Q.2 a. List and discuss two potentially negative effects on society of the development of Artificial Intelligence Technique.

Answer:

Q.2 b. Write down four applications of Artificial Intelligence.

Answer: Refer Pages 9-13 of Text Book-I

Q.2 c. The philosopher, Searle uses the experiment of Chinese room to demonstrate that the machine does not understand. Explain the experiment and Chinese room.

Answer: Refer Page 8 of Text Book-I

Q.3a. Convert the following sentences into classical form:

(i) Whoever can read is literate.
(ii) Dolphins are not literate.
(iii) Some Dolphins are intelligent.
Prove that: Some who are intelligent cannot read. (8)

Answer: Refer Page 42 of Reference-I

b. What is resolution? Explain SLD resolution technique used in PROLOG. Use suitable example. (8)

Answer: Refer Page 43 of Reference-I

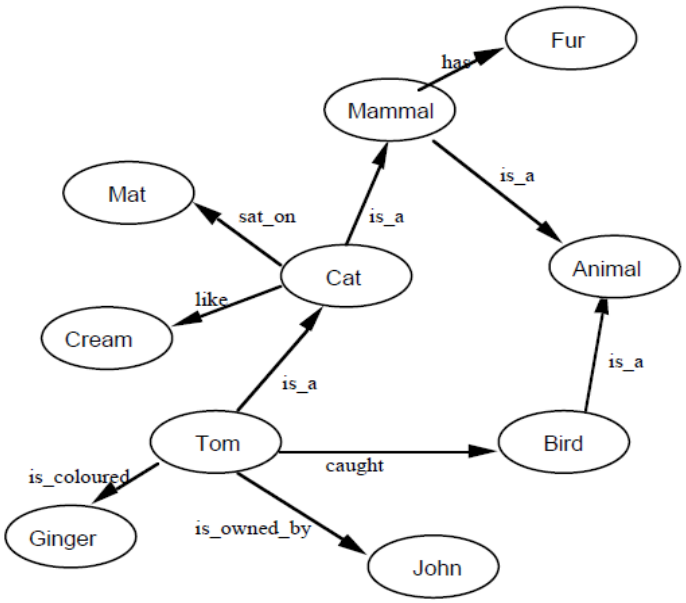
Q.4 a. Write down stages of knowledge acquisition. (8)

b. Explain principles of semantic networks. Make semantic network of following statements:

Tom is a ginger coloured cat owned by John. Tom caught a bird.

Answer:

Semantic Nets Semantic networks are an alternative to predicate logic as a form of knowledge representation. The idea is that we can store our knowledge in the form of a graph, with nodes representing objects in the world, and arcs representing relationships between those objects. For
example, the following



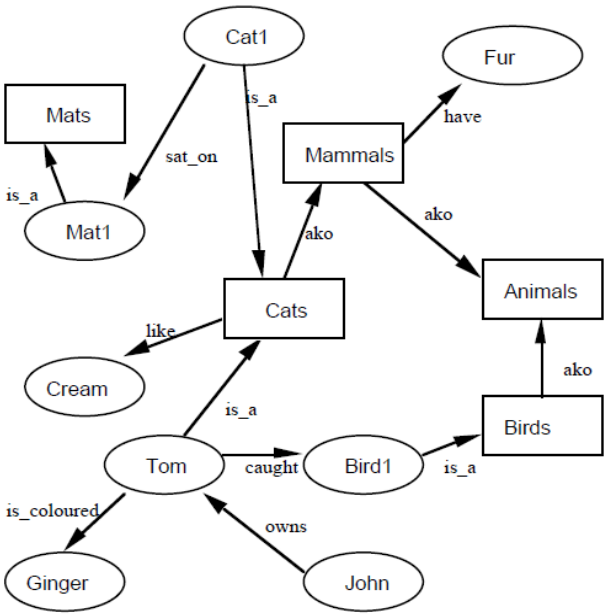
is intended to represent the data:
Tom is a cat.
Tom caught a bird.
Tom is owned by John.
Tom is ginger in colour.
Cats like cream.
The cat sat on the mat.
A cat is a mammal.
A bird is an animal.
All mammals are animals.
Mammals have fur.

