B.TECH.

TIT-702

SEVENTH SEMESTER EXAMINATION 2009-10

ARTIFICIAL INTELLIGENCE

Time : 3 Hours *Total Marks : 100*

Note: Attempt all questions:

Q.1. Attempt any four parts: (5×4=20)

(a) What do you understand by Artificial Intelligence? Why AI is important?

Ans. It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is relatd to the similar task of using computers to understand human intelligence,but AI does not have to confine itself to methods that are biologically observable.

Artificial intelligence is a branch of science which deals with helping machines find solutions to complex problems in a more human-like fashion. This generally involves borrowing characteristics from human intelligence, and applying them as algorithms in a computer friendly way. A more or less flexible or efficient approach can be taken depending on the requirements established, which influences how artificial the intelligent behaviour appears.

AI is generally associated with Computer science, but it has many important links with other fields such as Maths, Psychology, Cognition, Biology and Philosophy, among many others. Our ability to combine knowledge from all these fields will ultimately benefit our progress in the quest of creating an intelligent artificial being.

Importance of AI: AI techniques allow the greatest possible degree of automation in managing your documentation or moving it to the Web.

AI also makes it possible to automate processes that would otherwise have to be done manually.

(b) Discuss the problem of "Water Jug problem" with Heuristic search techniques.

Ans. The Water Jug Problem: There are two jugs called four and three; four holds a maximum of four gallons and three a maximum of three gallons. How can we get 2 gallons in the jug four.

The state space is a set of ordered pairs giving the number of gallons in the pair of jugs at any time i.e. (four, three) where four = 0, 1, 2, 3, 4 and three = 0, 1, 2, 3.

The start state is (0, 0) and the goal state is (2, *n*) where *n* is a don't case but is limited to three holding from 0 to 3 gallons.

The major production rules for solving this problem are shown below:

initial condition goal comment

1. (four, three) if four < 4 (4, three) fill four from tap.
2. (four, three) if three < 3 (four, 3) fill three from tap.
3. (four, three) if four > 0 (0, three) empty four into drain.
4. (four, three) if three > 0 (four, 0) empty three into drain.
5. (four, three) if four + three < 4 (four + three, 0) empty three into four
6. (four, three) if four + three < 3 (0, four + three) empty four into three.
7. (0, three) if three > 0 (three, 0) empty three into four.

8. (four, 0) if four > 0 (0, four) empty four into three.

9. (0, 2) (2, 0) empty three into four

10. (2, 0) (0, 2) empty four into three.

11. (four, three) if four < 4 (4, three-diff) pour diff, 4-four, into fo from three.

12. (three, four) if three < 3 (four-diff, 3) pour diff, 3-three, into three from four

and a solution is given below.

four three rule

0 0

0 3 2

3 0 7

3 3 2

4 2 1 1

0 2 3

2 0 1 0

The control strategy for selecing the rules has not yet been fully discussed but if all the rules are applied to each of the states those that are applicable will produce new states and the required goal state can be searched for.

We need to be able to create a formal description of the problem and eventually it will be possible for programs to take an informal description of a problem and produce a formal description of the same problem. For simple problems it is easier to achieve this goal by hand but there will be cases where this is far too difficult.

Summarising, we must perform the following tasks to produce a formal specification of a particular problem.

Formal Description of a Problem:

1. Define a state space that contains all possible configurations of the relevant objects, without enumerating all the state in it;

2. Define some of these states as possible initial states;

3. Specify one or more as acceptable solutions, these are goal states;

4. Specify a set of rules as the possible actions allowed. This involves thinking about the generality of the rules, the assumptions made in the informal presentation and how much work can be anticipated by inclusion in the rules.

Heuristic Search: To use heuristics to find a solution in acceptable time rather than a complete solution in infinite time.

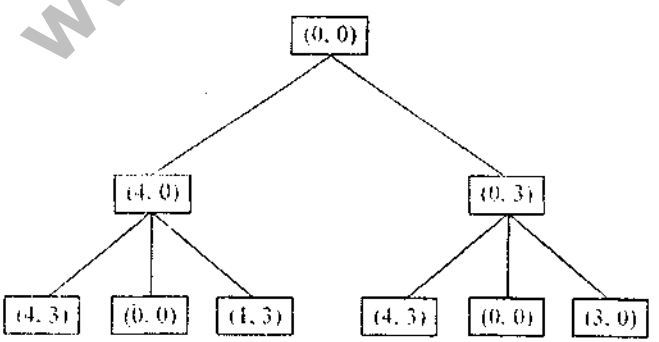


Fig. *A Search Tree for the Water Jug Problem.*

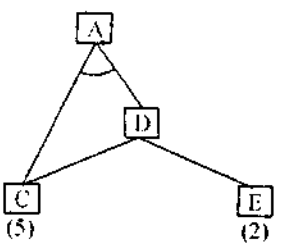


Fig. *Interacting Subgoals.*

(c) Differentiate between Depth-first search and Breadth first search.

Ans.

