Unit-I: Problem Solving & Search
Introduction- What Is Intelligence? Foundations Of Artificial Intelligence (Ai). History Of Ai, Structure of Agents.
Problem Solving - Formulating Problems, Problem Types, States and Operators, State Space;
Search Strategies. - Informed Search Strategies- Best First Search, A* Algorithm, Heuristic Functions, Iterative Deepening A*.
Adversarial Search/ Game Playing - Perfect Decision Game, Imperfect Decision Game, Evaluation Function, Alpha-Beta Pruning.

WHAT IS INTELLIGENCE

The capacity to learn and solve problems.

- the ability to solve novel problems (i.e., solve new problems)
- the ability to act rationally (i.e., act based on reason)
- the ability to act like humans

What is involved in intelligence?

- **Ability to interact with the real world**
  – to perceive, understand, and act
  – e.g., speech recognition and understanding and synthesis
  – e.g., image understanding
  – e.g., ability to take actions, have an effect
- **Reasoning and Planning**
  – modelling the external world, given input
  – solving new problems, planning, and making decisions
  – ability to deal with unexpected problems, uncertainties
- **Learning and Adaptation**
  – we are continuously learning and adapting
  – our internal models are always being ―updated
  • e.g., a baby learning to categorize and recognize animals

FOUNDATIONS OF ARTIFICIAL INTELLIGENCE

It is the study of how to make computers do things at which, now, people are better. The term AI is defined by each author in own ways which falls into 4 categories

1. The system that thinks like humans.
2. System that acts like humans.
3. Systems that think rationally.
4. Systems that act rationally.

Definitions Of AI

- **Building systems that think like humans**
  The exciting new effort to make computers think, machines with minds, in the full and literal sense‖ -- Haugeland, 1985
  The automation of activities that we associate with human thinking, … such as decision-making, problem solving, learning-- Bellman, 1978
- **Building systems that act like humans**
  The art of creating machines that perform functions that require intelligence when performed by people‖ -- Kurzweil, 1990
  The study of how to make computers do things at which, now, people are better‖ -- Rich and Knight, 1991

- **Building systems that think rationally**
  ―The study of mental faculties using computational models‖ -- Charniak and McDermott, 1985
  ―The study of the computations that make it possible to perceive, reason, and act‖ -Winston, 1992

- **Building systems that act rationally**
  ―A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes‖ -- Schalkoff, 1990
  ―The branch of computer science that is concerned with the automation of intelligent behavior‖ -- Luger and Stubblefield, 1993

Acting Humanly: The Turing Test Approach

- Test proposed by Alan Turing in 1950
- The computer is asked questions by a human interrogator.

The computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or not. Programming a computer to pass, the computer needs to possess the following capabilities:

- Natural language processing to enable it to communicate successfully in English.
- Knowledge representation to store what it knows or hears
- Automated reasoning to use the stored information to answer questions and to draw new conclusions.
- Machine learning to adapt to new circumstances and to detect and extrapolate patterns. To pass the complete Turing Test, the computer will need
- Computer vision to perceive the objects, and
- Robotics to manipulate objects and move about.

Thinking humanly: The cognitive modelling approach

We need to get inside actual working of the human mind:

(a) Through introspection – trying to capture our own thoughts as they go by.
(b) Through psychological experiments

Allen Newell and Herbert Simon, who developed GPS, the ―General Problem Solver‖ tried to trace the reasoning steps to traces of human subjects solving the same problems. The interdisciplinary field of cognitive science brings together computer models from AI and experimental techniques from psychology to try to construct precise and testable theories of the workings of the human mind

Thinking rationally: The "laws of thought approach"

The Greek philosopher Aristotle was one of the first to attempt to codify ―right thinking that is irrefutable (ie. Impossible to deny) reasoning processes. His syllogism provided patterns for argument structures that always yielded correct conclusions when given correct premises—for example, Socrates is a man; all men are mortal; therefore, Socrates is mortal.‖. These laws of thought were supposed to govern the operation of the mind; their study initiated a field called logic.

Acting rationally: The rational agent approach

An agent is something that acts. Computer agents are not mere programs, but they are expected to have the following attributes also: (a) operating under autonomous control, (b) perceiving their environment, (c) persisting over a prolonged time period, (e) adapting to change. A rational agent is one that acts so as to achieve the best outcome.

HISTORY OF AI

- 1943: early beginnings
  - – McCulloch & Pitts: Boolean circuit model of brain
- 1950: Turing
  - – Turing's "Computing Machinery and Intelligence―
- 1956: birth of AI
  - – Dartmouth meeting: "Artificial Intelligence―name adopted
- 1950s: initial promise
  - – Early AI programs, including
  - – Samuel's checkers program
  - – Newell & Simon's Logic Theorist
- 1955-65: ―great enthusiasm‖
  - – Newell and Simon: GPS, general problem solver
  - – Gelertner: Geometry Theorem Prover
  - – McCarthy: invention of LISP
- 1966—73: Reality dawns
  - – Realization that many AI problems are intractable
  - – Limitations of existing neural network methods identified
    - Neural network research almost disappears
- 1969—85: Adding domain knowledge
  - – Development of knowledge-based systems
  - – Success of rule-based expert systems,
    - E.g., DENDRAL, MYCIN
    - But were brittle and did not scale well in practice
- 1986-- Rise of machine learning
  - – Neural networks return to popularity
  - – Major advances in machine learning algorithms and applications
- 1990-- Role of uncertainty
  - – Bayesian networks as a knowledge representation framework
- 1995--AI as Science
  - – Integration of learning, reasoning, knowledge representation

– AI methods used in vision, language, data mining, etc

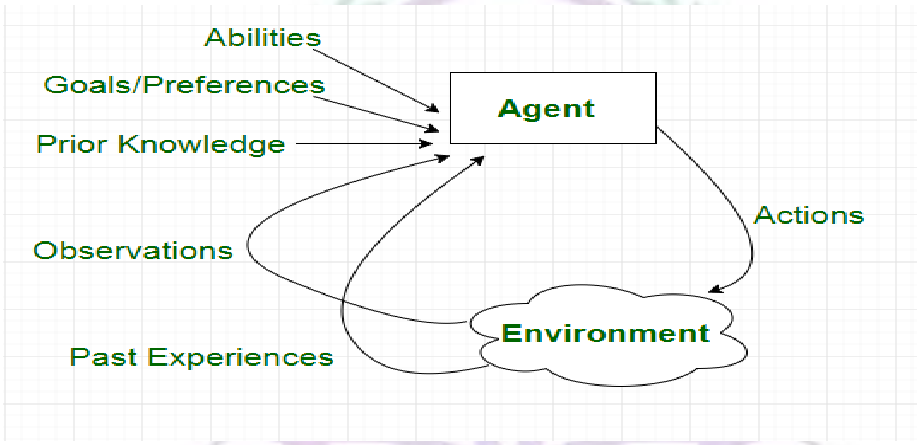
STRUCTURE OF AGENTS

Artificial intelligence is defined as the study of rational agents. A rational agent could be anything that makes decisions, as a person, firm, machine, or software. It carries out an action with the best outcome after considering past and current precepts (agent's perceptual inputs at a given instance). An AI system is composed of an agent and its environment. The agents act in their environment. The environment may contain other agents. An agent is anything that can be viewed as:

- perceiving its environment through sensors and
- acting upon that environment through actuators

Examples of Agent:

A software agent has Keystrokes, file contents, received network packages which act as sensors and displays on the screen, files, sent network packets acting as actuators.A Human-agent has eyes, ears, and other organs which act as sensors, and hands, legs, mouth, and other body parts acting as actuators. A Robotic agent has Cameras and infrared range finders which act as sensors and various motors acting as actuators.



Types of Agents

Agents can be grouped into four classes based on their degree of perceived intelligence and capability:

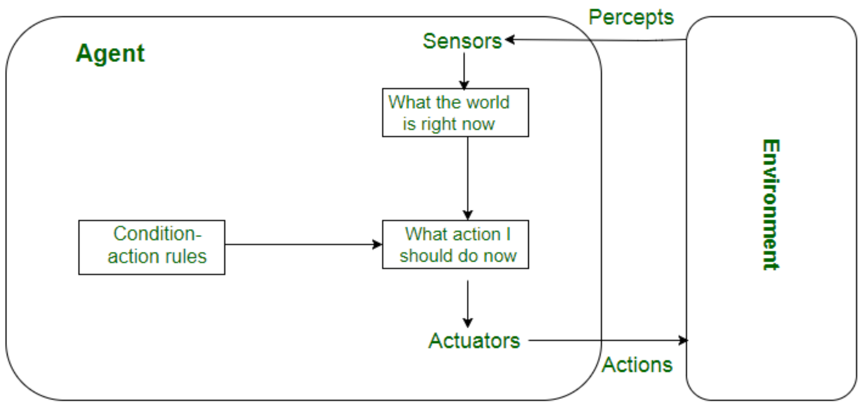
- Simple Reflex Agents
- Model-Based Reflex Agents
- Goal-Based Agents
- Utility-Based Agents
- Learning Agent

Simple Reflex Agents

Simple reflex agents ignore the rest of the percept history and act only based on the current percept. Percept history is the history of all that an agent has perceived to date. The agent function is based on the condition-action rule. A condition-action rule is a rule that maps a state i.e, condition to an action. If the condition is true, then the action is taken, else not. This agent function only succeeds when the environment is fully observable. For simple reflex agents operating in partially observable environments, infinite loops are often unavoidable. It may be possible to escape from infinite loops if the agent can randomize its actions. Problems with Simple reflex agents are:

- Very limited intelligence.
- No knowledge of non-perceptual parts of the state.
- Usually too big to generate and store.

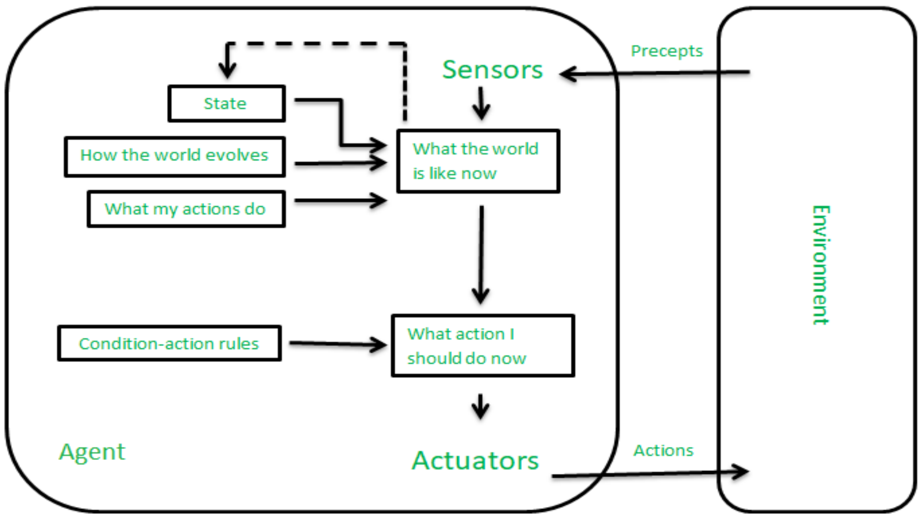
If there occurs any change in the environment, then the collection of rules needs to be updated.



Model-Based Reflex Agents

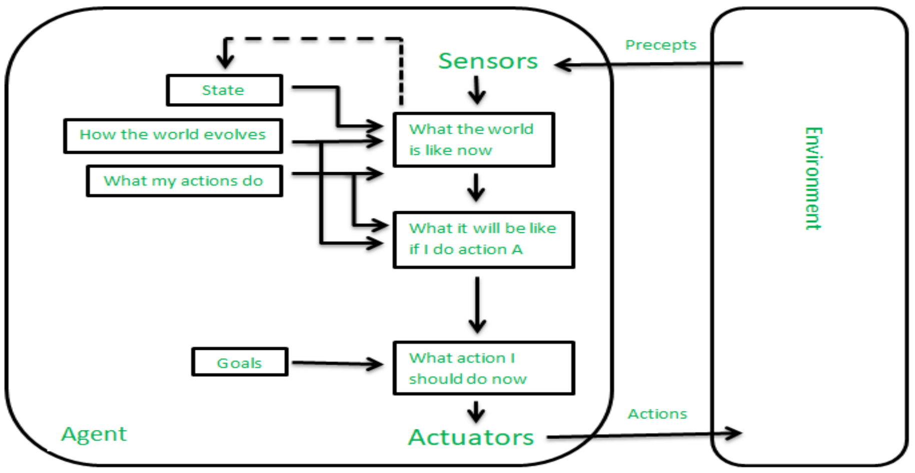
It works by finding a rule whose condition matches the current situation. A model-based agent can handle partially observable environments by the use of a model about the world. The agent has to keep track of the internal state which is adjusted by each percept and that depends on the percept history. The current state is stored inside the agent which maintains some kind of structure describing the part of the world which cannot be seen. Updating the state requires information about:

- how the world evolves independently from the agent, and
- how the agent's actions affect the world.



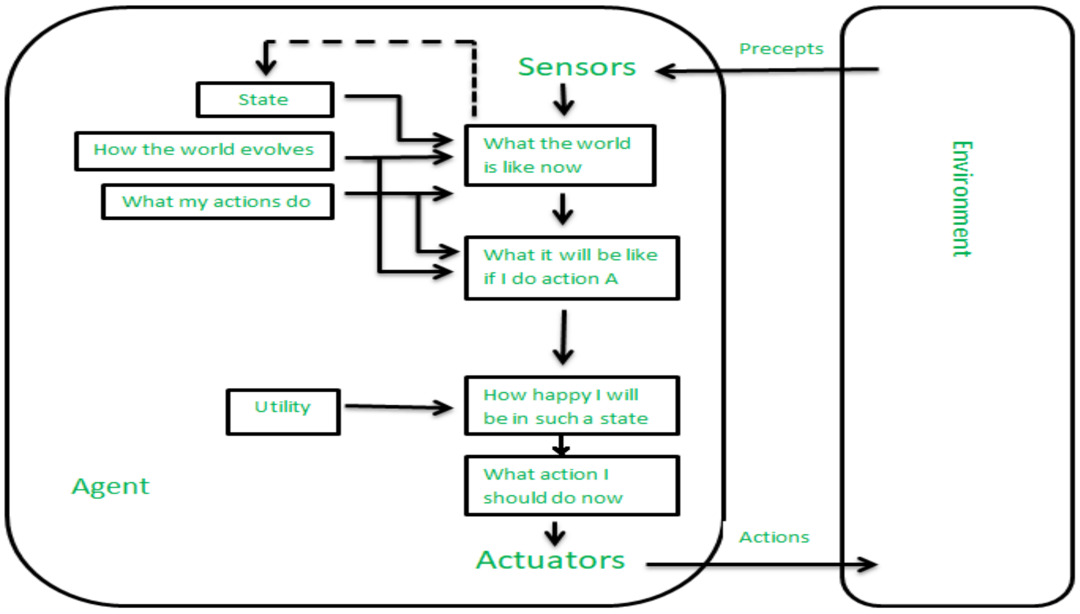
Goal-Based Agents

These kinds of agents take decisions based on how far they are currently from their goal (description of desirable situations). Their every action is intended to reduce its distance from the goal. This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state. The knowledge that supports its decisions is represented explicitly and can be modified, which makes these agents more flexible. They usually require search and planning. The goal-based agent's behaviour can easily be changed.



Utility-Based Agents

The agents which are developed having their end uses as building blocks are called utility-based agents. When there are multiple possible alternatives, then to decide which one is best, utility-based agents are used. They choose actions based on a preference (utility) for each state. Sometimes achieving the desired goal is not enough. We may look for a quicker, safer, cheaper trip to reach a destination. Agent happiness should be taken into consideration. Utility describes how "happy" the agent is. Because of the uncertainty in the world, a utility agent chooses the action that maximizes the expected utility. A utility function maps a state onto a real number which describes the associated degree of happiness.


Sensors
Precepts
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Utility
How happy I will be in such a state
What action I should do now
Agent
Actuators
Actions
Environment

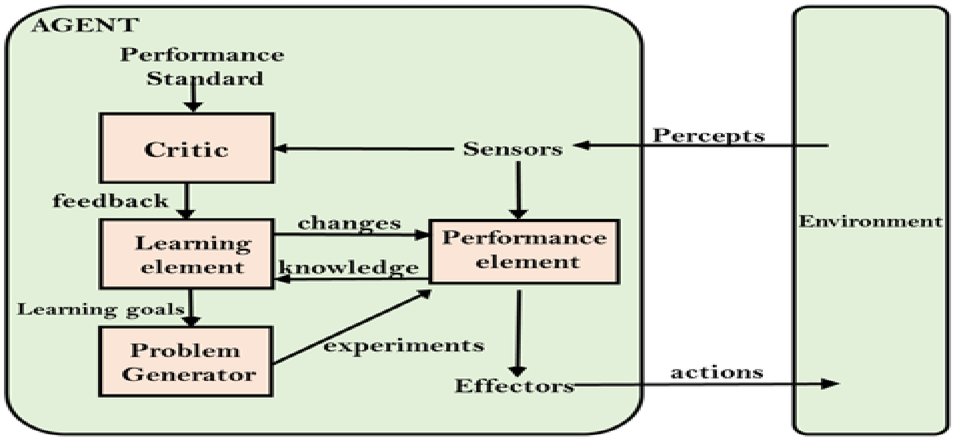
Learning Agent:

A learning agent in AI is the type of agent that can learn from its past experiences, or it has learning capabilities. It starts to act with basic knowledge and then can act and adapt automatically through learning. A learning agent has mainly four conceptual components, which are: Learning element: It is responsible for making improvements by learning from the environment .

Critic: The learning element takes feedback from critics which describes how well the agent is doing with respect to a fixed performance standard.

Performance element: It is responsible for selecting external action

Problem Generator: This component is responsible for suggesting actions that will lead to new and informative experiences.



PROBLEM SOLVING

FORMULATING PROBLEMS

Problem formulation is the process of deciding what actions and states to consider, given a goal

$$\begin{tikzcd}
\text{Formulate Goal, Formulate problem} \arrow[d] \\
\text{Search} \arrow[d] \\
\text{Execute}
\end{tikzcd}$$

WELL-DEFINED PROBLEMS AND SOLUTIONS A

problem can be defined formally by four components:

1. Initial state
2. Successor function
3. Goal test
4. Path cost

1. **Initial State**

The starting state which agent knows itself.

Successor Function

- A description of the possible actions available to the agent.
- State x, successor – FN (x) returns a set of < action, successor> ordered pairs, where each action is a legal action in a state x and each successor is a state that can be reached from x by applying that action.

State Space

The set of all possible states reachable from the initial state by any sequence of actions. The initial state and successor function defines the state space. The state space forms a graph in which nodes are state and axis between the nodes are action.

Path

A path in the state space is a sequence of state connected by a sequence of actions.

2. **Goal Test**

Test to determine whether the given state is the goal state. If there is an explicit set of possible goal states, then we can whether any one of the goal states is reached or not.

Example: In chess, the goal is to reach a state called ―checkmate‖ where the opponent ‗s king is under attack and can‗t escape.

3. **Path cost**

A function that assigns a numeric cost to each path. The cost of a path can be described as the sum of the costs of the individual actions along that path.

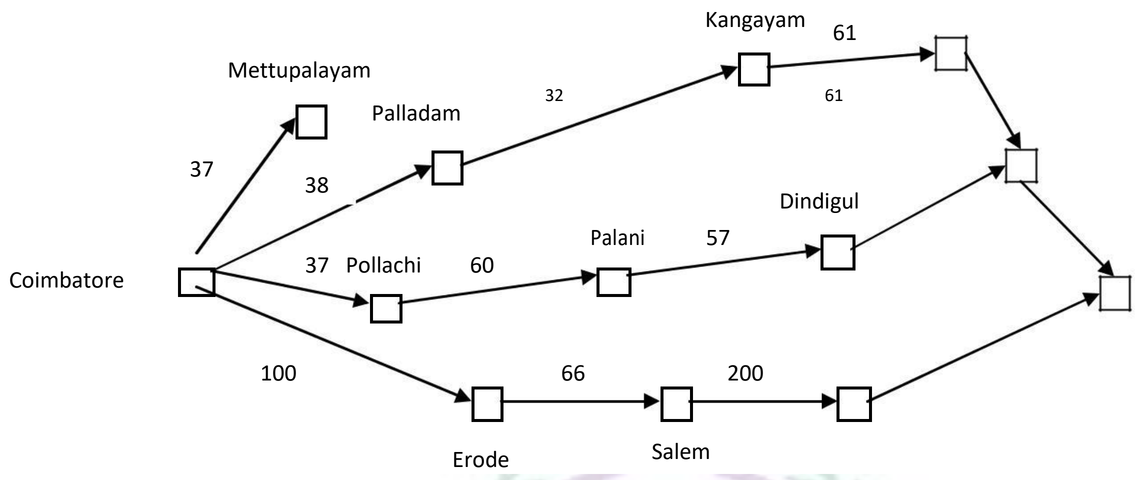
Step cost of taking an action _a‗to go from one state _x‗ to state _y‗ is denoted by C(x,a,y)

C-Cost, x,y- states , Action , Step costs are non-negative

These 4 elements are gathered into a data structure that is given as input to problem solving algorithm. A solution quality is measured by path cost function. An optimal solution has lowest path cost among all solutions.

Total cost = Path cost + Search cost

Example: Route finding problem



Route Finding Problem

Initial State: In (Coimbatore)

Successor Function: {< Go (Pollachi), In (Pollachi)>

< Go (Erode), In (Erode)>

< Go (Palladam), In (Palladam)>

< Go (Mettupalayam), In (Mettupalayam)>}

Goal Test: In (Chennai)

Path Cost: {(In (Coimbatore),}

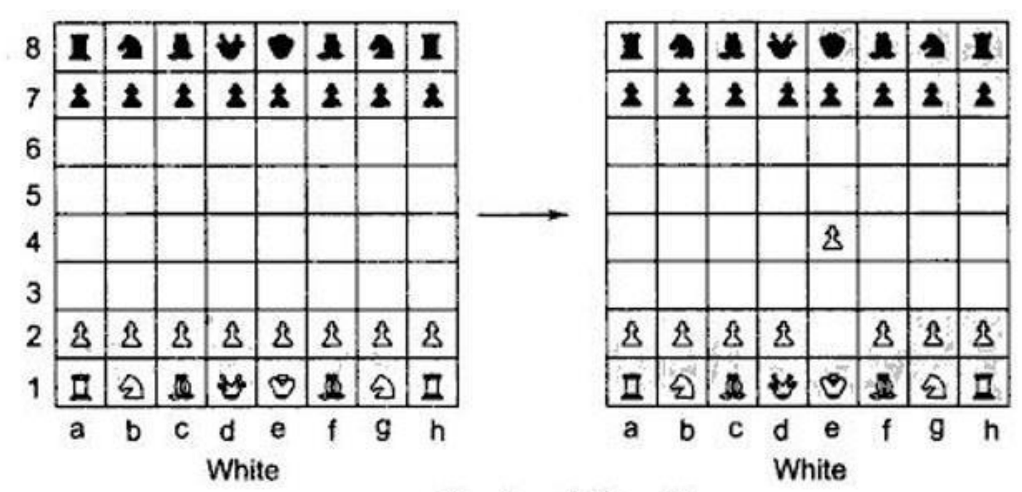
{Go (Erode),} = 100 [kilometers]

{In (Erode)}

Path cost = 100 + 66 + 200 + 140 = 506

Example: Playing chess

Initial State: Described as an 8 X 8 array where each positions contains a symbol standing for the appropriate piece in the official chess position.
Successor function: The legal states that results from set of rules.
They can be described easily by as a set of rules consisting of two parts: a left side that serves as a pattern to be matched against the current board position and a right side that describes the changes to be made to the board position to reflect the move. An example is shown in the following figure.



One Legal Chess Move

The legal states that results from set of rules

However if we write rules like the one above, we have to write a very large number of them since there has to be a separate set of rule for each of them roughly 10^{120} possible board positions.
Practical difficulties to implement large number of rules,

1. It will take too long to implement large number of rules and could not be done without mistakes.
2. No program could easily handle all those rules and storing it possess serious difficulties.

To minimize such problems, we have to write rules describing the legal moves in as a general way as possible. The following is the way to describe the chess moves.
Current Position
While pawn at square (e, 2), AND Square (e, 3) is empty, AND Square (e , 4) is empty.
Changing Board Position
Move pawn from Square (e, 2) to Square (e , 4) .
Some of the problems that fall within the scope of AI and the kinds of techniques will be useful to solve these problems.
GoalTest
Any position in which the opponent does not have a legal move and his or her king is under attack.
Example: Water Jug Problem

A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?
State: (x, y) x= 0, 1, 2, 3, or 4 y= 0, 1, 2, 3
x represents quantity of water in 4-gallon jug and y represents quantity of water in 3-gallon jug.
•**Start state**: (0, 0).

•**Goal state:** (2, n) for any n. Attempting to end up in a goal state.(since the problem doesn't specify the quantity of water in 3-gallon jug)

1.	(x, y) If $x < 4$	$\rightarrow (4, y)$	Fill the 4-gallon jug
2.	(x, y) If $y < 3$	$\rightarrow (x, 3)$	Fill the 3-gallon jug
3.	(x, y) If $x > 0$	$\rightarrow (x - d, y)$	Pour some water out of the 4-gallon jug
4.	(x, y) If $y > 0$	$\rightarrow (x, y - d)$	Pour some water out of the 3-gallon jug
5.	(x, y) If $x > 0$	$\rightarrow (0, y)$	Empty the 4-gallon jug on the ground
6.	(x, y) If $y > 0$	$\rightarrow (x, 0)$	Empty the 3-gallon jug on the ground
7.	(x, y) If $x + y \geq 4, y > 0$	$\rightarrow (4, y - (4 - x))$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full
8.	(x, y) If $x + y \geq 3, x > 0$	$\rightarrow (x - (3 - y), 3)$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
9.	(x, y) If $x + y \leq 4, y > 0$	$\rightarrow (x + y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10.	(x, y) If $x + y \leq 3, x > 0$	$\rightarrow (0, x + y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11.	$(0, 2)$	$\rightarrow (2, 0)$	Pour the 2 gallons from the 3-gallon Jug into the 4-gallon jug
12.	$(2, y)$	$\rightarrow (0, y)$	Empty the 2 gallons in the 4-gallon Jug on the ground

Production rules for the water jug problem

Trace of steps involved in solving the water jug problem First solution

Number of Steps	Rules applied	4-g jug	3-g jug
1	Initial state	0	0
2	R2 {Fill 3-g jug}	0	3
3	R7 {Pour all water from 3 to 4-g jug}	3	0
4	R2 {Fill 3-g jug}	3	3
5	R5 {Pour from 3 to 4-g jug until it is full}	4	2
6	R3 {Empty 4-gallon jug}	0	2
7	R7 {Pour all water from 3 to 4-g jug}	2	0

Goal State

Second Solution

Number of Steps	Rules applied	4-g jug	3-g jug
1	Initial state	0	0
2	R1 {Fill 4-gallon jug}	4	0
3	R6 {Pour from 4 to 3-g jug until it is full}	1	3
4	R4 {Empty 3-gallon jug}	1	0
5	R8 {Pour all water from 4 to 3-gallon jug}	0	1
6	R1 {Fill 4-gallon jug}	4	1
7	R6 {Pour from 4 to 3-g jug until it is full}	2	3
8	R4 {Empty 3-gallon jug}	2	0

Goal State

Example 8-puzzle Problem

The 8-puzzle problem consists of a 3 x 3 board with eight numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space. The object is to reach a specified goal state.

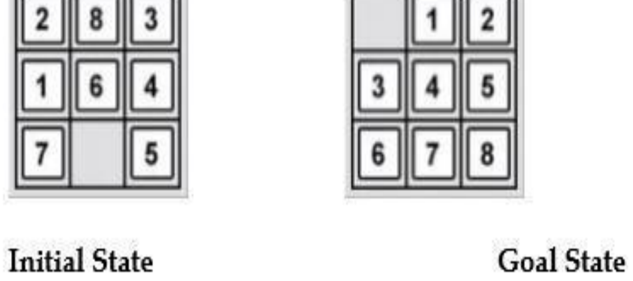
States: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.

Initial state: Any state can be designated as the initial state.

Successor function: This generates the legal states that result from trying the four actions (blank moves Left, Right, Up, or Down).

Goal test: This checks whether the state matches the goal configuration (Other goal configurations are possible.)

Path cost: Each step costs 1, so the path cost is the number of steps in the path.



8 Puzzle Problem

Exampe:8-queens problem

The goal of the 8-queens problem is to place eight queens on a chessboard such that no queen attacks any other. (A queen attacks any piece in the same row, column or diagonal.

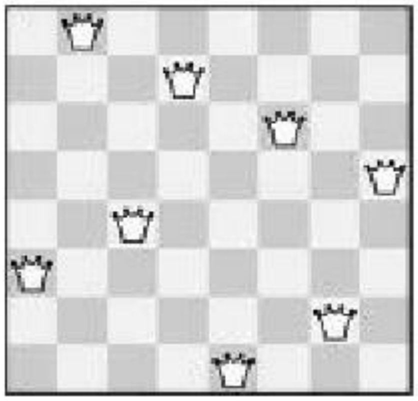
States: Any arrangement of 0 to 8 queens on the board is a state.

Initial state: No queens on the board.

Successor function: Add a queen to any empty square.

Goal test: 8 queens are on the board, none attacked.

Path cost: Zero (search cost only exists)


Solution to the 8 queens problem

SEARCH STRATEGIES:

- **Best-First Search:**

It is a general heuristic-based search technique. In best first search, in the graph of problem representation, one evaluation function (which corresponds to heuristic function) is attached with every node. The value of evaluation function may depend upon cost or distance of current node from goal node. The decision of which node to be expanded depends on the value of this evaluation function. The best first can be understood from following tree. In the tree, the attached value with nodes indicates utility value. The expansion of nodes according to best first search is illustrated in
best first search technique is applied, however it is beneficial sometimes to search a graph instead of tree to avoid the searching of duplicate paths. In the process to do so, searching is done in a directed graph in which each node represents a point in the problem space. This graph is known as OR-graph. Each of the branches of an OR graph represents an alternative problem-solving path.
Two lists of nodes are used to implement a graph search procedure discussed above. These are

1.OPEN: these are the nodes that have been generated and have had the heuristic function applied to them but not have been examined yet.

