

# Artificial Intelligence (CSE)

## UNIT-I

### 1.1 Introduction

#### **Formal Definition of AI:**

AI is a branch of computer science which is concerned with the study and creation of computer systems that exhibit

some form of intelligence

OR

those characteristics which we associate with intelligence in human behavior

AI is a broad area consisting of different fields, from machine vision, expert systems to the creation of machines that can "think". In order to classify machines as "thinking", it is necessary to define intelligence.

#### **Intelligence:**

Intelligence is a property of mind that encompasses many related mental abilities, such as the capabilities to

- reason
- plan
- solve problems
- think abstractly
- comprehend ideas and language and
- learn

#### **Intelligent Systems:**

An intelligent system is a system that can imitate, automate some intelligent behaviors of human being. Expert systems, intelligent agents and knowledge-based systems are examples of intelligent systems. Intelligent systems perform search and optimization along with learning capabilities. So they are technologically advanced machines that perceive and respond to the world around them. The field of intelligent systems also focuses on how these systems interact with human users in changing and dynamic physical and social environments.

#### **Categories of AI System**

- Systems that think like humans
- Systems that act like humans
- Systems that think rationally
- Systems that act rationally

#### **Systems that think like humans**

Most of the time it is a black box where we are not clear about our thought process. One has to know functioning of brain and its mechanism for possessing information. It is an area of cognitive science. The stimuli are converted into mental representation. Cognitive processes

manipulate representation to build new representations that are used to generate actions. Neural network is a computing model for processing information similar to brain.

### **Systems that act like humans**

The overall behavior of the system should be human like. It could be achieved by observation

### **Systems that think rationally**

Such systems rely on logic rather than human to measure correctness. For thinking rationally or logically, logic formulas and theories are used for synthesizing outcomes.

For example,

given John is a human and all humans are mortal then one can conclude logically that John is mortal

Not all intelligent behavior are mediated by logical deliberation.

### **Systems that act rationally**

Rational behavior means doing right thing. Even if method is illogical, the observed behavior must be rational.

## **Foundations of AI**

Foundation of AI is based on

- Philosophy
- Mathematics
- Economics
- Neuroscience
- Control Theory
- Linguistics
- Computer Engineering
- Psychology

### **Philosophy:**

- Can formal rules be used to draw valid conclusions?
- How does the mind arise from a physical brain?
- Where does knowledge come from?
- How does knowledge lead to action?

### **Mathematics:**

- More formal logical methods
  - Boolean logic
  - Fuzzy logic
- Uncertainty
  - The basis for most modern approaches to handle uncertainty in AI applications can be handled by Probability theory, modal and temporal logics

### **Economics:**

- How should we make decisions so as to maximize payoff?
- How should we do this when others may not go along?
- How should we do this when the payoff may be far in the future?

### **Neuroscience:**

- How do the brain works?
  - Early studies (1824) relied on injured and abnormal people to understand what parts of brain work
  - More recent studies use accurate sensors to correlate brain activity to human thought
    - By monitoring individual neurons, monkeys can now control a computer mouse using thought alone
  - How close are we to have a mechanical brain?
    - Parallel computation, remapping, interconnections,....

### **Control Theory:**

- Machines can modify their behavior in response to the environment (sense/action loop)
  - Water-flow regulator, steam engine governor, thermostat
- The theory of stable feedback systems (1894)
  - Build systems that transition from initial state to goal state with minimum energy
  - In 1950, control theory could only describe linear systems and AI largely rose as a response to this shortcoming

### **Linguistics:**

- How does language relate to thought?
- Speech demonstrates so much of human intelligence
  - Analysis of human language reveals thought taking place in ways not understood in other settings
    - Children can create sentences they have never heard before
- Language and thought are believed to be tightly intertwined

### **Computer Engineering:**

- How can we build an efficient computer ?

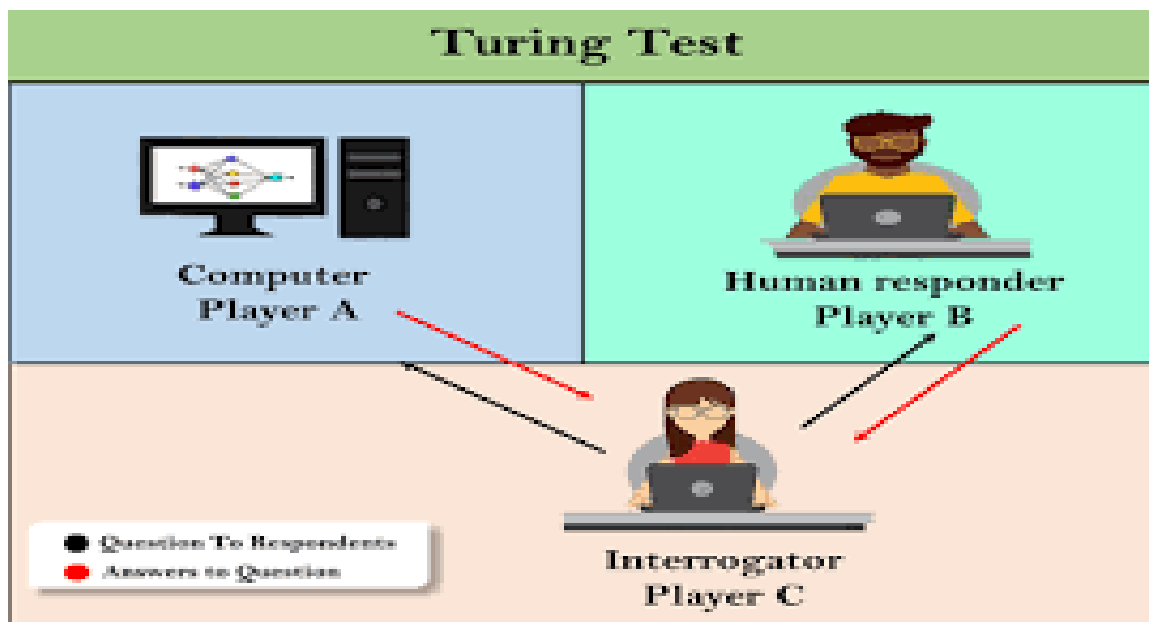
### **Psychology:**

- How do humans and animals think and act ?

# History of AI

## Maturation / Gestation of Artificial Intelligence (1943-1952):

- **Year 1943:** The first work which is now recognized as AI was done by Warren McCulloch and Walter Pitts in 1943. They proposed a model of **artificial neurons**.
- **Year 1949:** Donald Hebb demonstrated an updating rule for modifying the connection strength between neurons. His rule is now called **Hebbian learning**.
- **Year 1950:** The Alan Turing who was an English mathematician and pioneered Machine learning in 1950. Alan Turing publishes "**Computing Machinery and Intelligence**" in which he proposed a test. The test can check the machine's ability to exhibit intelligent behavior equivalent to human intelligence, called a **Turing test**.



## The birth of Artificial Intelligence (1952-1956):

- **Year 1955:** Allen Newell and Herbert A. Simon created the "first artificial intelligence program which was named as "Logic Theorist". This program had proved 38 of 52 Mathematics theorems, and find new and more elegant proofs for some theorems.
- **Year 1956:** The word "Artificial Intelligence" first adopted by American Computer scientist John McCarthy at the Dartmouth Conference. For the first time, AI coined as an academic field.
- At that time high-level computer languages such as FORTRAN, LISP, or COBOL were invented. And the enthusiasm for AI was very high at that time.

## The Golden years-Early enthusiasm (1956-1974):

- **Year 1966:** The researchers emphasized developing algorithms which can solve mathematical problems. Joseph Weizenbaum created the first chatbot in 1966, which was named as ELIZA.



- Year 1972: The first intelligent humanoid robot was built in Japan which was named as WABOT-1.

### **The first AI winter (1974-1980):**

- The duration between years 1974 to 1980 was the first AI winter duration. AI winter refers to the time period where computer scientist dealt with a severe shortage of funding from government for AI researches.
- During AI winters, an interest of publicity on artificial intelligence was decreased.

### **A boom of AI (1980-1987):**

- **Year 1980:** After AI winter duration, AI came back with "Expert System". Expert systems were programmed that emulate the decision-making ability of a human expert.
- In the Year 1980, the first national conference of the American Association of Artificial Intelligence **was held at Stanford University**.

### **The second AI winter (1987-1993)**

- The duration between the years 1987 to 1993 was the second AI Winter duration.
- Again Investors and government stopped in funding for AI research as due to high cost but not efficient result. The expert system such as XCON was very cost effective.

### **The emergence of intelligent agents (1993-2011)**

- **Year 1997:** In the year 1997, IBM Deep Blue beats world chess champion, Gary Kasparov, and became the first computer to beat a world chess champion.
- **Year 2002:** for the first time, AI entered the home in the form of Roomba, a vacuum cleaner.
- **Year 2006:** AI came in the Business world till the year 2006. Companies like Facebook, Twitter and Netflix also started using AI.

### **Deep learning, big data and artificial general intelligence (2011-present):**

- **Year 2011:** In the year 2011, IBM's Watson won jeopardy, a quiz show, where it had to solve the complex questions as well as riddles. Watson had proved that it could understand natural language and can solve tricky questions quickly.
- **Year 2012:** Google has launched an Android app feature "Google now", which was able to provide information to the user as a prediction.
- **Year 2014:** In the year 2014, Chatbot "Eugene Goostman" won a competition in the infamous "Turing test."

## **Sub-areas of AI**

Artificial Intelligence is having various sub fields in its domain. All the Sub-Fields can be distinguished as per various techniques :

### **Neural Networks**

- Neural Networks are inspired by human brains and copies the working process of **human brains**. It is based on a collection of connected units or nodes called **artificial neurons** or **perceptrons**.

- The Objective of this approach was to solve the problems in the same way that a human brain does.

## Vision

- In Artificial Intelligence Vision (Visioning Applications) means processing any image/video sources to extract meaningful information and take action based on that.
- In this field of artificial Intelligence we have also developed such kind of robots which are acquiring human activities within some days or sometimes some hours and train themselves . For e.g. object recognition, image understanding , Playing Robots etc.

## Machine Learning

- The capability of Artificial Intelligence systems to learn by extracting patterns from data is known as **Machine Learning**.
- It is an approach or subset of Artificial Intelligence that is based on the idea that machines can be given access to data along with the ability to learn from it.

## Speech Processing / Speech Recognition

- Speech Processing / Recognition is the ability of a computer and a program to identify words and phrases in the spoken language and convert them to machine readable format.
- The real life examples of Speech processing are Google Assistant ,Amazon Alexa and Apple's Siri Application etc.

## Robotics

- Robots are the **artificial agents** which behaves like human and build for the purpose of manipulating the objects by perceiving, picking, moving, modifying the physical properties of object, or to have an effect thereby freeing manpower from doing repetitive functions without getting bored, distracted, or exhausted.

## Applications

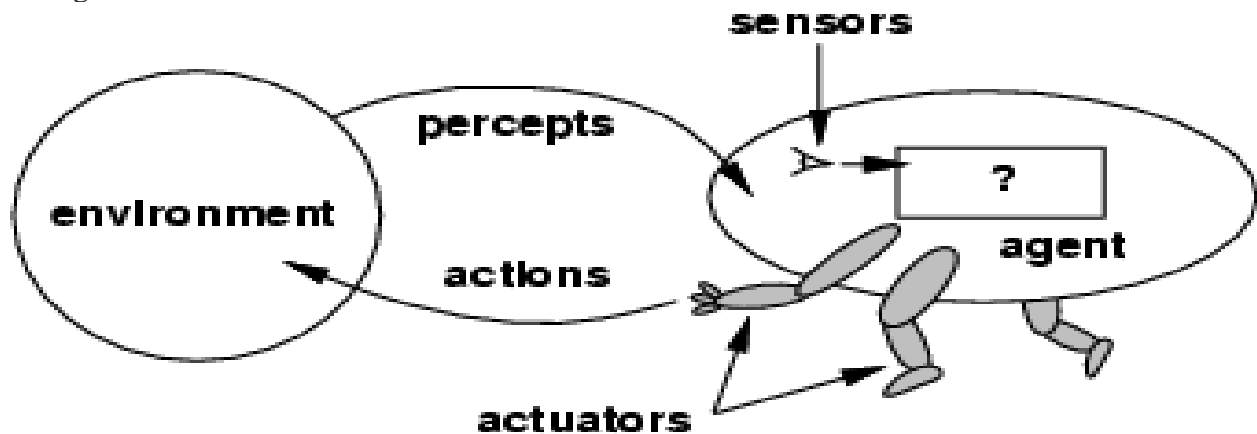
Some of the applications are given below:

- **Business** : Financial strategies, give advice
- **Engineering**: check design, offer suggestions to create new product
- **Manufacturing**: Assembly, inspection & maintenance
- **Mining**: used when conditions are dangerous
- **Hospital** : monitoring, diagnosing & prescribing
- **Education** : In teaching
- **Household** : Advice on cooking, shopping etc.
- **Farming** : prune trees & selectively harvest mixed crops.

## Structure of Agents

### Agents

**Definition:** An agent perceives its environment via sensors and acts upon that environment through its actuators



agent = architecture + program

Example: Vacuum Cleaner World

**iRobot Roomba® 400**  
Vacuum Cleaning Robot



**iRobot Corporation**

**Founder Rodney Brooks (MIT)**

- Powerful suction and rotating brushes
- Automatically navigates for best cleaning coverage
- Cleans under and around furniture, into corners and along wall edges
- Self-adjusts from carpets to hard floors and back again
- Automatically avoids stairs, drop-offs and off-limit areas
- Simple to use— just press the Clean button and Roomba does the rest

### Rational Agents:

An agent should strive to "do the right thing", based on what:

- it can perceive and
- the actions it can perform.

The right action is the one that will cause the agent to be most successful

### Definition:

For each possible percept sequence, a rational agent should select **an action that maximizes its performance measure (in expectation)** given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

### Fully observable / Partially observable Environments

- If an agent's sensors give it access to the complete state of the environment needed to choose an action, the environment is fully observable.
  - (e.g. Chess)
- An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data
  - (e.g. Kriegspiel chess )

## Characterizing a Task Environment

**PEAS:** Performance measure, Environment, Actuators, Sensors

**Example:** the task of designing a self-driving car

- **Performance measure** Safe, fast, legal, comfortable trip
- **Environment** Roads, other traffic, pedestrians
- **Actuators** Steering wheel, accelerator, brake, signal, horn
- **Sensors** Cameras, LIDAR (light/radar), speedometer, GPS, odometer, engine sensors, keyboard

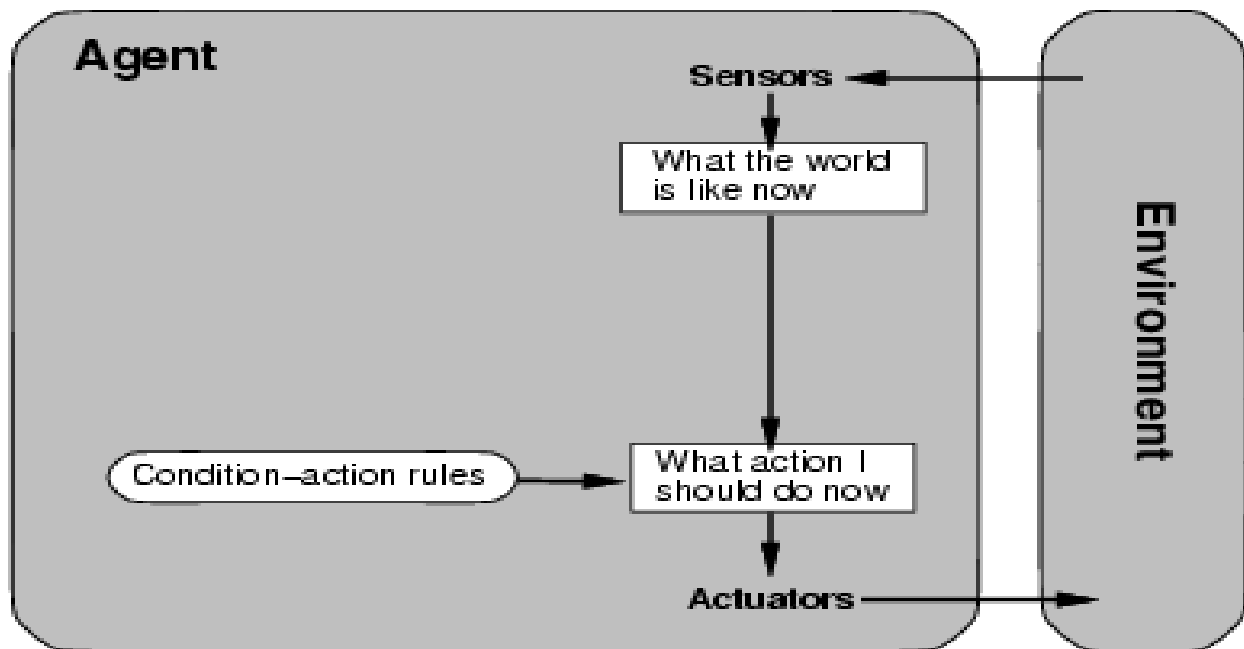
## Types of Agents

1. Simple reflex agents
2. Model based reflex agents
3. Goal based agents
4. Utility based agents
5. Learning agents

### 1. Simple reflex agents:

- Agents do not have memory of past world states or percepts. So, actions depend solely on current percept.
- In this agent Action becomes a “reflex.”
- So it uses condition-action rules.
- Agent selects actions on the basis of *current* percept only.

*Ex: If tail-light of car in front is red, then brake.*



Schematic diagram of Simple reflex agent

**function SIMPLE-REFLEX-AGENT(percept ) returns an action**

**persistent:** rules, a set of condition-action rules

state ← INTERPRET-INPUT(percept )

rule ← RULE-MATCH(state, rules)

action ← rule.ACTION

**return action**

**Example:** The agent program for a simple reflex agent in the two-state vacuum environment.

**function REFLEX-VACUUM-AGENT([location,status]) returns 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 agents

They use a model of the world to choose their actions. They maintain an internal state.

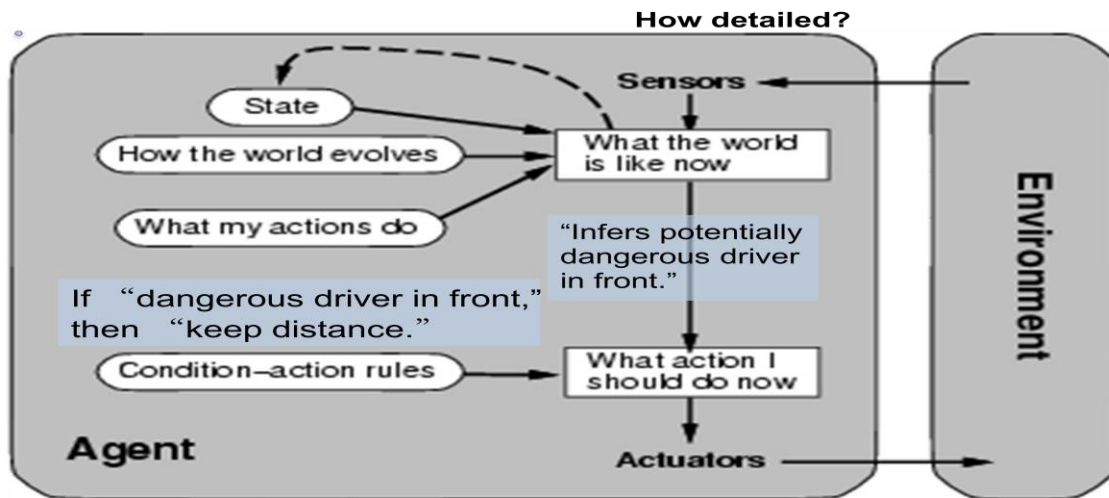
**Model** – knowledge about “how the things happen in the world”.

**Internal State** – It is a representation of unobserved aspects of current state depending on percept history.

**Updating the state requires the information about –**

- How the world evolves.
- How the agent’s actions affect the world.

## Model-based reflex agents



Schematic diagram of Model based reflex agent

**function MODEL-BASED-REFLEX-AGENT(percept ) returns an action persistent:**

state, the agent's current conception of the world state

model , a description of how the next state depends on current state and action

rules, a set of condition-action rules

action, the most recent action, initially none

state ← UPDATE-STATE(state, action, percept ,model )

rule ← RULE-MATCH(state, rules)

action ← rule.ACTION

**return action**

### An example: Brooks' Subsumption Architecture

- Main idea: build complex, intelligent robots by decomposing behaviors into a hierarchy of skills, each defining a percept-action cycle for one very specific task.
- Examples: collision avoidance, exploring, recognizing doorways, etc.
- Each behavior is modeled by a finite-state machine with a few states (though each state may correspond to a complex function or module; provides internal state to the agent).
- Behaviors are loosely coupled via asynchronous interactions.

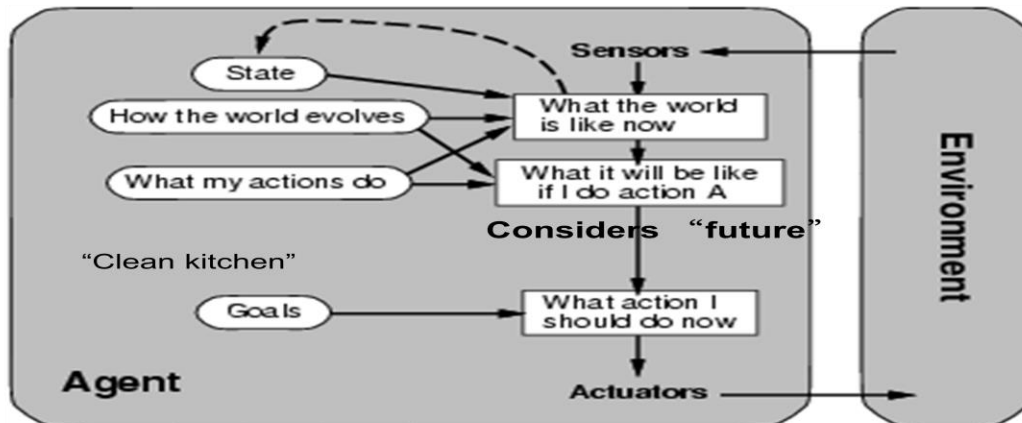
### 3. Goal-based agents

They choose their actions in order to achieve goals. Goal-based approach is more flexible than reflex agent since the knowledge supporting a decision is explicitly modeled, thereby allowing for modifications.

**Goal** – It is the description of desirable situations.

**Module:  
Problem Solving**

**Goal-based agents**



Agent keeps track of the world state as well as set of goals it's trying to achieve: chooses actions that will (eventually) lead to the goal(s).  
More flexible than reflex agents → may involve search and planning

**4. Utility-based agents**

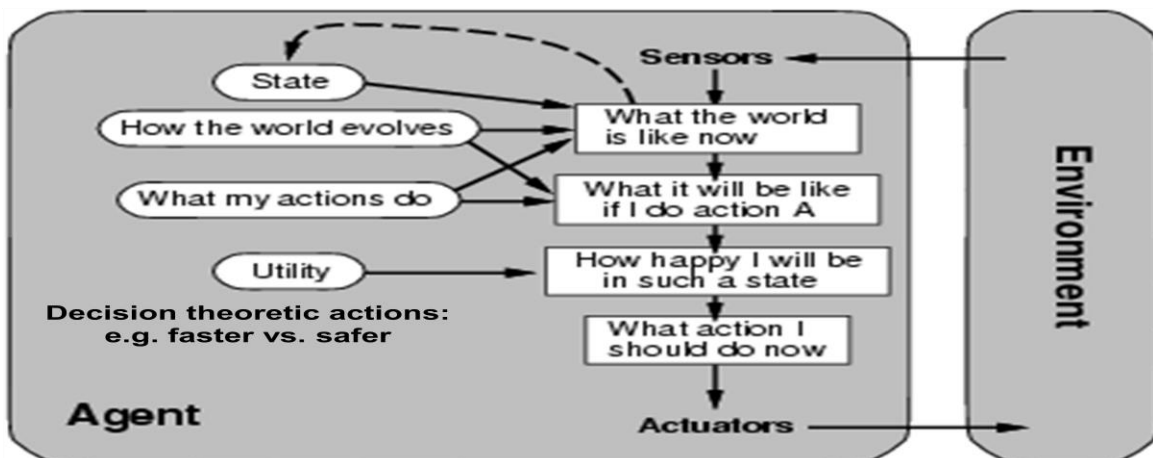
They choose actions based on a preference (utility) for each state.

