B.E.
Sixth Semester Examination, December-2007
Intelligent Systems (CSE-304-E)

Note : Attempt any five questions.

Q.1. (a) To which generation of programming languages LISP and prolog belong ? Also discuss the causes behind emergence of this generation of PL. **10**

Ans. During the 1970s Prolog became popular in Europe for artificial intelligence applications. In the United States, however, Prolog remained a relatively miner computer language. In the United States researchers preferred the LISP language for artificial intelligence applications. LISP was considered a more powerful language for these applications, though it was more difficult to learn & use than Prolog.

During these early years both Prolog and LISP were very slow in execution and consumed large amounts of memory. In addition, users needed considerable programming expertise to use a Prolog or LISP program. Doctors, lowyers, engineers and others with the most critical needs for expert systems written in these languages found that the languages complexity limited the use of expert systems to environments such as universities, the federal government, and corparate research departments. On a mainframe computer or microcomputer, you will find that Prolog often is the best language for any application that involves formal or symbolic reasoning.

Almost all languages developed for the computer during the last few decades are known generically as procedural languages. FORTRAN, COBOL, C, BASIC, dBASEIII, and Pascal are all examples procedural languages. To use a procedural languages, an algorithm, or procedure, must first be defined to solve the problem at hand. A program is then written using the procedural language to implement the procedure.

Procedural languages distingush between a program and the data it uses. The procedure and control structure a program uses are defined by a programmer. The program manipulates the data according to the defined procedure.

Prolog uses only data about objects and their relationships. Prolog also emphasizes symbolic processing. LISP and Prolog are best known object oriented languages.

Q. 1. (b) Write a procedure in prolog to reverse a list. **5**

Ans. Procedure in Prolog to Reverse a List : The reverse_list predicate reverses the order of a list. It format is :

```
reverse_list (Inputlist, Outputlist) :
    reverse (Input list, [ | Outputlist).
    reverse ([ ], Inputlist, Inputlist).
    reverse ([Head : Tail ], List 1, List 2) :
    reverse (Tail, [Head : List 1], List 2).
```

Here is an example using the reverse_list predicate :

```
Goal : reverse_list ( [a, b, c, d], X )
X = ["d", "c", "b", "a"]
1 solution.
```

Q. 1. (c) What do you mean by flat list and association list in the contect of LISP. **5**

Ans. Association List : An association list or simply a list is a list of pairs.

Association lists are a traditional implementation of dictionaries and environments, which map a key to an associated value.

```
>(define bind(keys values env)
  (cons(list keys values) env))
```

BIND

```
> (bind 'a', '1' ())
  ((a  1))
> (define bind-all (keys values env)
  (append (map list keys values) env ))
```

BIND-ALL

```
> (bind-all '(a b c)' (1 2 3)' ())
  ((A  1) (B  2 ) (C  3 ))
> (assoc 'a' ((a  1) (b  2) (c  3)))
> (A  1)
> (assoc 'c' ((a  1) (b  2) (c  3)))
  (c  3)
```

FLAT LIST : We get a flattered form of a list if ignore all but the initial opening and final closing parenthesis in the written representation of list.

```
> (flat '((a) ((b  b) ((( c  c c )))))
  ( a  b  b c  c  c)
> flat '(1 (2  3) (( 4  5  6))))
  (1  2  3  4  5  6 )
```

Q.2. Explain with example :

(i) Depth First Search

(ii) Breadth First Search

Also give an example of a problem for which BFS would work better than DFS. Also give an example of a problem for which DFS would work better than BFS. **20**

Ans. (i) Depth First Search : Depth-first searches are performed by diving downward into a tree as quickly as possible. It does this by always generating a child node from the most recently expanded node, then generating that child's children and so on until a goal is found or some cut-off depth point d is reached. If a goal is not found when a leaf node is reached or at the cut off point, the program backtracks to the most recently expanded node and generates another of its children. This process continues until a goal is found or failure occurs.

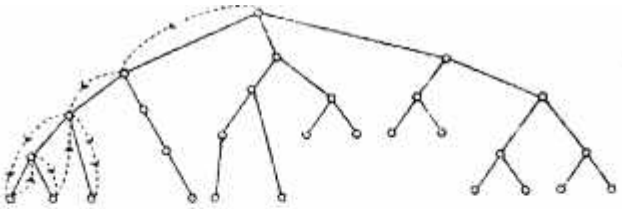
An Example of a Depth-first Search : An algorithm for the depth first search is the same as that for breadth-first except in the ordering of the nodes placed on the queue. Depth-first places the newly generated children at the head of the queue so that they will be chosen first.