S.No.	*Breadth First Search*	*Depth First Search*
1.	No blind ally exist.	Blind ally exist.
2.	If solution exist it will be found.	Even if solution exist, it explores one branch only and it may declare failure. Chance of success becomes still less when loop exists.
3.	May find many solutions. If many solutions exist, minimal solution can be found.	It stops after one solution is found. Minimal solution may not be found as it explores only one branch.
4.	Require more memory because all the off-springs of the tree must be explored on level n before a solution on level $(n + 1)$ is to be examined.	Require less memory because only one branch is scanned. It stops after solution is found.

(d) Explain AO* algorithm with example.

Ans. The AO* Algorithm: The problem reduction algorithm we just described is a simplification of an algorithm described in Martelli and Montanari [1973], Martelli and Montanari [1978], and Nilsson [1980]. Nilsson calls it the AO* algorithm, the name we assume.

Rather than the two lists, OPEN and CLOSED, that were used in the A* algorithm, the AO* algorithm will use a single structure GRAPH, representing the part of the search graph that has been explicitly generated so far. Each node in the graph will point both down to its immediate successors and up to its immediate predessors. Each node in the graph will also have associated with it an h' value, an estimate of the cost of a path from itself to a set of solution nodes. We will not store g (the cost of getting from the start node to the current node) as we did in the A* algorithm. It is not possible to compute a single such value since there may be many paths to the same state. And such a value is not necessary because of the top-down traversing of the best-known path, which guarantees that only nodes that are on the best path will ever be considered for expansion. So h' will serve as the estimate of goodness of node.

AO* Algorithm:

1. Initialise the graph to start node.

2. Traverse the graph following the current path accumulating nodes that have not yet been expanded or solved.

3. Pick any of these nodes and expand it and if it has no successors call this value FUTILITY otherwise calculate only f' for each of the successors.

4. If f' is 0 then mark the node as *SOLVED*.

5. Change the value of f' for the newly created node to reflect its successors by back propagation.

6. Wherever possible use the most promising routes and if a node is marked as SOLVED then mark the parent node as SOLVED.

7. If starting node is SOLVED or value greater than FUTILITY, stop, else repeat from 2.

(c) When will Hill climbing search technique fail? Do steepest ascent hill climbing always find solutions? How might some problem be overcome in search?

Ans. Even with these first-aid measures, hill climbing is not always very effective. It is particularly unsuited to problems where the value of the heuristic function drops off suddenly as you move away from a solution. This is often the case whenever any sort of threshold efect is present. Hill climbing is a local method, by which we mean that it decides what to do next by looking only at the "immediate" consequences of its choice rather than by exhaustively exploring all the consequences.

(f) Define and describe the difference between knowledge, belief, hypothesis and data.

Ans. Knowledge can be defined as the body of facts and principles accumulated by humankind or the act, fact or state of knowing.

Belief: It is defined as essentially any meaningful and coherent expression that can be represented. Thus a belief can be true or false.

Hypothesis: It is defined as justified belief that is not known to be true. Thus, a hypothesis is a belief that is backed up with some supporting evidence, but it may still be false. In other words, it is preliminary assumption or putative explanation that accounts for a set of facts, taken to be true for the purpose of investigating and testing.

Data: Data in computer terminology mean raw facts and figure. For example "Ram", "145721", 'B' are data. Data are proceed to form information.

Q.2. Attempt any four parts: **(5×4=20)**

(a) Explain Top Down parsing techniques with examples.

Ans. Types of Parsers: There are two types of parsers:

1. Top-down parsing: In this parsing method, we guess or predict what should be the next production to be applied. In case our guess is wrong, we stack an alternatives so that we can come back to them if necessary. The parser starts with the symbols S and attempts to rewrite it into a sequence of terminal symbols. These terminal symbols should match the classes of the words in the input sentence until it entirely consists of terminal symbols. These are then checked with the input sentence to see if they match. If not, the process is started all over again with a different set of rules. Thus is repeated until a specific rule is found which describes the structure of the sentence.

An improvement to this algorithm is to use a depth-first search with backtracking. In this case, when the first terminal symbol in the grammar is reached, we immediately check whether the first word of the sentence belongs to the same category as the terminal symbol. If yes, the process is continued for the rest of the sentence. If not, the process is backtrack and an alternative rule is tried.

The main problem with top-down parsers is that it takes exponential time on bad sentences for the CFGs. With well-behaved grammars, it can be linear time algorithm. Note that NL grammars are usually not very well-behaved.

Example: The following example shows the top-down parsing.

S → NP VP

NP · › ART N | NAME

PP → PREP NP

VP → V | V NP | V np pp | V PP

(b) Explain different type of Chomsky's hierarchy grammars.

Ans. Chomsky Hierarchy of Generative Grammars: Chomsky hierarchy of generative grammars clearify the view of grammar.

Chomsky delineated four types of grammars, numbered 0 to 3.

Type 0: Most general: Generally, with no restrictions on the form that rewrite rules can take. Both sides of the rewrite rules can have any number of terminal and non-terminal symbols. This type of grammar is obtained by making the simple restriction that y cannot be the empty string in the rewrite form $xyz \rightarrow xwz$. Languages generated by type 0 grammars are exactly those recognized by turing machines.

