

# FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE

## MODULE - I

### INTRODUCTION AND BASIC SEARCH STRATEGIES

#### What is Artificial Intelligence?

- In today's world, technology is growing very fast, and we are getting in touch with different new technologies day by day.
  - Here, one of the booming technologies of computer science is Artificial Intelligence which is ready to create a new revolution in the world by making intelligent machines. The Artificial Intelligence is now all around us. It is currently working with a variety of subfields, ranging from general to specific, such as self-driving cars, playing chess, proving theorems, playing music, Painting, etc.
  - AI is one of the fascinating and universal fields of Computer science which has a great scope in future. AI holds a tendency to cause a machine to work as a human.
  - Artificial Intelligence is composed of two words **Artificial** and **Intelligence**, where Artificial defines "*man-made*," and intelligence defines "*thinking power*", hence AI means "*a man-made thinking power*."
  - So, we can define AI as:
- "It is a branch of computer science by which we can create intelligent machines which can behave like a human, think like humans, and able to make decisions."

## **Why Artificial Intelligence?**

Before Learning about Artificial Intelligence, we should know that what is the importance of AI and why should we learn it. Following are some main reasons to learn about AI:

- With the help of AI, you can create such software or devices which can solve real-world problems very easily and with accuracy such as health issues, marketing, traffic issues, etc.
- With the help of AI, you can create your personal virtual Assistant, such as Cortana, Google Assistant, Siri, etc.
- With the help of AI, you can build such Robots which can work in an environment where survival of humans can be at risk.
- AI opens a path for other new technologies, new devices, and new Opportunities.

## **Goals of Artificial Intelligence**

Following are the main goals of Artificial Intelligence:

1. Replicate human intelligence
2. Solve Knowledge-intensive tasks
3. An intelligent connection of perception and action
4. Building a machine which can perform tasks that requires human intelligence such as:
  - Proving a theorem
  - Playing chess
  - Plan some surgical operation
  - Driving a car in traffic

5. Creating some system which can exhibit intelligent behavior, learn new things by itself, demonstrate, explain, and can advise to its user.

## **What Comprises to Artificial Intelligence?**

Artificial Intelligence is not just a part of computer science even it's so vast and requires lots of other factors which can contribute to it. To create the AI first we should know that how intelligence is composed, so the Intelligence is an intangible part of our brain which is a combination of **Reasoning, learning, problem-solving perception, language understanding, etc.**

To achieve the above factors for a machine or software Artificial Intelligence requires the following discipline:

- Mathematics
- Biology
- Psychology
- Sociology
- Computer Science
- Neurons Study
- Statistics

## **Advantages of Artificial Intelligence**

- High Accuracy with less errors
- High-Speed
- High reliability
- Useful for risky areas
- Digital Assistant
- Useful as a public utility

## **Disadvantages of Artificial Intelligence**

- High Cost
- Can't think out of the box
- Increase dependency on machines
- No Original Creativity

## **Application of AI**

Artificial Intelligence has various applications in today's society. It is becoming essential for today's time because it can solve complex problems with an efficient way in multiple industries, such as Healthcare, entertainment, finance, education, etc. AI is making our daily life more comfortable and fast.

### **1. AI in Healthcare**

- In the last, five to ten years, AI becoming more advantageous for the healthcare industry and going to have a significant impact on this industry.
- Healthcare Industries are applying AI to make a better and faster diagnosis than humans. AI can help doctors with diagnoses and can inform when patients are worsening so that medical help can reach to the patient before hospitalization.

### **2. AI in Gaming**

- AI can be used for gaming purpose. The AI machines can play strategic games like chess, where the machine needs to think of a large number of possible places.

### **3. AI in Finance**

- AI and finance industries are the best matches for each other. The finance industry is implementing automation, chatbot, adaptive intelligence, algorithm trading, and machine learning into financial processes.

### **4. AI in Data Security**

- The security of data is crucial for every company and cyber-attacks are growing very rapidly in the digital world. AI can be used to make your data more safe and secure. Some examples such as AEG bot, AI2 Platform, are used to determine software bug and cyber-attacks in a better way.

### **5. AI in Social Media**

- Social Media sites such as Facebook, Twitter, and Snap chat contain billions of user profiles, which need to be stored and managed in a very efficient way. AI can organize and manage massive amounts of data. AI can analyze lots of data to identify the latest trends, hash tag, and requirement of different users.

### **6. AI in Robotics:**

- Artificial Intelligence has a remarkable role in Robotics. Usually, general robots are programmed such that they can perform some repetitive task, but with the help of AI, we can create intelligent robots which can perform tasks with their own experiences without pre-programmed.
- Humanoid Robots are best examples for AI in robotics, recently the intelligent Humanoid robot named as Erica and Sophia has been developed which can talk and behave like humans.

### **7. AI in Entertainment**

- We are currently using some AI based applications in our daily life with some entertainment services such as Netflix or Amazon. With the help of ML/AI algorithms, these services show the recommendations for programs or shows.

## **8. AI in E-commerce**

- AI is providing a competitive edge to the e-commerce industry, and it is becoming more demanding in the e-commerce business. AI is helping shoppers to discover associated products with recommended size, color, or even brand.

## **9. AI in education:**

- AI can automate grading so that the tutor can have more time to teach. AI chatbot can communicate with students as a teaching assistant.
- AI in the future can be work as a personal virtual tutor for students, which will be accessible easily at any time and any place

## **History of Artificial Intelligence:**

- Maturation of Artificial Intelligence (1943-1952)
- The birth of Artificial Intelligence (1952-1956)
- The golden years-Early enthusiasm (1956-1974)
- Deep learning, big data and artificial general intelligence (2011-present)

## **Types of Artificial Intelligence:**

### **1. Weak AI or Narrow AI:**

- Narrow AI is a type of AI which is able to perform a dedicated task with intelligence. The most common and currently available AI is Narrow AI in the world of Artificial Intelligence.
- Narrow AI cannot perform beyond its field or limitations, as it is only trained for one specific task. Hence it is also termed as weak AI. Narrow AI can fail in unpredictable ways if it goes beyond its limits.
- Apple Siri is a good example of Narrow AI, but it operates with a limited pre-defined range of functions.

- IBM's Watson supercomputer also comes under Narrow AI, as it uses an Expert system approach combined with Machine learning and natural language processing.
- Some Examples of Narrow AI are playing chess, purchasing suggestions on e-commerce site, self-driving cars, speech recognition, and image recognition.

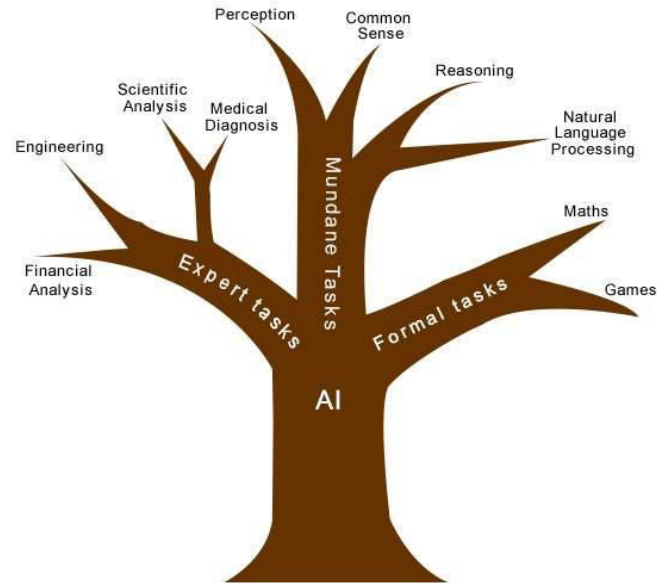
## **2. General AI:**

- General AI is a type of intelligence which could perform any intellectual task with efficiency like a human.
- The idea behind the general AI to make such a system which could be smarter and think like a human by its own.
- Currently, there is no such system exist which could come under general AI and can perform any task as perfect as a human.
- The worldwide researchers are now focused on developing machines with General AI.
- As systems with general AI are still under research, and it will take lots of efforts and time to develop such systems.

## **3. Super AI:**

- Super AI is a level of Intelligence of Systems at which machines could surpass human intelligence, and can perform any task better than human with cognitive properties. It is an outcome of general AI.
- Some key characteristics of strong AI include capability include the ability to think, to reason,solve the puzzle, make judgments, plan, learn, and communicate by its own.
- Super AI is still a hypothetical concept of Artificial Intelligence. Development of such systems in real is still world changing task.

# AI PROBLEMS:



Task Domains of Artificial Intelligence			
Mundane Tasks	(Ordinary)	Formal Tasks	Expert Tasks
Perception		<ul style="list-style-type: none"> <li>Mathematics</li> <li>Geometry</li> </ul>	<ul style="list-style-type: none"> <li>Engineering</li> <li>Fault Finding</li> </ul>
<ul style="list-style-type: none"> <li>Computer Vision</li> </ul>			



<ul style="list-style-type: none"> <li>• Speech, Voice</li> </ul>	<ul style="list-style-type: none"> <li>• Logic</li> <li>• Integration and Differentiation</li> </ul>	<ul style="list-style-type: none"> <li>• Manufacturing</li> <li>• Monitoring</li> </ul>
<p>Natural Language Processing</p> <ul style="list-style-type: none"> <li>• Understanding</li> <li>• Language Generation</li> <li>• Language Translation</li> </ul>	<p>Games</p> <ul style="list-style-type: none"> <li>• Go</li> <li>• Chess (Deep Blue)</li> <li>• Ckeckers</li> </ul>	<p>Scientific Analysis</p>
<p>Common Sense</p>	<p>Verification</p>	<p>Financial Analysis</p>
<p>Reasoning</p>	<p>Theorem Proving</p>	<p>Medical Diagnosis</p>
<p>Planing</p>		<p>Creativity</p>
<p>Robotics</p> <ul style="list-style-type: none"> <li>• Locomotive</li> </ul>		

- Humans learn **mundane (ordinary) tasks** since their birth. They learn by perception, speaking, using language, and locomotives. They learn Formal Tasks and Expert Tasks later, in that order.
- For humans, the mundane tasks are easiest to learn. The same was considered true before trying to implement mundane tasks in machines. Earlier, all work of AI was concentrated in the mundane task domain.
- Later, it turned out that the machine requires more knowledge, complex knowledge representation, and complicated algorithms for handling mundane tasks. This is the reason **why AI work is more prospering in the**

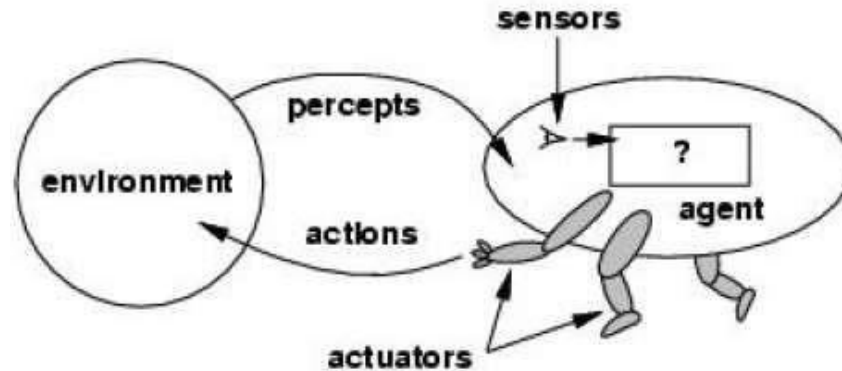
**Expert Tasks domain** now, as the expert task domain needs expert knowledge without common sense, which can be easier to represent and handle.

## **AGENTS:**

Rationality concept can be used to develop a smallest of design principle for building successful agents; these systems are reasonably called as Intelligent.

## **AGENTS AND ENVIRONMENT:**

- An **agent** is anything that can be viewed as perceiving its **environment** through **sensors** and **SENSOR** acting upon that environment through **actuators**. This simple idea is illustrated in Figure.
- A human agent has eyes, ears, and other organs for sensors and hands, legs, mouth, and other body parts for actuators.
- A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators.
- A software agent receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.



- **Percept**

We use the term **percept** to refer to the agent's perceptual inputs at any given instant.

- **Percept Sequence**

An agent's **percept sequence** is the complete history of everything the agent has ever perceived.

- **Agent function**

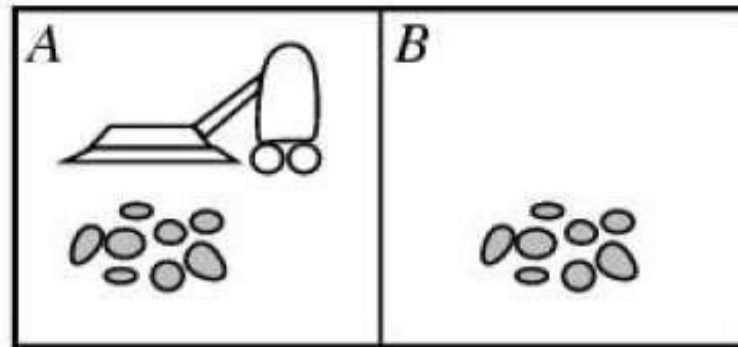
Mathematically speaking, we say that an agent's behavior is described by the **agent**

- **Function**

that maps any given percept sequence to an action.  $f : P^* \rightarrow A$

## Agent program

- The agent function for an artificial agent will be implemented by an **agent program**.
- It is important to keep these two ideas distinct.
- The agent function is an abstract mathematical description;
- The agent program is a concrete implementation, running on the agent architecture.
- To illustrate these ideas, we will use a very simple example-the vacuum- cleaner world shown in Figure.
- This particular world has just two locations: squares A and B.
- The vacuum agent perceives which square it is in and whether there is dirt in the square.
- It can choose to move left, move right, suck up the dirt, or do nothing.
- One very simple agent function is the following:
- if the current square is dirty, then suck, otherwise,
- it move to the other square.
- A partial tabulation of this agent function is shown in Figure.



## Agent function

Percept Sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
.....	.....

## Agent program

**function** Reflex-VACUUM-AGENT ([locations, status])

**returns an action if** status = Dirty **then return** Suck

**else if** location = A **then return** Right

**elseif** location = B **then return** Left

## STRUCTURE OF AGENTS:

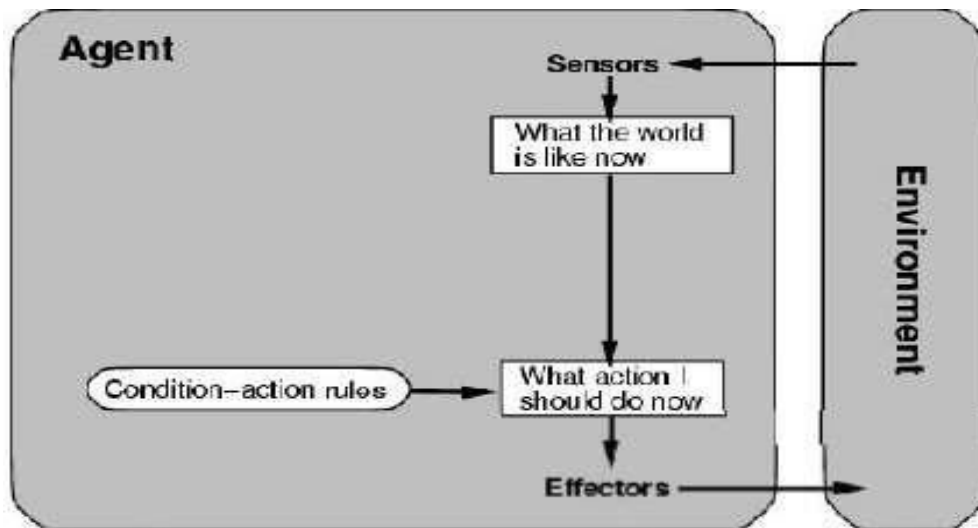
Agents can be grouped into five classes based on their degree of perceived intelligence and capability. All these agents can improve their performance and generate better action over the time. These are given below:

- Simple Reflex Agent
- Model-based reflex agent
- Goal-based agents
- Utility-based agent
- Learning agent

### 1. Simple Reflex agent:

- The Simple reflex agents are the simplest agents. These agents take decisions on the basis of the current percepts and ignore the rest of the percept history.
- These agents only succeed in the fully observable environment.
- The Simple reflex agent does not consider any part of percepts history during their decision and action process.

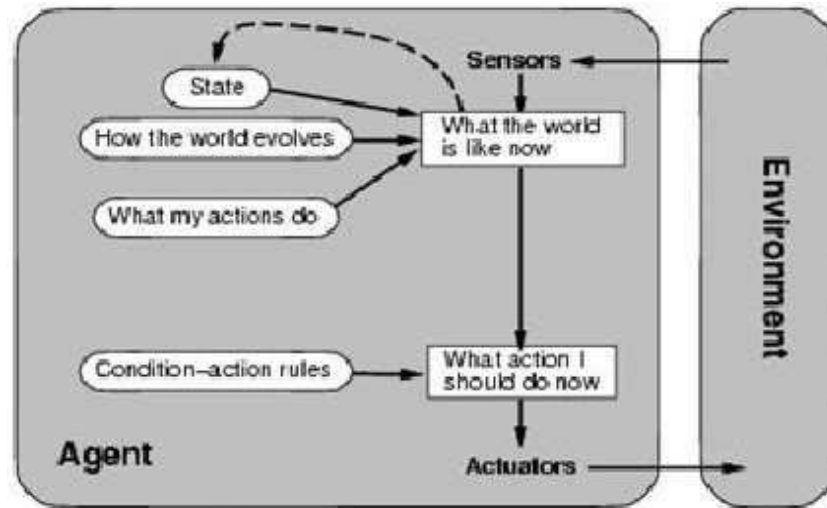
- The Simple reflex agent works on Condition-action rule, which means it maps the current state to action. Such as a Room Cleaner agent, it works only if there is dirt in the room.
- Problems for the simple reflex agent design approach:
  - They have very limited intelligence
  - They do not have knowledge of non-perceptual parts of the current state
  - Mostly too big to generate and to store.
  - Not adaptive to changes in the environment.
- *function REFLEX-VACUUM-AGENT ([location, status]) return an action if status == Dirty then return Suck*
- *else if location == A then return Right else if location == B then return Left*



## 2. Model-based reflex agent

- The Model-based agent can work in a partially observable environment, and track the situation.
- A model-based agent has two important factors:
  - **Model:** It is knowledge about "how things happen in the world," so it is called a Model-based agent.
  - **Internal State:** It is a representation of the current state based on percept history.
- These agents have the model, "which is knowledge of the world" and based on the model they perform actions.
- Updating the agent state requires information about:
  - a. How the world evolves
  - b. How the agent's action affects the world.





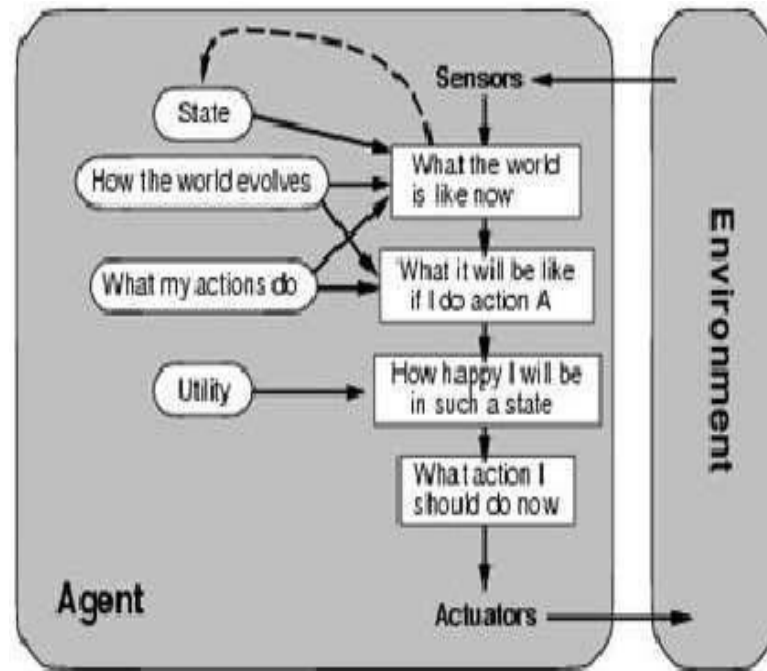
### 3. Goal-based agents

- The knowledge of the current state environment is not always sufficient to decide for an agent to what to do.
- The agent needs to know its goal which describes desirable situations.
- Goal-based agents expand the capabilities of the model-based agent by having the "goal" information.
- They choose an action, so that they can achieve the goal.
- These agents may have to consider a long sequence of possible actions before deciding whether the goal is achieved or not. Such considerations of different scenario are called searching and planning, which makes an agent proactive.



#### 4. Utility-based agents

- These agents are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.
- Utility-based agent act based not only goals but also the best way to achieve the goal.
- The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.
- The utility function maps each state to a real number to check how efficiently each action achieves the goals.

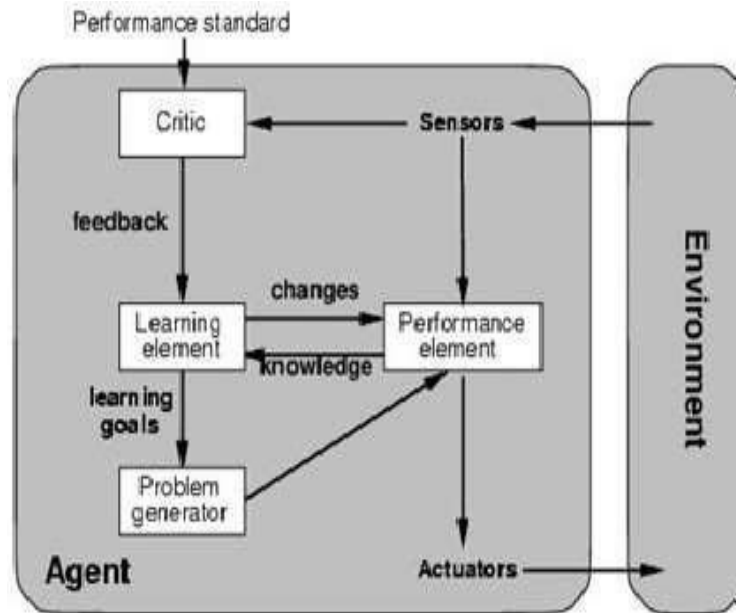


## 5. Learning Agents

- A learning agent in AI is the type of agent which can learn from its past experiences, or it has learning capabilities.
- It starts to act with basic knowledge and then able to act and adapt automatically through learning.
- A learning agent has mainly four conceptual components, which are:
  - a. **Learning element:** It is responsible for making improvements by learning from environment
  - b. **Critic:** Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.

- c. **Performance element:** It is responsible for selecting external action
- d. **Problem generator:** This component is responsible for suggesting actions that will lead to new and informative experiences.

Hence, learning agents are able to learn, analyze performance, and look for new ways to improve the performance.



## PROBLEM SOLVING AGENTS:

- A Problem solving agent is a goal-based agent.
- It decides what to do by finding sequence of actions that lead to desirable states.
- The agent can adopt a goal and aim at satisfying it.
- To illustrate the agent's behavior
- For example where our agent is in the city of Arad, which is in Romania. The agent has to adopt a goal of getting to Bucharest.
- Goal formulation, based on the current situation and the agent's performance measure, is the first step in problem solving.
- The agent's task is to find out which sequence of actions will get to a goal state.
- Problem formulation is the process of deciding what actions and states to consider given a goal.

### **Example: Route finding problem**

On holiday in Romania : currently in Arad. Flight leaves tomorrow from Bucharest

**Formulate goal:** be in Bucharest

- **Formulate problem:**

**states:** various cities

**actions:** drive between cities

**Find solution:** sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

- Goal formulation and problem formulation A **problem** is defined by four

items:

**initial state** e.g., "at Arad"

**successor function**  $S(x)$  = set of action-state pairs e.g.,  $S(\text{Arad}) = \{ [\text{Arad} \rightarrow \text{Zerind}; \text{Zerind}], \dots \}$

**goal test**, can be explicit, e.g.,  $x = \text{at Bucharest}$  implicit, e.g.,  $\text{NoDirt}(x)$

**path cost** (additive)

e.g., sum of distances, number of actions executed, etc.  $c(x; a; y)$  is the step cost, assumed to be  $\geq 0$

- **A solution** is a sequence of actions leading from the initial state to a goal state.

## Search

- An agent with several immediate options of unknown value can decide what to do by examining different possible sequences of actions that leads to the states of known value and then choosing the best sequence.
- The process of looking for sequences actions from the current state to reach the goal state is called **search**.
- The **search algorithm** takes a **problem** as **input** and returns a **solution** in the form of **action sequence**.
- Once a solution is found the **execution phase** consists of carrying out the recommended action.
- The following shows a simple "formulate, search, execute" design for the agent.
- Once solution has been executed, the agent will formulate a new goal.
- It first formulates a **goal** and a **problem** searches for a sequence of actions that would solve a problem and executes the actions one at a time.

The agent design assumes the Environment is

1. **Static:** The entire process carried out without paying attention to changes that might be occurring in the environment.
  2. **Observable :** The initial state is known and the agent's sensor detects all aspects that are relevant to the choice of action
  3. **Discrete :** With respect to the state of the environment and percepts and actions so that alternate courses of action can be taken
  4. **Deterministic:** The next state of the environment is completely determined by the current state and the actions executed by the agent. Solutions to the problem are single sequence of actions
- An agent carries out its plan with eye closed. This is called an open loop system because ignoring the percepts breaks the loop between the agent and the environment.