Depth-First Search :

(i) Place the starting node and on the queue.

(ii) If the queue is empty, return failure and stop.

(iii) If the first element on the queue is a goal node g, return success and stop. Otherwise.



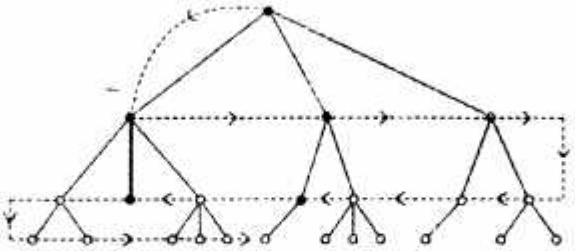
Depth-first search of a tree

(iv) Remove and expand the first element, and place the children at the front of the queue.

(v) Retrun to step (ii).

The depth-first search is preferred over the breadth-first when the search tree is known to have a plentiful number of goals. Otherwise, depth-first may never find a solution. The depth cut off also introduces some problems. If it is set too shallow, goals may be missed; if set too deep, extra computation may be performed.

(ii) Breadth First Search : Breadth-first searches are performed by exploring all nodes at a given depth before proceeding to the next level. This means that all immediate children of nodes are explored before any of the children's children are considered.



Breadth-first Search of a tree

It has obvious advantage of always finding a minimal path length solution when one exists. However, a great many nodes may need to be explored before a solution is found, especially if the tree is very full.

An algorithm for the breadth-first search is quite simple. It uses a queue structure to hold all generated but still unexplored nodes. The order in which nodes are placed on the queue for removal and exploration determines the type of search.

Breadth-first Search :

(i) Place the starting node S and on the queue.

(ii) If the queue is empty, return failure and stop.

(iii) If the first element on the queue is a goal node g, return success and stop, otherwise.

(iv) Remove and expand the first element from the queue and place all the children at the end of the queue in any order.

(v) Return to step (ii).

The time complexity of the breadth-first search is $O(b^d)$. This can be seen by noting that all nodes up to the goal depth d are generated. Therefore, the number generated is $b + b^2 + \ldots\ldots\ldots\ldots + b^d$ which is $O(b^d)$. The space complexity is also $O(b^d)$ since all nodes at a given depth must be stored in order to generate the nodes at the next depth that is, b^{d-1} node must be stored at depth $d - 1$ to generate nodes at depth d, which gives space complexity of $O(b^d)$. The use of both exponential time and space is one of the main drawbacks of the breadth-first search.

Q.3. What do you understand by forward reasoning and backward reasoning ? Explain with examples. A problem solving search can proceed either forward or backward. What factors determine the choice of direction for a particular problem ? **20**

Ans. The object of a search procedure is to discover a path through a problem space from an initial configuration to a goal state. While PROLOG only searches from a goal state, there are actually two directions in which such a search could proceed :

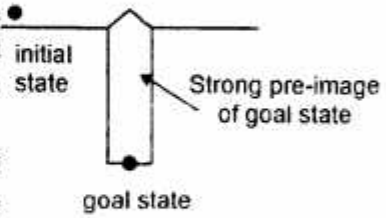
- Forward, from the start states
- Backward, from the goal states

Reason Forward from the Initial States : Begin building a tree of move sequences that might be solutions by starting with the initial configuration (s) at the root of the tree. Generate the next level of the tree by finding all the rules whose left sides match the roo node and using thier right sides to create the new configurations. Generate the next level by taking each node generated at the previous level and applying to it all of the rules whose left sides match it Continue until a configuration that matches the goal state is generated.

Reason Backward from the Goal States : Begin building a tree of move sequences that might be solutions by starting with the goal configurtion(s) at the root of the tree. Generate the next level of the tree by finding all the rules whose right sides match the root node. These are all the rules that, if only we could apply them, would generate the state we want. Use the left sides of the rules to generate the node at this second level of the tree. Generate the next level of the tree by taking each node at the prvious level and finding all the rules whose right sides match it. Then use the corresponding left sides to generate the new nodes. Continue until a node that matches the initial state is generated. This method of reasoning backward from the desired final state is often called goal-directed reasoning

Notice that the same rules can be used both to reason forward from the initial state and to reason backward from the goal state. To reason forward, the left sides are matched against the current state and the right sides are used to generate new nodes until the goal is reached. To reason backward the right sides are matched against the current node and the left sides are used to generate new nodes representing new goal states to be achieved.

