ARTIFICIAL INTELLIGENCE

UNIT-1

Introduction of Artificial Intelligence : What is AI ? The Importance of AI. AI and related fields. Introduction to Natural Language Processing .

UNIT-2

Knowledge : General Concepts, Definition and Importance of Knowledge, Knowledge based system, representation of Knowledge, Knowledge Organization , Knowledge Manipulation , Acquisition of Knowledge.

UNIT-3

LISP AND AI PROGRAMMING LANGUAGES : Introduction to LISP : Syntax and Numeric Functions, Basic List Manipulation Functions in LISP , Functions, Predicates, and Conditionals, Input, Output, and Local Variables, Iteration and Recursion, Property List and arrays, PROGLOG and Other AI Programming Languages.

UNIT-4

FORMALIZED SYMBOLIC LOGICS : Introduction , Syntax and Semantics for Propositional Logic , Syntax and Semantics for FOPL , Properties of Wffs , Conversion to Clausal Form, Inference Rules , The Resolution Principle , Representations Using Rules.

UNIT-5

Introduction to Expert System , Characteristics features of Expert System, Applications of Expert System. Importance of Expert System.

BOOKS :

1. Clockskin, W.F. and Mellish, C.S. : Programming in prolog, Narosa publ. House.
2. Charniak, E. : Introduction of Artificial Intellegence, Narosa publ. House.
3. Winston,P.H. : LISP, NArosa publ. House.
4. Milner : Common LISP : A tutorial , Prentice Hall Inc. 1988.
5. Marcellus : Expert Systems Programming in TURBO PROLOG, P.H.I. 1989.
6. Elaime R. : Artificial Intelligence, 1983.
7. Hunt, E.B. : Artificial intelligence, Academic Press 1975
8. Lloyd,J. : Foundation of Logic Programming, Springer-Verlag 1982.
9. clark, K.L. : Micro Prolog , Prentice Hall india.1987.

UNIT-1

- Introduction of Artificial Intelligence :
- What is AI ?
- The Importance of AI.
- AI and related fields.
- Introduction to Natural Language Processing .

Introduction of Artificial Intelligence:

Artificial Intelligence is one of the newest fields of intellectual research, but its foundations began thousands of years ago. In studying Artificial Intelligence, it is useful to have an understanding of the background of a number of other subjects, primarily philosophy, linguistics, psychology, and biology.

There are numerous definitions of what artificial intelligence is.

- Artificial intelligence is the study of systems that act in a way that to any observer would appear to be intelligent.
- Artificial Intelligence involves using methods based on the intelligent behavior of humans and other animals to solve complex problems.
- Artificial Intelligence is the study of human intelligence and actions replicated artificially, such that the resultant bears to its design can think and act like humans rationality (doing the right thing).
- The art of creating machines that performs functions that require intelligence when performed by humans.
- **A**rtificial Intelligence is a branch of Science which deals with helping machines finds 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.
- GPS - General Problem Solver.

Exactly what is AI?

AI means Artificial Intelligence which is a branch of computer science concern with the study and creation of computer systems that exhibit some form of intelligence as:

- systems that learn new concepts and tasks
- Systems that can reason and draw useful conclusions about the world around us.
- Systems that can understands a natural language or perceive(feel) and comprehend(understand or participate) a visual scene.
- Systems that perform other type of feats that require human types of intelligence.

Artificial intelligence can be viewed from a variety of perspectives.

- From the perspective of **intelligence**, AI is making machines "intelligent" acting as we would expect people to act like knowledge, Expert problem solving .

- From a **business** perspective AI is a set of very powerful tools, and methodologies for using those tools to solve business problems.
- From a **programming** perspective, AI programs focus on symbols rather than numeric processing, problem solving (achieve goals) and search (BFS, DFS). AI programming languages include: LISP (List Processing), developed in the 1950s and PROLOG(Program Logic) was developed in the 1970s. They are the early programming language strongly associated with AI. Artificial Intelligence is a new electronic machine that stores large amount of information and process it at very high speed.

AI requires an understanding of related terms such as intelligence, knowledge, reasoning, thought, cognition(gyan or bodh), learning and number of computer related terms.

Meaning of Intelligence: Ability of acquire, understand and apply knowledge. Or the ability to exercise thought and reasons.

The Importance of AI:

AI may be one of the most important developments of this century. Many countries have focus on importance of AI and make a plane to development for research work in AI and for this they have passed budget like-

Japanese were first announced fifth generation computer in October 1981 with budget of about one billion dollars and produce systems that can converse a natural language, understand speech and visual scenes, learn and refine their knowledge, make decisions and exhibit other human traits.

Following Japanese other countries plan to some form of AI program. British initiated a plan called Alvey Project with respectable project.

The European Common Market countries have jointly initiated a plan called ESPRIT program.

Other countries also initiated for AI research and development plans as French, Canada, Soviet Union, Italy, Austria, Irish Republic and Singapore.

The United State push forward in AI research. First in 1983 there was formation of consortium known as Microelectronics and computer technology corporation (MCC) headquartered in Austin, Texas. It develop advanced technologies that apply AI techniques like VLSI. Second development of DARPA (Defense Advanced Research Projects Agency) has increase AI research, including development support in three significant programs:

1) Development of an Autonomous land vehicle(ALV) – driverless military vehicle.
2) Development of pilot's associate- an Expert system which provides assistance to fighter pilots.
3) The strategic Computing program- an AI based military supercomputer project.

Early work in AI:

Early 1936: **Alan Turing**, sometimes regarded as the father of AI. He had demonstrated a simple computer processor (also called Turing machine) that manipulated symbols as well as numbers.

1952-1955: Chess playing programs developed by researchers like Claude Shannon at MIT and Allen Newell at RAND corporation(1972).

Mid 1950 is the official birth of AI.

1956-57: Logic Theorist, First automatic theorem proving program by Newell, Shaw and Simon. List Processing language also developed.

1961-65: A.L. Samuel developed a program which learned to play checkers at a master level.

1965: J.A. Robinson introduce resolution as an inference method in logic.

1965: Work on DENDRAL(First knowledge based expert system based on molecular structure) was begun at Stanford University by J.Lederberg.

1968: Work on MACSYMA(Intractive program to solve mathematical problem) was initiated at MIT by Carl Engleman.

AI and related fields:

AI is generally associated with Computer Science, but it has many important links with other fields such as Math, 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.

Some other fields of AI are-

- **Logical AI**
  What a program knows about the world in general the facts of the specific situation in which it must act, and its goals are all represented by sentences of some mathematical logical language. The program decides what to do by inferring that certain actions are appropriate for achieving its goals.

- **Search**
  AI programs often examine large numbers of possibilities, e.g. moves in a chess game or inferences by a theorem proving program.

- **Pattern Recognition**
  When a program makes observations of some kind, it is often programmed to compare what it sees with a pattern. For example, a vision program may try to match a pattern of eyes and a nose in a scene in order to find a face.

- **Representation**
  Facts about the world have to be represented in some way.

- **Inference**
  Like when we hear of a bird, we can infer that it can fly, but this conclusion can be reversed when we hear that it is a penguin.

- **Common sense knowledge and reasoning**
  This is the area in which AI is farthest from human-level, in spite of the fact that it has been an active research area since the 1950s.

- **Learning from experience**
  Programs can only learn what facts or behaviors their formalisms can represent

- **Planning**
  Planning programs start with general facts, they generate a strategy for achieving the goal.

- **Epistemology**
  This is a study of the kinds of knowledge that are required for solving problems in the world.

- **Ontology**
  Ontology is the study of the kinds of things that exist. In AI, the programs and sentences deal with various kinds of objects, and we study what these kinds are and what their basic properties are.

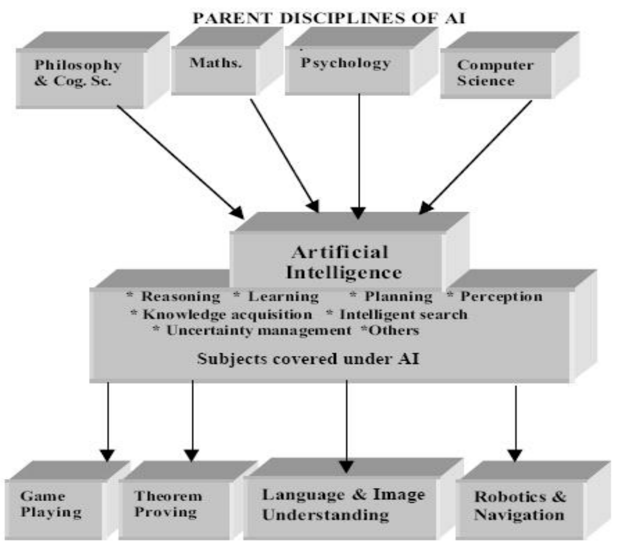
- **Heuristics**
  A heuristic is a way of trying to discover something or an idea imbedded in a program. Heuristic functions are used in some approaches to search to measure how far a node in a search tree seems to be from a goal. Heuristic predicates that compare two nodes in a search tree to see if one is better than the other, i.e. constitutes an advance toward the goal, may be more useful.

- **Genetic Programming**
  Genetic programming is a technique for getting programs to solve a task by mating random Lisp programs and selecting fittest in millions of generations.

Application of AI:

- **Game Playing**
  You can buy machines that can play master level of game like chess. There is some AI in them, that play well against people.
- **Speech Recognition**
  it is possible to instruct some computers using speech, in place of keyboard and mouse which is more convenient.
- **Understanding Natural Language** A collection of techniques used to enable computers to “understand” human language.
- **Expert Systems**
  One of the first expert systems was MYCIN in 1974, which diagnosed bacterial infections of the blood and suggested treatments. It did better than medical students

The applications of AI are shown :


PARENT DISCIPLINES OF AI
Philosophy & Cog. Sc.
Maths.
Psychology
Computer Science
Artificial Intelligence
* Reasoning * Learning * Planning * Perception
* Knowledge acquisition * Intelligent search
* Uncertainty management *Others
Subjects covered under AI
Game Playing
Theorem Proving
Language & Image Understanding
Robotics & Navigation

- Consumer Marketing- we can use any kind of credit/ATM/store card while shopping.
- Identification Technologies like ATM cards.
- Biometric Identification, like biometric signature, Face, eyes, fingerprints, voice pattern.
- Machine Translation where Language problems in international business like English to Russian.

Natural Language Processing: NLP

- NLP is the branch of computer science focused on developing systems that allow computers to communicate with people using everyday language. It is also called Computational Linguistics.
- NLP is a sub field of AI which deals with the methods of communicating with a computer in once on natural language. It includes understanding and generation as well as other task such as multi lingual translation.
- Natural languages are used by humans for communication. They are distinctly different from formal languages, such as C++, Java, and PROLOG because they are not ambiguous.
- A system that can work with one human language cannot necessarily deal with any other human language.

Advantages of NLP:

NLP minimizes many hardships a person's faces while communicating with the computer.

