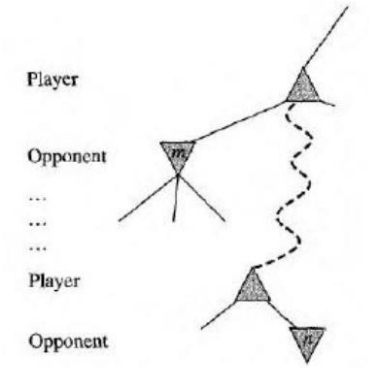


Figure Stages in the calculation of the optimal decision for the game tree in Figure 6.2. At each point, we show the range of possible values for each node. (a) The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of *at most* 3. (b) The second leaf below B has a value of 12; MIN would avoid this move, so the value of B is still at most 3. (c) The third leaf below B has a value of 8; we have seen all B's successors, so the value of *B* is exactly 3. Now, we can infer that the value of the root is *at least* 3, because MAX has a choice worth 3 at the root. (d) The first leaf below C has the value 2. Hence, *C*, which is a MIN node, has a value of *at most* 2. But we know that *B* is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successors of C. This is an example of alpha–beta pruning. (e) The first leaf below D has the value 14, so D is worth *at most* 14. This is still higher than MAX's best alternative(i.e., 3), so we need to keep exploring D's successors. Notice also that we now have bounds on all of the successors of the root, so the root's value is also at most 14. (f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX's decision at the root is to move to B, giving a value of 3.

2 marks

Alpha –beta pruning can be applied to trees of any depth, and it is often possible to prune entire sub trees rather than just leaves.



2 mark

Figure Alpha-beta pruning: the general case. If m is better than n for player, we will never get to n in play.

```
function ALPHA-BETA-SEARCH(state) returns an action
  inputs: state, current state in game

  v ← MAX-VALUE(state, −∞, +∞)
  return the action in SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(%α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

2 marks

Effectiveness of Alpha – Beta pruning

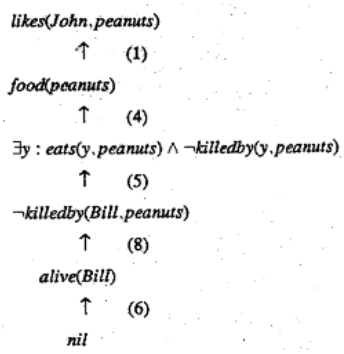
It needs to examine only nodes to pick the best move. 1 **mark**

b.

(a) To do this we need to make one assumption. The fourth sentence is ambiguous. Does it mean that if everyone eats it and isn't killed then it's food, or does it mean if at least one person eats it and isn't killed, it's food? We'll choose the latter. We need to decide whether to represent apples, chicken, and peanuts as constants, or as predicates so that we can talk about individual instances of those general types. For this problem, it is easier to represent them as constants so we'll do it that way, but this could make some other problem solving difficult, so you might want to try it the other way also. We will break sentence 5 into two separate formulas (5 and 6). Also, we will need to add (as axiom 8) the fact that if you're still alive, then you haven't been killed by anything.