It is argued that this form of representation is closer to the way humans structure knowledge by building mental links between things than the predicate logic we considered earlier. Note in particular how all the information about a particular object is concentrated on the node representing that object, rather than scattered around several clauses in logic. There is, however, some confusion here which stems from the imprecise nature of semantic nets. A particular problem is that we haven't distinguished between nodes representing classes of things, and nodes representing individual objects. So, for example, the node labelled Cat represents both the single (nameless) cat who sat on the mat, and the whole class of cats to which Tom belongs,which are mammals and which like cream. The is_a link has two different meanings – it can mean that one object is an individual item from a class, for example Tom is a member of the class of cats, or that one class is a subset of another, for example, the class of cats is a subset of the class of mammals. This confusion does not occur in logic, where the use of quantifiers, names and predicates makes it clear what we mean so:

Tom is a cat is represented by Cat(Tom)
The cat sat on the mat is represented by $\exists x \exists y(Cat(x) \wedge Mat(y) \wedge SatOn(x,y))$
A cat is a mammal is represented by $\forall x(Cat(X) \rightarrow Mammal(x))$

We can clean up the representation by distinguishing between nodes representing individual or instances, and nodes representing classes. The is_a link will only be used to show an individual belonging to a class. The link representing one class being a subset of another will

be labelled a_kind_of, or ako for short. The names instance and subclass are often used in the place of is_a and ako, but we will use these terms with a slightly different meaning in the section on Frames below. Note also the modification which causes the link labelled is_owned_by to be reversed in direction. This is in order to avoid links representing passive relationships. In general a passive sentence can be replaced by an active one, so "Tom is owned by John" becomes "John owns Tom". In general the rule which converts passive to active in English converts sentences of the form "X is Yed by Z" to "Z Ys X". This is just an example (though often used for illustration) of the much more general principle of looking beyond the immediate surface structure of a sentence to find its deep structure. The revised semantic net is:

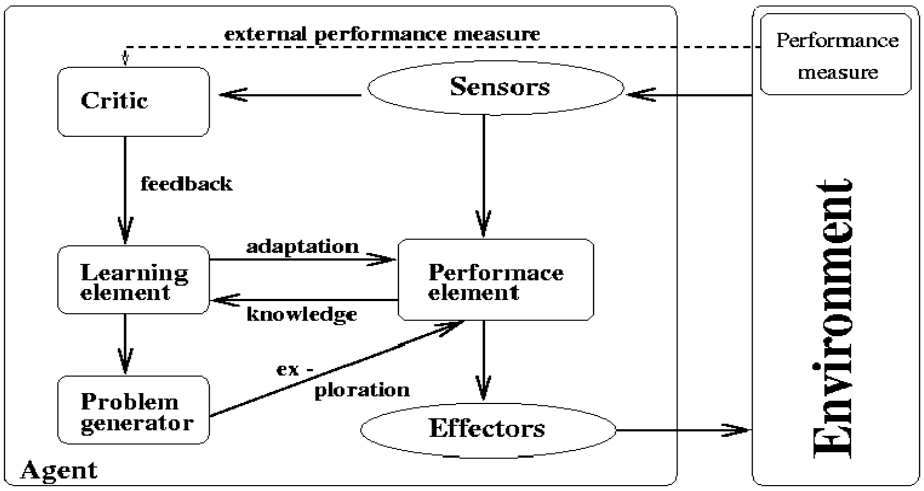


Note that where we had an unnamed member of some class, we have had to introduce a node with an invented name to represent a particular member of the class. This is a process similar to the Skolemisation we considered previously as a way of dealing with existential quantifiers. For example, "Tom caught a bird" would be represented in logic by $\exists x(bird(x) \wedge caught(Tom,x))$, which would be Skolemised by replacing the x with a Skolem constant; the same thing was done above where bird1 was the name given to the individual bird that Tom caught. There are still plenty of issues to be resolved if we really want to represent what is meant by the English phrases, or to be really clear about what the semantic net means, but we are getting towards a notation that can be used practically (one example of a thing we have skated over is how to deal
with mass nouns like "fur" or "cream" which refer to things that come in amounts rather than individual objects)

Q.5 a. Explain Hybrid representation systems. (8)

Answer:
A hybrid KR system is an implementation of a hybrid KR formalism consisting of two or more different sub formalisms. These sub formalism should be integrated through (i) a

representational theory, which explains what knowledge is to be-represented by what formalism, and (ii) a common semantics for the overall formalism, explaining in a semantic sound manner the relationship between expressions of different sub formalisms.The generalized architecture for a hybrid system is given in Fig



In general these systems consist of two different kinds of knowledge: The terminological knowledge, consisting of a set of concepts and roles defining a terminology, and the assertional knowledge, consisting of some logical formalism suited to represent general assertions.