Goals are inadequate when –

- There are conflicting goals, out of which only few can be achieved.
- Goals have some uncertainty of being achieved and you need to weigh likelihood of success against the importance of a goal.

**Module:  
Decision Making**

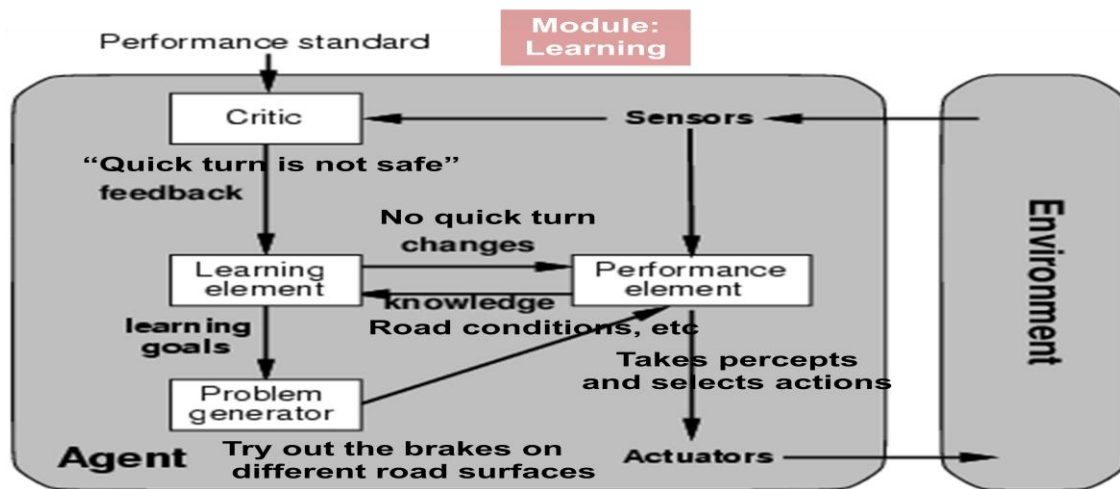
**Utility-based agents**



**5. Learning agents**

- Learning agents are such agents which adapts and improve over time.
- More complicated when agent needs to learn utility information: Reinforcement learning

## Learning agents



### Overview of Structure of Agents:

An agent perceives and acts in an environment, has an architecture, and is implemented by an agent program.

A rational agent always chooses the action which maximizes its expected performance, given its percept sequence so far.

An autonomous agent uses its own experience rather than built-in knowledge of the environment by the designer.

An agent program maps from percept to action and updates its internal state.

- **Simple reflex agents**  
are based on condition-action rules, implemented with an appropriate production system. They are stateless devices which do not have memory of past world states.
- **Agents with memory - Model-based reflex agents**  
have internal state, which is used to keep track of past states of the world.
- **Agents with goals – Goal-based agents**  
are agents that, in addition to state information, have goal information that describes desirable situations. Agents of this kind take future events into consideration.
- **Utility-based agents**  
base their decisions on classic axiomatic utility theory in order to act rationally.
- **Learning agents**  
they have the ability to improve performance through learning.

Representing knowledge is important for successful agent design.



## 1.2 Problem Solving

### Well defined Problems and Solutions:

A problem can be defined formally by five components:

#### Initial State:

The state where agents starts to perform the search to reach the goal state

#### Actions:

Set of applicable actions perform in state S.

#### Transition model:

What each actions does

#### Goal Test:

Which determines whether a given state is a goal state or not.

#### Path Cost:

Assigns a numeric cost to each path

### State Space

- Together the initial state, actions and transition model implicitly define the state space of the problem – the set of all states reachable from the initial state by any sequence of actions.
- The state space forms a directed network or graph in which the nodes are states and the links between nodes are actions.
- A path in the state space is a sequence of states connected by a sequence of actions.

### Example: Travelling in Romania

#### Scenario

- On holiday in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest

**Formulate Goal :** Be in Bucharest

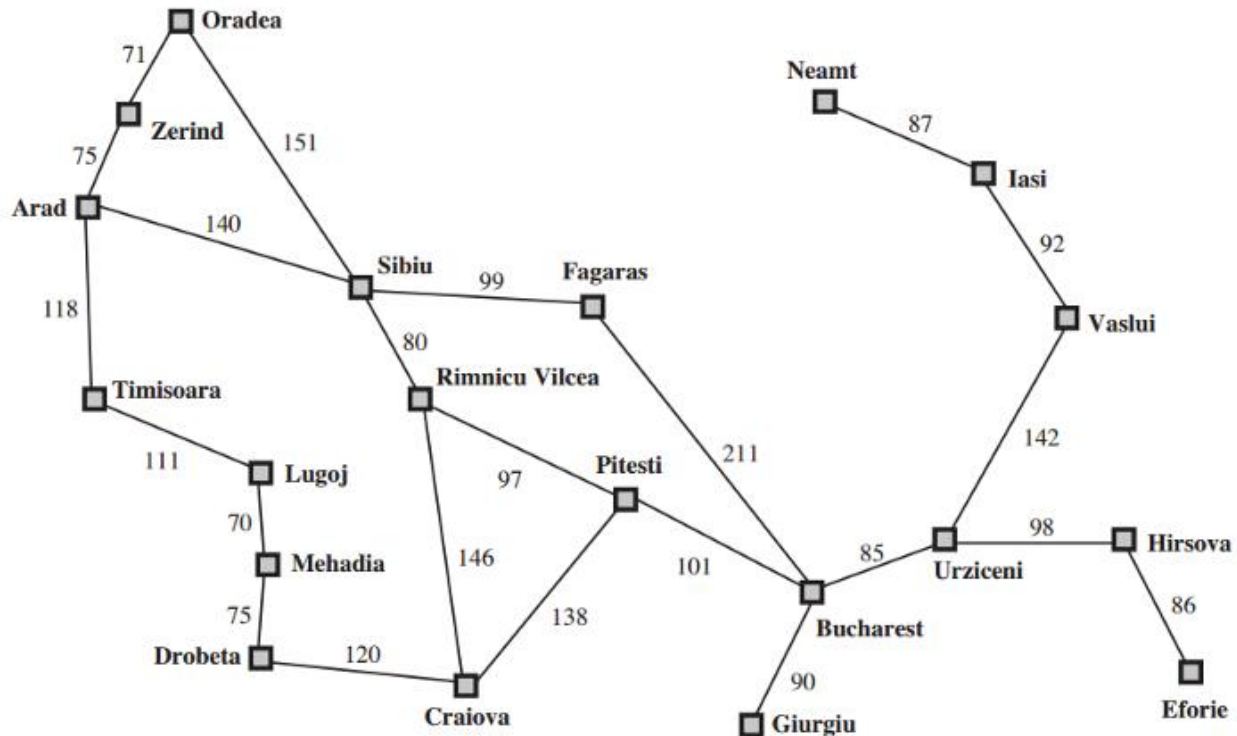
#### Formulate problem

**States:** various cities

**Actions:** drive between cities

**Find Solution:** Appropriate sequence of cities

e.g.: Arad, Sibiu, Fagaras, Bucharest



## Formulating Problems:

- We proposed a formulation of the problem of getting to Bucharest in terms of the initial state, actions, transition model, goal test and path cost.
- This formulation seems reasonable, but it is still a model—an abstract mathematical description.
- We left out many things while travelling like the travelling companions, the current radio program, the scenery out of the window, the proximity of law enforcement officers, the distance to the next rest stop, the condition of the road, the weather and so on.
- These details are irrelevant to the problem of finding a route to Bucharest. The process of removing detail from a representation is called abstraction.

## Problem types

1. Single –state problem
2. Multiple –state problem
3. Contingency problem.

### Single –state problem

- observable (at least initial state)
- deterministic
- static
- discrete

### Multiple –state problem

- partially observable (initial state not observable)
- deterministic

- static
- discrete

### Contingency problem

- partially observable (initial state not observable)
- non-deterministic

## Single-state problem formulation

Defined by the following four items

### 1. Initial state

**Example:**     *Arad*

### 2. Successor function     $S$

**Example:**      $S(Arad) = \{ (goZerind, Zerind), (goSibiu, Sibiu), \dots \}$

### 3. Goal test

**Example:**      $x = Bucharest$      (explicit test)

*noDirt(x)*     (implicit test)

### 4. Path cost     (optional)

**Example:**     sum of distances, number of operators executed, etc.

Solution :

A sequence of operators leading from the initial state to a goal state

## Selecting a state space

### Abstraction

Real world is absurdly complex

State space must be abstracted for problem solving

**(Abstract) state**     Set of real states

### **(Abstract) operator**

Complex combination of real actions

Example: *Arad*  $\rightarrow$  *Zerind* represents complex set of possible routes

**(Abstract) solution**

Set of real paths that are solutions in the real world

**Problems on State Space representation**

**Example: The 8-puzzle**

- It is also called as **N puzzle problem** or **sliding puzzle problem**.
- **N-puzzle** that consists of N tiles (N+1 tiles with an empty tile) where N can be 8, 15, 24 and so on.
- In our example **N = 8**. (that is **square root of (8+1) = 3 rows and 3 columns**).
- In the same way, if we have N = 15, 24 in this way, then they have Row and columns as follow (**square root of (N+1) rows and square root of (N+1) columns**).
- That is if **N=15** than number of rows and columns= 4, and if **N= 24** number of rows and columns= 5.
- So, basically in these types of problems we have given a **initial state or initial configuration (Start state)** and a **Goal state or Goal Configuration**.
- Here we are solving a problem of **8 puzzle** that is a **3x3 matrix**.

**Start State**

7	2	4
5		6
8	3	1

**Goal State**

1	2	3
4	5	6
7	8	

**States:** Integer location of tiles

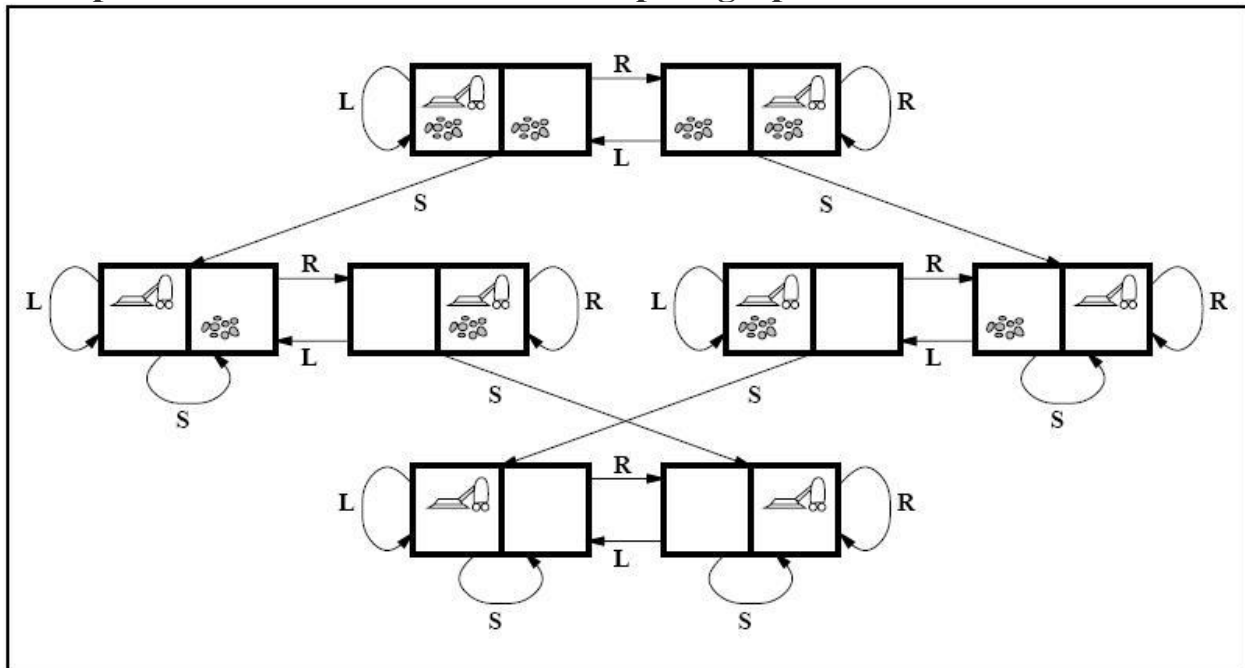
**Actions:** left, right, up, down

**Goal Test:** = goal state?

**Path Cost:** 1 per move

**8 Puzzle has  $9! / 2 = 181440$  states**

### Example: Vacuum-cleaner world state space graph



**States:** Integer dirt and robot locations

**Actions:** left, right, suck, noOp

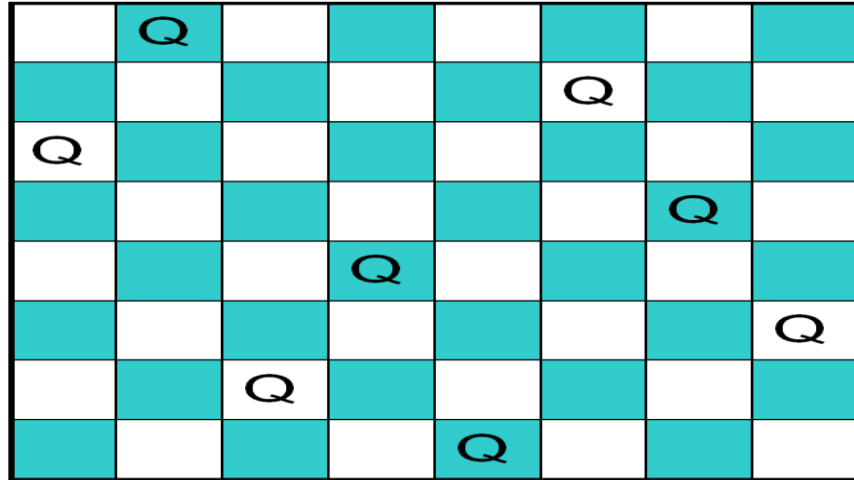
**Goal Test:** not dirty?

**Path Cost:** 1 per operation (0 for noOp)

### Example: Eight Queens

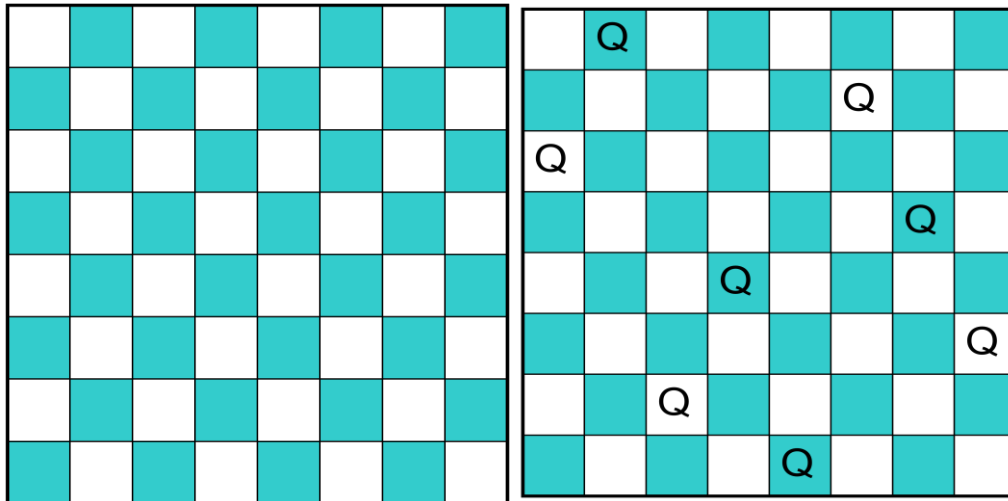
- Place eight queens on a chess board such that no queen can attack another queen
- No path cost because only the final state counts!
- Incremental formulations
- Complete state formulations

**Solutions:**



### Eight Queens Problem Formulation 1:

- **States:**
  - Any arrangement of 0 to 8 queens on the board
- **Initial state:**
  - No queens on the board
- **Successor function:**
  - Add a queen to an empty square
- **Goal Test:**
  - 8 queens on the board and none are attacked
- $64 \times 63 \times \dots \times 57 = 1.8 \times 10^{14}$  possible sequences



Empty Board

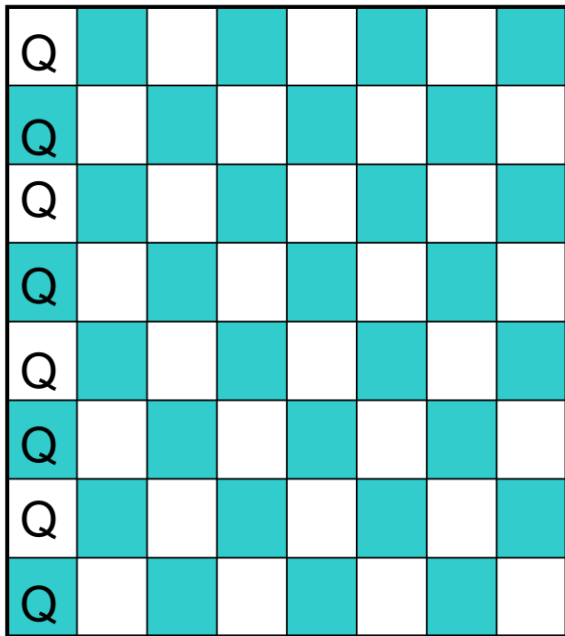
Desired Goal

### Eight Queens Problem Formulation 2:

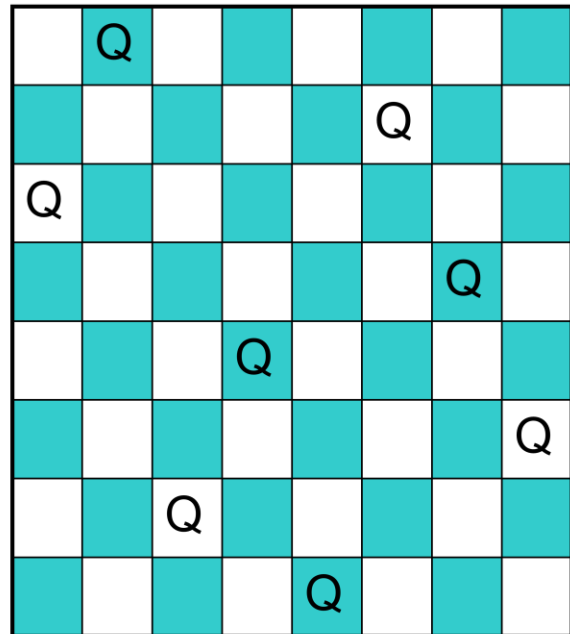
- **States:**
  - Any arrangement of 8 queens on the board
- **Initial state:**
  - All queens are at column 1

- **Successor function:**
  - Change the position of any one queen
- **Goal Test:**
  - 8 queens on the board and none are attacked

## Eight Queens Problem Formulation 2



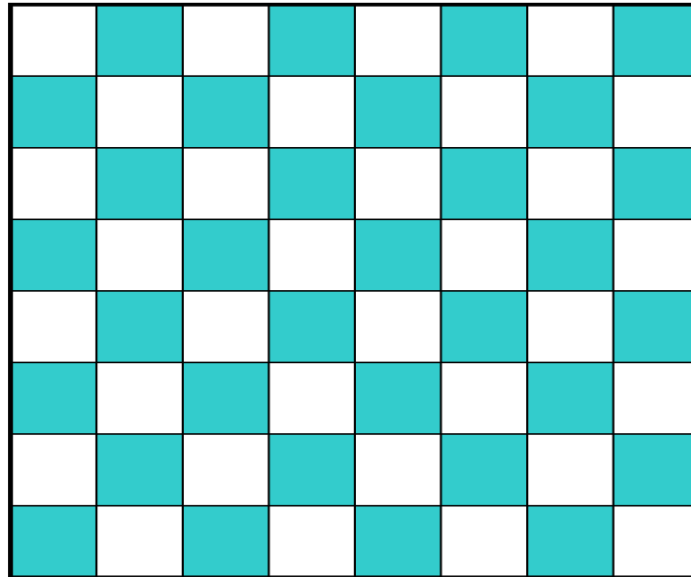
**Initial State**



**Desired Goal**

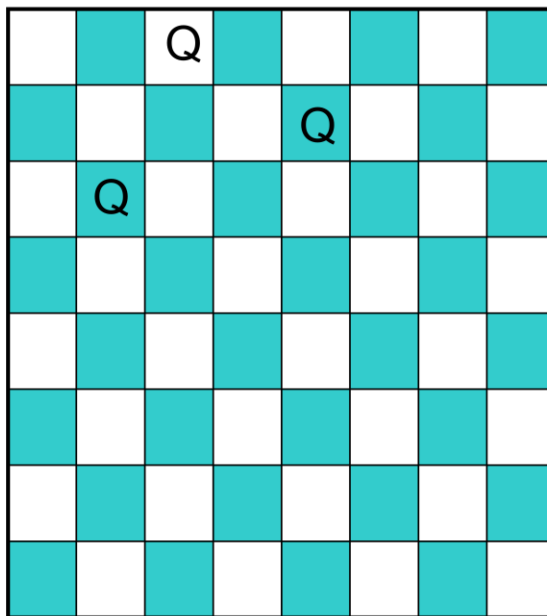
### Eight Queens Problem Formulation 3:

- **States:**
  - Any arrangement of  $k$  queens in the first  $k$  rows such that none are attacked
- **Initial state:**
  - No queens on the board
- **Successor function:**
  - Add a queen to the  $(k+1)^{\text{th}}$  row so that none are attacked
- **Goal Test:**
  - 8 queens on the board and none are attacked

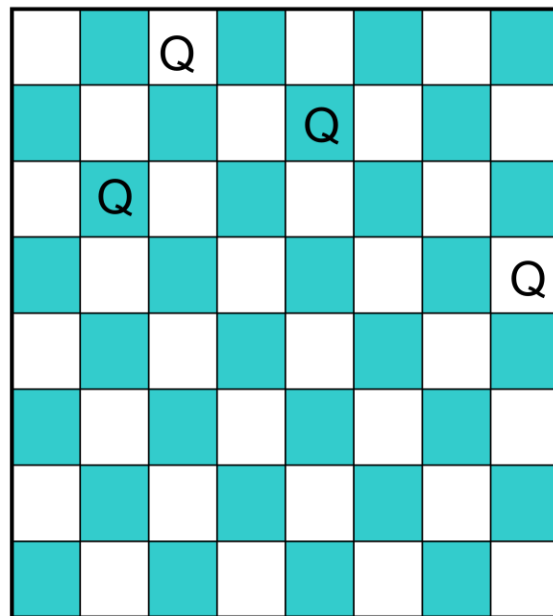


Initial State

## Eight Queens Problem Formulation 3



(k)<sup>th</sup> row , here k=3



Adding a queen at (k+1)<sup>th</sup> row

### Solving 8 Puzzle Problem :

The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state.