1. $\forall x : food(x) \rightarrow likes(John, x)$
2. $food(apples)$
3. $food(chicken)$
4. $\forall x : (\exists y : eats(y, x) \wedge \neg killedby(y, x)) \rightarrow food(x)$
5. $eats(Bill, peanuts)$
6. $alive(Bill)$
7. $\forall x : eats(Bill, x) \rightarrow eats(Sue, x)$
8. $\forall x : \forall y : alive(x) \rightarrow \neg killedby(x, y)$

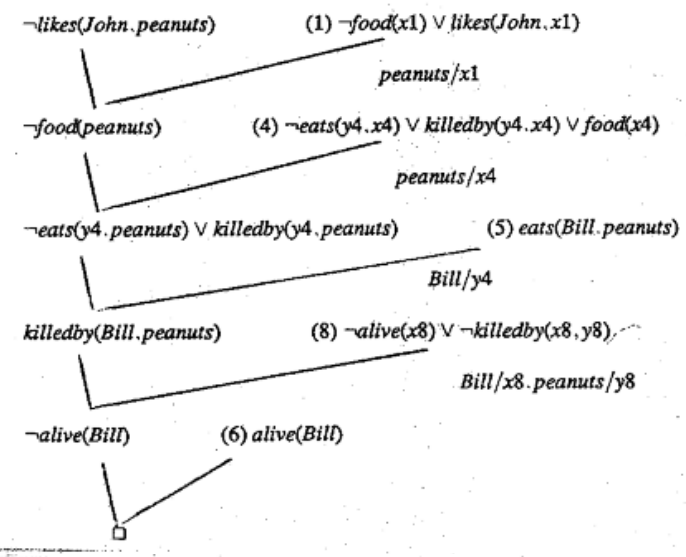
(b) Backward chaining proof:



(c) Clause form:

1. $\neg food(x1) \vee likes(John, x1)$
2. $food(apples)$
3. $food(chicken)$
4. $\neg eats(y4, x4) \vee killedby(y4, x4) \vee food(x4)$
5. $eats(Bill, peanuts)$
6. $alive(Bill)$
7. $\neg eats(Bill, x7) \vee eats(Sue, x7)$
8. $\neg alive(x8) \vee \neg killedby(x8, y8)$

(d) Resolution proof that John likes peanuts:



(e) We can answer the question of what Sue eats by the following resolution proof in which z is unified with $x7$ and then $x7$ is unified with peanuts:

3 x 3 = 9 marks

b. Consider the following sentences:

- **John likes all kinds of food.**
- **Apples are food.**
- **Chicken is food.**
- **Anything anyone eats and isn't killed by is food.**

- **Bill eats peanuts and is still alive.**
- **Sue eats everything Bill eats.**

(i) **Translate these sentences into formulas in predicate logic.**
(ii) **Convert the formulas of part(i) into clause form**
(iii) **Prove that John likes peanuts using backward chaining.** **(9)**

Answer:

Q.4 a. Describe Forward & Backward chaining rule system using suitable examples in both category . **(9)**

Answer:
Forward chaining :
A forward-chaining algorithm for propositional definite clauses was already given. The idea is simple:
Start with the atomic sentences in the knowledge base and apply ModusPonens in the
Forward direction, adding new atomic sentences, until no further inferences can be made.
First-order definite clauses
First-order definite clauses closely resemble propositional definite clauses they are disjunctions of literals of which *exactly one is positive. A* definite clause either is atomic or is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal.
2 marks
This knowledge base contains no function symbols and is therefore an instance of' the class DATALOG of **Data log** knowledge bases-that is, sets of first-order definite clauses with no function symbols.

There are three possible sources of complexity. First, the "inner loop" of the algorithm involves finding all possible unifiers such that the premise of a rule unifies with a suitable set of facts in the knowledge base. This is often called **pattern matching** and can be very expensive. Second, the algorithm rechecks every rule on every iteration to see whether its premises are satisfied,even if very few additions are made to the knowledge base on each iteration. Finally, the algorithm might generate many facts that are irrelevant to the goal.

Matching rules against Unknown facts *We can express every finite-domain CSP as a single definite clause together with some associated ground facts.*
Incremental forward chaining **2 marks**
Redundant rule matching can be avoided if we make the following observation: ***Every new fact inferred on iteration t must be derived from at least one new fact inferred on iteration t - 1.***
Backward chaining :

A backward chaining algorithm
Figure shows a simple backward-chaining algorithm,
Figure A simple backward-chaining algorithm.
Logic programming **2 marks**

Algorithm = Logic + Control.

Prolog is by far the most widely used logic programming language. Its users number in the hundreds of thousands. It is used primarily as a rapid-prototyping language and for symbol manipulation tasks such as writing compilers (Van Roy, 1990) and parsing natural language. Prolog programs are sets of definite clauses written in a notation somewhat different from standard first-order. Logic Prolog uses uppercase letters for variables and lowercase for constants. Clauses are written with the head preceding the body; —: -" is used for left implication, commas separate literals in the body, and a period marks the end of a sentence:
The execution of Prolog programs is done via depth-first backward chaining, where clauses are tried in the order in which they are written in the knowledge base. Some: aspects of Prolog fall outside standard logical inference: **2 marks**

Efficient implementation of logic programs
The execution of a Prolog program can happen in two modes: interpreted and compiled.