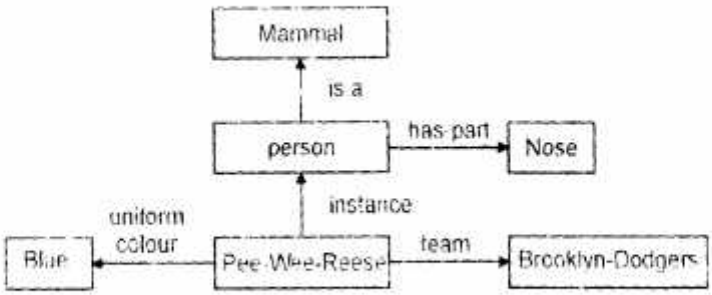
So we take an example of pick-and-place problem, how can we automatically generate a sequence of complaint motions ? One approach is to use the familiar problem-solving process of backward chaining. Our initial and goal states for peg-in-hole problem are represented as points in configuration space as shown in fig.



First, we compute the set of points in c-space from which we are guaranteed to reach the goal state in a single complaint motion, assuming a certain degree of uncertainty in initial position and direction of movement & certain facts about relative friction. This set of points is called the goal state's strong pre-image. Now we use backward chaining to design a set of motions that is guaranteed to get us from the initial state to some point in the goal state's strong pre-image. Recursively applying this procedure will eventually yield a set of motions that, while individually uncertain, combine to form a guaranteed plan.

Q. 4. (a) What are the semantic nets ? Why they came into existence ? Explain. **10**

Ans. In a semantic net, information is represented as a set of nodes connected to each other by a set of labelled arcs, which represent relationships among the nodes. A fragment of a typical semantic net is shown in fig.



A semantic Network

Semantic nets were used was to find relationship among objects by spreading activation out from each of two nodes and seeing where the activation met. This process is called intersection search.

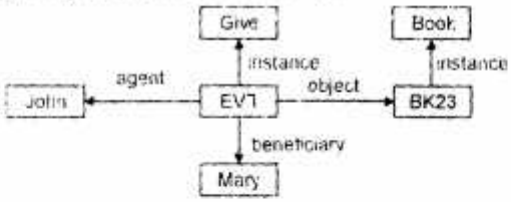
Semantic nets are a natural way to represent relationship that would appear as ground instances of binary predicates in predicate logic. For example, some of the ares from figure could be represented in logic as

is a (Person, Mammal)
instance (Pee-Wee-Reese, Person)
team (Pee Wee-Reese Brooklyn-Dodgers)
uniform-colour (Pee-Wee-Reese, Blue)

Knowledge expressed by predicates of other articles can also be expressed in semantic notes, for example.

man (Marcus)
could be rewritten as
instance (Marcus, Man)

Thereby making it easy to represent in a semantic net



A semantic Met Representing a Sentence

Three or more place predicates can also be converted to a binary form by creating one new object representing the entire predicate statement and then introducing binary predicates to describe the relationship to this new object of each of the original arguments. For example,

score (Cubs, Dodgers, 5-3)

This can be represented in a semantic net by creating a node to represent the specific game and then relating each of the there pieces of information it.

Q. 4. (b) What do you understad by rules based deduction system ? Explain. **10**

Ans. Deduction system is accomplished through a sequence of deductive inference steps using known facts. From the known facts, new facts or relationship are logically derived. For example, we could

learn deducting that Sue is the cousin of Bill, if we have knowledge of Sue and Bill's parents and rules for the cousin relationship. Deductive system usually requires more inference than the other methods.

The inference method used is, of course, a deductive type, which is a valid form of in ference.

Q. 5. Discuss the following :

(i) Baye's probabilistic interferences

(ii) Dempster shafer theory

Also discuss role of these theories in Artificial Intelligence. **20**

Ans. (i) Baye's Probabilistic Interferences : An important goal for many problem-solving systems is to collect evidence as the system goes along and to modify its behaviour on the basis of the evidence. To model this behaviour, we need a statistical theory of evidence. Bayesian statistics is such a theory. The fundamental notion of Bayesian statistics is that of conditional probability :

$$P(H / E)$$

Read this expression as the probability of hypothesis H given that we have observed evidence E. To compute this, we need to take into account the prior probability of H and the extent to which E provides evidence of H. To do this, we need to define a universe that contain an exhaustive, mutually exclusive set of H_i's, among which we are trying to discriminate. The, let :

$P(H_i / E)$ = The probability that hypothesis H_i is true given evidene E

$P(E / H_i)$ = The probability that we will observe evidence E given that hypothesis i is true.