1) One need not be a computer literate to communicate with it.
2) One can dispose of special query languages like SQL, which presently humans use to access information from databases.

3) Information generated worldwide daily from various sources and presented in news papers, magazines and written material can be condensed and presented to the user in a capsule form.
4) There is a human touch in the NLP systems. One feels at home by directly communicating with the machine.
5) NLP system when coupled with speech recognition and synthesis system is certain to give the humans a shot in the arm.

Major Problem with NLP program:

Developing programs that understand a natural language is a difficult problem.

- Natural Languages are large. They contain an infinite number of different sentences; new ones can always be produced.
- Also, there is much ambiguity in a natural language. Many words have several meanings such as can, bear, fly and orange and sentences can have different meanings in different context. This makes the creation of programs that "Understand" a natural language, one of the most challenging tasks in AI.

Other Problems with NLP: NLP requires the system to analyze the sentence and retrieve the correct meaning. Ambiguity of the words used and their meaning in their respective context are the major bottle necks in NLP. Following will explain this-

1) Words used by one set of people could have different meaning for a different set of people. Eg. " This flat is terrible," to an Englishman flat means house while for American it is a puncture.
2) The functional structure of the sentence itself can give arise to ambiguities. E.g. "I saw taj Mahal flying over Agara.", who is flying taj mahal or the person who spoke the sentence.
3) Extensive use of pronounce increases ambiguities. E.g. "Ravi went to the supermarket. He found his favorite brand of coffee powder in rack five. He paid for it and left. The question is- To what object the pronoun "It" refers to, the supermarket or the coffee powder or rack five?
4) Conjunction used in Natural Language to avoid repeating of phrases also cause NLP problems. E.g. Ram and Shyam went to a restaurant. While Ram had a cup of coffee and Shyam had tea. In the sentence, we have suppressed the term "Had a cup of " for Shyam but the meaning as well understood by humans while it might be difficult for the machine.
5) Ellipsis is a major problems with a NLP systems finds difficult to manage. In ellipsis, one does not state some words but leaves it to the audience to fill it up. Eg. "What is length of river Gangas? Of river covery? "

Because of these five problems, we find it difficult to build NLP systems.

Meaning of Linguistics:

In a natural language, sentence is the basic language element that made up of words and has a subject and a predicate. Sentences are classified by structure and predicate like- simple, compound and complex sentences. Sentences are used to assert, query and describe. Sentences are declarative, imperative, interrogative or exclamatory.

A word functions in a sentence as a part of speech which are noun(Ram), pronouns(He), verbs(come), adjectives(good), adverbs(very), prepositions(in), conjunctions(or) and interjections(oh!). Phrases are the part of words but acts as a single unit within a sentence. All these form the building blocks for syntactic structures.

Levels /stages/steps / Components for Natural Language Understanding:

A language understanding program must have

- Considerable knowledge about the structure of language including what the words are and how they combine into phrases and sentences.
- It must know the meaning of words and how they contribute the meanings of a sentence and to the context within which they are being used.
- Finally, a program must have some general world knowledge as well as knowledge of what humans know and how they reason.

NLP program perform following steps for a input string of words-

1) Detect a string of words.
2) Sentences are parsed or analyzed to determine their structure(syntax) and grammatical correctness.
3) The meaning (semantics) of sentences are determined.
4) Appropriated representation structure of sentences is created for the inferencing programs.

To perform above steps Knowledge of Natural languages are sometimes classified according to following levels:

1) Phonological: This is knowledge which relates sounds to the words we recognize. A phoneme is smallest unit of sound.
2) Morphological : This is lexical knowledge which relates to word construction from basic unites called morphemes. Individual words are analyzed into their components and non word tokens such as punctuation are separated from the words. Ex: friendly is separated as friend and ly.
3) Syntactic: This knowledge relates to linear sequence of words to form grammatically correct sentences in the language.
4) Semantic : This knowledge is concern with the meaning of words and phrases and how they combine to form sentence meanings.
5) Pragmatic : This is high level knowledge which relates to use of sentences in different context and how the context affects the meaning of the sentences.
6) World: World knowledge relates to the language a user must have in order to understand and carry on a conversation. It must include an understanding of the other person's belief and goal.

General Approaches to language Understanding:

Understanding written language or text is easier then understanding speech because to understand a speech, a program must have all the capabilities of text understanding program plus facilities to map spoken sounds into textual form (pattern recognition).

There have been three different approaches taken in the development of natural language understanding programs.

1) The use of keyword and pattern matching:

It uses some template like: "I am_______________", "I don't like________________". They are matched against input sentence. Each input template has associated with one or more output template used to produce a response to given input.

2) <u>Comparing and matching the input to real world situation (Scenario representations):</u> It uses frames and script. It need to build larger knowledge structures. Here stored situations or events are recalled for the use of understanding new situations and filled missing details.

3) <u>Combine syntactic (structural) and Semantic directed analysis</u>: Most popular. Parsers are used to analyze individual sentences and to build structures that can be used directly or transformed into required knowledge formats.

Grammars and languages:

A language L can be considered as a set of strings of finite or infinite length, where string is constructed by concatenating basic atomic elements called symbols. The finite set V of symbols of the language is called the alphabet or vocabulary.
Well formed sentences are constructed using a set of rules called a grammar G. language generated by the grammar G is denoted L(G).

More formally, we define a grammar G as-

$G=(V_n,V_t,S,P)$

Where-
V_n: is a set of non terminal symbol.
V_t: is a set of terminal symbols.
S: is a starting symbol.
P: is a Finite set of production or rewrite rules.

As an example of a simple of grammar G, we choose one which has component parts of constitutes from English with vocabulary Q given by-

Q_n={S,NP,N,VP,V,ART}

Q_t={boy, popsicle, frog, ate, kissed, flew, the, a}

And its production rule can be given as-

P: S→NP VP
NP→ART N
VP→V NP
N→boy|popsicle|frog
V→ate|kissed|flew
ART→ the|a

Where the | indicates alternative choices.
S- is the initial symbol i.e. sentence
NP- Noun Phrase
VP- Verb Phrase
N- Noun
V- Verb
ART- Article
To generate a sentence, the rules from P are applied sequentially starting with S and proceeding until all non terminal symbols are eliminated.

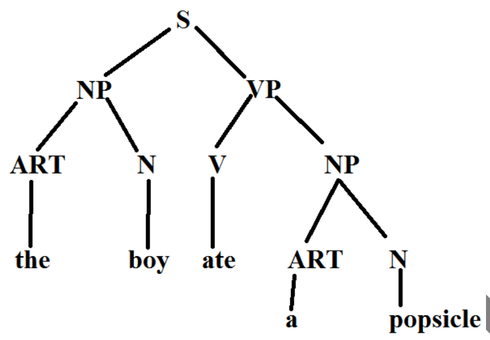
Ex: The boy ate a popsicle. This sentence can be generated using the following sequence of production rules:

S→NP VP
→ART N VP
→the N VP
→the boy VP
→the boy V VP
→the boy ate NP
→the boy ate ART N
→ the boy ate a N
→ the boy ate a popsicle.

It should be clear that a grammar does not guarantee the generation of meaningful sentences, only that they are structurally correct. For example, a grammatically correct, but meaningless sentence like "The popsicle flew a frog" can be generated with this grammar.

Structural Representations:

It is convenient to represent sentences as a tree or graph to help to expose the structure of the constituent parts. For example, the sentence " The boy ate a popsicle" can be represented as –



The left sub tree is a noun phrase and the right sub tree a verb phrase. The leaf or terminal nodes contain the terminal symbol from Vt.

A tree structure such as the above represents a large number of English sentences. It also represents a large class of ill formed strings that are non sentences like " The popsicle flew a frog". This satisfied the above structure, but has no meaning.

UNIT-2

- Knowledge : General Concepts
- Definition and Importance of Knowledge
- Knowledge based system
- Representation of Knowledge
- Knowledge Organization
- Knowledge Manipulation
- Acquisition of Knowledge.

Knowledge: General Concepts

In order to solve the complex problems encountered in AI, one generally needs a large amount of knowledge, and suitable mechanisms for representing and manipulating all that knowledge. Knowledge can take many forms. Some simple examples are:

- John has an umbrella.
- It is raining.
- An umbrella stops you getting wet when it's raining.
- An umbrella will only stop you getting wet if it is used properly.
- Umbrellas are not so useful when it is very windy.

So, we need to study, how should an AI agent store and manipulate knowledge like this? Knowledge Based System can become effective as problem solvers only when specific knowledge was brought to bear on the problems.

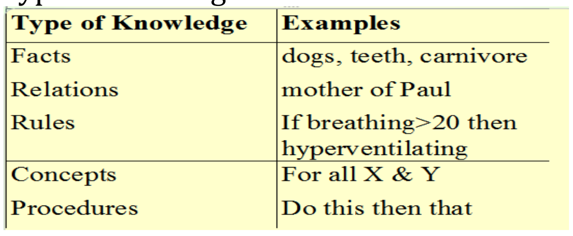
Definition and Importance of Knowledge:

- "The fact or condition of *knowing something* with familiarity gained through experience or association." (Webster's Dictionary, 1988).
- Knowing something via seeing, hearing, touching, feeling, and tasting.
- "The fact or condition of *being aware of* something" .(Ex. Sun is hot, balls are round, sky is blue,…)
- **Knowledge** is a theoretical or practical understanding of a subject or a domain.
- Knowledge is the sum of what is currently known.
- Knowledge includes and requires the use of data and information.
  Data: Raw facts, figures, measurements.
  Information: Refinement and use of data to answer specific question.
  Knowledge: Refined information
- Knowledge can be defined as the body of facts and principles accumulated by human-kind or the act, fact, or state of knowing.
- The meaning of knowledge is closely related to the meaning of intelligence. Intelligent requires the possession of and access to knowledge.
- A common way to represent knowledge external to a computer or a human is in the form of written language.
- Example:

- ✓ Ramu is tall – This expresses a simple fact, an attribute possessed by a person.
- ✓ Ramu loves his mother – This expresses a complex binary relation between two persons.

- Sources of Knowledge:
  1) **Documented** (books, journals, procedures, films, databases)
  2) **Undocumented** (people's knowledge and expertise, people's minds, other senses)

- Types Knowledge:



Type of Knowledge	Examples
Facts	dogs, teeth, carnivore
Relations	mother of Paul
Rules	If breathing>20 then hyperventilating
Concepts	For all X & Y
Procedures	Do this then that

- Categories of Knowledge:
  1) Declarative- (descriptive, facts, shallow knowledge)
  2) Procedural- (way things work, tells how to make inferences)
  3) Semantic- (symbols)
  4) Episodic- (autobiographical, experimental)
  5) Meta-knowledge- (Knowledge about the knowledge)

- <u>Procedural knowledge</u> is compiled knowledge related to the performance of some task. For example, the steps used to solve an algebraic equation.