## Well-defined problems and solutions

- A **problem** can be formally defined by **four components**:
- The **initial state** that the agent starts in . The initial state for our agent of example problem is described by  $In(Arad)$
- A **Successor Function** returns the possible **actions** available to the agent.
- Given a state  $x$ ,  $SUCCESSOR-FN(x)$  returns a set of  $\{action, successor\}$  ordered pairs where each action is one of the legal actions in state  $x$ , and each successor is a state that can be reached from  $x$  by applying the action.
- For example, from the state  $In(Arad)$ , the successor function for the Romania problem would return  $\{ [Go(Sibiu), In(Sibiu)], [Go(Timisoara), In(Timisoara)], [Go(Zerind), In(Zerind)] \}$

**State Space:** The set of all states reachable from the initial state. The state space forms a graph in which the nodes are states and the arcs between nodes are actions.

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

The **goal test** determines whether the given state is a goal state.

A **path cost** function assigns numeric cost to each action.

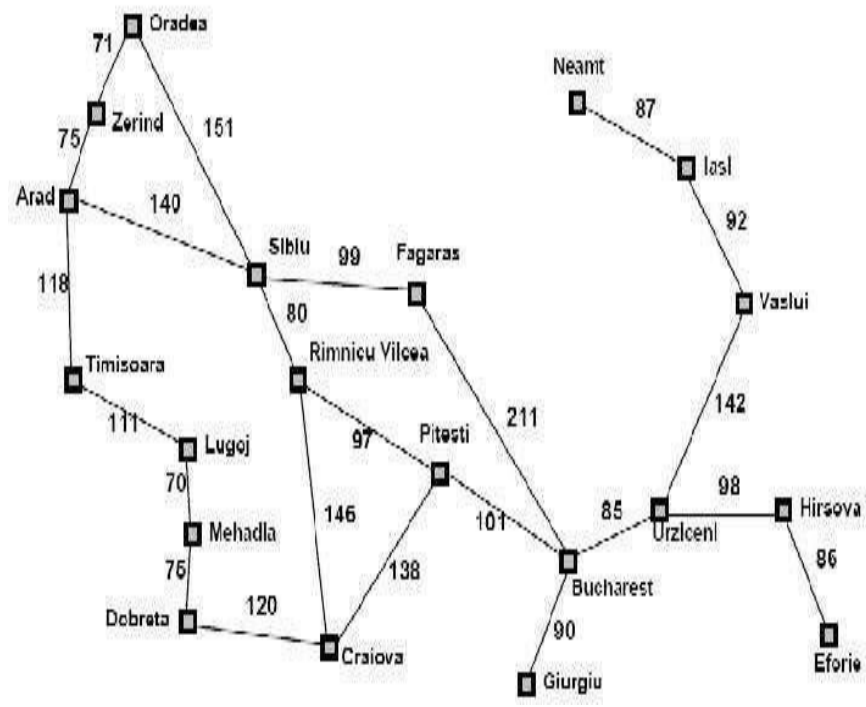
For the Romania problem the cost of path might be its length in kilometers.

The **step cost** of taking action  $a$  to go from state  $x$  to state  $y$  is denoted by  $c(x, a, y)$ . It is assumed that the step costs are non negative.

A **solution** to the problem is a path from the initial state to a goal state.

An **optimal solution** has the lowest path cost among all solutions.





Straight line distance to Bucharest:

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

## **TOY PROBLEMS**

- Vacuum World Example
- 8-puzzle
- 8 -queens problem

## **REAL WORLD PROBLEMS**

- Route Finding Problem
- Touring Problems
- Travelling Salesman Problem
- Robot Navigation

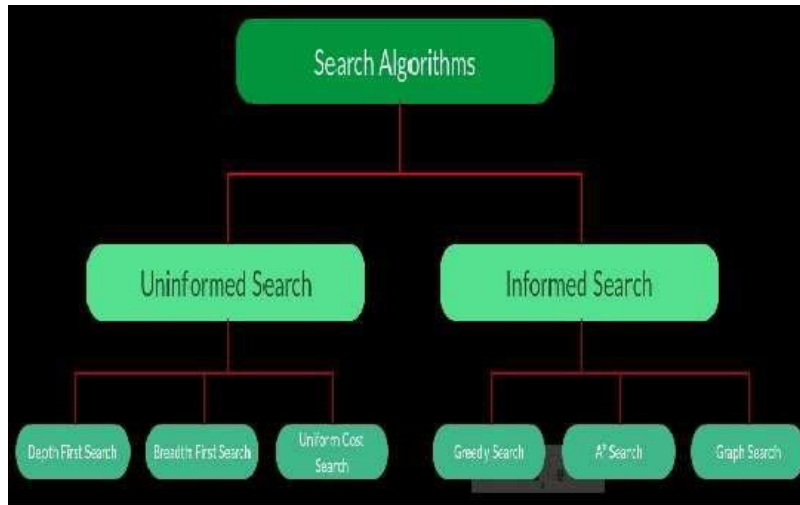
## **BASIC SEARCH STRATEGIES:**

### **Problem Spaces:**

Problem Space refers to the entire range of components that exist in the process of finding a solution to a problem. This range starts with “defining the problem,” then proceeds to the intermediate stage of “identifying and testing possible solutions” and ends with the final stage of “choosing and implementing a solution”. Plus, it includes all of the smaller steps that exist between these identified stages.

A simple example of this might be realizing that you don't have the right clothes for a social event, identifying what you need and where to go to buy the appropriate clothes and then buying those clothes and bringing them home. In between these stages you also have to take into consideration other associated concerns like "what can I afford? What stores carry what I want to buy?" and then "how do I find the time to shop for what I'm looking for?"

## TYPES OF SEARCH ALGORITHMS



## **I. Brute-Force Search Strategies (Uninformed /Blind)**

- Breadth-First Search
- Depth-First Search
- Bidirectional Search
- Uniform Cost Search
- Iterative Deepening Depth-First Search

## **II. Informed (Heuristic) Search Strategies**

- Pure Heuristic Search (Open and Closed List)
- Greedy Best First Search
- A \* Search

## **III. Local Search Algorithms**

- Hill-Climbing Search
- Local Beam Search
- Simulated Annealing
- Travelling Salesman Problem

### Uninformed Search Algorithms

Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

Following are the various types of uninformed search algorithms:

1. Breadth-first Search
2. Depth-first Search
3. Depth-limited Search
4. Iterative deepening depth-first search
5. Uniform cost search
6. Bidirectional Search.

## Breadth-first Search:

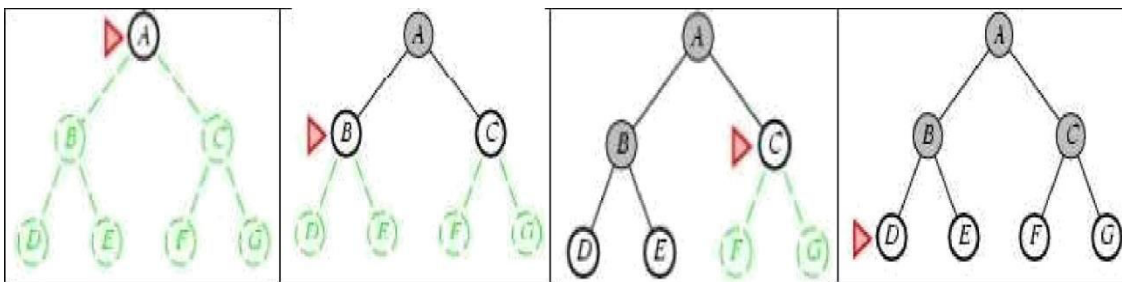
- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor nodes at the current level before moving to nodes of next level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure.

### Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

### Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.



**Time Complexity:** Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the  $d$  = depth of shallowest solution and  $b$  is a node at every state.

$$T(b) = 1 + b^2 + b^3 + \dots + b^d = O(b^d)$$

**Space Complexity:** Space complexity of BFS algorithm is given by the Memory size of frontier which is  $O(b^d)$ .

**Completeness:** BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

**Optimality:** BFS is optimal if path cost is a non-decreasing function of the depth of the node.

### **Depth-first Search:**

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

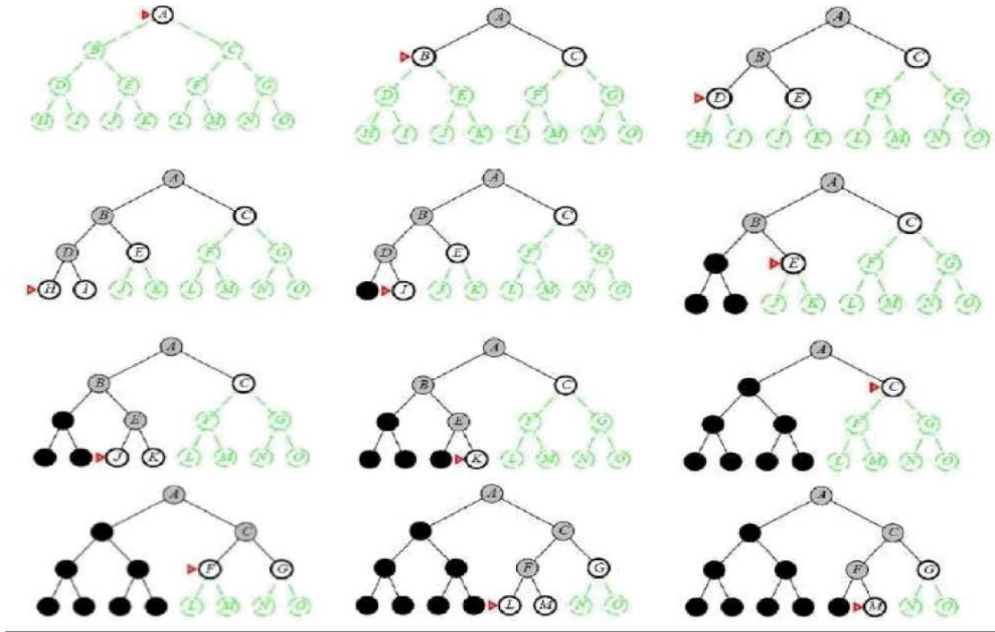
*Note: Backtracking is an algorithm technique for finding all possible solutions using recursion.*

#### **Advantage:**

- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

#### **Disadvantage:**

- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.



**Completeness:** DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

**Time Complexity:** Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$$T(n) = 1 + n^2 + n^3 + \dots + n^m = O(n^m)$$

Where,  $m$  = maximum depth of any node and this can be much larger than  $d$  (Shallowest solution depth)

**Space Complexity:** DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is  $O(bm)$ .

**Optimal:** DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

### Iterative deepening depth-first Search:

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.



This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

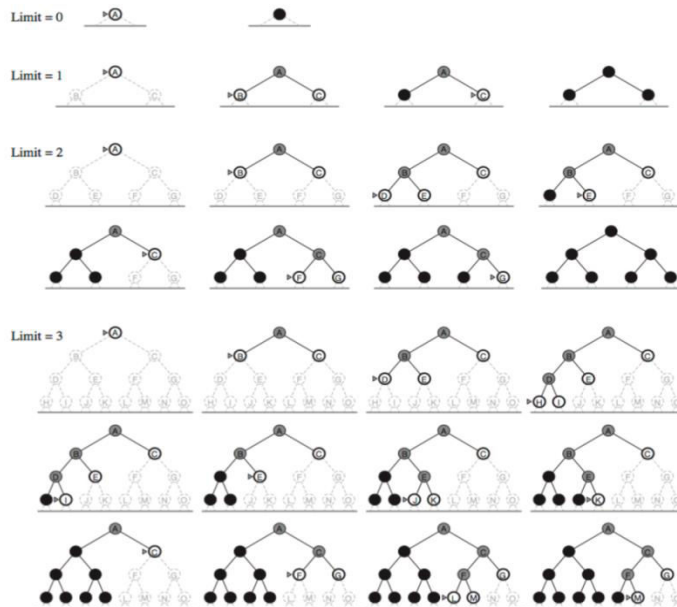
**Advantages:**

- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

**Disadvantages:**

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

# Iterative-Deepening Search



- **Completeness:**
- This algorithm is complete if the branching factor is finite.
- **Time Complexity:**

- Let's suppose  $b$  is the branching factor and depth is  $d$  then the worst-case time complexity is  $O(b^d)$ .
- **Space Complexity:**
- The space complexity of IDDFS will be  $O(bd)$ .
- **Optimal:**
- IDDFS algorithm is optimal if path cost is a non-decreasing function of the depth of the node.

## HILL CLIMBLING:

Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.

- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

### **Features of Hill Climbing:**

Following are some main features of Hill Climbing Algorithm:

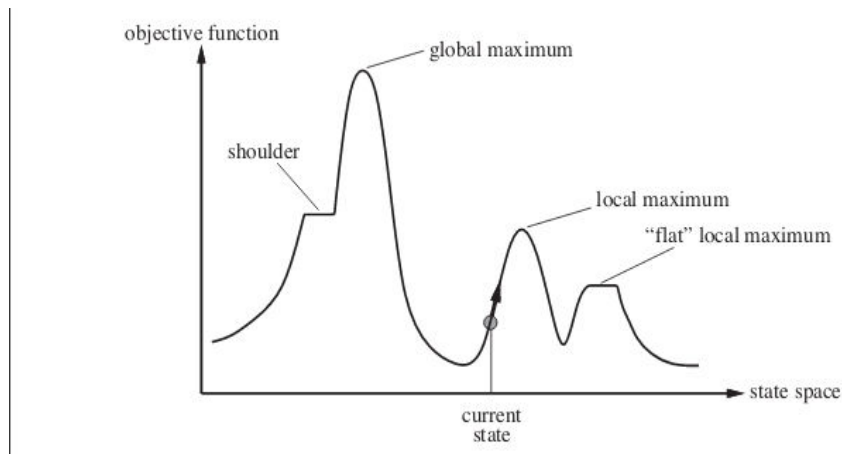
- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.

- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

### State-space Diagram for Hill Climbing:

The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.

On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis. If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum. If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.



### Different regions in the state space landscape:

**Local Maximum:** Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.

**Global Maximum:** Global maximum is the best possible state of state space landscape. It has the highest value of objective function.

**Current state:** It is a state in a landscape diagram where an agent is currently present.

**Flat local maximum:** It is a flat space in the landscape where all the neighbor states of current states have the same value.

**Shoulder:** It is a plateau region which has an uphill edge.

## Types of Hill Climbing Algorithm

- Simple hill Climbing:
- Steepest-Ascent hill-climbing:
- Stochastic hill Climbing:

### 1. Simple Hill Climbing:

Simple hill climbing is the simplest way to implement a hill climbing algorithm. **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state.** It only checks its one successor state, and if it finds better than the current state, then move else be in the same state. This algorithm has the following features:

- Less time consuming
- Less optimal solution and the solution is not guaranteed

Algorithm for Simple Hill Climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
  - a. If it is goal state, then return success and quit.
  - b. Else if it is better than the current state then assign new state as a current state.
  - c. Else if not better than the current state, then return to step2.

**Step 5:** Exit.

## 2. Steepest-Ascent hill climbing:

The steepest-Ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
  - a. Let SUCC be a state such that any successor of the current state will be better than it.
  - b. For each operator that applies to the current state:
    - a. Apply the new operator and generate a new state.
    - b. Evaluate the new state.
    - c. If it is goal state, then return it and quit, else compare it to the SUCC.
    - d. If it is better than SUCC, then set new state as SUCC.
    - e. If the SUCC is better than the current state, then set current state to SUCC.

**Step 3:** Exit.

## 3. Stochastic hill climbing:

Stochastic hill climbing does not examine for all its neighbor before moving. Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.

## **Problems in Hill Climbing Algorithm:**

**1. Local Maximum:** A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

**Solution:** Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.

**2. Plateau:** A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

**Solution:** The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.

**3. Ridges:** A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.

**Solution:** With the use of bidirectional search, or by moving in different directions, we can improve this problem.

## **Simulated Annealing:**

A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum. And if algorithm applies a random walk, by moving a successor, then it may complete but not efficient. **Simulated Annealing** is an algorithm which yields both efficiency and completeness.

In mechanical term **Annealing** is a process of hardening a metal or glass to a high temperature then cooling gradually, so this allows the metal to reach a low-energy crystalline state. The same process is used in simulated annealing in which the algorithm picks a random move, instead of picking the best move. If the random move improves the state, then it follows the same path. Otherwise, the algorithm follows the path which has a probability of less than 1 or it moves downhill and chooses another path.

## **INFORMED SEARCH ALGORITHM (HEURISTIC SEARCH):**

So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space. But informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.

The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

**Heuristics function:** Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by  $h(n)$ , and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

**Admissibility of the heuristic function is given as:**

$$h(n) \leq h^*(n)$$

**Here  $h(n)$  is heuristic cost, and  $h^*(n)$  is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.**

**Pure Heuristic Search:**

Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value  $h(n)$ . It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.

On each iteration, each node  $n$  with the lowest heuristic value is expanded and generates all its successors and  $n$  is placed to the closed list. The algorithm continues until a goal state is found.

In the informed search we will discuss two main algorithms which are given below:

- **Best First Search Algorithm(Greedy search)**
- **A\* Search Algorithm**

### **Best-first Search Algorithm (Greedy Search):**

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

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

Where,  $h(n)$  = estimated cost from node  $n$  to the goal.

The greedy best first algorithm is implemented by the priority queue.

#### **Best first search algorithm:**

- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node  $n$ , from the OPEN list which has the lowest value of  $h(n)$ , and places it in the CLOSED list.
- **Step 4:** Expand the node  $n$ , and generate the successors of node  $n$ .
- **Step 5:** Check each successor of node  $n$ , and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function  $f(n)$ , and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2

#### **Advantages:**

- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.

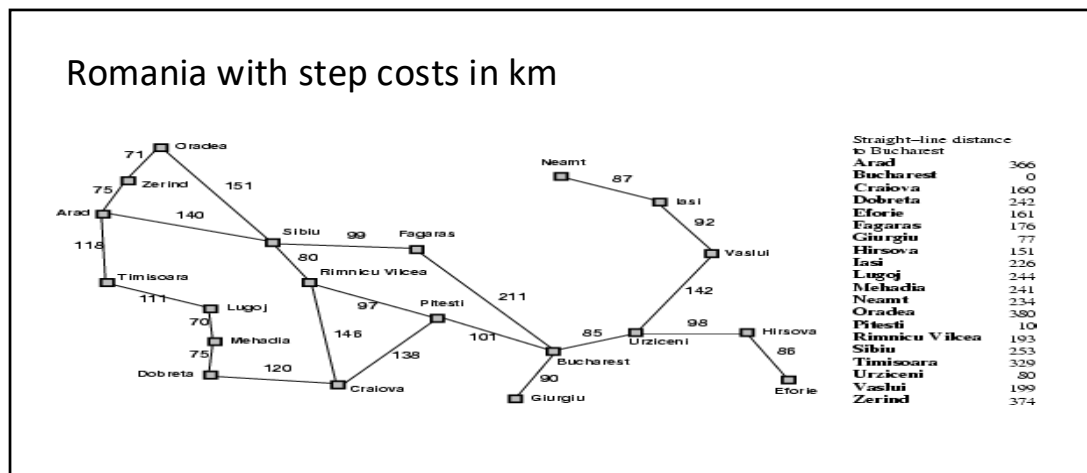


- This algorithm is more efficient than BFS and DFS algorithms.

### Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

### Example:



## Greedy best-first search example: Step:1



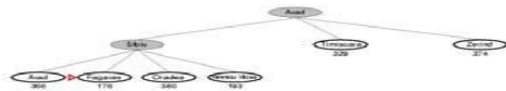
- Initial State = Arad
- Goal State = Bucharest

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

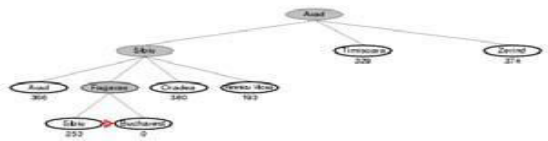
## Greedy: Step-2



## Greedy : Step-3



### Greedy : Step -4



### A\* Search Algorithm:

A\* search is the most commonly known form of best-first search. It uses heuristic function  $h(n)$ , and cost to reach the node  $n$  from the start state  $g(n)$ . It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A\* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A\* algorithm is similar to UCS except that it uses  $g(n)+h(n)$  instead of  $g(n)$ .

In A\* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.

- Evaluation function  $f(n) = g(n) + h(n)$
- $g(n)$  = cost so far to reach  $n$
- $h(n)$  = estimated cost from  $n$  to goal
- $f(n)$  = estimated total cost of path through  $n$  to goal

### **Algorithm of A\* search:**

**Step1:** Place the starting node in the OPEN list.

**Step 2:** Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

**Step 3:** Select the node from the OPEN list which has the smallest value of evaluation function ( $g+h$ ), if node  $n$  is goal node then return success and stop, otherwise

**Step 4:** Expand node  $n$  and generate all of its successors, and put  $n$  into the closed list. For each successor  $n'$ , check whether  $n'$  is already in the OPEN or CLOSED list, if not then compute evaluation function for  $n'$  and place into Open list.

**Step 5:** Else if node  $n'$  is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest  $g(n')$  value.

**Step 6:** Return to **Step 2**.

### Advantages:

- A\* search algorithm is the best algorithm than other search algorithms.
- A\* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.


### Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A\* search algorithm has some complexity issues.
- The main drawback of A\* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.


**A\* search Eg : Step1 1 1**

---

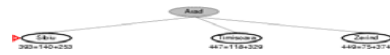
Frontier queue:

A diagram of an A\* search node. It consists of a red triangle pointing to the right, followed by the text 'A\*id' and '368+0+368' below it.

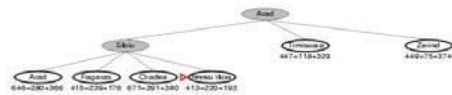
CS 391 - Intro to AI

25 

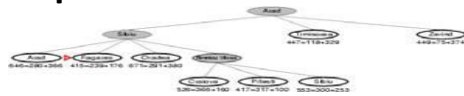
## Step 2



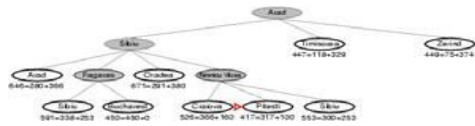
## Step 3



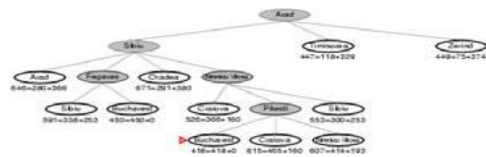
## Step 4



## Step 5



## Step 6



## Constraint satisfaction problems (CSPs):

CSP:

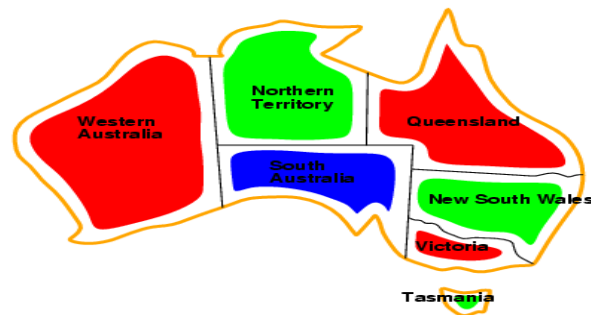
- state is defined by variables  $X_i$  with values from domain  $D_i$
- goal test is a set of constraints specifying allowable combinations of values for subsets of variables

Allows useful general-purpose algorithms with more power than standard search algorithms

## Example: Map-Coloring



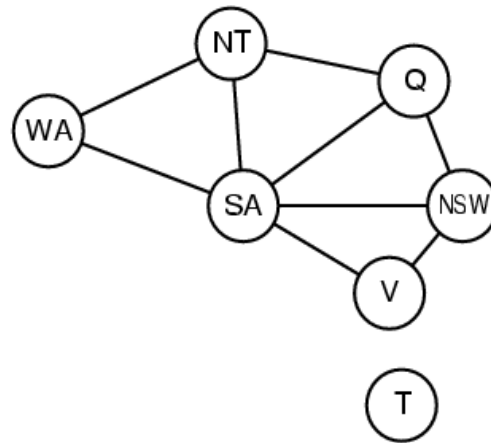
- Variables  $WA, NT, Q, NSW, V, SA, T$
- Domains  $D_i = \{\text{red, green, blue}\}$
- Constraints: adjacent regions must have different colors
- e.g.,  $WA \neq NT$



Solutions are complete and consistent assignments, e.g.,  $WA = \text{red}, NT = \text{green}, Q = \text{red}, NSW = \text{green}, V = \text{red}, SA = \text{blue}, T = \text{green}$

## Constraint graph

- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables, arcs are constraints



## Varieties of CSPs

### ■ Discrete variables

#### ■ finite domains:

- $n$  variables, domain size  $d \rightarrow O(d^n)$  complete assignments
- e.g., 3-SAT (NP-complete)

#### ■ infinite domains:

- integers, strings, etc.
- e.g., job scheduling, variables are start/end days for each job
- need a constraint language, e.g.,  $StartJob_1 + 5 \leq StartJob_3$
- Continuous variables
- e.g., start/end times for Hubble Space Telescope observations
- linear constraints solvable in polynomial time by linear programming



## Real-world CSPs

- **Assignment problems**
  - e.g., who teaches what class
- **Timetabling problems**
  - e.g., which class is offered when and where?
- **Transportation scheduling**
- **Factory scheduling**
- Notice **that many real-world problems involve real-valued variables**

## Types of CSP's:

- Backtracking
- Local search

## Backtracking (Depth-First) search:

Backtracking search, a form of depth-first search, is commonly used for solving CSPs. Inference can be interwoven with search.

**Commutativity:** CSPs are all commutative. A problem is commutative if the order of application of any given set of actions has no effect on the outcome.

**Backtracking search:** A depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.

Backtracking algorithm repeatedly chooses an unassigned variable, and then tries all values in the domain of that variable in turn, trying to find a solution. If an inconsistency is detected, then BACKTRACK returns failure, causing the previous call to try another value.

There is no need to supply BACKTRACKING-SEARCH with a domain-specific initial state, action function, transition model, or goal test.

BACKTRACKING-SEARCH keeps only a single representation of a state and alters that representation rather than creating a new ones.