2.CLOSED: these are the nodes that have already been examined. These nodes are kept in the memory if we want to search a graph rather than a tree because whenever a node will be generated, we will have to check whether it has been generated earlier.

The best first search is a way of combining the advantage of both depth first and breath first search. The depth first search is good because it allows a solution to be found without all competing branches have to be expanded. Breadth first search is good because it does not get trapped on dead ends of path. The way of combining this is to follow a single path at a time but switches between paths whenever some competing paths looks more promising than current one does. Hence at each step of best first search process, we select most promising node out of successor nodes that have been generated so far.

The functioning of best first search is summarized in the following steps:

1. It maintains a list open containing just the initial state.
2. Until a goal is found or there are no nodes left in open list do:
   a. Pick the best node from open,
   b. Generate its successor, and for each successor:
      i. Check, and if it has not been generated before evaluate it and add it to open and record its parent.
      ii. If it has been generated before, and new path is better than the previous parent then change the parent.

The algorithm for best first search is given as follows:

Algorithm: Best first search

1. Put the initial node on the list say _OPEN‘.
2. If (OPEN = empty or OPEN= goal) terminate search, else
3. Remove the first node from open(say node is a)
4. If (a=goal) terminate search with success else
5. Generate all the successor node of _a‘. Send node _a‘ to a list called _CLOSED‘. Find out the value of heuristic function of all nodes. Sort all children generated so far on the basis of their utility value. Select the node of minimum heuristic value for further expansion.
6. Go back to step 2.

   The best first search can be implemented using priority queue. There are variations of best first search. Example of these are greedy best first search, A* and recursive best first search.

Breadth First Search (Blind Search)

Let us discuss these strategies using water jug problem. These may be applied to any search problem.

Construct a tree with the initial state as its root.

Generate all the offspring of the root by applying each of the applicable rules to the initial state.

$$\begin{tikzcd} & (0, 0) \arrow[dl] \arrow[dr] & \\ (4, 0) & & (0, 3) \end{tikzcd}$$

Now for each leaf node, generate all its successors by applying all the rules that are appropriate.

(0, 0)

$$\begin{tikzcd}
 & & & & {} \arrow[dll] \arrow[drr] & & & \\
 & & (4,0) \arrow[dl] \arrow[d] \arrow[dr] & & & & (0,3) \arrow[dl] \arrow[d] \arrow[dr] & \\
 & (4,3) & (0,0) & (1,3) & & (4,3) & (0,0) & (3,0)
\end{tikzcd}$$

Continue this process until some rule produces a goal state.

Algorithm

```
1. Create a variable called NODE-LIST and set it to initial state.
2. Unit a goal state is found, or NODE-LIST is empty do
    a. Remove the first element from NODE-LIST and call it E. if NODE-LIST is
       empty, quit.
    b. For each way that each rule can match the state described in E do:
        i. Apply the rule to generate a new state.
        ii. If the new state is a goal state, quit and return this state.
        iii. Otherwise, add the new state to the end of NODE-LIST.
```

Depth First Search

Algorithm

```
1.If the initial state is the goal state, quit and return success.
2.Otherwise do the following until success or failure is signaled:
    a. Generate a successor, E, of initial state. If there are no more successor, signal failure.
    b. Call depth first search, with E as the initial state.
    c. If the success is returned, signal success. Otherwise continue in this loop.
```

Backtracking

- In this search, we pursue a single branch of the tree until it yields a solution or until a decision to terminate the path is made.
- It makes sense to terminate a path if it reaches dead-end, produces a previous state. In such a state backtracking occurs.
- Chronological Backtracking: order in which steps are undone depends only on the temporal sequence in which steps were initially made.
- Specifically, most recent step is always the first to be undone. This is also simple backtracking.

$$\begin{tikzcd}
 & & (0,0) \arrow[dl] \\
 & (4,0) \arrow[dl] & \\
(4,3) & &
\end{tikzcd}$$

Advantages of Depth First search

- DFS requires less memory since only the nodes on the current path are stored.
- By chance DFS may find a solution without examining much of the search space at all.

Advantages of Breath First search

- BFS cannot be trapped exploring a blind alley.
- If there is a solution, BFS is guaranteed to find it.
- If there are multiple solutions, then a minimal solution will be found.

Heuristic Search

• Heuristics are criteria for deciding which among several alternatives be the most effective to achieve some goal.

• Heuristic is a technique that improves the efficiency of a search process possibly by sacrificing claims of systematic and completeness. It no longer guarantees to find the best answer but almost always finds a very good answer.

• Using good heuristics, we can hope to get good solution to hard problems (such as travelling salesman) in less than exponential time.

• There are **general-purpose** heuristics that are useful in a wide variety of problem domains.

• We can also construct **special purpose** heuristics, which are domain specific.

General Purpose Heuristics

• A general-purpose heuristics for combinatorial problem is nearest neighbour algorithms which works by selecting the locally superior alternative.

• For such algorithms, it is often possible to prove an upper bound on the error which provide reassurance that one is not paying too high a price in accuracy for speed.

• In many AI problems, it is often hard to precisely measure the goodness of a particular solution.

• For real world problems, it is often useful to introduce heuristics based on relatively unstructured knowledge. It is impossible to define this knowledge in such a way that mathematical analysis can be performed.

• In AI approaches, behavior of algorithms are analyzed by running them on computer as contrast to analyzing algorithm mathematically.

•There are at least many reasons for the adhoc approaches in AI.

- It is a lot more fun to see a program do something intelligent than to prove it.
- AI problem domains are usually complex, so generally not possible to produce analytical proof that a procedure will work.
- It is even not possible to describe the range of problems well enough to make statistical analysis of program behaviour meaningful.

•But still it is important to keep performance question in mind while designing algorithm.

•One of the most important analysis of the search process is straightforward i.e., Number of nodes in a complete search tree of depth D and branching factor F is F*D‖.

•This simple analysis motivates to

- Look for improvements on the exhaustive search.

- Find an upper bound on the search time which can be compared with exhaustive search procedures.

PROBLEM SOLVING METHODS, HEURISTIC SEARCH TECHNIQUES

Search techniques are the general problem-solving methods. When there is a formulated search problem, a set of search states, a set of operators, an initial state and a goal criterion we can use search techniques to solve a problem.

The A* Algorithm:

The A* algorithm is a specialization of best first search. It most widely known form of best first search. It provides genera guidelines about how to estimate goal distance for general search graph. at each node along a path to the goal node, the A* algorithm generate all successor nodes and computes an estimate of distance (cost) from the start node to a goal node through each ofthe successors. if then chooses the successor with shortest estimated distance from expansion. It calculates the heuristic function based on distance of current node from the start node and distance of current node to goal node.

The form of heuristic estimation function for A* is defined as follows:

$f(n)=g(n)+h(n)$

where f(n)= evaluation function

g(n)= cost (or distance) of current node from start node.

h(n)= cost of current node from goal node.

In A* algorithm the most promising node is chosen from expansion. The promising node is decided based on the value of heuristic function. Normally the node having lowest value of f (n) is chosen for expansion. We must note that the goodness of a move depends upon the nature of problem, in some problems the node having least value of heuristic function would be most promising node, where in some situation, the node having maximum value of heuristic function is chosen for expansion. A* algorithm maintains two lists. One store the list of open nodes and other maintain the list of already expanded nodes. A* algorithm is an example of optimal search algorithm. A search algorithm is optimal if it has admissible heuristic. An algorithm has admissible heuristic if its heuristic function h(n) never overestimates the cost to reach the goal. Admissible heuristic are always optimistic because in them, the cost of solving the problem is less than what actually is. The A* algorithm works as follows:

A* algorithm:

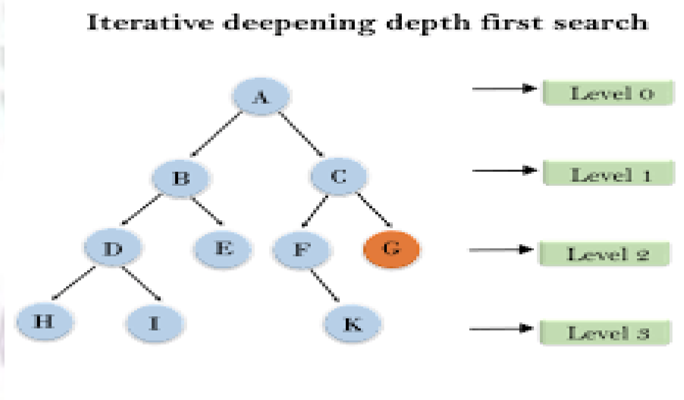
1. Place the starting nodes‘on _OPEN‘ list.
2. If OPEN is empty, stop and return failure.
3. Remove from OPEN the node _n‘that has the smallest value of f*(n). if node _n is a goal node, return success and stop otherwise.

4. Expand _generating all of its successors _n‘and place _n‘ on CLOSED. For every successor _n‘if _n‘ is not already OPEN , attach a back pointer to _n‘. compute f*(n) and place it on CLOSED.
5. Each _n‘ that is already on OPEN or CLOSED should be attached to back pointers which reflect the lowest f*(n) path. If _n‘was on CLOSED and its pointer was changed, remove it and place it on OPEN.
6. Return to step 2.

Iterative Deepening A*

Iterative deepening A* (IDA*) is a graph traversal and path search algorithm that can find the shortest path between a designated start node and any member of a set of goal nodes in a weighted graph. It is a variant of iterative deepening depth-first search that borrows the idea to use a heuristic function to evaluate the remaining cost to get to the goal from the A* search algorithm. Since it is a depth-first search algorithm, its memory usage is lower than in A*, but unlike ordinary iterative deepening search, it concentrates on exploring the most promising nodes and thus doesn’t go to the same depth everywhere in the search tree. Unlike A*, IDA* doesn’t utilize dynamic programming and therefore often ends up exploring the same nodes many times

IDA* is a memory constrained version of A*. It does everything that the A* does, it has the optimal characteristics of A* to find the shortest path but it uses less memory than A*.



Game Playing

Game Playing is one of the oldest sub-fields in AI. Game playing involves abstract and pure form of competition that seems to require intelligence. It is easy to represent the states and actions. To implement the game playing very little world knowledge is required.

The most common used AI technique in game is search. Game playing research has contributed ideas on how to make the best use of time to reach good decisions.

Game playing is a search problem defined by:

Initial state of the game

Operators defining legal moves

Successor function

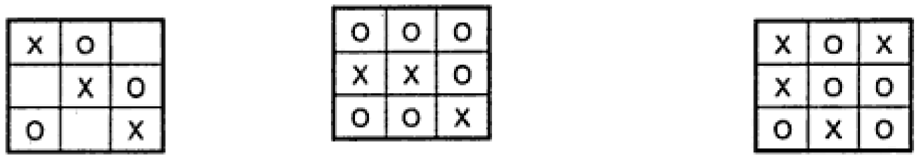
Terminal test defining end of game states
Goal test
Path cost/utility/payoff function
More popular games are too complex to solve, requiring the program to take its best guess. " for example in chess, the search tree has 1040 nodes (with branching factor of 35). It is the opponent because of whom uncertainty arises.
Characteristics of game playing

- There are always an "unpredictable" opponent:
- The opponent introduces uncertainty
- The opponent also wants to win

The solution for this problem is a strategy, which specifies a move for every possible opponent reply.
Time limits:
Game are often played under strict time constraints (eg:chess) and therefore must be very effectively handled.There are special games where two players have exactly opposite goals. There are also perfect information games(sch as chess and go) where both the players have access to the same information about the game in progress (e.g. tic-tac-toe). In imoerfect game information games (such as bridge or certain card games and games where dice is used). Given sufficient time and space, usually an optimum solution can be obtained for the former by exhaustive search, though not for the latter.There are basically two types of games

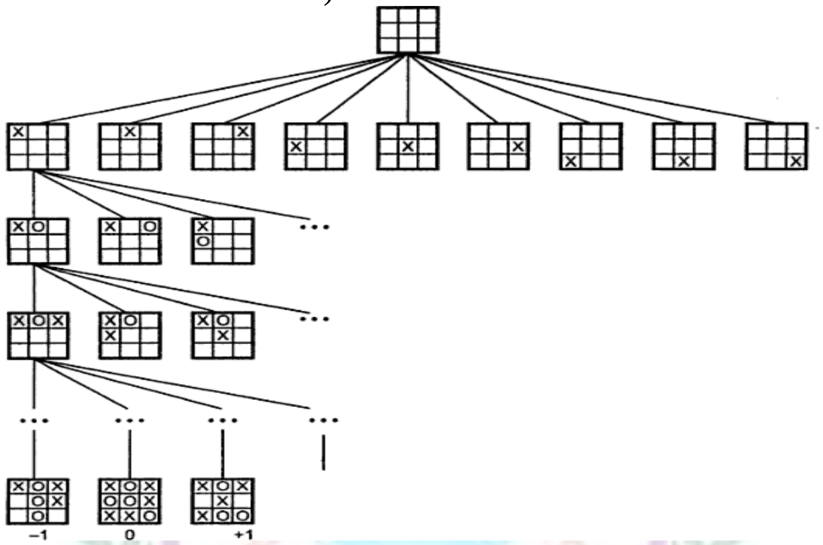
- Deterministic games
- Chance games

Game like chess and checker are perfect information deterministic games whereas games like scrabble and bridge are imperfect information. We will consider only two player discrete, perfect information games, such as tic-tac-toe, chess, checkers etc... . Two- player games are easier to imagine and think and more common to play.
Minimize search procedure
Typical characteristic of the games is to look ahead at future position in order to succeed. There is a natural correspondence between such games and state space problems.
In a game like tic-tac-toe
States-legal board positions
Operators-legal moves
Goal-winning position
The game starts from a specified initial state and ends in position that can be declared win for one player and loss for other or possibly a draw. Game tree is an explicit representation of all possible plays of the game. We start with a 3 by 3 grid..Then the two players take it in turns to place a there marker on the board(one player uses the „X‟ marker, the other uses the „O‟ marker). The winner is the player who gets 3 of these markers in a row, eg.. if X wins



Search tree for tic-tac-toe

The root node is an initial position of the game. Its successors are the positions that the first player can reach in one move; their successors are the positions resulting from the second player's replies and so on. Terminal or leaf nodes are presented by WIN, LOSS or DRAW. Each path from the root ro a terminal node represents a different complete play of the game. The moves available to one player from a given position can be represented by OR links whereas the moves available to his opponent are AND links.
The trees representing games contain two types of nodes:
MAX- nodes (assume at even level from root)
MIN - nodes [assume at odd level from root)

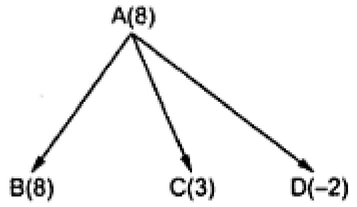


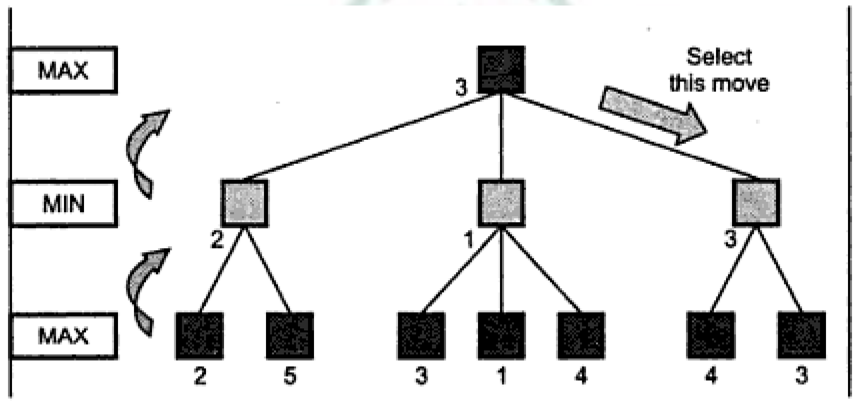
Search tree for tic-tac-toe
the leaves nodes are labelled WIN, LOSS or DRAW depending on whether they represent a win, loss or draw position from Max’s viewpoint. Once the leaf nodes are assigned their WIN-LOSS or DRAW status, each nodes in the game tree can be labelled WIN, LOSS or DRAW by a bottom up process.
Game playing is a special type of search, where the intention of all players must be taken into account.
Minimax procedure
Starting from the leaves of the tree (with final scores with respect to one player, MAX), and go backwards towards the root.
At each step, one player (MAX) takes the action that leads to the highest score, while the other player (MIN) takes the action that leads to the lowest score.
All the nodes in the tree will be scored and the path from root to the actual result is the one on which all node have the same score.
The minimax procedure operates on a game tree and is recursive procedure where a player tries to minimize its opponent’s advantage while at the same time maximize its own. The player hoping for positive number is called the maximizing player. His opponent is the minimizing player. If the player to move is the maximizing player, he is looking for a path leading to a large positive number and his opponent will try to force the play toward situation with strongly negative static evaluations. In game playing first construct the tree up till the depth-bound and then compute the evaluation function for the leaves. The next step is to propagate the values up to the starting. The procedure by which the scoring

information passes up the game tree is called the MINIMAX procedures since the score at each node is either minimum or maximum of the scores at the nodes immediately below

One-ply search

In this fig since it is the maximizing search ply 8 is transferred upwards to A





Two-ply search

Static evaluation function

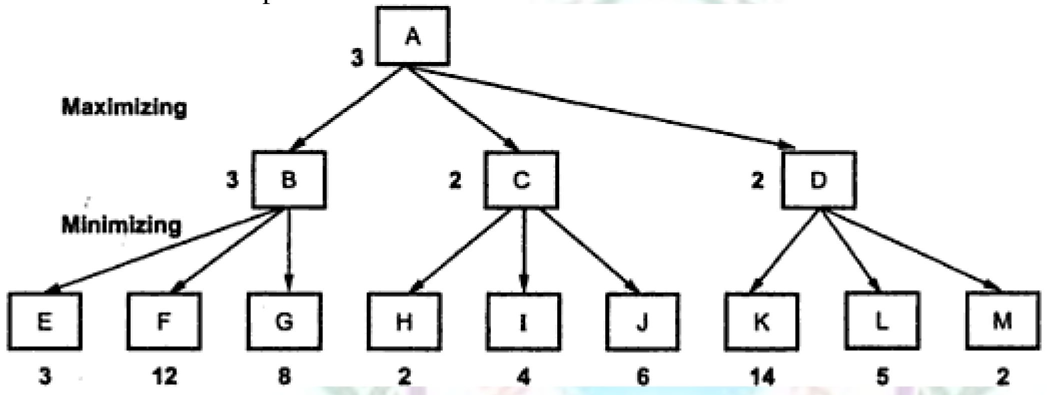
To play an entire game we need to combine search oriented and non-search-oriented techniques. The idea way to use a search procedure to find a solution to the problem statement is to generate moves through the problem space until a goal state is reached. Unfortunately for games like chess even with a good plausible move generator, it is not possible to search until goal state is reached. In the amount of time available it is possible to generate the tree at the most 10 to 20 ply deep. Then in order to choose the best move, the resulting board positions must be compared to discover which is most advantageous. This is done using the static evaluation function. The static evaluation function evaluates individual board positions by estimating how much likely they are eventually to lead to a win.

The minimax procedure is a depth-first, depth limited search procedure.

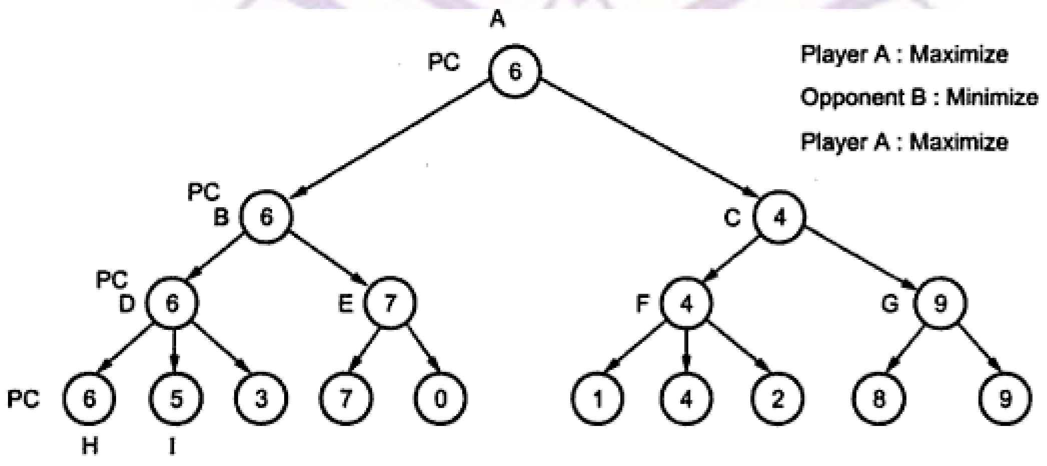
If the limit of search has reached, compute the static value of the current position relative to the appropriate layer as given below (maximizing or minimizing player). Report the result (value and path).

If the level is minimizing level(minimizer‟s turn)

Generate the successors of the current position. Apply MINIMAX to each of the successors. Return the minimum of the result.
If the level is a maximizing level. Generate the successors of current position Apply MINIMAX to each of these successors. Return the maximum of the result. The maximum algorithm uses the following procedures
MOVEGEN(POS)
It is plausible move generator. It returns a list of successors of „Pos‟.
STSTIC (Pos, Depth)
The static evaluation function that returns a number representing the goodness of „pos‟ from the current point of view.
DEEP-ENOUGH
It returns true if the search to be stopped at the current level else it returns false.
A MINIMAX example



Another example of minimax search procedure



In the above example, a Minimax search in a game tree is simulated. Every leaf has a corresponding value, which is approximated from player A‟s view point. When a path is chosen, the value of the child will be passed back to the parent. For example, the value for D is 6, which is the maximum value of its children, while the value for C is 4 which is the minimum value of F and G. In this example the best sequence of moves found by the maximizing/minimizing procedure is the path through nodes A, B, D and H, which is called the principal continuation. The nodes on the path are denoted as PC (principal continuation) nodes. For simplicity we can modify the game tree values slightly and use only maximization operations. The trick is to maximize the scores by negating the returned values from the children instead of searching for minimum scores and estimate the values at leaves from the player"s own viewpoint

Alpha-beta cutoffs:

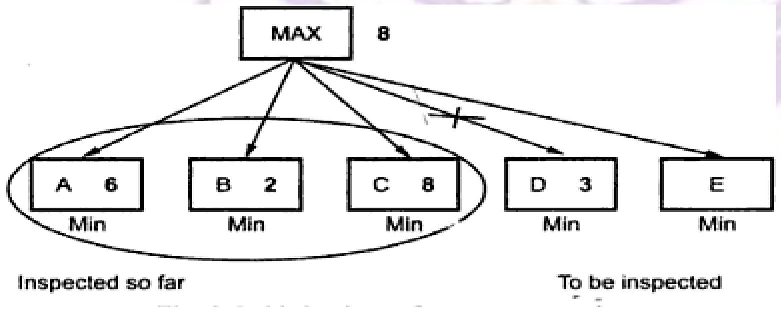
The basic idea of alpha-beta cutoffs is "It is possible to compute the correct minimax decision without looking at every node in the search tree". This is called pruning (allow us to ignore portions of the search tree that make no difference to the final choice).The general principle of alpha-beta pruning is Consider a node n somewhere in the tree, such that a player has a chance to move to this node.If player has a better chance m either at the parent node of n (or at any choice point further up) then n will never be reached in actual play.When we are doing a search with alpha-beta cut-offs, if a node"s value is too high, the minimizer will make sure it"s never reached (by turning off the path to get a lower value). Conversely, if a node"s value is too low, the maximizer will make sure it"s never reached. This gives us the following definitions

Alpha: the highest value that the maximize can guarantee himself by making some move at the current node OR at some node earlier on the path to this node.

Beta: the lowest value that the minimizer can guarantee by making some move at the current node OR at some node earlier on the path to this node.

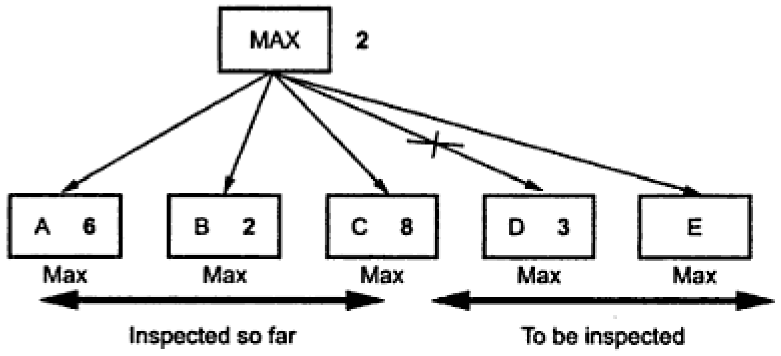
The maximize is constantly trying to push the alpha value up by finding better moves; the minimizer is trying to push the beta value down. If a node"s value is between alpha and beta, then the players might reach it. At the beginning, at the root of the tree, we don"t have any guarantees yet about what values the maximizer and minimizer can achieve. So we set beta to ∞ and alpha to -∞. Then as we move down the tree, each node starts with beta and alpha values passed down from its parent.

Consider a situation in which the MIN – children of a MAX-node have been partially inspected



Alpha-beta for a max node

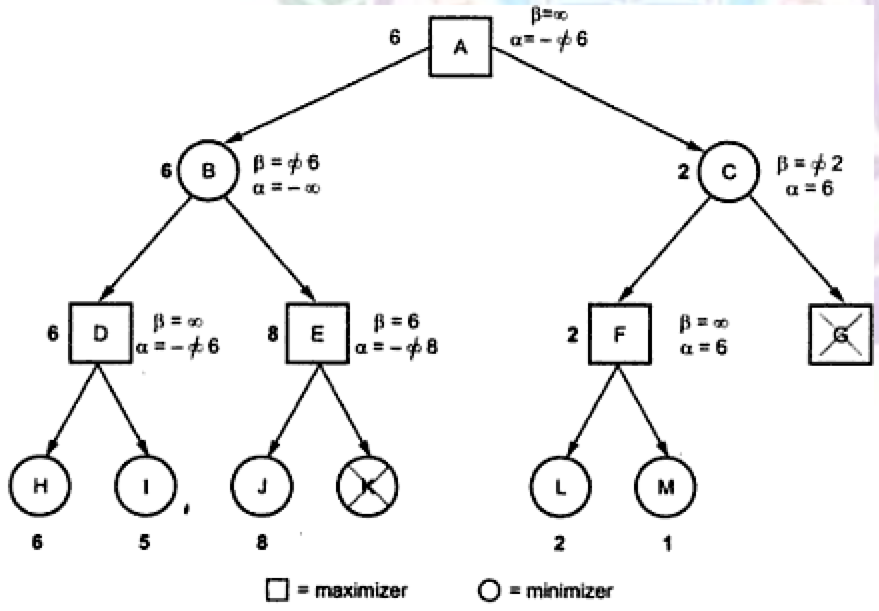
At this point the "tentative" value which is backed up so far of F is 8. MAX is not interested in any move which has a value of less than 8, since it is already known that 8 is the worst that MAX can do, so far. Thus the node D and all its descendent can be pruned or excluded from further exploration, since MIN will certainly go for a value of 3 rather than 8. Similarly for a MIN-node:



Alpha-beta for a min node

MIN is trying to minimize the game-value. So far, the value 2 is the best available form MIN‟s point of view. MIN will immediately reject node D, which can be stopped for further exploration.

In a game tree, each node represents a board position where one of the players gets to choose a move. For example, in the fig below look at the node C. As soon as we look at its left child, we realize that if the players reach node C, the minimizer can limit the utility to 2. But the maximize can get utility 6 by going to node B instead, so he would never let the game reach C. therefore we don‟t even have to look at C‟s other children



Tree with alpha-beta cut-offs

Initially at the root of the tree, there is no guarantee about what values the maximize and minimizer can achieve. So beta is set to ∞ and alpha to -∞. Then as we move down the tree, each node starts with beta and alpha values passed down from its parent. It it‟s a maximize node, and then alpha is increased if a child value is greater than the current alpha value. Similarly, at a minimizer node, beta may be decreased. This is shown in the fig.

At each node, the alpha and beta values may be updated as we iterate over the node’s children. At node E, when alpha is updated to a value of 8, it ends up exceeding beta. This is a point where alpha beta pruning is required we know the minimizer would never let the game reach this node so we don’t have to look at its remaining children. In fact, pruning happens exactly when the alpha and beta lines hit each other in the node value.

Algorithm-Alpha-beta

```
int AlphaBeta(int depth, int alpha, int beta)
{
     if (depth == 0)
          return Evaluate();
     GenerateLegalMoves();
     while (MovesLeft()) {
          MakeNextMove();
          val = -AlphaBeta(depth - 1, -beta, -alpha);
          UnmakeMove();
          if (val >= beta)
               return beta;
          if (val > alpha)
               alpha = val;
     }
     return alpha;
}
```

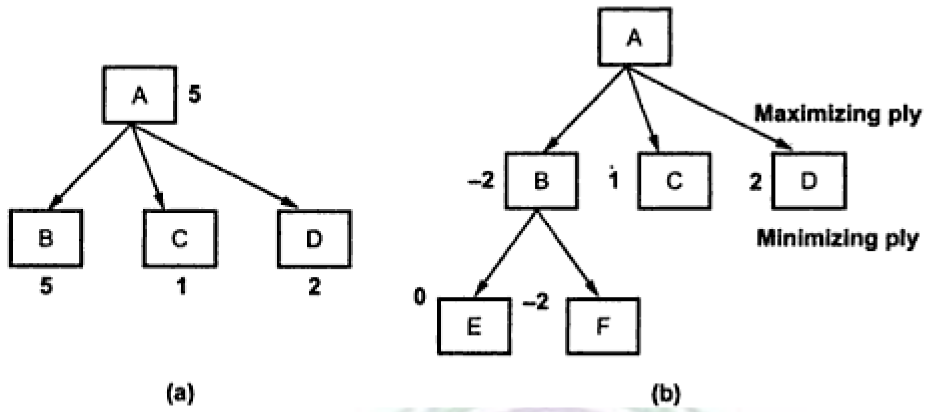
If the highlighted characters are removed, what is left is a min-max function. The function is passed the depth it should search, and –INIFINITY as alpha and +INIFINITY as beta.

```
val = AlphaBeta(5, -INFINITY, INFINITY);
```

This does a five-ply search

The Horizon effect

A potential problem in game tree search to a fixed depth is the horizon effect, which occurs when there is a drastic change in value immediately beyond the place where the algorithm stops searching. Consider the tree shown in the below fig.A. it has nodes A, B, C and D. at this level since it is a maximizing ply, the value which will passed up at A is 5.