Interpretation essentially amounts to running the FOL-BC-ASK algorithm 1 **mark**

b. With an example, explain the logics for non- monotonic reasoning. (9)

Answer:
Non monotonic reasoning:
The definite clause logic is monotonic in the sense that anything that could be concluded before a clause is added can still be concluded after it is added; adding knowledge does not reduce the set of propositions that can be derived. **2 marks**

A logic is non-monotonic if some conclusions can be invalidated by adding more knowledge. The logic of definite clauses with negation as failure is non-monotonic. Non-monotonic reasoning is useful for representing defaults. A default is a rule that can be used unless it overridden by an exception.

For example, to say that *b* is normally true if *c* is true, a knowledge base designer can write a rule of the form
b ←c ∧~ab_a.
where *ab a* is an atom that means abnormal with respect to some aspect *a*. Given *c*, the agent can infer *b*unless it is told *aba*. Adding *aba* to the knowledge base can prevent the conclusion of *b*. Rules that imply *aba* can be used to prevent the default under the conditions of the body of the rule.
2 marks
Example: Suppose the purchasing agent is investigating purchasing holidays. A resort may be adjacent to a beach or away from a beach. This is not symmetric; if the resort was adjacent to a beach, the knowledge provider would specify this. Thus, it is reasonable to have the clause
away_from_beach ←~on_beach.

This clause enables an agent to infer that a resort is away from the beach if the agent is not told it is adjacent to a beach.

A cooperative system tries to not mislead. If we are told the resort is on the beach, we would expect that resort users would have access to the beach. If they have access to a beach, we would expect them to be able to swim at the beach. Thus, we would expect the following defaults:
beach_access ←*on_beach* ∧~*abbeach_access*
swim_at_beach ←*beach_access* ∧~*abswim_at_beach* **2 marks**

A cooperative system would tell us if a resort on the beach has no beach access or if there is no swimming. We could also specify that, if there is an enclosed bay and a big city, then there is no swimming, by default:
abswim_at_beach ←*enclosed_bay* ∧*big_city* ∧~*abno_swimming_near_city.*
We could say that British Columbia is abnormal with respect to swimming near cities:
abno_swimming_near_city ←*in_BC* ∧~*abBC_beaches.*
Given only the preceding rules, an agent infers *away_from_beach*. If it is then told *on_beach*, it can no longer infer *away_from_beach*, but it can now infer *beach_access* and *swim_at_beach*. If it is also told *enclosed_bay* and *big_city*, it can no longer infer *swim_at_beach*. However, if it is then told*in_BC*, it can then infer *swim_at_beach*.
By having defaults of what is normal, a user can interact with the system by telling it what is abnormal, which allows for economy in communication. The user does not have to state the obvious. **2 marks**

One way to think about non-monotonic reasoning is in terms of arguments. The rules can be used as components of arguments, in which the negated abnormality gives a way to undermine arguments. Note that, in the language presented, only positive arguments exist that can be undermined. In more general theories, there can be positive and negative arguments that attack each other.

1 mark

Q.5 a. Explain briefly how Bayesian statistics provides reasoning under various kinds of uncertainty. (6)

Answer:
A statistical learning method begins with the simplest task: parameter learning with complete data. A parameter learning task involves finding the numerical parameters for a probability model whose structure is fixed. **2 marks**

Maximum-likelihood parameter learning: Discrete models
In fact, though, we have laid out one standard method for maximum-likelihood parameter learning:
1. Write down an expression for the likelihood of the data as a function of the parameter(s).
2. Write down the derivative of the log likelihood with respect to each parameter.
3. Find the parameter values such that the derivatives are zero.
A significant problem with maximum-likelihood learning in general: ―*when the data set is small enough that some events have not yet been observed-for instance, no cherry candies-the maximum Likelihood hypothesis assigns zero probability to those events".* **2 marks**

The most important point is that, *with complete data, the maximum-likelihood parameter learning problem for a Bayesian network decomposes into separate learning problems, one for each parametez3.* The second point is that the parameter values for a variable, given its parents, are just

the observed frequencies of the variable values for each setting of the parent values. As before, we must be careful to avoid zeroes when the data set is small.