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
if assignment is complete then return assignment
var ← SELECT-UNASSIGNED-VARIABLE(csp)
for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
  if value is consistent with assignment then
    add { var = value } to assignment
    inferences ← INFERENCE(csp, var, value)
    if inferences ≠ failure then
      add inferences to assignment
      result ← BACKTRACK(assignment, csp)
      if result ≠ failure then
        return result
    remove { var = value } and inferences from assignment
return failure
```

**Figure 6.5** A simple backtracking algorithm for constraint satisfaction problems. The algorithm is modeled on the recursive depth-first search of Chapter 3. By varying the functions SELECT-UNASSIGNED-VARIABLE and ORDER-DOMAIN-VALUES, we can implement the general-purpose heuristics discussed in the text. The function INFERENCE can optionally be used to impose arc-, path-, or  $k$ -consistency, as desired. If a value choice leads to failure (noticed either by INFERENCE or by BACKTRACK), then value assignments (including those made by INFERENCE) are removed from the current assignment and a new value is tried.

To solve CSPs efficiently without domain-specific knowledge, address following questions:

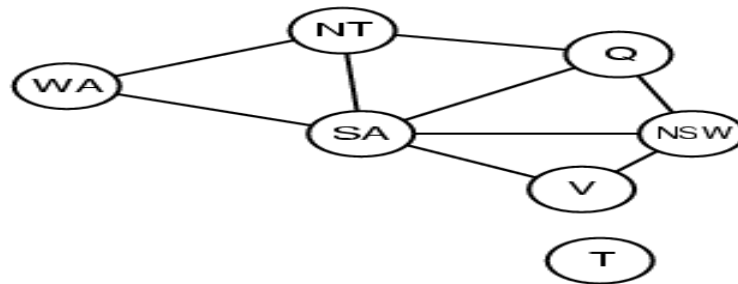
1)function **SELECT-UNASSIGNED-VARIABLE**: which variable should be assigned next?

function **ORDER-DOMAIN-VALUES**: in what order should its values be tried?

2)function **INFERENCE**: what inferences should be performed at each step in the search?

3)When the search arrives at an assignment that violates a constraint, can the search avoid repeating this failure?

## 1. Variable and value ordering



### SELECT-UNASSIGNED-VARIABLE

Variable selection—fail-first

**Minimum-remaining-values (MRV) heuristic:** The idea of choosing the variable with the fewest “legal” value. A.k.a. “most constrained variable” or “fail-first” heuristic, it picks a variable that is most likely to cause a failure soon thereby pruning the search tree. If some variable X has no legal values left, the MRV heuristic will select X and failure will be detected immediately—avoiding pointless searches through other variables.

E.g. After the assignment for WA=red and NT=green, there is only one possible value for SA, so it makes sense to assign SA=blue next rather than assigning Q.

[Powerful guide]

**Degree heuristic:** The degree heuristic attempts to reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints on other unassigned variables. [useful tie-breaker]

e.g. SA is the variable with highest degree 5; the other variables have degree 2 or 3; T has degree 0.

### ORDER-DOMAIN-VALUES

Value selection—fail-last

If we are trying to find all the solution to a problem (not just the first one), then the ordering does not matter.

**Least-constraining-value heuristic:** prefers the value that rules out the fewest choice for the neighboring variables in the constraint graph. (**Try to leave the maximum flexibility for subsequent variable assignments.**)

e.g. We have generated the partial assignment with WA=red and NT=green and that our next choice is for Q. Blue would be a bad choice because it eliminates the last legal value left for Q's neighbor, SA, therefore prefers red to blue.

The **minimum-remaining-values** and **degree** heuristic are domain-independent methods for deciding which variable to choose next in a backtracking search. The **least-constraining-value** heuristic helps in deciding which value to try first for a given variable.

## 2. Interleaving search and inference

### INFERENCE

**forward checking:** [One of the simplest forms of inference.] Whenever a variable X is assigned, the forward-checking process establishes arc consistency for it: for each unassigned variable Y that is connected to X by a constraint, delete from Y's domain any value that is inconsistent with the value chosen for X.

There is no reason to do forward checking if we have already done arc consistency as a preprocessing step.

	WA	NT	Q	NSW	V	SA	T
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After $WA=red$	(R)	G B	R G B	R G B	R G B	G B	R G B
After $Q=green$	(R)	B	(G)	R B	R G B	B	R G B
After $V=blue$	(R)	B	(G)	R	(B)		R G B

**Figure 6.7** The progress of a map-coloring search with forward checking.  $WA = red$  is assigned first; then forward checking deletes  $red$  from the domains of the neighboring variables  $NT$  and  $SA$ . After  $Q = green$  is assigned,  $green$  is deleted from the domains of  $NT$ ,  $SA$ , and  $NSW$ . After  $V = blue$  is assigned,  $blue$  is deleted from the domains of  $NSW$  and  $SA$ , leaving  $SA$  with no legal values.

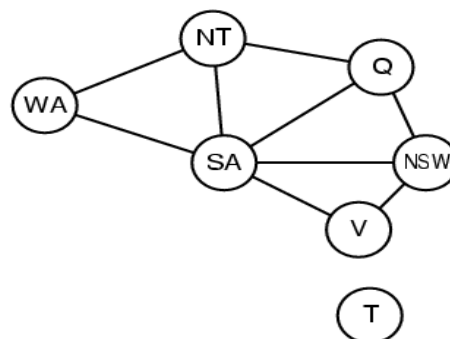
### Advantage:

For many problems the search will be more effective if we combine the MRV heuristic with forward checking.

Disadvantage: Forward checking only makes the current variable arc-consistent, but doesn't look ahead and make all the other variables arc-consistent.

**MAC (Maintaining Arc Consistency) algorithm:** [More powerful than forward checking, detect this inconsistency.] After a variable  $X_i$  is assigned a value, the INFERENCE procedure calls AC-3, but instead of a queue of all arcs in the CSP, we start with only the arcs( $X_j, X_i$ ) for all  $X_j$  that are unassigned variables that are neighbors of  $X_i$ . From there, AC-3 does constraint propagation in the usual way, and if any variable has its domain reduced to the empty set, the call to AC-3 fails and we know to backtrack immediately.

### 3. Intelligent backtracking



chronological backtracking: The BACKGRACKING-SEARCH in Fig 6.5. When a branch of the search fails, back up to the preceding variable and try a different value for it. (The most recent decision point is revisited.)

e.g.

Suppose we have generated the partial assignment { Q=red, NSW=green, V=blue, T=red }.

When we try the next variable SA, we see every value violates a constraint.

We back up to T and try a new color, it cannot resolve the problem.

**Intelligent backtracking:** Backtrack to a variable that was responsible for making one of the possible values of the next variable (e.g. SA) impossible.

**Conflict set for a variable:** A set of assignments that are in conflict with some value for that variable.

(e.g. The set { Q=red, NSW=green, V=blue } is the conflict set for SA.)

**backjumping method:** Backtracks to the most recent assignment in the conflict set.

(e.g. backjumping would jump over T and try a new value for V.)

Forward checking can supply the conflict set with no extra work.

Whenever forward checking based on an assignment  $X=x$  deletes a value from Y's domain, add  $X=x$  to Y's conflict set;

If the last value is deleted from Y's domain, the assignment in the conflict set of Y are added to the conflict set of X.

In fact, every branch pruned by backjumping is also pruned by forward checking. Hence simple backjumping is redundant in a forward-checking search or in a search that uses stronger consistency checking (such as MAC).

## Conflict-directed backjumping:

e.g.

consider the partial assignment which is proved to be inconsistent: {WA=red, NSW=red}.

We try T=red next and then assign NT, Q, V, SA, no assignment can work for these last 4 variables.

Eventually we run out of value to try at NT, but simple backjumping cannot work because NT doesn't have a complete conflict set of preceding variables that caused to fail.

The set {WA, NSW} is a deeper notion of the conflict set for NT, caused NT together with any subsequent variables to have no consistent solution. So the algorithm should backtrack to NSW and skip over T.

A backjumping algorithm that uses conflict sets defined in this way is called conflict-direct backjumping.

### How to Compute:

When a variable's domain becomes empty, the "terminal" failure occurs, that variable has a standard conflict set.

Let  $X_j$  be the current variable, let  $conf(X_j)$  be its conflict set. If every possible value for  $X_j$  fails, backjump to the most recent variable  $X_i$  in  $conf(X_j)$ , and set

$$conf(X_i) \leftarrow conf(X_i) \cup conf(X_j) - \{X_j\}.$$

The conflict set for an variable means, there is no solution from that variable onward, given the preceding assignment to the conflict set.

e.g.

assign WA, NSW, T, NT, Q, V, SA.

SA fails, and its conflict set is {WA, NT, Q}. (standard conflict set)

Backjump to Q, its conflict set is  $\{NT, NSW\} \cup \{WA, NT, Q\} - \{Q\} = \{WA, NT, NSW\}$ .

Backtrack to NT, its conflict set is  $\{WA\} \cup \{WA, NT, NSW\} - \{NT\} = \{WA, NSW\}$ .

Hence the algorithm backjump to NSW. (over T)

After backjumping from a contradiction, how to avoid running into the same problem again:

**Constraint learning:** The idea of finding a minimum set of variables from the conflict set that causes the problem. This set of variables, along with their corresponding values, is called a **no-good**. We then record the no-good, either by adding a new constraint to the CSP or by keeping a separate cache of no-goods.

Backtracking occurs when no legal assignment can be found for a variable. **Conflict-directed backjumping** backtracks directly to the source of the problem.

### Local search for CSPs

Local search algorithms for CSPs use a complete-state formulation: the initial state assigns a value to every variable, and the search change the value of one variable at a time.

**The min-conflicts heuristic:** In choosing a new value for a variable, select the value that results in the minimum number of conflicts with other variables.

The landscape of a CSP under the mini-conflicts heuristic usually has a series of plateau. Simulated annealing and Plateau search (i.e. allowing sideways moves to another state with the same score) can help local search find its way off the plateau. This wandering on the plateau can be directed with **tabu search**: keeping a small list of recently visited states and forbidding the algorithm to return to those tates.

**Constraint weighting:** a technique that can help concentrate the search on the important constraints.



Each constraint is given a numeric weight  $W_i$ , initially all 1.

At each step, the algorithm chooses a variable/value pair to change that will result in the lowest total weight of all violated constraints.

The weights are then adjusted by incrementing the weight of each constraint that is violated by the current assignment.

Local search can be used in an online setting when the problem changes, this is particularly important in scheduling problems.

## **The structure of problem:**

### **1. The structure of constraint graph**

The structure of the problem as represented by the constraint graph can be used to find solution quickly.

e.g. The problem can be decomposed into 2 **independent subproblems**: Coloring T and coloring the mainland.

**Tree:** A constraint graph is a tree when any two variable are connected by only one path.

**Directed arc consistency (DAC):** A CSP is defined to be directed arc-consistent under an ordering of variables  $X_1, X_2, \dots, X_n$  if and only if every  $X_i$  is arc-consistent with each  $X_j$  for  $j > i$ .

By using DAC, any tree-structured CSP can be solved in time linear in the number of variables.

How to solve a tree-structure CSP:

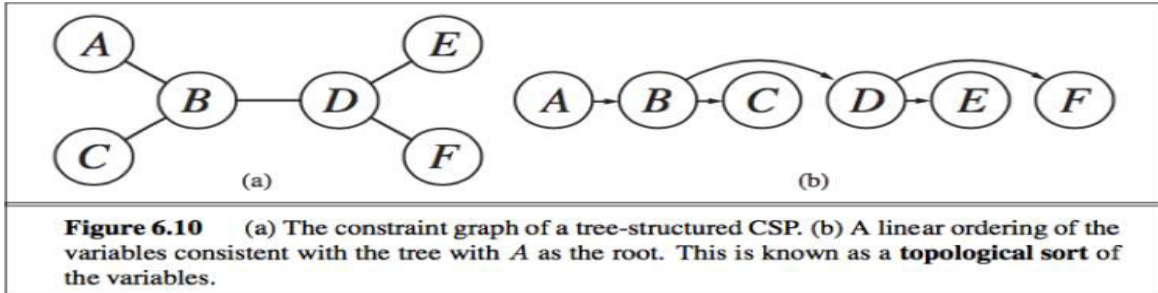
Pick any variable to be the root of the tree;

Choose an ordering of the variable such that each variable appears after its parent in the tree. (**topological sort**)

Any tree with  $n$  nodes has  $n-1$  arcs, so we can make this graph directed arc-consistent in  $O(n)$  steps, each of which must compare up to  $d$  possible domain values for 2 variables, for a total time of  $O(nd^2)$ .

Once we have a directed arc-consistent graph, we can just march down the list of variables and choose any remaining value.

Since each link from a parent to its child is arc consistent, we won't have to backtrack, and can move linearly through the variables.



There are 2 primary ways to reduce more general constraint graphs to trees:

1. Based on removing nodes;

e.g. We can delete *SA* from the graph by fixing a value for *SA* and deleting from the domains of other variables any values that are inconsistent with the value chosen for *SA*.

The general algorithm:

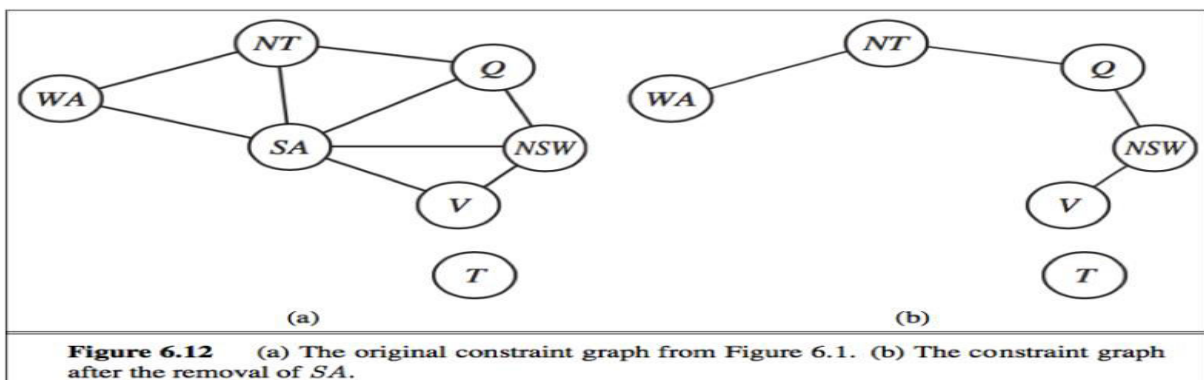
Choose a subset *S* of the CSP's variables such that the constraint graph becomes a tree after removal of *S*. *S* is called a **cycle cutset**.

For each possible assignment to the variables in *S* that satisfies all constraints on *S*,

(a) remove from the domain of the remaining variables any values that are inconsistent with the assignment for *S*, and

(b) If the remaining CSP has a solution, return it together with the assignment for *S*.

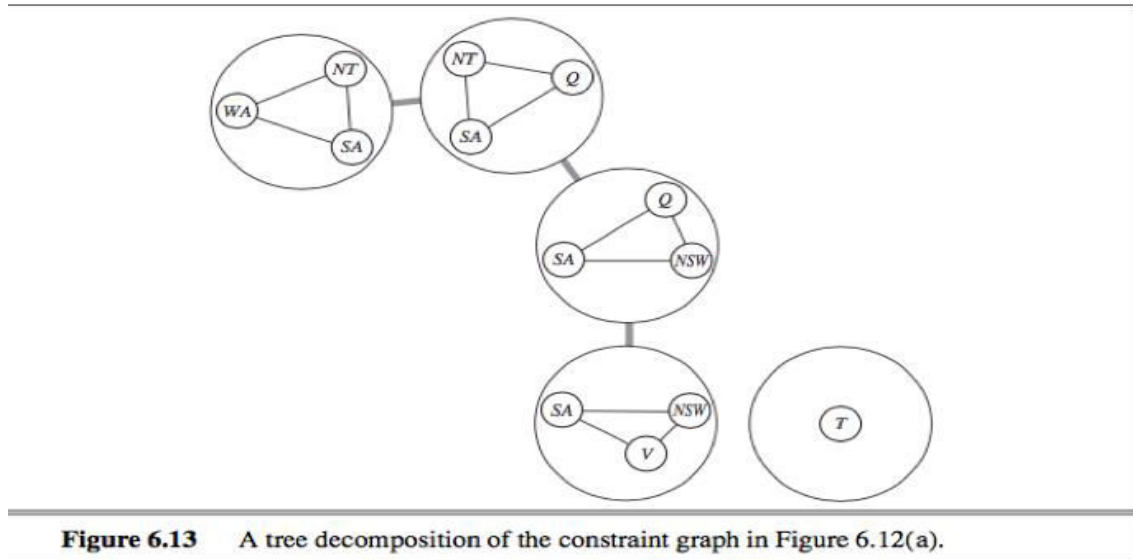
Time complexity:  $O(d^c \cdot (n-c)d^2)$ , *c* is the size of the cycle cut set.



**Cutset conditioning:** The overall algorithmic approach of efficient approximation algorithms to find the smallest cycle cutset.

## 2. Based on collapsing nodes together

**Tree decomposition:** construct a **tree decomposition** of the constraint graph into a set of connected subproblems, each subproblem is solved independently, and the resulting solutions are then combined.



A tree decomposition must satisfy 3 requirements:

- Every variable in the original problem appears in at least one of the subproblems.
- If 2 variables are connected by a constraint in the original problem, they must appear together (along with the constraint) in at least one of the subproblems.
- If a variable appears in 2 subproblems in the tree, it must appear in every subproblem along the path connecting those those subproblems.

We solve each subproblem independently.

If any one has no solution, the entire problem has no solution.

If we can solve all the subproblems, then construct a global solution as follows:

First, view each subproblem as a “mega-variable” whose domain is the set of all solutions for the subproblem.

Then, solve the constraints connecting the subproblems using the efficient algorithm for trees.

A given constraint graph admits many tree decomposition;

In choosing a decomposition, the aim is to make the subproblems as small as possible.

**Tree width:**

The tree width of a tree decomposition of a graph is one less than the size of the largest sub problems.

The tree width of the graph itself is the minimum tree width among all its tree decompositions.

Time complexity:  $O(nd^{w+1})$ ,  $w$  is the tree width of the graph.

The complexity of solving a CSP is strongly related to the structure of its constraint graph. Tree-structured problems can be solved in linear time. **Cutset conditioning** can reduce a general CSP to a tree-structured one and is quite efficient if a small cutset can be found. **Tree decomposition** techniques transform the CSP into a tree of sub problems and are efficient if the **tree width** of constraint graph is small.

**2. The structure in the values of variables**

By introducing a **symmetry-breaking constraint**, we can break the **value symmetry** and reduce the search space by a factor of  $n!$ .

Eg.

Consider the map-coloring problems with  $n$  colors, for every consistent solution, there is actually a set of  $n!$  solutions formed by permuting the color names.(value symmetry)

On the Australia map, WA, NT and SA must all have different colors, so there are  $3!=6$  ways to assign.

We can impose an arbitrary ordering constraint  $NT < SA < WA$  that requires the 3 values to be in alphabetical order. This constraint ensures that only one of the  $n!$  Solution is possible: {NT=blue, SA=green, WA=red}. (Symmetry-breaking constraint)

# MODULE - 2

## ADVANCED SEARCH

### Constructing Search Trees

#### Tree (data structure)

A tree data structure can be defined recursively as a collection of nodes (starting at a root node), where each node is a data structure consisting of a value, together with a list of references to nodes (the "children"), with the constraints that no reference is duplicated, and none points to the root.

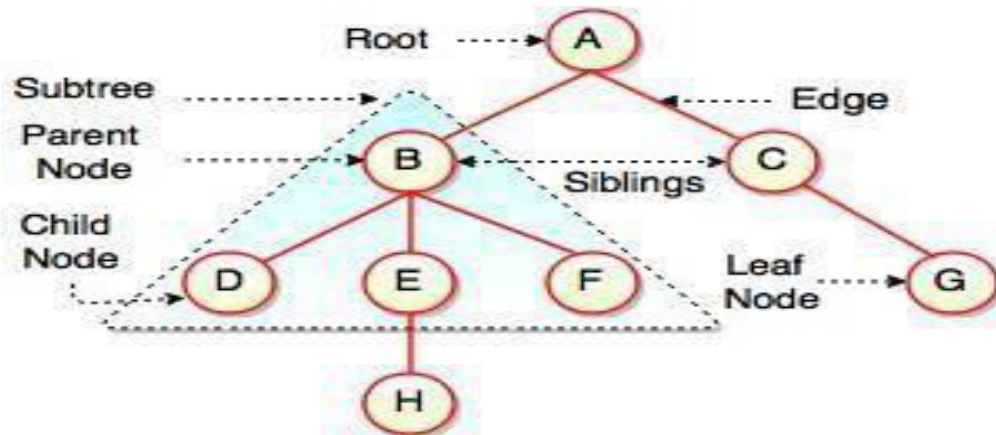


Fig. Structure of Tree

## **Terminology used in trees**

### **Node**

A node is a structure which may contain a value or condition, or represent a separate data structure.

### **Root**

The top node in a tree, the prime ancestor.

### **Child**

A node directly connected to another node when moving away from the root, an immediate descendant.

### **Parent**

The converse notion of a child, an immediate ancestor.

### **Siblings**

A group of nodes with the same parent.

### **Neighbor**

Parent or child.

### **Descendant**

A node reachable by repeated proceeding from parent to child. Also known

as subchild.

### **Ancestor**

A node reachable by repeated proceeding from child to parent.

### **Leaf / External node (not common)**

A node with no children.

### **Branch node / Internal node**

A node with at least one child.

### **Degree**

For a given node, its number of children. A leaf is necessarily degree zero. The degree of a tree is the degree of its root.

### **Degree of tree**

The degree of the root.

### **Edge**

The connection between one node and another.

### **Path**

A sequence of nodes and edges connecting a node with a descendant.

### **Distance**

The number of edges along the shortest path between two nodes.

## **Depth**

The distance between a node and the root. Level 1 + the number of edges between a node and the root, i.e. (Depth + 1)

## **Height**

The number of edges on the longest path between a node and a descendant leaf.

## **Width**

The number of nodes in a level.

## **Breadth**

The number of leaves.

## **Height of tree**

The height of the root node or the maximum level of any node in the tree.

## **Forest**

A set of  $n \geq 0$  disjoint trees.

## **Sub Tree**

A tree T is a tree consisting of a node in T and all of its descendants in T.

## **Ordered Tree**

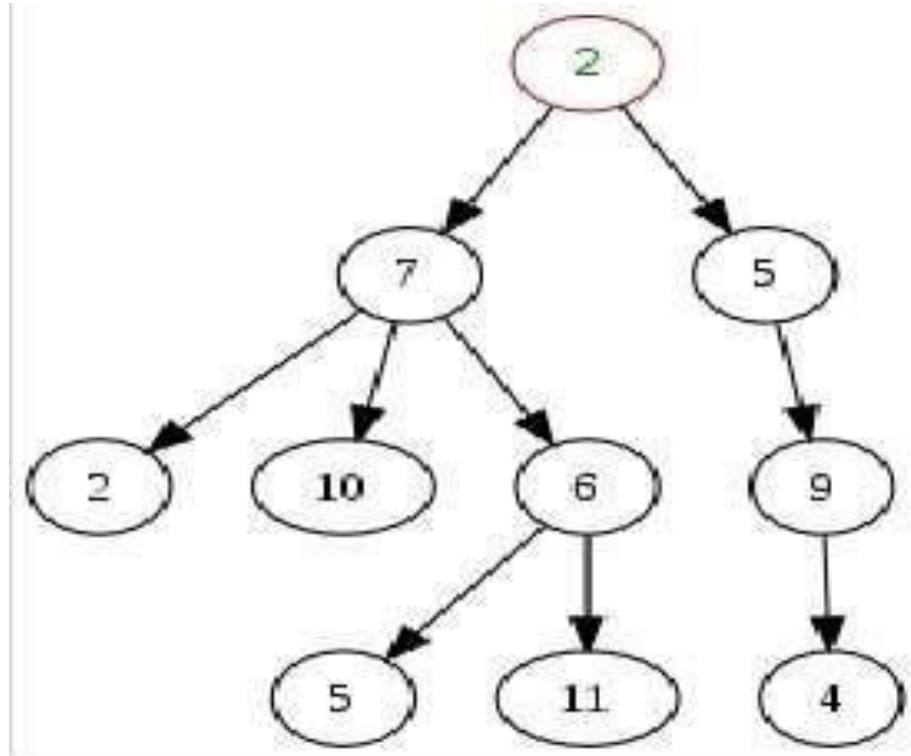
A rooted tree in which an ordering is specified for the children of each vertex.



## Size of a tree

Number of nodes in the tree.

## Example: Tree



## **Stochastic Search :**

- We have to analyze bio-inspired computational methods in a rigorous way with respect to their runtime behavior.
- As these algorithms make use of many random decisions, we treat them as randomized algorithms to study their behavior in a rigorous manner.
- The term stochastic search algorithms stresses this point of view and be used in the following to point that bio-inspired computational methods can be treated as algorithms which are based on random decisions.
- These algorithms are inspired by the evaluation process in nature and follows Darwin's principle of the survival of fittest.