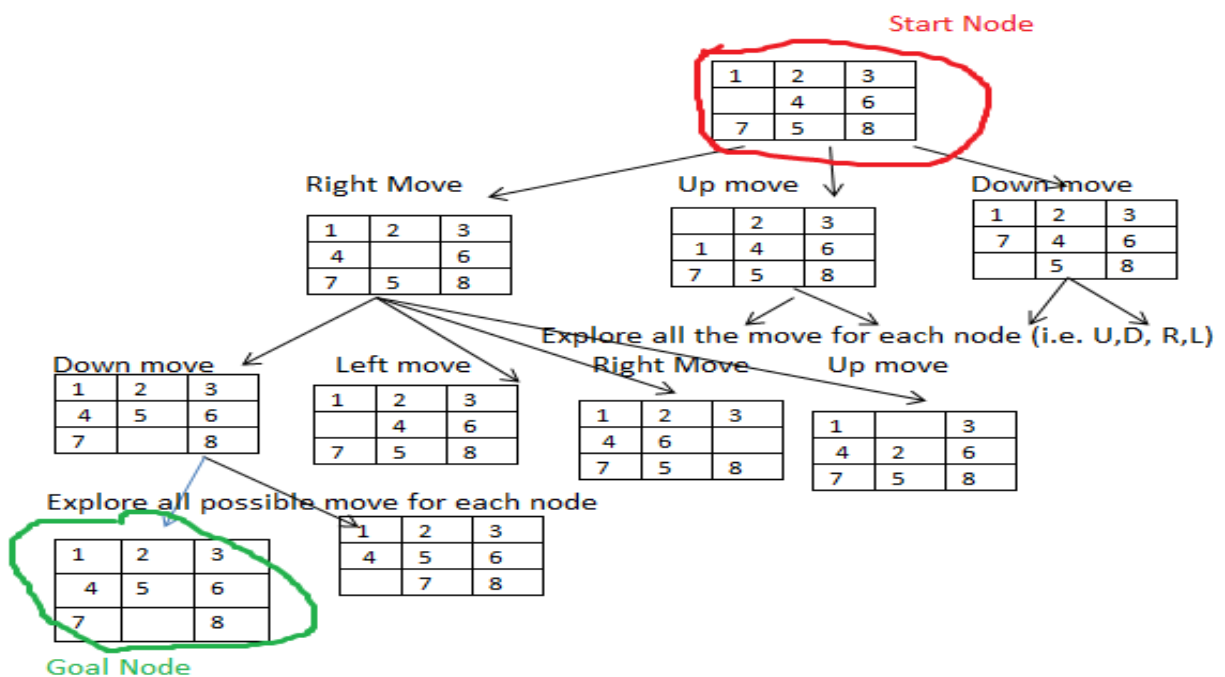
## Rules of solving puzzle

- Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile.
- The empty space can only move in four directions (Movement of empty space)
- Up
- Down
- Right
- Left
- The empty space cannot move diagonally and can take only one step at a time.

O	X	O
X	#	X
O	X	O

- **o- Position** total possible moves are (2),
- **x - position** total possible moves are (3) and
- **#-position** total possible moves are (4)

Let's solve the problem without **Heuristic Search** that is **Uninformed Search** or **Blind Search** (**Breadth First Search** and **Depth First Search**)



## Breadth First Search to solve Eight puzzle problem

## 1.3 Search Strategies

### Search Algorithm

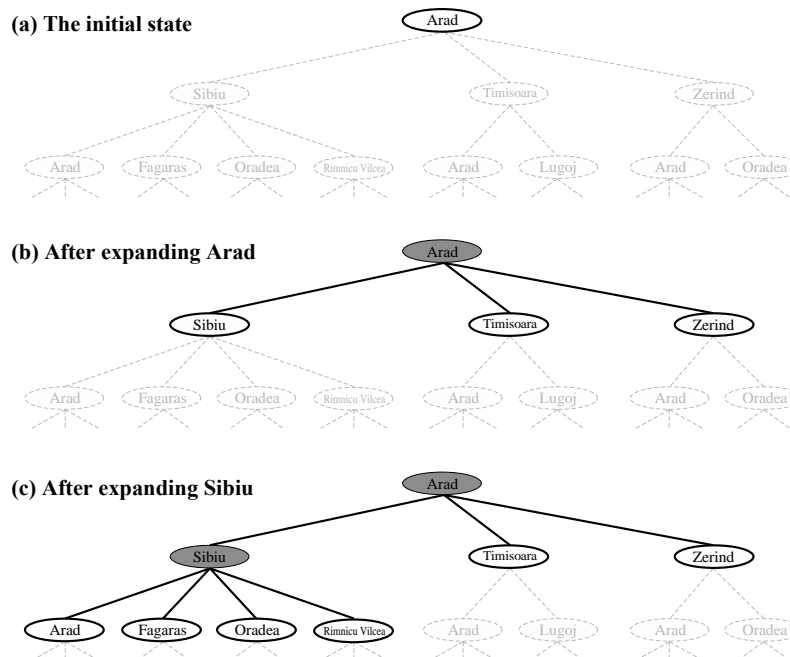
**Search Algorithm** teaches computers to “act rationally” by achieving a certain goal with a certain input value

**Search:** Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:

- **Search Space:** Search space represents a set of possible solutions, which a system may have.
- **Start State:** It is a state from where agent begins **the search**.
- **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.

**Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.

### Search Trees



**Partial search trees for finding a route from Arad to Bucharest**

**function TREE-SEARCH(problem) returns a solution, or failure**

    initialize the frontier using the initial state of problem

**loop do**

**if the frontier is empty then return failure**

        choose a leaf node and remove it from the frontier

**if the node contains a goal state then return the corresponding solution**

```

        expand the chosen node, adding the resulting nodes to the frontier
function GRAPH-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    initialize the explored set to be empty
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        add the node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier
        only if not in the frontier or explored set

```

### An informal description of the general tree-search and graph-search algorithms.

#### Properties of Search Algorithms:

Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:

- **Completeness:** A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.
- **Optimality:** If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.
- **Time Complexity:** Time complexity is a measure of time for an algorithm to complete its task.
- **Space Complexity:** It is the maximum storage space required at any point during the search, as the complexity of the problem.

#### Types of search algorithms:

Based on the search problems we can classify the search algorithms into uninformed (Blind search) search and Informed search (Heuristic search) algorithms.

##### Uninformed/Blind Search:

- The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes.
- Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search. It examines each node of the tree until it achieves the goal node.
- So Distance to goal not taken into account
- Ex: DFS, BFS, Iterative deepening DFS

##### Informed Search:

- Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called a Heuristic search.

- A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.
- Informed search can solve much complex problem which could not be solved in another way.
- The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristics, so it is also called Heuristic search.
- So Information about cost to goal taken into account
- Ex: Best first search , A\* 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, it is also called Heuristic search.
- 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.  $f(n) = g(n)$ .

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

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

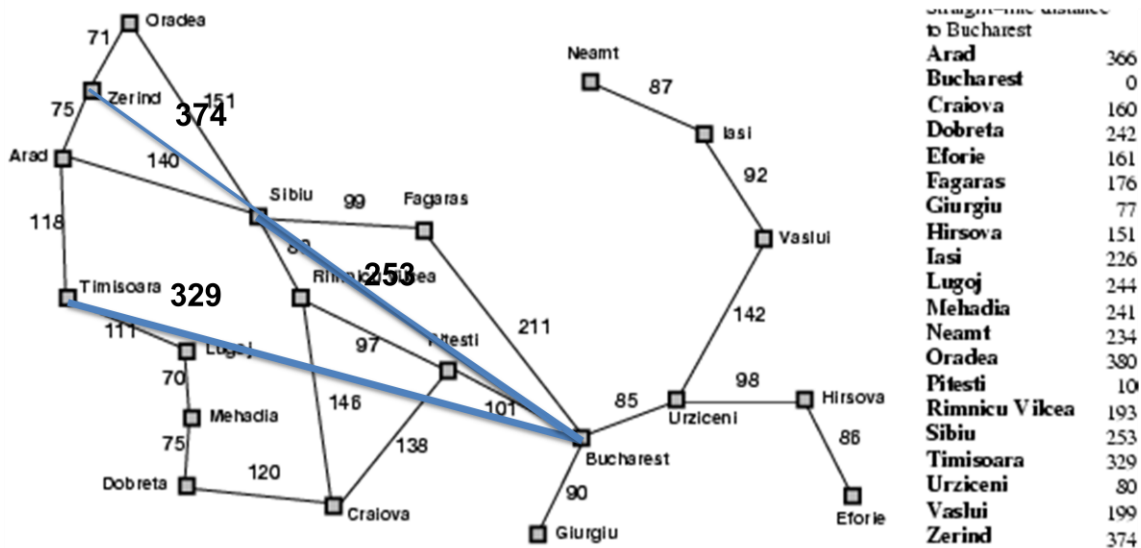
#### Example:

- Let us see how this works for route-finding problems in Romania; we use the **straight line distance** heuristic, which we will call  $h_{SLD}$ .
- If the goal is Bucharest, we need to know the straight-line distances to Bucharest, which are shown in below figure.
- For example,  $h_{SLD}(In(Arad))=366$ . Notice that the values of  $h_{SLD}$  cannot be computed from the problem description itself.
- Moreover, it takes a certain amount of experience to know that  $h_{SLD}$  is correlated with actual road distances and is, therefore, a useful heuristic.

## Romanian path finding problem

Base eg on GPS info.  
No map needed.

Straight-line  
dist. to Bucharest



### Searching for good path from Arad to Bucharest, what is a reasonable “desirability measure” to expand nodes on the fringe?

- The above shows the progress of a greedy best-first search using  $h_{SLD}$  to find a path from Arad to Bucharest.
- The first node to be expanded from Arad will be Sibiu because it is closer to Bucharest than either Zerind or Timisoara.
- The next node to be expanded will be Fagaras because it is closest. Fagaras in turn generates Bucharest, which is the goal.
- For this particular problem, greedy best-first search using  $h_{SLD}$  finds a solution without ever expanding a node that is not on the solution path; hence, its search cost is minimal. It is not

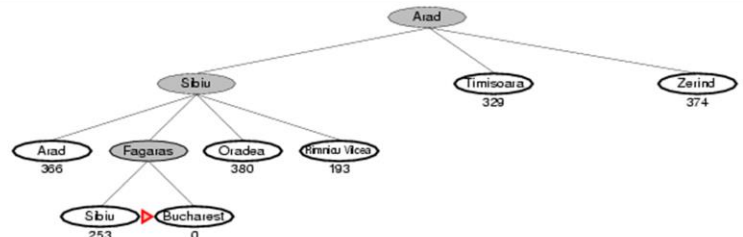
optimal, however: the path via Sibiu and Fagaras to Bucharest is 32 kilometers longer than the path through Rimnicu Vilcea and Pitesti.

- This shows why the algorithm is called “greedy”—at each step it tries to get as close to the goal as it can.

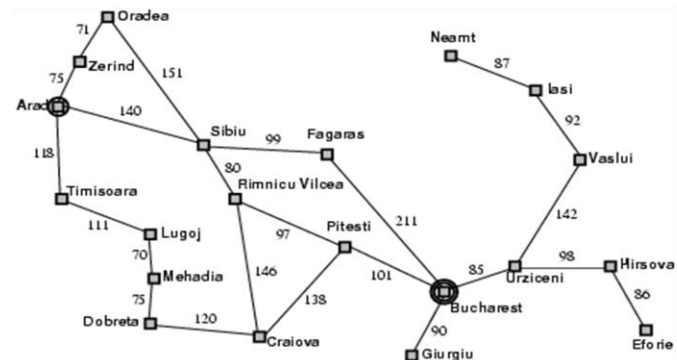
## Greedy best-first search example

### Straight-line dist. to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



So, Arad --- Sibiu --- Fagaras --- Bucharest  
 $140 + 99 + 211 = 450$

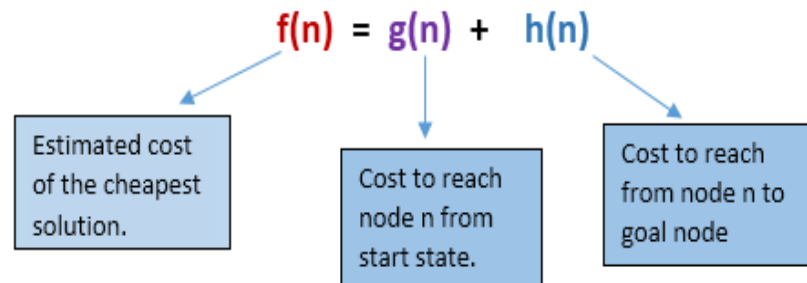


### Properties of greedy best-first search:

- Greedy best-first tree search is also incomplete even in a finite state space, much like depth-first search.
- Consider the problem of getting from Iasi to Fagaras. The heuristic suggests that Neamt be expanded first because it is closest to Fagaras, but it is a dead end.
- The solution is to go first to Vaslui—a step that is actually farther from the goal according to the heuristic—and then to continue to Urziceni, Bucharest, and Fagaras.
- The algorithm will never find this solution, however, because expanding Neamt puts Iasi back into the frontier,
- Iasi is closer to Fagaras than Vaslui is, and so Iasi will be expanded again, leading to an infinite loop.
- The worst-case time and space complexity for the tree version is  $O(b^m)$ ,  
 where  $b$  is the maximum branching factor of the search tree  
 $m$  is the maximum depth of the search space.
- With a good heuristic function, however, the complexity can be reduced substantially.
- The amount of the reduction depends on the particular problem and on the quality of the heuristic.

## A\* search

- 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  $g(n)$ .
- 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.
- 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**.



### Algorithm A\*:

OPEN = nodes on frontier

CLOSED=expanded nodes

OPEN = {<s,nil>}

**while** OPEN is not empty

    remove from OPEN the node <n,p> with minimum  $f(n)$

    place <n,p> on CLOSED

**if** n is a goal node, **return** success (path p)

**for each** edge connecting n & m with cost c

**if** <m,q> is on CLOSED and {p|e} is cheaper than q

**then** remove n from CLOSED , put <m,{p|e}> on OPEN

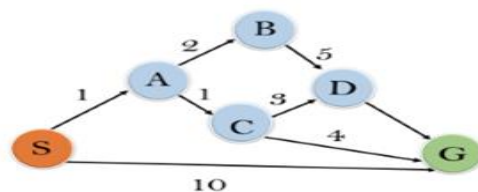
**else if** <m,q> is on OPEN AND {p|e} is cheaper than q

**then** replace q with {p|e}

**else if** m is not on OPEN put <m,{p|e}> on OPEN

**return** failure

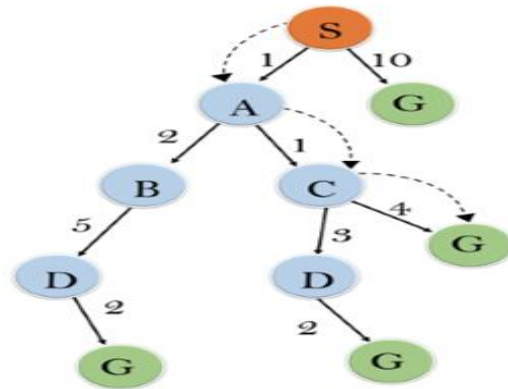
### Example 1:



State	$h(n)$
S	5
A	3
B	4
C	2
D	6
G	0

:

**Solution:**



**Start State:** {(S, 5)}

**Iteration1:** {(S--> A, 4), (S-->G, 10)}

**Iteration2:** {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

**Iteration 3:** {(S--> A-->C-->G, 6), (S--> A-->C-->D, 11), (S--> A-->B, 7), (S-->G, 10)}

**Iteration 4** will give the final result, as **S-->A-->C-->G** it provides the optimal path with cost 6

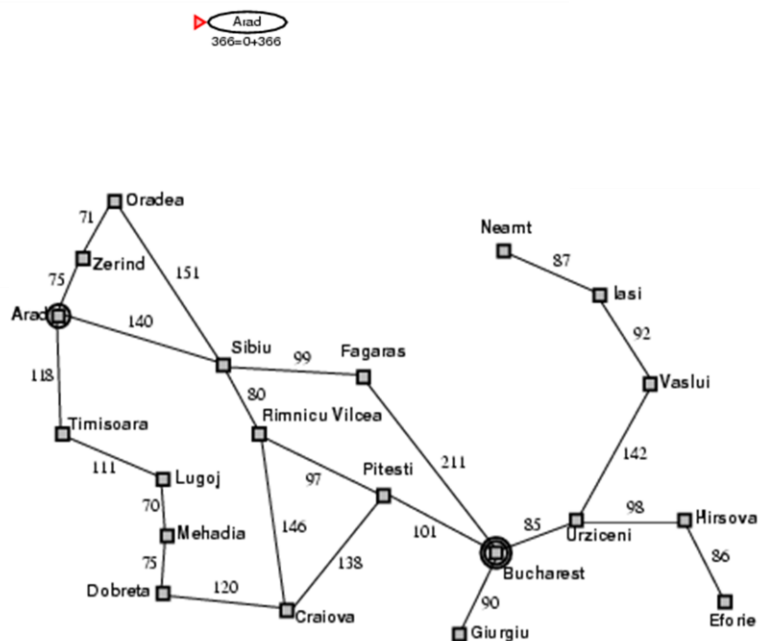
So A\* avoid expanding paths that are already expensive

*Using:  $f(n) = g(n) + h(n)$*

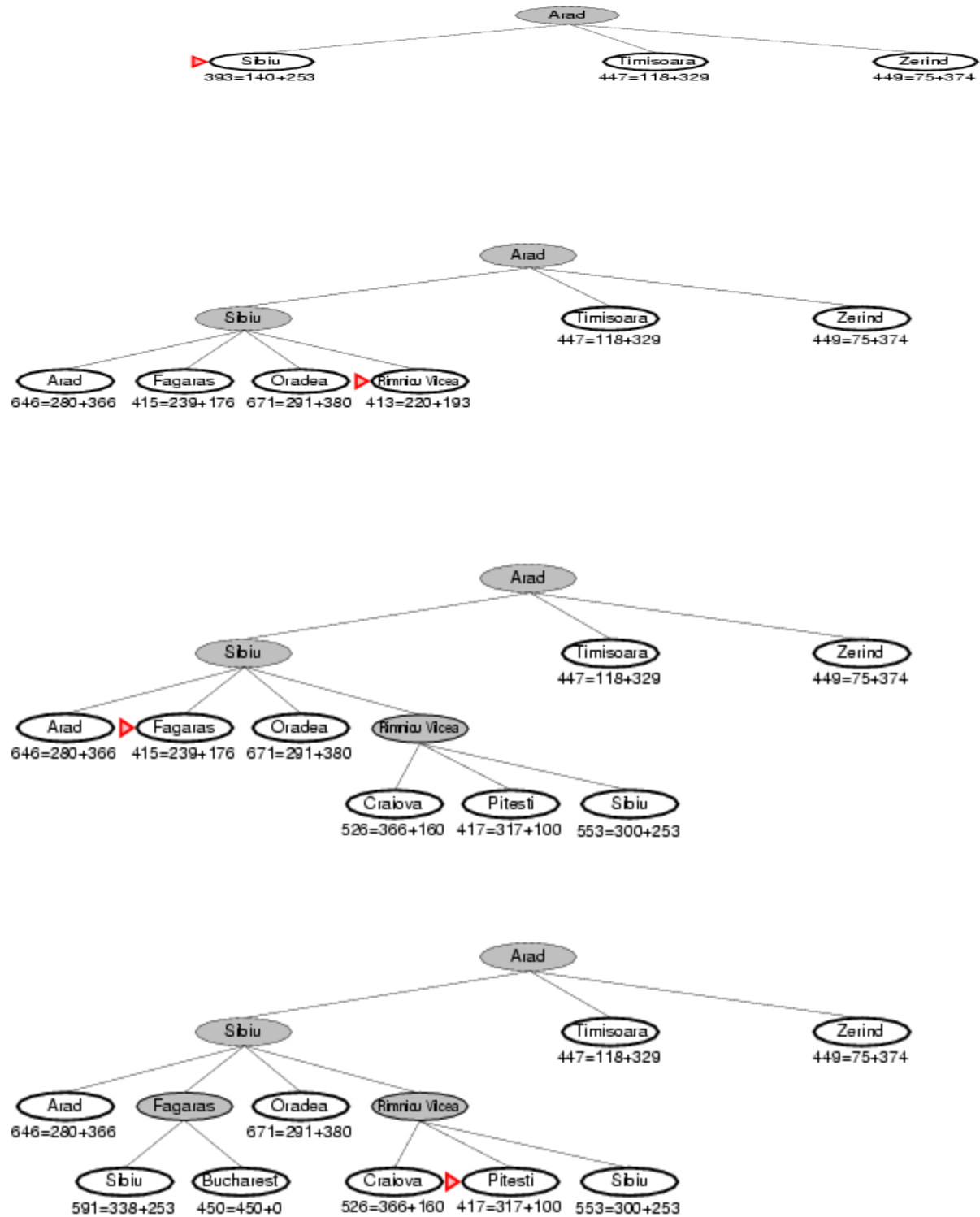
**A\* search example**

**Straight-line  
dist. to Bucharest**

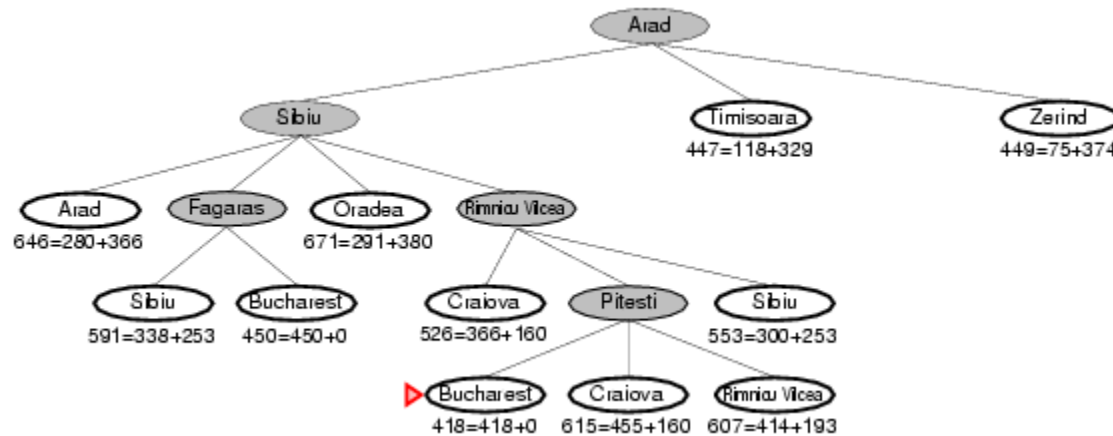
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	101
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374







**Bucharest appears on the fringe but not selected for expansion since its cost (450) is higher than that of Pitesti (417). Important to understand for the proof of optimality of A\***



**Optimal Path Found:** Arad --- Sibiu --- Rimnicu --- Pitesti --- Bucharest

Under some reasonable conditions for the heuristics, we have:

- **Complete :** Yes, unless there are infinitely many nodes with  $f(n) < f(\text{Goal})$
- **Time Complexity:** The time complexity of A\* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution  $d$ . So the time complexity is  $O(b^d)$ , where  $b$  is the branching factor.
- **Space Complexity:** The space complexity of A\* search algorithm is  $O(b^d)$
- **Optimal:** Yes  
Also, optimal use of heuristics information!
- Widely used. E.g. Google maps.
- After almost 40 yrs, still new applications found.
- Also, optimal use of heuristic information.

### A\*: Difficulties

- It becomes often difficult to use A\* as the OPEN queue grows very large.
- A solution is to use algorithms that work with less memory
- Memory bounded heuristic search algorithms reduce the memory requirements for A\* by introducing IDA\*.
- IDA\* is an iterative deepening algorithm.
- The cut-off for nodes expanded in an iteration is decided by the  $f$ -value of the nodes.

## IDA\* Algorithm

- $\text{IDA}^* = \text{A}^* + \text{Iterative Deepening DFS}$
- The key feature of the IDA\* algorithm is that it doesn't keep a track of each visited node which helps in saving memory consumption and can be used where memory is constrained.
- It is path search algorithm which finds shortest path from start node to goal node in weighted graph.
- It borrows the idea to use a heuristic function from A\*

- Its memory usage is weaker than in A\*, thereby reducing memory problem in A\*
- IDA\* is optimal in terms of space and cost
- Problem: Excessive node generation

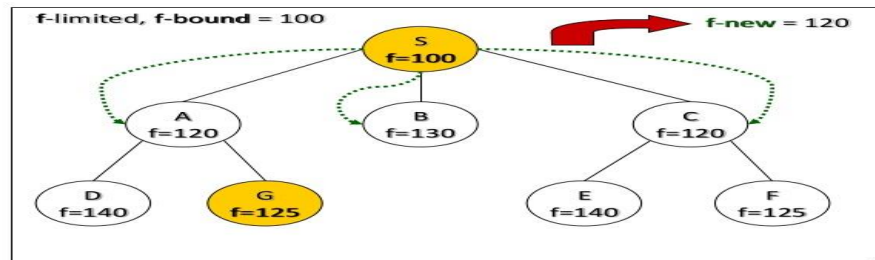
### Algorithm:

- Set certain threshold/f-bound
- If  $f(\text{node}) > \text{threshold/f-bound}$ , prune the node
- Set threshold/f-bound = minimum cost of any node that is pruned
- Terminates when goal is reached.