$P(H_i)$ = The a priori probablity that hypothesis i is true in the absence of any specific evidence.

These probabilities are called prior probabilities or priors.

K = The number of possible hypotheses.

Baye's theorem then states that

$$P(H_i/E) = \frac{P(E/H_i) \cdot P(H_i)}{\sum_{n=1}^{k} P(E/H_n) \cdot P(H_n)}$$

Using Baye's theorem intractable for several reasons :

- The knowledge acquisition problem is insurmountables; too many probabilities have to be provided.
- The space that would be required to store all the probabilities is too large.
- The time required to compute the probabilities is too large.

(ii) Dempster Shafer Theory : Dempster-Shafer theory is an approach considers sets of propositions and assigns to each of them an interval

[Belief, Plausibility]

in which the degree of belief must lie. Belief (usually denoted by bell) measures the strength of the evidence in favour of a set of propositions. It ranges from 0 (indicating no evidence) to 1 (denoting certainly).

Plausibility (Pl) is defined to be

$$Pl(s) = 1 - Bel(\neg s)$$

It also ranges from 0 to 1 and measures the extent to which evidence in favour of ¬s leaves room for belief in s. In particular, if we have certain evidence in favour of ¬s then Bel (¬s) will be 1 and Pl(¬s) will be 0. This tells us that the only possible value for Bel(s) is also 0.

Q.6. (a) What do you understand by planning ? Discuss the components of planning. **12**

Ans. The word 'Planning' refers to the process of computing several steps of a problem-solving procedure before executing any of them. When we describe computer problem-solving behaviour, the distinction between planning and doing fades a bit since rarely can be computer actually do much of anything besides plan. In solving the 8 puzzle, for example, it cannot actuates push any tiles around. So when discussed the computer solution of the 8-puzzle problem, what we were really doing was outlining the way the computer might generate a plan for solving it.

Components of a Planning System :

(a) Choosing Rules to Apply : The most widely used technique for selecting appropriate rules to apply is first to isolate a set of differences between the desired goal state and the element state and then to identify those rules that are relevant to reducing those differences. If several rules are found, a variety of other heuristic information can be explotted to choose among them. This technique is based on the means-ends analysis method.

(b) Applying Rules : Each rule simply specified the problem state that would result from its application. However, we must be able to deal with rules that specify only a small part of the complete problem state. There are many ways of doing this.

Detecting a Solution : A planning system has succeeded in finding a solution to a problem when it has found a sequence of operators that transforms the initial problem state into the goal state. How will it know when this has been done ? In simple problem-solving systems, this question is easily answered by a straightforward match of the state descriptions.

One representational technique has served as the basis for many of the planning systems that have been built. It is predicate logic, which is appealing because of the deductive mechanisms that it provides.

Detecting Dead Ends : As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution. The same reasoning mechanisms that can be used to detect a solution can often for detecting a dead end.

If the search process is reasoning forward from the initial state, it can prove any path that leads to a state from which the goal state cannot be reached. For example, suppose we have a fixed supply of paint : some white, some pink, and some red.

Repairing an Almost Correct Solution : The kinds of techniques we are discussing are often useful in solving nearly decomposable problems. One good way of solving such problems is to assume that they are completely decomposable, proceed to solve the subproblems separately, and then check that when the subsolutions are combind, they do in fact yield a solution to the original problem.

Q. 6. (b) Write short note on genetic algorithms. **8**

Ans. While neural network models are based on a computational "brain metaphor" a number of other learning techniques make use of a metaphor based on evolution. In this work, learning occurs through a selection process that begins with a large population of random programs. Learning algorithms inspired by evolution are called "genetic algorithms".

Q. 7. (a) Define expert systems and explain their uses in various fields. **8**

Ans. Expert System : Expert systems solve problems that are normally solved by human "experts". To solve expert level problem, experts systems need access to a substantial domain knowledge base, which must be built as efficiently as possible.

Expert systems are complex AI programs. Almost all the techniques that we described in Parts I and II have been exploited in at least one expert system. However, the most widely used way of representing domain knowledge in expert system is as a set of production rules, which are often coupled with a frame system that defines the objects that occur in the rules.