- <u>Declarative knowledge</u> is passive knowledge expressed as statements of facts about the world. For example, personnel data in a database, such data are explicit pieces of independent knowledge.

<u>Characteristics of Good knowledge:</u>

Knowledge should be:

◆ accurate
◆ non redundant
◆ consistent
◆ as complete as possible (or certainly reliable enough for conclusions to be drawn)

Importance of knowledge:

AI has given new meaning and importance to knowledge. Now a day's system can reason and draw conclusion only by embedding knowledge with AI .

Imagine being able to purchase an untiring, reliable advisor that gives high level professional advise in a specialized area's such as:

- Manufacturing technique
- Sound financial strategies
- Ways to improve one's health
- Top marketing sectors and strategies
- And many others important matters.

Knowledge-Based System?(KBS)

○ A system which is built around a knowledge base. i.e. a collection of knowledge, taken from a human, and stored in such a way that the system can *reason* with it.

○ Medical diagnosis, geological analysis, and chemical compound identification are examples of tasks to which Knowledge Base systems have been applied.

○ Knowledge Base systems are often called expert systems because the problems in their application domain are usually solved by human experts.

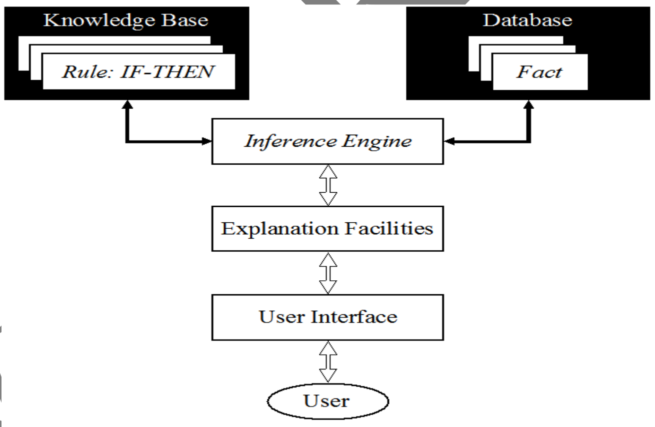
○ For example medical diagnosis is usually performed by a doctor.

○ KBS is Heuristic rather than algorithmic

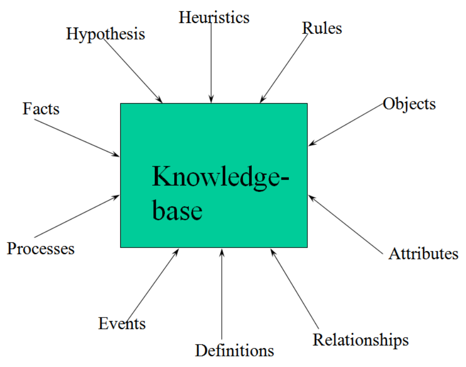
○ Knowledge is separated from how it is used:

KBS = knowledge-base + inference engine

Architecture Knowledge Base System:



Knowledge-base: The **knowledge base** contains the domain knowledge useful for problem solving.



In a rule-based expert system, the knowledge is represented as a set of rules. Each rule specifies a relation, recommendation, directive, strategy or heuristic and has the:

IF (condition) THEN (action) structure.

When the condition part of a rule is satisfied, the rule is said to ***fire*** and the action part is executed.

Ex: if num_wheel =4 and motor=yes then vehicle =automobile

The **database:**

includes a set of facts used to match against the IF (condition) parts of rules stored in the knowledge base.

The inference engine:

carries out the reasoning whereby the expert system reaches a solution. It links the rules given in the knowledge base with the facts provided in the database.

The explanation facilities:

enable the user to ask the expert system ***how*** a particular conclusion is reached and ***why*** a specific fact is needed. An expert system must be able to explain its reasoning and justify its advice, analysis or conclusion.

The user interface:

is the means of communication between a user seeking a solution to the problem and an expert system.

Representing the knowledge

- The object of a knowledge representation is to express knowledge in a computer tractable form, so that it can be used to enable our AI agents to perform well.
- Search-based problem solving programs require some knowledge to be implemented. Knowledge can be a particular states or path toward solution, rules, etc.
- Nevertheless large amount of knowledge as well as some means of manipulating that knowledge is required so as to create solutions for new problems.
- Before being used this knowledge must be represented in a particular way with a certain format.
- In the representation there are two different entities that must be considered:- fact and Representation of facts.

  Facts: truths in some relevant world. These are things that we want to represent.

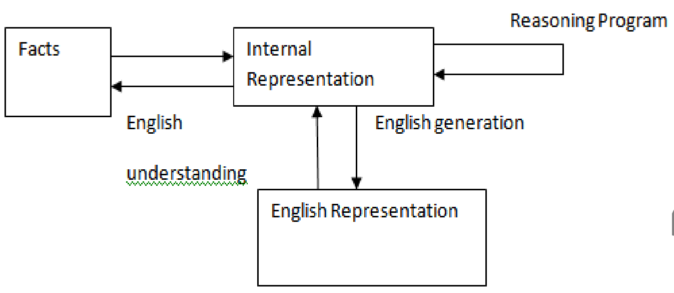
  Representation of facts in some chosen formalism. These are things that can actually be manipulated.

- Structuring of these entities can be done in two levels:

  The *knowledge level* at which facts are described

  The *symbol level* at which representation of some objects at the knowledge-level are defined in terms of symbols that can be manipulated by programs.

Mappings between Facts and Representation:



For Example: we can use mathematical logic as the representation formalism. Consider the English sentences below.

Spot is a dog

This fact can also be represented in logic as follows:-

Dog(Spot)

Approach to Knowledge Representation:

There are multiple techniques for knowledge representation. Some of them are-

- Rules
- Semantic Networks
- Frames
- Propositional and Predicate Logic

Rules Based:

If pulse is absent and breathing is absent

Then person is dead.

Propositional & Predicate Logic:

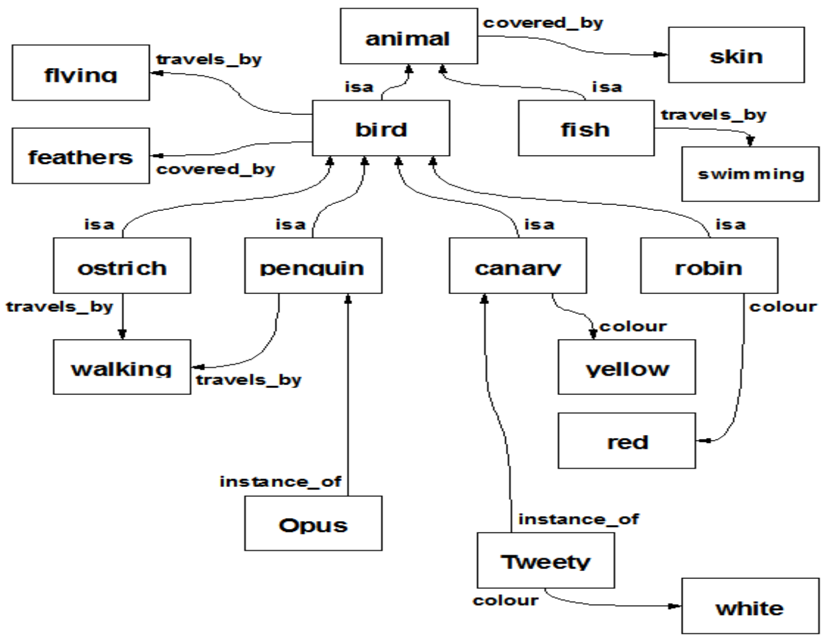
based on calculus
J=Passed assignment
K = Passed exam
Z = J and K
Student has passed assignment and passes exam

Semantic Networks:

It is based on-

- Graphical depictions
- Nodes and links
- Hierarchical relationships between concepts
- Reflects inheritance

In semantic networks knowledge is represented as a collection of concepts, represented by nodes (shown as boxes in the diagram), connected together by relationships, represented by arcs (shown as arrows in the diagram). certain arcs - particularly *isa* arcs - allow inheritance of properties.

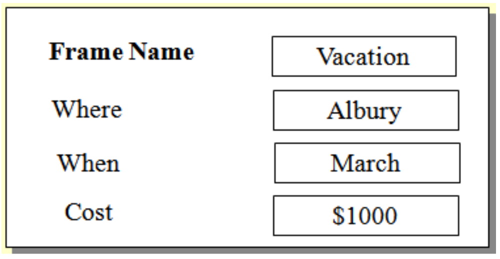


Frames:

The idea of frame hierarchies is very similar to the idea of class hierarchies found in object-orientated programming.

A frame system is a hierarchy of frames. Each frame has:

- a name.
- slots: these are the properties of the entity that has the name, and they have values. A particular value may be:
  - a default value
  - an inherited value from a higher frame



Frame Name	Vacation
Where	Albury
When	March
Cost	$1000

- In the higher levels of the frame hierarchy, typical knowledge about the class is stored.
- The value in a slot may be a range or a condition.
- In the lower levels, the value in a slot may be a specific value, to overwrite the value which would otherwise be inherited from a higher frame.
- An instance of an object is joined to its class by an 'instance_of' relationship.
- A class is joined to its superclass by a 'subclass_of' relationship.
- Frames may contain both procedural and declarative knowledge.

The four fundamental components of a good representation:

- ✓ The lexical part – that determines which symbols or words are used in the representation's vocabulary.
- ✓ The structural or syntactic part – that describes the constraints on how the symbols can be arranged, i.e. a grammar.
- ✓ The semantic part – that establishes a way of associating real world meanings with the representations.
- ✓ The procedural part – that specifies the access procedures that enables ways of creating and modifying representations and answering questions using them, i.e. how we generate and compute things with the representation.

<u>Knowledge Organization:</u>

The organization of knowledge in memory is key to efficient processing. Knowledge Base System may require tens of thousands of facts and rules to perform their intended tasks. It is essential then that the appropriate facts and rules be easy to locate and retrieve. Otherwise, much time will wasted in searching and testing large numbers of items in memory.

Knowledge can be organized in memory for easy access by a method known as indexing. It amounts to grouping the knowledge in way that keywords can be used to access the group. The keywords "point" to the knowledge groups. As a result, the search for some specific chunk of knowledge is limited to the group only, a fraction of the knowledge base rather than the whole memory.

The choice of representation can simplify the organizational and access operations. For example, frames linked together in a network represent a versatile organization structure. Each frame will contain all closely associated information about an object and pointers to related object frames making it possible to quickly gain access to this information. Subsequent processing then typically involves only a few related frames.

<u>Knowledge manipulation:</u>

Decision and actions in KBSs come from manipulation of the knowledge in specified ways. Typically, some form of input (from user) will initiate a search for a goal or decision. This require that known facts in the knowledge-base be located, compared(matched) and possibly altered in some way. This process may set up other sub goals and require further inputs and so on until a final solution is found. The manipulation are the computational equivalent of reasoning. This requires a form of inheritance or deduction, using the knowledge and inference rules.