- A Stochastic search algorithm is a **problem-independent algorithm** to solve problems from a considered search space , although it might have modules that are adjusted to the considered problem are combined with problem – dependent algorithms.
- Stochastic search algorithm is a general purpose algorithm .

### **Algorithms Used:**

- 1. Problem Dependent Algorithm:** Classical algorithm.
- 2. Problem Independent Algorithm:** To prove bounds on the runtime and/or approximation quality in mind.

### **Advantages:**

- Easy to adopt for different types of problems.

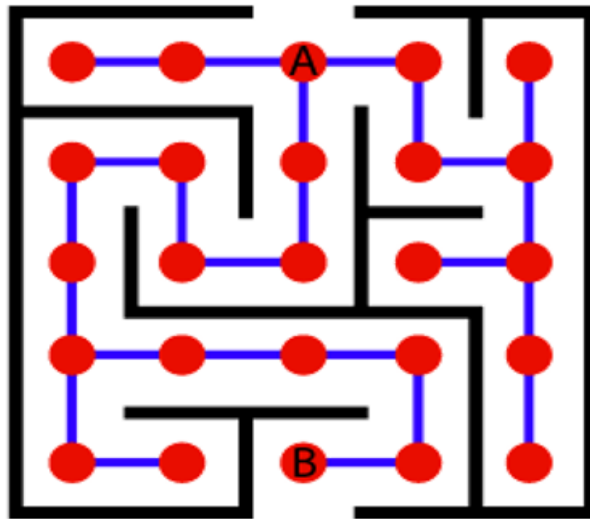
### **Disadvantages:**

- This algorithm is often not rigorously analyzed with respect to its runtime and/or approximation quality.

## **A\* Search Implementation:**

A\* Search Algorithm is one such algorithm that has been developed to help us. A\* was initially designed as a general graph traversal algorithm. It is widely used in solving path finding problems in video games. Because of its flexibility and versatility, it can be used in a wide range of contexts. A\* is formulated with weighted graphs, which means it can find the best path involving the smallest cost in terms of distance and time. This makes A\* algorithm in artificial intelligence an informed search algorithm for [best-first search](#). Let us have a detailed look into the various aspects of A\*.

## What is A\* Algorithm



The most important advantage of A\* search algorithm which separates it from other traversal techniques is that it has a brain. This makes A\* very smart and pushes it much ahead of other conventional algorithms.

Consider the diagram below:

Let's try to understand Basic AI Concepts and to comprehend how does A\* algorithm work. Imagine a huge maze, one that is too big that it takes hours to reach the endpoint manually. Once you complete it on foot, you need to go for another one. Which implies that you would end up investing a lot of time and effort to find the possible paths in this maze. Now, you want to make it less time-consuming. To make it easier, we will consider this maze as a search problem and will try to apply it to other possible mazes we might encounter in the due course, provided they follow the same structure and rules.

As the first step to convert this maze into a search problem, we need to define these six things.

1. A set of prospective states we might be in
2. A beginning and end state
3. A way to decide if we've reached the endpoint
4. A set of actions in case of possible direction/path changes
5. A function that advises us about the result of an action
6. A set of costs incurring in different states/paths of movement

To solve the problem, we need to map the intersections to the nodes (denoted by the red dots) and all the possible ways we can make movements towards the edges (denoted by the blue lines). A denotes the starting point and B denotes the endpoint. We define the starting and endpoint at the nodes A and B respectively. If we use an uninformed search algorithm, it would be like finding a path that is blind, while an informed algorithm for a search problem would take the path that brings you closer to your destination. For instance, consider Rubik's cube; it has many prospective states that you can be in and this makes the solution very difficult. This calls for the use of a guided search algorithm to find a solution. This explains the importance of A\*.

Unlike other algorithms, A\* decides to take up a step only if it is convincingly sensible and reasonable as per its functions. This means, it never considers any non-optimal steps. This is why A\* is a popular choice for AI systems that replicate the real world – like video games and machine learning.

### A\* Algorithm Steps

- **Firstly, add the beginning node to the open list**
- **Then repeat the following step**

– In the open list, find the square with the lowest F cost – and this denotes the current square.

– Now we move to the closed square.

– Consider 8 squares adjacent to the current square and

Ignore it if it is on the closed list, or if it is not workable. Do the following if it is workable

Check if it is on the open list; if not, add it. You need to make the current square as this square's a parent. You will now record the different costs of the square like the F, G and H costs.

If it is on the open list, use G cost to measure the better path. Lower the G cost, the better the path. If this path is better, make the current square as the parent square. Now you need to recalculate the other scores – the G and F scores of this square.

You'll stop:

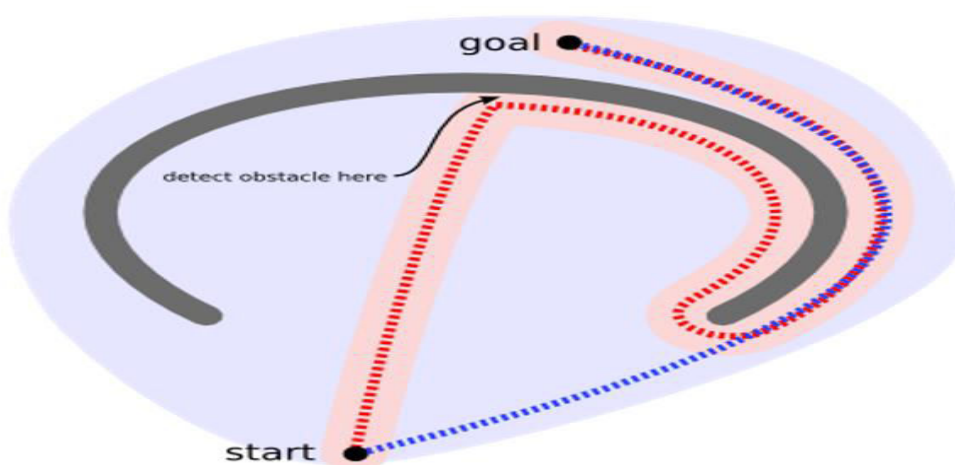
If you find the path, you need to check the closed list and add the target square to it.

There is no path if the open list is empty and you could not find the target square.

3. Now you can save the path and work backwards starting from the target square, going to the parent square from each square you go, till it takes you to the starting square. You've found your path now.

### Why is A\* Search Algorithm Preferred?

It's easy to give movement to objects. But pathfinding is not simple. It is a complex exercise. The following situation explains it.



The task is to take the unit you see at the bottom of the diagram, to the top of it. You can see that there is nothing to indicate that the object should not take the path denoted with pink lines. So it chooses to move that way. As and when it reaches the top it has to change its direction because of the 'U' shaped obstacle. Then it changes the direction, goes around the obstacle, to reach the top. In contrast to this, A\* would have scanned the area above the object and found a short path (denoted with blue lines). Thus, pathfinder algorithms like A\* help you plan things rather than waiting until you discover the problem. They act proactively rather than reacting to a situation. The disadvantage is that it is a bit slower than the other algorithms. You can use a combination of both to achieve better results – pathfinding algorithms give bigger picture and long paths with obstacles that change slowly; and movement algorithms for local picture and short paths with obstacles that change faster.

### **A\* Algorithm and Its Basic Concepts**

- A\* algorithm works based on heuristic methods and this helps achieve optimality.
- A\* is a different form of the best-first algorithm. Optimality empowers an algorithm to find the best possible solution to a problem. Such algorithms also offer completeness, if there is any solution possible to an existing problem, the algorithm will definitely find it.
- When A\* enters into a problem, firstly it calculates the cost to travel to the neighbouring nodes and chooses the node with the lowest cost. If The  $f(n)$  denotes the cost, A\* chooses the node with the lowest  $f(n)$  value. Here 'n' denotes the neighbouring nodes. The calculation of the value can be done as shown below:  

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

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

$$g(n) = \text{shows the shortest path's value from the starting node to node } n$$

$$h(n) = \text{The heuristic approximation of the value of the node}$$
The heuristic value has an important role in the efficiency of the A\* algorithm. To find the best solution, you might have to use different heuristic function according to the type of the problem.
- However, the creation of these functions is a difficult task, and this is the basic problem we face in AI.

### **// A\* Search Algorithm**

1. Initialize the open list
2. Initialize the closed list  
put the starting node on the open list (you can leave its f at zero)
3. while the open list is not empty
  - a) find the node with the least f on

the open list, call it "q"

b) pop q off the open list

c) generate q's 8 successors and set their parents to q

d) for each successor

i) if successor is the goal, stop search

successor.g = q.g + distance between  
successor and q

successor.h = distance from goal to  
successor (This can be done using many  
ways, we will discuss three heuristics-  
Manhattan, Diagonal and Euclidean  
Heuristics)

successor.f = successor.g + successor.h

ii) if a node with the same position as

successor is in the OPEN list which has a  
lower f than successor, skip this successor

iii) if a node with the same position as

successor is in the CLOSED list which has  
a lower f than successor, skip this successor  
otherwise, add the node to the open list

end (for loop)

e) push q on the closed list

end (while loop)

### **Mini-Max Algorithm:**

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe and various tow-players game. This Algorithm computes the minimax decision for the current state.

- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

### Pseudo-code for MinMax Algorithm:

function minimax(node, depth, maximizingPlayer) is

**if** depth == 0 or node is a terminal node then

**return static** evaluation of node

**if** MaximizingPlayer then     // for Maximizer Player

maxEva= -infinity

**for** each child of node **do**

eva= minimax(child, depth-1, **false**)

maxEva= max(maxEva,eva)     //gives Maximum of the values

**return** maxEva

**else**                     // for Minimizer player

minEva= +infinity

**for** each child of node **do**

eva= minimax(child, depth-1, **true**)

minEva= min(minEva, eva)     //gives minimum of the values

**return** minEva

**Initial call:**

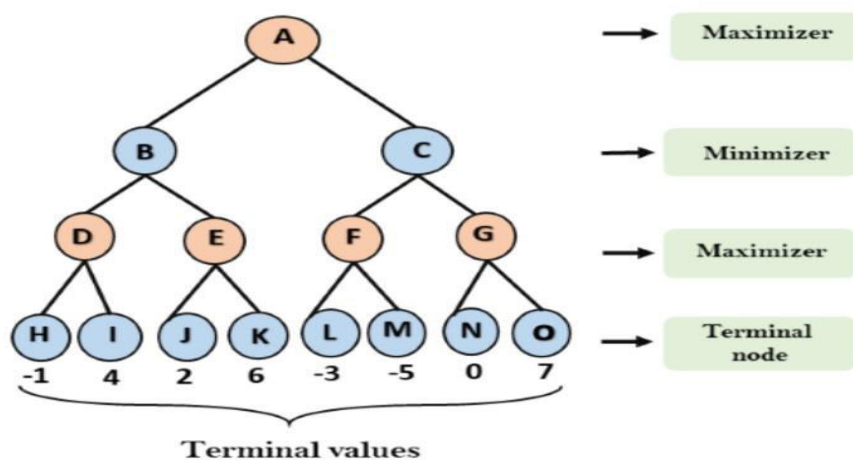
**Minimax(node, 3, true)**

### Working of Min-Max Algorithm:

- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.

- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

**Step-1:** In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



**Step 2:** Now, first we find the utilities value for the Maximizer, its initial value is  $-\infty$ , so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

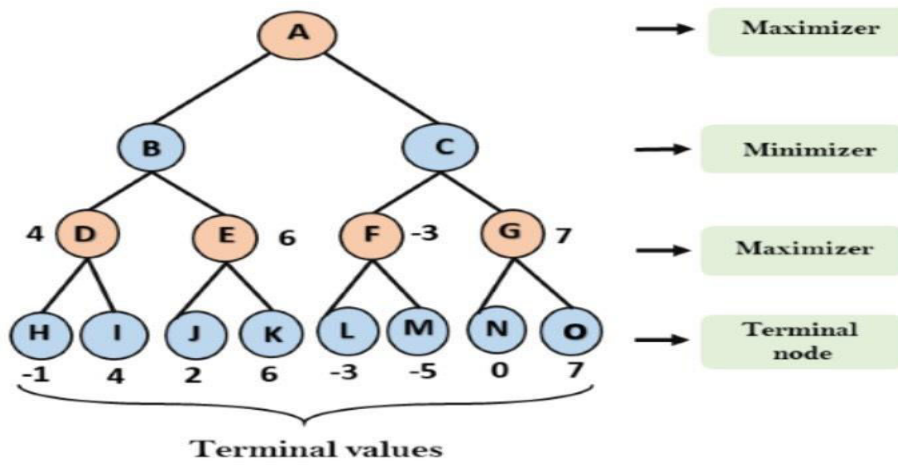
For node D  $\max(-1, -\infty) \Rightarrow \max(-1, 4) = 4$

For Node E  $\max(2, -\infty) \Rightarrow \max(2, 6) = 6$

For Node F  $\max(-3, -\infty) \Rightarrow \max(-3, -5) = -3$

For node G  $\max(0, -\infty) = \max(0, 7) = 7$

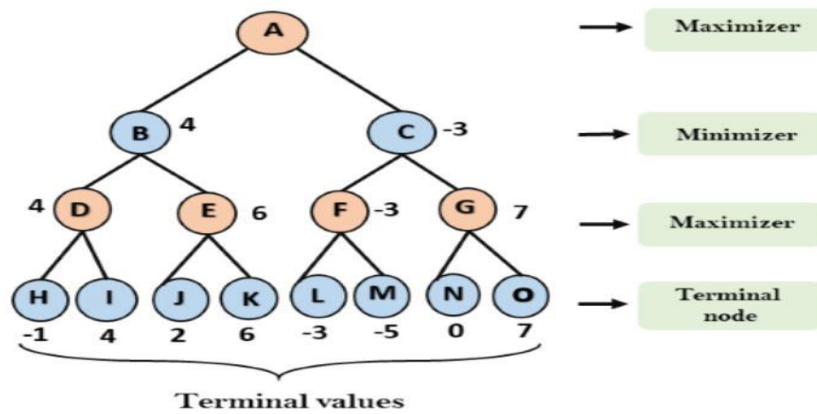




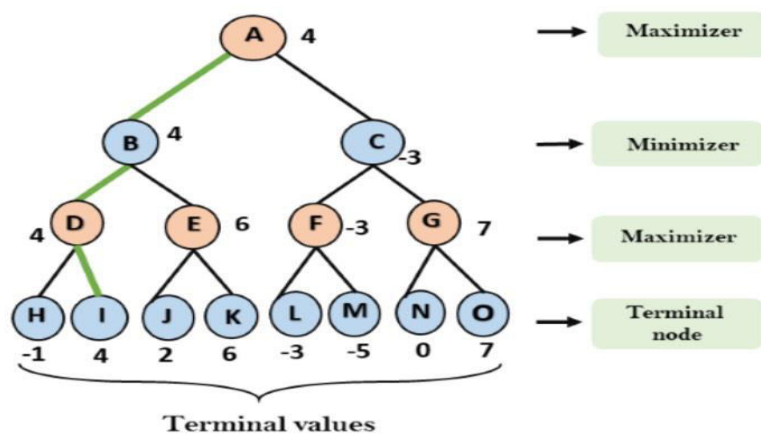
**Step 3:** In the next step, it's a turn for minimizer, so it will compare all nodes value with  $+\infty$ , and will find the 3<sup>rd</sup> layer node values.

For node B =  $\min(4, 6) = 4$

For node C =  $\min(-3, 7) = -3$



**Step 4:** Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.



For node A  $\max(4, -3) = 4$

That was the complete workflow of the minimax two player game.

### **Properties of Mini-Max algorithm:**

- **Complete-** Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- **Optimal-** Min-Max algorithm is optimal if both opponents are playing optimally.
- **Time complexity-** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is  $O(b^m)$ , where  $b$  is branching factor of the game-tree, and  $m$  is the maximum depth of the tree.
- **Space Complexity-** Space complexity of Mini-max algorithm is also similar to DFS which is  $O(bm)$ .

### **Limitation of the minimax Algorithm:**

The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc. This type of games has a huge branching factor, and the player has lots of choices to decide.

### **Alpha-Beta Pruning:**

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.

The two-parameter can be defined as:

**Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is  $-\infty$ .

**Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is  $+\infty$ .

The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Note: To better understand this topic, kindly study the minimax algorithm.

### **Condition for Alpha-beta pruning:**

The main condition which required for alpha-beta pruning is:

$$\alpha \geq \beta$$

### Key points about alpha-beta pruning:

- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

### Pseudo-code for Alpha-beta Pruning:

function minimax(node, depth, alpha, beta, maximizingPlayer) is

**if** depth == 0 or node is a terminal node then

**return static** evaluation of node

**if** MaximizingPlayer then     // for Maximizer Player

    maxEva= -infinity

**for** each child of node **do**

        eva= minimax(child, depth-1, alpha, beta, False)

    maxEva= max(maxEva, eva)

    alpha= max(alpha, maxEva)

**if** beta<=alpha

**break**

**return** maxEva

**else**                     // for Minimizer player

    minEva= +infinity

**for** each child of node **do**

        eva= minimax(child, depth-1, alpha, beta, **true**)

    minEva= min(minEva, eva)

    beta= min(beta, eva)

**if** beta<=alpha

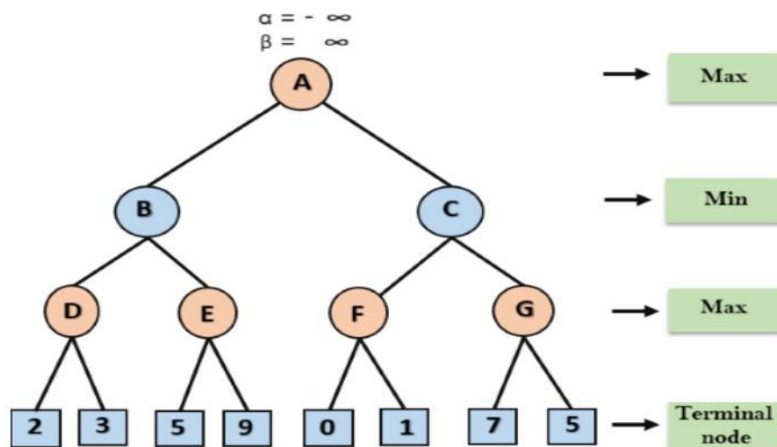
**break**

**return** minEva

## Working of Alpha-Beta Pruning:

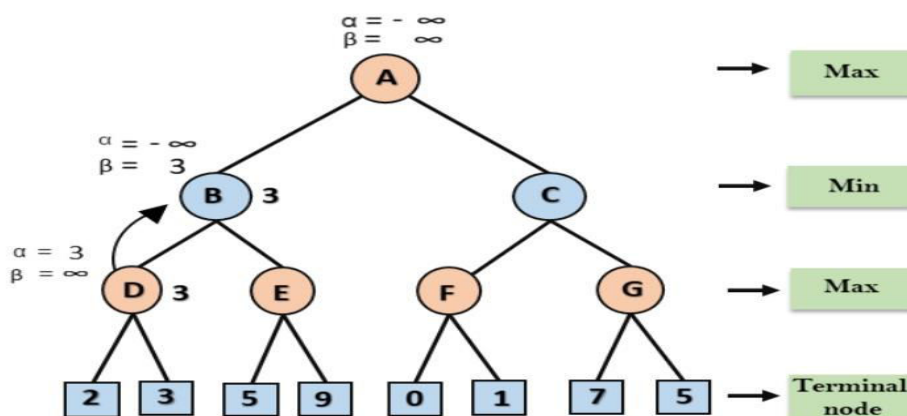
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

**Step 1:** At the first step the, Max player will start first move from node A where  $\alpha = -\infty$  and  $\beta = +\infty$ , these value of alpha and beta passed down to node B where again  $\alpha = -\infty$  and  $\beta = +\infty$ , and Node B passes the same value to its child D.



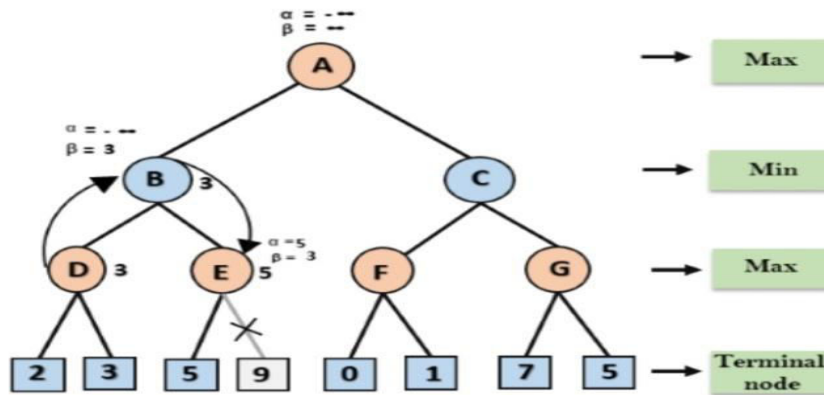
**Step 2:** At Node D, the value of  $\alpha$  will be calculated as its turn for Max. The value of  $\alpha$  is compared with firstly 2 and then 3, and the max  $(2, 3) = 3$  will be the value of  $\alpha$  at node D and node value will also 3.

**Step 3:** Now algorithm backtrack to node B, where the value of  $\beta$  will change as this is a turn of Min, Now  $\beta = +\infty$ , will compare with the available subsequent nodes value, i.e.  $\min(\infty, 3) = 3$ , hence at node B now  $\alpha = -\infty$ , and  $\beta = 3$ .



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of  $\alpha = -\infty$ , and  $\beta = 3$  will also be passed.

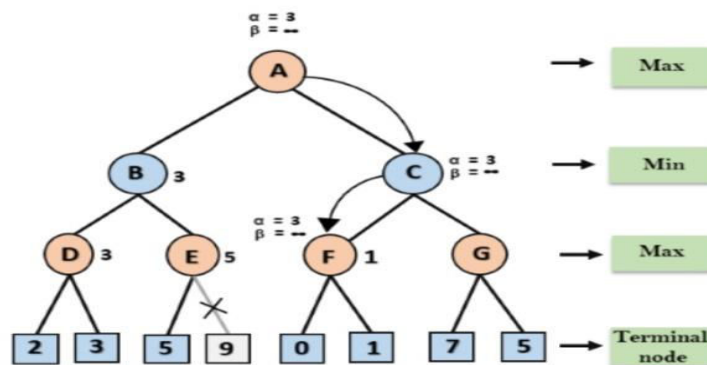
**Step 4:** At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so  $\max(-\infty, 5) = 5$ , hence at node E  $\alpha = 5$  and  $\beta = 3$ , where  $\alpha \geq \beta$ , so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



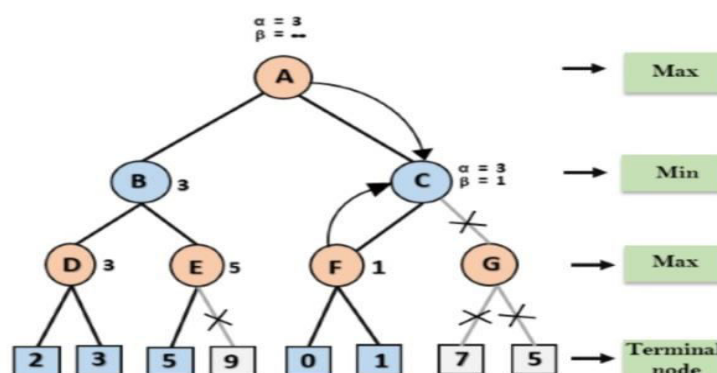
**Step 5:** At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as  $\max(-\infty, 3) = 3$ , and  $\beta = +\infty$ , these two values now passes to right successor of A which is Node C.

At node C,  $\alpha = 3$  and  $\beta = +\infty$ , and the same values will be passed on to node F.

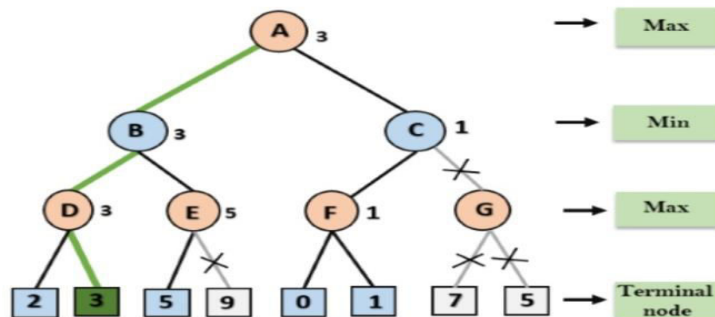
**Step 6:** At node F, again the value of  $\alpha$  will be compared with left child which is 0, and  $\max(3, 0) = 3$ , and then compared with right child which is 1, and  $\max(3, 1) = 3$  still  $\alpha$  remains 3, but the node value of F will become 1.



**Step 7:** Node F returns the node value 1 to node C, at C  $\alpha = 3$  and  $\beta = +\infty$ , here the value of beta will be changed, it will compare with 1 so  $\min(\infty, 1) = 1$ . Now at C,  $\alpha = 3$  and  $\beta = 1$ , and again it satisfies the condition  $\alpha \geq \beta$ , so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



**Step 8:** C now returns the value of 1 to A here the best value for A is  $\max(3, 1) = 3$ . Following is the final game tree which is showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



### Move Ordering in Alpha-Beta pruning:

The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.

It can be of two types:

**Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is  $O(b^m)$ .

**Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is  $O(b^{m/2})$ .

### Rules to find good ordering:

Following are some rules to find good ordering in alpha-beta pruning:

- Occur the best move from the shallowest node.
- Order the nodes in the tree such that the best nodes are checked first.
- Use domain knowledge while finding the best move. Ex: for Chess, try order: captures first, then threats, then forward moves, backward moves.
- We can bookkeep the states, as there is a possibility that states may repeat.

# Basic Knowledge Representation and Reasoning

## Propositional logic in Artificial intelligence:

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. 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:**

- a) It is Sunday.
- b) The Sun rises from West (False proposition)
- c)  $3+3=7$  (False proposition)
- d) 5 is a prime number.

### **Following are some basic facts about propositional logic:**

- 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**.
- A proposition formula which has both true and false values is called **contingency**
- 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
- Compound proposition



- **Atomic Proposition:** 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:**

- a)  $2+2$  is  $4$ , it is an atomic proposition as it is a **true** fact.
- b) "The Sun is cold" is also a proposition as it is a **false** fact.

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

**Example:**

- a) "It is raining today, and street is wet."
- b) "Ankit is a doctor, and his clinic is in Mumbai."

**Logical Connectives:**

Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

**Negation:** A sentence such as  $\neg P$  is called negation of P. A literal can be either Positive literal or negative literal.

**Conjunction:** A sentence which has  $\wedge$  connective such as,  $P \wedge Q$  is called a conjunction.

**Example:** Rohan is intelligent and hardworking. It can be written as,

**P= Rohan is intelligent,**

**Q= Rohan is hardworking.  $\rightarrow P \wedge Q$ .**

**Disjunction:** A sentence which has  $\vee$  connective, such as  $P \vee Q$ . is called disjunction, where P and Q are the propositions.

**Example: "Ritika is a doctor or Engineer",**

Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as  $P \vee Q$ .

**Implication:** A sentence such as  $P \rightarrow Q$ , is called an implication. Implications are also known as if-then rules. It can be represented as

**If** it is raining, then the street is wet.

Let P= It is raining, and Q= Street is wet, so it is represented as  $P \rightarrow Q$

**Biconditional:** A sentence such as  $P \Leftrightarrow Q$  is a **Biconditional sentence, example If I am breathing, then I am alive**

P= I am breathing, Q= I am alive, it can be represented as  $P \Leftrightarrow Q$ .

**Following is the summarized table for Propositional Logic Connectives:**



Connective symbols	Word	Technical term	Example
$\wedge$	AND	Conjunction	$A \wedge B$
$\vee$	OR	Disjunction	$A \vee B$
$\rightarrow$	Implies	Implication	$A \rightarrow B$
$\leftrightarrow$	If and only if	Biconditional	$A \leftrightarrow B$
$\neg$ or $\sim$	Not	Negation	$\neg A$ or $\sim B$

### Truth Table:

In propositional logic, we need to know the truth values of propositions in all possible scenarios. We can combine all the possible combination with logical connectives, and the representation of these combinations in a tabular format is called **Truth table**. Following are the truth table for all logical connectives:

#### For Negation:

P	$\neg P$
True	False
False	True

#### For Conjunction:

P	Q	$P \wedge Q$
True	True	True
True	False	False
False	True	False
False	False	False

#### For disjunction:

P	Q	$P \vee Q$
True	True	True
False	True	True
True	False	True
False	False	False

#### For Implication:

P	Q	$P \rightarrow Q$
True	True	True
True	False	False
False	True	True
False	False	True

#### For Biconditional:

P	Q	$P \leftrightarrow Q$
True	True	True
True	False	False
False	True	False
False	False	True

### Truth table with three propositions:

We can build a proposition composing three propositions P, Q, and R. This truth table is made-up of 8n Tuples as we have taken three proposition symbols.

P	Q	R	$\neg R$	$P \vee Q$	$P \vee Q \rightarrow \neg R$
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

### Precedence of connectives:

Just like arithmetic operators, there is a precedence order for propositional connectors or logical operators. This order should be followed while evaluating a propositional problem. Following is the list of the precedence order for operators:

Precedence	Operators
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction(AND)
Fourth Precedence	Disjunction(OR)
Fifth Precedence	Implication
Six Precedence	Biconditional

### 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.

Let's take two propositions A and B, so for logical equivalence, we can write it as  $A \Leftrightarrow B$ . In below truth table we can see that column for  $\neg A \vee B$  and  $A \rightarrow B$ , are identical hence A is Equivalent to B

A	B	$\neg A$	$\neg A \vee B$	$A \rightarrow B$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

### Properties of Operators:

#### ➤ Commutativity:

$P \wedge Q = Q \wedge P$ , or

$P \vee Q = Q \vee P$ .

#### ➤ Associativity:

$(P \wedge Q) \wedge R = P \wedge (Q \wedge R)$ ,

$(P \vee Q) \vee R = P \vee (Q \vee R)$

#### ➤ Identity element:

$P \wedge \text{True} = P$ ,

$P \vee \text{True} = \text{True}$ .

#### ➤ Distributive:

$P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$ .

$P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$ .

#### ➤ DE Morgan's Law:

$\neg (P \wedge Q) = (\neg P) \vee (\neg Q)$

$\neg (P \vee Q) = (\neg P) \wedge (\neg Q)$ .

### Double-negation elimination:

$\neg (\neg P) = P$ .

### Limitations of Propositional logic:

We cannot represent relations like ALL, some, or none with propositional logic.

#### Example:

- All the girls 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.

## **First-Order Logic:**

In the topic of Propositional logic, we have seen that how to represent statements using propositional logic. But unfortunately, 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:**

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is 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 (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:

**Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus, .....

**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, .....

As a natural language, first-order logic also has two main parts:

- **Syntax**
- **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. We write statements in short-hand notation in FOL.

### **Basic Elements of First-order logic:**

Following are the basic elements of FOL syntax:

<b>Constant</b>	1, 2, A, John, Mumbai, cat,....
<b>Variables</b>	x, y, z, a, b,....
<b>Predicates</b>	Brother, Father, >,....
<b>Function</b>	sqrt, LeftLegOf, ...
<b>Connectives</b>	$\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$
<b>Equality</b>	$=$
<b>Quantifier</b>	$\forall, \exists$

### 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.

We can represent atomic sentences as **Predicate (term1, term2, ....., term n)**.

**Example: Ravi and Ajay are brothers:  $\Rightarrow$  Brothers(Ravi, Ajay).**  
**Chinky is a cat:  $\Rightarrow$  cat (Chinky).**

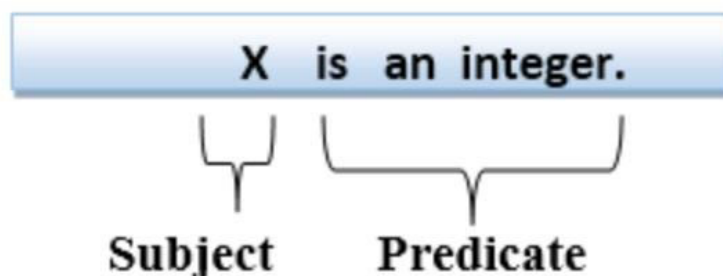
### Complex Sentences:

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, the first part x is the subject of the statement and 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)**
- **Existential quantifier, (for some, at least one).**

### Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol  $\forall$ , which resembles an inverted A.

If  $x$  is a variable, then  $\forall x$  is read as:

**For all  $x$**

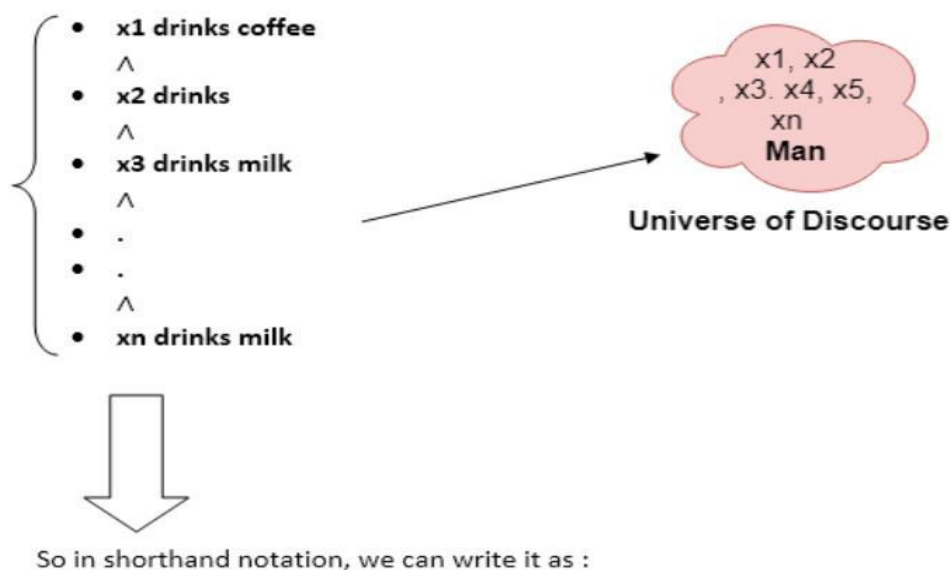
**For each  $x$**

**For every  $x$ .**

**Example:**

**All man drink coffee.**

Let a variable  $x$  which refers to a cat so all  $x$  can be represented in UOD as below:



**$\forall x \text{ man}(x) \rightarrow \text{drink}(x, \text{coffee})$ .**

It will be read as: There are all  $x$  where  $x$  is a man who drink coffee.

## Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

It is denoted by the logical operator  $\exists$ , which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

If  $x$  is a variable, then existential quantifier will be  $\exists x$  or  $\exists(x)$ . And it will be read as:

**There exists a 'x.'**

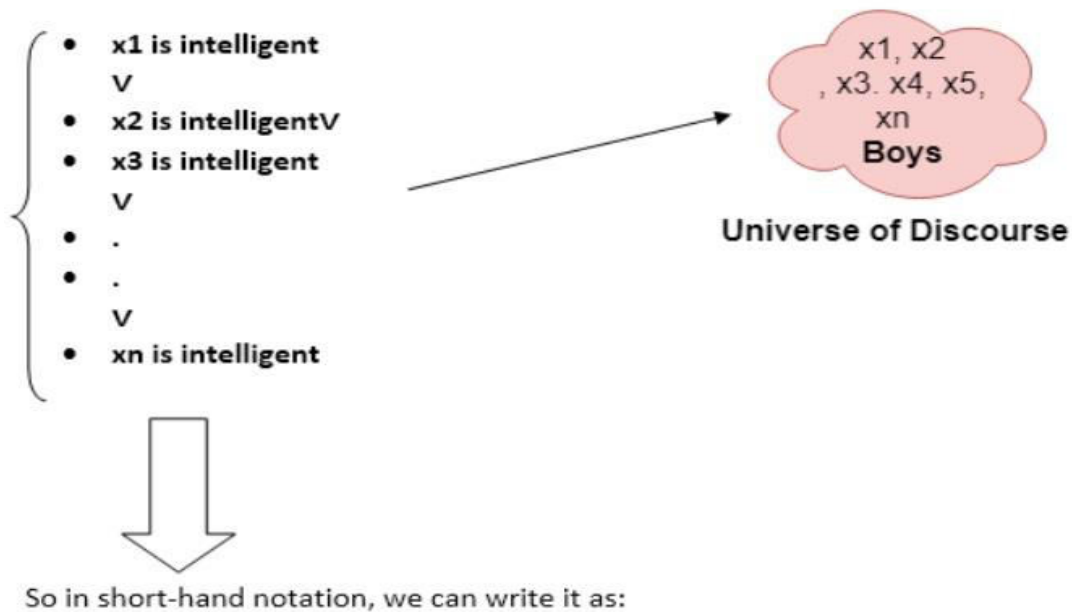
**For some 'x.'**

**For at least one 'x.'**

**Example:**

**Some boys are intelligent.**

## Some boys are intelligent.



**$\exists x: \text{boys}(x) \wedge \text{intelligent}(x)$**

It will be read as: There are some  $x$  where  $x$  is a boy who is intelligent.

## Points to remember:

- The main connective for universal quantifier  $\forall$  is implication  $\rightarrow$ .
- The main connective for existential quantifier  $\exists$  is and  $\wedge$ .

## Properties of Quantifiers:

- In universal quantifier,  $\forall x \forall y$  is similar to  $\forall y \forall x$ .
- In Existential quantifier,  $\exists x \exists y$  is similar to  $\exists y \exists x$ .
- $\exists x \forall y$  is not similar to  $\forall y \exists x$ .
- Some Examples of FOL using quantifier:

### 1. All birds fly.

In this question the predicate is "**fly(bird)**."

And since there are all birds who fly so it will be represented as follows.

$$\forall x \text{ bird}(x) \rightarrow \text{fly}(x).$$

### 2. Every man respects his parent.

In this question, the predicate is "**respect(x, y)**," where **x=man, and y= parent**.

Since there is every man so will use  $\forall$ , and it will be represented as follows:

$$\forall x \text{ man}(x) \rightarrow \text{respects}(x, \text{parent}).$$

### 3. Some boys play cricket.

In this question, the predicate is "**play(x, y)**," where **x= boys, and y= game**. Since there are some boys so we will use  $\exists$ , and it will be represented as:

$$\exists x \text{ boys}(x) \rightarrow \text{play}(x, \text{cricket}).$$

### 4. Not all students like both Mathematics and Science.

In this question, the predicate is "**like(x, y)**," where **x= student, and y= subject**.

Since there are not all students, so we will use  $\forall$  with negation, so following representation for this:

$$\neg \forall (x) [ \text{student}(x) \rightarrow \text{like}(x, \text{Mathematics}) \wedge \text{like}(x, \text{Science}) ].$$

### 5. Only one student failed in Mathematics.

In this question, the predicate is "**failed(x, y)**," where **x= student, and y= subject**.

Since there is only one student who failed in Mathematics, so we will use following representation for this:

$$\exists (x) [ \text{student}(x) \rightarrow \text{failed}(x, \text{Mathematics}) \wedge \forall (y) [ \neg(x=y) \wedge \text{student}(y) \rightarrow \neg \text{failed}(y, \text{Mathematics}) ] ].$$

## Free and Bound Variables:

The quantifiers interact with variables which appear in a suitable way. There are two types of variables in First-order logic which are given below:

**Free Variable:** A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

**Example:**  $\forall x \exists (y)[P(x, y, z)]$ , where **z is a free variable**.

**Bound Variable:** A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

**Example:**  $\forall x [A(x) B(y)]$ , here **x and y are the bound variables**.



## **Forward Chaining and backward chaining :**

In artificial intelligence, forward and backward chaining is one of the important topics, but before understanding forward and backward chaining lets first understand that from where these two terms came.

### **Inference engine:**

The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts. The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:

1. **Forward chaining**
2. **Backward chaining**

### **Horn Clause and Definite clause:**

Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm. Logical inference algorithms use forward and backward chaining approaches, which require KB in the form of the **first-order definite clause**.

- **Definite clause:** A clause which is a disjunction of literals with **exactly one positive literal** is known as a definite clause or strict horn clause.
- **Horn clause:** A clause which is a disjunction of literals with **at most one positive literal** is known as horn clause. Hence all the definite clauses are horn clauses.

**Example:**  $(\neg p \vee \neg q \vee k)$ . It has only one positive literal k.

It is equivalent to  $p \wedge q \rightarrow k$ .

### **1. Forward Chaining**

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

### **Properties of Forward-Chaining:**

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.

- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Consider the following famous example which we will use in both approaches:

**Example:**

**"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."**

Prove that **"Robert is criminal."**

To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

### **Facts Conversion into FOL:**

It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)

**American(p)  $\wedge$  weapon(q)  $\wedge$  sells(p, q, r)  $\wedge$  hostile(r)  $\rightarrow$  Criminal(p) ... (1)**

Country A has some missiles. **?p Owns(A, p)  $\wedge$  Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.

**Owns(A, T1) ..... (2)**

**Missile(T1) ..... (3)**

All of the missiles were sold to country A by Robert.

**?p Missiles(p)  $\wedge$  Owns(A, p)  $\rightarrow$  Sells(Robert, p, A) ..... (4)**

Missiles are weapons.

**Missile(p)  $\rightarrow$  Weapons(p) ..... (5)**

Enemy of America is known as hostile.

**Enemy(p, America)  $\rightarrow$  Hostile(p) ..... (6)**

Country A is an enemy of America.

**Enemy(A, America) ..... (7)**

Robert is American

**American(Robert). ..... (8)**

### **Forward chaining proof:**

#### **Step-1:**

In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: **American(Robert), Enemy(A,**

America), Owns(A, T1), and Missile(T1). All these facts will be represented as below.



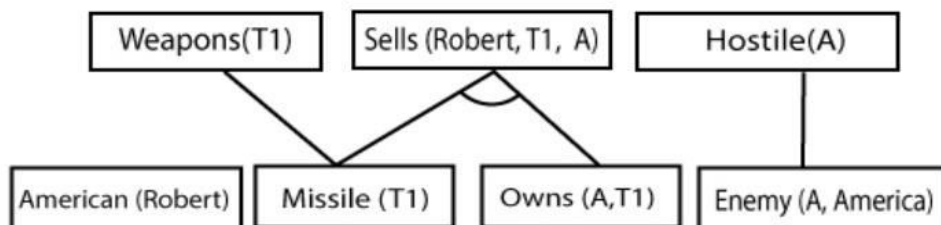
**Step-2:**

At the second step, we will see those facts which infer from available facts and with satisfied premises.

Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

Rule-(2) and (3) are already added.

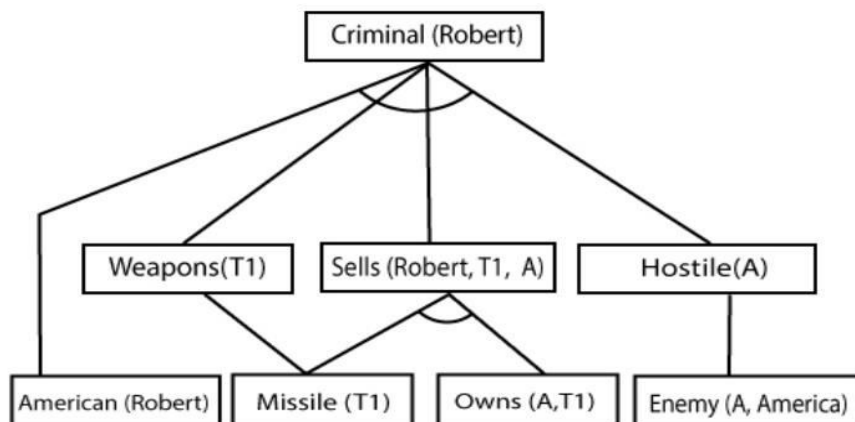
Rule-(4) satisfy with the substitution {p/T1}, so **Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).



Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).

**Step-3:**

At step-3, as we can check Rule-(1) is satisfied with the substitution {p/Robert, q/T1, r/A}, so we can add **Criminal(Robert)** which infers all the available facts. And hence we reached our goal statement.



**Hence it is proved that Robert is Criminal using forward chaining approach.**

## 2. Backward Chaining:

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

### Properties of backward chaining:

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.

The backward-chaining method mostly used a **depth-first search** strategy for proof.

### Example:

In backward-chaining, we will use the same above example, and will rewrite all the rules.

**American (p)  $\wedge$  weapon(q)  $\wedge$  sells (p, q, r)  $\wedge$  hostile(r)  $\rightarrow$  Criminal(p) ... (1)**

**Owns(A, T1) ..... (2)**

**Missile(T1) ..... (3)**

**?p Missiles(p)  $\wedge$  Owns (A, p)  $\rightarrow$  Sells (Robert, p, A) ..... (4)**

**Missile(p)  $\rightarrow$  Weapons (p) ..... (5)**

**Enemy(p, America)  $\rightarrow$  Hostile(p) ..... (6)**

**Enemy (A, America) ..... (7)**

**American(Robert). ..... (8)**

### Backward-Chaining proof:

In Backward chaining, we will start with our goal predicate, which is **Criminal(Robert)**, and then infer further rules.

#### Step-1:

At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.

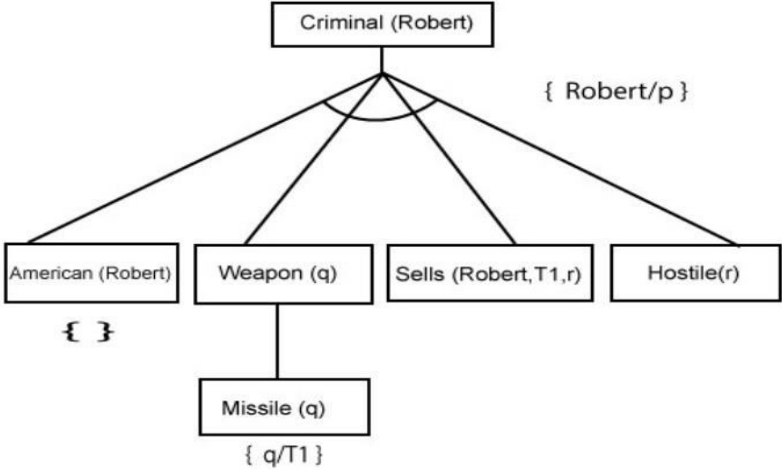
Criminal (Robert)

**Step-2:**

At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution { Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.

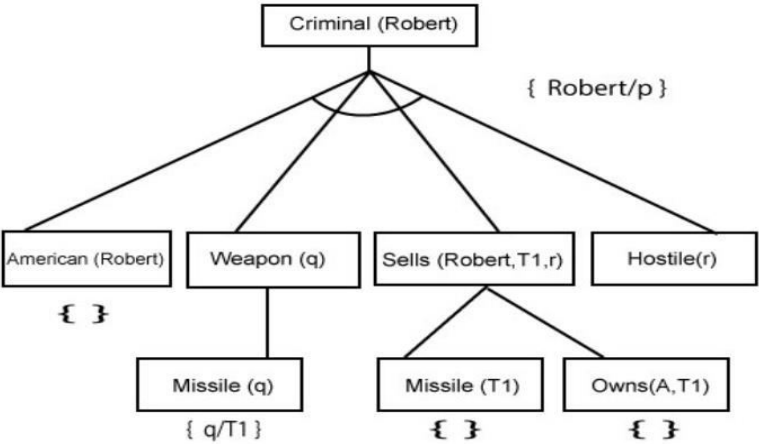
**Here we can see American (Robert) is a fact, so it is proved here.**

**Step-3:** At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



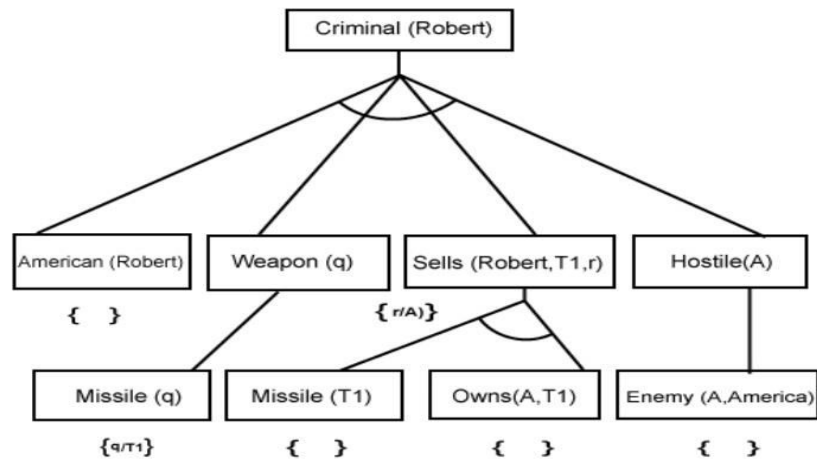
**Step-4:**

At step-4, we can infer facts Missile(T1) and Owns(A, T1) form Sells(Robert, T1, r) which satisfies the **Rule- 4**, with the substitution of A in place of r. So these two statements are proved here.



### Step-5:

At step-5, we can infer the fact **Enemy(A, America)** from **Hostile(A)** which satisfies Rule- 6. And hence all the statements are proved true using backward chaining.



## Introduction to Probabilistic Reasoning :

### 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  $A \rightarrow B$ , 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:**

### **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.

$$\text{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

$P(\neg A)$  = 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:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

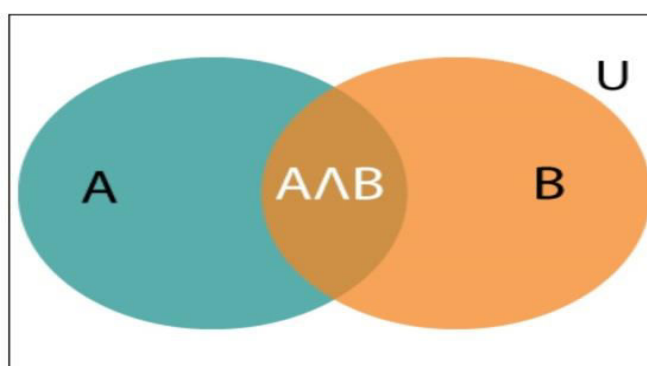
Where  $P(A \cap B)$ = Joint probability of a and B

$P(B)$ = 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:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

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  $P(A \cap B)$  by  $P(B)$ .



**Example:**

In a class, there are 70% of the students who like English and 40% of the students who likes English and mathematics, and then what is the percent of students those who like English also like mathematics?

**Solution:**

Let, A is an event that a student likes Mathematics

B is an event that a student likes English.



$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

Hence, 57% are the students who like English also like Mathematics.

### Bayes' theorem:

- 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:

$$P(A \cap B) = P(A|B) P(B) \text{ or}$$

Similarly, the probability of event B with known event A:

$$P(A \cap B) = P(B|A) P(A)$$

Equating right hand side of both the equations, we will get:

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)} \quad \dots(a)$$

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  $P(B) = \sum_{i=1}^k P(A_i) \cdot P(B|A_i)$ , hence the Bayes' rule can be written as:

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

Where  $A_1, A_2, A_3, \dots, A_n$  is a set of mutually exclusive and exhaustive events.

## Applying Bayes' Rule:

Bayes' rule allows us to compute the single term  $P(A|B)$  in terms of  $P(B|A)$ ,  $P(B)$ , 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:

$$P(\text{cause} | \text{effect}) = \frac{P(\text{effect} | \text{cause}) P(\text{cause})}{P(\text{effect})}$$

### 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:

$$P(a|b) = 0.8$$

$$P(b) = 1/30000$$

$$P(a) = 0.02$$

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)} = \frac{0.8 * (\frac{1}{20000})}{0.02} = 0.0013333333.$$

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

**Example-2:**

**Question:** From a standard deck of playing cards, a single card is drawn. The probability that the card is king is 4/52, then calculate posterior probability P(King|Face), which means the drawn face card is a king card.

**Solution:**

$$P(\text{king}|\text{face}) = \frac{P(\text{Face}|\text{king}) * P(\text{King})}{P(\text{Face})} \dots\dots(i)$$

P(king): probability that the card is King= 4/52= 1/13 P(face): probability that a card is a face card= 3/13 P(Face|King): probability of face card when we assume it is a king = 1

Putting all values in equation (i) we will get:

$$P(\text{king}|\text{face}) = \frac{1 * (\frac{1}{13})}{(\frac{3}{13})} = 1/3, \text{ it is a probability that a face card is a king card.}$$

**Application of Bayes' theorem in Artificial Intelligence:**

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

## MODULE – III

# ADVANCED KNOWLEDGE REPRESENTATION AND REASONING

### What is knowledge representation?

Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. **But how machines do all these things comes under knowledge representation and reasoning.** Hence we can describe Knowledge representation as following:

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior 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.

### Issues in Knowledge Representation

The issues that arise while using KR techniques are many. Some of these are explained below.

#### **Important Attributed:**

Any attribute of objects so basic that they occur in almost every problem domain?

There are two attributed “instance” and “isa”, that are general significance. These attributes are important because they support property inheritance.

#### **Relationship among attributes:**

Any important relationship that exists among object attributed?

The attributes we use to describe objects are themselves entities that we represent. The relationship between the attributes of an object, independent of specific knowledge they encode, may hold properties like:

1. **Inverse** — This is about consistency check, while a value is added to one attribute. The entities are related to each other in many different ways.
2. **Existence in an isa hierarchy** — This is about generalization-specification, like, classes of objects and specialized subsets of those classes, there are attributes and specialization of attributes. For example, the attribute height is a specialization of general attribute physical-size which is, in turn, a specialization of physical-attribute. These generalization-specialization relationships are important for attributes because they support inheritance.
3. **Technique for reasoning about values** — This is about reasoning values of attributes not given explicitly. Several kinds of information are used in reasoning, like, height: must be in a unit of length, Age: of a person cannot be greater than the age of person's parents. The values are often specified when a knowledge base is created.
4. **Single valued attributes** — This is about a specific attribute that is guaranteed to take a unique value. For example, a baseball player can at time have only a single height and be a member of only one team. KR systems take different approaches to provide support for single valued attributes.