KRYPTON

The system consists of two modules: the Terminological Box and the Assertional Box. The terminological box, or module, is based on the KL-ONE language -a representation system based on semantic networks and frames. The KRYPTON has been developed mainly from the work of KL-ONE. The difficulties in representing assertional knowledge using KL-ONE gives the idea of the integration of a theorem-prover and a KL-ONE-like language into a hybrid system. It is basically like a “tell-ask” module.

The most important feature introduced by KRYPTON is the notion of a Functional Approach to knowledge representation : KRYPTON is provided with a clear, implementation independent, description of what services are provided to the user. This Knowledge Level description is presented in the form of a formal definition of the syntax and semantics of the languages provided by the two modules along with the interaction between these two modules.

The set of primitives of the KRYPTON language vary from one presentation to another presentation of the language. In the complete form, the terminological box includes primitives for: Concept conjunction, value and number restriction on concepts, primitive sub-concept, concept decomposition, role differentiation, role chain, primitive subrole and role decomposition. And the assertional box provides a complete first-order logic language including the usual operators: Not, and, or, exists and for all.

KANDOR

The basic units of KANDOR are individuals and frames. Individuals are associated to objects in the real world and frames are associated to sets of these individuals. These units are

manipulated through the standard representational structures of frames, slots, restrictions, and slot fillers common to most frame-based systems. Each slot maps individuals into sets of values, called slot fillers, Elements of these sets can be other individuals, strings, or numbers. Frames in KANDOR have no assertion import; they look simply as descriptions of some set of individuals. There are two types of frames: Primitive and defined. To be an instance of a primitive frame, an individual must be explicitly specified as an instance of the frame when it is created.

To be an instance of a defined frame an individual must satisfy the conditions associated to the frame definition. There two types of conditions: Super-frames and restrictions. A super-frame is just another frame, and a restriction is a condition on a set of slots fillers for some slot. An individual satisfies the restriction if its slots fillers for that slot satisfy the condition.

KANDOR provides two main operations that require inferences to be made: Given an individual and a frame, determine whether the individual is an instance of the frame, arid, given two frames, it determines whether one frame is subset of another frame.

KANDOR has been used as the knowledge representation component of ARGON , which is an interactive information retrieval system which is designed to be used by non experts for retrieval purpose over a large, heterogeneous knowledge bases, possibly taken from a large number of sources or repositories.

BACK

The structure of a BACK represents as the same structure of KRYPTON, which contains an terminological box and an assertional box. One main aspect in the BACK implementation is the Balancedness of the formalisms involved. Although the fact that the reasoning in hybrid systems is frequently incomplete (because of efficiency requirements) sometimes leads to situations where one formalism allows to express something which obviously should have some impact on another formalism according to the semantics of the system, the incompleteness of the reasoning precludes this impact to be realized by the system. The formalisms of this type of systems are said to be “unbalanced”.

The main criteria taken into account in the development of the BACK system are the following: (i) The sub formalisms of the system should be balanced, (ii) the formalism should permit tractable inference algorithms covering almost all possible inferences, (iii) the assertional box formalism should be able to represent incomplete knowledge in a limited manner, (iv) the system should allow for extending the knowledge base incrementally (retractions are not considered) and (v) the system should reject assertional box entries which are inconsistent. The terminological language of BACK is more powerful than that of KRYPTON.

KL-TWO

The KL-TWO system is composed by two sub formalisms: PENNI, a modified version of the RUP (Reasoning Utility Package) system) and NIKL (New implementation of KL-ONE), a terminological reasoner in the KL-ONE [7] tradition. These two formalisms are complementary: PENNI is able to represent propositional assertions without any quantification and NIKL allows the representation of a simple class of universally quantified sentences. These sentences can be applied in PENNI to extend its propositional language with a limited form of quantification

The PENNI formalism consists of a database of propositional assertions, more specifically, a

data base of ground sentences of first-order logic without quantifiers. This database permits incremental assertions and retractions. Underlying the deductive mechanism of PENNI is a Truth Maintenance System (TMS) allowing all the useful operations that have been associated with such systems.
And the NIKL terminological reasoner allows the definition of composite concepts and roles through the use of structuring primitives and primitive concepts and roles. The primitives available in NIKL include: Concept conjunction, statement of the minimal number of role fillers, concept value restriction and role differentiation. The inference provided by NIKL is basically the sub assumption relation between concepts. It has been proved recently that the subsumption problem in NIKL is undecidable.
Two forms of hybrid reasoning are performed by the KL-TWO system: The forward reasoning, which is used to classify new assertions according to the concepts already defined in the NIKL knowledge base, and the backward reasoning, used to answer queries. Both mechanisms combine the inferences mechanism of PENNI and NIKL to perform their tasks