Type 1: Context-sensitive: This type of grammars have the added restriction that the length of the string on the right-hand side of the rewrite rule must be at least as long as the string on the left-hand side. In other words, the right-hand side of the production must contain at least as many symbols as the left-hand side. Furthermore, in productions of the form $xyz \rightarrow xwz$, y must be a single non-terminal symbol and w must be a non-empty string. Typically rewrite rules for type 1 grammar take the forms:

$S \rightarrow aS$
$S \rightarrow aAB$
$AB \rightarrow BA$
$aA \rightarrow ab$
$aA \rightarrow aa$

where the capitalized letters are non-terminals and the lower case letters are terminals.

Type 2: Context-free: The left-hand side of a production must contain exactly one symbol, a non-terminal. This type of grammar is characterized by rules with the general form. <symbol> → <symboll> ... <symbol k>, where $k \geq 1$ and where the left-hand side is a single non-terminal symbol, $A \rightarrow xyz$ where A is a single non-terminal. Productions of this type take forms such as

$S \rightarrow aS$
$S \rightarrow aSb$
$S \rightarrow aB$
$S \rightarrow aAB$
$A \rightarrow a$
$B \rightarrow b$

Type 3: Regular Expression: All productions are of the form $A \rightarrow aB$ or $A \rightarrow a$. Exactly these languages recognized by find state automata.

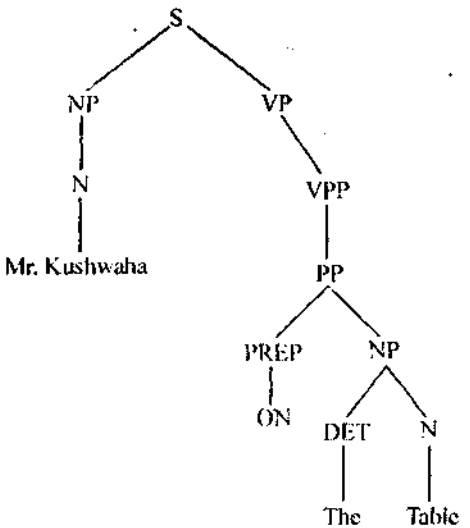
(c) Differentiate between deterministic and nondeterministic parsers.

Ans. The deterministic parser has only one production rule of a single non-terminal symbol whereas non-deterministic parser have multiple production rules for a non-terminal symbol. Both are used to find the behaviour of different languages. It is normally designed in either top down parser or buttom up parser techniques.

(d) Develop a parse tree for the sentence "Mr. Kushwaha on the table" using the following rules:

S → NP VP
NP → N
NP → DET N
VP → VPP
PP → PREP NP
N → Mr. Kushwaha/table
V → Slept
DET → the
PREP → ON

Ans. A parsing table for a given sentence is

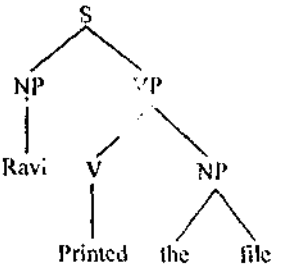


(e) Explain Fillmore's case grammar with example.

Ans. Fillnoer's Case Grammar: Fillmore grammar also called as case grammar. Case grammar provides a different approach to the problem of how syntactic and semantic interpretation can be combined. Grammar rules are written to describe syntactic rather than semantic regularities. But the structures, rules are produced correspond to semantic relations rather than to strictly syntactic ones. As an exmaple, consider the two sentences and to the simplified forms of their conventional parse trees shown in figure.

Case grammar describes the relationship between verbs and their arguments parsing using a case grammar is usually expectation-driven. Once verb of the sentence is located, it can be used to predict the noun phrases that will occur and to determine the relationship of those phrases to the rest of the sentences.

The figure shows the parsing of active voice sentence.



A case grammar describes the correct set of deep cases. Some of the cases are given as follows:

Agent — Investigator of the action.

Instrument — Cause of the event.

Dative — Entity affected by the action.

Factitive — Object.

Locative — Place of the event.

Source — Place from which something moves.

Fillmore characterized the relationships between the verb and noun phrase as "semantically relevant syntactic relationships" and called them cases. The cases assignment comes from the deep structure, even though the surface structure is different.

The structure that is built by the parse contains some semantic information, although further interpretation may be necessary. Gramamr rules are written to describe syntactic rather than semantic regularities. But the structures rules produce correspond to semantic relations rather than to strictly syntactic ones. For example:

$$S \rightarrow NPV$$

Parsing using case grammars is usuall, expectation-driven. Once the verb of the sentence has been located, it can be used to predict the noun phrases that will occur and to determine the relationship of those phrases to the rest of the sentence.

(f) Write short notes on sentence generation.

Ans. Sentence Generation: Language generation is opposite of language

understanding process. For sentence generation, a system must not only decide what to say but also how to say it. A sentence generation system must decide which form is better, which words and structures best express what we want to communicate and when to say what.

The study of language generation falls naturally into three areas:

1. Content Determination: It is concerned with what details to include in an explanation, a request, a question or argument in order to convey the meanings set forth by the goals of the speaker. This means that the speaker must know what the hearer already knows, what the hearer needs to know and what the hearer wants to know. These topics are related to the domain, task and discourse context.