Suppose node B is examined one more level as shown in fig B. then we see because of a minimizing ply value at B is -2 and hence the value passed to A is 2. This results in a drastic change in the situation. There are two proposed solutions to this problem, neither very satisfactory.
Secondary search
One proposed solution is to examine the search beyond the apparently best one to see if something is looming just over the horizon. In that case we can revert to the second-best move. Obviously then the second-best move has the same problem and there is not time to search beyond all possible acceptable moves.

UNIT-2: Knowledge, Reasoning & Planning

Reasoning - Knowledge based agent, Propositional Logic, Inference, Predicate logic (first order logic), Resolution

Structured Knowledge Representation – Frames, Semantic Nets

Planning - A Simple Planning Agent, Form Problem Solving to Planning, Basic representation of plans, partial order planning, hierarchical planning

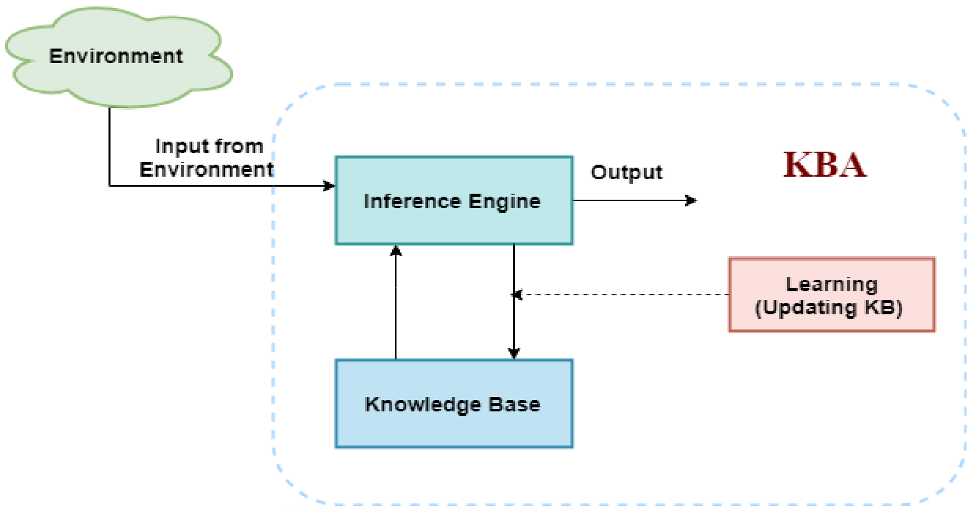
KNOWLEDGE BASE AGENT:

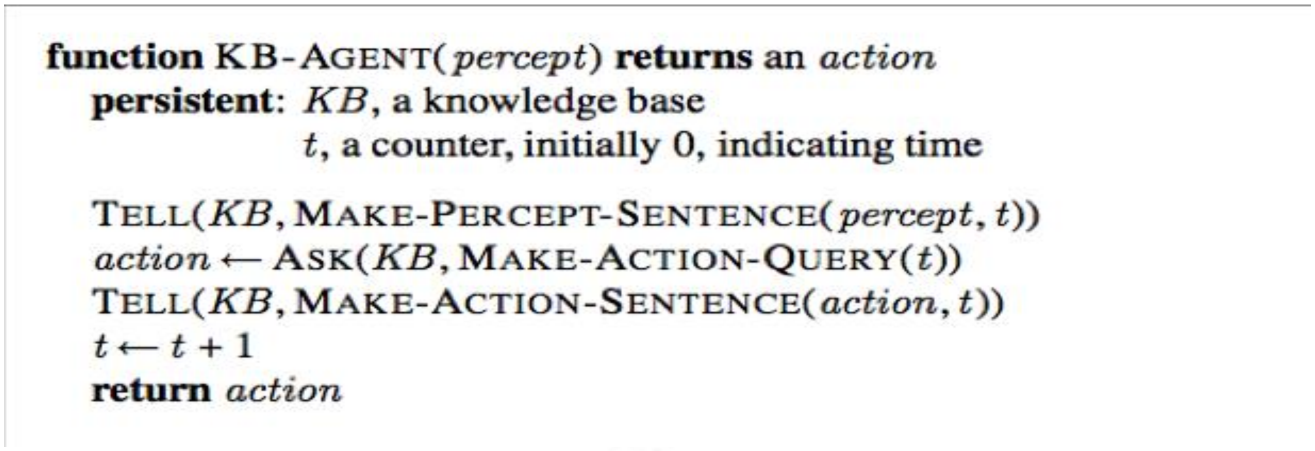
- An intelligent agent needs knowledge about the real world for taking decisions and reasoning to act efficiently.
- Knowledge-based agents are those agents who have the capability of maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently.
- Knowledge-based agents are composed of two main parts:
  1. Knowledge-base
  2. Inference system.

A knowledge-based agent must able to do the following:

- An agent should be able to represent states, actions, etc.
- An agent Should be able to incorporate new percepts
- An agent can update the internal representation of the world
- An agent can deduce the internal representation of the world
- An agent can deduce appropriate actions.

Architecture of knowledge Based Agent





```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Each time when the function is called, it performs its three operations:

- Firstly it TELLs the KB what it perceives.
- Secondly, it asks KB what action it should take
- Third agent program TELLS the KB that which action was chosen.

Various levels of knowledge-based agent:

- **Knowledge level:** Knowledge level is the first level of knowledge-based agent, and in this level, we need to specify what the agent knows, and what the agent goals are. With these specifications, we can fix its behaviour.
- **Logical level:** At this level, we understand that how the knowledge representation of knowledge is stored. At this level, sentences are encoded into different logics. At the logical level, an encoding of knowledge into logical sentences occurs.
- **Implementation level:** This is the physical representation of logic and knowledge. At the implementation level agent perform actions as per logical and knowledge level.

Approaches to designing a knowledge-based agent:

- **Declarative approach**: We can create a knowledge-based agent by initializing with an empty knowledge base and telling the agent all the sentences with which we want to start with. This approach is called Declarative approach.
- **Procedural approach**: In the procedural approach, we directly encode desired behaviour as a program code. Which means we just need to write a program that already encodes the desired behaviour or agent.

In the real world, a successful agent can be built by combining both declarative and procedural approaches, and declarative knowledge can often be compiled into more efficient procedural code.

KNOWLEDGE REPRESENTATION

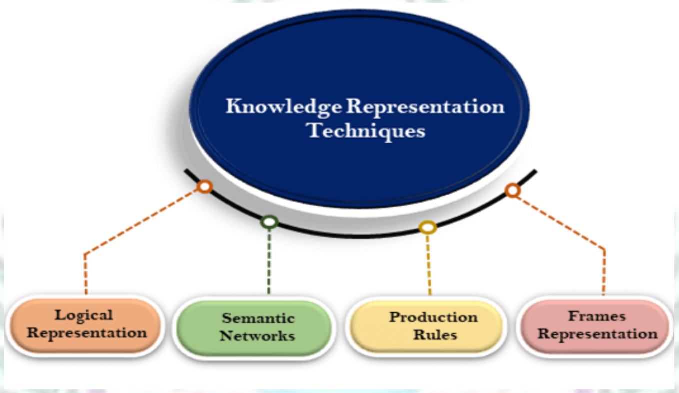
- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behaviour of agents.
- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.
- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an

intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

What to Represent:

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Techniques Of Knowledge Representation



Logical Representation

- Logical representation is a language with some concrete rules which deals with propositions and has no ambiguity in representation. Logical representation means drawing a conclusion based on various conditions. Each sentence can be translated into logics using syntax and semantics.
- Syntax:
- Syntaxes are the rules which decide how we can construct legal sentences in the logic.
- It determines which symbol we can use in knowledge representation.
- Semantics:
- Semantics are the rules by which we can interpret the sentence in the logic.
- Semantic also involves assigning a meaning to each sentence.

Logical representation can be categorised into mainly two logics:

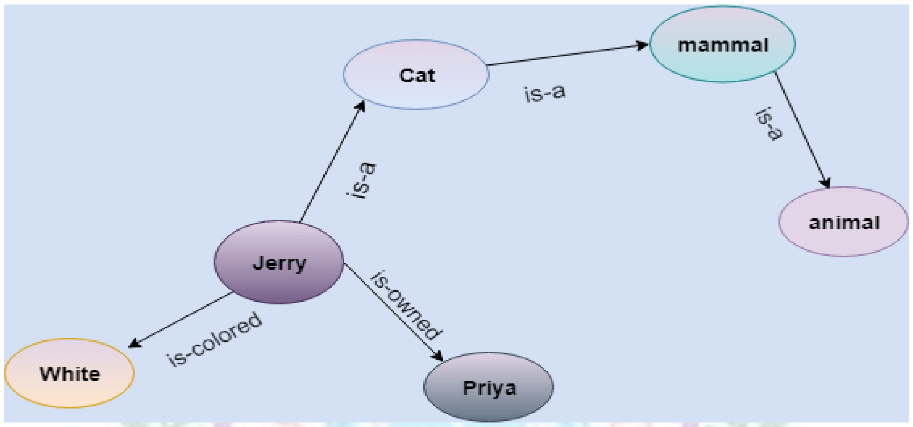
1. Propositional Logics
2. Predicate logics

Semantic Network Representation

Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical

networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects. Semantic networks can categorize the object in different forms and can also link those objects. Semantic networks are easy to understand and can be easily extended.
Statements:

1. Jerry is a cat.
2. Jerry is a mammal
3. Jerry is owned by Priya.
4. Jerry is brown coloured.
5. All Mammals are animal.



Production Rules

- Production rules system consist of (**condition, action**) pairs which mean, "If condition then action". It has mainly three parts:
- The set of production rules
- Working Memory
- The recognize-act-cycle

Example:

- IF (at bus stop AND bus arrives) THEN action (get into the bus)
- IF (on the bus AND paid AND empty seat) THEN action (sit down).
- IF (on bus AND unpaid) THEN action (pay charges).
- IF (bus arrives at destination) THEN action (get down from the bus).

Frame Representation

- A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world. Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations. It consists of a collection of slots and slot values. These slots may be of any type and sizes. Slots have names and values which are called facets.
- **Facets:** The various aspects of a slot is known as **Facets**. Facets are features of frames which enable us to put constraints on the frames. A frame may consist of any number of slots, and a slot may include any number of facets and facets may have any number of values. A frame is also known as **slot-filter knowledge representation** in artificial intelligence.
- Frames system consist of a collection of frames which are connected. In the frame, knowledge about an object or event can be stored together in the knowledge base. The frame is a type of

technology which is widely used in various applications including Natural language processing and machine visions.

- Frames system consist of a collection of frames which are connected. In the frame, knowledge about an object or event can be stored together in the knowledge base. The frame is a type of technology which is widely used in various applications including Natural language processing and machine visions.

Slots	Filters
Title	Artificial Intelligence
Genre	Computer Science
Author	Peter Norvig
Edition	Third Edition
Year	1996
Page	1152

PROPOSITIONAL LOGIC

- A proposition is a declarative statement which is either true or false.
- It is a technique of knowledge representation in logical and mathematical form.
- **Example:**

1. It is Sunday.
2. The Sun rises from West (False proposition)
3. 3+3= 7(False proposition)
4. 5 is a prime number.

Following are some basic facts about PL

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and **logical connectives**.
- These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- A proposition formula which is always false is called **Contradiction**.
- Statements which are questions, commands, or opinions are not propositions such as "**Where is Rohini**", "**How are you**", "**What is your name**", are not propositions.

Syntax of propositional logic:

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

- **Atomic Propositions:** Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

Example: 2+2 is 4, it is an atomic proposition as it is a true fact.
"The Sun is cold" is also a proposition as it is a false fact

- **Compound propositions:** Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

Example : "*It* is raining today, and street is wet."
"Ankit is a doctor, and his clinic is in Mumbai."

Logical Connectives:

There are mainly five connectives, which are given as follows:

- **Negation:** A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.
- **Conjunction:** A sentence which has ∧ connective such as, **P ∧ Q** is called a conjunction.
- **Disjunction:** A sentence which has ∨ connective, such as **P ∨ Q**. is called disjunction.
- **Implication**: A sentence such as P → Q, is called an implication. Implications are also known as if-then rules.
- **Biconditional**: A sentence such as P⇔ Q is a Biconditional sentence

Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A ∨ B
→	Implies	Implication	A → B
⇔	If and only if	Biconditional	A⇔ B
¬ or ~	Not	Negation	¬ A or ¬ B

Logical equivalence:

- Logical equivalence is one of the features of propositional logic. Two propositions are said to be logically equivalent if and only if the columns in the truth table are identical to each other.

A	B	¬A	¬A∨ B	A→B
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Properties of Operators:

- **Commutativity:**

P∧ Q= Q ∧ P or P ∨ Q = Q ∨ P.

- **Associativity:**
  (P ∧ Q) ∧ R= P ∧ (Q ∧ R) (P ∨ Q) ∨ R= P ∨ (Q ∨ R)
- **Identity element:**
  P ∧ True = P P ∨ True= True.
- **Distributive:**
  P∧ (Q ∨ R) = (P ∧ Q) ∨ (P ∧ R).
  P ∨ (Q ∧ R) = (P ∨ Q) ∧ (P ∨ R).
- **DE Morgan's Law:**
  ¬ (P ∧ Q) = (¬P) ∨ (¬Q)
  ¬ (P ∨ Q) = (¬ P) ∧ (¬Q).
- **Double-negation elimination:**
  ¬ (¬P) = P.

Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic. Example:
  - **All the humans are intelligent.**
  - **Some apples are sweet.**
- Propositional logic has limited expressive power.
- In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

INFERENCE:

We need intelligent computers which can create new logic from old logic or by evidence, **so generating the conclusions from evidence and facts is termed as Inference**.

- **Inference rules:** Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal.

Following are some terminologies related to inference rules:

- **Implication:** It is one of the logical connectives which can be represented as P → Q. It is a Boolean expression.
- **Converse:** The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as Q → P.
- **Contrapositive:** The negation of converse is termed as contrapositive, and it can be represented as ¬ Q → ¬ P.
- **Inverse:** The negation of implication is called inverse. It can be represented as ¬ P → ¬ Q.
- Some of the compound statements are equivalent to each other, which we can prove using truth table:

P	Q	P → Q	Q→ P	¬ Q → ¬ P	¬ P → ¬ Q.
T	T	T	T	T	T
T	F	F	T	F	T
F	T	T	F	T	F
F	F	T	T	T	T

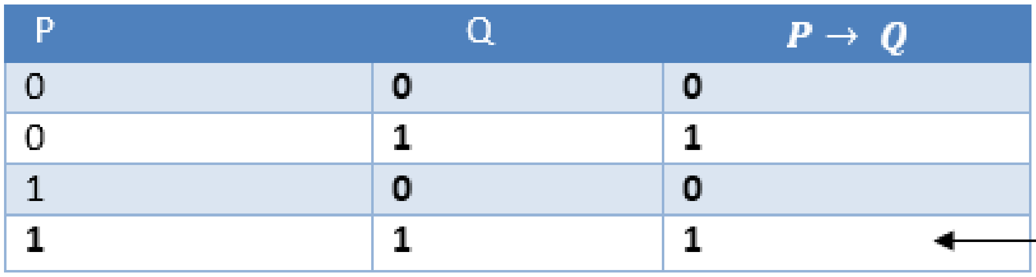
From the above truth table, we can prove that P → Q is equivalent to ¬ Q → ¬ P, and Q→ P is equivalent to ¬ P → ¬ Q.

Types of Inference rules:

1. Modus Ponens: It is one of the most important rules of inference, and it states that if P and P → Q is true, then we can infer that Q will be true.

Example:

- Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
  Statement-2: "I am sleepy" ==> P
  Conclusion: "I go to bed." ==> Q.
  Hence, we can say that, if P→ Q is true and P is true then Q will be true.



P	Q	*P → Q*
0	0	0
0	1	1
1	0	0
1	1	1

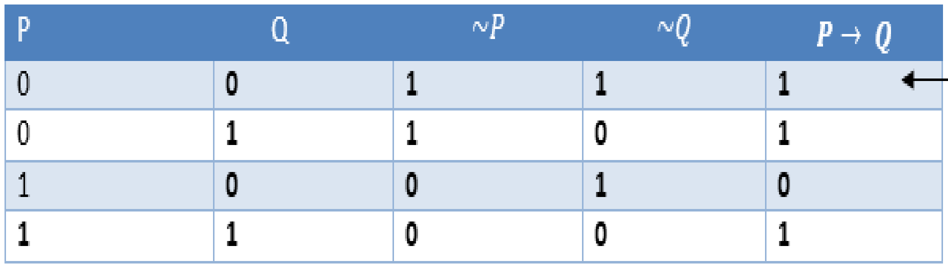
2. Modus Tollens:

The Modus Tollens rule state that if P→ Q is true and ¬ Q will also true

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q

Statement-2: "I do not go to the bed."==> ~Q

Statement-3: Which infers that "I am not sleepy" => ~P true, then ¬ P will also true.



P	Q	*~P*	*~Q*	*P → Q*
0	0	1	1	1
0	1	1	0	1
1	0	0	1	0
1	1	0	0	1

3. Hypothetical Syllogism:

The Hypothetical Syllogism rule state that if P→R is true whenever P→Q is true, and Q→R is true

- **Example:**
- **Statement-1:** If you have my home key then you can unlock my home. **P→Q**
  Statement-2: If you can unlock my home then you can take my money. **Q→R**
  Conclusion: If you have my home key then you can take my money. **P→R**

P	Q	R	$P \rightarrow Q$	$Q \rightarrow R$	$P \rightarrow R$
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	1

4. Disjunctive Syllogism:

The Disjunctive syllogism rule state that if P∨Q is true, and ¬P is true, then Q will be true

- **Example:**

  Statement-1: Today is Sunday or Monday. ==>P∨Q
  Statement-2: Today is not Sunday. ==> ¬P
  Conclusion: Today is Monday. ==> Q

P	Q	$\neg P$	$P \vee Q$
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	1

5. Addition:

The Addition rule is one the common inference rule, and it states that If P is true, then P∨Q will be true.

Example :

Statement:I have a vanilla ice-cream. ==> P
Statement-2: I have Chocolate ice-cream.
Conclusion: I have vanilla or chocolate ice-cream. ==> (P∨Q)

P	Q	$P \vee Q$
0	0	0
1	0	1
0	1	1
1	1	1

6. Simplification:

The simplification rule state that if **P∧ Q** is true, then **Q or P** will also be true

P	Q	$P \wedge Q$
0	0	0
1	0	0
0	1	0
1	1	1

7. Resolution:

The Resolution rule state that if P∨Q and ¬ P∧R is true, then Q∨R will also be true

P	¬ P	Q	R	$P \vee Q$	¬ P∧R	$Q \vee R$
0	1	0	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	0	0
1	0	0	1	1	0	1
1	0	1	0	1	0	1
1	0	1	1	1	0	1

FIRST-ORDER LOGIC(FOL)

In propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power.

Consider the following sentence, which we cannot represent using PL logic.

- **"Some humans are intelligent", or**
- **"Sachin likes cricket."**

To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

First-Order logic:

- It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- Also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic also assumes the following things in the world:
  - **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus etc
  - **Relations: It can be unary relation such as:** red, round, is adjacent, **or n-any relation such as:** the sister of, brother of, has color, comes between
  - **Function:** Father of, best friend, third inning of, end of, ...

First-order logic also has two main parts:

1. **Syntax**
2. **Semantics**

Syntax of First-Order logic:

- The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols
- Basic Elements of First-order logic:

Constant	**1, 2, A, John, Mumbai, cat,....**
Variables	**x, y, z, a, b,**
Predicates	**Brother, Father, >,**
Function	**sqrt, LeftLegOf,**
Connectives	**∧, ∨, ¬, ⇒, ⇔**
Equality	**==**
Quantifier	**∀, ∃**

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.

Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky)

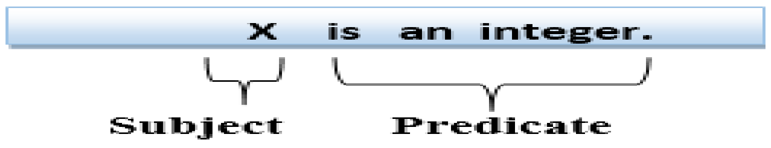
Complex Sentences:
Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- *Subject* : Subject is the main part of the statement.
- *Predicate* : A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts:

- first part x is the subject of the statement
- second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
    - **Universal Quantifier, (for all, everyone, everything)**

All man drink coffee ∀x man(x) → drink (x, coffee).

- **Existential quantifier, (for some, at least one).**

Some boys are intelligent ∃x: boys(x) ∧ intelligent(x)

RESOLUTION:

It is a powerful inference technique which takes two clauses as input and produces a new clause as output. The output class is called resolvent. If the resolvent obtained is a fact , then it is said that we have proved a fact and if resolvent consists of an empty clause(or a contradiction) then we are said to have proved that the set consisting of resolved clauses is unsatisfiable.

Resolution Method in AI:

Resolution method is an inference rule which is used in both Propositional as well as First-order Predicate Logic in different ways. This method is basically used for proving the satisfiability of a sentence. In resolution method, we use Proof by Refutation technique to prove the given statement.

The key idea for the resolution method is to use the knowledge base and negated goal to obtain null clause (which indicates contradiction). Resolution method is also called Proof by Refutation. Since the knowledge base itself is consistent, the contradiction must be introduced by a negated goal. As a result, we have to conclude that the original goal is true.

- Resolution Method in Propositional Logic

In propositional logic, resolution method is the only inference rule which gives a new clause when two or more clauses are coupled together. Using propositional resolution, it becomes easy to make a theorem prover sound and complete for all.

The process followed to convert the propositional logic into resolution method contains the below steps:

- Convert the given axiom into clausal form, i.e., disjunction form.
- Apply and proof the given goal using negation rule.
- Use those literals which are needed to prove.
- Solve the clauses together and achieve the goal.

But, before solving problems using Resolution method, let’s understand two normal forms

1.CNF(Conjunctive Normal Form)

2.DNF(Disjunctive Normal Form)

Conjunctive Normal Form(CNF)

- In propositional logic, the resolution method is applied only to those clauses which are disjunction of literals. There are following steps used to convert into CNF:

1) Eliminate bi-conditional implication by replacing A ⇔ B with (A → B) Λ (B →A)

2) Eliminate implication by replacing A → B with ¬A V B.

3) In CNF, negation(¬) appears only in literals, therefore we move it inwards as:

- ¬ (¬A) ≡ A (double-negation elimination)
- ¬ (A Λ B) ≡ (¬A V ¬B) (De Morgan)
- ¬(A V B) ≡ (¬A Λ ¬B) (De Morgan)
- 4) Finally, using distributive law on the sentences, and form the CNF as:
- (A1 V B1) Λ (A2 V B2) Λ …. Λ (An V Bn).

Note: CNF can also be described as AND of ORS

- **Disjunctive Normal Form (DNF)**

This is a reverse approach of CNF. The process is similar to CNF with the following difference: (A1 Λ B1) V (A2 Λ B2) V…V (An Λ Bn). In DNF, it is OR of ANDS, a sum of products, or a cluster concept, whereas, in CNF, it is ANDs of Ors.

Example of Propositional Resolution

- Consider the following Knowledge Base:

1. The humidity is high or the sky is cloudy.
2. If the sky is cloudy, then it will rain.

3. If the humidity is high, then it is hot.
4. It is not hot.

- Goal: It will rain.

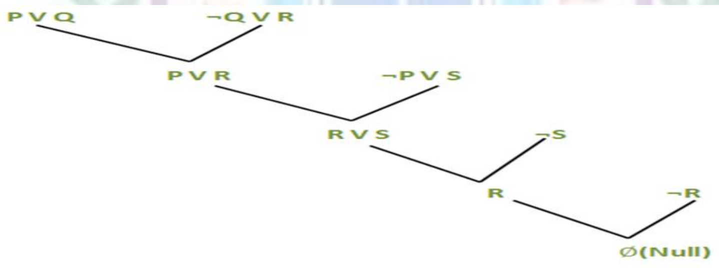
Use propositional logic and apply resolution method to prove that the goal is derivable from the given knowledge base.

Solution: Let's construct propositions of the given sentences one by one:

1. Let, P: Humidity is high.
   Q: Sky is cloudy.
   It will be represented as P V Q.
2. Q: Sky is cloudy. …from(1)
   Let, R: It will rain.
   It will be represented as b Q → R.
3. P: Humidity is high. …from(1)
   Let, S: It is hot.
   It will be represented as P → S.
4. ¬S: It is not hot.

Applying resolution method:

In (2), Q → R will be converted as (¬Q V R)
In (3), P → S will be converted as (¬P V S)
Negation of Goal (¬R): It will not rain.
Finally, apply the rule as shown below:



- After applying Proof by Refutation (Contradiction) on the goal, the problem is solved, and it has terminated with a Null clause (Ø). Hence, the goal is achieved. Thus, It is not raining.

Resolution Method in FOPL/Predicate Logic

Resolution method in FOPL is an uplifted version of propositional resolution method. In FOPL, the process to apply the resolution method is as follows:

- Convert the given axiom into CNF, i.e., a conjunction of clauses. Each clause should be dis-junction of literals.
- Apply negation on the goal given.
- Use literals which are required and prove it.
- Unlike propositional logic, FOPL literals are complementary if one unifies with the negation of other literal.

Example:

{Bird(F(x)) V Loves(G(x), x)} and {¬Loves(a, b) V ¬Kills(a, b)}

Eliminate the complementary literals Loves(G(x),x) and Loves(a,b)) with θ ={a/G(x), v/x} to give the following output clause:

- {Bird(F(x)) V ¬Kills(G(x),x)}
- The rule applied on the following example is called Binary Resolution as it has solved exactly two literals. But, binary resolution is not complete. An alternative approach is to extend the factoring i.e., to remove redundant literals to the first order case. Thus, the combination of binary resolution and factoring is complete.
- Eliminate the complementary literals Loves(G(x),x) and Loves(a,b)) with θ ={a/G(x), v/x} to give the following output clause:

  {Bird(F(x)) V ¬Kills(G(x),x)}
- The rule applied on the following example is called Binary Resolution as it has solved exactly two literals. But, binary resolution is not complete. An alternative approach is to extend the factoring i.e., to remove redundant literals to the first order case. Thus, the combination of binary resolution and factoring is complete.

Standardize variables: If two sentences use same variable, it is required to change the name of one variable. This step is taken so as to remove the confusion when the quantifiers will be dropped
Example: { ∀x: A(x) V ∀x: B(x)}

- Skolemize: It is the process of removing existential quantifier through elimination.
- Drop universal quantifiers: If we are on this step, it means all remaining variables must be universally quantified. Drop the quantifier.
- Distribute V over Λ: Here, the nested conjunction and disjunction are flattened.

Example of FOPL resolution
Consider the following knowledge base:

1. Gita likes all kinds of food.
2. Mango and chapati are food.
3. Gita eats almond and is still alive.
4. Anything eaten by anyone and is still alive is food.

- Goal: Gita likes almond.

Solution: Convert given sentences into FOPL as:

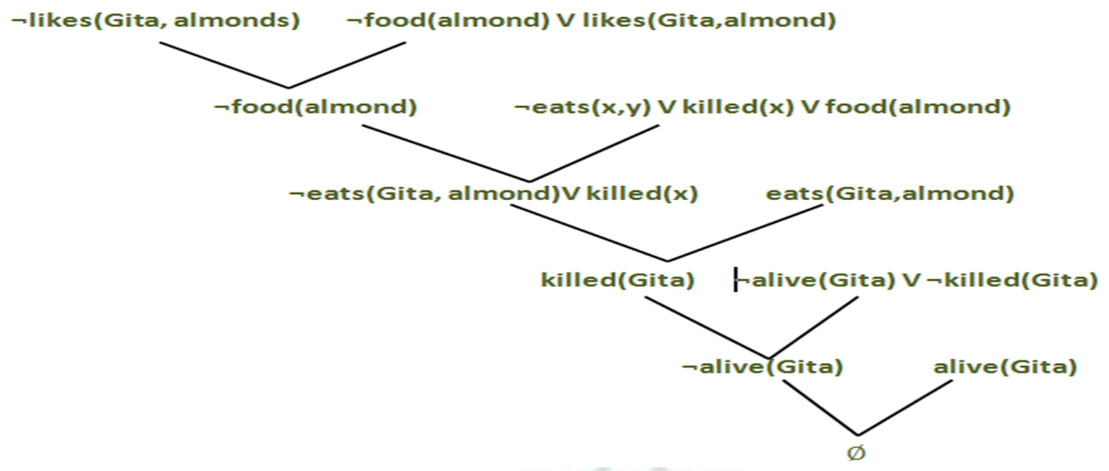
- Let, x be the light sleeper.

1. ∀x: food(x) → likes(Gita,x)
2. food(Mango),food(chapati)
3. ∀x∀y: eats(x,y) Λ ¬ killed(x → food(y)
4. eats(Gita, almonds) Λ alive(Gita)
5. ∀x: ¬killed(x) → alive(x)
6. ∀x: alive(x) → ¬killed(x)

Goal: likes(Gita, almond)
Negated goal: ¬likes(Gita, almond)

- Now, rewrite in CNF form:

1. ¬food(x) V likes(Gita, x)
2. food(Mango),food(chapati)
3. ¬eats(x,y) V killed(x) V food(y)
4. eats(Gita, almonds), alive(Gita)
5. killed(x) V alive(x)
6. ¬alive(x) V ¬killed(x)


¬likes(Gita, almonds)
¬food(almond) V likes(Gita,almond)
¬food(almond)
¬eats(x,y) V killed(x) V food(almond)
¬eats(Gita, almond)V killed(x)
eats(Gita,almond)
killed(Gita)
¬alive(Gita) V ¬killed(Gita)
¬alive(Gita)
alive(Gita)
Ø

Hence, we have achieved the given goal with the help of Proof by Contradiction. Thus, it is proved that Gita likes almond.

Example:

1. John likes all kind of food.
2. Apple and vegetable are food
3. Anything anyone eats and not killed is food.
4. Anil eats peanuts and still alive
5. Harry eats everything that Anil eats.

Prove by resolution that:

John likes peanuts.

SEMANTIC NETWORK

Basic idea behind the semantic network is that meaning of the concept is derived from its relationship with other concepts, and that , the information is stored by the interconnecting nodes with labelled arcs .

Example:

Every human, animal and birds are living things who can breathe and eat. All birds can fly. Every man and woman are human who have two legs. A cat has fur and is an animal. All animals have skin and can move. A giraffe is an animal and has long legs and is tall. A parrot is a bird and is green in colour. John is a man.

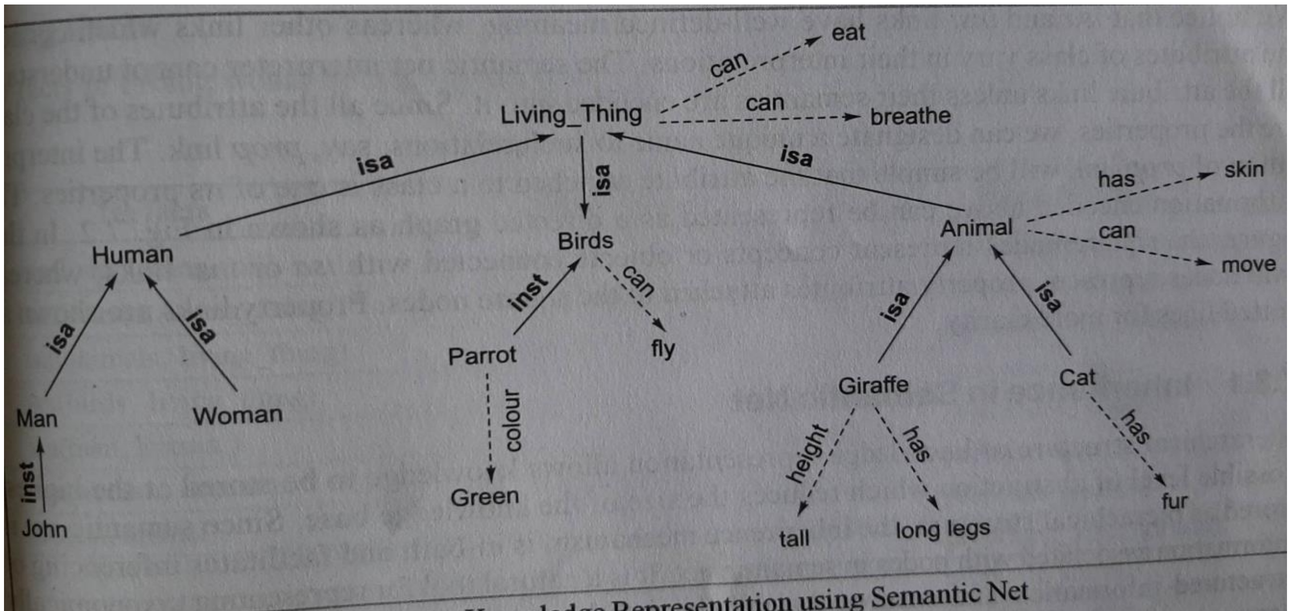


Figure 7.1 Knowledge Representation using Semantic Net

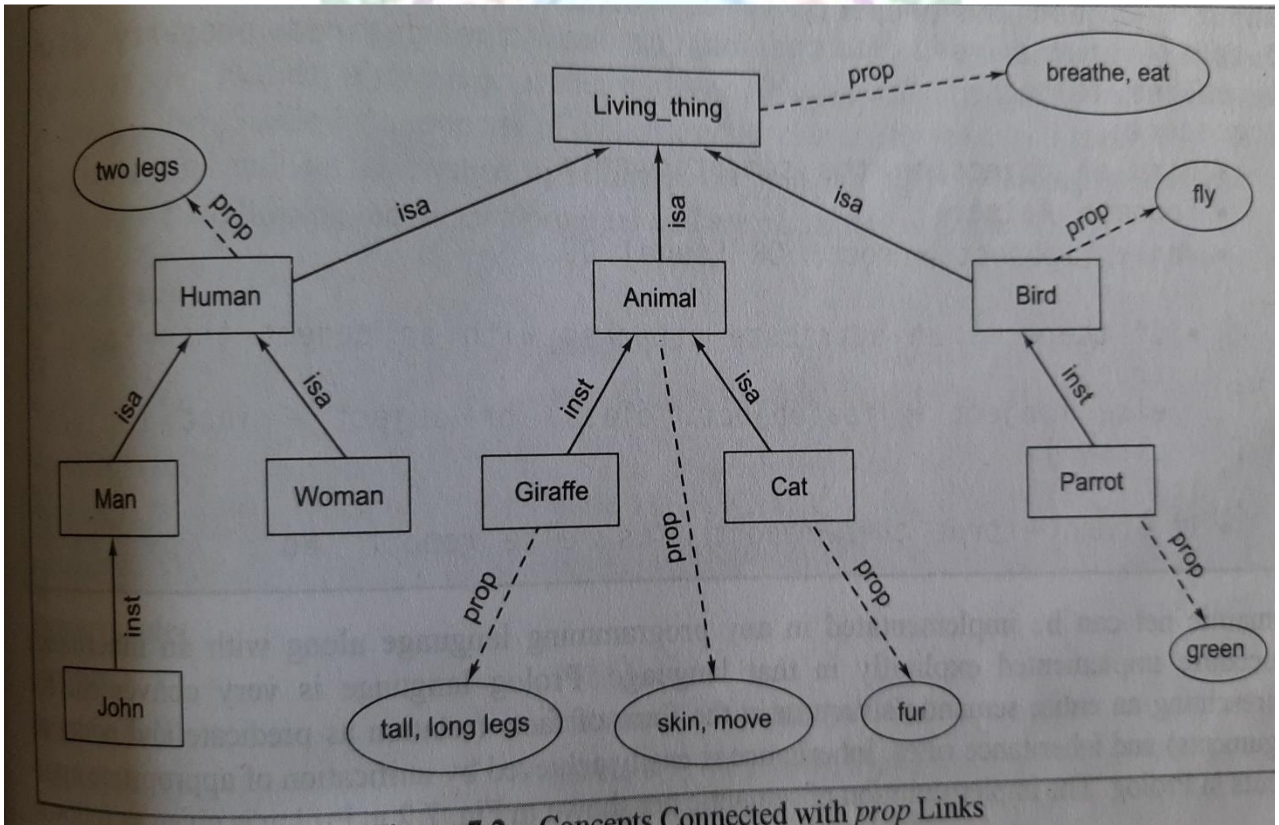


Figure 7.2 Concepts Connected with *prop* Links

- *Isa:* this relation connects two classes, where once concept is a kind of subclass of another concept.
- *Inst:* this relation relates specific members of a class . Such as John is instance of man

- *Property relations:* relations such as {colour, height..} represented with dotted lines pointing from concept to property

Inheritance in Semantic Net

- Since semantic net is stored as hierarchal structure , the inheritance mechanismis in-built and facilitates inferencing of information associated with nodes in semantic net.
- It also helps us to maintain the consistency of knowledge base by adding new concepts and members to existing ones
- PROLOG languages is very convenient for representing an entire sematic structure in the form of facts and inheritance rules

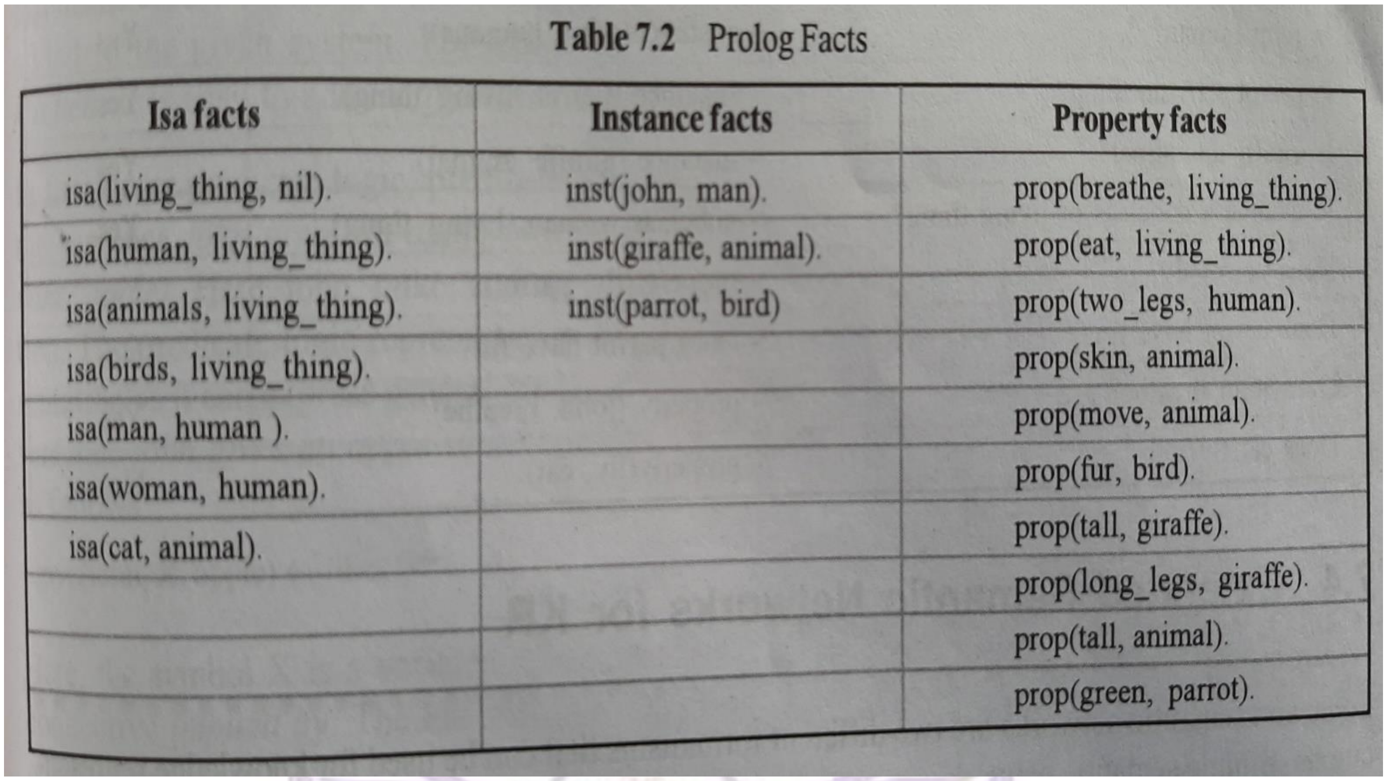


Table 7.2 Prolog Facts

Isa facts	Instance facts	Property facts
isa(living_thing, nil).	inst(john, man).	prop(breathe, living_thing).
isa(human, living_thing).	inst(giraffe, animal).	prop(eat, living_thing).
isa(animals, living_thing).	inst(parrot, bird)	prop(two_legs, human).
isa(birds, living_thing).		prop(skin, animal).
isa(man, human).		prop(move, animal).
isa(woman, human).		prop(fur, bird).
isa(cat, animal).		prop(tall, giraffe).
		prop(long_legs, giraffe).
		prop(tall, animal).
		prop(green, parrot).

PLANNING

- It is the key abilities of intelligent systems
- It refers to the process of computing several steps of problem solving before executing any of them
- It increases the autonomy and flexibility of systems by constructing sequence of actions that help them in achieving the goals.
- It combines two major areas of AI:

Search and logic

- Planner is a program that searches for a solution or one that proves the existence of a solution

- It proves to be useful as a problem solving techniques in case of non-decomposable problems.

Planning System Components

Simple problem solving tasks basically involve the following tasks:

1. Choose the best rule based upon heuristics.
2. Apply this rule to create a new state.
3. Detect when a solution is found.
4. Detect dead ends so that they can be avoided.

Choice of best rule

Methods used involve

- finding the differences between the current states and the goal states and then
- choosing the rules that reduce these differences most effectively.
- Means end analysis good example of this.

Example:

If we wish to travel by car to visit a friend

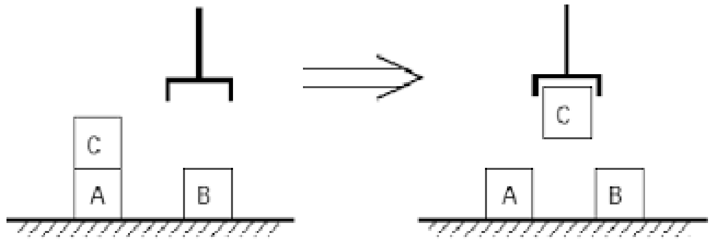
- the first thing to do is to fill up the car with fuel.
- If we do not have a car then we need to acquire one.

The largest difference must be tackled first

Blocks World Planning

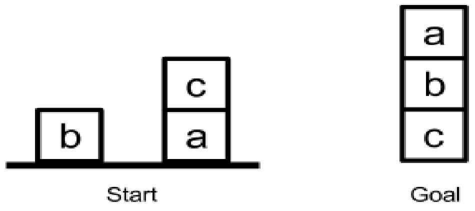
The world consists of:

- A flat surface such as a tabletop
- An adequate set of identical blocks which are identified by letters.
- The blocks can be stacked one on one to form towers of apparently unlimited height.
- The stacking is achieved using a robot arm which has fundamental operations and states which can be assessed using logic and combined using logical operations.
- The robot can hold one block at a time and only one block can be moved at a time.



We shall use the four actions:

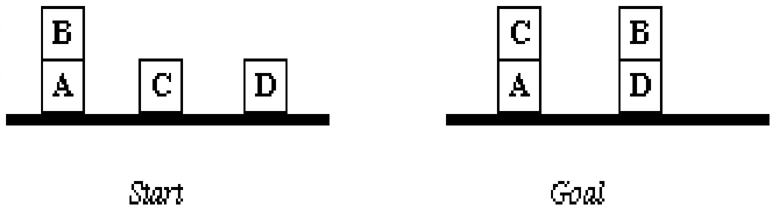
- UNSTACK(A,B)
  -- pick up clear block A from block B;
- STACK(A,B)
  -- place block A using the arm onto clear block B;
- PICKUP(A)
  -- lift clear block A with the empty arm;
- PUTDOWN(A)
  -- place the held block A onto a free space on the table



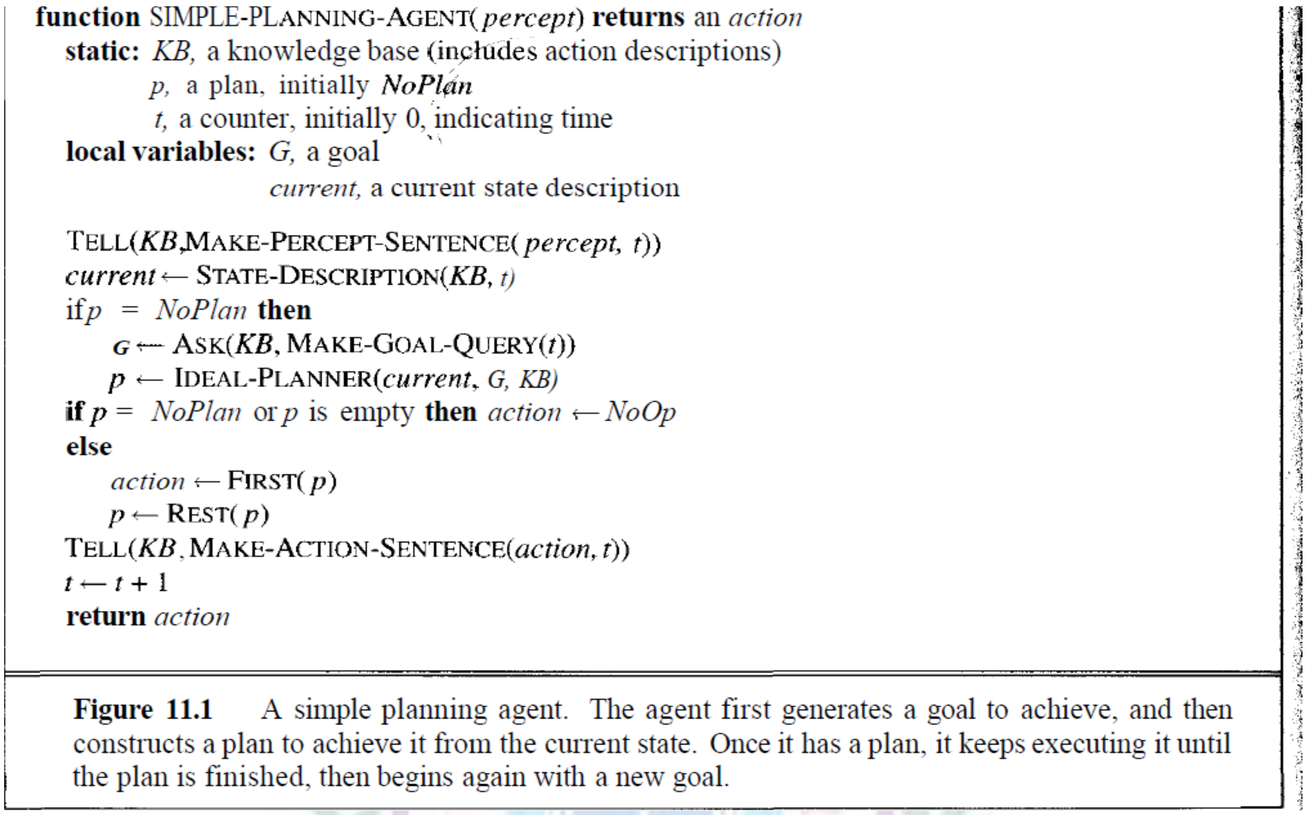
And the five predicates:
ON(A,B)
-- block A is on block B.
ONTABLE(A)
-- block A is on the table.
CLEAR(A)
-- block A has nothing on it.
HOLDING(A)
-- the arm holds block A.
ARMEMPTY
-- the arm holds nothing.

Goal Stack Planning

- **Basic Idea** to handle interactive compound goals uses goal stacks, Here the stack contains :
- goals,
- operators -- ADD, DELETE and PREREQUISITE lists
- a database maintaining the current situation for each operator used.



We can describe the *start* state:
ON(*B*,*A*)^ONTABLE(*A*)^ONTABLE(*C*)^ONTABLE(*D*)^ ARMEMPTY
and *goal* state:
ON(*C*, *A*) ^ON(*B*,*D*) ^ ONTABLE(*A*) ^ONTABLE(*D*)