CAKE
The CAKE system was developed as a knowledge representation and reasoning facility for the Programmer's Apprentice project different from the previously presented hybrid systems. CAKE does not present complementary representation formalisms in which different types of knowledge are represented, but it uses its two formalisms to represent the same knowledge. The two formalisms present in CAKE are: A predicate calculus package which is based on the RUP (Reasoning Utility Package) system and a specialized, semantic network like formalism which is used to represent the structure of programs. This last formalism, called Plan Diagrams or simply Plans, was developed without any special concern about formal semantics but was only designed to fit the requirements of the program representation problem.
The current architecture of CAKE consists of eight layers: The bottom five layers forming the predicate calculus level and the top three layers corresponding to the Plan level. The predicate calculus layers, from bottom to top, and their functions are the following: (i) Truth Maintenance, unit propositional resolution, retraction and explanation, (ii) Equality, uniqueness of terms, congruence closure, (iii) Demons, pattern directed invocation, priority queues, (iv) Algebraic, commutativity, associativity, etc, lattices, Boolean algebras, (v) Types, type inheritance and functionality. The Plan layers are the following: (i) Plan Calculus, data and control flow graphs, abstract data types, (ii) Plan Recognition, flow graph parsing and recognition heuristics, (iii) Plan Synthesis, search of refinement space and synthesis heuristics.

b. Explain Dempster and Shafer's theory of evidences in detail.

Answer:
The Dempster–Shafer theory (DST**)** is a mathematical theory of evidence. It allows one to combine evidence from different sources and arrive at a degree of belief (represented by a belief function) that takes into account all the available evidence. The theory was first developed by Arthur P. Dempster and Glenn Shafer.

Dempster–Shafer theory is a generalization of the Bayesian theory of subjective probability; whereas the latter requires probabilities for each question of interest, belief functions base

degrees of belief (or confidence, or trust) for one question on the probabilities for a related question. These degrees of belief may or may not have the mathematical properties of probabilities; how much they differ depends on how closely the two questions are related.[5] Put another way, it is a way of representing epistemic plausibilities but it can yield answers that contradict those arrived at using probability theory.

Often used as a method of sensor fusion, Dempster–Shafer theory is based on two ideas: obtaining degrees of belief for one question from subjective probabilities for a related question, and Dempster's rule[6] for combining such degrees of belief when they are based on independent items of evidence. In essence, the degree of belief in a proposition depends primarily upon the number of answers (to the related questions) containing the proposition, and the subjective probability of each answer. Also contributing are the rules of combination that reflect general assumptions about the data.

In this formalism a degree of belief (also referred to as a mass) is represented as a belief function rather than a Bayesian probability distribution. Probability values are assigned to *sets* of possibilities rather than single events: their appeal rests on the fact they naturally encode evidence in favor of propositions.

Dempster–Shafer theory assigns its masses to all of the non-empty subsets of the entities that compose a system

Formal definition

Let *X* be the *universal set*: the set representing all possible states of a system under consideration. The power set

$$2^X$$

is the set of all subsets of *X*, including the empty set $\varnothing$. For example, if:

$$X = \{a, b\}$$

then

$$2^X = \{\varnothing, \{a\}, \{b\}, X\}.$$

The elements of the power set can be taken to represent propositions concerning the actual state of the system, by containing all and only the states in which the proposition is true.

The theory of evidence assigns a belief mass to each element of the power set. Formally, a function

$$m : 2^X \to [0, 1]$$

is called a *basic belief assignment* (BBA), when it has two properties. First, the mass of the empty set is zero:

$$m(\varnothing) = 0.$$

Second, the masses of the remaining members of the power set add up to a total of 1:

$$\sum_{A \in 2^X} m(A) = 1$$

The mass *m*(*A*) of *A*, a given member of the power set, expresses the proportion of all relevant and available evidence that supports the claim that the actual state belongs to *A* but to no particular subset of *A*. The value of *m*(*A*) pertains *only* to the set *A* and makes no additional claims about any subsets of *A*, each of which have, by definition, their own mass.