2. Text Planning: It is the process of organizing the content to be communicated so as to best achieve the goals of the speaker.

3. Realization: It is the process of mapping the organized content to actual text. This requires that specific words and phrases be chosen and formulated into a syntactic structure.

The aim of a typical text generation system is to produce a text which satisfies some set of pre-stated goals. Such systems are provided with a knowledge base which contains information to be expressed and a set of goals.

The typical stages in a natural language generation system are:

(a) Content determination: Determination of the salient features that are worth being said. Methods used in this stage are related to data mining.

(b) Discourse planning: Overall organization of the information to be conveyed.

(c) Sentence aggregation: Merging of similar sentences to improve readability and naturalness. For example, the sentences "the next train is Dehradoon Express" and "the next train leaves Delhi at 10 am" can be aggregated to form "the next train, which leaves at 10 am, is the Dehradoon Express".

(d) Lexicalization: Putting words to the concepts.

(e) Referring expression generation: Linking words in the sentences by introduction pronouns and other types of means of reference.

(f) Syntactic and morphological realization: The state is the inverse of parsing given all the information collected above, syntactic and morphological rules are applied to produce the surface string.

(g) Orthographic realization: Matters like casing, punctuation and formatting are resolved.

Q.3. Attempt any two parts: **(10×2=20)**

(a) (i) Explain semantic Nets with example.

(ii) Draw a hierarchical network to represent the information.

Mouse is a rodent; rodent is a mammal; A mammal has color and also drinks water.

Ans. (a) Semantic Nets: An important feature of human memory is the high number of connections or associations between the different pieces of information contained in it. Semantic networks are one of the knowledge representation languages based on this intuition.

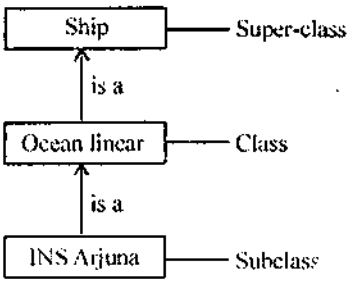
The basic idea of a semantic network representation is very simple. There are two types of primitive, nodes and links or arcs. Links are unidirectional connections between nodes. Nodes correspond to objects, or classes of objects, in the world, whereas links correspond to relationships between these objects.

Semantic nets, sementic network or associated network, is used to describe a knowledge representation system based on network structure. Originally they were developed for use as psychological models of human memory but now they are being used as standard methods for knowledge representation system in Artificial Intelligence and Expert Systems.

The nodes in a semantic net stand for 'OBJECTS, CONCEPTS OR EVENTS'. Arcs can be defined in a variety of ways, depending on the kind of knowledge being represented. In natural language processing arcs are represented by AGENTS, OBJECTS, RECIPIENTS etc.

As simple example consider, "NIS Arjuna is a liner" and "Every ocean liner is a ship".

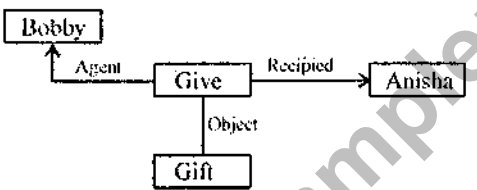
These statements are represented in semantic nets in the figure.



Semantic nets have also been used. in natural language research this with the help of an example.

Example: Draw the semantic network of a Bobby gives Anisha a gift.

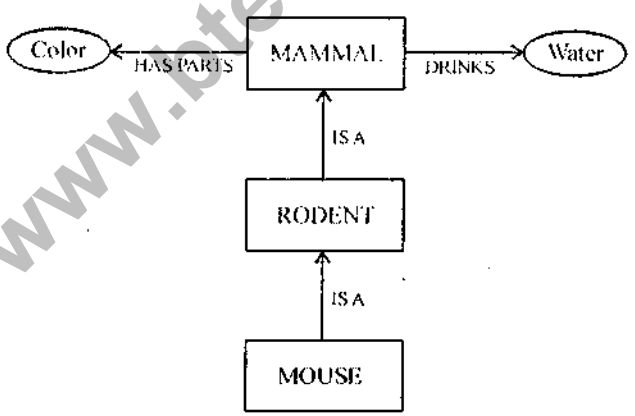
Ans.



Note that the arcs define the relationship between the predicate GIVE and the concepts, BOBBY, ARISHA GIFT associated with that predicate.

The semantic representation is useful because it provides standard way of analyzing the meaning of a sentence.

(ii)



(b) Explain Inference Rules in brief with example. Represent such expression in FOPL.

(i) All employees earning $500 or more per year pay taxes.

(ii) Some employees are sick today.

Ans. Inference rules should be sound, and even better complete. We saw from last lecture that we want our inference rules to always produce entailed sentences. This means that all inference rules should always be sound.

If an inference system can produce all entailed sentences then we say it is complete.

Inference rules can be used manaully or automated.

Doing manual inference you are free to use a large number of different inference rules.

Automated reasoning can be done by the automated computation of Truth Tables or by using inference rules.

However, the automated use of inference rules has to be done differently to how humans use inference rules.