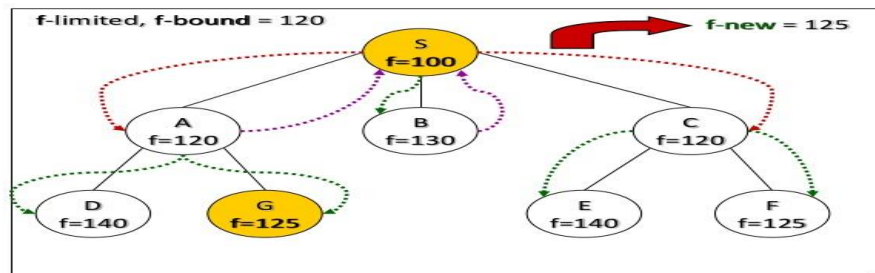
### IDA\* Search Procedure

- Consider the threshold/f-bound value and cycle through the algorithm
- In every branch visit the depth till the  $f(\text{node})$  is greater than the threshold/f-bound and note down that  $f(\text{node})$  value, do this till all branches are explored upto certain depth.
- Then the cycle continues from the starting node again with the new threshold/f-new value that is the minimum of  $f(\text{node})$  values noted down.
- This continues until the goal is found or the time limit is exceeded

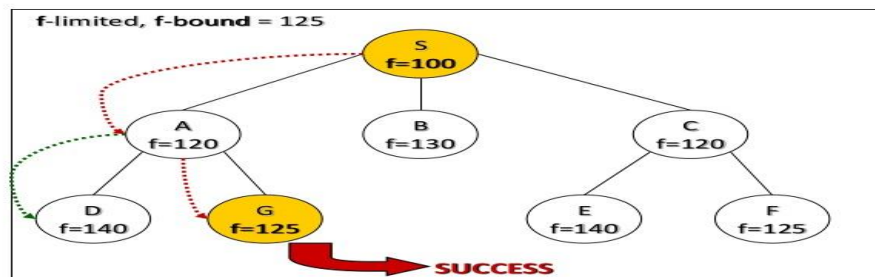
#### Example



#### Example



#### Example



### Properties of IDA\*:

- **Complete:** Yes, similar to A\*
- **Time:** Depends strongly on the number of different values that the heuristic value can take on. If A\* expands N nodes, IDA\* expands  $1+2+\dots+N=O(N^2)$  nodes.
- **Space:** It is DFS, it only requires space proportional to the longest path it explores.
- **Optimal:** Yes, similar to A\*.

## 1.4 Adversarial Search

### Introduction:

- Adversarial search is a **game-playing** technique where the agents are surrounded by a competitive environment.
- A conflicting goal is given to the agents (multi agent). These agents compete with one another and try to defeat one another in order to win the game. Such conflicting goals give rise to the adversarial search.
- Here, game-playing means discussing those games where **human intelligence** and **logic factor** is used, excluding other factors such as **luck factor**.
- **Tic-tac-toe, chess, checkers**, etc., are such type of games where no luck factor works, only mind works.
- Mathematically, this search is based on the concept of '**Game Theory**.'
- According to game theory, a game is played between two players. To complete the game, one has to win the game and the other loses automatically.

### Perfect decision games:

- In game theory, a sequential game has **perfect information** if each player, when making any decision, is perfectly informed of all the events that have previously occurred, including the "initialization event" of the game (e.g. the starting hands of each player in a card game).
- Perfect information is importantly different from complete information, which implies common knowledge of each player's utility functions, payoffs, strategies and "types". A game with perfect information may or may not have complete information.
- Chess is an example of a game with perfect information as each player can see all the pieces on the board at all times.
- Other examples of games with perfect information include tic-tac-toe, checkers, infinite chess.

### Imperfect decision games:

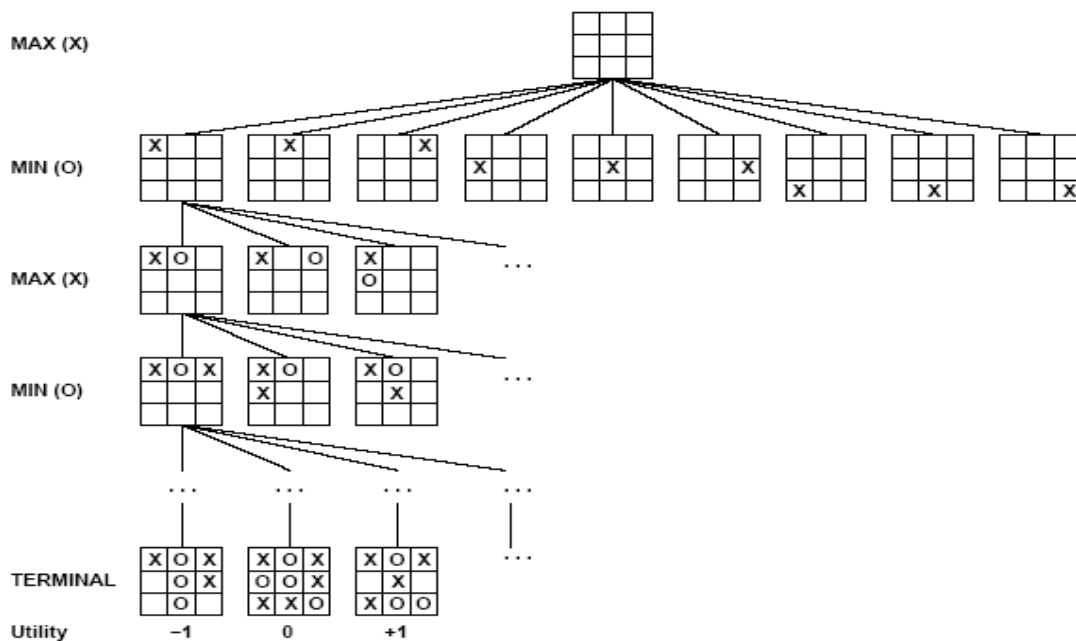
- In game theory, a game is imperfect game if each player, when making any decision, is uninformed of all the events that have previously occurred.
- Imperfect information implies no idea of the move taken by the opponent it means the player is unaware of the actions chosen by other players.
- Card games where each player's cards are hidden from other players such as poker and bridge are examples of games with imperfect information

### Game Setup:

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over

- Winner gets award, loser gets penalty.
- Games as search:
  - Initial state: e.g. board configuration of chess
  - Successor function: list of (move , state) pairs specifying legal moves.
  - Terminal test: Is the game finished?
  - Utility function: Gives numerical value of terminal states.  
E.g. win (+1), lose (-1) and draw (0) in tic-tac-toe or chess
- Size of Search tree is  $O(b^d)$   
where  $b$  = branching factor ,  $d$  = number of moves by both players
- Example: Chess :  $b \sim 35$  and  $D \sim 100$
- So size of search tree is  $\sim 10^{154}$  (!!) which completely impractical to search this.
- Game-playing emphasizes being able to make optimal decisions in a finite amount of time, somewhat realistic as a model of a real-world agent and even if games themselves are artificial

### Partial Game Tree for Tic-Tac-Toe

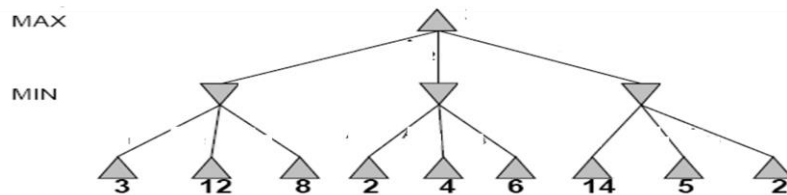


### MINI - MAX ALGORITHM:

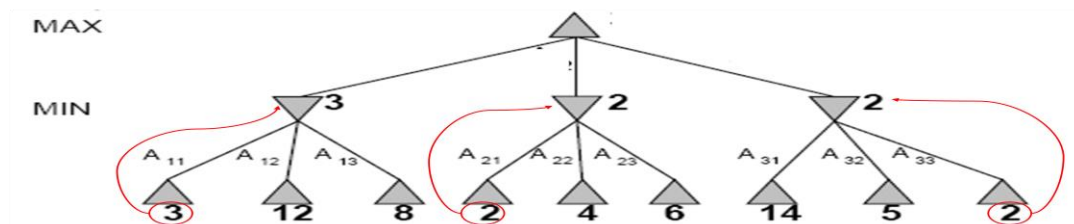
- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.

- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, 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.

### Two-Ply Game Tree

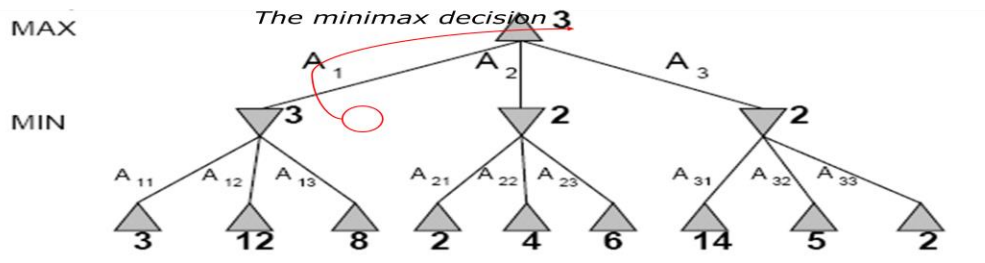


### Two-Ply Game Tree



# Two-Ply Game Tree

Minimax maximizes the utility for the worst-case outcome for max



## Pseudo code for Minimax Algorithm:

**function** MINIMAX-DECISION(*state*) **returns** an action

**inputs:** *state*, current state in game

$v \leftarrow \text{MAX-VALUE}(\text{state})$

**return** the action in **SUCCESSORS**(*state*) with value  $v$

**function** MAX-VALUE(*state*) **returns** a utility value

**if** **TERMINAL-TEST**(*state*) **then return** **UTILITY**(*state*)

$v \leftarrow -\infty$

**for**  $a, s$  in **SUCCESSORS**(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$

**return**  $v$

**function** MIN-VALUE(*state*) **returns** a utility value

**if** **TERMINAL-TEST**(*state*) **then return** **UTILITY**(*state*)

$v \leftarrow -\infty$

**for**  $a, s$  in **SUCCESSORS**(*state*) **do**

$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$

**return**  $v$

## Draw back:

- 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.
- This limitation of the minimax algorithm can be improved from **alpha-beta pruning**

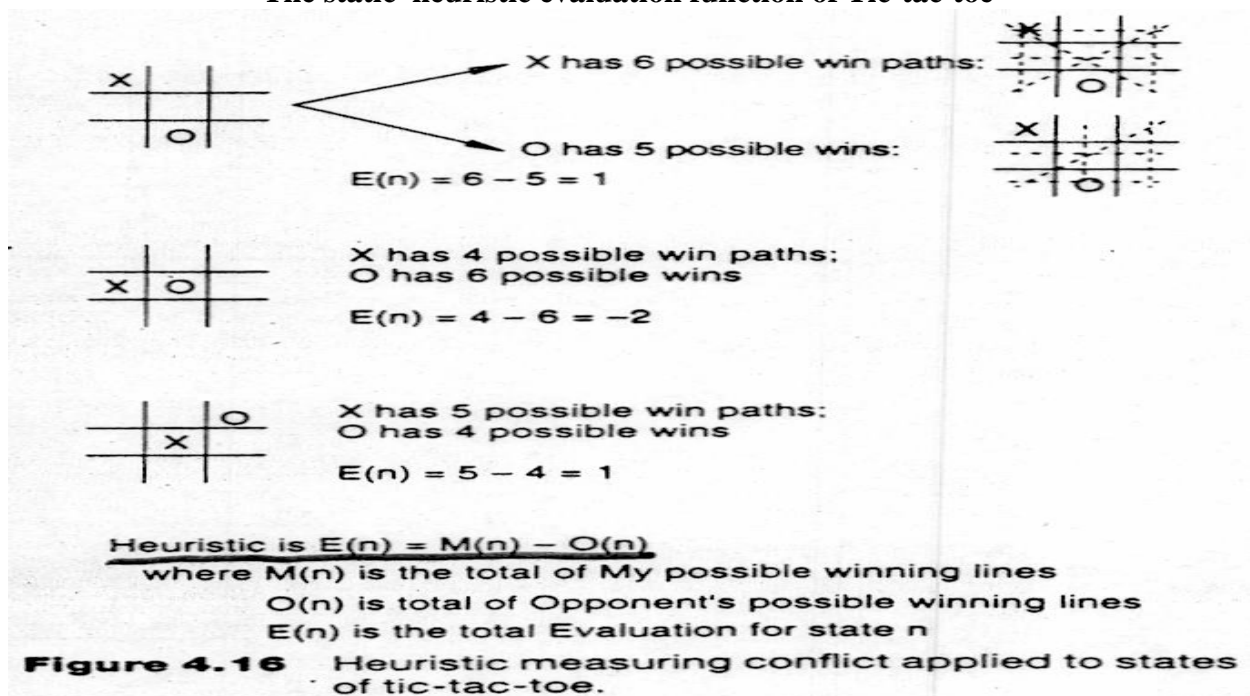
## Static (Heuristic) Evaluation Functions:

### An Evaluation Function:

- estimates how good the current board configuration is for a player.

- Typically, one figures how good it is for the player, and how good it is for the opponent, and subtracts the opponents score from the players
- Tic-tac-toe : Number of successful moves by X – Number of successful moves by O
- Chess: Value of all white pieces - Value of all black pieces
- Typical values from -infinity (loss) to +infinity (win) or [-1, +1].
- If the board evaluation is X for a player, it's -X for the opponent.
- Many clever ideas about how to use the evaluation function.  
e.g. null move heuristic: let opponent move twice.
- Example:  
Evaluating chess boards,  
Tic-tac-toe

### The static heuristic evaluation function of Tic-tac-toe



### $\alpha$ - $\beta$ 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.

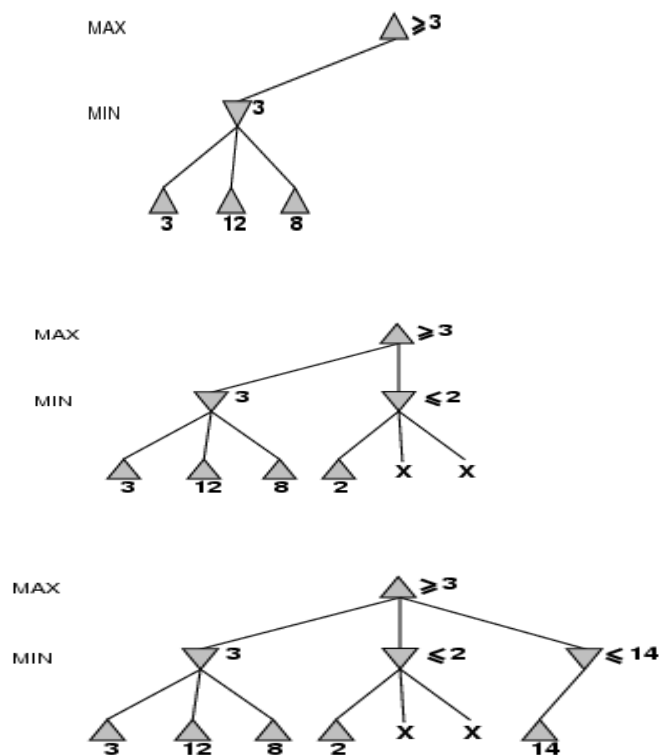
The two-parameter can be defined as:

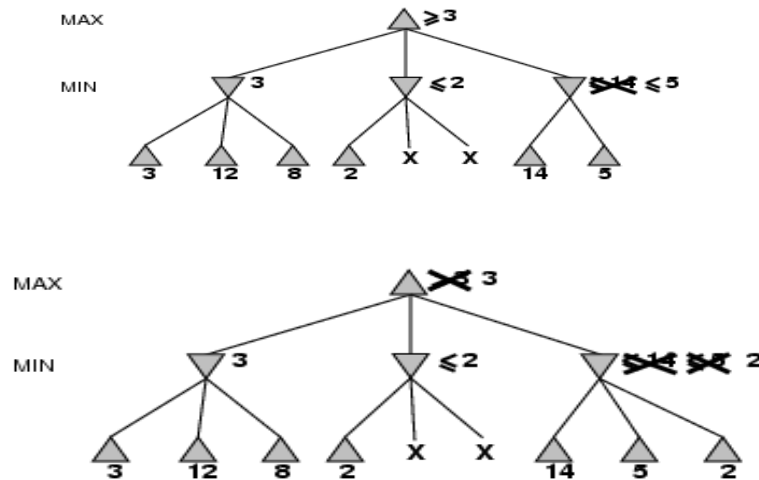
- **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is  $-\infty$ .
- **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is  $+\infty$ .
- 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.
- Using  $\alpha$ - $\beta$  Pruning improve search by reducing the size of the game tree.

### Principle of $\alpha$ - $\beta$ Pruning

- If a move is determined worse than another move already examined, then there is no need for further examination of the node.

### Example 1:

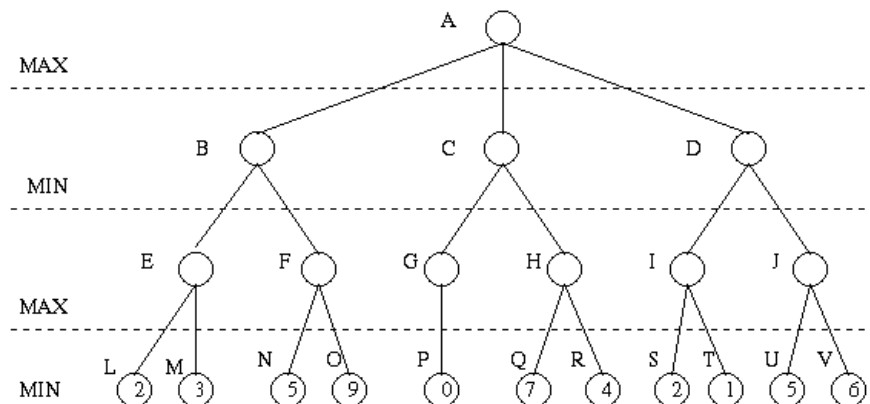




### Rules of Thumb

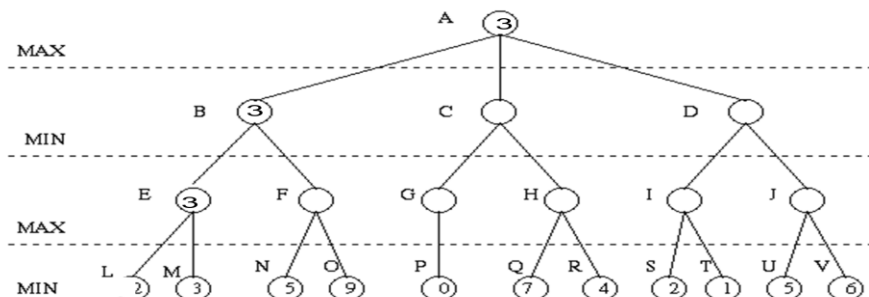
- $\alpha$  is the best (highest) found so far along the path for Max
- $\beta$  is the best (lowest) found so far along the path for Min
- Search below a MIN node may be **alpha-pruned** if its  $\beta \leq \alpha$  of some MAX ancestor
- Search below a MAX node may be **beta-pruned** if its  $\alpha \geq \beta$  of some MIN ancestor.

### Example 2:



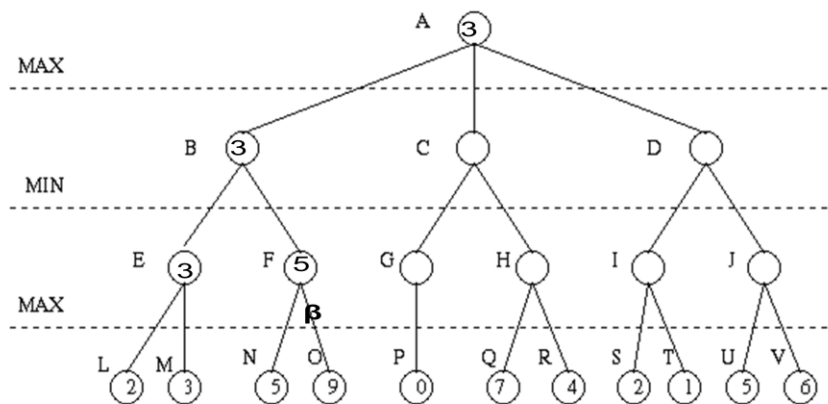
1. Search below a MIN node may be alpha-pruned if the beta value is  $\leq$  to the alpha value of some MAX ancestor.

2. Search below a MAX node may be beta-pruned if the alpha value is  $\geq$  to the beta value of some MIN ancestor.



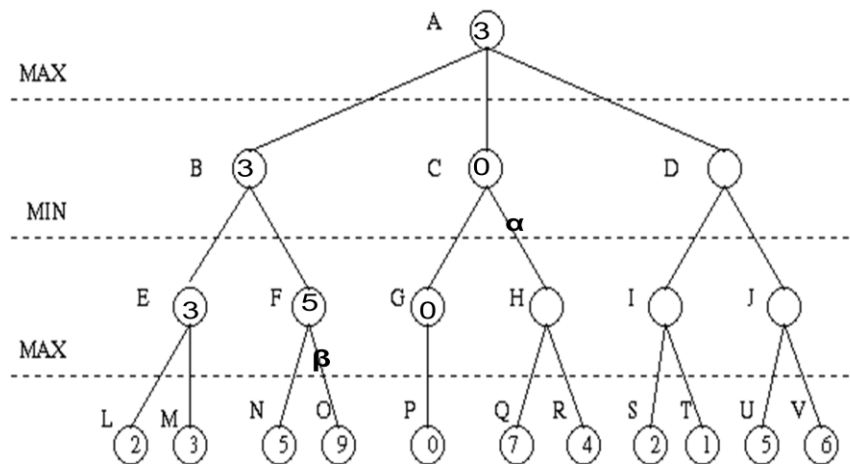
1. Search below a MIN node may be alpha-pruned if the beta value is  $\leq$  to the alpha value of some MAX ancestor.

2. Search below a MAX node may be beta-pruned if the alpha value is  $\geq$  to the beta value of some MIN ancestor.



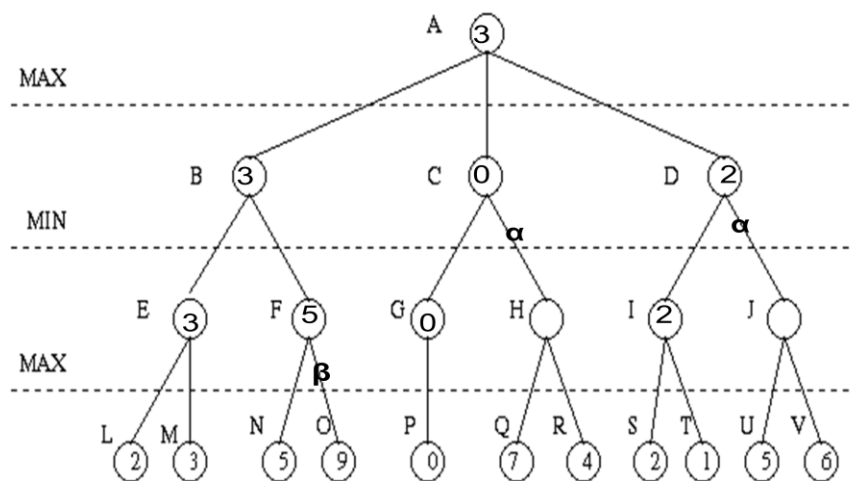
1. Search below a MIN node may be alpha-pruned if the beta value is  $\leq$  to the alpha value of some MAX ancestor.