From the mass assignments, the upper and lower bounds of a probability interval can be defined. This interval contains the precise probability of a set of interest (in the classical sense), and is bounded by two non-additive continuous measures called **belief** (or **support**) and **plausibility**:

$$\mathrm{bel}(A) \leq P(A) \leq \mathrm{pl}(A).$$

The belief bel(*A*) for a set *A* is defined as the sum of all the masses of subsets of the set of interest:

$$\mathrm{bel}(A) = \sum_{B | B \subseteq A} m(B).$$

The plausibility pl(*A*) is the sum of all the masses of the sets *B* that intersect the set of interest *A*:

$$\mathrm{pl}(A) = \sum_{B | B \cap A \neq \varnothing} m(B).$$

The two measures are related to each other as follows:

$$\mathrm{pl}(A) = 1 - \mathrm{bel}(\overline{A}).$$

And conversely, for finite *A*, given the belief measure bel(*B*) for all subsets *B* of *A*, we can find the masses m(*A*) with the following inverse function:

$$m(A) = \sum_{B | B \subseteq A} (-1)^{|A - B|} \, \mathrm{bel}(B)$$

where $|A - B|$ is the difference of the cardinalities of the two sets.

It follows from the last two equations that, for a finite set X, you need know only one of the three (mass, belief, or plausibility) to deduce the other two; though you may need to know the values for many sets in order to calculate one of the other values for a particular set. In the case of an infinite X, there can be well-defined belief and plausibility functions but no well-defined mass function.

Q.6 a. Explain heuristics Search techniques. How are these techniques different from blind search techniques? (8)

Answer:

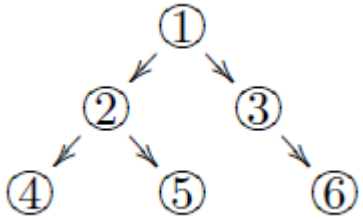
Q.6 b. Explain briefly Breadth first search and depth first search techniques. Write algorithm also. (8)

Answer:

Breadth-first : first, check all nodes on the same depth. In other words, visit all the nearest neighbours of the current root nodes; then move to the next depth level by considering the new visited nodes as the root nodes.

Depth-first : First, check all nodes in the same branch. In other words, visit each neighbour of the most recently visited node.

Consider the following graph:



Starting at root node 1, give the order in which the nodes will be visited by the breadth-first and depth-first algorithms.
Answer:
Breadth-first : 1 - 2 - 3 - 4 - 5 - 6

Depth-first : 1 - 2 - 4 - 5 - 3 - 6

Breadth First Search

Breadth first search traverses a graph in what is sometime called *level order*. Intuitively it

starts at the source node and visits all the nodes directly connected to the source. We call these level 1 nodes. Then it visits all the unvisited nodes connected to level 1 nodes, and calls these level 2 nodes etc.

The simple way to implement breadth first search is using a queue. In fact when hear you *breadth* you should think *queue*, and when you hear *depth* you should think *stack*. We have the algorithm output a tree representing the breadth first search, and store the level of each node. We have a parent array to store the search tree, a Level array to store the level, and a visited array to remember who has already been placed on the queue

```
Initialize:     Queue Q = source;  level[source] = 0, p[source] = nil;

                For all nodes x do visited[x] = false;

                visited[source]= true;



Loop:  While Q is not empty do {

                x = deleteq(Q);

                for all y adjacent to x do if visited[y] = false {

                        visited[y]=true; level[y]=level[x]+1; p[y] = x; addq(Q, y)}

        }
```

The total time for initializing is O(n) and the total time for the queuing operations is O(n) because each node is put on the queue exactly once. The total time in the main loop is O(e) because we look at each edge at most twice, once from each direction. This gives a time complexity of O(n+e).