All form of reasoning require a certain amount of searching and matching. In fact, these two operations by far consume the greatest amount of computation time in AI systems. For this reason it is important to have techniques available that limit the amount of search and matching required to complete any given task.

Much research has been done in these areas to find better methods. The research has paid off with methods which help to make many otherwise intractable problems solvable. They help to limit or avoid the so-called combinatorial explosion in problems which are so common in search.

Knowledge Acquisition:

Knowledge acquisition is the process by which knowledge available in the world is transformed and transferred into a representation that can be used by an expert system. World knowledge can come from many sources and be represented in many forms.

Knowledge acquisition is a multifaceted problem that encompasses many of the technical problems of knowledge engineering, the enterprise of building knowledge base systems. (Gruber).

Five stages for Knowledge acquisition:

1. Identification: - break problem into parts
2. Conceptualisation: identify concepts
3. Formalisation: representing knowledge
4. Implementation: programming
5. Testing: validity of knowledge

Knowledge Engineer(KR)

- Interacts between expert and Knowledge Base
- Needs to be skilled in extracting knowledge
- Uses a variety of techniques

The basic model of knowledge acquisition:

It requires that the knowledge engineer mediate between the expert and the knowledge base. The knowledge engineer elicits knowledge from the expert, refines it in conjunction with the expert and represents the knowledge in the knowledge base using a suitable knowledge structure.

Elicitation of knowledge done either manually or with a computer.

Manual:

- ◆ interview with experts.
- ◆ structured, semi structured, unstructured interviews.
- ◆ track reasoning process and observing.

Semi Automatic: Use a computerised system to support and help experts and knowledge engineers.
Automatic: Minimise the need for a knowledge engineer or expert.

Knowledge Acquisition Difficulties

- ◆ Knowledge is not easy to acquire or maintain
- ◆ More efficient and faster ways needed to acquire knowledge.
- ◆ System's performance dependant on level and quality of knowledge "in knowledge lies power."
- ◆ Transferring knowledge from one person to another is difficult. Even more difficult in AI. **Other Problems**

Other Reasons

- ◆ Experts busy or unwilling to part with knowledge.
- ◆ Methods for eliciting knowledge not refined.
- ◆ Collection should involve several sources not just one.
- ◆ It is often difficult to recognise the relevant parts of the expert's knowledge.
- ◆ Experts change

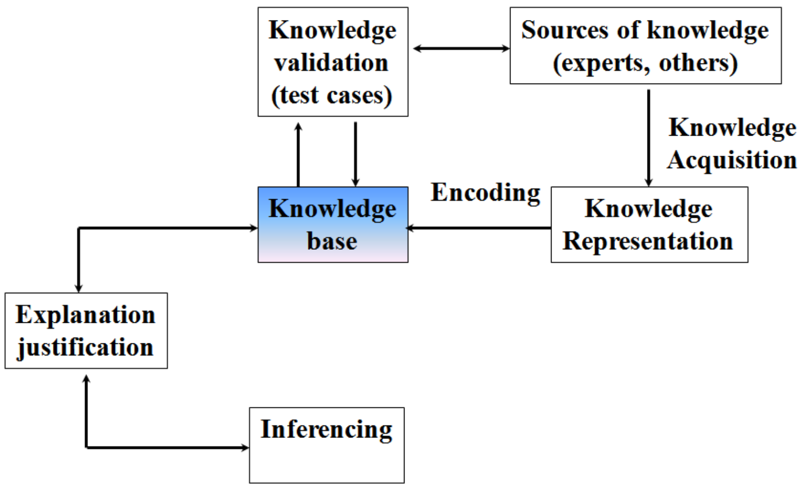
Knowledge Engineering(KR)

- Art of bringing the principles and tools of AI research to bear on difficult applications problems requiring experts' knowledge for their solutions.

- Technical issues of acquiring, representing and using knowledge appropriately to construct and explain lines-of-reasoning.

- Art of building complex computer programs that represent and reason with knowledge of the world .

Knowledge Engineering Process Activities:


Knowledge validation (test cases)
Sources of knowledge (experts, others)
Knowledge Acquisition
Encoding
Knowledge base
Knowledge Representation
Explanation justification
Inferencing

- *Narrow perspective:* knowledge engineering deals with knowledge acquisition, representation, validation, inferencing, explanation and maintenance
- *Wide perspective:* KE describes the *entire process* of developing and maintaining AI systems

-----------End of unit2-----------

UNIT-5

- Introduction to Expert System
- Characteristics features of Expert System
- Applications of Expert System
- Importance of Expert System

Introduction to Expert System:

A Knowledge-based expert system use human knowledge to solve problems that normally would require human intelligence. Expert systems are designed to carry the intelligence and information found in the intellect of experts and provide this knowledge to other members of the organization for problem solving purposes.

With the growing importance of human resource management and increasing size of the organizations, maintenance of employee related data and generating appropriate reports are the crucial aspects of any organization. Therefore more and more organizations are adopting computer based human resource management systems (HRMS).

Basic Meaning of Expert System:

Expert system is one of the areas of artificial intelligence. An expert system also known as knowledge based system is a computer program that contains the knowledge and analytical skills of one or more human experts in a specific problem domain.

Expert system is a computer program that simulates the judgment and behavior of a human that has expert knowledge and experience in a particular field. It contains a knowledge base containing accumulated experience and a set of rules.

The goal of the design of the expert system is to capture the knowledge of a human expert relative to some specific domain and code this in a computer in such a way that the knowledge of the expert is available to a less experienced user.

Expert system provides high quality experience, domain specific knowledge; apply heuristics, forward or backward reasoning, uncertainty and explanation capability. Rule based expert system contains knowledge base, Inference engine, knowledge acquisition, explanation facility and user interface. For knowledge representation techniques, forward and backward chaining rules are used.

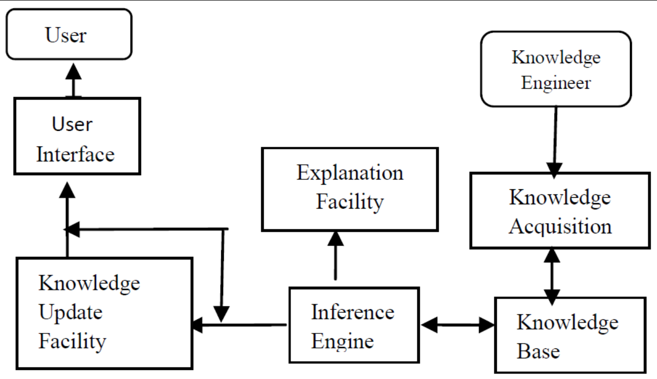
Expert systems are designed to emulate an expert in a specialized knowledge domain such as medicine or any other area of knowledge where there is a shortage of expert knowledge. The knowledge base elicited from the expert by a trained knowledge engineer using various methods can include methodical interviews and the repertory grid technique. Often the expert knowledge area is "fuzzy" in nature and contains a great deal of procedural knowledge, so the knowledge engineer must be an expert in the process of knowledge elicitation.

Characteristics of an Expert System:

1) The most important ingredient in any expert system is the knowledge.
2) In expert systems, knowledge is separated from its processing i.e. the knowledge base and the inference engine are split up.
3) Expert system contains a knowledge base having accumulated experience and a set of rules for applying the knowledge base to each particular situation that is described to the program.
4) Expert system provides the high-quality performance which solves difficult programs in a domain as good as or better than human experts.
5) Expert System possesses vast quantities of domain specific knowledge to the minute details.
6) Expert systems apply heuristics to guide the reasoning and thus reduce the search area for a solution.
7) A unique feature of an expert system is its explanation capability. It enables the expert system to review its own reasoning and explain its decisions.
8) Expert systems employ symbolic reasoning when solving a problem. Symbols are used to represent different types of knowledge such as facts, concepts and rules.
9) Expert system can advice, modifies, update, expand & deals with uncertain and irrelevant data.

ARCHITECTURE OF AN EXPERT SYSTEM:

Expert system can be built using a piece of development software known as a ‘shell’. The core components of expert systems are the knowledge base and the reasoning engine.



Knowledge Base:

It is a warehouse of the domain specific knowledge captured from the human expert via the knowledge acquisition module. To represent the knowledge production rules, frames, logic, semantic net etc. is used.

Inference Engine:

Inference Engine is a brain of expert system. It uses the control structure (rule interpreter) and provides methodology for reasoning. The major task of inference engine is to trace its way through a forest of rules to arrive at a conclusion. Here two approaches are used i.e. forward chaining and backward chaining.

Knowledge Acquisition:

Knowledge acquisition is the accumulation, transfer and transformation of problem-solving expertise from experts and/or documented knowledge sources to a computer program for constructing or expanding the knowledge base. For knowledge acquisition, techniques used are protocol analysis, interviews, and observation.

Explanation Facility:

It is a subsystem that explains the system's actions. Here user would like to ask the basic questions why and how and serves as a tutor in sharing the system's knowledge with the user.

User interface:

It provides facilities such as menus, graphical interface etc. to make the dialog user friendly. Responsibility of user interface is to convert the rules from its internal representation (which user may not understand) to the user understandable form.

To build the expert system is known as Knowledge Engineering. The expert and knowledge engineer should anticipate user's need while designing an expert system.

<u>Application of Expert System:</u>

Expert system can be applicable in many areas. Some more are-

1) Different type of medical diagnosis.
2) Diagnosis of complex electronic and electromechanical systems.
3) Diagnosis of diesel electric location systems.
4) Diagnosis of software development projects.

5) Planning experiments in biology, chemistry and molecular genetics.
6) Forecasting crop damage.
7) Identification of chemical compound structures and chemical compounds.
8) Location of faults in computer and communications system.
9) Evaluation of loan applicants for lending institution.
10) Analysis of structural systems for design or as a result of earth quake.

Early application area of expert systems -

1) DENDRAL(1960): First Expert System which recognizes the structures of chemical compounds.
2) MYCIN: which diagnoses bacterial blood infections.
3) PUFF: which diagnose pulmonary disorders.
4) SCHOLAR: which gives Geography Tutorials.
5) SOPHIE: which teaches how to detect breakdown in electrical circuits.
6) SHDRLU: which manipulates polygons in a restricted environment.
7) Waterman's Poker Player: Game playing systems.
8) AM: Automatic theorem Provers.
9) NOAH and MOLGEN: Planning systems.
10) Holland: Prediction systems such as Political Forecasting Systems.