Automated reasoning and other forms of logic:

Modules Ponens (MP)

A → B

A

therefore B

Conjunction Introduction (^I)

A

B

therefore A ^ B

Conjunction Elimination (^I)

A ^ B A ^ B

therefore A and therefore B.

(c) Explain Horn clause. What is the procedure of clausal conversion with example?

Ans. Horn Clauses: In logic, a Horn clause is a clause (a disjunction of literals) with at most one positive literal. A Horn clause with exactly one positive literal is a definite clause; a Horn clause with no positive literals is sometimes called a goal clause, especially in logic programming. A Horn formula is a conjunctive normal form formula whose clauses are all Horn; in other words, it is a conjunction of Horn clauses. A dual-Horn clause is a clause with at most one negative literal.

The following is an example of a (definite) Horn clause:

$\neg a \vee \neg b \vee ... \vee \neg c \vee d \rightarrow$ at most 1 + ve literal.

Such a formula can be rewritten in the following form, which is more common in logic programming and similar fields.

$$(a \wedge b \wedge ... \wedge c) \rightarrow d$$

The relevant of Horn clauses to theorem proved by first order resolution is that the resolution of two Horn clauses is a Horn clause. Moreover, the resolution of a goal clause and a definite clause is again a goal clause. In automated theorem proving, this can lead to greater efficiencies in proving a theorem (represented as a goal clause).

Horn clauses are also of interest in computational complexity, where the problem of finding a set of variable assignments to make a conjunction of Horn clauses true is a *P*-complete problem, sometimes called HORNSAT. This is *P*'s version of the Boolean satisfiability problem, a central NP-complete problem.

Q.4. Attempt any two parts: (10×2=20)

(a) Differentiate between expert system and problem solving system. Why is it important that an expert system be able to explain the why and how question related to a problem solving session?

Ans. Comparison to problem-solving systems: The principal distinction between expert system and traditional problem solving programs is the way in which the problem related expertise is coded. In traditional applications, problem expertise is encoded in both program and data structures. In the expert system approach all of the problem related expertise is encoded in data structures only; no problem-specific information is encoded in the program structure. This organization has several benefits.

An example may help contrast the traditional problem solving program with the expert system approach. The example is the problem of tax advice. In the traditional approach data structures

describe the taxpayer and tax tables, and a program in which there are statements representing an expert tax consultant's knowledge, such as statements which relate information about the taxpayer to tax table choices. It is this representation of the tax expert's knowledge that is difficult for the tax expert to understand or modify.

In the expert system approach, the information about taxpayers and tax computations is again found in data structures, but now the knowledge describing the relationships between them is encoded in data structures as well. The programs of an expert system are independent of the problem domain (taxes) and serve to process the data structures without regard to the nature of the problem area they describe. For example, there are programs to acquire the described data values through user interaction, programs to represent and process special organizations of description, and programs to process the declarations that represent semantic relationships within the problem domain and an algorithm to control the processing sequence and focus.

The general architecture of an expert system involves two principal components: a problem dependent set of data declarations called the knowledge base or rule base, and a problem independent (although highly data structure dependent) program which is called the inference engine.

Another major distinction between expert systems and traditional systems is illustrated by the following answer given by the system when the user answers a question with another question, "Why", as occurred in the above example. The answer is:

A. I am trying to determine the type of restaurant to suggest. So far Indian is not a likely choice. It is possible that French is a likely choice. I know that if the diner is a wine drinker, and the preferred wine is French, then there is strong evidence that the restaurant choice should include French.

It is very difficult to implement a general explanation system (answering questions like "Why" and "How") in a traditional computer program. An expert system can generate an explanation by retracing the steps of its reasoning. The response of the expert system to the question WHY is an exposure of the underlying knowledge structure. It is a rule; a set of antecedent conditions which, if true, allow the assertion of a consequent. The rule references values, and tests them against various constraints or asserts constraints onto them. This, in fact, is a significant part of the knowledge structure. There are values, which may be associated with some organizing entity. For example, the individual diner is an entity with various attributes (values) including whether they drink wine and the kind of wine. There are also rules, which associate the currently known values of some attributes with assertions that can be made about other attributes. It is the orderly processing of these rules that dictates the dialog itself.

(b) Write short notes on:
(i) MYCIN
(ii) Compare the different type of problem solved by MYCIN and DENDRAL.

Ans. MYCIN: MYCIN is an expert system for diagnosing and recommending treatment of bacterial infections of the blood (such as meningitis and bacteremia). It is intended to support clinicians in the early diagnosis and treatment of meningitis, which can be fatal if not treated in time. However, the laboratory tests for these conditions take several days to complete, so doctors have to make decisions with incomplete information associated with medical knowledge. MYCIN incorporated a calculus of uncertainty called certainty factors which seemed to fit well with how doctors assessed the impact of evidence of the diagnosis. This system was able to perform as well as some experts and considerably better than junior doctors.

MYCIN represented its knowledge as a set of IF-THEN rules with certainty factors. MYCIN is primarily a goal directed system, using the basic backward chaining reasonsing strategy. However, MYCIN used various heuristics to

control the search for a solution.