Depth First Search

Depth first search (DFS) traverses a graph by going as deeply as possible before backtracking. It is surprisingly rich with potential for other algorithms. It also returns a search tree. It does not return the level of each node, but can return a numbering of the nodes in the order that they were visited. We first show a depth first search skeleton and define the different kinds classes of edges. Then we show how to augment the skeleton to solve two very basic algorithms: topological sorting, connected components. Each of leverages the power of DFS at a different location in the skeleton. We conclude with a sophisticated use of DFS that finds strongly connected components of a directed graph. You may recall that in month 0 we discussed a method in linear algebra using matrix multiplication that solved this algorithm in O(n^3). Our method will work in O(n+e)

Depth First Search Skeleton

DFS(G, s)

Mark s visited; Dfsnum[s] = count; count++; //count is a global counter initialized to 1.
/* Process s – previsit stage */
Recursive Loop: For every y adjacent to s do
If y is unvisited then {DFS(G, y); parent[y] = x;} else…
/* process edges {s,y}*/;
/* Process s – postvisit stage */
Mark s finished

Q.7 a. Write down the comparisons between conventional computers and neural networks. (8)

Answer:
Comparison between conventional computers and neural networks

(1.) Parallel processing
One of the major advantages of the neural network is its ability to do many things at once. With traditional computers, processing is sequential--one task, then the next, then the next, and so on. The idea of threading makes it appear to the human user that many things are happening at one time. For instance, the Netscape throbber is shooting meteors at the same time that the page is loading. However, this is only an appearance; processes are not actually happening simultaneously.
The artificial neural network is an inherently multiprocessor-friendly architecture. Without much modification, it goes beyond one or even two processors of the von Neumann architecture. The artificial neural network is designed from the onset to be parallel. Humans can listen to music at the same time they do their homework--at least, that's what we try to convince our parents in high school. With a massively parallel architecture, the neural network can accomplish a lot in less time. The tradeoff is that processors have to be specifically designed for the neural network.

(2.) The ways in which they function
Another fundamental difference between traditional computers and artificial neural networks is the way in which they function. While computers function logically with a set of rules and calculations, artificial neural networks can function via images, pictures, and concepts.
Based upon the way they function, traditional computers have to learn by rules, while artificial neural networks learn by example, by doing something and then learning from it. Because of these fundamental differences, the applications to which we can tailor them are extremely different. We will explore some of the applications later in the presentation.

(3.) Self-programming

The "connections" or concepts learned by each type of architecture is different as well. The von Neumann computers are programmable by higher level languages like C or Java and then translating that down to the machine's assembly language. Because of their style of learning, artificial neural networks can, in essence, "program themselves." While the conventional computers must learn only by doing different sequences or steps in an algorithm, neural networks are continuously adaptable by truly altering their own programming. It could be said that conventional computers are limited by their parts, while neural networks can work to become more than the sum of their parts.

(4.) Speed

The speed of each computer is dependant upon different aspects of the processor. Von Neumann machines requires either big processors or the tedious, error-prone idea of parallel processors, while neural networks requires the use of multiple chips customly built for the application.

b. Explain working of inference engine in an expert system using suitable examples. (8)

Answer:

Inference Engine

The inference engine is the main processing element of the expert system. The inference engine chooses rules from the agenda to fire. If there are no rules on the agenda, the inference engine must obtain information from the user in order to add more rules to the agenda. It makes use of knowledge base, in order to draw conclusions for situations. It is responsible for gathering the information from the user, by asking various questions and applying it wherever necessary.

Working of Inference Engine

The inference engine can be described as a form of finite state machine with a cycle consisting of three action states: ***match rules, select rules,*** and ***execute rules***. Rules are represented in the system by a notation called predicate logic.

In the first state, match rules, the inference engine finds all of the rules that are satisfied by the current contents of the data store. When rules are in the typical *condition-action* form, this means testing the conditions against the working memory. The rule matchings that are found are all candidates for execution: they are collectively referred to as the *conflict set*. Note that the same rule may appear several times in the conflict set if it matches different subsets of data items. The pair of a rule and a subset of matching data items is called an *instantiation* of the rule.

In many applications, where large volume of data are concerned and/or when performance time considerations are critical, the computation of the conflict set is a non-trivial problem. Earlier research work on inference engines focused on better algorithms for matching rules to

data. The Rete algorithm, developed by Charles Forgy, is an example of such a matching algorithm; it was used in the OPS series of production system languages. Daniel P. Miranker later improved on Rete with another algorithm, TREAT, which combined it with optimization techniques derived from relational database systems.

The inference engine then passes along the conflict set to the second state, select rules. In this state, the inference engine applies some selection strategy to determine which rules will actually be executed. The selection strategy can be hard-coded into the engine or may be specified as part of the model. In the larger context of AI, these selection strategies are often referred to as heuristics following Allen Newell's Unified theory of cognition.