```
function SIMPLE-PLANNING-AGENT(percept) returns an action
   static: KB, a knowledge base (includes action descriptions)
           p, a plan, initially NoPlan
           t, a counter, initially 0, indicating time
   local variables: G, a goal
                    current, a current state description

   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
   current ← STATE-DESCRIPTION(KB, t)
   if p = NoPlan then
      G ← ASK(KB, MAKE-GOAL-QUERY(t))
      p ← IDEAL-PLANNER(current, G, KB)
   if p = NoPlan or p is empty then action ← NoOp
   else
      action ← FIRST(p)
      p ← REST(p)
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

Figure 11.1 A simple planning agent. The agent first generates a goal to achieve, and then constructs a plan to achieve it from the current state. Once it has a plan, it keeps executing it until the plan is finished, then begins again with a new goal.

Intelligent agents must operate in the world. They are not simply passive reasons (Knowledge Representation, reasoning under uncertainty) or problem solvers (Search), they must also act on the world.

We want intelligent agents to act in “intelligent ways”. Taking purposeful actions, predicting the expected effect of such actions, composing actions together to achieve complex goals. E.g. if we have a robot we want robot to decide what to do; how to act to achieve our goals.

Planning Problem

How to change the world to suit our needs

Critical issue: we need to reason about what the world will be like after doing a few actions, not just what it is like now

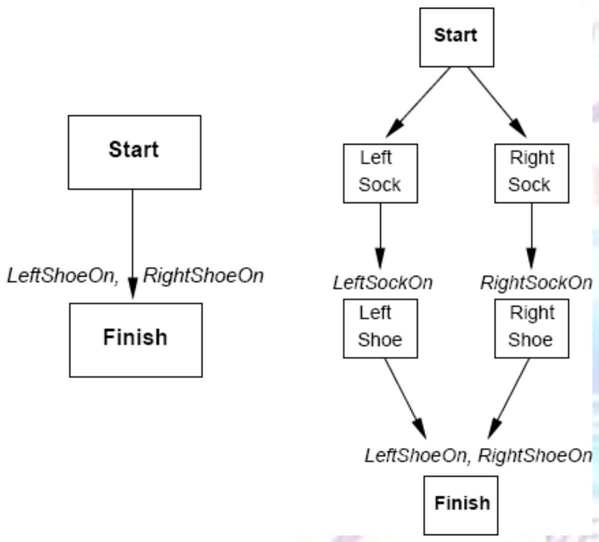
GOAL: Craig has coffee

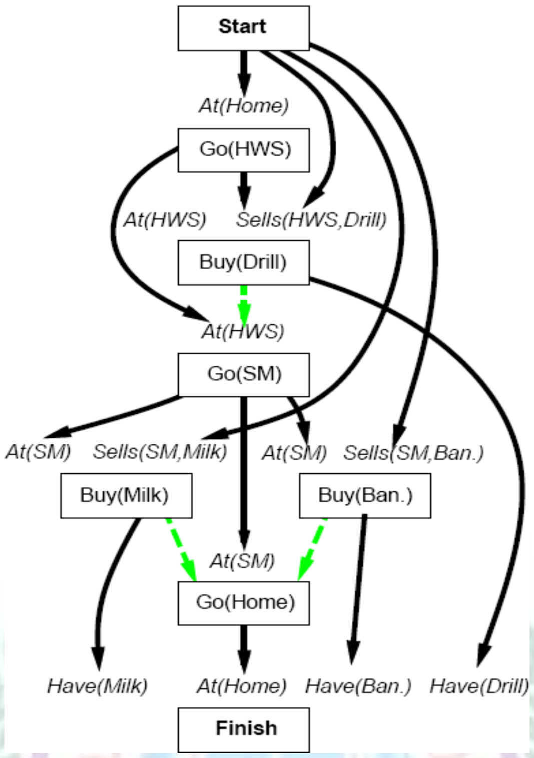
CURRENTLY: robot in mailroom, has no coffee, coffee not made, Craig in office etc.

TO DO: goto lounge, make coffee

PARTIAL ORDER PLAN

- A partially ordered collection of steps
  - Start step has the initial state description and its effect
  - Finish step has the goal description as its precondition
  - Causal links from outcome of one step to precondition of another step
  - Temporal ordering between pairs of steps
- An open condition is a precondition of a step not yet causally linked
- A plan is ***complete*** if every precondition is achieved
- A precondition is ***achieved*** if it is the effect of an earlier step and no possibly intervening step undoes it

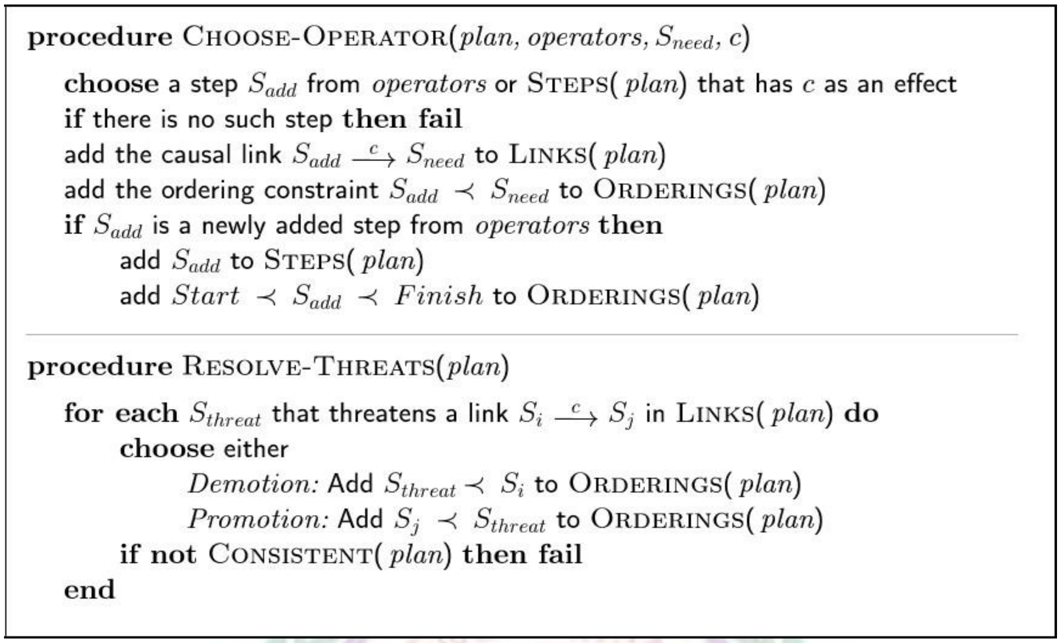




Partial Order Plan Algorithm

```
function POP(initial, goal, operators) returns plan
  plan ← MAKE-MINIMAL-PLAN(initial, goal)
  loop do
     if SOLUTION?(plan) then return plan
     $S_{need}$, c ← SELECT-SUBGOAL(plan)
     CHOOSE-OPERATOR(plan, operators, $S_{need}$, c)
     RESOLVE-THREATS(plan)
  end

function SELECT-SUBGOAL(plan) returns $S_{need}$, c
  pick a plan step $S_{need}$ from STEPS(plan)
     with a precondition c that has not been achieved
  return $S_{need}$, c
```



```
procedure CHOOSE-OPERATOR($plan$, $operators$, $S_{need}$, $c$)
    choose a step $S_{add}$ from $operators$ or STEPS($plan$) that has $c$ as an effect
    if there is no such step then fail
    add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS($plan$)
    add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS($plan$)
    if $S_{add}$ is a newly added step from $operators$ then
        add $S_{add}$ to STEPS($plan$)
        add $Start \prec S_{add} \prec Finish$ to ORDERINGS($plan$)

procedure RESOLVE-THREATS($plan$)
    for each $S_{threat}$ that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS($plan$) do
        choose either
            Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS($plan$)
            Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS($plan$)
        if not CONSISTENT($plan$) then fail
    end
```

Stanford Research Institute Problem Solver (STRIPS)

STRIPS is a classical planning language, representing plan components as states, goals, and actions, allowing algorithms to parse the logical structure of the planning problem to provide a solution.

In STRIPS, state is represented as a conjunction of positive literals. Positive literals may be a propositional literal (e.g., Big ^ Tall) or a first-order literal (e.g., At(Billy, Desk)). The positive literals must be grounded – may not contain a variable (e.g., At(x, Desk)) – and must be function-free – may not invoke a function to calculate a value (e.g., At(Father(Billy), Desk)). Any state conditions that are not mentioned are assumed false.

The goal is also represented as a conjunction of positive, ground literals. A state satisfies a goal if the state contains all of the conjuncted literals in the goal; e.g., Stacked ^ Ordered ^ Purchased satisfies Ordered ^ Stacked.

Actions (or operators) are defined by action schemas, each consisting of three parts:

- The action name and any parameters.
- Preconditions which must hold before the action can be executed. Preconditions are represented as a conjunction of function-free, positive literals. Any variables in a precondition must appear in the action‟s parameter list.
- Effects which describe how the state of the environment changes when the action is executed. Effects are represented as a conjunction of function-free literals. Any

variables in a precondition must appear in the action‟s parameter list. Any world state not explicitly impacted by the action schema‟s effect is assumed to remain unchanged.

The following, simple action schema describes the action of moving a box from location x to location y:

Action: *MoveBox*(*x*, *y*)
Precond: *BoxAt*(*x*)
Effect: *BoxAt*(*y*), ¬ *BoxAt*(*x*)

If an action is applied, but the current state of the system does not meet the necessary preconditions, then the action has no effect. But if an action is successfully applied, then any positive literals, in the effect, are added to the current state of the world; correspondingly, any negative literals, in the effect, result in the removal of the corresponding positive literals from thestate of the world.

For example, in the action schema above, the effect would result in the proposition BoxAt(*y*) being added to the known state of the world, while BoxAt(*x*) would be *removed* from the known state of the world. (Recall that state only includes positive literals, so a negation effect results in the *removal* of positive literals.) Note also that positive effects can not get duplicated in state; likewise, a negative of a proposition that is not currently in state is simply ignored. For example,if *Open*(*x*) was not previously part of the state, ¬ *Open*(*x*) would have no effect.

A STRIPS problem includes the complete (but relevant) initial state of the world, the goal state(s), and action schemas. A STRIPS algorithm should then be able to accept such a problem, returning a solution. The solution is simply an action sequence that, when applied to the initial state, results in a state which satisfies the goal.

STRIPS Planning Algorithm

As previously referenced, STRIPS began as an automated planning algorithm and has double- meaning to describe the language (described above) used to provide input to that algorithm. While the algorithm does not scale well to real-world problems, it, like the language, serves as a foundational starting point to developing and understanding more powerful automated planning algorithms. The STRIPS algorithm [3] is found below, followed by a brief commentary:

```
STRIPS(A, s, g)
p = empty plan
loop…
if s satisfies g then return p
a = [an applicable action in A, relevant for g]
if a = null, then return failure
p' = STRIPS(A, s, precond(a))
if p' = failure, then return failure
s = apply p' to s
```

```
s = apply a to s
p = p + p' + a
```

In the above STRIPS algorithm, *A* represents all of the possible, grounded actions (i.e., action schemas with variables replaced with values), while *s* is the current state, and *g* is the goal state. Critically important to note is that this algorithm is a *backward-search* algorithm. In other words, the goal state of the planning problem is fed into the algorithm as *s*, while the initial state of the planning problem is provided as *g*. The returned solution, if any, simply needs to be reversed to provide the solution to the planning problem. Simply put, it works backwards from the goal to figure out a logical path to get there.

Explanation

Most expert systems have explanation facilities that allow the user to ask questions- why and how it reached some conclusion.The questions are answered by referring to the system goals, the rules being used, and existing problem solving. The rules typically reflect empirical or "compiled" knowledge. They are codesof an expert's rule of thumb, not the expert's deeper understanding.

Example:

Dialog with an expert system designed to give advice on car problems.

System	Does the car start?
User	No.
System	Does the engine turn over?
User	yes
System	Do you smell gas?
User	yes
System	Recommendation-wait 5 minutes and try again.
User	Why?
System	I used the rule: If not start, and engine_turn_over, and smell_gas Then recommendation is „wait 5 minutes and try again"
User	

Note: The rule gives the correct advice for a flooded car. And knows the questions to be ask to determine if the car is flooded, but it does not contain the knowledge of what a flooded car is and why waiting will help.