One strategy is to first ask the user a numebr of more or less preset questions that are always required. This allows the system to rule out totally unlikely diagnoses. Once these questions have been asked the system can then focus on particular, more specific possible blood disorders and go into full backward chaining more to try and prove each one. This rules out a lot of unnecessary search and also follows the pattern of human patient-doctor interviews.

In the first stage, initial data about the case is gathered so the system can come up with a very broad diagnosis. In the second, more directed questions are asked to test specific hypothesis. At the end of this section a diagnosis is proposed. In the third stage questions are asked to determine an appropriate treatment, given the diagnosis and facts about the patient. At any stage the user can ask why a question was asked or how a conclusion was reached. When treatment is recommended the user can ask for alternative treatments if the first is not viewed as satisfactory.

Method: MYCIN's expertise knowledge lies in the domain of bacterial infections. MYCIN's pool of knowledge consists of approximately 500 antecedent-consequent rules, implemented in LISP which give MYCIN the ability to recognize about 100 causes of bacterial infections. Physicians usually begin antibiotic treatment for patients who have bacterial infections without knowing exactly which organism is the culprit. For the desperately sick, therapy must begin at once, not 2 days after. This requirement leaves two choices: the physician can either prescribe a broad-spectrum drug which covers all possibilities, or he can prescribe a better, disease specific drug.

MYCIN helps the physician to prescribe disease-specific drugs. MYCIN informs itself about particular cases by requesting information from the physician about a patient's symptoms, general condition, history and laboratory-test results. Thus, the questions start as though taken from a checklist, but the questions then vary as evidence builds. At the end, it provides:

1. a list of possible culprit bacteria ranked from high to low based on probability of each diagnosis.
2. it's confidence in each dignosis probability.
3. the reason behind each diagnosis.

In this way MYCIN not only lists the questions and rules which led it to rank a diagnosis in a particular way but also gives its recommended course of drug treatment.

It has the following organizational features:

(a) Knowledge Representation: Is in the form of production rules implemented in LISP.

(b) Reasoning: Mechanism is backward chaining, goal-driven reasoning and uses certainty factors to reason with uncertain information.

(c) Heuristics: When the general category of infection has been established, MYCIN examines each candidate diagnosis in a depth-first manner.

(d) Dialogue-Explanation: The dialogue is computer controlled, with MYCIN driving the consultation through asking questions. Explanations are generated by tracing back through the rules which have been fired. Both "how?" and "why?" explanations are supported.

Difference between DENDRAL and MYCIN:

Organizational Feature	*DENDRAL*	*MYCIN*
Knowledge representation	Production rules and algorithms for generating graph structures.	Production rules implemented in LISP.
Reasoning	Forward chaining (data driven)	Backward chaining (Goal driven)
Heuristics	Generate and test	Depth-first search
Dialogue/Explanation	Mixed control	Computer control

DENDRAL was one of the earliest expert systems. It was designed to analyse mass spectra. Based on the mass of fragments seen in the spectra, it would be possible to make inferences as to the nature of the molecule tested, identifying functional groups or even the entire molecule. In the lab this can be a very difficult process. As a note, DENDRAL did contain rules, but it worked differently from most expert systems.

MYCIN was an expert system developed over five or six years in the early 1970s at the Stanford University. It is derived from the earlier Dendral expert system, but considerably modified and extended the basic Dendral software. This computer system was designed to diagnose infectious blood diseases and recommend antibiotics, with the dosage adjusted for patient's body weight. It's name is derived from the antibiotics themselves, as many antibiotics have the suffix "—mycin".

(c) Explain Meta knowledge. Under what conditions would it make sense to use both forward and backward chaining? Give an exmaple where both are used.

Ans. Meta knowledge: Meta knowledge or meta-knowledge is knowledge about knowledge, i.e., knowledge about what we know. More precisely speaking, meta-knowledge is systematic problem and domain-independent knowledge which performs or enables operations on another more or less specific domain-dependent knowledge is different domains/areas of human activities. Meta-knowledge can be considered as a fundamental conceptual instrument in such research and scientific domains as, knowledge engineering, knowledge management and other dealings with study and operations on knowledge, seen as an unified object/entities, abstracted from local conceptualizations and terminologies.

In AI, computer science and cognitive science, there are different interpretations what meta-knowledge means. According to one of the theories, there are three levels of meta knowledge which can be considered like a fried egg in a frying-pan. The yolk (yellow portion of egg) is knowledge that "we know". Then white of the egg is knowing "what we do not know". The frying-pan is like the rest of the universe. This is the knowledge that "we do not know that we do not know".

The procedures, methodologies and strategies of teaching, coordination of e-learning courses are individual meta-meta-knowledge of an intelligent entity (a person, organization or society). In such perspective, every universal meta-knowledge frameworks have to be also valid for the organization of meta-levels of any individual meta-knowledge.

Q.5. Attempt any two questions: (10×2=20)

(a) What is LISP? Why it is popular among AT practitioner? Write a LISP program which returns the minimum of three numbers.

Ans. LISP Programming:

1. The name LISP originally stood for "LISt processor".

2. Like most traditional programming languages LISP is procedural.

(*a*) LISP programs describe HOW to perform an algorithm.