In OPS5, for instance, a choice of two conflict resolution strategies is presented to the programmer. The LEX strategy orders instantiations on the basis of recency of the time tags attached to their data items. Instantiations with data items having recently matched rules in previous cycles are considered with higher priority. Within this ordering, instantiations are further sorted on the complexity of the conditions in the rule. The other strategy, MEA, puts special emphasis on the recency of working memory elements that match the first condition of the rule. (The latter heuristic is heavily used in means-ends analysis.)

Finally the selected instantiations are passed over to the third state, execute rules. The inference engine executes or fires the selected rules, with the instantiation's data items as parameters. Usually the actions in the right-hand side of a rule change the data store, but they may also trigger further processing outside of the inference engine (interacting with users through a graphical user interface or calling local or remote programs, for instance). Since the data store is usually updated by firing rules, a different set of rules will match during the next cycle after these actions are performed.

The inference engine then cycles back to the first state and is ready to start

ver again. This control mechanism is referred to as the *recognize-act cycle*. The inference engine stops either on a given number of cycles, controlled by the operator, or on a *quiescent* state of the data store when no rules match the data.

General Types of Inferencing:

In simple rule-based systems, there are two kinds of inference, *forward chaining* and *backward chaining*.

Forward chaining: data gets put into working memory. This triggers rules whose conditions match the new data. These rules then perform their actions. The actions may add new data to memory, thus triggering more rules. And so on. This is also called *data-directed* inference, because inference is triggered by the arrival of new data in working memory.

Backward chaining: the system needs to know the value of a piece of data. It searches for rules whose conclusions mention this data. Before it can use the rules, it must test their conditions. This may entail discovering the value of more pieces of data, and so on. This is also called

goal-directed inference, or *hypothesis driven*, because inferences are not performed until the system is made to prove a particular goal (i.e. a question).

Q.8 a. Differentiate between neural networks and expert system. (8)

Answer:

CHARACTERISTICS	**TRADITIONAL COMPUTING (including Expert Systems)**	**ARTIFICIAL NEURAL NETWORKS**
Processing style Functions	Sequential Logically (left brained) via Rules Concepts Calculations	Parallel Gestault (right brained) via Images Pictures Controls
Learning Method Applications	by rules (didactically) Accounting word processing math inventory digital communications	by example (Socratically) Sensor processing speech recognition pattern recognition text recognition

Table 1 Comparison of Computing Approaches

Characteristics	**Von Neumann Architecture Used for Expert Systems**	**Artificial Neural Networks**
Processors	VLSI (traditional processors)	Artificial Neural Networks; variety of technologies; hardware development is on going
Processing Approach	Separate	The same
Processing	Processes problem	Multiple,

Approach	rule at a one time; sequential	simultaneously
Connections	Externally programmable	Dynamically self programming
Self learning	Only algorithmic parameters modified	Continuously adaptable
Fault tolerance	None without special processors	Significant in the very nature of the interconnected neurons
Neurobiology in design	None	Moderate
Programming	Through a rule based complicated	Self-programming; but network must be set up properly
Ability to be fast	Requires big processors	Requires multiple custom-built chips

Table 2 Comparisons of Expert Systems and Neural Networks.

Q.8 b. What are the advantages and disadvantages of Neural network computing?

Answer:

Advantages:

- A neural network can perform tasks that a linear program cannot.
- When an element of the neural network fails, it can continue without any problem by their parallel nature.
- A neural network learns and does not need to be reprogrammed.
- It can be implemented in any application and without any problem.
- Does not use pre-programmed knowledge base
- Suited to analyze complex pattern
- Have no restrictive assumptions
- Allows for qualitative data
- Can handle noisy data
- Can overcome autocorrelation
- User-friendly: clear output, and robust and flexible

Disadvantages:

- The neural network needs training to operate.

- The architecture of a neural network is different from the architecture of microprocessors therefore needs to be emulated.
- Requires high processing time for large neural networks.
- The neural network requires high quality data,
- Variables must be carefully selected a priori,
- Risk of overfitting,
- Requires a definition of architecture,
- Long processing time,
- Possibility of illogical network behavior, and
- Large training sample required

Q.9 a. Explain how AI can be used in solving Real-World problems and in enhancing scalability.

Answer: Page 270 of Reference-I

Q.9 b. What do you mean by multi-agent systems (MAS)? Why are these successful?

Answer: Page 278 of Reference-I

TEXT BOOK

I. Introduction to Artificial Intelligence, Rajendra Akerkar, PHI, 2005