HIERARCHICAL PLANNING

- In order to solve hard problems, a problem solver may have to generate long plans.
- It is important to be able to eliminate some of the details of the problem until a solution that addresses the main issues is found.
- Then an attempt can make to fill in the appropriate details.
- Early attempts to do this involved the use of macro operators, in which larger operators were built from smaller ones.
- In this approach, no details eliminated from actual descriptions of the operators.

ABSTRIPS A better approach developed in ABSTRIPS systems which actually planned in a hierarchy of abstraction spaces, in each of which preconditions at a lower level of abstraction ignored. ABSTRIPS approach is as follows:

- First solve the problem completely, considering only preconditions whose criticality value is the highest possible.
- These values reflect the expected difficulty of satisfying the precondition.
- To do this, do exactly what STRIPS did, but simply ignore the preconditions of lower than peak criticality.
- Once this done, use the constructed plan as the outline of a complete plan and consider preconditions at the next-lowest criticality level.
- Augment the plan with operators that satisfy those preconditions.
- Because this approach explores entire plans at one level of detail before it looks at the lower-level details of any one of them, it has called length-first approach.

The assignment of appropriate criticality value is crucial to the success of this hierarchical planning method. Those preconditions that no operator can satisfy are clearly the most critical. Example, solving a problem of moving the robot, for applying an operator, PUSH-THROUGH DOOR, the precondition that there exist a door big enough for the robot to get through is of high criticality since there is nothing we can do about it if it is not true.

UNIT-3: Expert Systems, Reasoning with Uncertainty
Expert System and Applications: Introduction, Phases in Building Expert Systems, Expert System Architecture, Applications.
Uncertainty - Basic probability, Bayes rule, Belief networks, Inference in Bayesian Networks, Fuzzy sets, and fuzzy logic: Fuzzy logic system architecture, membership function.
Decision making- Utility theory, utility functions.

EXPERT SYSTEMS

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

To solve expert-level problems, expert systems will need efficient access to a substantial domain knowledge base, and a reasoning mechanism to apply the knowledge to the problems they are given. Usually they will also need to be able to explain, to the users who rely on them, how they have reached their decisions.

They will generally build upon the ideas of knowledge representation, production rules, search, and so on, that we have already covered.

Often we use an expert system shell which is an existing knowledge independent framework into which domain knowledge can be inserted to produce a working expert system. We can thus avoid having to program each new system from scratch.

Typical Tasks for Expert Systems

There are no fundamental limits on what problem domains an expert system can be built to deal with. Some typical existing expert system tasks include:

1. The interpretation of data
   Such as sonar data or geophysical measurements
2. Diagnosis of malfunctions
   Such as equipment faults or human diseases
3. Structural analysis or configuration of complex objects

   Such as chemical compounds or computer systems

4. Planning sequences of actions
   Such as might be performed by robots
5. Predicting the future
   Such as weather, share prices, exchange rates

However, these days, “conventional” computer systems can also do some of these things

Characteristics of Expert Systems

Expert systems can be distinguished from conventional computer systems in that:

1. They simulate human reasoning about the problem domain, rather than simulating the domain itself.

2. They perform reasoning over representations of human knowledge, in addition to doing numerical calculations or data retrieval. They have corresponding distinct modules referred to as

the inference engine and the knowledge base.

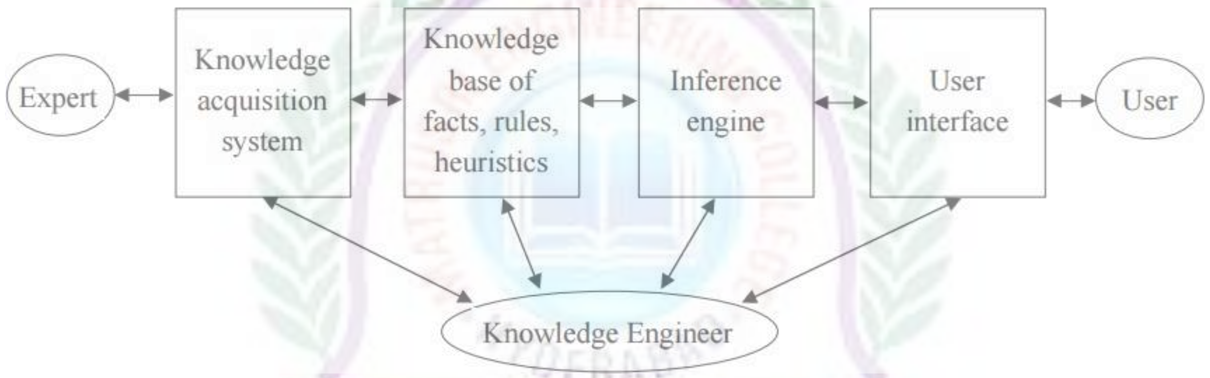
3. Problems tend to be solved using heuristics (rules of thumb) or approximate methods or probabilistic methods which, unlike algorithmic solutions, are not guaranteed to result in a correct or optimal solution.

4. They usually have to provide explanations and justifications of their solutions or recommendations in order to convince the user that their reasoning is correct.

Note that the term Intelligent Knowledge Based System (IKBS) is sometimes used as a synonym for Expert System.

EXPERT SYSTEM ARCHITECTURE

The process of building expert systems is often called knowledge engineering. The knowledge engineer is involved with all components of an expert system:


Expert
Knowledge acquisition system
Knowledge base of facts, rules, heuristics
Inference engine
User interface
User
Knowledge Engineer

Building expert systems is generally an iterative process. The components and their interaction will be refined over the course of numerous meetings of the knowledge engineer with the experts and users. We shall look in turn at the various components.

Knowledge Acquisition

The knowledge acquisition component allows the expert to enter their knowledge or expertise into the expert system, and to refine it later as and when required.

Historically, the knowledge engineer played a major role in this process, but automated systems that allow the expert to interact directly with the system are becoming increasingly common.

The knowledge acquisition process is usually comprised of three principal stages:

1. Knowledge elicitation is the interaction between the expert and the knowledge engineer/program to elicit the expert knowledge in some systematic way.

2. The knowledge thus obtained is usually stored in some form of human friendly intermediate representation.

3. The intermediate representation of the knowledge is then compiled into an executable form (e.g. production rules) that the inference engine can process.

In practice, much iteration through these three stages is usually required!

Knowledge Elicitation

The knowledge elicitation process itself usually consists of several stages:

1. Find as much as possible about the problem and domain from books, manuals, etc. In particular, become familiar with any specialist terminology and jargon.

2. Try to characterize the types of reasoning and problem solving tasks that the system will be required to perform.

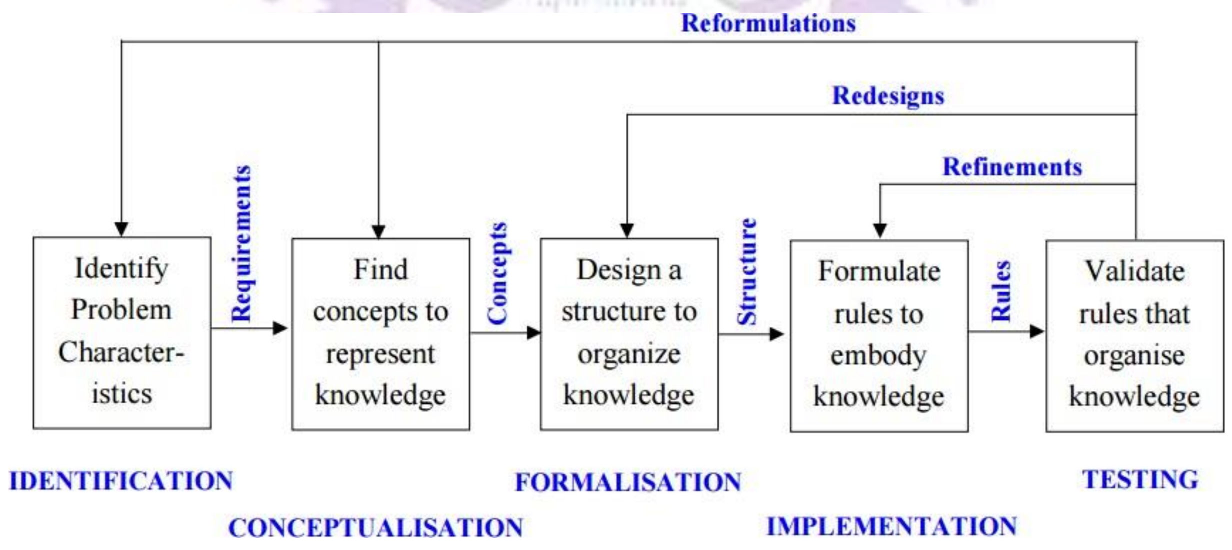
3. Find an expert (or set of experts) that is willing to collaborate on the project. Sometimes experts are frightened of being replaced by a computer system!

4. Interview the expert (usually many times during the course of building the system). Find out how they solve the problems your system will be expected to solve. Have them check and refine your intermediate knowledge representation.

This is a time intensive process, and automated knowledge elicitation and machine learning techniques are increasingly common modern alternatives.

Stages of Knowledge Acquisition

The iterative nature of the knowledge acquisition process can be represented in the following diagram.


Reformulations
Redesigns
Refinements
Identify Problem Character-istics
Requirements
Find concepts to represent knowledge
Concepts
Design a structure to organize knowledge
Structure
Formulate rules to embody knowledge
Rules
Validate rules that organise knowledge
IDENTIFICATION
CONCEPTUALISATION
FORMALISATION
IMPLEMENTATION
TESTING

Levels of Knowledge Analysis

Knowledge identification: Use in depth interviews in which the knowledge engineer encourages the expert to talk about how they do what they do. The knowledge engineer should understand the domain well enough to know which objects and facts need talking about.

Knowledge conceptualization: Find the primitive concepts and conceptual relations of the problem domain.

Epistemological analysis: Uncover the structural properties of the conceptual knowledge, such as taxonomic relations (classifications).

Logical analysis: Decide how to perform reasoning in the problem domain. This kind of knowledge can be particularly hard to acquire.

Implementation analysis: Work out systematic procedures for implementing and testing the system.

Capturing Tacit/Implicit Knowledge

One problem that knowledge engineers often encounter is that the human experts use tacit/implicit knowledge (e.g. procedural knowledge) that is difficult to capture.

There are several useful techniques for acquiring this knowledge:

1. **Protocol analysis**: Tape-record the expert thinking aloud while performing their role and later analyze this. Break down their protocol/account into the smallest atomic units of thought, and let these become operators.

2. **Participant observation**: The knowledge engineer acquires tacit knowledge through practical domain experience with the expert.

3. **Machine induction**: This is useful when the experts are able to supply examples of the results of their decision making, even if they are unable to articulate the underlying knowledge or reasoning process.

Which is/are best to use will generally depend on the problem domain and the expert.

Representing the Knowledge

We have already looked at various types of knowledge representation. In general, the knowledge acquired from our expert will be formulated in two ways:

1. **Intermediate representation** – a structured knowledge representation that the knowledge engineer and expert can both work with efficiently.

2. **Production system** – a formulation that the expert system's inference engine can process efficiently.

It is important to distinguish between:

1. **Domain knowledge** – the expert's knowledge which might be expressed in the form of rules, general/default values, and so on.

2. **Case knowledge** – specific facts/knowledge about particular cases, including any derived knowledge about the particular cases.

The system will have the domain knowledge built in, and will have to integrate this with the different case knowledge that will become available each time the system is used.

Meta Knowledge

Knowledge about knowledge

- Meta knowledge can be simply defined as knowledge about knowledge.
- Meta knowledge is knowledge about the use and control of domain knowledge in an expert system.

Roles in Expert System Development

Three fundamental roles in building expert systems are:

1. Expert - Successful ES systems depend on the experience and application of knowledge that the people can bring to it during its development. Large systems generally require multiple experts.

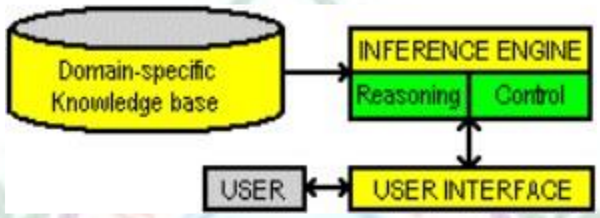
2. Knowledge engineer - The knowledge engineer has a dual task. This person should be able to elicit knowledge from the expert, gradually gaining an understanding of an area of expertise. Intelligence, tact, empathy, and proficiency in specific techniques of knowledge acquisition are all required of a knowledge engineer. Knowledge-acquisition techniques include conducting interviews with varying degrees of structure, protocol analysis, observation of experts at work, and analysis of cases.

On the other hand, the knowledge engineer must also select a tool appropriate for the project and use it to represent the knowledge with the application of the knowledge acquisition facility.

3. User - A system developed by an end user with a simple shell, is built rather quickly an inexpensively. Larger systems are built in an organized development effort. A prototype-oriented iterative development strategy is commonly used. ESs lends themselves particularly well to prototyping.

TYPICAL EXPERT SYSTEM

1. A problem-domain-specific knowledge base that stores the encoded knowledge to support one problem domain such as diagnosing why a car won't start. In a rule-based expert system, the knowledge base includes the if-then rules and additional specifications that control the course of the interview.



2. An inference engine a set of rules for making deductions from the data and that implements the reasoning mechanism and controls the interview process. The inference engine might be generalized so that the same software is able to process many different knowledge bases.

3. The user interface requests information from the user and outputs intermediate and final results. In some expert systems, input is acquired from additional sources such as data bases and sensors.

An expert system shell consists of a generalized inference engine and user interface designed to work with a knowledge base provided in a specified format. A shell often includes tools that help with the design, development and testing of the knowledge base. With the shell approach, expert systems representing many different problem domains may be developed and delivered with the same software environment. .

There are special high level languages used to program expert systems egg PROLOG

The user interacts with the system through a user interface which may use menus, natural language or any other style of interaction). Then an inference engine is used to reason with both the expert knowledge (extracted from our friendly expert) and data specific to the particular problem being solved. The expert knowledge will typically be in the form of a set of IF-THEN rules. The case specific data includes both data provided by the user and partial conclusions (along with certainty measures) based on this data. In a simple forward chaining rule-based system the case specific data will be the elements in working memory.

How an expert system works Car engine diagnosis

1. IF engine_getting_petrol
   AND engine_turns_over
   THEN problem_with_spark_plugs

2. IF NOT engine_turns_over
   AND NOT lights_come_on
   THEN problem_with_battery

3. IF NOT engine_turns_over
   AND lights_come_on
   THEN problem_with_starter
4. IF petrol_in_fuel_tank
   THEN engine_getting_petrol

There are three possible problems with the car:

- problem_with_spark_plugs,
- problem_with_battery,
- problem_with_starter.

The system will ask the user:

- Is it true that there's petrol in the fuel tank?

  Let's say that the answer is yes. This answer would be recorded, so that the user doesn't get asked the same question again. Anyway, the system now has proved that the engine is getting petrol, so now wants to find out if the engine turns over. As the system doesn't yet know whether this is the case, and as there are no rules which conclude this, the user will be asked:

- Is it true that the engine turns over?

Lets say this time the answer is no. There are no other rules which can be used to prove ``problem_with_spark_plugs'' so the system will conclude that this is not the solution to the problem, and will consider the next hypothesis: problem_with_battery. It is true that the engine does not turn over (the user has just said that), so all it has to prove is that the lights don't come one. It will ask the user

- Is it true that the lights come on?

Suppose the answer is no. It has now proved that the problem is with the battery. Some systems might stop there, but usually there might be more than one solution, (e.g., more than one fault with the car), or it will be uncertain which of various solutions is the right one. So usually all hypotheses are considered. It will try to prove ``problem_with_starter'', but given the existing data (the lights come on) the proof will fail, so the system will conclude that the problem is with the battery. A complete interaction with our very simple system might be:

- System: Is it true that there's petrol in the fuel tank?

User: Yes.

System: Is it true that the engine turns over?

User: No.

System Is it true that the lights come on?

User: No.

System: I conclude that there is a problem with battery.

Note that in general, solving problems using backward chaining involves searching through all the possible ways of proving the hypothesis, systematically checking each of them.

Questions

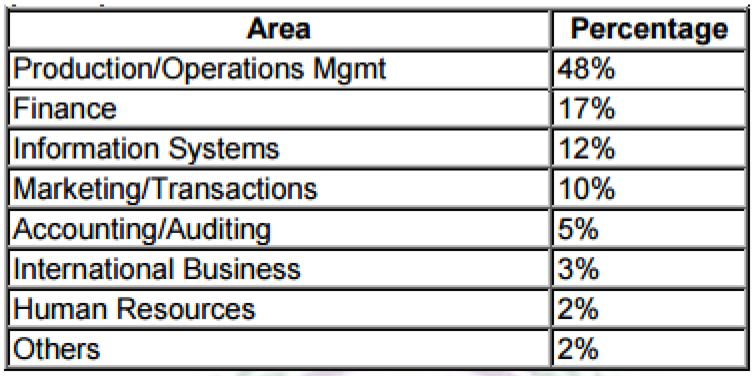
1. ``Briefly describe the basic architecture of a typical expert system, mentioning the function of each of the main components.''

2. ``A travel agent asks you to design an expert system to help people choose where to go on holiday. Design a set of decisions to help you give advice on which holiday to take.

Expert System Use

Expert systems are used in a variety of areas, and are still the most popular developmentalapproach in the artificial intelligence world.

The table below depicts the percentage of expert systems being developed in particular areas:



Area	Percentage
Production/Operations Mgmt	48%
Finance	17%
Information Systems	12%
Marketing/Transactions	10%
Accounting/Auditing	5%
International Business	3%
Human Resources	2%
Others	2%

- Medical screening for cancer and brain tumours
- Matching people to jobs
- Training on oil rigs
- Diagnosing faults in car engines
- Legal advisory systems
- Mineral prospecting

Expert Systems shells

Initially each expert system is build from scratch (LISP). Systems are constructed as a set of declarative representations (mostly rules) combined with an interpreter for those representations. It helps to separate the interpreter from domain-specific knowledge and to create a system that could be used construct new expert system by adding new knowledge corresponding to the new problem domain. The resulting interpreters are called shells. Example of shells is EMYCIN (for Empty MYCIN derived from MYCIN).

Shells − A shell is nothing but an expert system without knowledge base. A shell provides the developers with knowledge acquisition, inference engine, user interface, and explanation facility.For example, few shells are given below −

- ➢ Java Expert System Shell (JESS) that provides fully developed Java API for creating anexpert system.
- ➢ Vidwan, a shell developed at the National Centre for Software Technology, Mumbai in1993. It enables knowledge encoding in the form of IF-THEN rules.

Shells provide greater flexibility in representing knowledge and in reasoning than MYCIN. They support rules, frames, truth maintenance systems and a variety of other reasoning mechanisms.

Early expert system shells provide mechanisms for knowledge representation, reasoning and explanation. Later these tools provide knowledge acquisition. Still expert system shells need to integrate with other programs easily. Expert systems cannot operate in a vacuum. The shells must provide an easy-to-use interface between an expert system written with the shell and programming environment.

UNCERTAINTY

Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write A→B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.

So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

Causes of uncertainty:

Following are some leading causes of uncertainty to occur in the real world.

- Information occurred from unreliable sources.
- Experimental Errors
- Equipment fault
- Temperature variation
- Climate change.

Probabilistic reasoning:

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.

In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI:

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

- Bayes' rule and Bayesian Statistics

As probabilistic reasoning uses probability and related terms, so before understanding probabilistic reasoning, let's understand some common terms:

Probability: Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

$0 \leq P(A) \leq 1$, where P(A) is the probability of an event A.

P(A) = 0, indicates total uncertainty in an event A.

P(A) =1, indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

P(¬A) = probability of a not happening event.

$$P(\neg A) + P(A) = 1.$$

Event: Each possible outcome of a variable is called an event.
Sample space: The collection of all possible events is called sample space.
Random variables: Random variables are used to represent the events and objects in the real world.
Prior probability: The prior probability of an event is probability computed before observing new information.
Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.
Conditional probability:
Conditional probability is a probability of occurring an event when another event has already happened.
Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:
Where $P(A \wedge B)$ = Joint probability of a and B
$P(B)$= Marginal probability of B.
If the probability of A is given and we need to find the probability of B, then it will be given as:
It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of $P(A \wedge B)$ by $P(B)$.

Bayes' Rule :

Bayes' theorem is also known as Bayes' rule, Bayes' law, or Bayesian reasoning, which determines the probability of an event with uncertain knowledge.

In probability theory, it relates the conditional probability and marginal probabilities of two random events.

Bayes' theorem was named after the British mathematician Thomas Bayes. The Bayesian inference is an application of Bayes' theorem, which is fundamental to Bayesian statistics.

It is a way to calculate the value of $P(B|A)$ with the knowledge of $P(A|B)$.

Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.

Example: If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.

Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:
As from product rule we can write:
$P(A \wedge B)= P(A|B) P(B)$ or
Similarly, the probability of event B with known event A:
$P(A \wedge B)= P(B|A) P(A)$
Equating right hand side of both the equations, we will get:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \quad \text{....(a)}$$

The above equation (a) is called as Bayes' rule or Bayes' theorem. This equation is basic of most modern AI systems for probabilistic inference.
It shows the simple relationship between joint and conditional probabilities. Here,
$P(A|B)$ is known as posterior, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.
$P(B|A)$ is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.
$P(A)$ is called the prior probability, probability of hypothesis before considering the evidence
$P(B)$ is called marginal probability, pure probability of an evidence.

In the equation (a), in general, we can write P (B) = P(A)*P(B|Ai), hence the Bayes' rule can be written as:

$$P(A_i \mid B) = \frac{P(A_i) * P(B \mid A_i)}{\sum_{i=1}^{k} P(A_i) * P(B \mid A_i)}$$

Where A1, A2, A3,........, An is a set of mutually exclusive and exhaustive events.

Applying Bayes' rule:

Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(B), and P(A). This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one. Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

$$\mathbf{P(cause \mid effect)} = \frac{P(effect \mid cause)\ P(cause)}{P(effect)}$$

Example-1:

Question: what is the probability that a patient has diseases meningitis with a stiff neck?

Given Data:

A doctor is aware that disease meningitis causes a patient to have a stiff neck, and it occurs 80% of the time. He is also aware of some more facts, which are given as follows:

The Known probability that a patient has meningitis disease is 1/30,000.

The Known probability that a patient has a stiff neck is 2%.

Let a be the proposition that patient has stiff neck and b be the proposition that patient has meningitis. , so we can calculate the following as:

P(a|b) = 0.8

P(b) = 1/30000

P(a)= .02

$$\mathbf{P(b \mid a)} = \frac{P(a \mid b) P(b)}{P(a)} = \frac{0.8 * (\frac{1}{30000})}{0.02} = 0.001333333.$$

Hence, we can assume that 1 patient out of 750 patients has meningitis disease with a stiff neck.

Application of Bayes' theorem in Artificial intelligence:

Following are some applications of Bayes' theorem:

- It is used to calculate the next step of the robot when the already executed step is given.
- Bayes' theorem is helpful in weather forecasting.
- It can solve the Monty Hall problem.

Bayesian networks

- ➢ Represent dependencies among random variables
- ➢ Give a short specification of conditional probability distribution
- ➢ Many random variables are conditionally independent
- ➢ Simplifies computations
- ➢ Graphical representation
- ➢ DAG – causal relationships among random variables
- ➢ Allows inferences based on the network structure

Definition of Bayesian networks

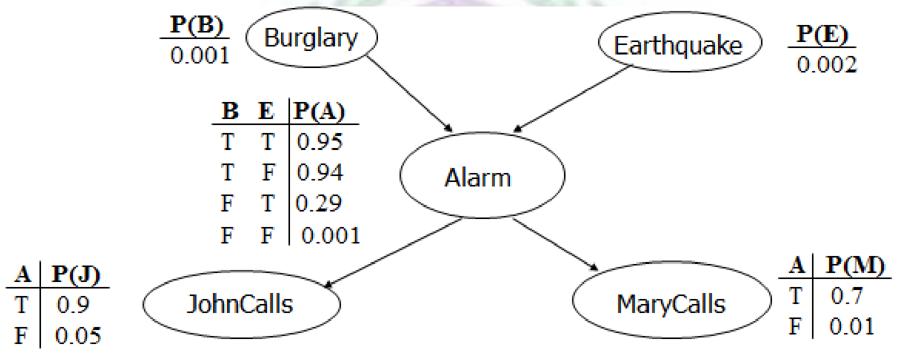
A BN is a DAG in which each node is annotated with quantitative probability information, namely:

- Nodes represent random variables (discrete or continuous)
- Directed links X→Y: X has a direct influence on Y, X is said to be a parent of Y
- each node X has an associated conditional probability table, **P(X$_i$ | Parents(X$_i$))** that quantify the effects of the parents on the node

Example: Weather, Cavity, Toothache, Catch

- Weather, Cavity → Toothache, Cavity → Catch

Example



Conditional probability table

B	E	P(A | B, E) T	P(A | B, E) F
T	T	0.95	0.05
T	F	0.94	0.06
F	T	0.29	0.71
F	F	0.001	0.999

Bayesian network semantics

- Represent a probability distribution
- Specify conditional independence – build the network
- each value of the probability distribution can be computed as:

$$P(X_1=x_1 \wedge \ldots X_n=x_n) = P(x_1,\ldots, x_n) = \Pi_{i=1,n} P(x_i \mid Parents(x_i))$$

where Parents(xi) represent the specific values of Parents(Xi)

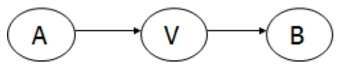
Building the network

$$\mathbf{P(X_1=x_1 \wedge \ldots X_n=x_n) = P(x_1,\ldots, x_n) =}$$

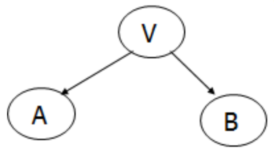
$$\mathbf{P(x_n \mid x_{n-1},\ldots, x_1) * P(x_{n-1},\ldots, x_1) = \ldots =}$$

$$\mathbf{P(x_n \mid x_{n-1},\ldots, x_1) * P(x_{n-1} \mid x_{n-2},\ldots, x_1) * \ldots P(x_2|x_1) * P(x_1) = \Pi_{i=1,n} P(x_i \mid x_{i-1},\ldots, x_1)}$$

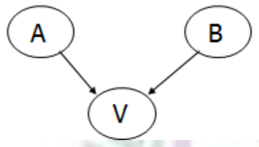
- We can see that P(Xi | Xi-1,…, X1) = P(xi | Parents(Xi)) if Parents(Xi) □ { Xi-1,…, X1}
- The condition may be satisfied by labelling the nodes in an order consistent with a DAG
- Intuitively, the parents of a node Xi must be all the nodes Xi-1,, X1 which have a direct influence on Xi.
- Pick a set of random variables that describe the problem
- Pick an ordering of those variables
- while there are still variables repeat
- choose a variable Xi and add a node associated to Xi
- assign Parents(Xi) □ a minimal set of nodes that already exists in the network such that the conditional independence property is satisfied
- define the conditional probability table for Xi
- Because each node is linked only to previous nodes □ DAG
- P(MaryCalls | JohnCals, Alarm, Burglary, Earthquake) = P(MaryCalls | Alarm)



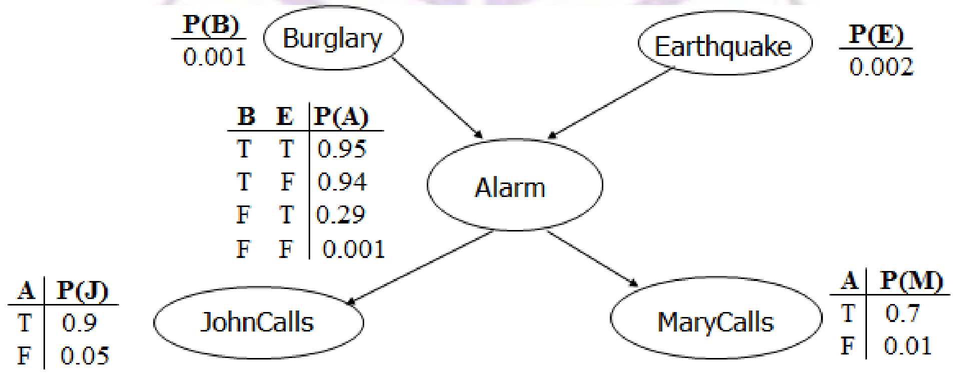
P(A ∧ V ∧ B) = P(A) * P(V|A) * P(B|V)



P(A ∧ V ∧ B) = P(V) * P(A|V) * P(B|V)



P(A ∧ V ∧ B) = P(A) * P(B) * P(V|A,B)



P(B)
0.001

P(E)
0.002

B	E	P(A)
T	T	0.95
T	F	0.94
F	T	0.29
F	F	0.001

A	P(J)
T	0.9
F	0.05

A	P(M)
T	0.7
F	0.01

P(J ∧ M ∧ A ∧~B ∧~E) =

P(J|A)* P(M|A)*P(A|~B ∧~E)*P(~B) ∧P(~E)= 0.9 * 0.7 * 0.001 * 0.999 * 0.998 = 0.00062

P(A|B) = P(A|B,E) *P(E|B) + P(A| B,~E)*P(~E|B) = P(A|B,E) *P(E) + P(A| B,~E)*P(~E)

= 0.95 * 0.002 + 0.94 * 0.998 = 0.94002

Different types of inferences

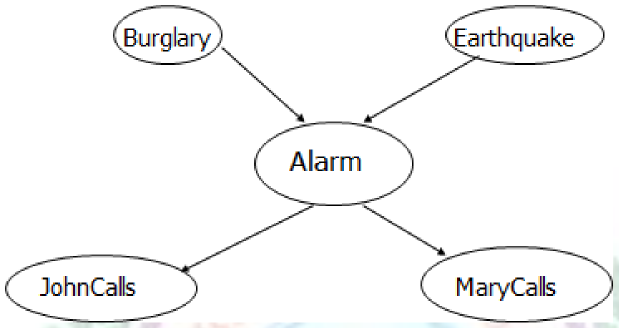
Diagnosis inferences (effect → cause)

P(Burglary | JohnCalls)

Causal inferences (cause → effect)

P(JohnCalls |Burglary),

P(MaryCalls | Burgalry)



Intercausal inferences (between cause and common effects)

P(Burglary | Alarm ∧Earthquake)

Mixed inferences

P(Alarm | JohnCalls ∧ ~Earthquake) → diag + causal

P(Burglary | JohnCalls ∧ ~ Earthquake) → diag + intercausal

FUZZY LOGIC :

Fuzzy Logic (FL) is a method of reasoning that resembles human reasoning. The approach of FL imitates the way of decision making in humans that involves all intermediate possibilities between digital values YES and NO.