(*b*) This contrasts with declarative languages like PROLOG, whose programs are DECLARATIVE assertions about a problem domain.

3. LISP is a FUNCTIONAL language. Its syntax and semantics are derived from the mathematical theory of recursive functions.

4. Everything in LISP is a function.

A function is a procedure that takes in one or more input arguments and returns only one value, with no side effect.

$$g(a, b) = a + b$$

LISP is NOT a 'pure' functional language.

Some of its functions produce side effects, e.g., SETF.

(b) (i) What do you understand by pattern recognition?

(ii) Differentiate between structured description and symbolic description.

Ans. Pattern Recognition: Recognition is the process of establishing a close match between some new stimulus and previously stored stimulus patterns. This process is begin performed continually throughout the lives of all living things.

Through visual sensing and recognition, we identify many special objects, such as home, office, school, restaurants, faces of people, handwriting and printed words. Through aural sensing and recognition, we identify familiar voices, songs and pieces of music and bird and other animal sounds. Through touch, we identify physical objects such as pens, cups, automobile controls and food items.

At more abstract levels of recognition, we recognize or identify such things as ideas concepts, procedures, plans, old arguments, metaphors and so on.

Our pervasive use and dependence on our ability to recognize patterns has motivated much research toward the discovery of mechanical or artificial methods comparable to those used by intelligent beings. Systems have now been developed to reliably perform character and speech recognition; fingerprint and photograph identification; electroencephelogram (EEG), electrocardiagram (ECG), oil log-well and other graphical pattern analysis.

Object classification is closely related to recognition. The ability to classify or group objects according to some commonly shared features is a form of class recognition. Like recognition, classification depends on the ability to discover common patterns among objects. This ability must be acquired through some learning process. Prominent feature patterns which characterize classes of objects must be discovered, generalized and stored for subsequent recall and comparison.

Pattern recognition is a field within the area of machine learning. Alternatively, it can be defined as "the act of taking in raw data and taking an action based on the category of the data".

Pattern recognition aims to classify data (patterns) based on either a prior knowledge or on statistical information extracted from the patterns. The patterns to be classified are usually groups of measurements or observations, defining points in an appropriate multidimensional space.

The classification or description scheme is usually based on the availability of a set of patterns that have already been clasisified or described. This set of patterns is termed as the training set and the resulting learning strategy is characterised as supervised learning. Learning can also be unsupervised, in the sense that the system is not given a prior labelling of patterns.

The classification or description scheme usually uses one of the following approaches:

1. Statistical (or decision theory),
2. Syntactic (or structural)

Statistical pattern recognition is based on statistical characterisations of patterns, assuming that the patterns are generated by a probabilistic system. Structural pattern recognition is based on the structural interrelationships of features. A wide range of algorithms can be applied for pattern recognition from very simple Bayesian classifiers to much more powerful neural networks.

Typical applications are automatic speech recognition. Classification of text into several categories, the automatic recognition of handwritten postal codes on postal envelopes, or the automatic recognition of images of human faces. The last two examples form the subtopic "image analysis" of pattern recognition that deals with digital images as input to pattern recognition systems.

Pattern recognition is more complex when templets are used to generate variants. Pattern recognition is studied in many fields, including psychology, ethology and computer science.

***(ii)* Structured Description (Syntactic Pattern Recognition):** Syntactic pattern recognition or structural pattern recognition is a form of pattern recognition, where items are presented in pattern structures which can take into account more complex interrelationship between features. In statistical classification simple numerical feature vectors are used.

Syntactic pattern recognition can be used (instead of statistical pattern recognition) if there is clear structure in the patterns. In this case differences in the structures of the classes are encoded as different grammars.

An example of this would be diagnosis of the heart diseases with ECG measurements. ECG waveforms can be approximated with diagonal and vertical line segments.

Another way to represent relations are graphs, where nodes are connected if corresponding subpatterns are related. An item can be labelled as belonging to a class if its graph representation is isomorphic with prototype graphs of the class.

Structural methods provide description of items, which may be useful in its own light. For example, syntactic pattern recognition can be used to find out what objects are represent in an image. The syntactic recognition approach is based on the uniqueness of syntactic "structure" among the object clases.

Let grammar $G(v_n, v_t, p, s)$ define the objects shown in figure, where the terminal v_t consist of the following shape primitives.

Using syntactic analysis, that is pasing and analyzing the string structures, classification is accomplished by assigning an object to class C_i when the string describing it has been generated by the grammar G_i. This requires that the string be recognized as a member of the language $L(G_i)$. If there are only two classes, it is sufficient to have a single grammar G (two grammars are needed when strings of neither class can occur).

Symbolic Description: It is important to distinguish between the mechanism for building the meaning of the fake word/phrase and the mechanism that uses that meaning to build a question with the word/phrase as an answer:

For example, the following questions use the same meaning for the fake word 'trunquallizer', but refer to that meaning in different ways:

1. What do you use to sedate an elephant?
2. What kind of medicine do you give to a stressed-out elephant?

On the other hand, these questions are all put together in the same way, but from different fake meaning of the fake word:

1. What do you use to sedate an elephant?
2. What do you use to medicate a nose?

We will adopt the term schema for the first sort of symbolic description and template for the second sort.