Importance of Expert System:

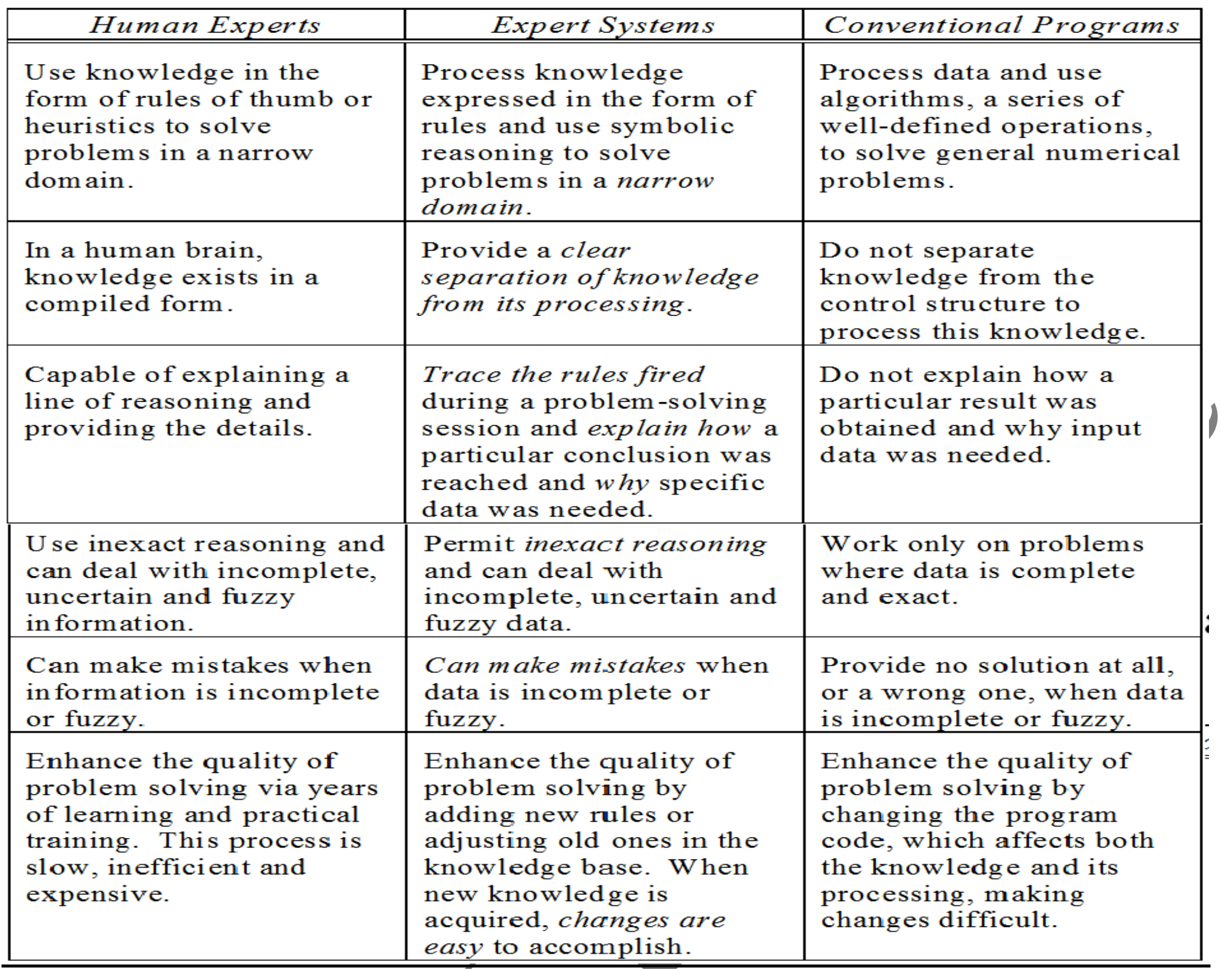
The value of expert systems was well established by early 1980s. A number of successful applications had been completed by then and they proved to be cost effective. A example which illustrates this point well is the diagnostic system developed by the Campbell Soup Company.

Campbell Soup uses large cookers to cook soups and other canned products at eight plants located throughout the company. For the maintenance of cooker fault, only a single human expert was to diagnosis. He had to flying all site when necessary. Since this individual will retire after some year then company decided computer based expert system to diagnosis cooker fault problem.

After some month, Texas Instruments, developed an Expert System used to provide training to new maintenance personal.

Thus computer expert system never retire, so its very great importance in many areas.

Comparison of expert systems with conventional systems and human experts :



Human Experts	*Expert Systems*	*Conventional Programs*
Use knowledge in the form of rules of thumb or heuristics to solve problems in a narrow domain.	Process knowledge expressed in the form of rules and use symbolic reasoning to solve problems in a *narrow domain*.	Process data and use algorithms, a series of well-defined operations, to solve general numerical problems.
In a human brain, knowledge exists in a compiled form.	Provide a *clear separation of knowledge from its processing*.	Do not separate knowledge from the control structure to process this knowledge.
Capable of explaining a line of reasoning and providing the details.	*Trace the rules fired* during a problem-solving session and *explain how* a particular conclusion was reached and *why* specific data was needed.	Do not explain how a particular result was obtained and why input data was needed.
Use inexact reasoning and can deal with incomplete, uncertain and fuzzy information.	Permit *inexact reasoning* and can deal with incomplete, uncertain and fuzzy data.	Work only on problems where data is complete and exact.
Can make mistakes when information is incomplete or fuzzy.	*Can make mistakes* when data is incomplete or fuzzy.	Provide no solution at all, or a wrong one, when data is incomplete or fuzzy.
Enhance the quality of problem solving via years of learning and practical training. This process is slow, inefficient and expensive.	Enhance the quality of problem solving by adding new rules or adjusting old ones in the knowledge base. When new knowledge is acquired, *changes are easy* to accomplish.	Enhance the quality of problem solving by changing the program code, which affects both the knowledge and its processing, making changes difficult.

UNIT-3

LISP AND AI PROGRAMMING LANGUAGES :

- Introduction to LISP :
- Syntax and Numeric Functions
- Basic List Manipulation Functions in LISP
- Functions, Predicates, and Conditionals
- Input, Output, and Local Variables
- Iteration and Recursion

- Property List and arrays

PROGLOG and Other AI Programming Languages

Brief overview of LISP:

History: LISP is one of the oldest computer programming languages. It was invented by "John McCarthy" during the late 1950s after FORTRAN.

Different Flavors of LISP: Several dialects of LISP are FRANZLISP, INTERLISP, MACLISP, QLISP, SCHEME and COMMON LISP.

Important Features of LISP: It is suited for AI programming because of its ability to process symbolic information effectively. LISP has simple syntax with little or no data typing and dynamic memory management.

Running the LISP program: LISP program run on an interpreter or as compiled code. The interpreter examines source programs in a repeated loop (read-evaluate-print). This loop reads LISP program code, evaluate it, and print the value returned by the program. LISP interpreter read code using -> prompt.

Ex:
```
-> (+ 5 6 7)
19
-> Input here next lisp instruction.
```

Basic Building of LISP: (atom, list and string)

Atom, list and string are the valid object in LISP, also called " Symbolic Expressions or S-expressions".

Atom: A number, continuous character (include digits, alphabets, special characters).

Ex: Numbers: 2013, 21989
Continuous Characters: lokesh, rathore, this-is-a-atom, ab123
Rules: Not allowed characters : space, ( ), ', digit at first place

List: A sequence of atoms and/or other lists encloses within parentheses. Elements of list are called top element of list.

Ex: (this is a list) It contain 4 atom/ 4 top element-this, is, a, list
Ex: (a (a b) c (def)) It has four top element-a, (a b), c, (def)
(a b) is a sublist has two top element a and b
(def) is also a sub list has only one top element- def.
Ex: (mon tue wed thus fri sat sun)
Ex: () empty list
Rules: Each elements are separated by space. (closed by).
Invalid list: (a,b)a b(

String: A group of characters enclosed in double quotation marks.

Ex: "this is a string"

"a b c d e fgh #$%@!"
Invalid: this is not string, "welcome to LISP

Special Value: There are three types under special values-

1) Constant: any number like 12, 199
2) t: for true and
3) nil: for false or () empty list. nil is a unique object in LISP which is list and atom.

When any one is given at prompt then LISP interpreter return as it is form.

```
-> 12
12
-> t
T
-> nil
Nil
```

Function call: In LISP a number of functions are available to perform any operation. Every operation is called function and applying content (atom, list, string) called argument. Everything written in list form and function-name written as prefix.

Syntax for function call : (function-name arg1 arg2 -----)

When a function is called, the arguments are first evaluated from left to right and function is executed using the evaluated argument values.

Syntax and Numeric function : (+,-,*,/) Arithmetic function operate only passing numeric arguments (integer or real value).

Syntax: -> (op data-1 data-2 ------ data-n)

Op means (+ or – or * or /)

1) + (plus): Add zero or more given arguments.
Ex:
-> (+) It has zero argument so gives value 0
-> (+ 2 4 6) gives 12
-> (+ 2 4 (+ 6 8)) internally (+ 2 4 14) then gives 20

2) * (product): Multiply zero or more given arguments.
Ex:
-> (*) It has zero argument so gives value 1
-> (* 2 4) gives 8
-> (* 2 4 6) gives 48

3) – (minus): Subtract two given arguments.
Ex:
-> (-) It has zero argument so gives error

-> (- 4 1) gives 3

4) / (Divide): Divide two given arguments.

-> (/) It has zero argument so gives error

-> (/ 9 3) gives 3

Prevent evaluation of List or atom : We precede atom or list with ' (single quotation mark).

Ex:

```
-> 'man
Man
-> '(+ 3 5)
(+ 3 5)            Not evaluate due to preceding (')
->(+ 3 5)
8   Evaluated due to absent of preceding (') mark.
```

Declaring Variable (setq): Used to hold data (atom/ list) for further use and return last bound data. Unbound variable gives error.

Syntax: setq(variable-name bound-data)

It need two argument-First must be variable name and second may be any atom or list with or without (') preceding mark. Second Data have (') assign to variable without evaluate.

Ex1:

```
-> (setq x 10)
10
->x
10
```

Single value evaluated and assign to variable x.

Ex2:

```
-> (setq x (+3 8)
11
->x
11
```

First (+3 8) evaluated and result assign to variable x.

Ex3:

```
-> (setq x '(+3 8))
(+ 3 8)
->x
(+ 3 8)
```

Here (+3 8) does not evaluated due to preceding (') and assign completely to variable x.

Basic List Manipulation function: (car, cdr, cons and list):

When we need not to evaluate atom or list of given list data. Only extract need of data from list then we use following set of manipulation function. Arguments for such function must be precede with (') mark.

1) Car:

This function need only <u>one argument</u> in list form and as the result returns <u>top element</u> of given list.

Example-
```
-> (car  '(a b c))
Output: a
```
a is the top of list (a b c)

2) Cdr:

This function need only <u>one argument</u> in list form and as the result returns a list <u>except first top</u> element of list.

Example-
```
-> (cdr  '(a b c))
Output: (b c)
```
Element a removes from list (a b c)

3) Cons:

This function need only <u>two arguments,</u> an element (atom or list) and a list, as the result returns a list with the element <u>inserted at the beginning</u> of list.