The conventional logic block that a computer can understand takes precise input and produces a definite output as TRUE or FALSE, which is equivalent to human‟s YES or NO.

The inventor of fuzzy logic, Lotfi Zadeh, observed that unlike computers, the human decision making includes a range of possibilities between YES and NO, such as −

CERTAINLY YES
POSSIBLY YES
CANNOT SAY
POSSIBLY NO
CERTAINLY NO

The fuzzy logic works on the levels of possibilities of input to achieve the definite output.

Fuzzy logic is useful for commercial and practical purposes.

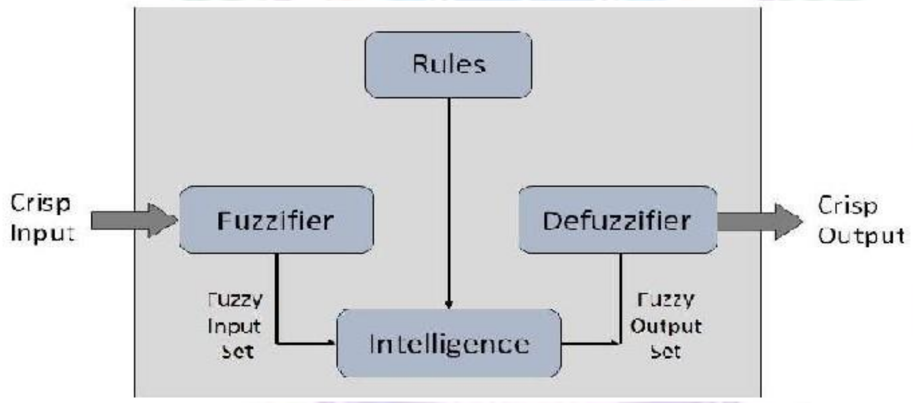
- It can control machines and consumer products.
- It may not give accurate reasoning, but acceptable reasoning.
- Fuzzy logic helps to deal with the uncertainty in engineering.

FUZZY LOGIC SYSTEMS ARCHITECTURE

- Fuzzification Module − It transforms the system inputs, which are crisp numbers, into fuzzy sets. It splits the input signal into five steps such as −

LP	x is Large Positive
MP	x is Medium Positive
S	x is Small
MN	x is Medium Negative
LN	x is Large Negative

- Knowledge Base − It stores IF-THEN rules provided by experts.
- Inference Engine − It simulates the human reasoning process by making fuzzy inference on the inputs and IF-THEN rules.
- Defuzzification Module − It transforms the fuzzy set obtained by the inference engine into a crisp value.



The membership functions work on fuzzy sets of variables.

MEMBERSHIP FUNCTION:

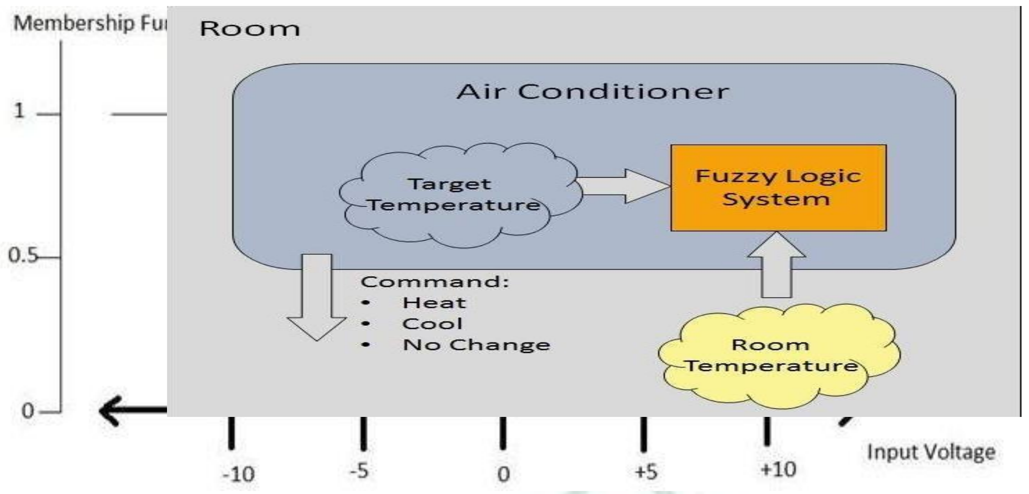
Membership functions allow you to quantify linguistic term and represent a fuzzy set graphically. A membership function for a fuzzy set A on the universe of discourse X is defined as $\mu A:X \rightarrow [0,1]$.

Here, each element of X is mapped to a value between 0 and 1. It is called membership value or degree of membership. It quantifies the degree of membership of the element in X to the fuzzy set A.

- x axis represents the universe of discourse.
- y axis represents the degrees of membership in the [0, 1] interval.

There can be multiple membership functions applicable to fuzzify a numerical value. Simple membership functions are used as use of complex functions does not add more precision in the output.

All membership functions for LP, MP, S, MN, and LN are shown as below −



The triangular membership function shapes are most common among various other membership function shapes such as trapezoidal, singleton, and Gaussian.

Here, the input to 5-level fuzzifier varies from -10 volts to +10 volts. Hence the corresponding output also changes.

Example of a Fuzzy Logic System

Let us consider an air conditioning system with 5-lvel fuzzy logic system. This system adjusts the temperature of air conditioner by comparing the room temperature and the target temperature value.

Algorithm

- Define linguistic variables and terms.
- Construct membership functions for them.
- Construct knowledge base of rules.
- Convert crisp data into fuzzy data sets using membership functions. (fuzzification)
- Evaluate rules in the rule base. (Interface engine)
- Combine results from each rule. (Interface engine)
- Convert output data into non-fuzzy values. (defuzzification)

Logic Development

Step 1: Define linguistic variables and terms

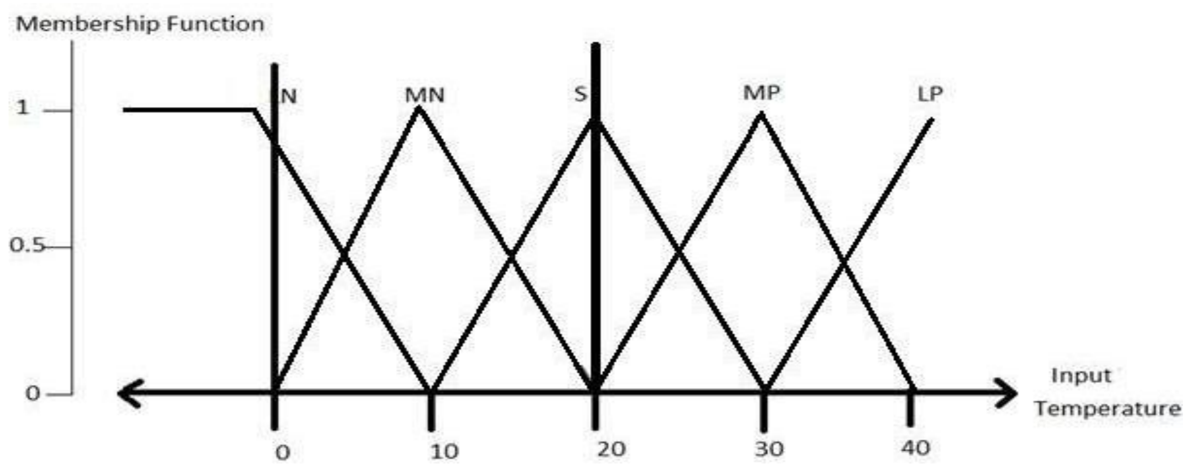
Linguistic variables are input and output variables in the form of simple words or sentences. For room temperature, cold, warm, hot, etc., are linguistic terms.

Temperature (t) = {very-cold, cold, warm, very-warm, hot}

Every member of this set is a linguistic term and it can cover some portion of overall temperature values.

Step 2: Construct membership functions for them

The membership functions of temperature variable are as shown −



Step3: Construct knowledge base rules

Create a matrix of room temperature values versus target temperature values that an air conditioning system is expected to provide.

RoomTemp. /Target	Very_Cold	Cold	Warm	Hot	Very_Hot
Very_Cold	No_Change	Heat	Heat	Heat	Heat
Cold	Cool	No_Change	Heat	Heat	Heat
Warm	Cool	Cool	No_Change	Heat	Heat
Hot	Cool	Cool	Cool	No_Change	Heat
Very_Hot	Cool	Cool	Cool	Cool	No_Change

Build a set of rules into the knowledge base in the form of IF-THEN-ELSE structures.

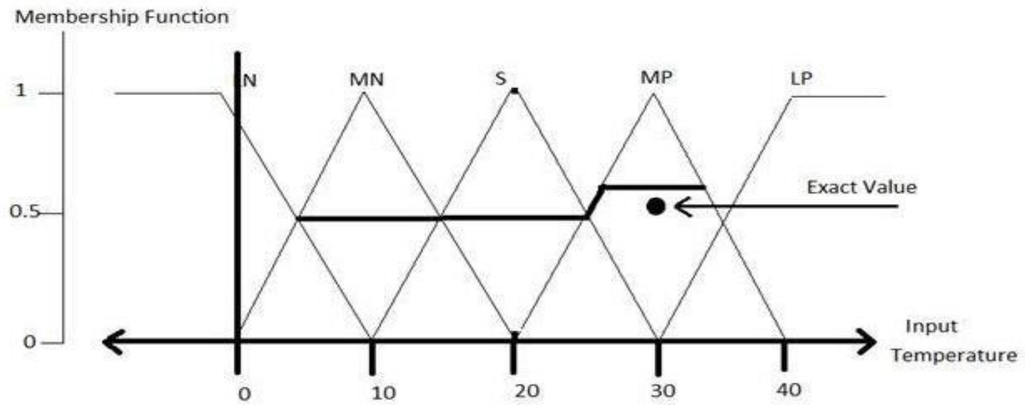
Sr. No.	Condition	Action
1	IF temperature=(Cold OR Very_Cold) AND target=Warm THEN	Heat
2	IF temperature=(Hot OR Very_Hot) AND target=Warm THEN	Cool
3	IF (temperature=Warm) AND (target=Warm) THEN	No_Change

Step 4: Obtain fuzzy value

Fuzzy set operations perform evaluation of rules. The operations used for OR and AND are Max and Min respectively. Combine all results of evaluation to form a final result. This result is a fuzzy value.

Step 5: Perform defuzzification

Defuzzification is then performed according to membership function for output variable.



Application Areas of Fuzzy Logic

The key application areas of fuzzy logic are as given −

Automotive Systems

- Automatic Gearboxes
- Four-Wheel Steering
- Vehicle environment control

Consumer Electronic Goods

- Hi-Fi Systems
- Photocopiers
- Still and Video Cameras
- Television

Domestic Goods

- Microwave Ovens
- Refrigerators
- Toasters
- Vacuum Cleaners
- Washing Machines

Environment Control

- Air Conditioners/Dryers/Heaters
- Humidifiers

Advantages of FLSs

- Mathematical concepts within fuzzy reasoning are very simple.
- You can modify a FLS by just adding or deleting rules due to flexibility of fuzzy logic.
- Fuzzy logic Systems can take imprecise, distorted, noisy input information.
- FLSs are easy to construct and understand.
- Fuzzy logic is a solution to complex problems in all fields of life, including medicine, as it resembles human reasoning and decision making.

UNIT-4: Learning

Machine-Learning Paradigms: Introduction, Machine Learning Systems, Supervised and Unsupervised Learning, Inductive Learning, Learning Decision Trees
Artificial Neural Networks: Introduction, Artificial Neural Networks, Single-Layer Feed-Forward Networks, Multi-Layer Feed-Forward Networks
Reinforcement learning – Learning from rewards, Passive and Active reinforcement learning, Applications.

Machine Learning

- Like human learning from past experiences, a computer does not have “experiences”.
- A computer system learns from data, which represent some “past experiences” of an application domain.
- Objective of machine learning: learn a target function that can be used to predict the values of a discrete class attribute, e.g., approve or not approved, and high-risk or low risk.
- The task is commonly called: Supervised learning, classification, or inductive learning Supervised Learning

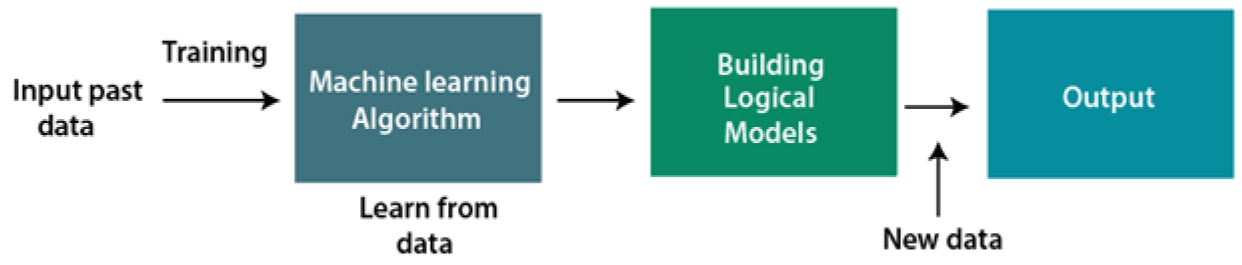
Supervised learning is a machine learning technique for learning a function from training data. The training data consist of pairs of input objects (typically vectors), and desired outputs. The output of the function can be a continuous value (called regression) or can predict a class labelof the input object (called classification). The task of the supervised learner is to predict thevalue of the function for any valid input object after having seen a number of training examples (i.e. pairs of input and target output). To achieve this, the learner has to generalize from the presented data to unseen situations in a "reasonable" way.

Another term for supervised learning is classification. Classifier performance depend greatly on the characteristics of the data to be classified. There is no single classifier that works best on all given problems. Determining a suitable classifier for a given problem is however still more an artthan a science. The most widely used classifiers are the Neural Network (Multi-layer Perceptron), Support Vector Machines, k-Nearest Neighbors, Gaussian Mixture Model,Gaussian, Naive Bayes, Decision Tree and RBF classifiers.

MACHINE LEARNING SYSTEM

A Machine Learning system learns from historical data, builds the prediction models, and whenever it receives new data, predicts the output for it. The accuracy of predicted output depends upon the amount of data, as the huge amount of data helps to build a better model which predicts the output more accurately.

Suppose we have a complex problem, where we need to perform some predictions, so instead of writing a code for it, we just need to feed the data to generic algorithms, and with the help of these algorithms, machine builds the logic as per the data and predict the output. Machine learning has changed our way of thinking about the problem. The below block diagram explains the working of Machine Learning algorithm:



Features of Machine Learning:

Machine learning uses data to detect various patterns in each dataset.

- It can learn from past data and improve automatically.
- It is a data-driven technology.
- Machine learning is much like data mining as it also deals with the huge amount of the data.

Need for Machine Learning

The need for machine learning is increasing day by day. The reason behind the need for machine learning is that it can do tasks that are too complex for a person to implement directly. As a human, we have some limitations as we cannot access the huge amount of data manually, so for this, we need some computer systems and here comes the machine learning to make things easy for us.

Training machine learning algorithms by providing them the huge amount of data and let them explore the data, construct the models, and predict the required output automatically. The performance of the machine learning algorithm depends on the amount of data, and it can be determined by the cost function. With the help of machine learning, we can save both time and money.

The importance of machine learning can be easily understood by its uses cases, Currently, machine learning is used in self-driving cars, cyber fraud detection, face recognition, and friend suggestion by Facebook, etc. Various top companies such as Netflix and Amazon have build machine learning models that are using a vast amount of data to analyse the user interest and recommend product accordingly.

Following are some key points which show the importance of Machine Learning:

- Rapid increment in the production of data
- Solving complex problems, which are difficult for a human
- Decision making in various sector including finance
- Finding hidden patterns and extracting useful information from data.

SUPERVISED AND UNSUPERVISED LEARNING

Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.

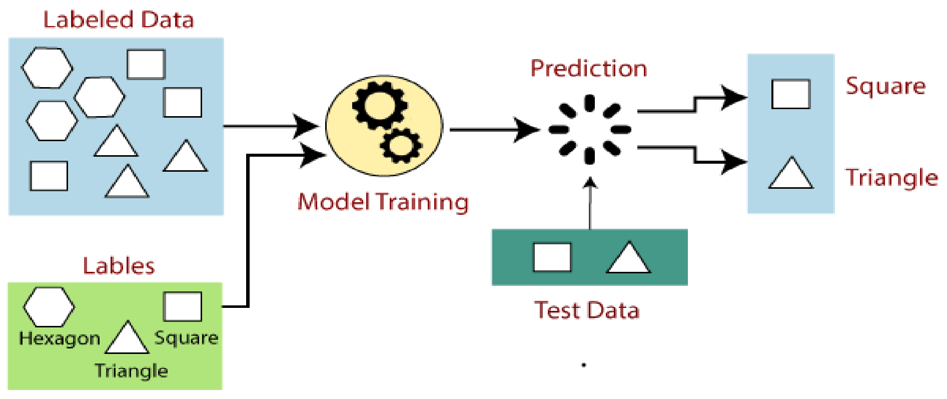
In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.

Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to find a mapping function to map the input variable(x) with the output variable(y).

In the real-world, supervised learning can be used for Risk Assessment, Image classification, Fraud Detection, spam filtering, etc.

How Supervised Learning Works

In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.The working of Supervised learning can be easily understood by the below example and diagram:

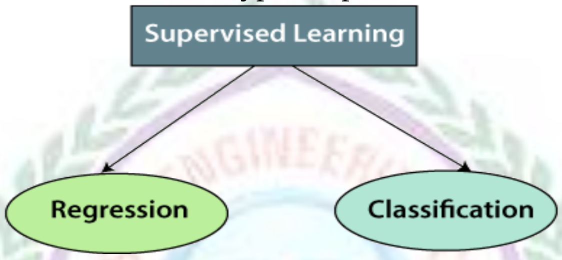


Steps Involved In Supervised Learning:

- First Determine the type of training dataset
- Collect/Gather the labelled training data.
- Split the training dataset into training dataset, test dataset, and validation dataset.
- Determine the input features of the training dataset, which should have enough knowledge so that the model can accurately predict the output.
- Determine the suitable algorithm for the model, such as support vector machine, decision tree, etc.
- Execute the algorithm on the training dataset. Sometimes we need validation sets as the control parameters, which are the subset of training datasets.
- Evaluate the accuracy of the model by providing the test set. If the model predicts the correct output, which means our model is accurate.

Types of supervised Machine learning Algorithms:

Supervised learning can be further divided into two types of problems:



Regression: Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc. Below are some popular Regression algorithms which come under supervised learning:

- Linear Regression
- Regression Trees
- Non-Linear Regression
- Bayesian Linear Regression
- Polynomial Regression

Classification: Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false, etc.

- Spam Filtering,
- Random Forest
- Decision Trees
- Logistic Regression
- Support vector Machines

UNSUPERVISED LEARNING

As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things.

It can be defined as:

Unsupervised learning is a type of machine learning in which models are trained using unlabelled dataset and are allowed to act on that data without any supervision.

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.

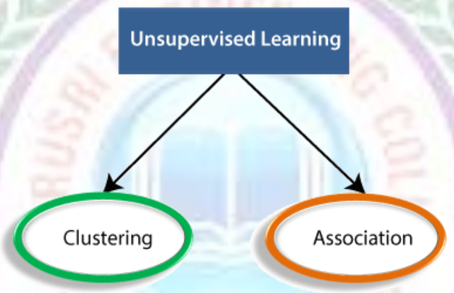
Why use Unsupervised Learning

Below are some main reasons which describe the importance of Unsupervised Learning:

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.
- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

Types of Unsupervised Learning Algorithm:

The unsupervised learning algorithm can be further categorized into two types of problems:



Clustering: Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.

Association: An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.

Unsupervised Learning algorithms:

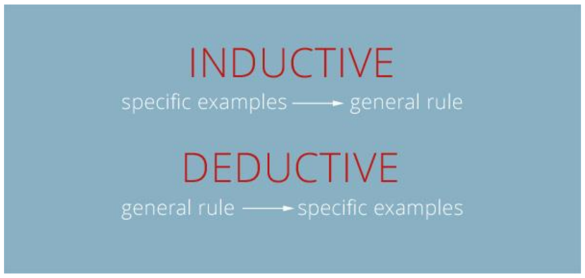
Below is the list of some popular unsupervised learning algorithms:

- K-means clustering
- KNN (k-nearest neighbors)
- Hierarchal clustering
- Anomaly detection
- Neural Networks
- Principle Component Analysis
- Independent Component Analysis
- Apriori algorithm
- Singular value decomposition

Supervised Learning	Unsupervised Learning
Supervised learning algorithms are trained using labeled data.	Unsupervised learning algorithms are trained using unlabeled data.
Supervised learning model takes direct feedback to check if it is predicting correct output or not.	Unsupervised learning model does not take any feedback.
Supervised learning model predicts the output.	Unsupervised learning model finds the hidden patterns in data.
In supervised learning, input data is provided to the model along with the output.	In unsupervised learning, only input data is provided to the model.
The goal of supervised learning is to train the model so that it can predict the output when it is given new data.	The goal of unsupervised learning is to find the hidden patterns and useful insights from the unknown dataset.
Supervised learning needs supervision to train the model.	Unsupervised learning does not need any supervision to train the model.
Supervised learning can be categorized in Classification and Regression problems.	Unsupervised Learning can be classified in Clustering and Associations problems.
Supervised learning can be used for those cases where we know the input as well as corresponding outputs.	Unsupervised learning can be used for those cases where we have only input data and no corresponding output data.
Supervised learning model produces an accurate result.	Unsupervised learning model may give less accurate result as compared to supervised learning.
Supervised learning is not close to true Artificial intelligence as in this, we first train the model for each data, and then only it can predict the correct output.	Unsupervised learning is more close to the true Artificial Intelligence as it learns similarly as a child learns daily routine things by his experiences.
It includes various algorithms such as Linear Regression, Logistic Regression, Support Vector Machine, Multi-class Classification, Decision tree, Bayesian Logic, etc.	It includes various algorithms such as Clustering, KNN, and Apriori algorithm.

INDUCTIVE LEARNING

Inductive learning, also known as discovery learning, is a process where the learner discovers rules by observing examples. This is different from deductive learning, where students are given rules that they then need to apply.


INDUCTIVE
specific examples → general rule
DEDUCTIVE
general rule → specific examples

We can often work out rules for ourselves by observing examples to see if there is a pattern; to see if things regularly happen in the same way. We then try applying the rule in different situations to see if it works.
With inductive language learning, tasks are designed specifically to help guide the learner and assist them in discovering a rule.

Inductive learning vs. deductive learning

- It can be difficult to learn a lot of new rules, but the mental effort of working out rules for ourselves, using inductive learning, helps us remember them.
- Also, knowing a rule doesn't always mean that it is easy to apply in real life. When children are first shown how to ride a bicycle, it's not possible for them to cycle unaided immediately. Parents guide and assist their children until they have gained the confidence and skills that enable them to ride on their own.
- It is thought that inductive learning is probably the way we learn our first language. It can be a very effective method of learning the grammar of a second language.
- However, one disadvantage of the inductive approach is the risk that the learner will formulate a rule incorrectly. For this reason, it is important to check that the learner has inferred the correct rule. Also, if a rule is complex it may be better to use the deductive approach and give the rule first, or give some guidance.

So, which approach is best for language learning?

- There is no simple answer. Some learning points are more appropriate for inductive learning than others. For example, it would be very difficult to work out the rules for the use in English of the articles "the", "a" and "an" using an inductive approach. There are so many rules and exceptions to these rules that students would need dozens of examples to cover all of the different uses.
- The same is true of prepositions. Often there are no clear grammar rules that apply to prepositions and their use is a question of collocation – some prepositions are commonly used in certain phrases or contexts.
- For example, we say "ON Tuesday" not "IN Tuesday" and we say "IN May" not "ON May". Why is this so?
- There is no simple rule we can give learners. In this case the best option is to learn the items as a phrase: on + day of the week and in + month of the year.
- On the other hand, there are some rules that are easier for learners to work out. With a few well-chosen example sentences, you could provide enough evidence for learners to discover the rule that modal verbs are followed by the infinitive without "to". In this case, an inductive approach can work well.
- What's more, with an inductive approach, learners first see the target item in context. They focus on meaning first, and then on form.

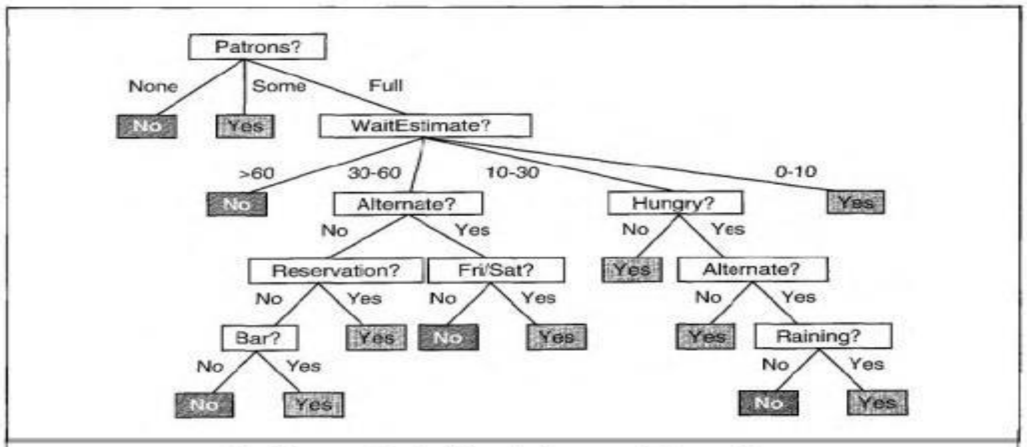
DECISION TREE

- A decision tree takes as input an object or situation described by a set of attributes and returns a "decision" – the predicted output value for the input.
- A decision tree reaches its decision by performing a sequence of tests. Example : "HOW TO" manuals (for car repair)

A decision tree reaches its decision by performing a sequence of tests. Each internal node in the tree corresponds to a test of the value of one of the properties, and the branches from the nodeare labeled with the possible values of the test. Each leaf node in the tree specifies the value to bereturned if that leaf is reached. The decision tree representation seems to be very natural for humans; indeed, many "How To" manuals (e.g., for car repair) are written entirely as a single decision tree stretching over hundreds of pages.

A somewhat simpler example is provided by the problem of whether to wait for a table at a restaurant. The aim here is to learn a definition for the goal predicate Will Wait. In setting this upas a learning problem, we first have to state what attributes are available to describe examples in the domain. we will see how to automate this task; for now, let's suppose we decide on the following list of attributes:

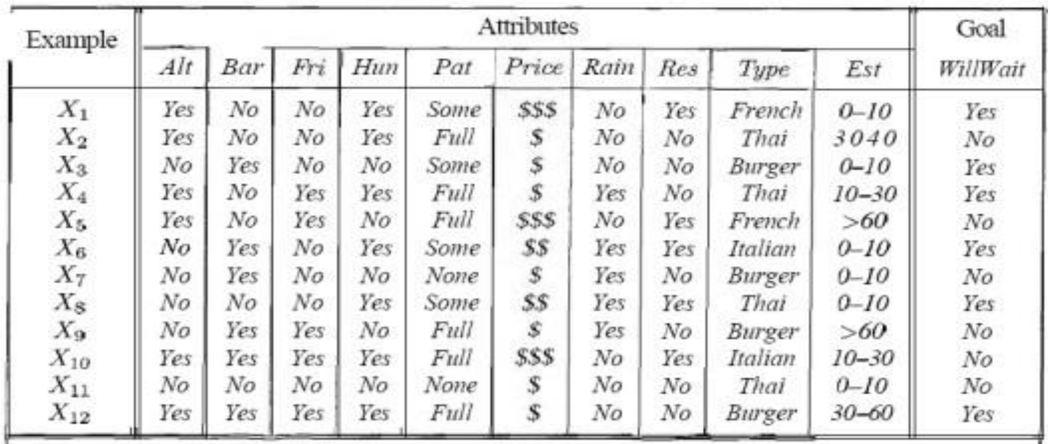
1. Alternate: whether there is a suitable alternative restaurant nearby.
2. Bar: whether the restaurant has a comfortable bar area to wait in.
3. Fri/Sat: true on Fridays and Saturdays.
4. Hungry: whether we are hungry.
5. Patrons: how many people are in the restaurant (values are None, Some, and Full).
6. Price: the restaurant's price range ($, $$, $$$).
7. Raining: whether it is raining outside.
8. Reservation: whether we made a reservation.
9. Type: the kind of restaurant (French, Italian, Thai, or burger).
10. Wait Estimate: the wait estimated by the host (0-10 minutes, 10-30, 30-60, >60).



A decision tree for deciding whether to wait for a table.

Decision tree induction from examples

An example for a Boolean decision tree consists of a vector of' input attributes, X, and a single Boolean output value y. A set of examples (X1,Y1) . . . , (X2, y2) is shown in Figure. The positive examples are the ones in which the goal *Will Wait* is true (XI, ***X3,*** . . .); the negative examples are the ones in which it is false (X2, *X5*, . . .). The complete set of examples is called the **training set.**

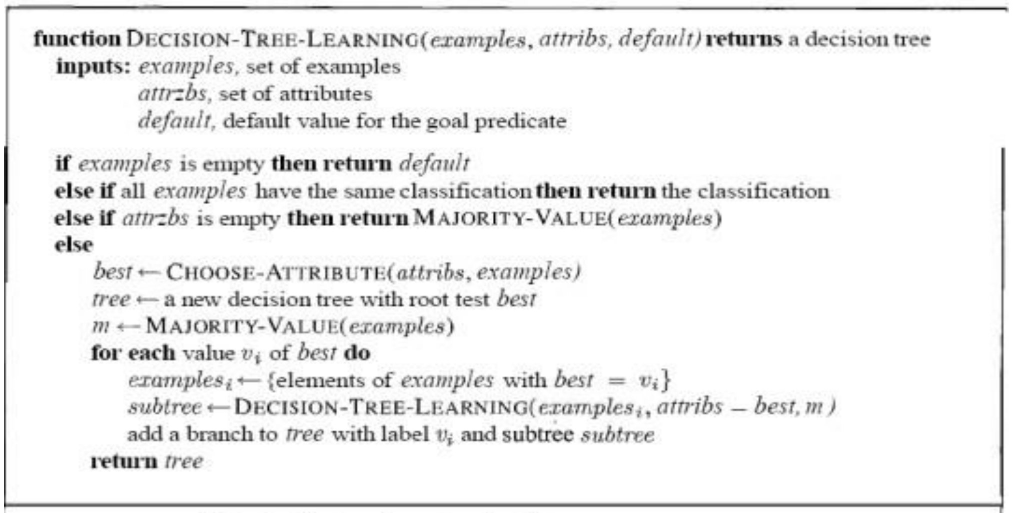


Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*3040*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

Examples for the restaurant domain.