Naive Bayes models

Probably the most common Bayesian network model used in machine learning is the **naïve Bayes** model. In this model, the "class" variable C (which is to be predicted) is the root and the "attribute" variables *Xi* are the leaves. The model is "naive7' because it assumes that the attributes are conditionally independent of each other, given the class. **2 marks**

Maximum-like likelihood parameter learning: Continuous models

Continuous probability models such as the **linear-Gaussian** model. The principles for maximum likelihood learning are identical to those of the discrete case. Let us begin with a very simple case: learning the parameters of a Gaussian density function on a single variable. That is, the data are generated also follows:

The parameters of this model are the mean, Y and the standard deviation *a.* The quantity (yj - (B1xj + 02)) is the **error** for (zj, yj)-that is, the difference between the actual value yj and the predicted value (01 x j + $2)-SO E is the well-known **sum of squared errors.**

This is the quantity that is minimized by the standard **linear regression** procedure. Now we can understand why: minimizing the sum of squared errors gives the maximum likelihood straight-line model, *provided that the data are generated with Gaussian noise of* fi*xed variance.* **2 marks**

Bayesian parameter learning

The Bayesian approach to parameter learning places a hypothesis prior over the possible values of the parameters and updates this distribution as data arrive. This formulation of learning and prediction makes it clear that Bayesian learning requires no extra "principles of learning." Furthermore, *there is, in essence, just one learning algorithm,* i.e., the inference algorithm for Bayesian networks.

Learning net structures

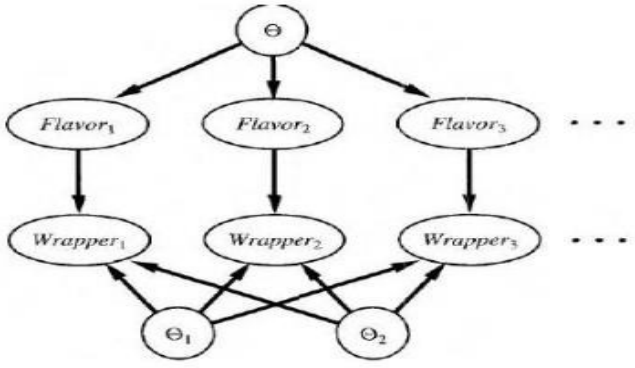


Figure: A Bayesian network that corresponds lo a Bayesian learning process. Posterior distributions for the parameter variables 0, el, and *e2* can be inferred from their prior distributions and the evidence in the *Flavor,* and *Wrapper* variables.