2. Search below a MAX node may be beta-pruned if the alpha value is  $\geq$  to the beta value of some MIN ancestor.



1. Search below a MIN node may be alpha-pruned if the beta value is  $\leq$  to the alpha value of some MAX ancestor.

2. Search below a MAX node may be beta-pruned if the alpha value is  $\geq$  to the beta value of some MIN ancestor.



## The $\alpha$ - $\beta$ algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action  
  inputs: state, current state in game  
   $v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$   
  return the action in SUCCESSORS(state) with value  $v$ 
```

---

```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value  
  inputs: state, current state in game  
            $\alpha$ , the value of the best alternative for MAX along the path to state  
            $\beta$ , the value of the best alternative for MIN along the path to state  
  if TERMINAL-TEST(state) then return UTILITY(state)  
   $v \leftarrow -\infty$   
  for  $a, s$  in SUCCESSORS(state) do  
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$   
    if  $v \geq \beta$  then return  $v$   
     $\alpha \leftarrow \text{MAX}(\alpha, v)$   
  return  $v$ 
```

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value  
  inputs: state, current state in game  
            $\alpha$ , the value of the best alternative for MAX along the path to state  
            $\beta$ , the value of the best alternative for MIN along the path to state  
  if TERMINAL-TEST(state) then return UTILITY(state)  
   $v \leftarrow +\infty$   
  for  $a, s$  in SUCCESSORS(state) do  
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$   
    if  $v \leq \alpha$  then return  $v$   
     $\beta \leftarrow \text{MIN}(\beta, v)$   
  return  $v$ 
```

### Properties of $\alpha$ - $\beta$ Prune:

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning b(e.g., chess, try captures first, then threats, forward moves, then backward moves...)
- With "perfect ordering," time complexity =  $O(b^{m/2})$ 
  - $\rightarrow$  doubles depth of search that alpha-beta pruning can explore

# Artificial Intelligence (CSE)

## UNIT-II

### 2.1.1 Logical Agents:

Knowledge-based agents – agents that have an explicit representation of knowledge that can be reasoned with.

These agents can manipulate this knowledge to infer new things at the “knowledge level”

#### A Knowledge Based Agent

- A knowledge-based agent includes a knowledge base and an inference system.
- A knowledge base is a set of representations of facts of the world.
- Each individual representation is called a sentence.
- The sentences are expressed in a knowledge representation language.
- The agent operates as follows:
  1. It TELLS the knowledge base what it perceives.
  2. It ASKS the knowledge base what action it should perform.
  3. It performs the chosen action.

#### Architecture of Knowledge based agent:

##### Knowledge Level.

- The most abstract level: describe agent by saying what it knows.
- Example: A taxi agent might know that the Golden Gate Bridge connects San Francisco with the Marin County.

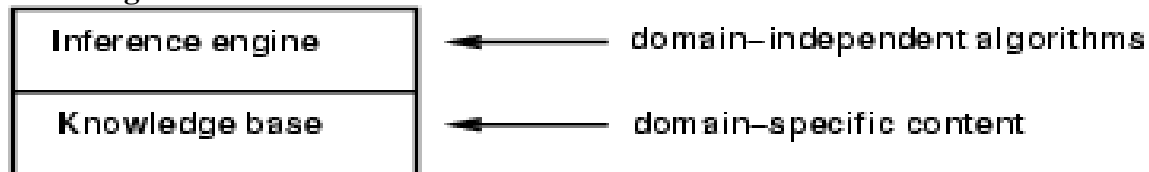
##### Logical Level.

- The level at which the knowledge is encoded into sentences.
- Example: Links(GoldenGateBridge, SanFrancisco, MarinCounty).

##### Implementation Level.

- The physical representation of the sentences in the logical level.
- Example: '(links goldengatebridge sanfrancisco marincounty)

#### Knowledge Bases:



- Knowledge base = set of sentences in a formal language
- Declarative approach to building an agent (or other system):
  - Tell it what it needs to know
  - Then it can Ask itself what to do - answers should follow from the KB
- Agents can be viewed at the knowledge level - i.e., what they know, regardless of how implemented
- Or at the implementation level

i.e., data structures in KB and algorithms that manipulate them

### A Simple Knowledge Based Agent

```
function KB-AGENT(percept) returns an action  
  static: KB, a knowledge base  
          t, a counter, initially 0, indicating time  
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))  
  action ← ASK(KB, MAKE-ACTION-QUERY(t))  
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))  
  t ← t + 1  
  return action
```





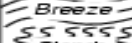










The agent must be able to:

- Represent states, actions, etc.
- Incorporate new percepts
- Update internal representations of the world
- Deduce hidden properties of the world
- Deduce appropriate actions

### The Wumpus world environment

- The Wumpus computer game
- The agent explores a cave consisting of rooms connected by passageways.
- Lurking somewhere in the cave is the Wumpus, a beast that eats any agent that enters its room.
- Some rooms contain bottomless pits that trap any agent that wanders into the room.
- Occasionally, there is a heap of gold in a room.
- The goal is to collect the gold and exit the world without being eaten

### A Wumpus World

4				
3		  		
2				
1	 START			
	1	2	3	4

## **A Wumpus World PEAS description**

### **Performance measure:**

- gold +1000, death -1000
- -1 per step, -10 for using the arrow

### **Environment:** 4 x 4 grid of rooms

- Squares adjacent to wumpus are smelly
- Squares adjacent to pit are breezy
- Glitter iff gold is in the same square
- Shooting kills wumpus if you are facing it
- Shooting uses up the only arrow
- Grabbing picks up gold if in same square
- Releasing drops the gold in same square

**Actuators:** Left turn, Right turn, Forward, Grab, Release, Shoot

**Sensors:** Stench, Breeze, Glitter, Bump, Scream (shot Wumpus)

## **Agents in a Wumpus World: Percepts**

The agent perceives

- a stench in the square containing the wumpus and in the adjacent squares (not diagonally)
- a breeze in the squares adjacent to a pit
- a glitter in the square where the gold is
- a bump, if it walks into a wall
- a woeful scream everywhere in the cave, if the wumpus is killed

The percepts are given as a five-symbol list. If there is a stench and a breeze, but no glitter, no bump, and no scream, the percept is [Stench, Breeze, None, None, None]

### **Percepts given to the agent**

- Stench
- Breeze
- Glitter
- Bump (ran into a wall)
- Scream (wumpus has been hit by arrow)

**Principle Difficulty:** Agent is initially ignorant of the configuration of the environment – going to have to reason to figure out where the gold is without getting killed!

## **Wumpus World Actions**

- **go forward**
- **turn right**
- **turn left**
- **grab:** Pick up an object that is in the same square as the agent

- **shoot:** Fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits and kills the wumpus or hits the outer wall. The agent has only one arrow, so only the first Shoot action has any effect
- **climb** is used to leave the cave. This action is only effective in the start square
- **die:** This action automatically and irretrievably happens if the agent enters a square with a pit or a live wumpus

### Wumpus goal:

The agent's goal is to find the gold and bring it back to the start square as quickly as possible, without getting killed

- 1000 points reward for climbing out of the cave with the gold
- 1 point deducted for every action taken
- 10000 points penalty for getting killed

Note that in each case for which the agent draws a conclusion from the available information, that conclusion is *guaranteed to be correct if the available information is correct*. This is a fundamental property of logical reasoning.

### The Wumpus Agent First Step

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1	2,1	3,1	4,1
<b>A</b> OK	OK		

(a)

**A** = Agent  
**B** = Breeze  
**G** = Glitter, Gold  
**OK** = Safe square  
**P** = Pit  
**S** = Stench  
**V** = Visited  
**W** = Wumpus

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 P?	3,2	4,2
OK			
1,1	2,1	3,1 P?	4,1
V OK	<b>A</b> B OK		

(b)

### Later

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2	2,2	3,2	4,2
<b>A</b> S OK	OK		
1,1	2,1	3,1 P!	4,1
V OK	B V OK		

(a)

**A** = Agent  
**B** = Breeze  
**G** = Glitter, Gold  
**OK** = Safe square  
**P** = Pit  
**S** = Stench  
**V** = Visited  
**W** = Wumpus

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3	3,3 P?	4,3
1,2	<b>A</b> S G B		
1,2	2,2	3,2	4,2
S V OK	V OK		
1,1	2,1	3,1 P!	4,1
V OK	B V OK		

(b)

### Let's Play



1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

**A** = Agent  
**B** = Breeze  
**G** = Glitter, Gold  
**OK** = Safe square  
**P** = Pit  
**S** = Stench  
**V** = Visited  
**W** = Wumpus

### Wumpus World Characterization:

- **Fully Observable:** No – only local perception
- **Deterministic:** Yes – outcomes exactly specified
- **Episodic:** No – sequential at the level of actions
- **Static :** Yes – Wumpus and Pits do not move
- **Discrete:** Yes
- **Single-agent:** Yes – Wumpus is essentially a natural feature

### 2.1.2 Logic in general:

- Logics are formal languages for representing information such that conclusions can be drawn
- Syntax defines how symbols can be put together to form the sentences in the language
- Semantics define the "meaning" of sentences;
  - i.e., define truth of a sentence in a world (given an interpretation)
  - E.g., the language of arithmetic
  - $x+2 \geq y$  is a sentence;  $x+2+y > \{\}$  is not a sentence
  - $x+2 \geq y$  is true iff the number  $x+2$  is no less than the number  $y$
  - $x+2 \geq y$  is true in a world where  $x = 7, y = 1$
  - $x+2 \geq y$  is false in a world where  $x = 0, y = 6$

### Propositional Logic:

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. Propositions can be either true or false, but it cannot be both. Propositional logic is also called Boolean logic as it works on 0 and 1.

Examples:

Today is Tuesday.

The Sun rises from West (False proposition)  
 $2+2=5$  (False proposition)  
 $2+3=5$

### Syntax of Propositional Logic:

The syntax of propositional logic defines the allowable sentences. The atomic sentences consist of a single proposition symbol. Each such symbol stands for a proposition that can be true or false. We use symbols that start with an uppercase letter and may contain other letters or subscripts.

Example: P, Q, R, W1, 3 and North.

The names are arbitrary but are often chosen to have some mnemonic value—we use W1,3 to stand for the proposition that the wumpus is in [1,3]. There are two proposition symbols with fixed meanings:

True is the always-true proposition and

False is the always-false proposition.

Complex sentences are constructed from simpler sentences, using parentheses and logical connectives.

In propositional logic there are two types of propositions. They are:

1. Atomic Propositions
2. Compound Propositions

**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.

Ex: The sun rises from west . (False Atomic Propositions)

The sun rises from east (True Atomic Propositions)

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

Ex: Arsalan is an engineer and arsalan lives in dubai.

Arsalan is an engineer or arsalan is a doctor.

Propositional logic is the simplest logic – illustrates basic ideas

The proposition symbols  $P_1$ ,  $P_2$  etc are (atomic) sentences

- If S is a sentence,  $\neg(S)$  is a sentence (negation)
- If  $S_1$  and  $S_2$  are sentences,  $(S_1 \wedge S_2)$  is a sentence (conjunction)
- If  $S_1$  and  $S_2$  are sentences,  $(S_1 \vee S_2)$  is a sentence (disjunction)
- If  $S_1$  and  $S_2$  are sentences,  $(S_1 \Rightarrow S_2)$  is a sentence (implication)
- If  $S_1$  and  $S_2$  are sentences,  $(S_1 \Leftrightarrow S_2)$  is a sentence (biconditional)

$P$	$Q$	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

**Truth Table for connectives**

### Examples of Connectives:

$(P \wedge Q)$  Arsalan likes football and Arsalan likes baseball.  
 $(P \vee Q)$  Arsalan is a doctor or Arsalan is an engineer.  
 $(P \Rightarrow Q)$  If it is raining, then the street is wet.  
 $(P \Leftrightarrow Q)$  I am breathing if and only if I am alive

### Precedence of Connectives:

To eliminate the ambiguity we define a precedence for each operator. The “not” operator ( $\neg$ ) has the highest precedence, followed by  $\wedge$  (conjunction),  $\vee$  (disjunction),  $\Rightarrow$  (implication),  $\Leftrightarrow$  (biconditional).

**Example:**  $\neg A \wedge B$  the  $\neg$  binds most tightly, giving us the equivalent of  $(\neg A) \wedge B$  rather than  $\neg (A \wedge B)$

### Semantics of Propositional Logic:

Each model specifies true/false for each proposition symbol

E.g.     A       B       C  
              false   true   false

With these symbols, 8 possible models, can be enumerated automatically.

Rules for evaluating truth with respect to a model  $m$ :

$\neg S$	is true iff	S is false
$S_1 \wedge S_2$	is true iff	$S_1$ is true and $S_2$ is true
$S_1 \vee S_2$	is true iff	$S_1$ is true or $S_2$ is true
$S_1 \Rightarrow S_2$	is true iff	$S_1$ is false or $S_2$ is true
i.e.,	is false iff	$S_1$ is true and $S_2$ is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$ is true and $S_2 \Rightarrow S_1$ is true

Simple recursive process evaluates an arbitrary sentence,

e.g.,  $\neg A \wedge (B \vee C) = \text{true} \wedge (\text{true} \vee \text{false}) = \text{true} \wedge \text{true} = \text{true}$

### Limitations of Propositional Logic:

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

Examples:

All boys are smart.

All girls are hardworking.

Some mangoes are sweet.

Propositional logic has limited expressive power and in propositional logic, we cannot describe statements in terms of their properties or logical relationships.

### 2.1.3 First Order Logic (FOL):

Propositional logic assumes the world contains facts whereas first-order logic (like natural language) assumes the world contains

- **Objects:** people, houses, numbers, colors, baseball games, wars, ...
- **Relations:** red, round, prime, brother of, bigger than, part of, comes between, ...
- **Functions:** father of, best friend, one more than, plus, ...

### Syntax of FOL:

- Constants      KingJohn, 2,...
- Predicates     Brother, >,...
- Functions      Sqrt, ...
- Variables      x, y, a, b,...
- Connectives    $\neg, \Rightarrow, \wedge, \vee, \Leftrightarrow$
- Equality        =
- Quantifiers     $\forall, \exists$

### Logics in general:

Ontological Commitment:

- What exists in the world — TRUTH
- PL : facts hold or do not hold.
- FOL : objects with relations between them that hold or do not hold

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	degree of truth $\in [0, 1]$	known interval value

- Constant Symbols:
  - Stand for objects
  - e.g., KingJohn, 2,...
- Predicate Symbols
  - Stand for relations
  - e.g., Brother(Richard, John), greater\_than(3,2)...
- Function Symbols
  - Stand for functions
  - e.g., Sqrt(3) ...

### Relations:

- Some relations are properties: they state
- some fact about a single object:  
Round(ball), Prime(7).
- n-ary relations state facts about two or more objects:  
Married(John,Mary), LargerThan(3,2).
- Some relations are functions: their value is another object: Plus(2,3), Father(Dan).

### Terms:

- Term = logical expression that refers to an object.
- There are 2 kinds of terms:
  - constant symbols: Table, Computer
  - function symbols: Sqrt(3), Plus(2,3) etc
- Functions can be nested:
  - Pat\_Grandfather(x) = father(father(x))
- Terms can contain variables.

- No variables = **ground term**.

### Atomic Sentence:

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.

Atomic sentence = *predicate (term<sub>1</sub>,...,term<sub>n</sub>)*

Atomic sentences state facts using terms and predicate symbols P(x,y) interpreted as “x is P of y”

**Examples:** LargerThan(2,3) is false.

Married(Father(Richard), Mother(John)) could be true or false

### Complex Sentence:

Complex sentences are made from atomic sentences using connectives

$\neg S, S_1 \wedge S_2, S_1 \vee S_2, S_1 \Rightarrow S_2, S_1 \Leftrightarrow S_2,$

E.g. *Sibling(KingJohn,Richard)  $\Rightarrow$  Sibling(Richard,KingJohn)*

Complex sentences are made by combining atomic sentences using connectives. We make complex sentences with connectives (just like in propositional logic).

### Some more examples:

1. Brother(Richard, John)  $\wedge$  Brother(John, Richard)
2. King(Richard)  $\vee$  King(John)
3. King(John)  $\Rightarrow \neg$  King(Richard)

Note: Semantics are the same as in propositional logic

### Variables:

- Person(John) is true or false because we give it a single argument ‘John’
- We can be much more flexible if we allow variables which can take on values in a domain. e.g., all persons x, all integers i, etc.
  - E.g., can state rules like Person(x)  $\Rightarrow$  HasHead(x)
  - or Integer(i)  $\Rightarrow$  Integer(plus(i,1))

### 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:

1. **Universal Quantifier, (for all, everyone, everything)**
2. **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.

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

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

**Note: In universal quantifier we use implication " $\rightarrow$ ".**

### 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.

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

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

Note:

- In Existential quantifier we always use AND or Conjunction symbol ( $\wedge$ ).
- 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)." 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:  
 $\exists (x) [ \text{student}(x) \rightarrow \text{failed}(x, \text{Mathematics}) \wedge \forall (y) [ \neg (x=y) \wedge \text{student}(y) \rightarrow \neg \text{failed}(x, \text{Mathematics}) ]$ .

### 2.1.4 Inference Techniques in Logical Agents

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

**Techniques:**

- Forward Chaining
- Backward Chaining
- Resolution

**Forward chaining:** Forward chaining is the process where it matches the set of conditions and infer results from these conditions. It is data driven approach using which it reaches (infers) to the goal condition.

Example: Given A (is true)

B  $\rightarrow$  C

A  $\rightarrow$  B

C  $\rightarrow$  D

Prove D is also true.

Solution: Starting from A, A is true then B is true (A  $\rightarrow$  B)

B is true then C is true (B  $\rightarrow$  C)

C is true then D is true Proved (C  $\rightarrow$  D)

**Backward chaining:** Backward chaining is the process where it performs backward search from goal to the conditions used to get the goal. It is goal driven approach using which it reaches to the initial condition.

Example: Given A (is true)

B  $\rightarrow$  C

A  $\rightarrow$  B

C  $\rightarrow$  D

Prove D is also true.

Solution: Starting from D,

Let D is true then C is true (C  $\rightarrow$  D)

C is true then B is true (B  $\rightarrow$  C)

B is true then A is true Proved (A  $\rightarrow$  B)

### 2.1.5 Resolution in FOL:

Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by a Mathematician John Alan Robinson in the year 1965. Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the conjunctive normal form or clausal form.

- **Clause:** Disjunction of literals (an atomic sentence) is called a clause. It is also known as a unit clause.
- **Conjunctive Normal Form:** A sentence represented as a conjunction of clauses is said to be conjunctive normal form or CNF.

#### Steps in Resolution:

- Conversion of facts into first-order logic.
- Negate the statement which needs to prove (proof by contradiction)
- Convert FOL statements into CNF
- Draw resolution graph (unification).

To better understand all the above steps, we will take an example in which we will apply resolution.

#### Example:

##### Facts:

All peoples who are graduating are happy

All happy people smile

Some one is graduating

Prove that “Is someone smiling?” using resolution

#### 1. Conversion of Facts into first order logic:

- All peoples who are graduating are happy  
 $\forall x(\text{Graduating}(x) \Rightarrow \text{happy}(x))$
- All happy people smile  
 $\forall x(\text{happy}(x) \Rightarrow \text{smile}(x))$
- Some one is graduating  
 $\exists x \text{ graduating}(x)$
- “Is someone smiling”  
 $\exists x \text{ smile}(x)$

#### 2. Negate the statement which needs to prove (proof by contradiction)

- We need to proof “Is someone smiling”.
- So negate the statement (proof by contradiction)  
 $\neg \exists x \text{ smile}(x)$

#### 3. Convert FOL statements into CNF

##### Steps to convert FOL to CNF

- Eliminate Implication
- Standardize variable
- Move Negation inwards



- Skolemization
- Drop Universal Quantifier

### Eliminate Implication

$$\alpha \Rightarrow \beta \equiv \neg \alpha \vee \beta$$

$$\alpha \Leftrightarrow \beta \equiv (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$$

Example:

Eliminate Implication:  $\alpha \Rightarrow \beta \equiv \neg \alpha \vee \beta$

- $\forall x(\neg \text{Graduating}(x) \vee \text{happy}(x))$
- $\forall x(\neg \text{happy}(x) \vee \text{smile}(x))$
- $\exists x \text{ graduating}(x)$
- $\neg \exists x \text{ smile}(x)$

### Standardize variable

$\forall x \text{ regular}(x)$

$\forall x \text{ regular}(x)$

$\exists x \text{ busy}(x)$

$\exists y \text{ busy}(y)$

$\exists x \text{ attentive}(x)$

$\exists z \text{ attentive}(z)$

Example:

- $\forall x(\neg \text{Graduating}(x) \vee \text{happy}(x))$
- $\forall y(\neg \text{happy}(y) \vee \text{smile}(y))$
- $\exists z \text{ graduating}(z)$
- $\neg \exists w \text{ smile}(w)$

### Move Negation inwards

- $\neg (\forall x P(x)) \equiv \exists x \neg P(x)$
- $\neg (\exists x P(x)) \equiv \forall x \neg P(x)$
- $\neg (\alpha \vee \beta) \equiv \neg \alpha \wedge \neg \beta$
- $\neg (\alpha \wedge \beta) \equiv \neg \alpha \vee \neg \beta$
- $\neg (\neg \alpha) \equiv \alpha$

Example:

- $\forall x(\neg \text{Graduating}(x) \vee \text{happy}(x))$
- $\forall y(\neg \text{happy}(y) \vee \text{smile}(y))$
- $\exists z \text{ graduating}(z)$
- $\forall w \neg \text{smile}(w)$

**Skolemization:** Removing of Existential Quantifier and replacing it by Skolem Constants

$\exists(y) \text{ busy}(y)$

$\exists(x) \text{ attentive}(x)$

After Skolemization it changes into:

$\text{busy}(A)$

$\text{attentive}(B)$

Example:

$\forall x(\neg \text{Graduating}(x) \vee \text{happy}(x))$

$\forall y(\neg \text{happy}(y) \vee \text{smile}(y))$

After Skolemization it changes into:

$\text{graduating}(A)$

$\forall w \neg \text{smile}(w)$

#### Drop Universal Quantifier:

$\forall(y) \text{ busy}(y)$

$\forall(x) \text{ attentive}(x)$

After dropping it changes into:

$\text{busy}(y)$

$\text{attentive}(x)$

Example:

- $\neg \text{Graduating}(x) \vee \text{happy}(x)$
- $\neg \text{happy}(y) \vee \text{smile}(y)$
- $\text{graduating}(A)$
- $\neg \text{smile}(w)$

Now the sentences are in CNF

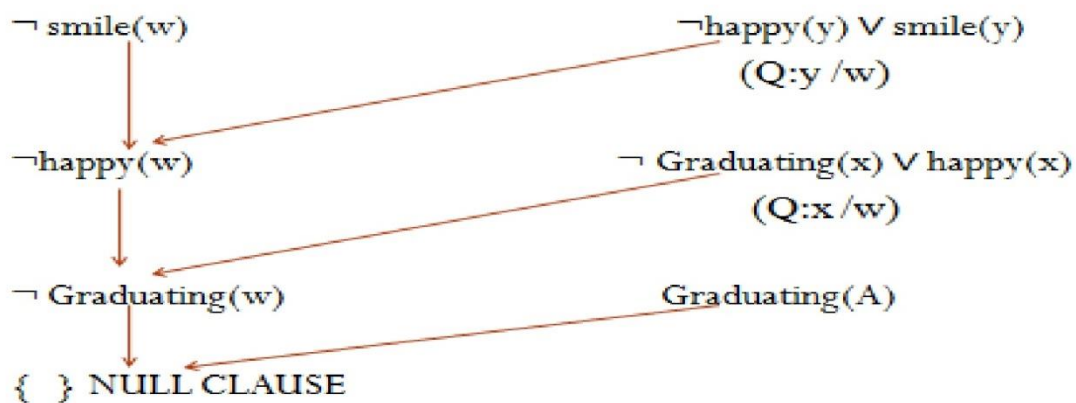
#### 4. Resolution Graph

- If fact F is to be proved then it start with  $\neg F$
- It contradicts all other rules in KB
- The process stop when it returns Null clause

Example:

- $\neg \text{Graduating}(x) \vee \text{happy}(x)$
- $\neg \text{happy}(y) \vee \text{smile}(y)$
- $\text{graduating}(A)$
- $\neg \text{smile}(w)$

#### Resolution Graph:



Hence Someone is Smiling

### 2.2.1 Semantic Network:

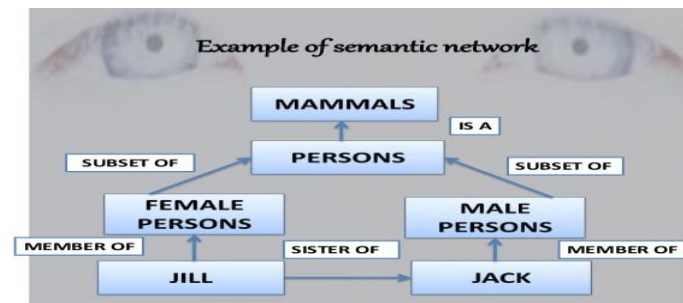
Semantic nets were originally proposed in the early 1960 by M. Ross Quillian to represent the meaning of English words. Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks. This network 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.