Decision Tree Algorithm

The basic idea behind the Decision-Tree-Learning-Algorithm is to test the most important attribute first. By "most important," we mean the one that makes the most difference to the classification of an example. That way, we hope to get to the correct classification with a small number of tests, meaning that all paths in the tree will be short and the tree as a whole will be small.



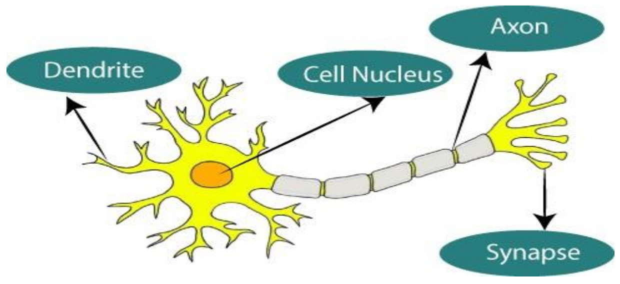
```
function DECISION-TREE-LEARNING(examples, attribs, default) returns a decision tree
  inputs: examples, set of examples
          attrzbs, set of attributes
          default, default value for the goal predicate

  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attrzbs is empty then return MAJORITY-VALUE(examples)
  else
     best ← CHOOSE-ATTRIBUTE(attribs, examples)
     tree ← a new decision tree with root test best
     m ← MAJORITY-VALUE(examples)
     for each value $v_i$ of best do
        $examples_i$ ← {elements of examples with best = $v_i$}
        subtree ← DECISION-TREE-LEARNING($examples_i$, attribs – best, m)
        add a branch to tree with label $v_i$ and subtree subtree
     return tree
```

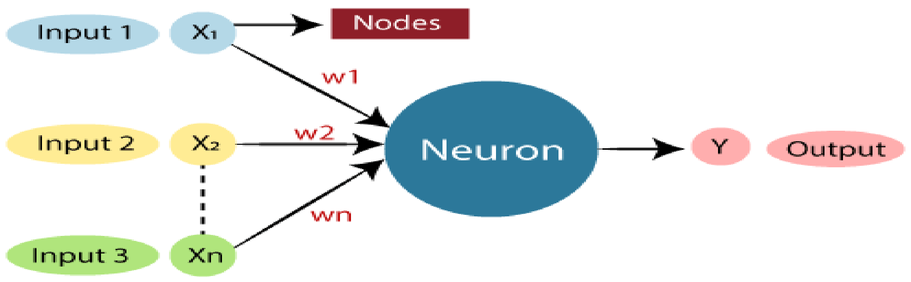
The decision tree learning algorithm.

ARTIFICIAL NEURAL NETWORKS

The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.



The given figure illustrates the typical diagram of Biological Neural Network. The typical Artificial Neural Network looks something like the given figure.



Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

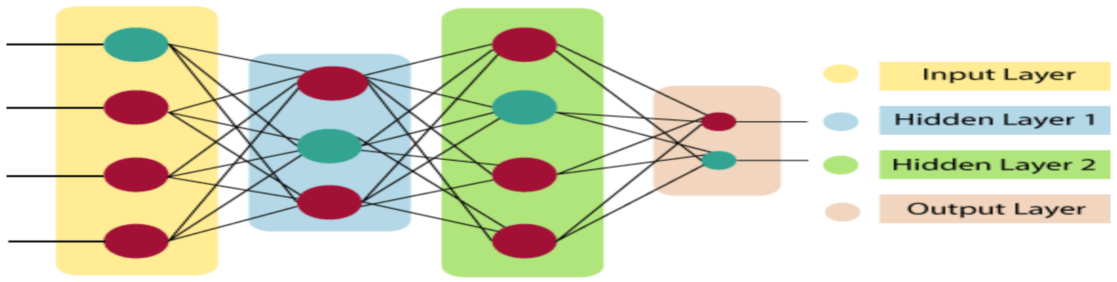
Relationship between Biological neural network and artificial neural network:

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

An Artificial Neural Network in the field of Artificial intelligence where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.

There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data when necessary from our memory parallelly. We can say that the human brain is made up of incredibly amazing parallel processors.We can understand the artificial neural network with an example, consider an example of a digital logic gate that takes an input and gives an output. "OR" gate, which takes two inputs. If one or both the inputs are "On," then we get "On" in output. If both the inputs are "Off," then we get "Off" in output. Here the output depends upon input. Our brain does not perform the same task. The outputs to inputs relationship keep changing because of the neurons in our brain, which are "learning."

The Architecture Of An Artificial Neural Network:



Input Layer:

As the name suggests, it accepts inputs in several different formats provided by the programmer.

Hidden Layer:
The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.

Output Layer:
The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.

The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

$$\sum_{i=1}^{n} Wi * Xi + b$$

It determines weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.

Advantages of Artificial Neural Network (ANN)
Parallel processing capability:
Artificial neural networks have a numerical value that can perform more than one task simultaneously.

Storing data on the entire network:
Data that is used in traditional programming is stored on the whole network, not on a database. The disappearance of a couple of pieces of data in one place doesn't prevent the network from working.

Capability to work with incomplete knowledge:
After ANN training, the information may produce output even with inadequate data. The loss of performance here relies upon the significance of missing data.

Having a memory distribution:
For ANN is to be able to adapt, it is important to determine the examples and to encourage the network according to the desired output by demonstrating these examples to the network. The succession of the network is directly proportional to the chosen instances, and if the event can't appear to the network in all its aspects, it can produce false output.
Having fault tolerance:
Extortion of one or more cells of ANN does not prohibit it from generating output, and this feature makes the network fault-tolerance.
Disadvantages of Artificial Neural Network:
Assurance of proper network structure:
There is no particular guideline for determining the structure of artificial neural networks. The appropriate network structure is accomplished through experience, trial, and error.
Unrecognized behavior of the network:
It is the most significant issue of ANN. When ANN produces a testing solution, it does not provide insight concerning why and how. It decreases trust in the network.
Hardware dependence:
Artificial neural networks need processors with parallel processing power, as per their structure. Therefore, the realization of the equipment is dependent.
Difficulty of showing the issue to the network:

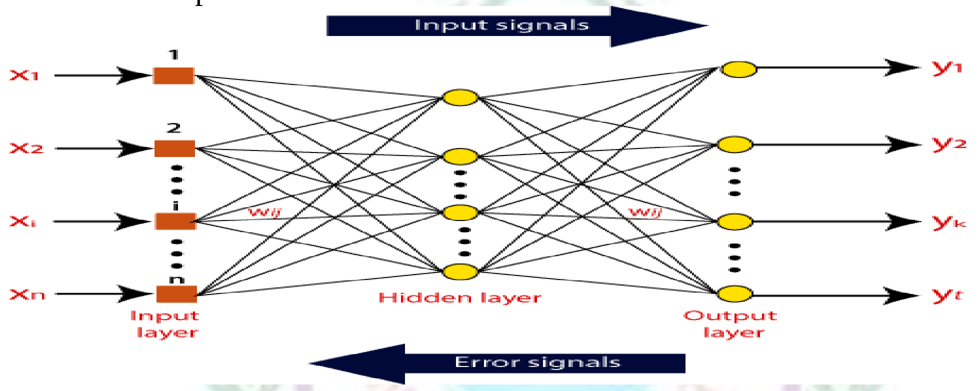
ANNs can work with numerical data. Problems must be converted into numerical values before being introduced to ANN. The presentation mechanism to be resolved here will directly impact the performance of the network. It relies on the user's abilities.

The duration of the network is unknown:

The network is reduced to a specific value of the error, and this value does not give us optimum results. “Science artificial neural networks that have steeped into the world in the mid-20th century are exponentially developing. In the present time, we have investigated the pros of artificial neural networks and the issues encountered in the course of their utilization. It should not be overlooked that the cons of ANN networks, which are a flourishing science branch, are eliminated individually, and their pros are increasing day by day. It means that artificial neural networks will turn into an irreplaceable part of our lives progressively important.”

How do artificial neural networks work?

Artificial Neural Network can be best represented as a weighted directed graph, where the artificial neurons form the nodes. The association between the neurons outputs and neuron inputs can be viewed as the directed edges with weights. The Artificial Neural Network receives the input signal from the external source in the form of a pattern and image in the form of a vector. These inputs are then mathematically assigned by the notations x(n) for every n number of inputs.



Afterward, each of the input is multiplied by its corresponding weights (these weights are the details utilized by the artificial neural networks to solve a specific problem). In general terms, these weights normally represent the strength of the interconnection between neurons inside the artificial neural network. All the weighted inputs are summarized inside the computing unit.

If the weighted sum is equal to zero, then bias is added to make the output non-zero or something else to scale up to the system's response. Bias has the same input, and weight equals to 1. Here the total of weighted inputs can be in the range of 0 to positive infinity. Here, to keep the response in the limits of the desired value, a certain maximum value is benchmarked, and the total of weighted inputs is passed through the activation function.The activation function refers to the set of transfer functions used to achieve the desired output. There is a different kind of the activation function, but primarily either linear or non-linear sets of functions. Some of the commonly used sets of activation functions are the Binary, linear, and Tan hyperbolic sigmoidal activation functions. Let us take a look at each of them in details:

Binary:

In binary activation function, the output is either a one or a 0. Here, to accomplish this, there is a threshold value set up. If the net weighted input of neurons is more than 1, then the final output of the activation function is returned as one or else the output is returned as 0.

Sigmoidal Hyperbolic:

The Sigmoidal Hyperbola function is generally seen as an "S" shaped curve. Here the tan hyperbolic function is used to approximate output from the actual net input. The function is defined as:

F(x) = (1/1 + exp(-????x))

Where ???? is considered the Steepness parameter.

Types of Artificial Neural Network:

There are various types of Artificial Neural Networks (ANN) depending upon the human brain neuron and network functions, an artificial neural network similarly performs tasks. The majority of the artificial neural networks will have some similarities with a more complex biological partner and are very effective at their expected tasks. For example, segmentation or classification.

Feedback ANN:

In this type of ANN, the output returns into the network to accomplish the best-evolved results internally. As per the University of Massachusetts, Lowell Centre for Atmospheric Research. The feedback networks feed information back into itself and are well suited to solve optimization issues. The Internal system error corrections utilize feedback ANNs.

Feed-Forward ANN:

A feed-forward network is a basic neural network comprising of an input layer, an output layer, and at least one layer of a neuron. Through assessment of its output by reviewing its input, the intensity of the network can be noticed based on group behavior of the associated neurons, and the output is decided. The primary advantage of this network is that it figures out how to evaluate and recognize input patterns.

Prerequisite

No specific expertise is needed as a prerequisite before starting this tutorial.

Audience

Our Artificial Neural Network Tutorial is developed for beginners as well as professionals, to help them understand the basic concept of ANNs.

The Perceptron Training Rule

Although we are interested in learning networks of many interconnected units, let us begin by understanding how to learn the weights for a single perceptron. Here the precise learning problem is to determine a weight vector that causes the perceptron to produce the correct 1 output for each of the given training examples.

Several algorithms are known to solve this learning problem. Here we consider two: the perceptron rule and the delta rule. These two algorithms are guaranteed to converge to somewhat different acceptable hypotheses, under somewhat different conditions. They are important to ANNs because they provide the basis for learning networks of many units.

One way to learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example. This process is repeated, iterating through the training examples as many times as needed untilthe perceptron classifies all training examples correctly. Weights are modified at each step according to the perceptron training rule, which revises the weight wi associated with input xi according to the rule

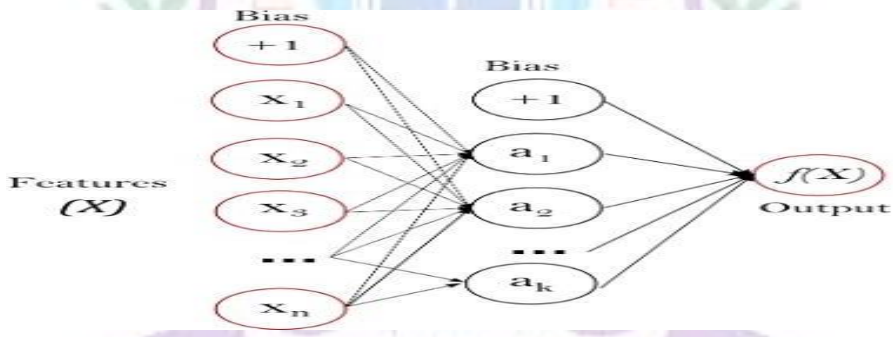
$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Multi-layer feed -Forward Networks

Multi-layer Feed -forward Networks is a supervised learning algorithm that learns a function f(·):Rm→Ro by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features X=x1,x2,...,xm and a target y, it can learn a non- linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. Figure shows a one hidden layer MLP with scalar output.



The leftmost layer, known as the input layer, consists of a set of neurons {xi|x1,x2,...,xm} representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation w1x1+w2x2+...+wmxm, followed by a non-linear activation function g(·):R→R - like the hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.

The module contains the public attributes coefs_ and intercepts_. coefs_ is a list of weight matrices, where weight matrix at index i represents the weights between layer i and layer i+1. intercepts_ is a list of bias vectors, where the vector at index i represents the bias values added to layer i+1.

The advantages of Multi-layer Perceptron are:

- Capability to learn non-linear models.
- Capability to learn models in real-time (on-line learning) using partial_fit. The disadvantages of Multi-layer Perceptron (MLP) include:
- MLP with hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy.

- MLP requires tuning a number of hyperparameters such as the number of hidden neurons, layers, and iterations.
- MLP is sensitive to feature scaling.

REINFORCEMENT LEARNING

Learning what to do to maximize reward

- Learner is not given training
- Only feedback is in terms of reward
- Try things out and see what the reward is
- Different from Supervised Learning
- Teacher gives training examples

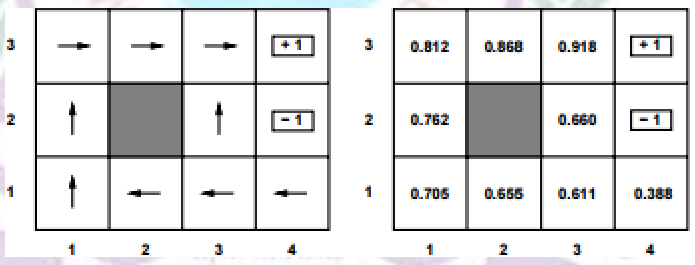
Applications

- Robotics: Quadruped Gait Control, Ball Acquisition (Robocup)
- Control: Helicopters
- Operations Research: Pricing, Routing, Scheduling
- Game Playing: Backgammon, Solitaire, Chess, Checkers
- Human Computer Interaction: Spoken Dialogue Systems
- Economics/Finance: Trading

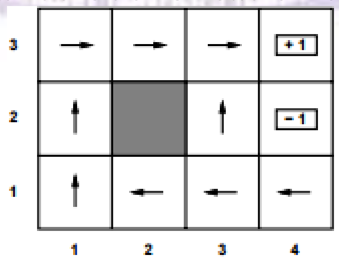
Types of Reinforcement Learning

- Passive Vs Active
- Passive: Agent executes a fixed policy and evaluates it
- Active: Agents updates policy as it learns
- Model based Vs Model free
- Model-based: Learn transition and reward model, use it to get optimal policy
- Model free: Derive optimal policy without learning the model

Passive Learning



- Evaluate how good a policy π is
- Learn the utility Uπ(s) of each state
- Same as policy evaluation for known transition & reward models



Agent executes a sequence of trials:

$(1, 1) \rightarrow (1, 2) \rightarrow (1, 3) \rightarrow (1, 2) \rightarrow (1, 3) \rightarrow (2, 3) \rightarrow (3, 3) \rightarrow (4, 3)_{+1}$

$(1, 1) \rightarrow (1, 2) \rightarrow (1, 3) \rightarrow (2, 3) \rightarrow (3, 3) \rightarrow (3, 2) \rightarrow (3, 3) \rightarrow (4, 3)_{+1}$

$(1, 1) \rightarrow (2, 1) \rightarrow (3, 1) \rightarrow (3, 2) \rightarrow (4, 2)_{-1}$

Goal is to learn the expected utility $U_\pi(s)$

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s\right]$$

Direct Utility Estimation

- Reduction to inductive learning
- Compute the empirical value of each state
- Each trial gives a sample value
- Estimate the utility based on the sample values
- Example: First trial gives
- State (1,1): A sample of reward 0.72
- State (1,2): Two samples of reward 0.76 and 0.84
- State (1,3): Two samples of reward 0.80 and 0.88
- Estimate can be a running average of sample values
- Example: U(1, 1) = 0.72,U(1, 2) = 0.80,U(1, 3) = 0.84, . . .
- Ignores a very important source of information
- The utility of states satisfy the Bellman equations

$$U^\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U^\pi(s')$$

- Search is in a hypothesis space for U much larger than needed
- Convergence is very slow
- Make use of Bellman equations to get $U^\pi(s)$

- Need to estimate T(s, π(s), s‟) and R(s) from trials
- Plug-in learnt transition and reward in the Bellman equations
- Solving for U^π: System of n linear equations

- Estimates of T and R keep changing
- Make use of modified policy iteration idea
- Run few rounds of value iteration
- Initialize value iteration from previous utilities
- Converges fast since T and R changes are small
- ADP is a standard baseline to test „smarter‟ ideas
- ADP is inefficient if state space is large
- Has to solve a linear system in the size of the state space
- Backgammon: 10^{50} linear equations in 10^{50} unknowns

Temporal Difference Learning

- Best of both worlds
- Only update states that are directly affected
- Approximately satisfy the Bellman equations
- Example:

$(1, 1) \rightarrow (1, 2) \rightarrow (1, 3) \rightarrow (1, 2) \rightarrow (1, 3) \rightarrow (2, 3) \rightarrow (3, 3) \rightarrow (4, 3)_{+1}$

$(1, 1) \rightarrow (1, 2) \rightarrow (1, 3) \rightarrow (2, 3) \rightarrow (3, 3) \rightarrow (3, 2) \rightarrow (3, 3) \rightarrow (4, 3)_{+1}$

$(1, 1) \rightarrow (2, 1) \rightarrow (3, 1) \rightarrow (3, 2) \rightarrow (4, 2)_{-1}$

- After the first trial, U(1, 3) = 0.84,U(2, 3) = 0.92
- Consider the transition (1, 3) → (2, 3) in the second trial

- If deterministic, then U(1, 3) = −0.04 + U(2, 3)
- How to account for probabilistic transitions (without a model)

TD Vs ADP

- TD is mode free as opposed to ADP which is model based
- TD updates observed successor rather than all successors
- The difference disappears with large number of trials
- TD is slower in convergence, but much simpler computation per observation

Active Learning

- Agent updates policy as it learns
- Goal is to learn the optimal policy
- Learning using the passive ADP agent
- Estimate the model R(s),T(s, a, s‘‘) from observations
- The optimal utility and action satisfies

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$$

- Solve using value iteration or policy iteration
- Agent has “optimal” action
- Simply execute the “optimal” action

Exploitation vs Exploration

- The passive approach gives a greedy agent
- Exactly executes the recipe for solving MDPs
- Rarely converges to the optimal utility and policy
- The learned model is different from the true environment
- Trade-off
- Exploitation: Maximize rewards using current estimates
- Agent stops learning and starts executing policy
- Exploration: Maximize long term rewards
- Agent keeps learning by trying out new things
- Pure Exploitation
- Mostly gets stuck in bad policies
- Pure Exploration
- Gets better models by learning
- Small rewards due to exploration
- The multi-armed bandit setting
- A slot machine has one lever, a one-armed bandit
- n-armed bandit has n levers
- Which arm to pull?
- Exploit: The one with the best pay-off so far
- Explore: The one that has not been tried

Q-Learning

- Exploration function gives a active ADP agent
- A corresponding TD agent can be constructed

- Surprisingly, the TD update can remain the same
- Converges to the optimal policy as active ADP
- Slower than ADP in practice
- Q-learning learns an action-value function Q(a; s)
- Utility values U(s) = $\max_a$ Q(a; s)

- A model-free TD method
- No model for learning or action selection
- Constraint equations for Q-values at equilibrium

$$Q(a,s) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(a',s')$$

- Can be updated using a model for T(s; a; s‟)

- The TD Q-learning does not require a model

$$Q(a,s) \leftarrow Q(a,s) + \alpha(R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s))$$

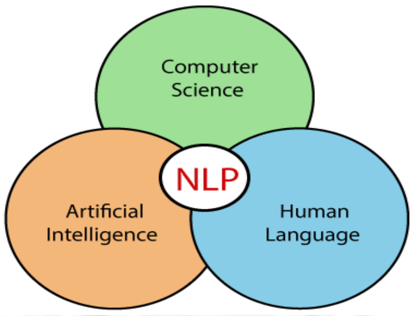
- Calculated whenever a in s leads to s‟
- The next action a_{next} = $\text{argmax}_{a‟}$ f (Q(a‟; s‟);N(s‟; a‟))

- Q-learning is slower than ADP
- Trade-o: Model-free vs knowledge-based methods

UNIT-5: Communicating & Perceiving
Introduction to NLP- Progress & applications of NLP, Components of NLP, Grammars, Parsing
Automatic Speech Recognition (ASR) – Speech Processing, Ex: DRAGON, HARPY
Machine Vision – Applications, Basic Principles of Vision, Machine vision techniques: Low, Middle and High-level vision
AI Today & Tomorrow - Achievements, ubiquitous AI

INTRODUCTION TO NATURAL LANGUAGE PROCESSISING(NLP)

- NLP stands for Natural Language Processing, which is a part of Computer Science, Human language, and Artificial Intelligence.
- It is the technology that is used by machines to understand, analyze, manipulate, and interpret human's languages. It helps developers to organize knowledge for performing tasks such as translation, automatic summarization, Named Entity Recognition (NER), speech recognition, relationship extraction, and topic segmentation.



History of NLP
(1940-1960) - Focused on Machine Translation (MT)
The Natural Languages Processing started in the year 1940s.
1948 - In the Year 1948, the first recognizable NLP application was introduced in Birkbeck College, London.
1950s - In the Year 1950s, there was a conflicting view between linguistics and computer science. Now, Chomsky developed his first book syntactic structures and claimed that language is generative in nature.
1957 - Chomsky also introduced the idea of Generative Grammar, which is rule based descriptions of syntactic structures.
(1960-1980) - Flavored with Artificial Intelligence (AI)
In the year 1960 to 1980, the key developments were:
Augmented Transition Networks (ATN)-Augmented Transition Networks is a finite state machine that is capable of recognizing regular languages.
Case Grammar - Case Grammar was developed by Linguist Charles J. Fillmore in the year 1968. Case Grammar uses languages such as English to express the relationship between nouns and verbs by using the preposition.
In Case Grammar, case roles can be defined to link certain kinds of verbs and objects.
For example: "Neha broke the mirror with the hammer". In this example case grammar identify Neha as an agent, mirror as a theme, and hammer as an instrument.
1980 - Current
Till the year 1980, natural language processing systems were based on complex sets of hand-written rules. After 1980, NLP introduced machine learning algorithms for language processing.
In the beginning of the year 1990s, NLP started growing faster and achieved good process accuracy, especially in English Grammar. Other factors may include the availability of computers with fast CPUs and more memory. The major factor behind the advancement of natural language processing was the Internet.
Modern NLP consists of various applications, like speech recognition, machine translation, and machine text reading. When we combine all these applications then it allows the artificial intelligence to gain knowledge

of the world. Let's consider the example of AMAZON ALEXA, using this robot you can ask the question to Alexa, and it will reply to you.

Advantages of NLP

- NLP helps users to ask questions about any subject and get a direct response within seconds.
- NLP offers exact answers to the question means it does not offer unnecessary and unwanted information.
- NLP helps computers to communicate with humans in their languages.
- It is very time efficient.
- Most of the companies use NLP to improve the efficiency of documentation processes, accuracy of documentation, and identify the information from large databases.

Disadvantages of NLP

- NLP may not show context.
- NLP is unpredictable
- NLP may require more keystrokes.
- NLP is unable to adapt to the new domain, and it has a limited function that's why NLP is built for a single and specific task only.

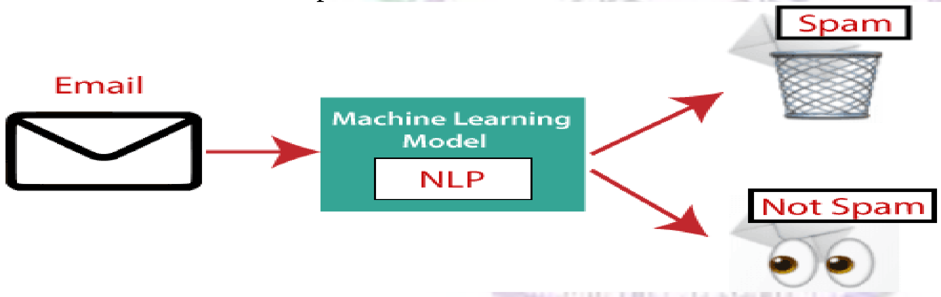
APPLICATIONS OF NLP

1. Question Answering

Question Answering focuses on building systems that automatically answer the questions asked by humans in a natural language.

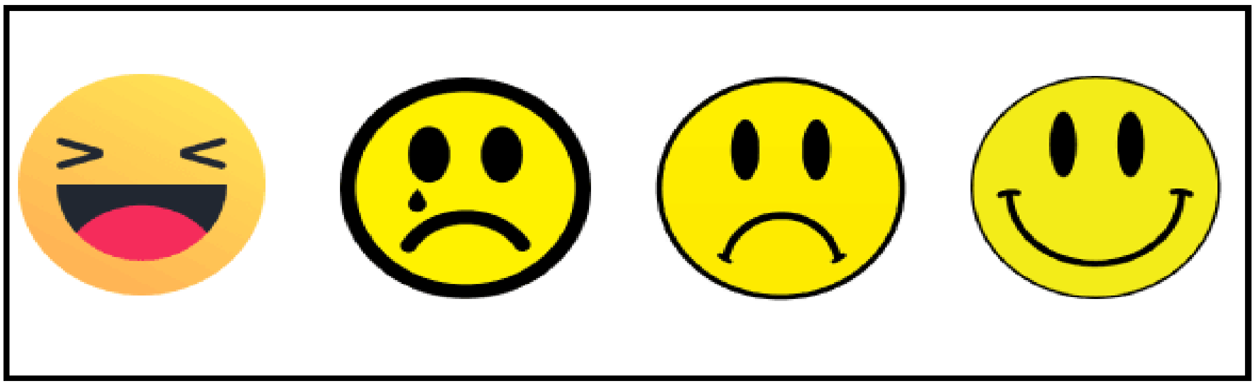
2. Spam Detection

Spam detection is used to detect unwanted e-mails getting to a user's inbox.



3. Sentiment Analysis

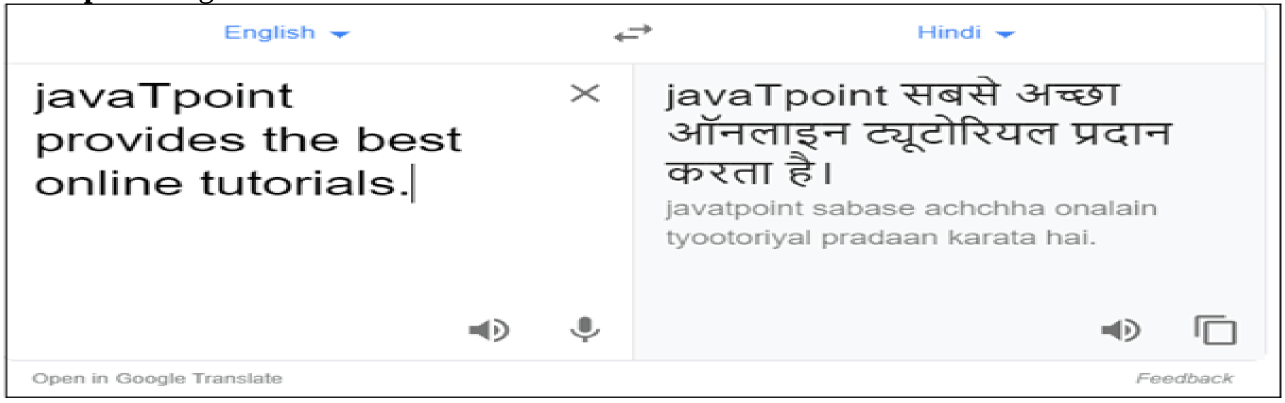
Sentiment Analysis is also known as opinion mining. It is used on the web to analyse the attitude, behaviour, and emotional state of the sender. This application is implemented through a combination of NLP (Natural Language Processing) and statistics by assigning the values to the text (positive, negative, or natural), identify the mood of the context (happy, sad, angry, etc.)



4. Machine Translation

Machine translation is used to translate text or speech from one natural language to another natural language.