2 marks

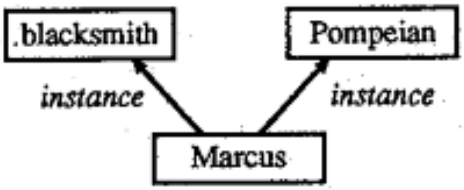
There are two alternative methods for deciding when a good structure has been found. The first is to test whether the conditional independence assertions implicit in the structure are actually satisfied in the data. **1 mark**

b. Construct semantic net representations for the following:

1. **Pompenian(Marcus), Blacksmith(Marcus)**
2. **Mary gave the green flowered vase to her favourite cousin (6)**

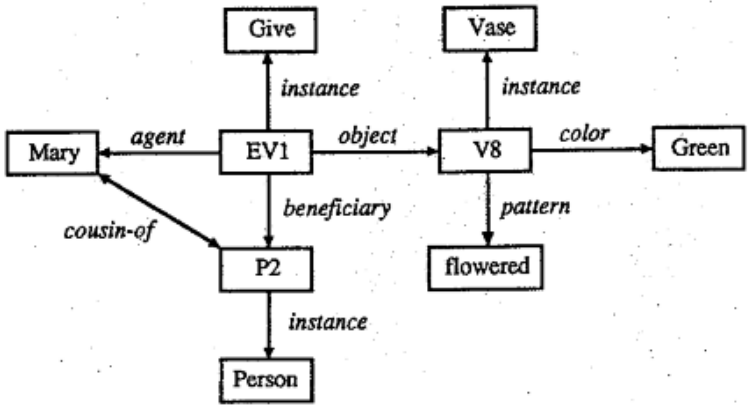
Answer:
Pompenian(Marcus), Blacksmith(Marcus)

(a) Marcus:



3 marks

2. Mary gave the green flowered vase to her favourite cousin



3 marks

c. What is Hopfield Network? Explain using a simple Hopfield net, how this network operates and achieves its desirable features.? (6)

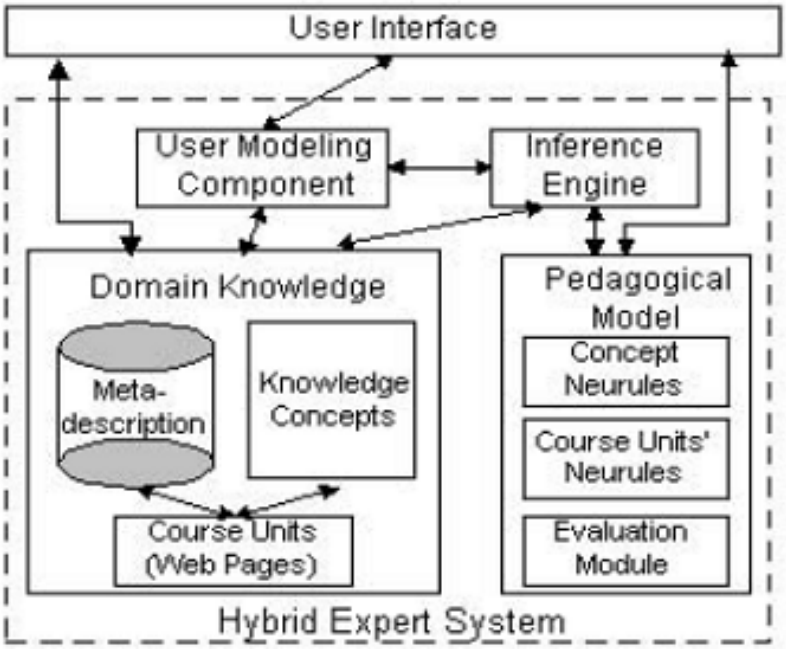
Answer: Refer pages 377-379 of Text Book.

Q.6 a. Define expert systems. Discuss the architecture of the expert system. (6)

Answer:

An expert system is a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice. **4 marks**

The architecture of the expert system is:



5 marks

b. Write a prolog program that finds the length of a list. Explain your program with help of a suitable example. (6)

Answer:

```
length([],0).
length([_|Tail], Len):-
        length(Tail, Len1), Len is Len1+1.
```

4M.

(Use any list to show working) 2M

c. Explain Neuro Fuzzy Systems . Give an example where this system can be used for effectively. (6)

Answer: Refer page 455 of Text Book

Q7. a. What are intelligent agents? List out some of the properties of agents. (9)

Answer:

b. Define NLP. Explain how an Augmented Transition Network works using parsing of following sentence : (9)

The long file has printed

Answer: Refer pages 295-297 of Text Book-I

TEXT BOOKS

I. Elaine Rich and Kevin Knight, "Artificial Intelligence", Tata McGraw-Hills, Reprint 2003
II. S Russell and Peter Norvig, Artificial Intelligence – A Modern Approach, Pearson Education, Reprint 2003
III. Saroj Kaushik, "Logic and Prolog Programming" , New Age International Ltd, publisher, 2007