#### Representation:

Semantic network representation consists of mainly two types of relations:

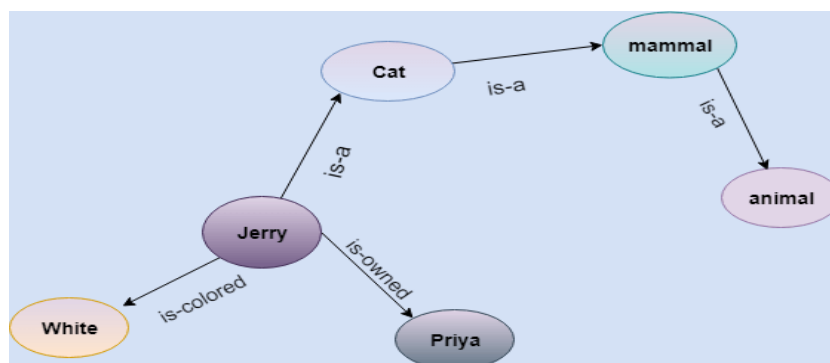
1. IS-A relation (Inheritance)
2. Kind-of-relation

Example 1: Following are some statements which we need to represent in the form of nodes and arcs.



#### Example 2: Statements

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



#### Advantages of Semantic Networks:

- Semantic networks are a natural representation of knowledge.
- Semantic networks convey meaning in a transparent manner because it is easy to visualize and understand

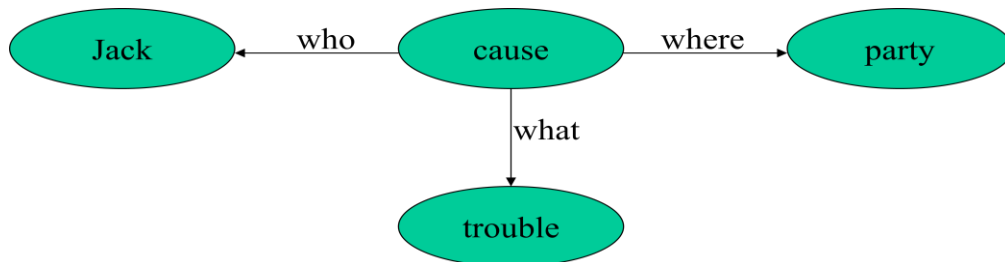
- These networks are simple and easily understandable.
- The knowledge engineer can arbitrarily defined the relationships.
- Related knowledge is easily categorised.
- Efficient in space requirements.
- Node objects represented only once.

### Limitations of Semantic Networks:

- The limitations of conventional semantic networks were studied extensively by a number of workers in AI.
- 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.
- 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.
- These types of representations are inadequate as they do not have any equivalent quantifier, e.g., for all, for some, none, etc.
- Many believe that the basic notion is a powerful one and has to be complemented by, for example, **logic** to improve the notion's expressive power and robustness.
- Others believe that the notion of semantic networks can be improved by incorporating **reasoning** used to describe events.
- Binary relations are usually easy to represent, but sometimes is difficult.

E.g. try to represent the sentence:

"Jack caused trouble to the party".



- Other problematic statements. . .
  - negation "John does not go fishing";
  - disjunction "John eats pizza or fish and chips";
  - ...
- Quantified statements are very hard for semantic nets. E.g.:
  - "Every dog has bitten a delivery boy"
  - "Every dog has bitten every delivery boy"

**Solution:** Partitioned semantic networks can represent quantified statements.

### Partitioned Semantic Networks:

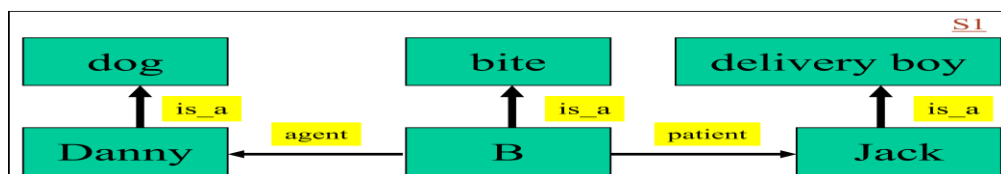
Hendrix developed the so-called partitioned semantic network to represent the difference between the description of an individual object or process and the description of a set of objects. The set description involves quantification. Hendrix partitioned a semantic network whereby a semantic network, loosely speaking, can be divided into one or more networks for the description of an individual.

The central idea of partitioning is to allow groups, nodes and arcs to be bundled together into units called spaces – fundamental entities in partitioned networks, on the same level as nodes and arcs. Every node and every arc of a network belongs to (or lies in/on) one or more spaces. Some spaces are used to encode 'background information' or generic relations; others are used to deal with specifics called 'scratch' space.

Suppose that we wish to make a specific statement about a dog, Danny, who has bitten a delivery boy, Peter:

"Danny the dog bit Jack the delivery boy"

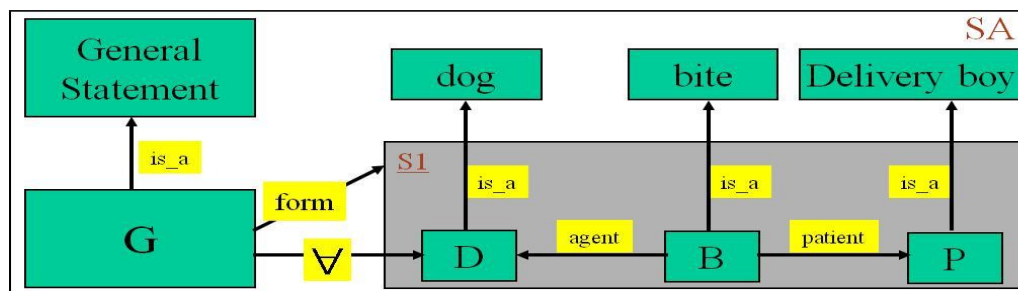
Hendrix's Partitioned network would express this statement as an ordinary semantic network:



Suppose that we now want to look at the statement:

"Every dog has bitten a delivery boy"

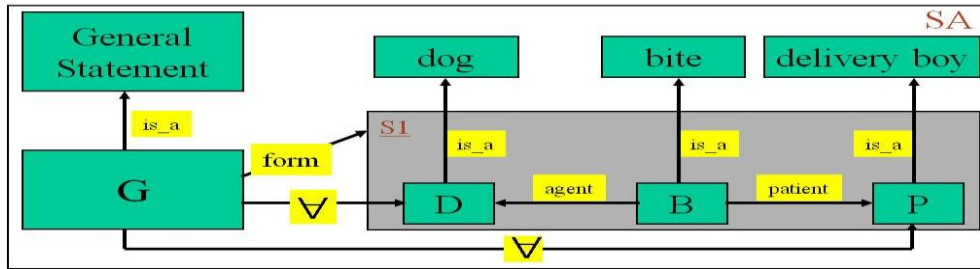
∀



Hendrix partitioned semantic network now comprises two partitions SA and S1. Node G is an **instance** of the special class of general statements about the world comprising link statement, **form**, and one **universal quantifier**

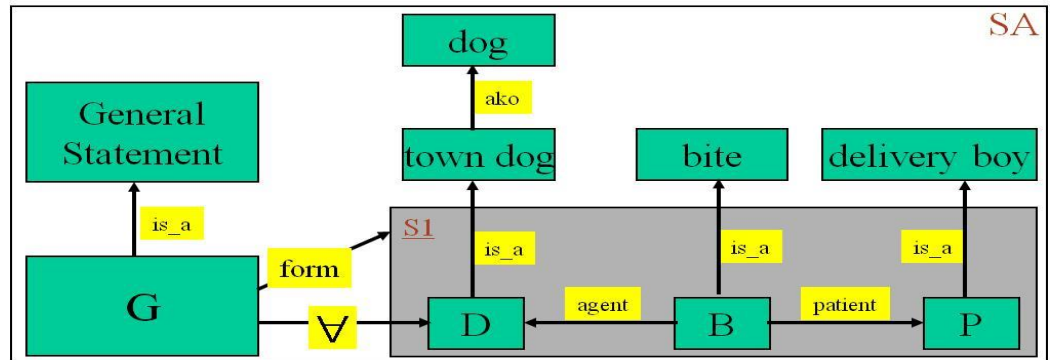
Suppose that we now want to look at the statement:

"Every dog has bitten every delivery boy"



Suppose that we now want to look at the statement:

"Every dog in town has bitten the delivery boy"



Note: 'ako' = 'A Kind Of'

The partitioning of a semantic network renders them more logically adequate, in that one can distinguish between individuals and sets of individuals and indirectly more heuristically adequate by way of controlling the search space by delineating semantic networks.

Hendrix's partitioned semantic networks-oriented formalism has been used in building natural language front-ends for data bases and for programs to deduct information from databases.

### 2.2.2 Frames:

A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world. A frame is analogous to a record structure, corresponding to the fields and values of a record are the slots and slot fillers of a frame

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.

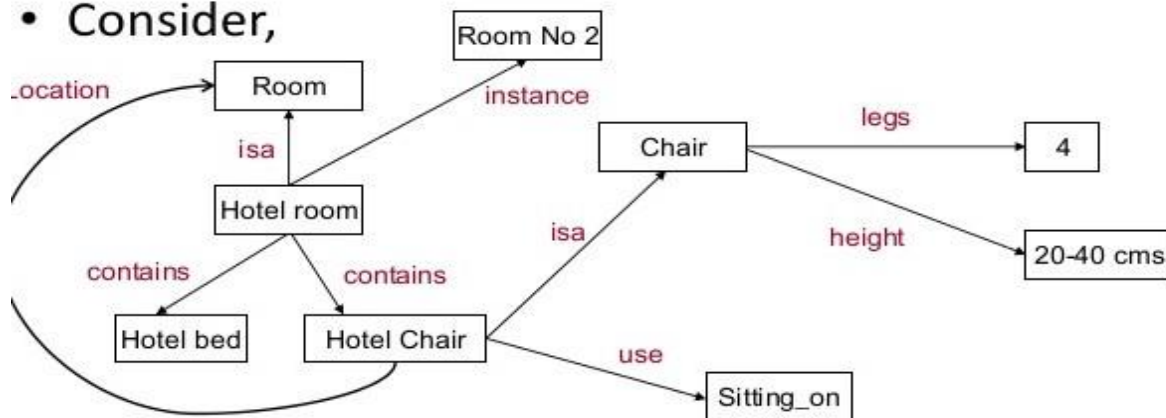
Example: IF-NEEDED facts are called when data of any particular slot is needed.

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.

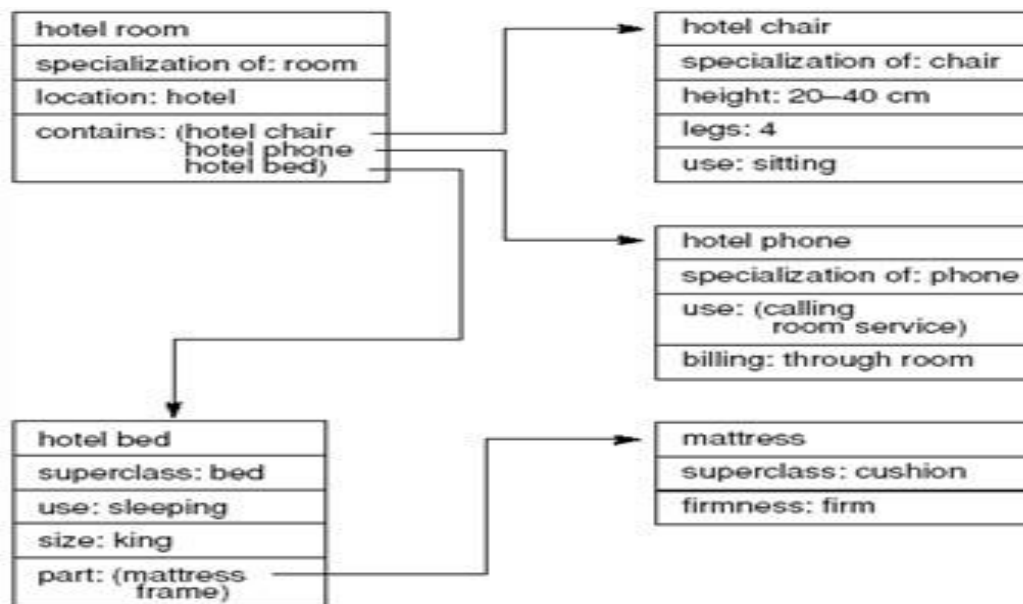
Frames are derived from semantic networks and later evolved into our modern-day classes and objects. A single frame is not much useful. Frames system consist of a collection of frames which are connected. In the frame, knowledge about an object or event can be stored together in the knowledge base. The frame is a type of technology which is widely used in various applications including Natural language processing and machine visions.

### Graphs Representation of Hotel Room:

#### • Consider,



### Frame Description of Hotel Room:



### Advantages of Frame representation:

- The frame knowledge representation makes the programming easier by grouping the related data.
- The frame representation is comparably flexible and used by many applications in AI.
- It is very easy to add slots for new attribute and relations.

- It is easy to include default data and to search for missing values.
- Frame representation is easy to understand and visualize.

### **Limitations of Frame representation:**

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

### **2.3.1 Planning Agent:**

- An agent interacts with the world via perception and actions
- Perception involves sensing the world and assessing the situation and creating some internal representation of the world
- Actions are what the agent does in the domain. Planning involves reasoning about actions that the agent intends to carry out
- *Planning* is the reasoning side of acting
- This reasoning involves the representation of the world that the agent has, as also the representation of its actions.
- Hard constraints where the objectives *have to* be achieved completely for success
- The objectives could also be soft constraints, or *preferences*, to be achieved as much as possible

### **Planning:**

Planning is arranging a sequence of actions to achieve a goal

○ or

Planning is the task of coming up with a sequence of actions that will achieve the goal

### **Planning Problem;**

- Find a sequence of actions that achieves a given goal when executed from a given initial world state
- That is, given
  - a set of operator descriptions defining the possible primitive actions by the agent,
  - an initial state description, and
  - a goal state description or predicate,
  - compute a plan, which is
  - a sequence of operator instances which after executing them in the initial state changes the world to a goal state
- Goals are usually specified as a conjunction of goals to be achieved

### **2.3.2: Planning Vs Problem Solving:**

- Planning and problem solving methods can often solve the same sorts of problems
- Planning is more powerful and efficient because of the representations and methods used
- States, goals, and actions are decomposed into sets of sentences (usually in first-order logic)
- Search often proceeds through *plan space* rather than *state space* (though there are also state-space planners)
- Sub goals can be planned independently, reducing the complexity of the planning problem

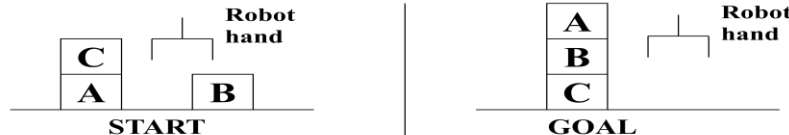
### **Example: Block World problem**



Block world problem assumptions

- Square blocks of same size
- Blocks can be stacked one upon another.
- Flat surface (table) on which blocks can be placed.
- Robot arm that can manipulate the blocks. It can hold only one block at a time.

In block world problem, the state is described by a set of predicates representing the facts that were true in that state. One must describe for every action, each of the changes it makes to the state description. In addition, some statements that everything else remains unchanged is also necessary.



**Sequence of actions :**

1. **Grab C**
2. **Pickup C**
3. **Place on table C**
4. **Grab B**
5. **Pickup B**
6. **Stack B on C**
7. **Grab A**
8. **Pickup A**
9. **Stack A on B**

### 2.3.3 STRIPS Mechanism:

One such mechanism was used in early robot problem solving system named STRIPS (developed by Fikes, 1971).

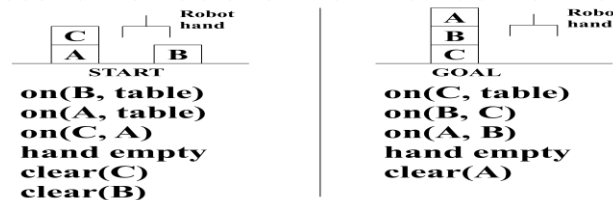
In this approach, each operation is described by three lists.

- Pre\_Conc list contains predicates which have to be true before operation.
- ADD list contains those predicates which will be true after operation
- DELETE list contain those predicates which are no longer true after operation

Predicates not included on either of these lists are assumed to be unaffected by the operation. Frame axioms are specified implicitly in STRIPS which greatly reduces amount of information stored.

### Example : Blocks World

**•STRIPS : A planning system – Has rules with precondition deletion list and addition list**



**Rules:**

- **RI : pickup(x)**  
**Precondition & Deletion List:** hand empty, on(x,table), clear(x)  
**Add List:** holding(x)

- **R2 : *putdown(x)***  
**Precondition & Deletion List:** holding(x)  
**Add List:** hand empty, on(x,table), clear(x)
- **R3 : *stack(x,y)***  
**Precondition & Deletion List:** holding(x), clear(y)  
**Add List:** on(x,y), clear(x), hand empty
- **R4 : *unstack(x,y)***  
**Precondition & Deletion List:** on(x,y), clear(x), hand empty  
**Add List:** holding(x), clear(y)

### Plan for the block world problem:

For the given problem, Start  $\rightarrow$  Goal can be achieved by the following sequence :

1. Unstack(C,A)
2. Putdown(C)
3. Pickup(B)
4. Stack(B,C)
5. Pickup(A)
6. Stack(A,B)

Execution of a plan: achieved through a data structure called Triangular Table.

# Triangular Table

1	on(C,A) clear(C) hand empty	unstack(C,A)					
2		holding(C)	putdown(C)				
3	on(B,table)		hand empty	pickup(B)			
4			clear(C)	holding(B)	stack(B,C)		
5	on(A,table)	clear(A)			hand empty	pickup(A)	
6					clear(B)	holding(A)	stack(A,B)
7			on(C,table)		on(B,C)		on(A,B) clear(A)
	0	1	2	3	4	5	6

- For n operations in the plan, there are :
  - (n+1) rows :  $1 \rightarrow n+1$
  - (n+1) columns :  $0 \rightarrow n$
- At the end of the  $i^{\text{th}}$  row, place the  $i^{\text{th}}$  component of the plan.
- The row entries for the  $i^{\text{th}}$  step contain the pre-conditions for the  $i^{\text{th}}$  operation.
- The column entries for the  $j^{\text{th}}$  column contain the add list for the rule on the top.
- The  $\langle i,j \rangle^{\text{th}}$  cell (where  $1 \leq i \leq n+1$  and  $0 \leq j \leq n$ ) contain the pre-conditions for the  $i^{\text{th}}$  operation that are added by the  $j^{\text{th}}$  operation.

- The first column indicates the starting state and the last row indicates the goal state.

### Goal Stack Planning:

Goal Stack planning (in short GSP) is one of the most important and simplest planning algorithms, which is specifically used by **STRIPS**. The stack is used in an algorithm to hold the action and satisfy the goal.

A knowledge base is used to hold the current state, actions. This approach uses a Stack for plan generation. The stack can contain Sub-goal and actions described using predicates. The Sub-goals can be solved one by one in any order. Goal stack is similar to a node in a search tree, where the branches are created if there is a choice of an action.

### Goal Stack planning algorithm:

- Start by pushing the original goal on the stack. Repeat this until the stack becomes empty. If stack top is a compound goal, then push its unsatisfied sub goals on the stack.
- If stack top is a single unsatisfied goal then, replace it by an action and push the action's precondition on the stack to satisfy the condition.
- If stack top is an action, pop it from the stack, execute it and change the knowledge base by the effects of the action.
- If stack top is a satisfied goal, pop it from the stack.

	Rules	Precondition	Action
1	<i>pickup(A)</i>	hand empty, on(A,table), clear(A)	Holding(A)
2	<i>putdown(A)</i>	Holding(A)	hand empty, on(A,table), clear(A)
3	<i>Stack(A,B)</i>	holding(A), clear(B)	on(A,B), clear(A), hand empty
4	<i>Unstack(A,B)</i>	on(A,B), clear(A), hand empty	holding(A), clear(B)

### 2.3.4 Total Order planning:

- Forward / backward state-space searches are forms of totally ordered plan search.
- Explore only strictly linear sequences of actions, directly connected to the start or goal state.
- Cannot take advantages of problem decomposition

### Partial Order planning:

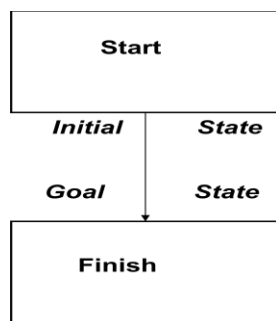
- A **linear planner** builds a plan as a **totally ordered sequence** of plan steps
- A **non-linear planner (aka partial-order planner)** builds up a plan as a set of steps with some temporal constraints
  - constraints like  $S1 < S2$  if step S1 must come before S2.
- One **refines** a partially ordered plan (POP) by either:

- **adding a new plan step**, or
  - **adding a new constraint** to the steps already in the plan.
- Works on several sub goals independently
  - Solves them with sub plans
  - Combine the sub plans
  - Flexibility in ordering the sub plans
  - Least commitment strategy:  
delaying a choice during search

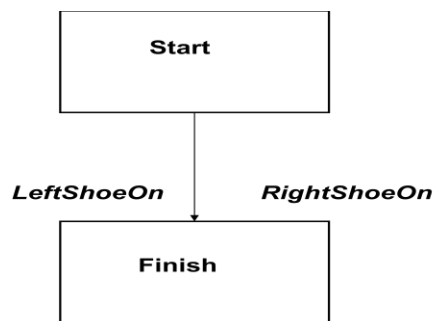
**POP Example-** putting on a pair of shoes

- **Goal( RightShoeOn, LeftShoeOn)**
- **Init()**
- **Actions: RightShoe**
  - **PRECOND:** RightSockOn
  - **EFFECT:** RightShoeOn
- **Actions: RightSock**
  - **PRECOND:** None
  - **EFFECT:** RightSockOn
- **Actions: LeftShoe**
  - **PRECOND:** LeftSockOn
  - **EFFECT:** LeftShoeOn
- **Actions: LeftSock**
  - **PRECOND:** None
  - **EFFECT:** LeftSockOn

**A simple graphical notation:**

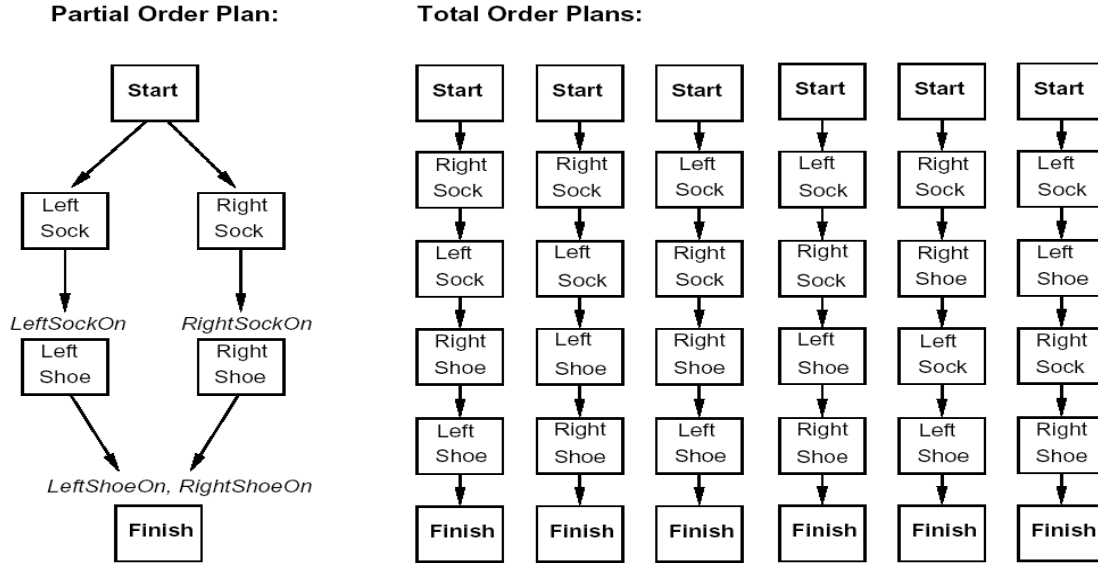


(a)



(b)

## Partial Order Plan vs. Total Order Plan:



The space of POPs is smaller than TOPs and hence involve less search because in TOP it requires six corresponding linearization.