Example: Google Translator



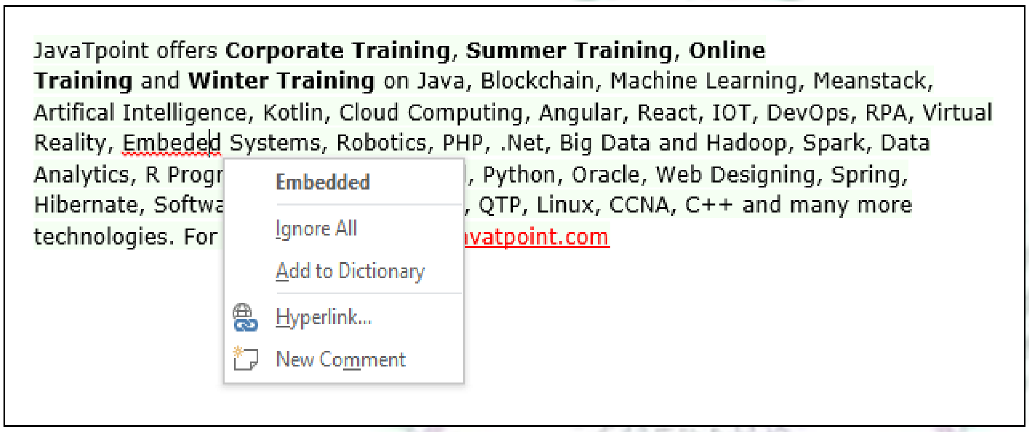
javaTpoint provides the best online tutorials.

javaTpoint सबसे अच्छा ऑनलाइन ट्यूटोरियल प्रदान करता है।

javatpoint sabase achchha onalain tyootoriyal pradaan karata hai.

5. Spelling correction

Microsoft Corporation provides word processor software like MS-word, PowerPoint for the spelling correction



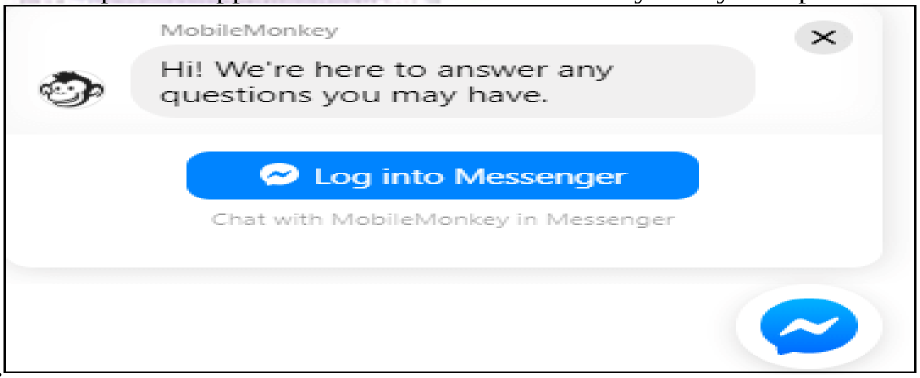
JavaTpoint offers **Corporate Training**, **Summer Training**, **Online Training** and **Winter Training** on Java, Blockchain, Machine Learning, Meanstack, Artifical Intelligence, Kotlin, Cloud Computing, Angular, React, IOT, DevOps, RPA, Virtual Reality, Embeded Systems, Robotics, PHP, .Net, Big Data and Hadoop, Spark, Data Analytics, R Progr , Python, Oracle, Web Designing, Spring, Hibernate, Softwa , QTP, Linux, CCNA, C++ and many more technologies. For vatpoint.com

6. Speech Recognition

Speech recognition is used for converting spoken words into text. It is used in applications, such as mobile, home automation, video recovery, dictating to Microsoft Word, voice biometrics, voice user interface, and so on.

7. Chatbot

Implementing the Chatbot is one of the important applications of NLP. It is used by many companies to provide the customer's chat services.



MobileMonkey

Hi! We're here to answer any questions you may have.

8. Information extraction

Information extraction is one of the most important applications of NLP. It is used for extracting structured information from unstructured or semi-structured machine-readable documents.

9. Natural Language Understanding (NLU)
It converts a large set of text into more formal representations such as first-order logic structures that are easier for the computer programs to manipulate notations of the natural language processing.

COMPONENTS OF NLP

1. **Natural Language Understanding (NLU)**

Natural Language Understanding (NLU) helps the machine to understand and analyse human language by extracting the metadata from content such as concepts, entities, keywords, emotion, relations, and semantic roles.
NLU mainly used in Business applications to understand the customer's problem in both spoken and written language.NLU involves the following tasks -

- It is used to map the given input into useful representation.
- It is used to analyze different aspects of the language.

2. **Natural Language Generation (NLG)**

Natural Language Generation (NLG) acts as a translator that converts the computerized data into natural language representation. It mainly involves Text planning, Sentence planning, and Text Realization.

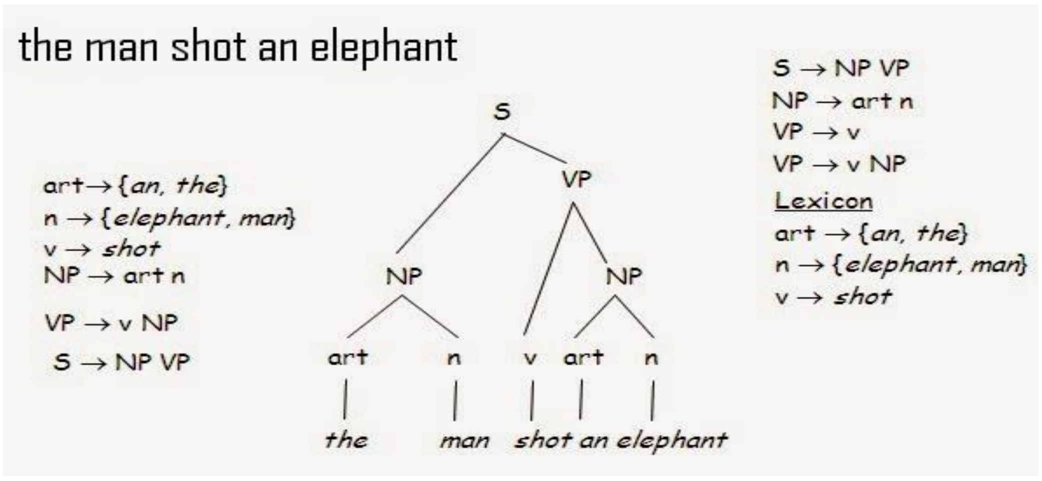
GRAMMARS AND PARSING IN NLP

- Phonetics & Phonology
- Morphological Analysis
- Syntactic Analysis
- Lexical Analysis
- Semantic Analysis
- Discourse Integration
- Pragmatic Analysis

1) Phonetics & Phonology:

Phonetics:	Pronunciation of different speakers. Deals with physical building blocks of language sound system. Pace of Speech. Ex: I ate eight cake, different 'k' sounds in 'kite', 'coat If we speak in foreign English (I 8 8 cake) similarly k (Hindi क for kite & coat).
Phonology:	Processing of a speech. Organization of speech sound within language. Ex: Bank (finance) v/s. Bank (River), In hindi, aa-jayenge (will come) or aaj-ayenge (will come today).

2) Morphological Analysis: Morphology is the structure of words. Various form of basic word. Make more words from less. Ex: dog-dogs, run-ran, bus-buses etc.
3) Syntactic Analysis: It is concerned with the construction of sentences. Indicates how the words are related to each other. Syntax tree is assigned by a grammar and a lexicon.



4) Lexical Analysis: Obtaining the properties of word.
Ex: 'dog' then you can easily bring an image of dog & its properties like 4 leg, carnivore and animate. This property is also matches with another animals like Lion.
5) Semantic Analysis: Concerned with the meaning of language. The first step in any semantic processing is to look up the individual word in the dictionary and extract their meaning. Ex: The sentence "you have a colourless green idea...." would rejected as semantically because colourless & green makes no sense. 'Bhalo' in Bengali means 'Good' and 'Bhalo' in Gujarati means 'Spear'.

6) Discourse Integration: The meaning of Individual sentence is depending on previous sentence.
Ex: Bill had a red balloon. John wanted it.

7) Pragmatic Analysis: Understanding the text & dialogues. It derives Knowledge from external commonsense information.
Ex: Do you know what time it is?
It does not mean the speaker asking you the time!!!
We should understand what to do
Why NLP is difficult?
NLP is difficult because Ambiguity and Uncertainty exist in the language.
Ambiguity- There are the following three ambiguity -

- **Lexical Ambiguity**-Lexical Ambiguity exists in the presence of two or more possible meanings of the sentence within a single word.

Example: Manya is looking for a match.
In the above example, the word match refers to that either Manya is looking for a partner or Manya is looking for a match. (Cricket or other match)

- **Syntactic Ambiguity-** Syntactic Ambiguity exists in the presence of two or more possible meanings within the sentence.

Example: I saw the girl with the binocular.
In the above example, did I have the binoculars? Or did the girl have the binoculars?

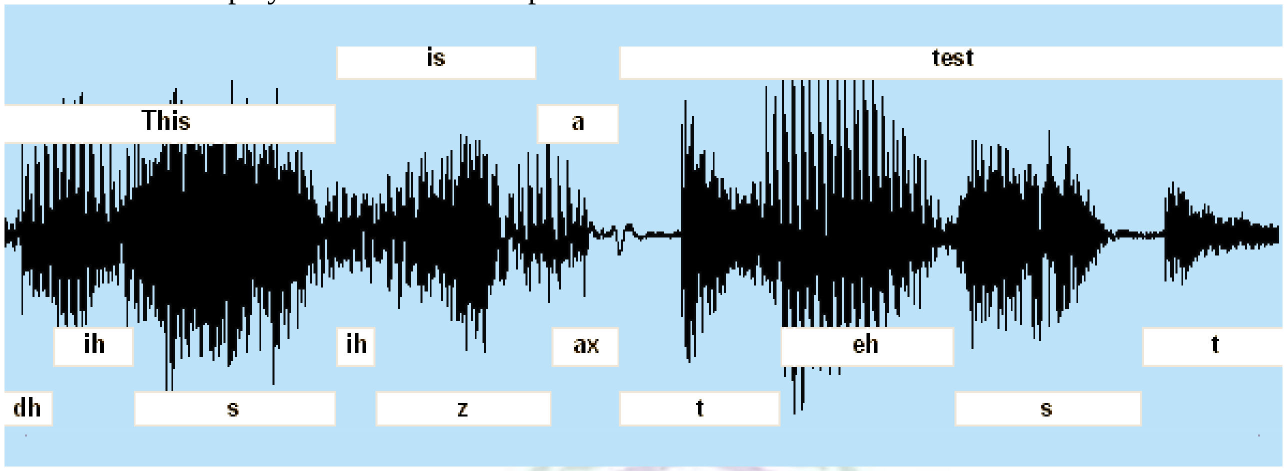
- **Referential Ambiguity-** Referential Ambiguity exists when you are referring to something using the pronoun.

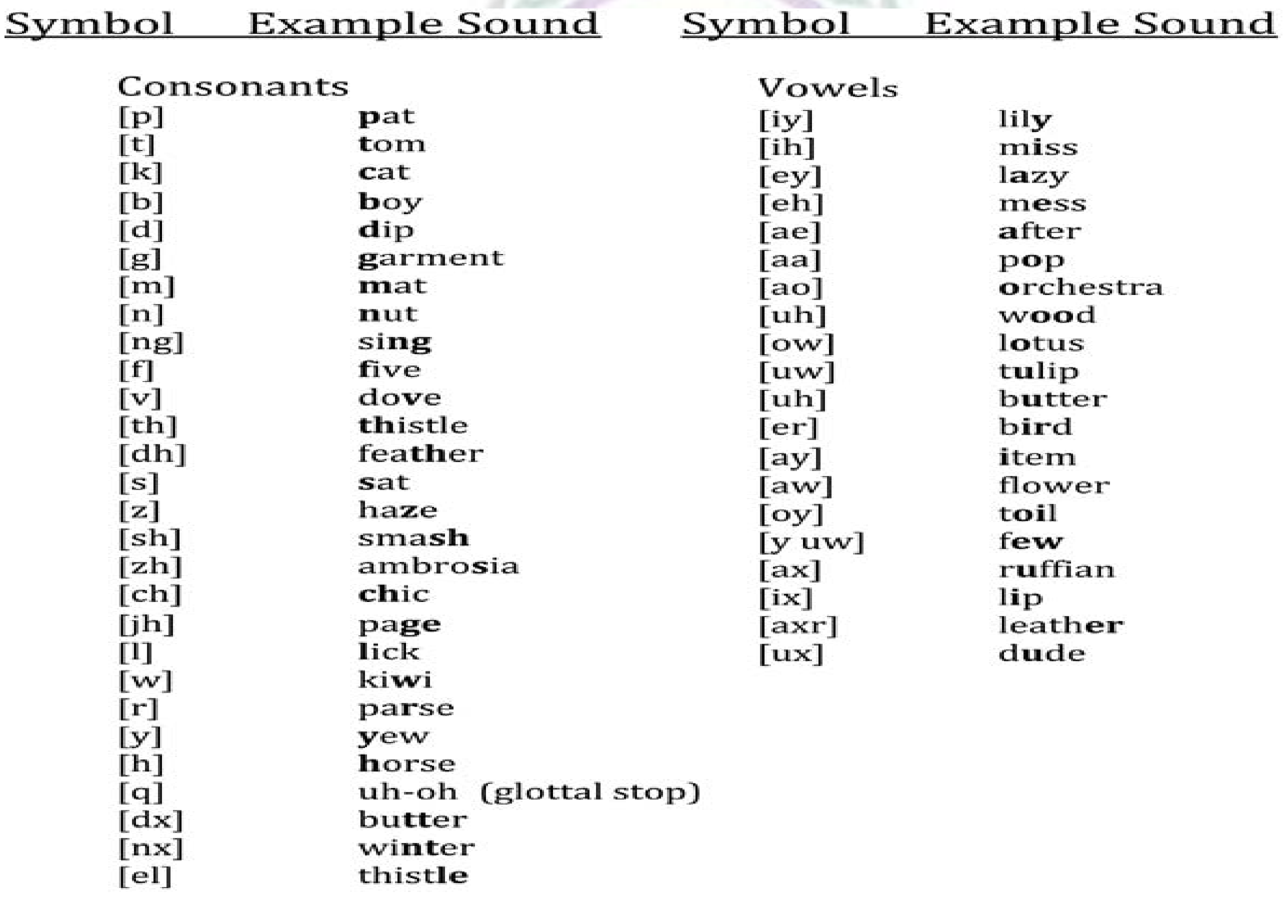
Example: Kiran went to Sunita. She said, "I am hungry."
In the above sentence, you do not know that who is hungry, either Kiran or Sunita.

Automatic Speech Recognition(ASR)

- Speech recognition is meant the process of converting an acoustic stream of speech input, as gathered by a microphone and associated electronic equipment, into a text representation of its component words
- Speech understanding in contrast, requires that what is spoken be understood

- Understanding speech is more difficult than understanding text because there is the additional problem of processing the speech waveform to extract the words being uttered. Speech, as it is captured by a microphone, is converted into an electronic signal or waveform, which can be displayed on an oscilloscope
- Speech, as it is captured by a microphone, is converted into an electronic signal or waveform, which can be displayed on an oscilloscope





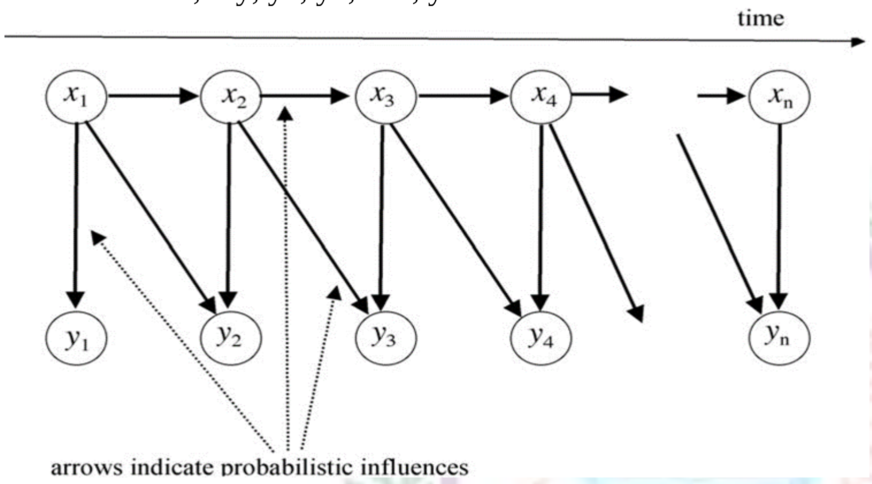
Symbol	Example Sound	Symbol	Example Sound
Consonants		Vowels	
[p]	pat	[iy]	lily
[t]	tom	[ih]	miss
[k]	cat	[ey]	lazy
[b]	boy	[eh]	mess
[d]	dip	[ae]	after
[g]	garment	[aa]	pop
[m]	mat	[ao]	orchestra
[n]	nut	[uh]	wood
[ng]	sing	[ow]	lotus
[f]	five	[uw]	tulip
[v]	dove	[uh]	butter
[th]	thistle	[er]	bird
[dh]	feather	[ay]	item
[s]	sat	[aw]	flower
[z]	haze	[oy]	toil
[sh]	smash	[y uw]	few
[zh]	ambrosia	[ax]	ruffian
[ch]	chic	[ix]	lip
[jh]	page	[axr]	leather
[l]	lick	[ux]	dude
[w]	kiwi		
[r]	parse		
[y]	yew		
[h]	horse		
[q]	uh-oh (glottal stop)		
[dx]	butter		
[nx]	winter		
[el]	thistle		

DRAGON

- James K. Baker, began work on a speech understanding system he called “DRAGON.
- DRAGON was designed to understand sentences about chess moves.
- DRAGON introduced powerful new techniques for speech processing
- It used statistical techniques to make guesses about the most probable strings of words that might have produced the observed speech signal.
- , suppose we let x stand for a string of words and y stand for the speech waveform that is produced when x is spoken. (Actually, we’ll let y be some information-preserving representation of the

waveform in terms of its easily measurable properties such as the amounts of energy the waveform contains in various frequency bands

- suppose we let x stand for a string of words and y stand for the speech waveform that is produced when x is spoken
- Because the same speaker may say the same words somewhat differently on different occasions, and different speakers certainly will say them differently, the word string x does not completely determine what the speech waveform y will be. That is, given any x, we can only say what the probabilities of the different y's might be
- these probabilities are written in functional form as p(y x) (read as the "probability of y given x")
- IDEA
- The string of words, in turn, gives rise to a string of phones – the phonetic units. Finally, the phone string is expressed by a speech waveform at the bottom of the hierarchy.
- At each level, we have a sequence of entities, say, x1, x2, . . . xn, producing a sequence of other entities, say, y1, y2, . . . , yn



The DRAGON system made some simplifying assumptions. It assumed that each x_i in the sequence of x's is influenced only by its immediate precedent, x_{i-1}, and not by any other of the x_i. This assumption is called the Markov assumption.

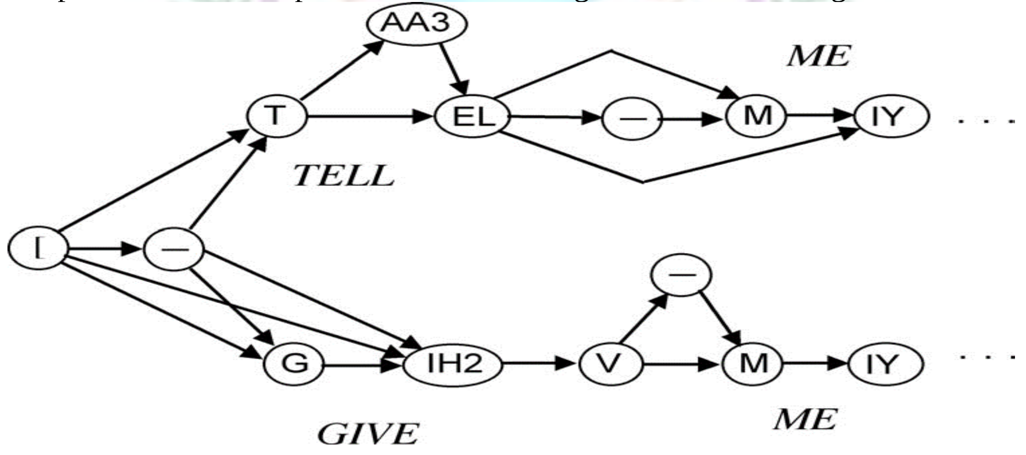
- Google uses the Markov assumption, for example, in its computation of page rank
- it was assumed that each y_i was influenced only by x_i and x_{i-1}. All of these "influences" are probabilistic. That is, given quantities like x_3 and x_4, for example, the value of y_4 is not completely determined. One can only say what the probabilities of the values of y_4 might be; these are given by the functional expression $p(y_4\ x_3, x_4)$.
- DRAGON combined these separate levels into a network consisting of a hierarchy of probabilistic functions of Markov processes. Entities representing segments of the speech waveform were at the bottom, entities representing phones were in the middle, and entities representing words were at the top. At each level, Bayes's rule was used to compute probabilities of the x's given the y's. Because only the speech waveform at the bottom level was actually observed, the phones and words were said to be "hidden." For this reason, the entire network employed hidden Markov models (HMMs). DRAGON was the first example of the use of HMMs in AI.

- Using this network, recognition of an utterance was then achieved by finding the highest probability path through the network. Computing the probabilities for syntactically valid word sequences, given the sequence of segments of the observed speech waveform, is a problem that is similar to one I described earlier, namely, computing the confidences of strings of characters

HARPY

- HARPY was a second system produced at CMU under DARPA's speech understanding research effort
- Bruce T. Lowerre designed and implemented the system as part of his Ph.D. research.

- HARPY combined some of the ideas of HEARSAY-I and DRAGON
- HARPY used heuristic search methods
- Versions of HARPY were developed for understanding spoken sentences about several different task areas
- The main one involved being able to answer questions about, and to retrieve documents from, a database containing summaries (called "abstracts") of AI papers. Here are some examples:
- Which abstracts refer to theory of computation?" "List those articles."
- "Are any by Feigenbaum and Feldman?"
- "What has McCarthy written since nineteen seventy-four?"
- HARPY could handle a vocabulary of 1,011 words
- HARPY used what is called a "semantic grammar," one that has expanded categories such as Topic, Author, Year, and Publisher that were semantically related to its subject area, namely, data about AI papers
- The network was constructed from what were called "knowledge sources" (KSs), which consisted of information needed for the recognition process
- A second knowledge source used by HARPY described how each word in HARPY's vocabulary might be pronounced
- successful recognition of word boundaries requires a third knowledge source dealing with such phenomena
- A fourth knowledge source specified the phones involved in the pronunciation of words and transitions between words.
- HARPY combined all of this knowledge into a giant network of phones representing all the possible ways that syntactically legal sentences might be spoken
- Each "phone node" in the network was paired with a representation of a segment of a speech waveform, called a "spectral template," expected to be associated with that particular phone
- These templates were obtained initially by having a speaker read about 700 sentences. They could be "tuned" for a new speaker by having the speaker read about 20 selected sentences during a "learning" session. A partial network of phones is shown in Figure toillustrate the general idea



- HARPY's actual network had 15,000 nodes. The network is for those parts of the sentences that begin with "Tell me. . . " and "Give me. " The symbols inside the nodes represent phones, using
- DRAGON's notation for them. Arrows represent possible transitions from one phone to the next. Note that there are multiple paths, corresponding to different ways to pronounce the words.
- To recognize the words in a spoken sentence, the observed speech waveform was first divided into variable-length segments that were guessed to correspond to the sequence of phones in the waveform. A spectral template was computed for each of these segments. The recognition process then proceeded as follows: The spectral template corresponding to the first spectral segment in the speech waveform was compared against all of the templates corresponding to the phones at the beginning of the network.
- HARPY's method of searching for a best path through the network can be compared with the A∗ heuristic search process described earlier. Whereas A∗ kept the entire search "frontier" available for

possible further searching, HARPY kept on its frontier only those nodes on the best few paths found so far.

- HARPY's designers called this technique "beam search" because the nodes visited by the search process were limited to a narrow beam through the network. Because nodes not in the beam were eliminated as the process went on, it is possible that the best complete path found by HARPY might not be the overall best one in the network
- Even so, the path found usually corresponded to a correct interpretation of the spoken sentence.
- At the end of the DARPA speech understanding project, HARPY was tested on 100 sentences spoken by three male and two female speakers. It was able to understand over 95% of these sentences correctly, thereby meeting DARPA's goal of less that 10% error
- HARPY was the only system to meet DARPA's goals

MACHINE VISION :

INTRODUCTION:

Machine vision is used more and more frequently in industrial practice as well as applications for service equipment, military technology and in consumer products, toys and model making. It is becoming increasingly available, and in common practice, every university educated mechanical engineer encounters it at some point.

Machine vision is a field of science trying to mimic some aspects of human vision. Beside own image perceptions, intelligence of a person and his/her previous experience have a significant impact on evaluation of image information. Machine vision works similarly. A mere capture and transmission of image provides basically nothing in the field of automation. However, with image processing, it is possible to achieve certain, limited (in comparison to human) intelligence of the given machine. That is due to the fact that image processing allows extracting necessary information from the given image for the given type of task. This information is then used by the machine control system for decision-making.

Knowledge of machine vision principles is not crucial only when designing a visualisation system, but when designing technological operation before and after the visual evaluation as well. When operating the system, knowledge of the principles are important as well, since the operators may prevent a malfunction or shorten the downtime.

APPLICATIONS OF MACHINE VISION

- Absence/presence detection
- Automated vision testing and measurement
- Bar code reading
- Colour verification
- Defect detection
- Optical character recognition and verification
- Part verification
- Pattern matching
- Sorting
- Traceability
- Vision guided robots

BASIC PRINCIPLES OF VISION

- Vision inspires creativity, ignites purpose and innovation, and fosters growth. It provides a blueprint and allows a business to stay on track and have a gauge to properly measure your progress.
- Vision along with a company's mission, purpose, values, and core value are the foundation of every business. Just as a house cannot be built without a foundation and blueprint, neither can a business.

Without clear vision:

- Inconsistency in work product or service
- Stagnation
- Decreased revenue and profits
- Loss of customers

- High employee turnover
- High level of chaos
- Poor customer service
- No real direction or objectives to meet
- Continually addressing the same problems
- Most time is spent on putting out fires
- Continually hiring the wrong people
- Unproductive communication
- Lack of leadership and accountability

With clear vision:

- Consistency in work product or service
- Growth
- Profitability
- Increase in ideal clients
- Customer retention
- Abundance of referrals
- Structure and order
- Time to work on business and strategies
- Strategic objectives being set and met
- Policies and procedures in place
- Award winning customer service
- Positions being filled by the right people
- Low employee turnover
- Productive and effective communication
- High level of leadership and accountability

MACHINE VISION TECHNIQUES :

Computer vision concepts can be broadly categorized as low, mid and high level vision techniques.

- Low level vision constitutes of image processing techniques, feature detection and matching and early segmentation.
- Mid-level vision is where things start to come together attributing meaning.
- High level vision tasks are the algorithms which makes sense of the visual content and make computer vision live up to the capabilities of human vision.

Low-level vision or early vision considers local properties of an image. For instance, this image has several edges. When we move to mid-level vision, so that's where we start to group things together. Preliminary analysis on these pixels tells us there is an object as well as a background in this picture. Now, moving on to the high-level vision, that's where we identify what's there in the image. In this instance, it's an aeroplane.

AI Today & Tomorrow:

Achievements:

AI at Home

Thermostats for heat and air-conditioning systems that anticipate temperature changes and the needs of occupants, communicate with other home devices, and take appropriate actions in advance

- Microwave ovens that read barcodes on packages to determine how long to cook an item;
- Smart running shoes with a computer chip that senses the runner's size and stride length and directs on-going changes in the heal cushioning via a miniature screw and cable system;
- Washing machines that automatically adjust to different conditions to wash clothes better;
- Refrigerators that automatically inventory their contents and inform owners of needed items;
- Cameras with computer vision systems to identify faces and to control focusing, exposure, and framing.
- Hearing aids that adapt to ambient sound levels and block out "cocktail party" chatter.
- Robotic pet "animals" and toys that interact with people.
- Floor-washing and vacuum-cleaning robots; and

- Caretaker robots for the elderly or infirm
- **Advanced Driver Assistance Systems**
  Adaptive cruise control (ACC) for providing more intelligent control of Speed, enabling the vehicle to slow down or speed up depending on Traffic conditions as perceived by radar or laser sensors.

• Intelligent speed adaptation (ISA) for monitoring local speed limits, Slowing the vehicle down (or warning the driver) when it enters zones with speed limits.
• Lane control systems for monitoring the presence of vehicles or
Obstructions in adjacent lanes and for monitoring when a driver drifts into an adjacent lane or off the roadway.
• Automatic parking systems for assisting a driver when executing a Parallel parking manoeuvre;
• Traffic sign recognition systems.
• Driver drowsiness detection systems; and
• Intelligent tire pressure control systems

- **Route Finding in Maps**
  Many automobiles have devices that "talk" you to your destination using on-board GPS systems, map databases, and speech synthesis.
- Most commonly used graph-searching procedure is a∗ , a heuristic search method that takes into account both the distance travelled so far and an estimate of the distance to the goal
- Most route-finding programs can (and do) take into account criteria other than distance, such as estimated travel times. For example, Microsoft's clear flow program, which uses Bayesian networks informed by traffic-monitoring sensors to estimate traffic densities, can base route recommendations either on shortest time, shortest distance, or current traffic conditions.

You Might Also Like. . .

- Amazon's recommendations are based on what is called social or collaborative filtering. A database of preferences (for books, movies, or whatever) is maintained for every user.
- Another type of recommending system uses what is called content-based filtering, in which a user's preferences for books, movies, documents, or whatever are analysed to find similarities with other items of the same kind (instead of with other users having the same preferences).
  Computer Games
- The game Black and White 2, developed by Lionhead Studios and marketed by Electronic Arts, uses a combination of neural nets and decision trees. According to a Web site for the game, the NPCs (evil and benevolent deities) "can learn strategies, master new abilities and skills, [and] lead armies into battle. . . Every choice you make will have an impact. Each action and inaction prompts obvious changes to buildings, flora and fauna, all morphing to reflect your personality
- The game F.E.A.R. (First Encounter Assault Recon) by Jeff Orkin and Monolith Productions uses A∗ to plan sequences of NPC actions in addition to its usual role in path finding
- Computer games provide us with a source of cheap, reliable, and flexible technology for developing our own virtual environments for research

UBIQUITOUS AI

Ubiquitous means present, appearing, or found everywhere.

- For brakes that know how to stop on wet pavement
- For instruments that can converse with their users
- For bridges that watch out for the safety of those who cross them
- For streetlights that care about those who stand under them – who know the way, so no one need get lost
- For little boxes that make out your income tax for you.