### **Choosing Granularity:**

At what level of detail should the knowledge be represented?

Regardless of the KR formalism, it is necessary to know:

- At what level should the knowledge be represented and what are the primitives?
- Should there be a small number or should there be a large number of low-level primitives or High-level facts.
- High-level facts may not be adequate for inference while Low-level primitives may require a lot of storage.

## Example of Granularity:

- Suppose we are interested in following facts:

*John spotted Sue.*

*This could be represented as*

Spotted (agent(John),object (Sue))

Such a representation would make it easy to answer questions such are:

- Who spotted Sue?

Suppose we want to know:

- Did John see Sue?

Given only one fact, we cannot discover that answer.

We can add other facts, such as

Spotted(x, y) -> saw(x, y)

We can now infer the answer to the question.

### Set of objects:

How should sets of objects be represented?

There are certain properties of objects that are true as member of a set but not as individual;

### Example: Consider the assertion made in the sentences:

*“there are more sheeps than people in Australia”, and*

*“English speakers can be found all over the world.”*

To describe these facts, the only way is to attach assertion to the sets representing people, sheep, and English.

The reason to represent sets of objects is: if a property is true for all or most elements of a set, then it is more efficient to associate it once with the set rather than to associate it explicitly with every elements of the set.

This is done,

- in logical representation through the use of universal quantifier, and
- in hierarchical structure where node represent sets and inheritance propagate set level assertion down to individual.

### Finding Right structure:

Given a large amount of knowledge stored in a database, how can relevant parts are accessed when they are needed?

This is about access to right structure for describing a particular situation.

This requires, selecting an initial structure and then revising the choice.

While doing so, it is necessary to solve following problems:

- How to perform an initial selection of the most appropriate structure.
- How to fill in appropriate details from the current situations.
- How to find a better structure if the one chosen initially turns out not to be appropriate.
- What to do if none of the available structures is appropriate.
- When to create and remember a new structure.

There is no good, general purpose method for solving all these problems. Some knowledge representation techniques solve some of these issues.

## Non-monotonic Reasoning

- In Non-monotonic reasoning, some conclusions may be invalidated if we add some more information to our knowledge base.
- Logic will be said as non-monotonic if some conclusions can be invalidated by adding more knowledge into our knowledge base.
- Non-monotonic reasoning deals with incomplete and uncertain models.
- "Human perceptions for various things in daily life, "is a general example of non-monotonic reasoning.

**Example:** Let suppose the knowledge base contains the following knowledge:

- **Birds can fly**
- **Penguins cannot fly**
- **Pitty is a bird**

So from the above sentences, we can conclude that **Pitty can fly**.

However, if we add one another sentence into knowledge base "**Pitty is a penguin**", which concludes "**Pitty cannot fly**", so it invalidates the above conclusion.

### **Advantages of Non-monotonic reasoning:**

- For real-world systems such as Robot navigation, we can use non-monotonic reasoning.

- In Non-monotonic reasoning, we can choose probabilistic facts or can make assumptions.

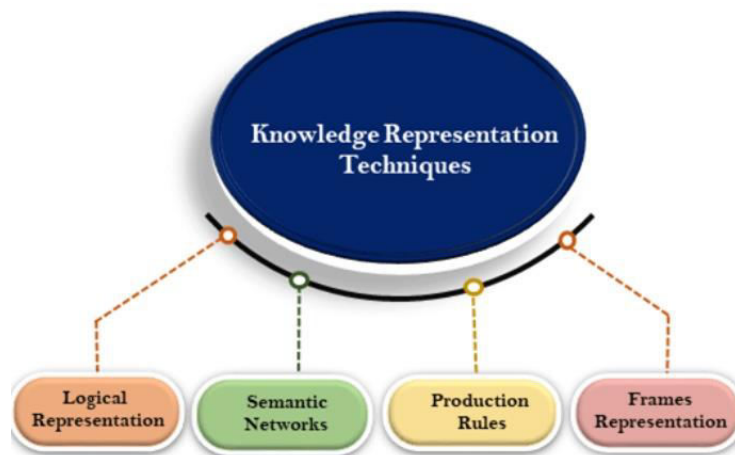
### **Disadvantages of Non-monotonic Reasoning:**

- In non-monotonic reasoning, the old facts may be invalidated by adding new sentences.
- It cannot be used for theorem **proving**.

## **Schemes/Techniques of knowledge representation**

Here are mainly four ways of knowledge representation which are given as follows:

1. Logical Representation
2. Semantic Network Representation
3. Frame Representation
4. Production Rules





# 1. 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. This representation lays down some important communication rules. It consists of precisely defined syntax and semantics which supports the sound inference. 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.
- How to write those symbols.

## **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 categorized into mainly two logics:

- a. Propositional Logics
- b. Predicate logics

## **Advantages of logical representation:**

1. Logical representation enables us to do logical reasoning.
2. Logical representation is the basis for the programming languages.

## **Disadvantages of logical Representation:**

1. Logical representations have some restrictions and are challenging to work with.
2. Logical representation technique may not be very natural, and inference may not be so efficient.

*Note: Do not be confused with logical representation and logical reasoning as logical representation is a representation language and reasoning is a process of thinking logically*

## **2. 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.

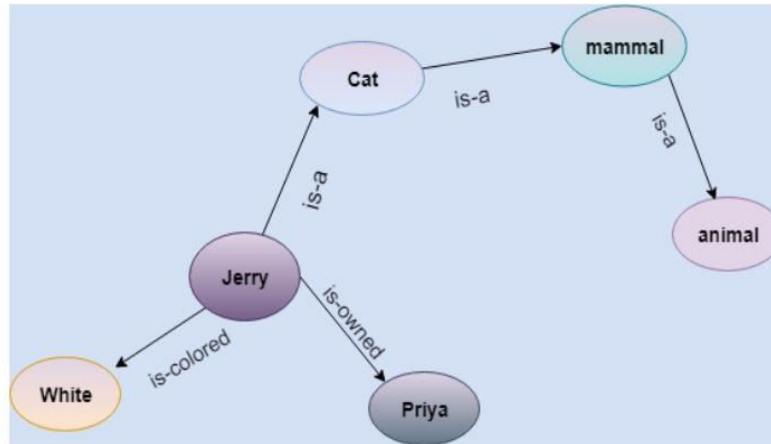
This representation consists of mainly two types of relations:

- a) IS-A relation (Inheritance)
- b) Kind-of-relation

**Example:** Following are some statements which we need to represent in the form of nodes and arcs.

### **Statements:**

- Jerry is a cat.
- Jerry is a mammal
- Jerry is owned by Priya.
- Jerry is brown colored.
- All Mammals are animal.



In the above diagram, we have represented the different type of knowledge in the form of nodes and arcs. Each object is connected with another object by some relation.

### **Drawbacks in Semantic representation:**

1. Semantic networks take more computational time at runtime as we need to traverse the complete network tree to answer some questions. It might be possible in the worst case scenario that after traversing the entire tree, we find that the solution does not exist in this network.
2. Semantic networks try to model human-like memory (Which has 1015 neurons and links) to store the information, but in practice, it is not possible to build such a vast semantic network.
3. These types of representations are inadequate as they do not have any equivalent quantifier, e.g., for all, for some, none, etc.
4. Semantic networks do not have any standard definition for the link names.
5. These networks are not intelligent and depend on the creator of the system.

### **Advantages of Semantic network:**

1. Semantic networks are a natural representation of knowledge.
2. Semantic networks convey meaning in a transparent manner.
3. These networks are simple and easily understandable.

### 3. 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. Example: IF-NEEDED facts are called when data of any particular slot is needed. 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 are derived from semantic networks and later evolved into our modern-day classes and objects. A single frame is not much useful. Frames system consists 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.

#### Example: 1

Let's take an example of a frame for a book

Slots	Filters
<b>Title</b>	Artificial Intelligence
<b>Genre</b>	Computer Science
<b>Author</b>	Peter Norvig

<b>Edition</b>	Third Edition
<b>Year</b>	1996
<b>Page</b>	1152

### Example 2:

Let's suppose we are taking an entity, Peter. Peter is an engineer as a profession, and his age is 25, he lives in city London, and the country is England. So following is the frame representation for this:

Slots	Filter
<b>Name</b>	Peter
<b>Profession</b>	Doctor
<b>Age</b>	25
<b>Marital status</b>	Single
<b>Weight</b>	78

### Advantages of frame representation:

1. The frame knowledge representation makes the programming easier by grouping the related data.
2. The frame representation is comparably flexible and used by many applications in AI.

3. It is very easy to add slots for new attribute and relations.
4. It is easy to include default data and to search for missing values.
5. Frame representation is easy to understand and visualize.

#### **Disadvantages of frame representation:**

1. In frame system inference mechanism is not be easily processed.
2. Inference mechanism cannot be smoothly proceeded by frame representation.
3. Frame representation has a much generalized approach.

#### **4. 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

In production rules agent checks for the condition and if the condition exists then production rule fires and corresponding action is carried out. The condition part of the rule determines which rule may be applied to a problem. And the action part carries out the associated problem-solving steps. This complete process is called a recognize-act cycle.

The working memory contains the description of the current state of problems-solving and rule can write knowledge to the working memory. This knowledge match and may fire other rules.

If there is a new situation (state) generates, then multiple production rules will be fired together, this is called conflict set. In this situation, the agent needs to select a rule from these sets, and it is called a conflict resolution.

#### **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).

**Advantages of Production rule:**

1. The production rules are expressed in natural language.
2. The production rules are highly modular, so we can easily remove, add or modify an individual rule.

**Disadvantages of Production rule:**

1. Production rule system does not exhibit any learning capabilities, as it does not store the result of the problem for the future uses.
2. During the execution of the program, many rules may be active hence rule-based production systems are inefficient

## **Representing Knowledge in an Uncertain Domain**

### **REASONING UNDER UNCERTAINTY**

**BASIC PROBABILITY:**

The word 'Probability' means the chance of occurring of a particular event. It is generally possible to predict the future of an event quantitatively with a certain probability of being correct. The probability is used in such cases where the outcome of the trial is uncertain.

**Probability Definition:**

The probability of happening of an event A, denoted by P(A), is defined as

$$P(A) = \frac{\text{number of cases favourable to A}}{\text{number of possible outcomes}}$$

Thus, if an event can happen in  $m$  ways and fails to occur in  $n$  ways and  $m+n$  ways is equally likely to occur then the probability of happening of the event  $A$  is given by

$$P(A) = \frac{m}{m+n}$$

And the probability of non-happening of  $A$  is

$$P(\bar{A}) = \frac{n}{m+n}$$

**Note:**

1. The probability of an event which is certain to occur is one.
2. The probability of an event which is impossible to zero.
3. If the probability of happening of an event  $P(A)$  and that of not happening is  $P(\bar{A})$ , then

$$P(A) + P(\bar{A}) = 1, 0 \leq P(A) \leq 1, 0 \leq P(\bar{A}) \leq 1.$$

**Important Terms related to Probability:**

**1. Trial and Event:** The performance of an experiment is called a trial, and the set of its outcomes is termed an event.

**Example:** Tossing a coin and getting head is a trial. Then the event is {HT, TH, HH}

**2. Random Experiment:** It is an experiment in which all the possible outcomes of the experiment are known in advance. But the exact outcomes of any specific performance are not known in advance.



### **Example:**

1. Tossing a Coin
2. Rolling a die
3. Drawing a card from a pack of 52 cards.
4. Drawing a ball from a bag.

**3. Outcome:** The result of a random experiment is called an Outcome.

**Example:** 1. Tossing a coin is an experiment and getting head is called an outcome.

2. Rolling a die and getting 6 is an outcome.

**4. Sample Space:** The set of all possible outcomes of an experiment is called sample space and is denoted by S.

**Example:** When a die is thrown, sample space is  $S = \{1, 2, 3, 4, 5, 6\}$   
It consists of six outcomes 1, 2, 3, 4, 5, 6

**5. Complement of Event:** The set of all outcomes which are in sample space but not an event is called the complement of an event.

**6. Impossible Events:** An event which will never be happened.

**Example1:** Tossing double-headed coins and getting tails in an impossible event.

**Example2:** Rolling a die and getting number  $> 10$  in an impossible outcome.  
 $P(\text{impossible outcome}) = 0$

**7. Sure Outcome/Certain Outcome:** An Outcome which will definitely be happen

**Example1:** Tossing double-headed coins and getting heads only.

**Example2:** Rolling a die and getting number  $< 6$   
 $P(\text{sure outcome}) = 1$   
{1, 2, 3, 4, 5 6} is called sure event  
 $P(\text{sure outcome}) = 1$

**8. Possible Outcome:** An outcome which is possible to occur is called Possible Outcome.

**Example1:** Tossing a fair coin and getting a head on it.

**Example2:** Rolling a die and getting an odd number.

**9. Equally Likely Events:** Events are said to be equally likely if one of them cannot be expected to occur in preference to others. In other words, it means each outcome is as likely to occur as any other outcome.

**Example:** When a die is thrown, all the six faces, i.e., 1, 2, 3, 4, 5 and 6 are equally likely to occur.

**10. Mutually Exclusive or Disjoint Events:** Events are called mutually exclusive if they cannot occur simultaneously.

**Example:** Suppose a card is drawn from a pack of cards, then the events getting a jack and getting a king are mutually exclusive because they cannot occur simultaneously.

**11. Exhaustive Events:** The total number of all possible outcomes of an experiment is called exhaustive events.

**Example:** In the tossing of a coin, either head or tail may turn up. Therefore, there are two possible outcomes. Hence, there are two exhaustive events in tossing a coin.

**12. Independent Events:** Events A and B are said to be independent if the occurrence of any one event does not affect the occurrence of any other event.

$$P(A \cap B) = P(A)P(B).$$

**Example:** A coin is tossed thrice, and all 8 outcomes are equally likely

A: "The first throw results in heads."

B: "The last throw results in Tails."

Prove that event A and B are independent.

## Solution:

Sample Space: [HHH, HHT, HTH, THH, TTT, TTH, THT, HTT]

A: [HHH, HHT, HTH, HTT]

B: [HHT, TTT, THT, HTT]

$A \cap B$ : [HHT, HTT]

$$P(A) = \frac{4}{8} = \frac{1}{2}$$

$$P(B) = \frac{4}{8} = \frac{1}{2}$$

$$P(A \cap B) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$$

**13. Dependent Event:** Events are said to be dependent if occurrence of one affect the occurrence of other events

## BAYE'S 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:

1.  $P(A \wedge B) = P(A|B) P(B)$  or

Similarly, the probability of event B with known event A:

1.  $P(A \wedge B) = P(B|A) P(A)$

Equating right hand side of both the equations, we will get:

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)} \quad \dots(a)$$

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  $P(B) = \sum_{i=1}^k P(A_i) * P(B|A_i)$ , hence the Bayes' rule can be written as:

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

Where  $A_1, A_2, A_3, \dots, A_n$  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:

$$P(\text{cause} | \text{effect}) = \frac{P(\text{effect} | \text{cause}) P(\text{cause})}{P(\text{effect})}$$

### 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:

$$P(a|b) = 0.8$$

$$P(b) = 1/30000$$

$$P(a) = .02$$

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

# **FUZZY LOGIC:**

What is Fuzzy Logic?

Why Fuzzy Logic?

Architecture

Membership Function

Applications of fuzzy logic

Advantages and Disadvantages of Fuzzy logic

Fuzzy Logic vs Probability

## **What is 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.

## Why Fuzzy Logic?

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.

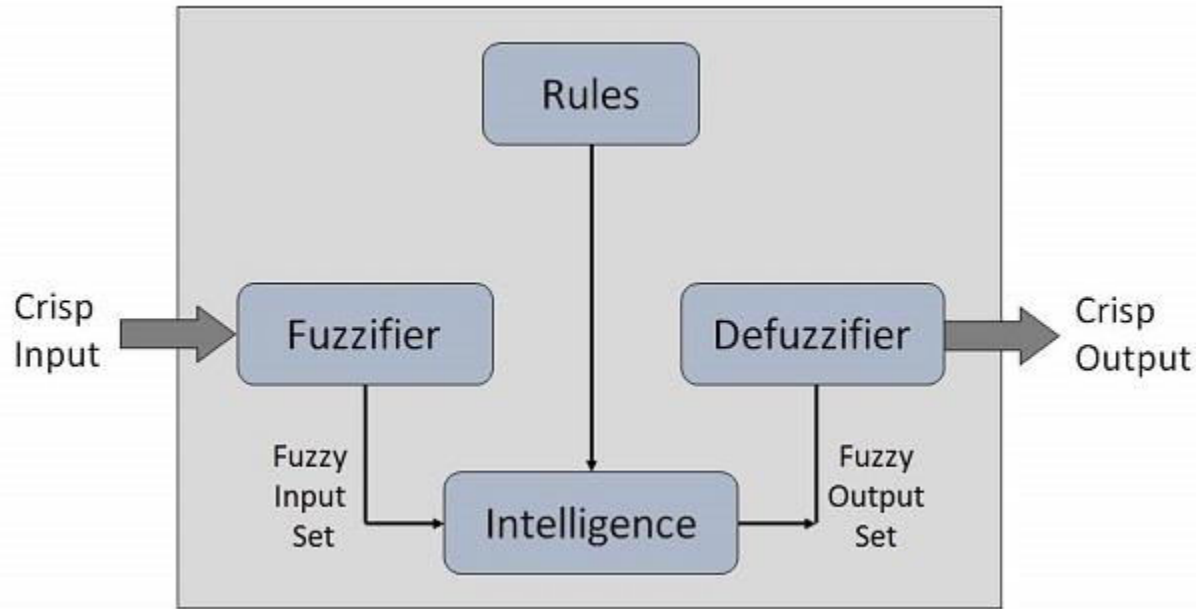
## Fuzzy Logic Systems Architecture

It has four main parts as shown –

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

<b>LP</b>	x is Large Positive
<b>MP</b>	x is Medium Positive
<b>S</b>	x is Small
<b>MN</b>	x is Medium Negative
<b>LN</b>	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  $\mu_A: X \rightarrow [0,1]$ .

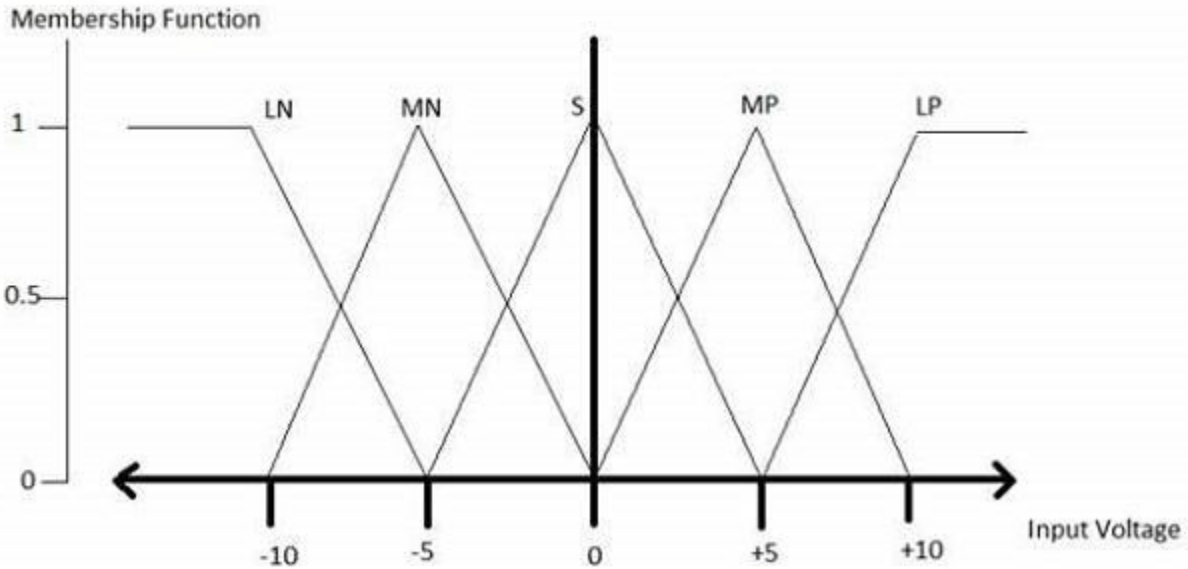
Here, each element of  $X$  is mapped to a value between 0 and 1. It is called **membership value** or **degree of membership**. It quantifies the degree of membership of the element in  $X$  to the fuzzy set  $A$ .

- 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.

### 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.

## Disadvantages of FLSs

- There is no systematic approach to fuzzy system designing.
- They are understandable only when simple.
- They are suitable for the problems which do not need high accuracy.

## Fuzzy Logic vs Probability

<b>Fuzzy Logic</b>	<b>Probability</b>
In fuzzy logic, we basically try to capture the essential concept of vagueness	Probability is associated with events and not facts and those events will either occur or not occur
Fuzzy logic captures the meaning of	Probability theory captures partial

partial truth	knowledge
Fuzzy logic takes truth degrees as a mathematical basis	Probability is a mathematical model of ignorance

## **TRUTH MAINTENANCE SYSTEM:**

A truth maintenance system, or TMS, is a knowledge representation method for representing both beliefs and their dependencies and an algorithm called the "truth maintenance algorithm" that manipulates and maintains the dependencies. The name *truth maintenance* is due to the ability of these systems to restore consistency.

A truth maintenance system maintains consistency between old believed knowledge and current believed knowledge in the knowledge base (KB) through revision. If the current believed statements contradict the knowledge in the KB, then the KB is updated with the new knowledge. It may happen that the same data will again be believed, and the previous knowledge will be required in the KB. If the previous data are not present, but may be required for new inference. But if the previous knowledge was in the KB, then no retracing of the same knowledge is needed. The use of TMS avoids such retracing; it keeps track of the contradictory data with the help of a dependency record. This record reflects the retractions and additions which makes the inference engine (IE) aware of its current belief set.

Each statement having at least one valid justification is made a part of the current belief set. When a contradiction is found, the statement(s) responsible for the contradiction are identified and the records are appropriately updated. This process is called dependency-directed backtracking.