### How to define Partial order planning:

- A set of actions, that make up the steps of the plan.
- A set of ordering constraints  $A < B$   
A before B
- A set of casual links  $A \xrightarrow{p} B$
- A achieves P for B  $\text{RightSock} \xrightarrow{\text{RightSockOn}} \text{RightShoe}$
- A set of open preconditions:  
A precondition is open, if it is not achieved by some action in the plan

### The Initial Plan:

- Initial Plan contains:
- Start:  
PRECOND: None  
EFFECT: Add all propositions that are initially true.
- Finish:  
PRECOND: Goal State  
EFFECT: None
- Ordering constraints  $\text{Start} < \text{Finish}$
- Casual Links:  $\{\}$
- Open preconditions:  
 $\{\text{preconditions of finish}\}$

- **Successor Function:** Arbitrarily picks one open precondition  $p$  on an action  $B$  and generates a successor plan, for every possible consistent way of choosing an action  $A$ , that achieves  $p$ .
- **Consistency:**

Casual Link  $A \xrightarrow{p} B$  and the ordering constraints are added  $()$  ( $A < B$  Start  $< A$   $A < Finish$ )

- **Resolve Conflict :** add  $B < C$  or  $C < A$
- **Goal Test:**  
There are no open preconditions

## Example: Final Plan

The final plan has the following components:

- **Actions:** {RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish}
- **Orderings:** { RightSock < RightShoe, LeftSock < LeftShoe}
- **Open preconditions:** {}
- **Links**



### 2.3.5 Hierarchical (HTN) Planning:

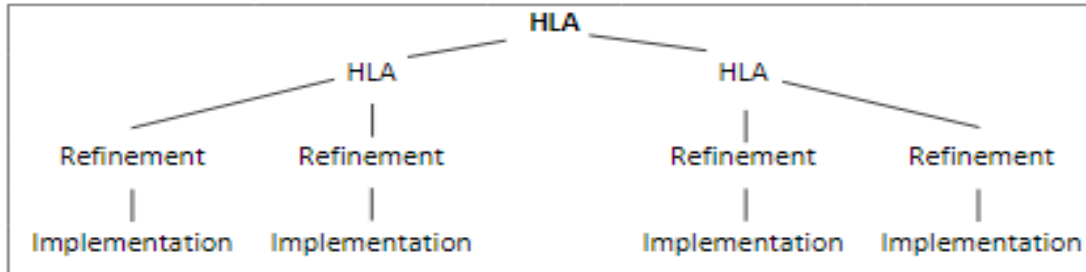
In order to solve hard problems, a problem solver may have to generate long plans, it is important to be able to eliminate some of the details of the problem until a solution that addresses the main aspect is found. Then an attempt can be made to fill the appropriate details. Early attempts to do this involved the use of macro operators.

But in this approach, no details were eliminated from actual descriptions of the operators.

- As an example, suppose you want to visit a friend in London but you have a limited amount of cash to spend. First preference will be to find the airfares, since finding an affordable flight will be the most difficult part of the task. You should not worry about of about your drive way, planning a route to the airport etc, until you are sure you have a flight.

#### Structure of Hierarchical (HTN) Planning :

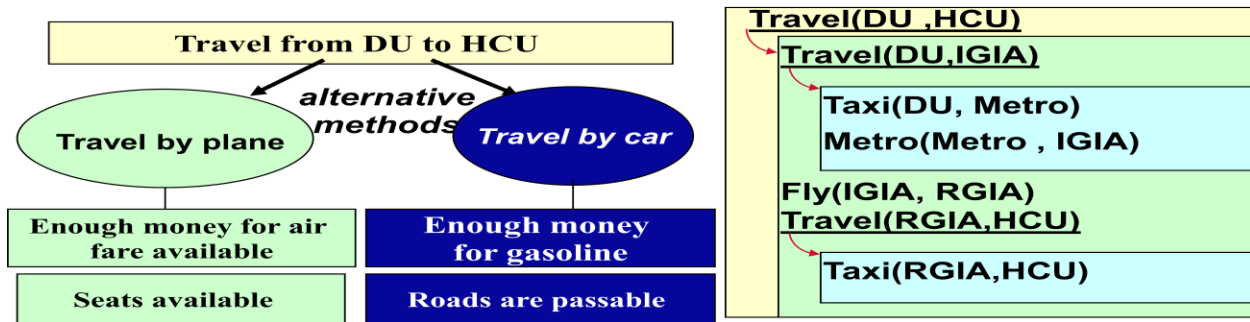
Hierarchical Task Networks (from here on abbreviated as HTN) is a hierarchical planning technique where actions are divided into different levels, or hierarchies. There are different types of actions residing within these hierarchies, one of them being High Level Actions (from here on abbreviated as HLA). These HLAs could be any action that you can divide into smaller actions called refinements. These refinements may be sequences of actions or even other HLAs. Refinements in turn are broken down into implementations (also called primitive actions) which refers to any action that has no further refinements or implementations.



Figur 1. Example of basic hierarchical structure.

## Hierarchical (HTN) Planning

**Principle:** Complex tasks are decomposed into simpler tasks. The goal is to decompose all the tasks into *primitive* tasks, which define actions that change the world.



### Advantages of Hierarchical (HTN) Planning:

- The main advantage of HTN planners is their ability to reason on a higher level of abstraction compared to classical planning.
- As aforementioned, in a standard scenario, HTN planners using angelic search have the ability to invent a plan using HLAs alone without considering the implementations of the plan on beforehand which is very effective when considering the computational aspect.
- Hierarchical plans are generally easy for humans to understand. - A possible explanation for this is that humans are believed to plan on a higher level of abstraction, much like HTN planners do.

### Disadvantages of Hierarchical (HTN) Planning:

- The general main disadvantage of HTN planners is that many of them require a deterministic environment and are unable to handle uncertain outcomes of actions.
- Replanning or repairing features are however sub optimal when viewed from a computational aspect, even if a planner is able to repair a plan during execution, it may hold up production or construction while replanning occurs.
- Furthermore, it is sometimes hard to compute exact reachable sets, leading to wiggly sets, which complicate the planning process.

# UNIT-3



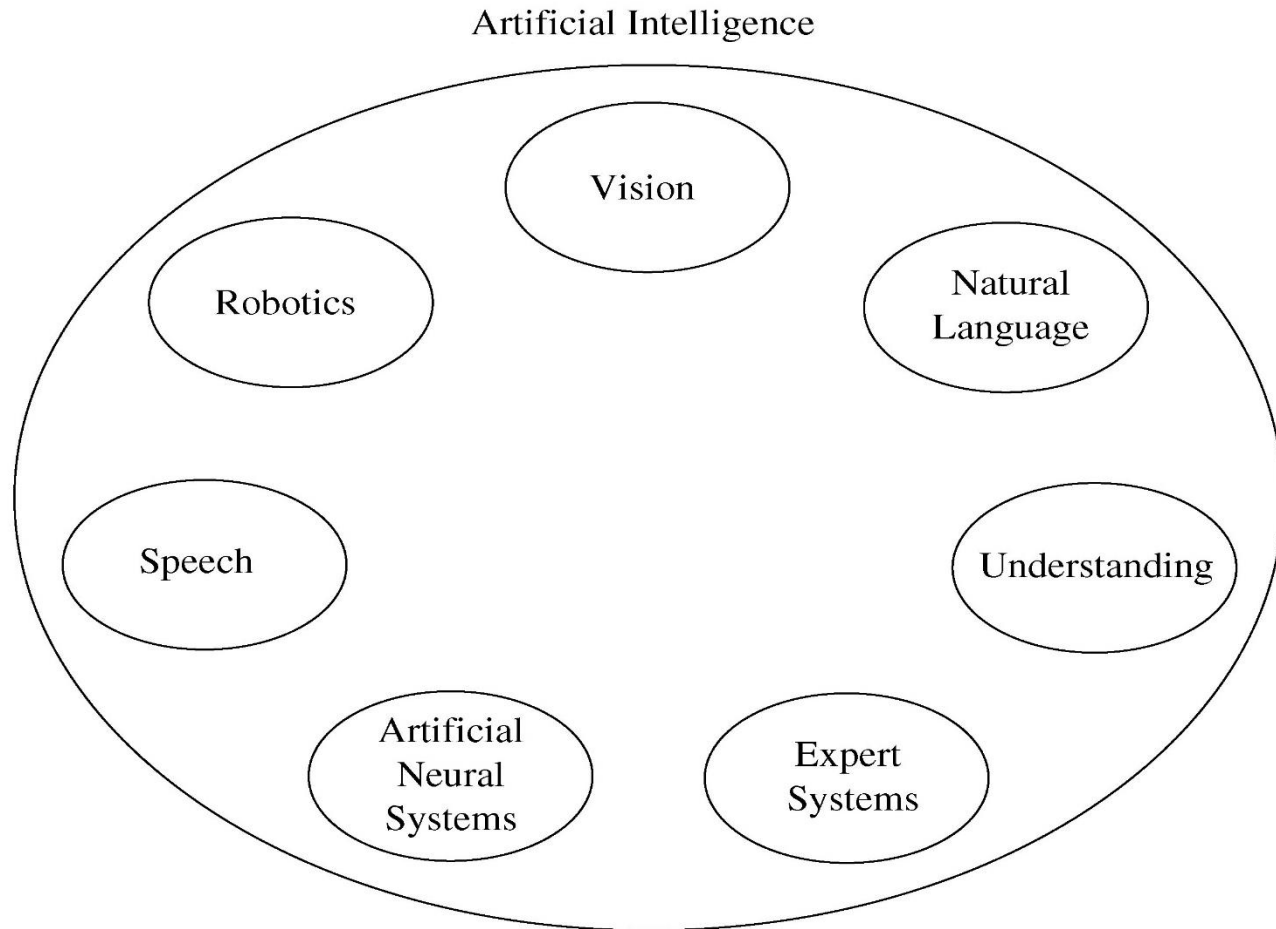
# Expert Systems

- An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert.

(Or)

- The expert systems are the computer applications developed to solve complex problems in a particular domain, at the level of extra-ordinary human intelligence and expertise.
- 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.

# Areas of Artificial Intelligence



# Expert Systems

- Expert Systems 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.

# Characteristics of Expert Systems

- **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.

# Capabilities of Expert Systems

The expert systems are capable of –

- Advising
- Instructing and assisting human in decision making
- Demonstrating
- Deriving a solution
- Diagnosing
- Explaining
- Interpreting input
- Predicting results
- Justifying the conclusion
- Suggesting alternative options to a problem

# In capabilities of Expert Systems

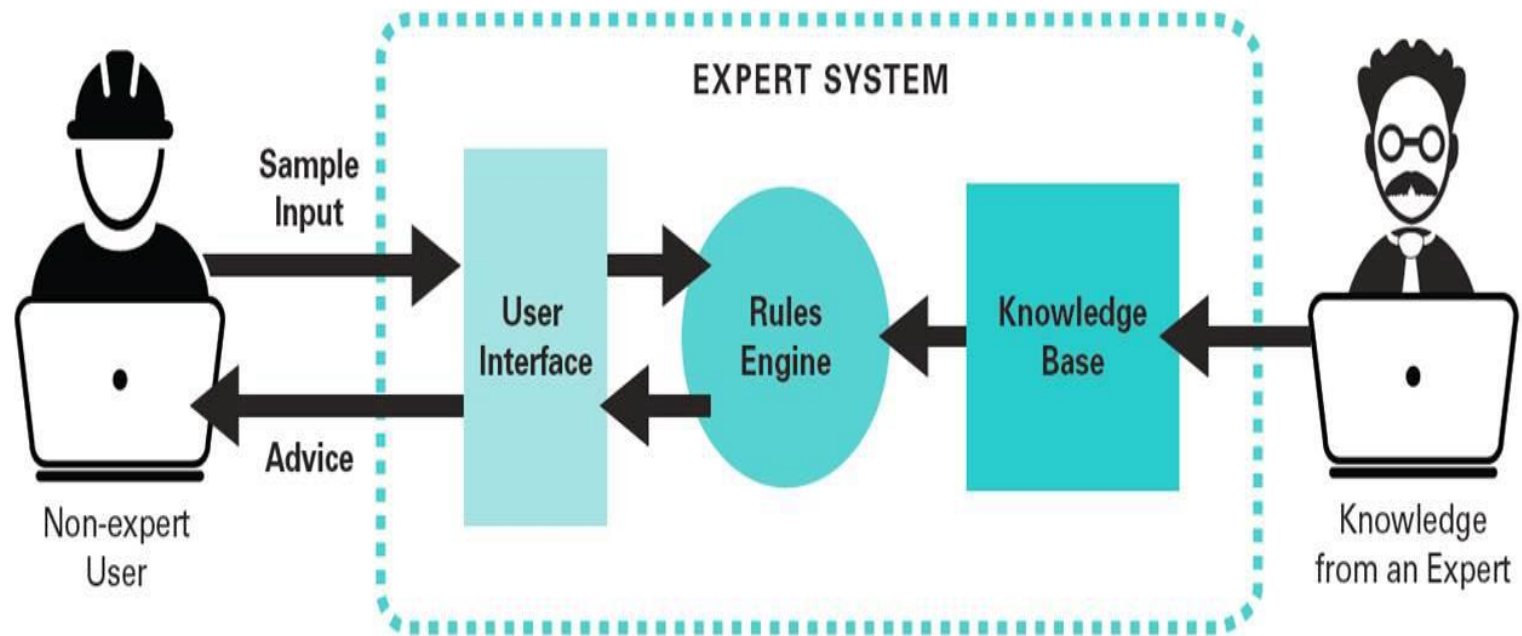
They are incapable of –

- Substituting human decision makers
- Possessing human capabilities
- Producing accurate output for inadequate knowledge base
- Refining their own knowledge

# Components of Expert Systems

The components of ES include –

- Knowledge Base
- Inference Engine
- User Interface



The user inputs information about: the equipment,  
the component  
and the oil being tested . . . The Expert System does the rest.



# Components of Expert Systems

## Knowledge Base:

- It contains domain-specific and high-quality knowledge.
- Knowledge is required to exhibit intelligence. The success of any ES majorly depends upon the collection of highly accurate and precise knowledge.

## What is Knowledge?

- The data is collection of facts. The information is organized as data and facts about the task domain. **Data, information, and past experience** combined together are termed as knowledge.

# Components of Expert Systems

## Components of Knowledge Base:

The knowledge base of an ES is a store of both, factual and heuristic knowledge.

- **Factual Knowledge** – It is the information widely accepted by the Knowledge Engineers and scholars in the task domain.
- **Heuristic Knowledge** – It is about practice, accurate judgement, one's ability of evaluation, and guessing.

# Components of Expert Systems

## Inference Engine(Rules of Engine)

- The inference engine is known as the brain of the expert system as it is the main processing unit of the system. It applies inference rules to the knowledge base to derive a conclusion or deduce new information. It helps in deriving an error-free solution of queries asked by the user.
- With the help of an inference engine, the system extracts the knowledge from the knowledge base.

There are two types of inference engine:

- **Deterministic Inference engine:** The conclusions drawn from this type of inference engine are assumed to be true. It is based on **facts** and **rules**.

# Components of Expert Systems

- **Probabilistic Inference engine:** This type of inference engine contains uncertainty in conclusions, and based on the probability.

Inference engine uses the below modes to derive the solutions:

- **Forward Chaining:** It starts from the known facts and rules, and applies the inference rules to add their conclusion to the known facts.
- **Backward Chaining:** It is a backward reasoning method that starts from the goal and works backward to prove the known facts.

# Components of Expert Systems

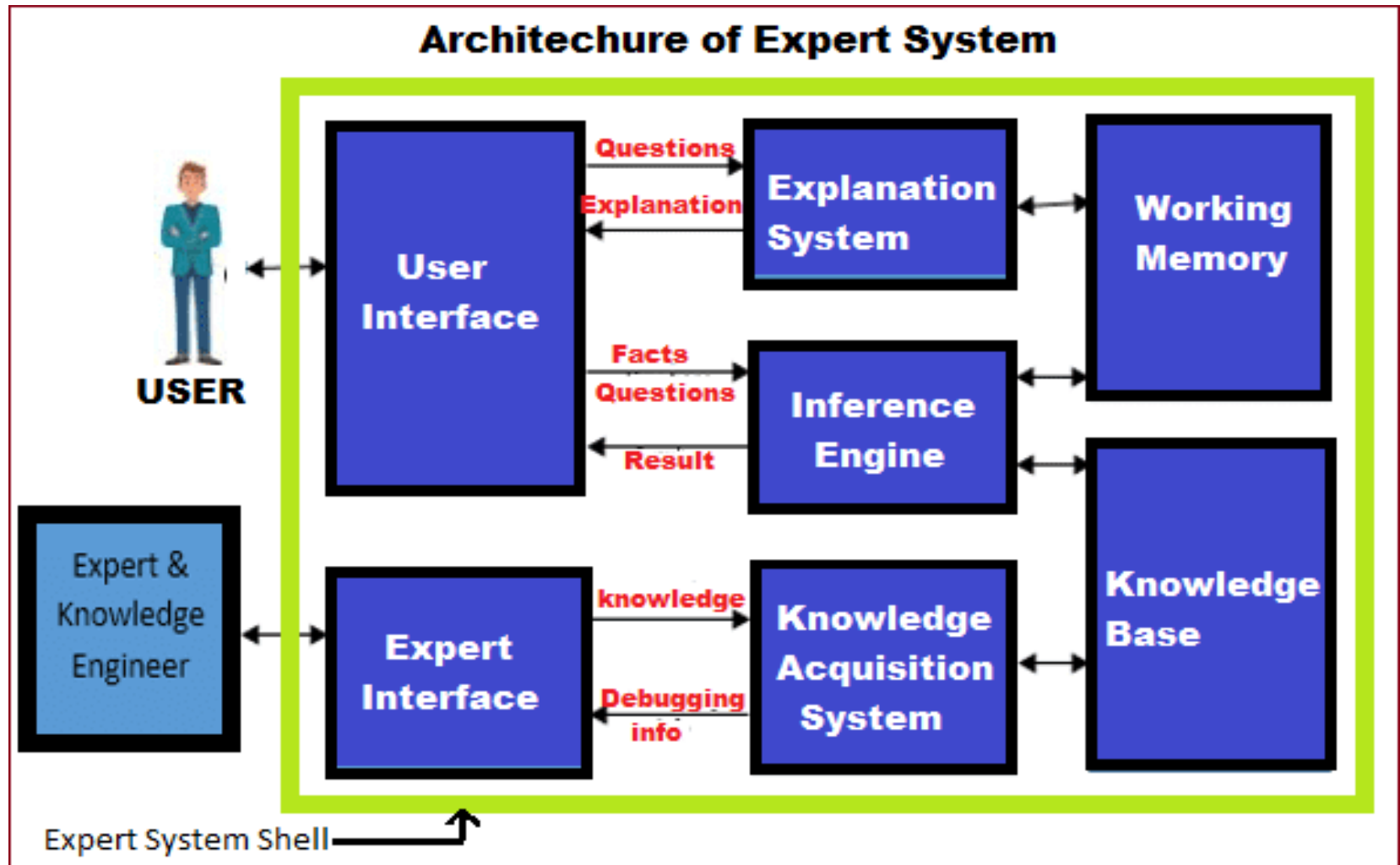
## User Interface:

- With the help of a user interface, the expert system interacts with the user, takes queries as an input in a readable format, and passes it to the inference engine. After getting the response from the inference engine, it displays the output to the user.
- In other words, **it is an interface that helps a non-expert user to communicate with the expert system to find a solution.**

# Popular Examples of 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.

# Architecture of Expert System



# Architecture of Expert System

- **Knowledge Base:** It is warehouse of special heuristics or rules, which are used directly by knowledge, facts (productions). It has knowledge that is needed for understanding, formulating, & problem solving.
- **Working Memory:** It helps to describe the current running problem and record intermediate output.

Records Intermediate Hypothesis & Decisions: 1. Plan, 2. Agenda, 3. Solution

- **Inference Engine:** It is important part of expert system which helps to manage entire structure of expert system, and it delivers to different methodology for reasoning.
- **Explanation System:** It helps to trace responsibility and justify the behavior of expert system by firing questions and answers, such as Why, How, What, Where, When, Who.



# Architecture of Expert System

- **User Interface:** It allows users to insert their queries with using own Natural Language Processing otherwise menus & graphics.
- **Knowledge Engineer:** Main objective of this engineer is to design system for specific problem domain with using of expert system shell.
- **Users:** They are non expert person who want to seek direct advice.
- **Expert system shell:** It contains the special software development environment, and it has basic components of expert system such as – Knowledge-based management system, Workplace, Explanation facility, Reasoning capacity, Inference engine, user interface.
- This shell is linked along with pre-defined method for designing different applications through configuring of those components.

# Phases in building Expert System

The following points highlight the five main phases to develop an expert system.

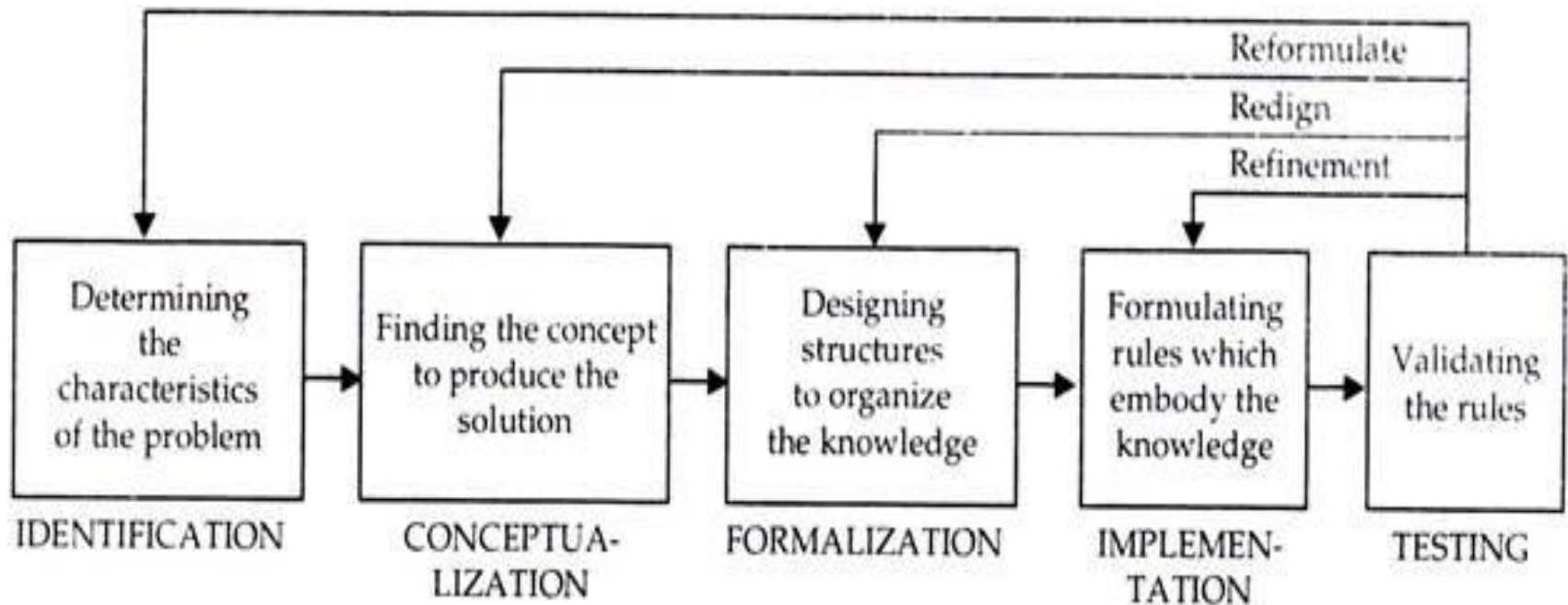
The phases are:

1. Identification
2. Conceptualization
3. Formalization (Designing)
4. Implementation
5. Testing (Validation, Verification and Maintenance).