All the rules we show are English versions of the actual rules that the system use.

R1 (Sometimes also called XCON) is a program that configures DEC VAX systems. Its rules look like this :

If : the most assent active context is deistributing mass bus devices, and
there is a single-part disk drive that has not been assigned to a massbus, and
there are no unassigned dual-part disk drives, and the number of devices that each massbus should support is known, and
there is a massbus that has been assigned at least one disk drive and that should support additional disk drives.
and the type of cable needed to connect the dist drive to the previous device on the massbus is known then : assign the disk drive to the mass bus.

Notice the R1's rules, unlike MYCIN's, contain no numeric measures of certainty.

PROSPECTOR is a program that provides advice on mineral exploration. Its rules look like this :

If : magnetite or pyrite in disseminated or veinlet form is present.

then : (2, –4) there is favourable mineralization and texture for the propylitic stage.

DESIGN ADVISOR is a system that critiques chip designs. Its rules look like :

If : the sequential level count of ELEMENT is greater than 2, UNLESS the signal of ELEMENT is resetable.

then : critique for poor resetability.

DEFEAT : poor resetability of ELEMENT.

due to : sequential level count of ELEMENT greater than 2.

by : ELEMENT is directly reusable.

Q. 7. (b) Discuss the steps involved in Natural Language processing. **12**

Ans. Steps Involved in Natural Language Processing :

- **Morphological Analysis :** Individual words are analysed into their component, and non word tokens, such as punctuation are separated from the words.
- **Syntactic Analysis :** Linear sequences of words are transformed into structures that show how the words relate to each other. Some word sequences may be rejected if they violate the language's rules for how words may be combined. For example, an English syntactic analyzer would reject the sentence "Boy the go the to store."
- **Semantic Analysis :** The structures created by the syntactic analyzer are assigned meanings. In other words, a mapping is made between the syntactic structures and objects in the task domain. Structures for which no such mapping is possible may be rejected. For example, in most universes, the sentence "colourless green ideas sleep furiously" would be rejected as semantically anomolous.
- **Discourse Integration :** The meaning of an individual sentence may depend on the sentences that precede, it and may influence the meanings of the sentences that follow it.

- Pragmatic Analysis : The structures representing what was said is reinterpreted to determins what was actually meant. For example, the sentence "Do you know what time it is" ? Should be interpreted as a request to be told the time.

Q.8. Discuss following : **20**

(a) Neural Network **(b) Explanation based Learning**

Ans. (a) Neural Network : The very first effort in machine learning tried to mimic animal learning at a neural level. These effort were quite different from the symbolic manipulation methods. Collections of idealized neurons mere presented with stimulit and prodded into changing their behaviour via forms of reward and punishment.

Hopfield introduced a neural network that he proposed as a theory of memory. A Hopfield network has the following features :

- **Distributed Representation :** A memory is stored as a pattern of activation across a set of processing elements. Furthermore, memories can be superimposed on one another; different memories are represented by different patterns over the same set of processing elements.
- **Distributed, Asynchronous Control :** Each processing element makes decision based only on its own local situation. All these local actions add up to a global solution.
- **Content-Addressable Memory :** A number of patterns can be stored in a network. To retrieve a pattern, we need only specify a portion of it. The network automatically finds the closest match.
- **Fault Tolerance :** If a few processing elements misbehave or fail completely, the network will still function properly.

The **perception**, an invention of Rosenblatt, was one of the earliest neural network models. A perception models a neuron by taking a weighted sum of its inputs and sending the output 1 if the sum is greater than some adjustable threshold value (otherwise it sends 0).

(b) Explanation Based Learning : Learning complex concepts using these procedures typically requires a substantial number of training instances. But people seen to be able to learn quite a bit from single examples.

A number of independent studies led to the characterization of this approach as explanation based learning. An EBL system attempts to learn from a single example x by explaining why x is an example of the target concept. This explanation is then generalized, & the system's performance is improved through the availability of this knowledge.

Mitchell et al And DeJony and Money both describe general frame works for EBL programs and give general learning algorithms. We can think of EBL programs as accepting the following as input :

- A Training Example – What the learning program "sees" in the world.
- A Goal Concept – A high level description of what the program is supposed to learn.
- An Operationality Criterion – A description of which concepts are usuable.
- A Domain Theory – A set of rules that describe relationships between objects and actions in a domain.

From this, EBL computes a generalization of the training example that is sufficient to describe the goal concept and also satisfies the operationality criterion.