Lexicon (dictionary): There is a (finite) set of lexemes. A lexeme is an abstract entity, roughly corresponding to a meaning of a word or phrase. Each lexeme has exactly one tentry in the lexicon, so if a word has two meanings, it will have two corresponding lexemes. Each lexeme may have some properties which are true of it and there are a number of possible relations which may hold between exemes. Each lexeme is also associated with a near-surface from which roughly indicates the written form of the word or phrase.

For example, one could define a relationship "superordinate-synonym" such that X is a superordinate-synonym of Y if there is a lexeme Z which is a superordinate of Y and Z, then it is a synonym of X.

We shall use the term "lexical relation" to denote any relation between lexemes.

Schemata: The word schema means a large set of facts with variables. A schema or symbolic description stipulates a set of relationships which must hold between the lexemes used to build, for example, a joke. More specifically, a schema determines how real words/phrases are glued together to make a fake word/phrase and which parts of the lexical entries for real words/phrases are used to construct the meaning of the fake word/phrase. Strictly speaking, the relationships are between

sequences of lexemes, but in most cases the sequence will be of length one.

(c) Explain PROLOG. Write a PROLOG program that creates knowledge base of family relationships such as father, mother, brother, sister, parents. Use clauses such as male, female to define rules.

Ans. PROLOG: PROLOG (PROgramming is LOGic) was invented by Alian Colmerauer and his associates at the University of Marseilles during the early 1970s. PROLOG uses the syntax of predicate logic to perform symbolic and logical computations.

PROLOG has been successful as an AI programming language because of following reasons:

1. Most AI programs reason using logic. The syntax and semantics of PROLOG are very close to formal logic.

2. PROLOG language has in-build inference engine and automatic backtracking facility. Hence, various search strategies can be efficiently implemented.

3. PROLOG has high productivity and ease of program maintenance.

4. PROLOG language is based on the universal formalism of HORN clauses. We already know that in HORN clause one condition is followed by zero or more conditions.

5. PROLOG language can be implemnted with ease on parallel machines because of inherent AND parallelism.

6. Understanding of the language is easy because the PROLOG clauses have a procedural and declarative meaning.

7. We are in a position to test different modules as each clause in PROLOG can be executed separately as if it is a separate program.

Features of PROLOG: Various features of PROLOG are:

1. PROLOG is a declarative language. Facts and relationships about the system under investigation are encoded by making this information as part of the knowledge base of the system. The user can then query the system for information either explicitly stated in the knowledge base or which is implied form this information.

2. PROLOG uses the language of predicate calculus. In predicate calculus, objects are represented by terms which may have one of the following forms:

(a) A constant symbol representing a concept or single individual. A constant symbol is equivalent to a PROLOG atom and examples include greek, alberta and peace.

(b) A variable symbol which may represent different individuals at different times. Variables must be introduced along with quantifiers.

(c) A compound term which resembles the symbolic-expression in LISP. A compound term consists of a function symbol in conjunction with an ordered set of terms as its arguments. The function symbol represents how the function depends on the arguments.

likes (ram, pat).

After building this data base, a query of the form:

likes (ram, pat)?

will elicit the response

* * yes

while the query

likes (ram, arisha)?

will generate a

* * no

Since this fact has not yet been entered into the data base. The PROLOG "no" should be interpreted as "now known".

3. Prolog handles lists and recursion naturally. A very useful, recursively defined predicate is member (*A, L*)? Which answer(s) the question, "Is a a member of list L"? If we have a list L defined as :

(Similar to that in LISP, elements are separated by commas and enclosed in square list.

L = [a, b, c, d, ...]

then we may write

L = [Head (Tail)]

where

Head = a (the first element of the list)

Tail = [b, c, d, ...] (the remainder of the list)

A is defined to be a member of list L if it satisfies either of the following two rules:

(a) A is a member of list L is A is the first element of L.

(b) If A is not the first element of L, then A is a member of L if and only if A is a member of the tail of the list L.

These two rules may be encoded recursively in PROLOG by the statements;

member (A, [A (_1).

member (A, [_|Tail]) : member (A, Tail).

We may not that the "–" symbol signifies "this variables matches anything". Once this recursive function has been defined, we may ask:

Member (cat, [the, cat, sat, on, the hat])?

and PROLOG answers

* * yes

but if we ask :

member (dog, [the, cat, sat, rat, bet, hat])?

PROLOG says

** no

4. PROLOG provides for very efficient coding for problems requiring inference. That is, to solve the logical inference problem:

All humans are mortal.

Socrates is a human.

Is Socrates mortal?

can easily be encoded in PROLOG as

mortal (X):- human (X)

human (socrates).

mora (socrates)?

to which PROLOG responds:

**yes

Disadvantages of PROLOG: PROLOG has some serious disadvantages compared to LISP. These include:

1. In general, LISP has better I/O features than PROLOG.

2. In general, PROLOG does not support graphics. An exception is the recently released TURBO PROLOG

3. The order inw hich rules are entered greatly affects the efficiency of PROLOG. The order4 of LISP function has minimal effect on LISP efficiency.