The TMS algorithm maintains the records in the form of a dependency network. Each node in the network is an entry in the KB (a premise, antecedent, or inference rule etc.) Each arc of the network represents the inference steps through which the node was derived.

A premise is a fundamental belief which is assumed to be true. They do not need justifications. The set of premises are the basis from which justifications for all other nodes will be derived.

There are two types of justification for a node. They are:

1. Support list [SL]
2. Conditional proof (CP)

Many kinds of truth maintenance systems exist. Two major types are single-context and multi-context truth maintenance. In single context systems, consistency is maintained among all facts in memory (KB) and relates to the notion of consistency found in classical logic. Multi-context systems support Para consistency by allowing consistency to be relevant to a subset of facts in memory, a context, according to the history of logical inference. This is achieved by tagging each fact or deduction with its logical history. Multi-agent truth maintenance systems perform truth maintenance across multiple memories, often located on different machines. de Kleer's assumption-based truth maintenance system (ATMS, 1986) was utilized in systems based upon KEE on the Lisp Machine. The first multi-agent TMS was created by Mason and Johnson. It was a multi-context system. Bridgeland and Huhns created the first single-context multi-agent system.

## **REPRESENTING KNOWLEDGE UNDER UNCERTAINTY:**

Following are the kind of knowledge which needs to be represented in AI systems:

- **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).

**Knowledge:** Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

## Types of knowledge

Following are the various types of knowledge:



### 1. Declarative Knowledge:

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called descriptive knowledge and expressed in declarative sentences.
- It is simpler than procedural language.

### 2. Procedural Knowledge

- It is also known as imperative knowledge.

- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

### **3. Meta-knowledge:**

- Knowledge about the other types of knowledge is called Meta-knowledge.

### **4. Heuristic knowledge:**

- Heuristic knowledge is representing knowledge of some experts in a field or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

### **5. Structural knowledge:**

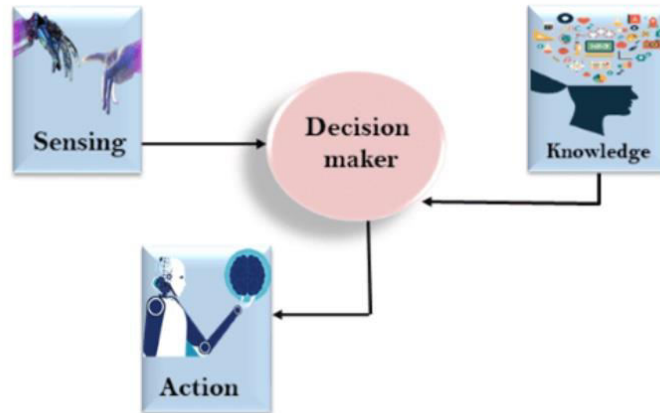
- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

### **The relation between knowledge and intelligence:**

Knowledge of real-worlds plays a vital role in intelligence and same for creating artificial intelligence. Knowledge plays an important role in demonstrating intelligent behavior in AI agents. An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

Let's suppose if you met some person who is speaking in a language which you don't know, then how you will be able to act on that. The same thing applies to the intelligent behavior of the agents.

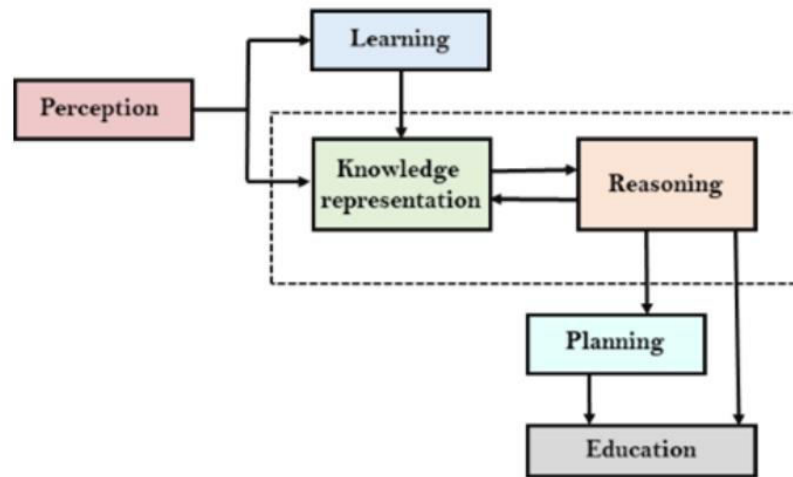
As we can see in below diagram, there is one decision maker which act by sensing the environment and using knowledge. But if the knowledge part will not present then, it cannot display intelligent behavior.



### **AI knowledge cycle:**

An Artificial intelligence system has the following components for displaying intelligent behavior:

- Perception
- Learning
- Knowledge Representation and Reasoning
- Planning
- Execution



The above diagram is showing how an AI system can interact with the real world and what components help it to show intelligence. AI system has Perception component by which it retrieves information from its environment. It can be visual, audio or another form of sensory input. The learning component is responsible for learning from data captured by Perception component. In the complete cycle, the main components are knowledge representation and Reasoning. These two components are involved in showing the intelligence in machine-like humans. These two components are independent with each other but also coupled together. The planning and execution depend on analysis of Knowledge representation and reasoning.

### **Approaches to knowledge representation:**

There are mainly four approaches to knowledge representation, which are given below:

#### **1. Simple relational knowledge:**

- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This approach has little opportunity for inference.



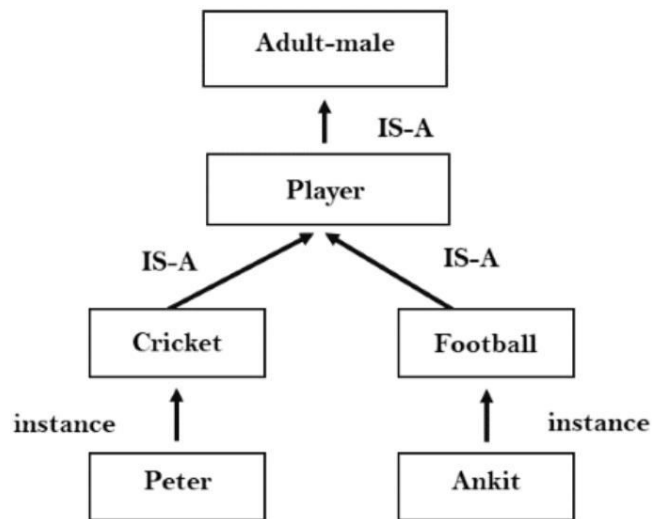
**Example: The following is the simple relational knowledge representation.**

Player	Weight	Age
Player1	65	23
Player2	58	18
Player3	75	24

## **2. Inheritable knowledge:**

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- All classes should be arranged in a generalized form or a hierarchal manner.
- In this approach, we apply inheritance property.
- Elements inherit values from other members of a class.
- This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.
- Every individual frame can represent the collection of attributes and its value.
- In this approach, objects and values are represented in Boxed nodes.
- We use Arrows which point from objects to their values.

### Example:



### 3. Inferential knowledge:

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.

**Example:** Let's suppose there are two statements:

- Marcus is a man
- All men are mortal

Then it can represent as;

**man(Marcus)**

**$\forall x = \text{man}(x) \text{ -----} \rightarrow \text{mortal}(x)$**

### 4. Procedural knowledge:

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.

- In this approach, one important rule is used which is **If-Then rule**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

### **Requirements for knowledge Representation system:**

A good knowledge representation system must possess the following properties.

1. **Representational Accuracy:**  
KR system should have the ability to represent all kind of required knowledge.
2. **Inferential Adequacy:**  
KR system should have ability to manipulate the representational structures to produce new knowledge corresponding to existing structure.
3. **Inferential Efficiency:**  
The ability to direct the inferential knowledge mechanism into the most productive directions by storing appropriate guides.
4. **Acquisitional efficiency-** The ability to acquire the new knowledge easily using automatic methods.

### **BAYESIAN NETWORKS:**

Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as:

"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network, belief network, decision network, or Bayesian model**.

Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

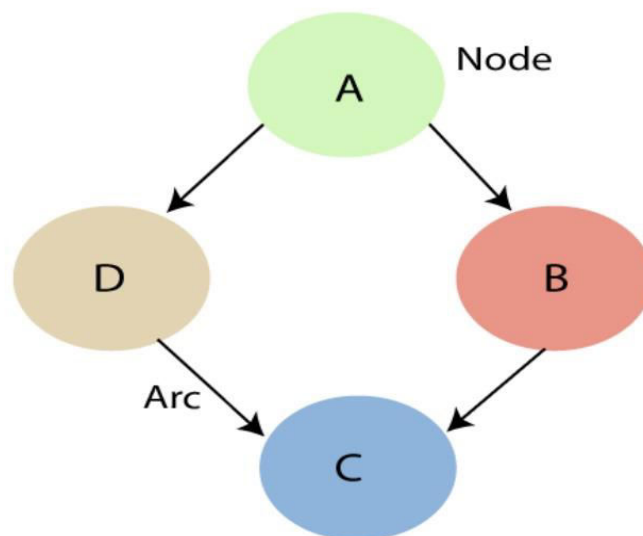
Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction, and decision making under uncertainty.**

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- **Directed Acyclic Graph**
- **Table of conditional probabilities.**

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram**.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:



- Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.

- **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph.  
These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other
  - **In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.**
  - **If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.**
  - **Node C is independent of node A.**

Note: The Bayesian network graph does not contain any cyclic graph. Hence, it is known as a directed acyclic graph or DAG.

The Bayesian network has mainly two components:

- **Causal Component**
- **Actual numbers**

Each node in the Bayesian network has condition probability distribution  $P(X_i | \text{Parent}(X_i))$ , which determines the effect of the parent on that node.

Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

### **Joint probability distribution:**

If we have variables  $x_1, x_2, x_3, \dots, x_n$ , then the probabilities of a different combination of  $x_1, x_2, x_3 \dots x_n$ , are known as Joint probability distribution.

$P[x_1, x_2, x_3, \dots, x_n]$ , it can be written as the following way in terms of the joint probability distribution.

$$= P[x_1 | x_2, x_3, \dots, x_n] P[x_2, x_3, \dots, x_n]$$

$$= P[X_1 | X_2, X_3, \dots, X_n] P[X_2 | X_3, \dots, X_n] \dots P[X_{n-1} | X_n] P[X_n].$$

In general for each variable  $X_i$ , we can write the equation as:

$$P(X_i | X_{i-1}, \dots, X_1) = P(X_i | \text{Parents}(X_i))$$

### **Explanation of Bayesian network:**

Let's understand the Bayesian network through an example by creating a directed acyclic graph:

**Example:** Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

### **Problem:**

**Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.**

### **Solution:**

- The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.
- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- The conditional distributions for each node are given as conditional probabilities table or CPT.

- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
- In CPT, a boolean variable with k boolean parents contains  $2^k$  probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

**List of all events occurring in this network:**

- **Burglary (B)**
- **Earthquake(E)**
- **Alarm(A)**
- **David Calls(D)**
- **Sophia calls(S)**

We can write the events of problem statement in the form of probability: **P[D, S, A, B, E]**, can rewrite the above probability statement using joint probability distribution:

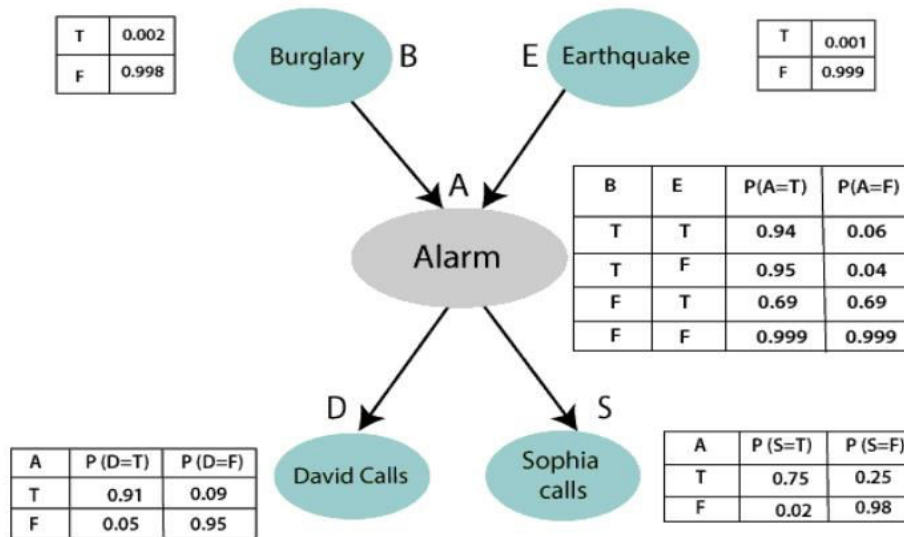
$$P[D, S, A, B, E] = P[D | S, A, B, E] \cdot P[S, A, B, E]$$

$$= P[D | S, A, B, E] \cdot P[S | A, B, E] \cdot P[A, B, E]$$

$$= P[D | A] \cdot P[S | A, B, E] \cdot P[A, B, E]$$

$$= P[D | A] \cdot P[S | A] \cdot P[A | B, E] \cdot P[B, E]$$

$$= P[D | A] \cdot P[S | A] \cdot P[A | B, E] \cdot P[B | E] \cdot P[E]$$



Let's take the observed probability for the Burglary and earthquake component:

$P(B = \text{True}) = 0.002$ , which is the probability of burglary.

$P(B = \text{False}) = 0.998$ , which is the probability of no burglary.

$P(E = \text{True}) = 0.001$ , which is the probability of a minor earthquake

$P(E = \text{False}) = 0.999$ , Which is the probability that an earthquake not occurred.

We can provide the conditional probabilities as per the below tables:

**Conditional probability table for Alarm A:**

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	E	P(A= True)	P(A= False)
True	True	0.94	0.06
True	False	0.95	0.04



False	True	0.31	0.69
False	False	0.001	0.999

### Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

### Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

$$P(S, D, A, \neg B, \neg E) = P(S|A) * P(D|A) * P(A|\neg B \wedge \neg E) * P(\neg B) * P(\neg E).$$

$$= 0.75 * 0.91 * 0.001 * 0.998 * 0.999$$

$$= \mathbf{0.00068045}.$$

**Hence, a Bayesian network can answer any query about the domain by using Joint distribution.**

**The semantics of Bayesian Network:**

There are two ways to understand the semantics of the Bayesian network, which is given below:

**1. To understand the network as the representation of the Joint probability distribution.**

It is helpful to understand how to construct the network.

**2. To understand the network as an encoding of a collection of conditional independence statements.**

It is helpful in designing inference procedure.

## MODULE – IV

### LEARNING

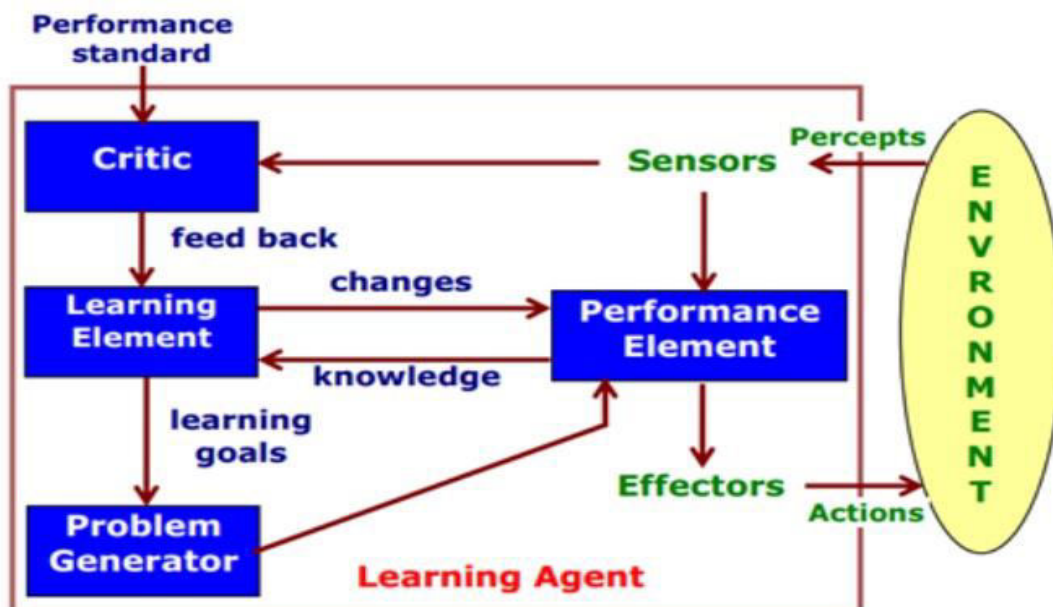
#### WHAT IS LEARNING

“Learning denotes changes in a system that enables system to do the same task more efficiently next time.”

#### Machine Learning:-Definition

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P,if its performance at tasks in T,as measured by P,improves with experience E.

#### Components of Learning System



#### Performance Element:

The performance element is the agent that acts in the world .It perceps and decides on external actions.

### **Learning Element:**

It responsible for making improvements, takes knowledge about performance element and some feedback, determines how to modify performance element.

### **Critic:**

It tells the learning element how agent is doing by comparing with the fixed standard of performance.

### **Problem Generator:**

This component suggests problems or actions that will generate new examples or experience that helps the system to train further.

Let us see the role of each component with an example.

Example: Automated Taxi on city roads

Performance Element:consists of knowledge and procedures for driving actions.

eg:turning ,accelerating,breaking are the performance elements on roads.

Learning Element:It formulates goals.

Eg:learn rules for breaking,accelerating,learn geography of the city.

Critic: Observes world and passes information to learning element.

Eg: quick right turn across three lanes of traffic ,observe reaction of other drivers.

Problem Generator: Try south city road

Learning Paradigm:

- Rote learning
- Induction
- Clustering
- Analogy
- Discovery
- Genetic algorithms
- Reinforcement

## **ROTE LEARNING:**

Rote learning technique avoids understanding the inner complexities but focuses on memorizing the material so that it can be recalled by the learner exactly the way it read or heard.

Learning by memorization: This avoids understanding the inner complexities the subject that is being learned.

Learning something from Repeating: saying the same thing and trying to remember how to say it; it does not help to understand, it helps to remember, like we learn a poem, song, etc.

There are two types of inductive learning,

- Supervised
- Unsupervised

**Supervised learning :( The machine has access to a teacher who corrects it.)**

Learning is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). Example: Face recognition

**Unsupervised Learning :( No access to teacher. Instead, the machine must search for “order” and “structure” in the environment.)**

since there is no desired output in this case that is provided therefore categorization is done so that the algorithm differentiates correctly between the face of a horse, cat or human (clustering of data)

**Clustering:**

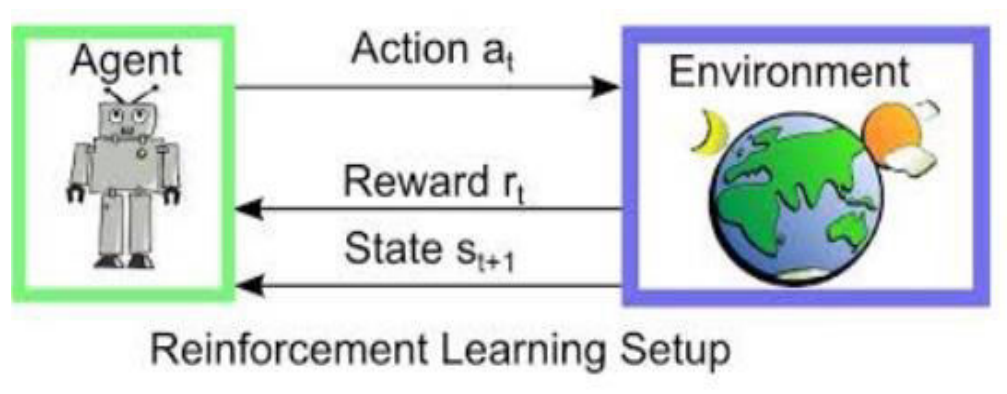
In **clustering** or **unsupervised learning**, the target features are not given in the training examples. The aim is to construct a natural classification that can be used to cluster the data. The general idea behind clustering is to partition the

examples into **clusters** or **classes**. Each class predicts feature values for the examples in the class. Each clustering has a prediction error on the predictions. The best clustering is the one that minimizes the error.

**Example:** An intelligent tutoring system may want to cluster students' learning behaviour so that strategies that work for one member of a class may work for other members.

### **Reinforcement Learning:**

Imagine a robot that can act in a world, receiving rewards and punishments and determining from these it should do. This is the problem of **reinforcement learning**. Most Reinforcement Learning research is conducted with in the mathematical framework of **Markov Decision Process**.



### **LEARNING BY TAKING ADVICE:**

The idea of advice taking in AI based learning was proposed as early as 1958 (McCarthy). However very few attempts were made in creating such systems until the late 1970s. Expert systems providing a major impetus in this area.

- This type is the easiest and simple way of learning.

- In this type of learning, a programmer writes a program to give some instructions to perform a task to the computer. Once it is learned (i.e. programmed), the system will be able to do new things.
- Also, there can be several sources for taking advice such as humans(experts), internet etc.
- However, this type of learning has a more necessity of inference than rote learning.
- As the stored knowledge in knowledge base gets transformed into an operational form, the reliability of the knowledge source is always taken into consideration.

### **LEARNING IN PROBLEM SOLVING:**

- Humans have a tendency to learn by solving various real world problems.
- The forms or representation, or the exact entity, problem solving principle is based on reinforcement learning.
- Therefore, repeating certain action results in desirable outcome while the action is avoided if it results into undesirable outcomes.
- As the outcomes have to be evaluated, this type of learning also involves the definition of a utility function. This function shows how much is a particular outcome worth?
- There are several research issues which include the identification of the learning rate, time and algorithm complexity, convergence, representation (frame and qualification problems), handling of uncertainty (ramification problem), adaptivity and "unlearning" etc.
- In reinforcement learning, the system (and thus the developer) know the desirable outcomes but does not know which actions result into desirable outcomes.
- In such a problem or domain, the effects of performing the actions are usually compounded with side-effects. Thus, it becomes impossible to specify the actions to be performed in accordance to the given parameters.
- Q-Learning is the most widely used reinforcement learning algorithm.

- The main part of an algorithm is a simple value iteration update. For each state 'S', from the state set S, and for each action, a, from the action set 'A', it is possible to calculate an update to its expected reduction reward value, with the following expression:

$$Q(st, at) \leftarrow Q(st, at) + \alpha_t(st, at) [r_t + \gamma \max_a Q(st+1, a) - Q(st, at)]$$

- where  $r_t$  is a real reward at time t,  $\alpha_t(s,a)$  are the learning rates such that  $0 \leq \alpha_t(s,a) \leq 1$ , and  $\gamma$  is the discount factor such that  $0 \leq \gamma < 1$ .

## **LEARNING BY EXAMPLE (INDUCTION LEARNING):**

- Induction learning is carried out on the basis of supervised learning.
- In this learning process, a general rule is induced by the system from a set of observed instance.
- However, class definitions can be constructed with the help of a classification method.

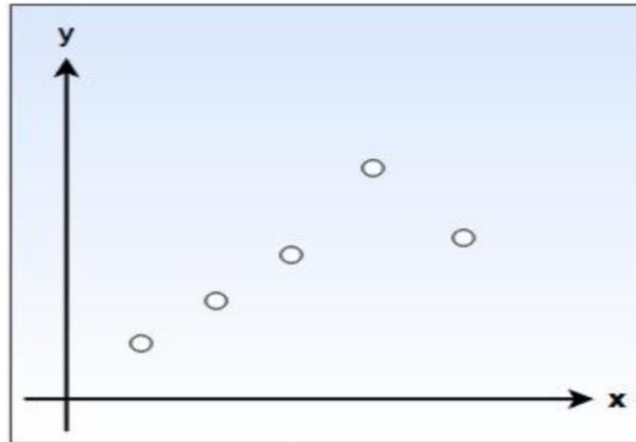
### **For Example:**

Consider that ' $f$ ' is the target function and example is a pair  $(x, f(x))$ , where 'x' is input and  $f(x)$  is the output function applied to 'x'.

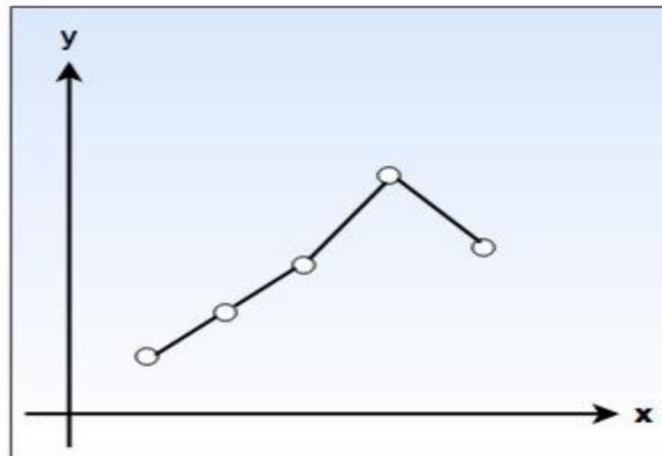
**Given problem:** Find hypothesis  $h$  such as  $h \approx f$

- So, in the following fig-a, points  $(x,y)$  are given in plane so that  $y = f(x)$ , and the task is to find a function  $h(x)$  that fits the point well.



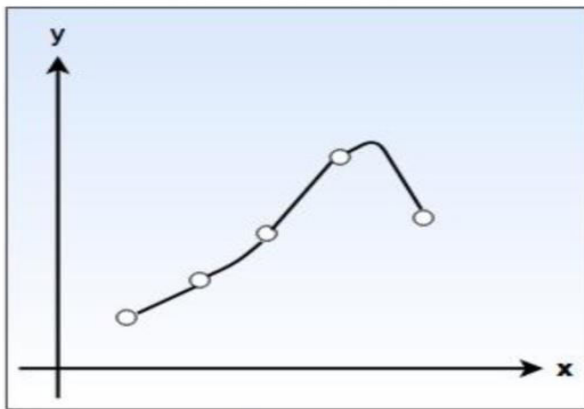


**Fig(a)**



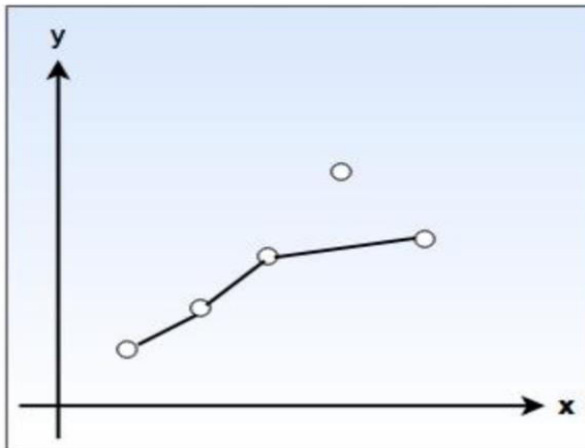
**Fig(b)**

- In fig-b, a piecewise-linear 'h' function is given, while the fig-c shows more complicated 'h' function.



**Fig(c)**

- Both the functions agree with the example points, but differ with the values of 'y' assigned to other x inputs.