Example-
```
-> (cons  'a  '(b c))
Output: (a b c)
```
Element a add at top of list (b c)

4) List:

This function is used to <u>create a list</u> for given a number of elements as arguments. It returns a list.

Example-
```
-> (list  'a  '(b c) 'd)
Output: (a (b c) d)
```
Create a list contain a,(b c) and d.

5) Append:

Merges two or more lists into a single list.

Example-
```
-> (append '(a) '(b c))
Output: (a b c)
```
All argument must be list.

6) Last:

Returns a list containing the last element.

Example-
```
-> (last '(a b c d))
Output: (d)
```

7) Member: (searching function)

Returns remainder of second argument list starting with element matching first argument.

Example-
```
-> (member 'b '(a b c d))
Output: (b c d)
```

8) Reverse:

Returns list with top elements in reverse order.

Example-
```
-> (reverse '(a (b c) d))
Output: (d (b c) a)
```

Exercise: Find output

```
1)  (+)
2)  (*)
3)  (-)
4)  (/)
5)  (+ 5 6 8)
6)  (-  8 4)
7)  (-  8 4 2)
8)  (* 3 6 5)
9)  (/ 8 2)
10) (/ 8 4 2)
11)(+ (* (/ 9 5) 50) 32)
12) (setq x 12)
13)(setq x (+ 6 5)
14)(setq x '(a b c))
15)x
16)'x
17)(cons '(* 2 3)  '(1))
18)(cons  (* 2 3)  '(1))
19) (car (cdr '(a b c d)))
20)  (cdr car '((a b) c d ))
21) (cons 'one '(two three))
22)(cons (car '(a b c)(cdr '(a b c)) ))
23)(list '(a b) 'c 'd)
24)  (append '(a (b c))  '(d e))
25)  (append  '(a) '(b c) '(d))
26)  (last '(a b (c d) (e)))
27)  (member '(d) '(a (d) e f))
28)  (reverse '(a b (c d) e))
```

Defining Functions:

It is possible to user define functions in LISP. i.e. user can define own function according to his requirement. In LISP this task is performed using predefine defun function.

Syntax:

->(defun udfn(p_1 p_2 ....p_n) body)

Here-

defun= function maker

ufn= user define function name

p= list of parameter which receive values

body= single line of logic in list form.

Ex1: function for average of three numbers.

```
->  (defun averagethree(n1 n2 n3) (/ (+ n1 n2 n3) 3) )
Output:AVERAGETHREE
```

Now we call this function as usual-

-> (averagethree 10 20 30)
Output: 20
->

Ex2: Convert centigrade to Fahrenheit.(f=c*9/5+32)
->(defun ctof(c) (+ (/ (* c 9) 5) 32))
Output:ctof
-> (ctof 0)
Output: 32

Predicate functions:

Predicates are function that test their arguments for some specific condition and return true (t) or false (nil). (Except for the predicate-member)
The most common predicates are:

1) atom: It needs one argument, if argument is an atom then returns t(true) otherwise nil(false).
   Ex:
   -> (atom 'a) output: t
   -> (atom '(a)) output: nil
2) listp: It needs one argument, if argument is a list then returns t(true) otherwise nil(false).
   Ex:
   -> (listp '(a)) output: t
   -> (list 'a) output: nil
3) numberp: It needs one argument, if argument is a number then returns t(true) otherwise nil(false).
   Ex:
   -> (numberp 123) output: t
   -> (numberp 12ab)) output: nil
4) evenp: It needs one argument, if argument is a even number then returns t(true) otherwise nil(false).
   Ex:
   -> (evenp 8) output: t
   -> (evenp 9) output: nil
5) oddp: It needs one argument, if argument is a odd number then returns t(true) otherwise nil(false).
   Ex:
   -> (oddp 9) output: t
   -> (oddp 8) output: nil
6) null: It needs one argument, if argument is a nil or empty list () then returns t(true) otherwise nil(false).
   Ex:
   -> (null nil) output: t
   -> (null ()) output: t
   -> (null (8)) output: nil
7) zerop: It needs one argument, if argument is a zero value then returns t(true) otherwise nil(false).

Ex:
-> (zerop 0) output: t
-> (zerop 8) output: nil

8) equal: It needs two arguments, if both arguments have same value then returns t(true) otherwise nil(false).
Ex:
-> (equal 'a (car '(a b)) output: t
-> (equal 'a (cdr '(a b)) output: nil

9) greaterp (>): It needs one or more arguments. If it has only one argument then returns t(true). If more than one arguments are used than it returns t(true) if the arguments, from left to right, are successively larger, otherwise nil(false) is returned.
Ex:
-> (greaterp 2) output: t
-> (greaterp 2 4 8) output: t
-> (greaterp 2 4 4) output: nil

10)lessp (<):It needs one or more arguments. If it has only one argument then returns t(true). If more than one arguments are used than it returns t(true) if the arguments, from left to right, are successively smaller, otherwise nil(false) is returned.
Ex:
-> (lessp 2) output: t
-> (lessp 8 4 2) output: t
-> (lessp 8 4 4) output: nil

11)>= : It has similar leaning of greaterp, except they return t(true) if successive elements are also equal.
Ex:
-> (>= 2) output: t
-> (>= 2 4 8) output: t
-> (>= 2 4 4) output: t

12)<= : It has similar leaning of lessp, except they return t(true) if successive elements are also equal.
-> (<= 2) output: t
-> (<= 8 4 2) output: t
-> (<= 8 4 4) output: t

The conditional (cond function):

If we want to perform any action based on given condition then it is possible using cond function in LISP. It is like the if…then…else construct.

Syntax :
(cond (<test$_1$> <action$_1$>) (<test$_1$> <action$_1$>)………(<test$_k$> <action$_k$>))

Features of cond:
- Cond function can test one or more conditions.
- If <test1> evaluates t(true) then <action$_1$> portion is evaluated, its value is returned and the remaining condition parts are skipped.
- If <test$_1$> evaluates to nil, control passes to the second part without evaluating <action$_1$> and the procedure is repeated.

- If all tests evaluates to nil, cond returns nil.

Ex: Test ginen number is even or odd.

```
-> (defun evenodd(n)(cond ( (evenp n) 'Even) (t 'odd)))
EVENODD
-> (evenodd 12)
Even
->(evenodd 5)
Odd.
```

Logical functions: (and, or, not)

Logical functions may also be used for flow of control. Not takes one argument whereas and, or both take any number of arguments and are evaluated from left to right.

- Not: if the argument evaluates to nil It returns t(true). if its argument evaluates to non-nil, it returns nil.
- And: If all arguments evaluate to non-nil, the value of the last argument is returned, otherwise nil is returned.
- Or: Arguments are evaluated until one evaluates to non-nil, in which case it returns the argument value, otherwise it returns nil.

Ex of not predicate:

```
-> (setq x '(a b c))
   (A B C)
-> (not (atom x))
   T
-> (not (listp x))
   NIL
```

Example of or predicate:

```
-> (or (member 'e x) (member 'b x))
   (B C)
-> (or (equal 'c(car x)) (equal 'b(car x)))
   NIL
```

Example of and predicate:

```
-> (and (listp x) (equal 'c (caddr x)))
   C
```

Example of and or not:

```
-> (or (and (atom x) (equal 'a x)) (and (not (atom x)) (atom (car x))))
   T
```

Input function: (read)

Read takes no arguments. When read appears in a procedure, processing halts until a single s-expression is entered from the keyboard.

For example:

```
-> (+  5 (read))
  6
  11
```

Here read statement wait for an input from the keyboard. If we entered 6, read returns this value.

Assign to variable:

```
->(setq x (read))
  8   (enter)
  8 (output, 8 has assigned to x)
-> x (enter)
  8 (value of x)
```

Output function (print, prinl, princ, terpri and format):

Each one need only one argument which has to print on output screen. There is some minor differences.

Print function: It prints the argument as it is received, and then return argument so that it can print as well as pass to other function. It print argument from beginning of new line and is followed by a space.

For example:

```
->(print '(hello))
   Hello (by print and return to interpreter)
   Hello (by interpreter)
->(print "hello")
   "hello" (by print and return to interpreter)
   "hello" (by interpreter)
```

Prinl function: Same as print except that new line and a space are not provided as well as does not return its argument to another function.

For example:

```
-> ((prinl '(hello)) (print '(hello)))
    (hello)(hello)
->
```

Princ function: it is same as prinl except that It does not print unwanted quotation marks like "" with argument. It return argument after print.

```
-> (princ "hello")
   Hello "HELLO"
->
```

terpri function: it takes no argument. It introduces a new-line wherever it appears and then returns nil.

For example: to compute the area of a circle

```
-> (defun circle-area ()
        (terpri)
        (princ "enter the radius")
        (setq rad (read))
        (princ "The area of the circle is: ")
        (princ (* 3.1416 rad rad))
        (terpri))
  CIRCLE-AREA
-> (circle-area)
    Enter the radius 4
   The area of the circle is: 50.2656
```

In this program princ function print multiple items on the same line, so we use terpri to introduce a new-line sequence.

format function: Using this function we can print output with variables value and newline characters, i.e. we can set format according to our needs.

Syntax: (format <destination> <string> arg1, arg2, …)

Here-

Destination=place of output. monitor(t) or some other external file.
String= Desired output that Intermixed with format directives.
Arg1,arg2,..= Arguments that will print in place of directive appear in the string.

List of some important directives:

~D : for integer argument
~5D: an integer field of width 5
~F : for floating point argument
~C: for character type argument
~%: for new-line

Ex:

```
-> (format t “Circle radius = ~2F~% Circle area = ~3F” x y)
   "Circle radius = 3.0
    Circle area = 9.42 "
```

Constructing local variable:

Local variable will be preference over global variable. Local variable can be created as-

1) Paramètres of function definition.
2) Variable declaration using setq function within user define function body.
3) Creating local variable using let and prog function.

Ex:

```
->(setq y '(a b c))

  (a b c)

->(setq x '(d e f)

  (d e f)
```

Here x and y are global variable for any function body’s x and y.

```
->(defun local-var(x) (setq  y (cons x y)))

 Local-var
```

Here (x) and setq y are local whereas y within (cons) is global.

```
-> (local-var  6)

(6 a b c)

-> x
```

(d e f)

->y

(a b c)

It's clear that parameter and variable of any function are called local variable.

Creating local variable using let: Using let function we can create more than one variables and used in any expression.

Syntax- (let ((var$_1$ val$_1$) (var$_2$ val$_2$)…) <s-expressions>)

Values are assigned to associated variables. Local variables can be used in expression and it return evaluated value of expression.

Ex:

```
-> (let ((x 2)(y 4)) (+ x y))
6
->(let ((x 'a) (y 'b) (z 'c)) (cons x (cons y (list z))))
(A B C)
```

Creating local variable using prog function: The prog function is similar to let function but it may be any number of s-expressions.

Syntax- (prog ((var$_1$ val$_1$) (var$_2$ val$_2$)…) <s-expressions1> <s-expressions2>……)

s-expressions are execute in sequence unless it encounters return function.

Ex: -> prog ((x 4)(y 5)) (cond ((equal x y) (return 'equal)) (t (return 'not-equal)))

Not-equal

Iteration and recursion: Iteration is one form of loop whereas recursion is a special kind of user function.

Iteration: When a set of statements are executed more than one times using a loop until a given condition satisfied called iteration. Every times statements do not relocated.

Syntax: using do function we can create an iteration.

```
        (do (<var1 val1> <var-update1>)
            (<var2 val2> <var-update2>)
                .
                .
             (<test> <return-value>)
             (<s-expression>)
        )
```

Here:

- <val> are initial values that bounds to <var> variables parallel first time then <test> is evaluated.
- If<test> gives nil value then all <var> are parallel updated with <var-update> value. This process continued until test get non-nil(true).
- If <test> get non-nil any time then iteration process stops and it return <return-value>.
- <s-expression> is optional. If present, executed with each iteration

Ex: factorial function

```
->(defun factorial(n)
   (   do
         (i n (- i 1))
         (f n (* f (- i 1))
         ((equal 1 i) f)
   )
  )
```

Factorial

```
-> (factorial 4)
24
```

Initially value 4 assign to i and f, since i is not 1 so test give nil and second times i update by 3(- i 1) and f updated by 12(* f (- i 1). After some iteration i updated by 1 then test give non-nil and iteration will terminate with f value.

Recursion: It is a special kind of function in which all statements of function are executed more than one times until any condition satisfied, but it will called recursion if every times all statements relocated and previous will remains in memory.
Simply when a function calls itself until a given condition called recursion.
Ex: factorial function

```
->(defun factorial(n)
    (cond (  (equal n 1)  1)
           (t (* n  (factorial(- n 1))))))
```

Factorial

```
-> (factorial 4)
24
```

Initially value 4 assign to n, since n is not 1 so this function again call with one less value. When it get 1 then recursively recalculate.

Property lists:

The most useful unique features of LISP as an AI language is the ability to assign properties to atoms. There are following property list functions used to assign, retrieve, replace and remove properties to an atom.

1) putprop
2) remprop
3) get
4) setf

1) putprop: It is used to assign a value with properties to an atom and returns assigned value.
   Syntax: (putprop atom value property)
   Ex:
   ```
   ->(putprop 'car 2013 'year)
   2013
    ->(putprop 'car 'red 'color)
   red
   ```
2) remprop: It is used to remove any property from atom and return value.
   Syntax: (remprop atom property)
   Ex:
   ```
   -> (remprop 'car 'year)
   2013
   ```
3) get: It is used to retrieve value of property for given atom.
   Syntax: (get atom property)
   Ex:
   ```
   -> (get 'car 'color)
   Red
   ```
4) setf: It is used to reset property of any atom by new value.
   syntax: (setf (get atom property) new-value)
   Ex:
   ```
   -> (setf (get 'car 'color) 'blue)
   blue
   ```

Array:

Collection of items that stores continuously called array. To create and access array we use following three function.

1) make-array: It is used to create a new blank array of given size in memory.
   Syntax:
   ```
   (setf arr-name (make-array '(size)))
   ```
   Ex:
   ```
   ->(setf myarray(make-array '(5)))
   Output: #A(nil,nil,nil,nil,nil)
   ```

2) aref: It is used to access any specific array item.
   Syntax:
   ```
   (aref arr-name index)
   Index= 0 to size-1
   ```

```
Ex: Assign values to array
-> (setf (aref myarray 0) 10)
Output: 10
-> (setf (aref myarray 4) 50)
Output: 50
Ex: Read array values
-> (aref myarray 0)
Output: 10
-> (aref myarray 4)
Output: 50
```

Miscellaneous functions:

1) **mapcar function:** We can add each element of given list by a number using mapcar function.

```
->(mapcar '1+ '(5 10  15  20  25))
Output: (6 11 16 21 26)
```

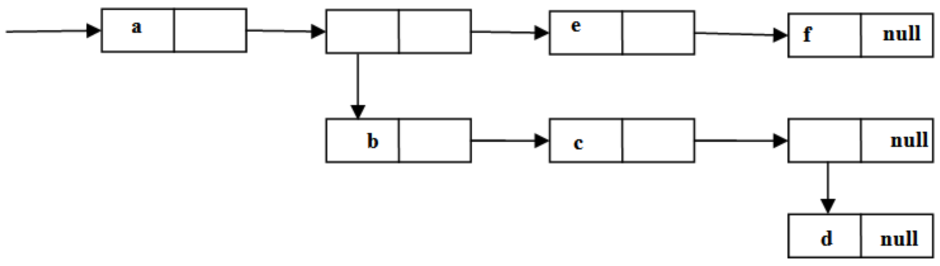
2) **lambda function:** using this function we can use one function body more than one times for single function call.

Ex: find cube of each number of given list.

```
-> (defun cube-list(lst)
      (mapcar #'(lambda(x) (* x x x)) lst)     )
Output: cube-list
-> (cube-list (1 2 3 4))
Output:( 1 8 27 64)
```

When a function is called by another function, it should be preceded by #' indicate following item is a function.

Internal storage: Lists are made through the use of linked cell structures in memory. For example: (a (b c (d)) e f) can be represented as-



PROLOG and other AI programming languages

History:

PROLOG(PROgramming in LOGic) was invented by "Alain Colmerauer" and his associates at the university of Marseilles during the early 1970s.

Resolution process of PROLOG:

It uses the syntax of predicate logic to perform symbolic, logical computations. Programming in PROLOG is accomplished by creating a database of facts and rules about objects, their properties, and their relationships to other objects. Queries can then be posed about the objects and valid conclusions will be determined and returned by the program. Responses to user queries are determined through a form of inferencing control known as resolution.

Facts representation in PROLOG:

Facts are declared with predicates and constants written in lowercase letters. The arguments of predicates are enclosed in parentheses and separated with commas(,).

Ex: some facts about family relationship can be written in PROLOG as-

1) sister(sue, bill)   - sue is sister of bill.
2) parent(ann, sam)   - ann is parent of sam.
3) parent(joe,ann)   - joe is parent of ann.
4) male(joe)   - joe is a male.
5) female(ann)   - ann is a female.

Here sister, parent, male and female are predicate, whereas sue, bill, ann and sam are arguments.

Rules in PROLOG:

Rules are composed of a condition or "if" part and a conclusion or "then" part separated by a symbol (:-). Rules are used to represent general relations which hold when all of the conditions in the if part are satisfied. Rules may contain variables(uppercase).

Ex: PROLOG rules for grand father, we write

Grandfather(X,Z):- parent(X,Y), parent(Y,Z), male(X)

This rule has following meaning-
For all X, Y and Z,
X is the grand father of Z
If X is the parent of Y, and Y is the parent of Z and X is the male.

Query in PROLOG:

When a database of facts and rules such as that above have given, then we make queries by typing after (?) symbol such as-

```
?- parent(X, sam)            -who is parent of sam
X=ann                       -Answer
?- male(joe)
Yes
?- grandfather(X,Y)
X=hoe, Y=sam
?- female(joe)
no
```

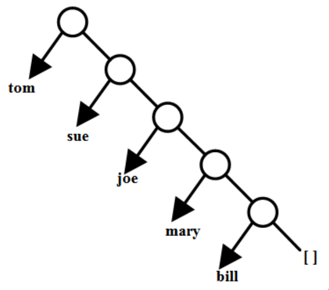
PROLOG searches database for submitted query. If a proper match found between predicates of query and database then returns response with proper result or failure occurs.

Lists in PROLOG: Each item must be separated by commas(,) and enclosed in square brackets[ ].

Ex: [tom, sue,joe,mary,bill] is the list of student in PROLOG.

A list may be empty or non empty. A non empty list has head and tail. Tom is the head and remaining sublist [sue,joe,mary,bill] is tail.

Binary tree representation of PROLOG list:



List may be written as [a,b,c,d]=[a,b|[c,d]]=[a,b,c,d|[]].

List manipulate predicates:

A number of list manipulation predicates are also available in PROLOG as- append, member, conc, add, delete and so on.

For example:

?- member(c,[a,b,c,d])

Yes

?- member(b,[a,[b,c]],d])

no

PROLOG has numeric functions, relations and list handling capabilities.

Other programming languages used in AI:

C, object oriented extensions to LISP such as Flavors, and languages like Smalltalk. The language C has been used by some AI programmers because of its popularity and its portability. Although Object oriented languages have been gaining much popularity.

==========End of unit-3===========

UNIT-4
FORMALIZED SYMBOLIC LOGICS: Introduction
Syntax and Semantics for Propositional Logic
Syntax and Semantics for FOPL
Properties of Wffs
Conversion to Clausal Form
Inference Rules
The Resolution Principle
Representations Using Rules.

Introduction:

FOPL (First Order Predicate Logic) is one of the oldest and most important scheme for knowledge representation in logical reasoning and mathematical form. Application of logic as a practical means of representing and manipulating knowledge in a computer was not demonstrated until the early 1960s. But today, FOPL or predicate calculus has one of the most important roles in AI for representing knowledge.
FOPL is important for AI professionals because-

1) Logic offers the formal approach to reasoning.
2) Structure of FOPL permits the accurate representation of natural language reasonably well.
3) To understand AI articles, knowledge of FOPL required because it is commonly used to design AI programs.

In FOPL English like statements are translated into symbolic structures consists using predicates, functions, variable, constant, quantifiers and logical connectives . They form a structure using syntax for FOPL. Once structures of facts or other type of knowledge is created, inference rules are applied on it for new "deduced" structures.
Ex: " All employees of the AI-software company are programmers" can be written in FOPL as –

(∀x)(AI-SOFTWARE-CO-EMPLOYEE(x)→PROGRAMMER(x))

∀x is read as "for all x".

→ is read as "implies" or "then".

AI-SOFTWARE-CO-EMPLOYEE(x) and PROGRAMMER(x) are predicates and read as " if x is an AI Software company employee" , "x is a programmer" respectively.

x is a variable assumed for a person's name. for example if it is known as lokesh is employee of AI software company it means

AI-SOFTWARE-CO-EMPLOYEE(lokesh)

Its conclusion can draw that lokesh is a programmer.

PROGRAMMER(LOKESH)

Thus we conclude that in this way we can translate knowledge in the form of English sentences can be translated into FOPL statements.

Syntax and Semantics for Proposition Logic(PL)

Propositional Logic is a special case of FOPL. Propositions are elementary atomic sentences.