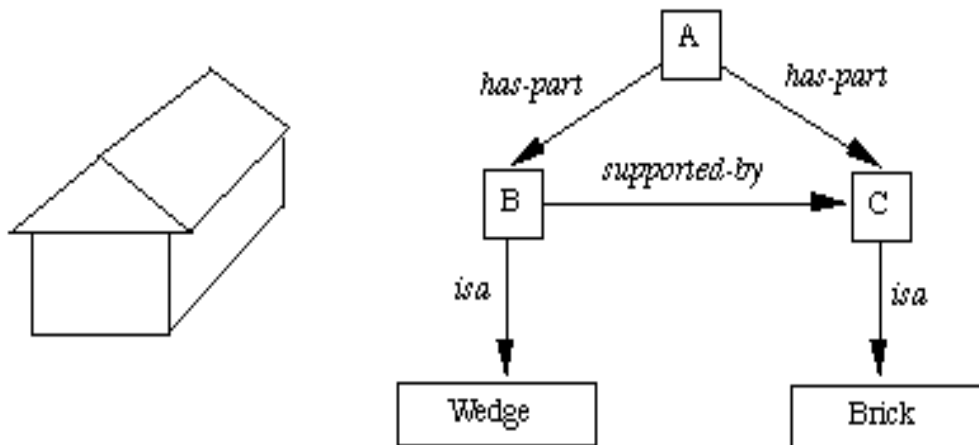
**Fig(d)**

- As shown in fig.(d), we have a function that apparently ignores one of the example points, but fits others with a simple function. The true  $f$  is unknown, so there are many choices for  $h$ , but without further knowledge, we have no way to prefer (b), (c), or (d).

## WINSTON LEARNING PROGRAM:

### A Blocks World Learning Example -- Winston (1975)

- The goal is to construct representation of the definitions of concepts in this domain.
- Concepts such a house - brick (rectangular block) with a wedge (triangular block) suitably placed on top of it, tent - 2 wedges touching side by side, or an arch - two non-touching bricks supporting a third wedge or brick, were learned.
- The idea of near miss objects -- similar to actual instances was introduced.
- Input was a line drawing of a blocks world structure.
- Input processed (see VISION Sections later) to produce a semantic net representation of the structural description of the object (Fig. 27)



**Fig: House object and semantic net**

- Links in network include left-of, right-of, does-not-marry, supported-by, has-part, and isa.
- The marry relation is important -- two objects with a common touching edge are said to marry. Marrying is assumed unless does-not-marry stated.

There are three basic steps to the problem of concept formulation:

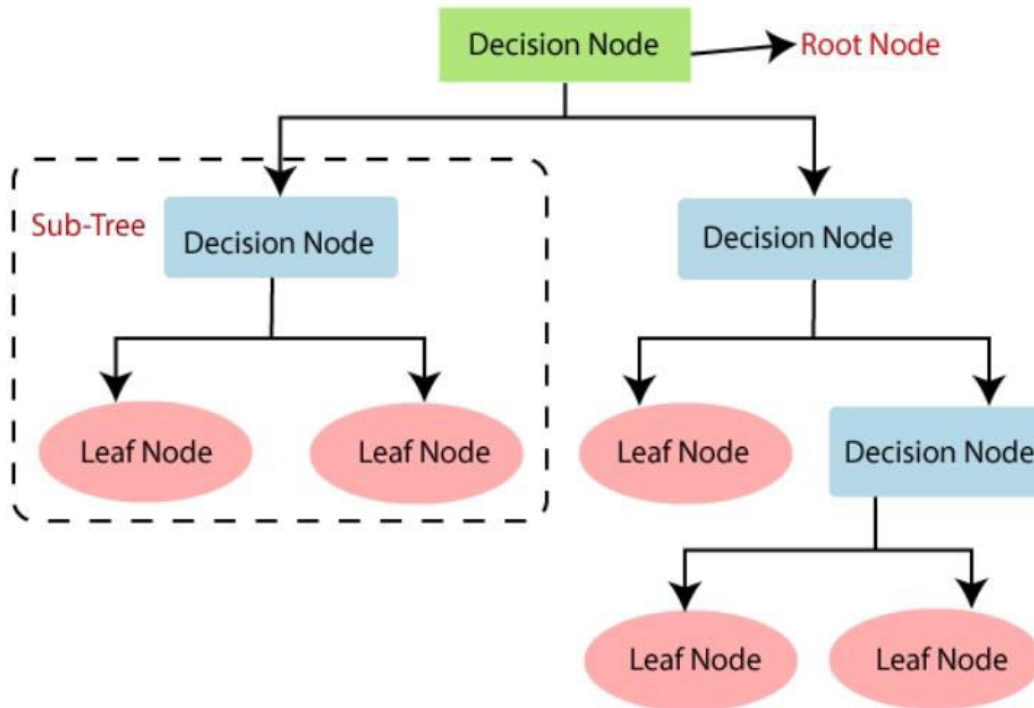
1. Select one known instance of the concept. Call this the concept definition.
2. Examine definitions of other known instances of the concept. Generalise the definition to include them.

3. Examine descriptions of near misses. Restrict the definition to exclude these.

Both steps 2 and 3 rely on comparison and both similarities and differences need to be identified.

## **DECISION TREES:**

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node**. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- **It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.**
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm**, which stands for **Classification and Regression Tree algorithm.**
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- Below diagram explains the general structure of a decision tree:



## Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

## Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

### How does the Decision Tree algorithm Work?

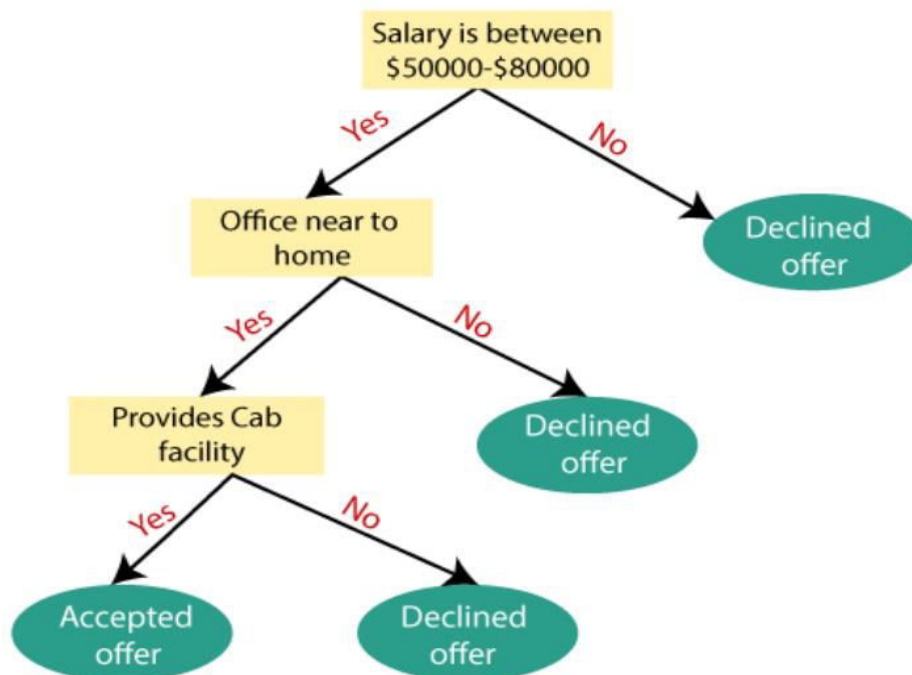
In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM)**.
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

## Example:

Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



## Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM**. By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

## 1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

$$\text{Information Gain} = \text{Entropy}(S) - [(\text{Weighted Avg}) * \text{Entropy}(\text{each feature})]$$

**Entropy:** Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

$$\text{Entropy}(s) = -P(\text{yes}) \log_2 P(\text{yes}) - P(\text{no}) \log_2 P(\text{no})$$

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

## 2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$



## **Pruning:** Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

- **Cost Complexity Pruning**
- **Reduced Error Pruning.**

## **Advantages of the Decision Tree**

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

## **Disadvantages of the Decision Tree**

- The decision tree contains lots of layers, which makes it complex.
- It may have an over fitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

## MODULE – V

### EXPERT SYSTEMS

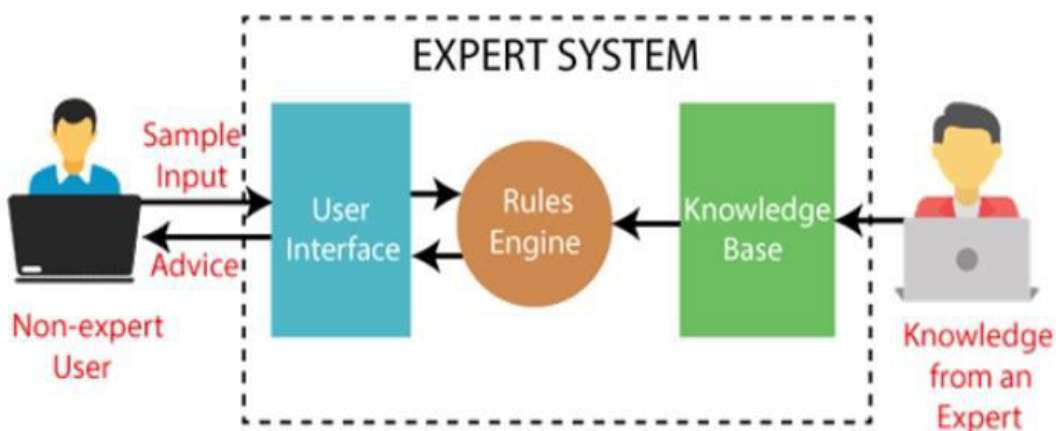
#### **What is an Expert System?**

An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert. It performs this by extracting knowledge from its knowledge base using the reasoning and inference rules according to the user queries.

The expert system is a part of AI, and the first ES was developed in the year 1970, which was the first successful approach of artificial intelligence. It solves the most complex issue as an expert by extracting the knowledge stored in its knowledge base. The system helps in decision making for complex problems using **both facts and heuristics like a human expert**. It is called so because it contains the expert knowledge of a specific domain and can solve any complex problem of that particular domain. These systems are designed for a specific domain, such as **medicine, science**, etc.

The performance of an expert system is based on the expert's knowledge stored in its knowledge base. The more knowledge stored in the KB, the more that system improves its performance. One of the common examples of an ES is a suggestion of spelling errors while typing in the Google search box.

Below is the block diagram that represents the working of an expert system:



**Below are some popular examples of the Expert System:**

- **DENDRAL:** It was an artificial intelligence project that was made as a chemical analysis expert system. It was used in organic chemistry to detect unknown organic molecules with the help of their mass spectra and knowledge base of chemistry.
- **MYCIN:** It was one of the earliest backward chaining expert systems that was designed to find the bacteria causing infections like bacteraemia and meningitis. It was also used for the recommendation of antibiotics and the diagnosis of blood clotting diseases.
- **PXDES:** It is an expert system that is used to determine the type and level of lung cancer. To determine the disease, it takes a picture from the upper body, which looks like the shadow. This shadow identifies the type and degree of harm.
- **CaDeT:** The CaDet expert system is a diagnostic support system that can detect cancer at early stages.

## Characteristics of Expert System

- **High Performance:** The expert system provides high performance for solving any type of complex problem of a specific domain with high efficiency and accuracy.
- **Understandable:** It responds in a way that can be easily understandable by the user. It can take input in human language and provides the output in the same way.
- **Reliable:** It is much reliable for generating an efficient and accurate output.
- **Highly responsive:** ES provides the result for any complex query within a very short period of time.

## KNOWLEDGE BASE (Representing and Using Domain Knowledge)

- Expert system is built around a knowledge base module.
- Expert system contains a formal representation of the information provided by the domain expert. This information may be in the form of problem-solving rules, procedures, or data intrinsic to the domain. To incorporate these information into the system, it is necessary to make use of one or more knowledge representation methods. Some of these methods are described here.
- Transferring knowledge from the human expert to a computer is often the most difficult part of building an expert system.
- The knowledge acquired from the human expert must be encoded in such a way that it remains a faithful representation of what the expert knows, and it can be manipulated by a computer.
- Three common methods of knowledge representation evolved over the years are IF-THEN rules, Semantic networks and Frames.

- The first two methods were illustrated in the earlier lecture slides on knowledge representation therefore just mentioned here. The frame based representation is described more.

## 1. IF-THEN rules

Human experts usually tend to think along :

**condition**  $\Rightarrow$  **action or Situation**  $\Rightarrow$  **conclusion**

Rules "**if-then**" are predominant form of encoding knowledge in expert systems. These are of the form :

If  $a_1, a_2, \dots, a_n$

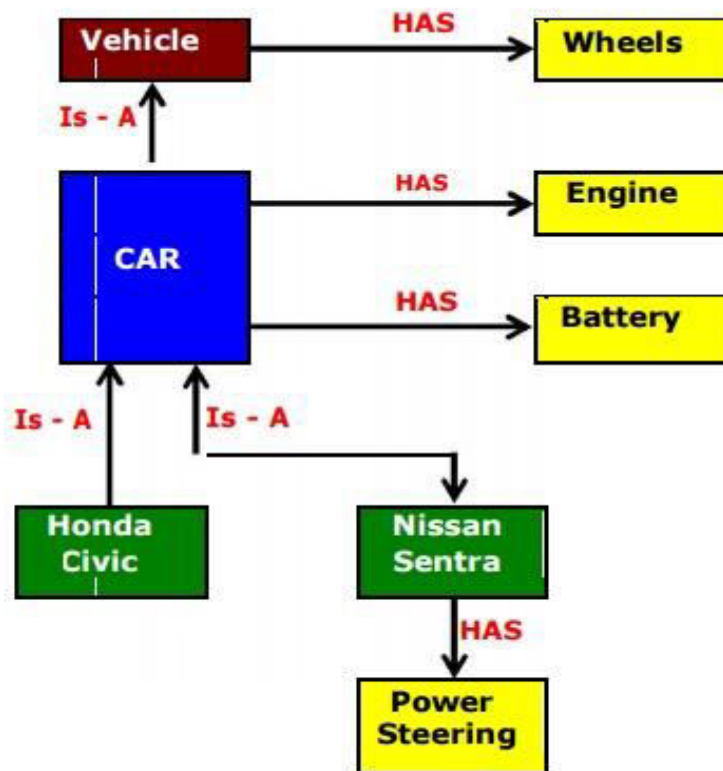
Then  $b_1, b_2, \dots, b_n$  where

each  $a_i$  is a condition or situation, and

each  $b_i$  is an action or a conclusion.

## 2. Semantic Networks

- In this scheme, knowledge is represented in terms of objects and relationships between objects.
- The objects are denoted as nodes of a graph. The relationship between two objects are denoted as a link between the corresponding two nodes.
- The most common form of semantic networks uses the links between nodes to represent **IS-A** and **HAS** relationships between objects.
- **Example of Semantic Network**
- The Fig. below shows a car IS-A vehicle; a vehicle HAS wheels.
- This kind of relationship establishes an inheritance hierarchy in the network, with the objects lower down in the network inheriting properties from the objects higher up.



### 3. Frame

- In this technique, knowledge is decomposed into highly modular pieces called frames, which are generalized record structures. Knowledge consist of concepts, situations, attributes of concepts, relationships between concepts, and procedures to handle relationships as well as attribute values.
- ‡ Each concept may be represented as a separate frame.
- ‡ The attributes, the relationships between concepts, and the procedures are allotted to slots in a frame.
- ‡ The contents of a slot may be of any data type - numbers, strings, functions or procedures and so on.
- ‡ The frames may be linked to other frames, providing the same kind of inheritance as that provided by a semantic network.

- A frame-based representation is ideally suited for objected-oriented programming techniques. An example of Frame-based representation of knowledge is shown in next slide.

**Example : Frame-based Representation of Knowledge.**

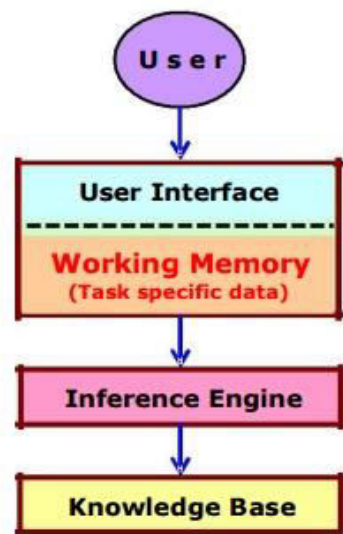
Two frames, their slots and the slots filled with data type are shown.

<b>Frame</b>	<i>Car</i>	<b>Frame</b>	<i>Car</i>
<b>Inheritance Slot</b>	<i>Is-A</i>	<b>Inheritance Slot</b>	<i>Is-A</i>
<b>Value</b>	<i>Vehicle</i>	<b>Value</b>	<i>Car</i>
<b>Attribute Slot</b>	<i>Engine</i>	<b>Attribute Slot</b>	<i>Make</i>
<b>Value</b>	<i>Vehicle</i>	<b>Value</b>	<i>Honda</i>
<b>Value</b>	<i>1</i>	<b>Value</b>	
<b>Value</b>		<b>Value</b>	
<b>Attribute Slot</b>	<i>Cylinders</i>	<b>Attribute Slot</b>	<i>Year</i>
<b>Value</b>	<i>4</i>	<b>Value</b>	<i>1989</i>
<b>Value</b>	<i>6</i>	<b>Value</b>	
<b>Value</b>	<i>8</i>	<b>Value</b>	
<b>Attribute Slot</b>	<i>Doors</i>	<b>Attribute Slot</b>	
<b>Value</b>	<i>2</i>	<b>Value</b>	
<b>Value</b>	<i>5</i>	<b>Value</b>	
<b>Value</b>	<i>4</i>	<b>Value</b>	

**Working Memory**

- Working memory refers to task-specific data for a problem. The contents of the working memory, changes with each problem situation. Consequently, it is the most dynamic component of an expert system, assuming that it is kept current.
- ‡ Every problem in a domain has some unique data associated with it.
- ‡ Data may consist of the set of conditions leading to the problem, its parameters and so on.

- ‡ Data specific to the problem needs to be input by the user at the time of using, means consulting the expert system. The Working memory is related to user interface
- below fig shows how Working memory is closely related to user interface of the expert system.



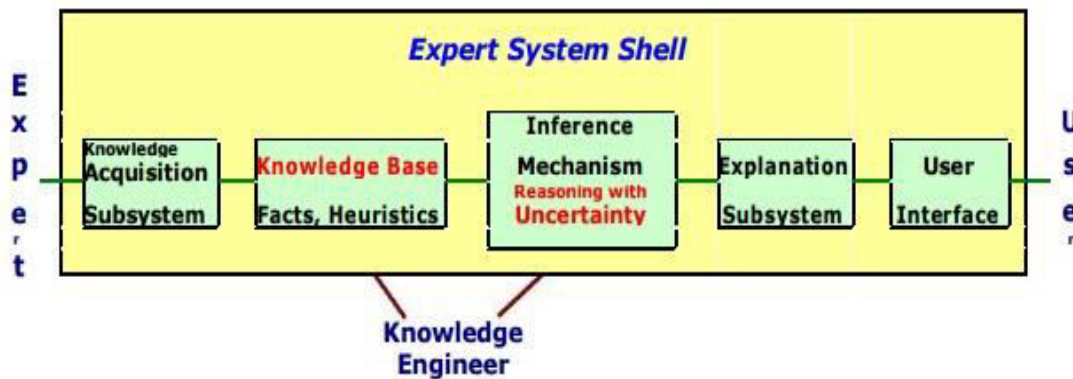
## **EXPERT SYSTEMS SHELLS:**

An Expert system shell is a software development environment. It contains the basic components of expert systems. A shell is associated with a prescribed method for building applications by configuring and instantiating these components.

### **Shell components and description**

The generic components of a shell : the knowledge acquisition, the knowledge Base, the reasoning, the explanation and the user interface are shown below. The knowledge base and reasoning engine are the core components.





### ■ Knowledge Base

A store of factual and heuristic knowledge. Expert system tool provides one or more knowledge representation schemes for expressing knowledge about the application domain. Some tools use both Frames (objects) and IF-THEN rules. In PROLOG the knowledge is represented as logical statements.

### ■ Reasoning Engine

Inference mechanisms for manipulating the symbolic information and knowledge in the knowledge base form a line of reasoning in solving a problem. The inference mechanism can range from simple modus ponens backward chaining of IF-THEN rules to Case-Based reasoning.

### Knowledge Acquisition subsystem

A subsystem to help experts in build knowledge bases. However, collecting knowledge, needed to solve problems and build the knowledge base, is the biggest bottleneck in building expert systems.

## **EXPLANATION SUBSYSTEMS:**

A subsystem that explains the system's actions. The explanation can range from how the final or intermediate solutions were arrived at justifying the need for additional data.

### **User Interface**

A means of communication with the user. The user interface is generally not a part of the expert system technology. It was not given much attention in the past. However, the user interface can make a critical difference in the perceived utility of an Expert system.

### **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 codes of an expert's rules 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.

## **Types of Explanation**

There are four types of explanations commonly used in expert systems.

- Rule trace reports on the progress of a consultation;
- ‡ Explanation of how the system reached to the given conclusion;
- ‡ Explanation of why the system did not give any conclusion.
- Explanation of why the system is asking a question;

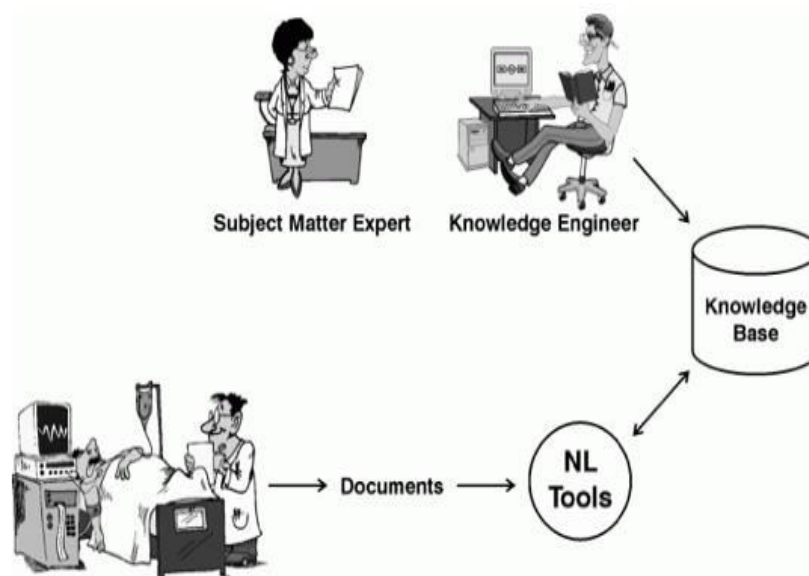
## **KNOWLEDGE ACQUISTION:**

- The success of knowledge based systems lies in the quality and extent of the knowledge available to the system. Acquiring and validating a large groups of consistent, correlated knowledge is not a trivial problem.

- This has given the acquisition process an especially important role in the design and implementation of these systems. Consequently, effective acquisition methods have become one of the principal challenges for the AI researches.
- 
- The goals of this branch of AI are the discovery and development of efficient, cost effective methods of acquisition. Some important progress has recently been made in this area with the development of sophisticated editors and some general concepts related to acquisition and learning.

## Definition

Knowledge acquisition is the process of adding new knowledge to a knowledge base and refining or otherwise improving knowledge that was previously acquired. Acquisition is usually associated with some purpose such as expanding the capabilities of a system or improving its performance at some specified task. It is goal oriented creation and refinement of knowledge . It may consist of facts, rules , concepts, procedures, heuristics, formulas, relationships, statistics or other useful information. Sources of this knowledge may include one or more of the following.



Experts in the domain of interest

- Text Books
- Technical papers
- Databases
- Reports
- The environment

To be effective, the newly acquired knowledge should be integrated with existing knowledge in some meaningful way so that nontrivial inferences can be drawn from the resultant body of knowledge . the knowledge should, of course, be accurate, non redundant, consistent(non contradictory ), and fairly complete in the sense that it is possible to reliably reason about many of the important conclusions for which the systems was intended.

### **Types of learning**

Classification or taxonomy of learning types serves as a guide in studying or comparing differences among them. One can develop learning taxonomies based on the type of knowledge representation used (predicate calculus , rules, frames), the type of knowledge learned (concepts, game playing, problem solving), or by the area of application(medical diagnosis , scheduling , prediction and so on).

The classification is intuitively more appealing and is one which has become popular among machine learning researchers. it is independent of the knowledge domain and the representation scheme is used. It is based on the type of inference strategy employed or the methods used in the learning process. The five different learning methods under this taxonomy are:

- Memorization (rote learning)
- Direct instruction(by being told)
- Analogy

- Induction
- Deduction

Learning by memorization is the simplest form of learning. It requires the least amount of inference and is accomplished by simply copying the knowledge in the same form that it will be used directly into the knowledge base. We use this type of learning when we memorize multiplication tables ,

### **Example**

- A slightly more complex form of learning is by direct instruction. This type of learning requires more understanding and inference than rote learning since the knowledge must be transformed into an operational form before being integrated into the knowledge base. We use this type of learning when a teacher presents a number of facts directly to us in a well organized manner.
- The third type listed, analogical learning, is the process of learning a new concept or solution through the use of similar known concepts or solutions. We use this type of learning when solving problems on an examination where previously learned examples serve as a guide or when we learn to drive a truck using our knowledge of car driving. We make frequent use of analogical learning. This form of learning requires still more inferring than either of the previous forms, since difficult transformations must be made between the known and unknown situations. This is a kind of application of knowledge in a new situation.
- The fourth type of learning is also one that is used frequently by humans. It is a powerful form of learning which, like analogical learning, also requires more inferring than the first two methods. This form of learning requires the use of inductive inference, a form of invalid but useful inference. We use inductive learning when we formulate a general concept after seeing a number of instances or examples of the concept. For example, we learn the concepts of color and sweet taste after experiencing the sensation associated with several examples of colored objects or sweet foods.
- The final type of acquisition is deductive learning. It is accomplished through a sequence of deductive inference steps using known facts. From the known facts, new facts or relationships are logically derived.

- Deductive learning usually requires more inference than the other methods. The inference method used is, of course, a deductive type, which is a valid form of inference.
- In addition to the above classification, we will sometimes refer to learning methods as weak methods or knowledge-rich methods. Weak methods are general purpose methods in which little or no initial knowledge is available. These methods are more mechanical than the classical AI knowledge – rich methods. They often rely on a form of heuristics search in the learning process.