About PL:

- In PL sentences are known as *formulas* or *well-formed formulas.*
- Propositions may be either true or false.
  Ex: Atomic formulas are
  It is raining.
  People live on the moon.
- Compound PL are formed using *connectives not and or if…then and if and only if*.
  Ex: compound formulas are
  It is raining and the wind is blowing.
  If you study hard then you will be passed.
- Capital letters are used for propositions; T and F for true and false.
- Logical connectives used in PL are
  ~ for not or negation
  & for and or conjunction
  V for or or disjunction (inclusive disjunction)
  → for if……then or implication
  ↔ for if and only if or double implication
- ( ), { }are the delimiters for punctuation.
- For example: compound sentences "It is raining and the wind is blowing" can be written in PL as (R & B)
  Here R, B stand for the propositions.
  R for "It is raining"
  B for "the wind is blowing"
  (R V B) means "It is raining" or "the wind is blowing" or both

Syntax:

The syntax of PL is recursively as follows. T and F are formulas. If P and Q are formulas, the following are formulas:

(~P)

(P & Q)

(P V Q)

(P→Q)

(P↔Q)

All formulas are generated from a finite number of the above operations.

Example of a compound formula is : ((P&(~QVR))→(Q→S))

The precedence of connectives from highest to lowest is ~,&,V,→,↔. For example we add parenthesis on following formula

P&~QVR→S↔UVW

We write ((((P&(~Q))VR)→S)↔(UVW))

Semantics/Meaning of PL:

Meaning of formula is just truth values (true or false) i.e. assignment of truth value to formulas. An interpretation of formulas is assignment of truth value to each propositional symbol. The semantic rules of PL are represented as

Rule number	True statement	False statement
1	T	F
2	~f	~t
3	t & t	f & a
4	t V a	a & f
5	a V t	f V f
6	a→t	t→f
7	f→a	t↔f
8	t↔t	f↔t
9	f↔f	

Here - t denote true, f denote false and a denote any statement.

For example: Given a formula ((P & ~Q)→R)VQ.

Let an interpretation I assign true to P, false to Q and false to R.

Now apply above semantics on give formula as

((t & ~f)→f)Vf

Step1 : ((t & t)→f)Vf	rule-2
Step2: (t→f)Vf	rule-3
Step3: f V f	rule-6
Step4: f	rule-5

Thus on the basis of PL semantic rule given PL formula gives f false value.

Properties of statements:

1) Satisfiable : statements are true for given interpretation.
2) Contradiction (Unsatisfiable): statements are not true for given interpretation.
3) Valid: statements are true for every interpretation. Valid statements are also called tautologies.
4) Equivalence: Two statements have same truth value under every interpretation.
5) Logical consequence: P is a logical consequence of (P & Q) since any interpretation for (P & Q) is true, P is also true.

Some important laws of PL:

Idempotency	PVP=P P&P=P
Associativity	(PVQ)VR=PV(QVR) (P&Q)&R=P&(Q&R)
Commutativity	PVQ=Q V P P & Q=Q & P P↔Q = Q ↔ P
Distributivity	P&(QVR)= (P&Q)V(P&R) PV (Q&R)= (PVQ) & (PVR)
De Morgan's laws	~(PVQ)=~P & ~Q ~(P & Q)= ~PV~Q
Conditional elimination	P→Q = ~P V Q
Bi-conditional elimination	P↔Q=(P→Q)&(Q→P)

Determine equivalence of two sentences: It is done by truth table. For example, we can show using truth table that (P→Q)=(~PVQ) and (P↔Q)=(P→Q)&(Q→P) are constructed equivalence or not.

P	Q	~P	(P→Q)	(~PVQ)	(P→Q)	(Q→P)	(P→Q)&(Q→P)	(P↔Q)
t	t	F	t	t	t	t	t	t
t	f	F	f	f	f	t	f	f
f	t	T	t	t	t	f	f	f
f	f	T	t	t	t	t	t	t
Input values			Both column has equal value				Both column has equal value	

Inference rules: Inference rules of PL provide the means to perform logical proofs or deductions. Some inference rules are

1) Modus ponens: From P and P→Q infer Q.
2) Chain rule: From P→Q, and Q→R, infer P→R.
3) Substitution: P V P' is valid; Q V Q' is also valid.
4) Simplification: From P & Q infer P.
5) Conjunction: From P and from Q, infer P&Q.
6) Transposition: From P→Q, infer ~Q→~P.

Limitation of Proportional Logic(PL):

- It does not represent expressiveness and accurate requirement of any scheme.
- PL does not permit us to make generalized statements about classes of similar objects.
- Ex: "All student of CS must take pascal". "John is a CS major". PL is unable to conclude that " John must take pascal", since second statement does not occur as a part of first one.

Syntax and Semantics for FOPL:

FOPL was developed by logicians to extend the expressiveness of PL. It is generalization of PL that permits reasoning about world objects as relational entities as well classes or subclasses of objects. This generalization comes from the introduction of predicates in place of propositions, use of functions and use of variables together with variable quantifiers.

Syntax of FOPL:

The symbols and rules of combination permitted in FOPL are defined as follows.

Connectives. Five symbols are ~(not or negation), &(and or conjunction), ⋁(o or disjunction), →(implication), ↔(equivalence or iff)

Quantifiers. Two symbols are ∃(existential quantification-for some) and ∀ (universal quantification-for all).

Constants. Fixed value terms belong to a given domain D and denoted by numbers(23), words(lokesh) and small letters near the beginning of alphabet(a,b,c).

Variables. terms that can assume different value over a given domain D and denoted by words or small letters near the end of alphabets(x,y,z).

Functions. It denotes relations defined on a domain D. Symbols f, g, h and wors like father-of, age-of represents function.

Predicates. Predicates symbols denotes relations or functional mapping from the elements of domain D to the values true or false. Capital letters like P, Q, R etc used to represent predicates.

Terminology of FOPL: Constants, variable and function are referred to as <u>terms</u>, and predicates are referred to as <u>atomic formulas or atom</u>. Atom or its negation referred as <u>literal</u>. () and {} are the used for punctuations.

Example of FOPL representation:

Given-

E1: All employees earning $1400 or more per year pay taxes.

E2: Some employees are sick today.

E3: No employee earns more than the president.

For FOPL we must define abbreviations for predicates and functions.

E(x) for x is an employee.

P(x) for x is president.

i(x) for for the income of x (lower case denotes a function)

GE(u,v) for u is greater than or equal to v.

S(x) for x is sick today.

T(x) for x pays taxes.

On the basis of above abbreviations, we can represent E1, E2 and E3 as

E1': ∀x ((E(x)& GE(i(x),1400))→T(x))

E2': ∃y (E(y)→S(y))

E3': ∀xy ((E(x) & P(y))→~GE(i(x),i(y)))

E1 read as- "for all x if x is employee and income of x is greater than or equal to 1400 then x pays taxes"

E2 read as-" for some y if y is employee then y is sick today"

E3 read as-"for all x and for all y, if x is employee and y is president then income of x should not greater than or equal to income of y."

The expression E1',E2' and E3' are known as well-formed formulas or wffs(woofs).

Semantics of FOPL:

When an assignment of values is given to each term and to each predicate symbol in a wff, we say an interpretation is given to the wff. Literals are always true or false value under an interpretation. The value of any given wff can be determined by referring to a truth table, similar to PL.

Rule number	True statement	False statement
1	T	F
2	~f	~t
3	t & t	f & a
4	t V a	a & f
5	a V t	f V f
6	a→t	t→f
7	f→a	t↔f
8	t↔t	f↔t
9	f↔f	

If the truth values for two different wffs are same under every interpretation, they are said to be equivalent. A predicate or wffs has no variables is called a ground atom.

The predicate P(x) in ∀x P(x), is true only if it is true for every value of x in the domain.

The predicate P(x) in ∃x P(x), is true only if it is true for at least one value of x in the domain.

Properties of wffs (properties of FOPL):

All properties of PL(propositional logic) are exists as well as some more additional properties are included due to ∀(for all) and ∃(for some). Some important laws of PL:

~(~F)=F	Double negation
FVF=F F&F=F	Idempotency
(FVG)VH=FV(GVH) (F&G)&H=F&(G&H)	Associativity
FVG=G V F F & G=G & F F↔G = G ↔ F	Commutativity
F&(GVH)= (F&G)V(F&H) FV (G&H)= (FVG) & (FVH)	Distributivity
~(FVG)=~F & ~G ~(F & G)= ~FV~G	De Morgan's laws
F→G = ~F V G	Conditional elimination
F↔G=(F→G)&(G→F)	Bi-conditional elimination
F→G=~FVG	
F↔G=(~FVG)&(~GVF)	
∀x F[x] V G=∀x(F[x] V G)	
∃x F[x] V G=∃x(F[x] V G)	
∀x F[x] & G=∀x(F[x] & G)	
∃x F[x] & G=∃x(F[x] & G)	
~(∀x) F[x]= ∃x(~ F[x])	
~(∃x) F[x]= ∀x(~ F[x])	
∀x F[x] & ∀x G[x]=∀x(F[x] & G[x])	
∃x F[x] & ∃x G[x]=∃x(F[x] & G[x])	

It has also

1) Satisfiable : statements are true for given interpretation.
2) Contradiction (Unsatisfiable): statements are not true for given interpretation.
3) Valid: statements are true for every interpretation. Valid statements are also called tautologies.
4) Equivalence: Two statements have same truth value under every interpretation.
5) Logical consequence

Conversion to Clausal form:

To transform a sentence into clausal form requires the following steps:

Step1: Eliminate all implication and equivalence symbols.

Step2: Move negation symbols into individual atoms.

Step3: Rename variables if necessary so that all remaining quantifiers have different variable assignments.

Step4: Skolemize by replacing all existentially quantified variables with Skolem (special) functions and eliminate the corresponding quantifiers.

Step5: Move all universal quantifiers to the left of the expression and put the expression on the right into CNF.

Step6: Eliminate all universal quantifiers and conjunctions since they are retained implicitly.

The resulting expressions are clauses and the set of such expressions is said to be in clausal form.

Example: Convert the expression

∃x ∀y(∀z P(**f(x)**,y,z)→(∃u Q(x,u) & ∃v R(y,v))) into clausal form.

Step1: ∃x ∀y(~(∀z) P(f(x),y,z) ∨ (∃u Q(x,u) & (∃v) R(y,v))) Eliminate →

Step2: ∃x ∀y(∃z ~P(f(x),y,z) ∨ (∃u Q(x,u) & (∃v) R(y,v))) Move ~

Step3: Not required for rename variable.

Step4: ∀y(~P(f(a),y,g(y)) ∨ (Q(a,h(y)) & R(y,i(y)))) It is replace x by a, z by g(y), u by h(y), v by i(y) function and eliminate ∃x,∃z,∃u,∃v.

Step5: **∀y((**~P(f(a),y,g(y)) ∨ Q(a,h(y))) & (~P(f(a),y,g(y)) ∨ R(y,i(y)))) move universal quantifier to left.

Step6: ~P(f(a),y,g(y)) ∨ Q(a,h(y))

~P(f(a),y,g(y)) ∨ R(y,i(y))

Finally, We obtain above clausal form.