# Phases in building Expert System

- A knowledge engineer is an AI specialist, perhaps a computer scientist or programmer, who is skilled in the ‘Art’ of developing expert systems.
- You don’t need a degree in “knowledge engineering” to call yourself a knowledge engineer; in fact, nearly everyone who has ever contributed to the technical side of the expert system development process could be considered a knowledge engineer.
- A domain expert is an individual who has significant expertise in the domain of the expert system being developed.
- It is not critical that the domain expert understand AI or expert systems; that is one of the functions of the knowledge engineer.

# Phases in building Expert System



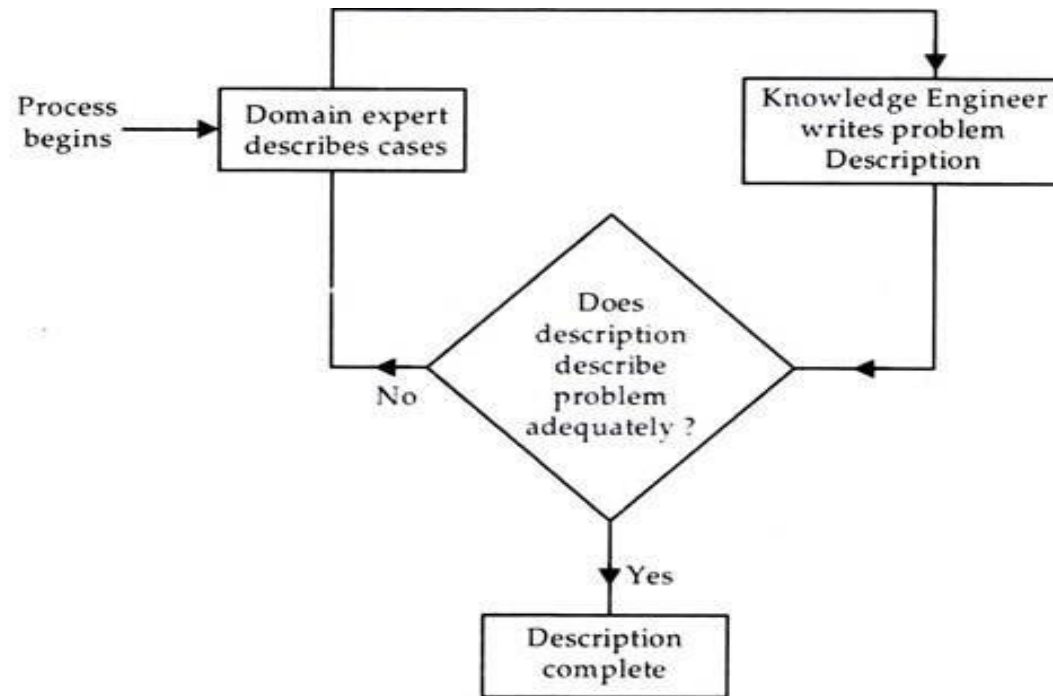
*Fig. 12.8. The five stages of expert system development.*

The knowledge engineer and the domain expert usually work very closely together for long periods of time throughout the several stages of the development process.

# 1. Identification Phase

- To begin, the knowledge engineer, who may be unfamiliar with this particular domain, consults manuals and training guides to gain some familiarity with the subject. Then the domain expert describes several typical problem states.
- The knowledge engineer attempts to extract fundamental concepts from the similar cases in order to develop a more general idea of the purpose of the expert system.
- After the domain expert describes several cases, the knowledge engineer develops a 'first-pass' problem description.
- Typically, the domain expert may feel that the description does not entirely represent the problem.
- The domain expert then suggests changes to the description and provides the knowledge engineer with additional examples to illustrate further the problem's fine points.

# 1. Identification Phase



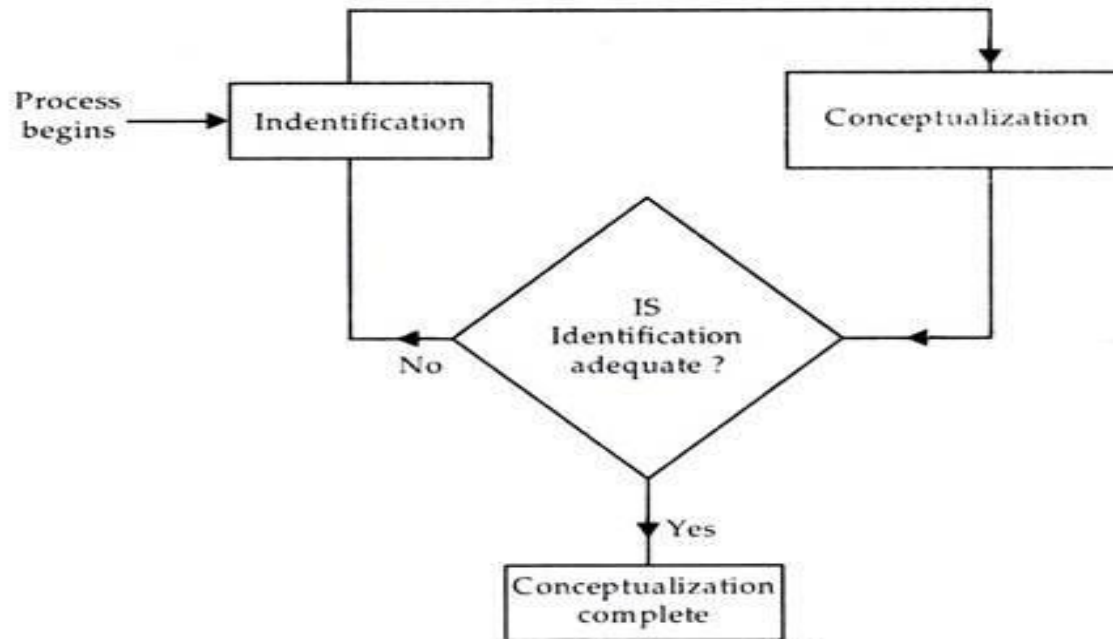
**Fig. 12.9.** *The iterative process of identifying the problem which the expert system is to solve.*

Next, the knowledge engineer revises the description, and the domain expert suggests further changes. This process is repeated until the domain expert is satisfied that the knowledge engineer understands the problems and until both are satisfied that the description adequately portrays the problem which the expert system is expected to solve.

## 2. Conceptualisation Phase

- In the conceptualisation stage, the knowledge engineer frequently creates a diagram of the problem to depict graphically the relationships between the objects and processes in the problem domain.
- It is often helpful at this stage to divide the problem into a series of sub-problems and to diagram both the relationships among the pieces of each sub-problem and the relationships among the various sub-problems.
- As in the identification stage, the conceptualisation stage involves a circular procedure of iteration and reiteration between the knowledge engineer and the domain expert. When both agree that the key concepts-and the relationships among them-have been adequately conceptualised, this stage is complete.

## 2. Conceptualisation Phase



**Fig. 12.10.** *The iterative relationship between the identification and conceptualisation stages of expert system development.*

Not only is each stage in the expert system development process circular, the relationships among the stages may be circular as well. Since each stage of the development process adds a level of detail to the previous stage, any stage may expose a weakness in a previous stage



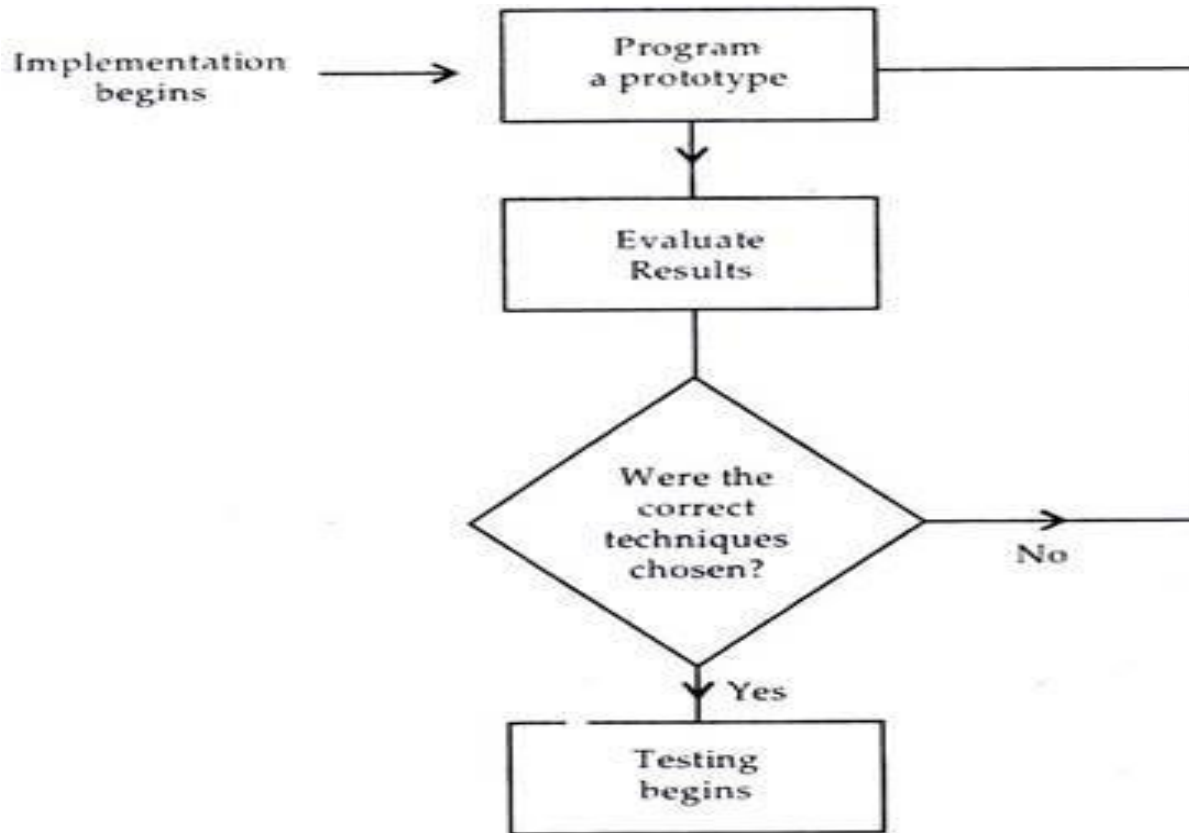
# 3. Formalisation (Designing) Phase

- The formalisation process is often the most interactive stage of expert system development, as well as the most time consuming.
- The knowledge engineer must develop a set of rules and ask the domain expert if those rules adequately represent the expert's knowledge.
- The domain expert reviews the rules proposed by the knowledge engineer and suggests changes, which are then incorporated into the knowledge base by the knowledge engineer.
- As in the other development stages, this process also is iterative: the rule review is repeated and the rules are refined continually until the results are satisfactory. It is not unusual for the formalisation process of a complex expert system to last for several years.

## 4.Implementation Phase

- During the implementation stage the formalised concepts are programmed into the computer which has been chosen for system development, using the predetermined techniques and tools to implement a 'first-pass' (prototype) of the expert system.
- If the prototype works at all, the knowledge engineer may be able to determine if the techniques chosen to implement the expert system were the appropriate ones.
- On the other hand, the knowledge engineer may discover that the chosen techniques simply cannot be implemented. It may not be possible, for example, to integrate the knowledge representation techniques selected for different sub-problems.
- At that point, the concepts may have to be re-formalised, or it even may be necessary to create new development tools to implement the system efficiently.

# 4.Implementation Phase



**Fig. 12.11.** *The implementation stage of expert system development.*

Once the prototype system has been refined sufficiently to allow it to be executed, the expert system is ready to be tested thoroughly to ensure that it expertise's correctly.

# 5. Testing (Validation, Verification and Maintenance) Phase

- Testing provides an opportunity to identify the weaknesses in the structure and implementation of the system and to make the appropriate corrections.
- Depending on the types of problems encountered, the testing procedure may indicate that the system was implemented incorrectly, or perhaps that the rules were implemented correctly but were poorly or incompletely formulated.
- Results from the tests are used as 'feedback' to return to a previous stage and adjust the performance of the system.

## 5. Testing (Validation, Verification and Maintenance) Phase

- Once the system has proven to be capable of correctly solving straight-forward problems, the domain expert suggests complex problems which typically would require a great deal of human expertise.
- These more demanding tests should uncover more serious flaws and provide ample opportunity to 'fine tune' the system even further.
- Ultimately, an expert system is judged to be entirely successful only when it operates at the level of a human expert.
- The testing process is not complete until it indicates that the solutions suggested by the expert system are consistently as valid as those provided by a human domain expert.

# Applications of Expert System

- **In designing and manufacturing domain :**It can be broadly used for designing and manufacturing physical devices such as camera lenses and automobiles.
- **In the knowledge domain:** These systems are primarily used for publishing the relevant knowledge to the users. The two popular ES used for this domain is an advisor and a tax advisor.
- **In the finance domain :**In the finance industries, it is used to detect any type of possible fraud, suspicious activity, and advise bankers that if they should provide loans for business or not.
- **In the diagnosis and troubleshooting of devices:** In medical diagnosis, the ES system is used, and it was the first area where these systems were used.
- **Planning and Scheduling:** The expert systems can also be used for planning and scheduling some particular tasks for achieving the goal of that task.

# Uncertainty

- In AI knowledge representation uses techniques such as 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

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

**Examples:**

- In a drawer of ten socks where 8 of them are yellow, there is a 20% chance of choosing a sock that is not yellow.
- There are 9 red candies in a bag and 1 blue candy in the same bag. The chance of picking a blue candy is 10%.

$P(\neg A)$  = 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.

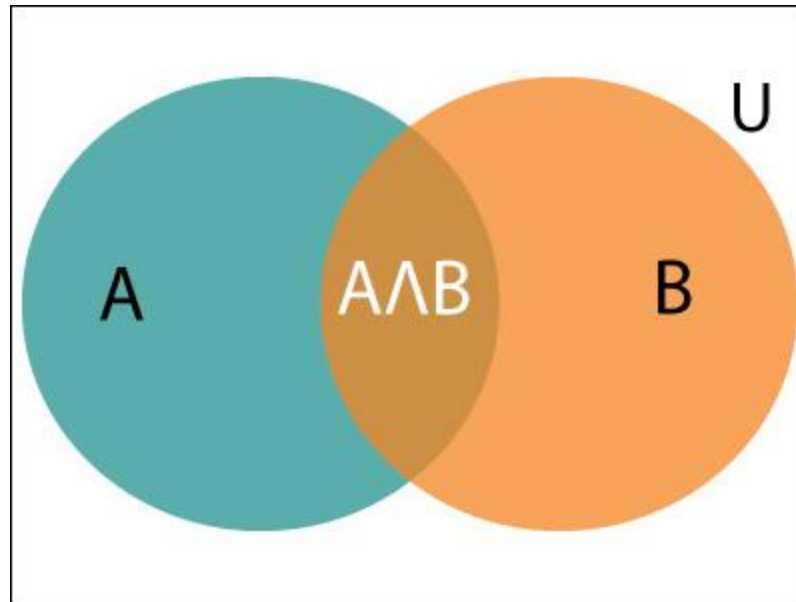
# Conditional Probability

If the probability of A is given and we need to find the probability of B, then it will be given as:

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

# Conditional Probability

It can be explained by using 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 C Language and 40% of the students who likes C and Java, and then what is the percent of students those who like C Language also like Java?

**Solution:**

Let, A is an event that a student likes Java

B is an event that a student likes C Language.

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

Hence, 57% are the students who like C also like Java.

# Prior Probability

**Prior Probability-** Degree of belief in an event, in the absence of any other information

**Example:**

- $P(\text{rain tomorrow}) = 0.7$
- $P(\text{no rain tomorrow}) = 0.3$



# Conditional Probability

What is the probability of an event , given knowledge of another event.

Example:

- $P(\text{raining} \mid \text{sunny})$
- $P(\text{raining} \mid \text{cloudy})$
- $P(\text{raining} \mid \text{cloudy, cold})$

# Conditional Probability...

In some cases , given knowledge of one or more random variables, we can improve our prior belief of another random variable.

For Example:

- $P(\text{slept in stadium}) = 0.5$
- $P(\text{slept in stadium} \mid \text{liked match}) = 0.33$
- $P(\text{didn't slept in stadium} \mid \text{liked match}) = 0.67$

# 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 ...

- Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.
- **Example:** If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.
- Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:

As from product rule we can write:

$$P(A \wedge B) = P(A|B)P(B) \text{ and}$$

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

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

# Bayes Theorem ...

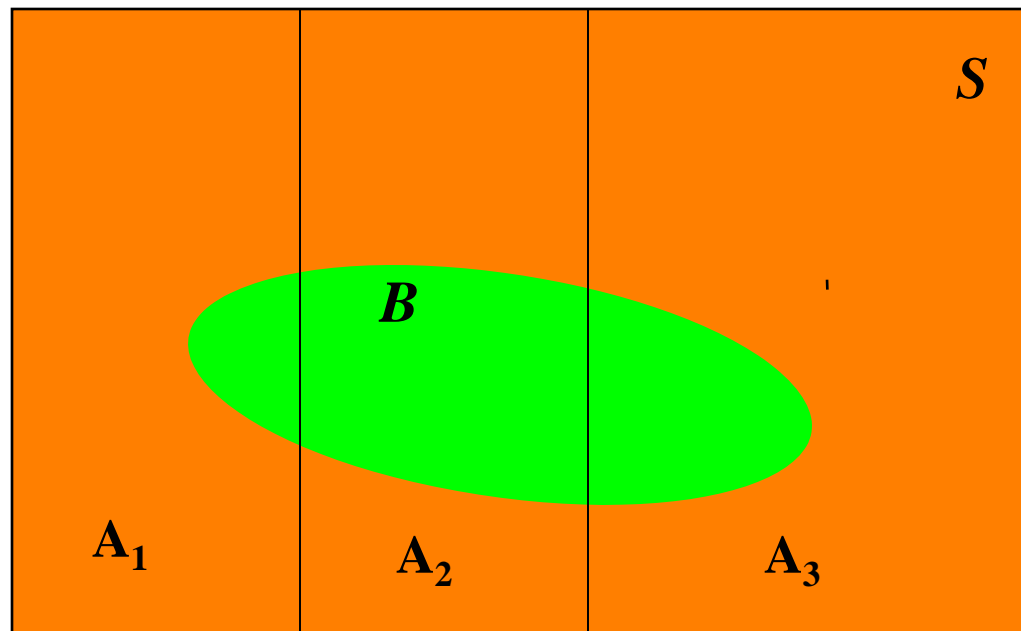
Equating right hand side of both the equations, we will get:

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

- 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.

# Bayes Theorem ...

## Multiple Prior Probabilities



$$P(B) = P(A_1).P(B | A_1) + P(A_2).P(B | A_2) + P(A_3).P(B | A_3)$$



# Bayes Theorem ...

➤  $P(A)$  is called the **prior probability**, probability of hypothesis before considering the evidence

➤  $P(B)$  is called **marginal probability**, pure probability of an evidence.

In the equation (a), in general, we can write

$P(B) = P(A_i) * P(B|A_i)$ , 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:

- Suppose a patient exhibits symptoms that make her physician concerned that she may have a particular disease. The disease is relatively rare in this population, with a prevalence of 0.2% (meaning it affects 2 out of every 1,000 persons). The physician recommends a screening test that costs Rs.10000 and requires a blood sample. Before agreeing to the screening test, the patient wants to know what will be learned from the test, specifically she wants to know the probability of disease, given a positive test result, i.e.,  $P(\text{Disease} \mid \text{Screen Positive})$ .
- The physician reports that the screening test is widely used and has a reported sensitivity of 85%. In addition, the test comes back positive 8% of the time and negative 92% of the time.

# Example 1:

- The information that is available is as follows:
- $P(\text{Disease})=0.002$ , i.e., prevalence = 0.002
- $P(\text{Screen Positive} \mid \text{Disease})=0.85$ , i.e., the probability of screening positive, given the presence of disease is 85% (the sensitivity of the test), and
- $P(\text{Screen Positive})=0.08$ , i.e., the probability of screening positive overall is 8% or 0.08. We can now substitute the values into the above equation to compute the desired probability,
- We know that  $P(\text{Disease})=0.002$ ,  $P(\text{Screen Positive} \mid \text{Disease})=0.85$  and  $P(\text{Screen Positive})=0.08$ . We can now substitute the values into the above equation to compute the desired probability,
- $P(\text{Disease} \mid \text{Screen Positive}) = (0.85)(0.002)/(0.08) = 0.021$

# Example 1:

- The information that is available is as follows:
- $P(\text{Disease})=0.002$ , i.e., prevalence = 0.002
- $P(\text{Screen Positive} \mid \text{Disease})=0.85$ , i.e., the probability of screening positive, given the presence of disease is 85% (the sensitivity of the test), and
- $P(\text{Screen Positive})=0.08$ , i.e., the probability of screening positive overall is 8% or 0.08. We can now substitute the values into the above equation to compute the desired probability,
- We know that  $P(\text{Disease})=0.002$ ,  $P(\text{Screen Positive} \mid \text{Disease})=0.85$  and  $P(\text{Screen Positive})=0.08$ . We can now substitute the values into the above equation to compute the desired probability,
- $P(\text{Disease} \mid \text{Screen Positive}) = (0.85)(0.002)/(0.08) = 0.021$
- The patient undergoes the test and it comes back positive, there is a 2.1% chance that he has the disease.
- Also, note, however, that without the test, there is a 0.2% chance that she has the disease (the prevalence in the population).

## Example 2:

- In a recent newspaper article, it was reported that light trucks, which include SUV's, pick-up trucks and minivans, accounted for 40% of all personal vehicles on the road in 2018. Assume the rest are cars. Of every 100,000 car accidents, 20 involve a fatality; of every 100,000 light truck accidents, 25 involve a fatality. If a fatal accident is chosen at random, what is the probability the accident involved a light truck?

## Example 2:

Events

$C$ - Cars

$T$  –Light truck

$F$  –Fatal Accident

$N$ - Not a Fatal Accident

Given,  $P(F|C) = 20/10000$  and  $P(F|T) = 25/100000$

$P(T) = 0.4$

In addition we know  $C$  and  $T$  are complementary events

$P(C) = 1 - P(T) = 0.6$

Our goal is to compute the conditional probability of a Light truck accident given that it is fatal  $P(T/F)$ .

## Example 2:

Consider  $P(T/F)$

Conditional probability of a Light truck accident given that it is fatal

How do we calculate?

Using conditional probability formula

$$\begin{aligned} P(T / F) &= \frac{P(T \cap F)}{P(F)} = \frac{P(F / T) \cdot P(T)}{P(F / T) \cdot P(T) + P(F / C) \cdot P(C)} \\ &= \frac{(0.00025)(0.4)}{(0.00025)(0.4) + (0.0002)(0.6)} \\ &= 0.4545 \end{aligned}$$



# 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**.

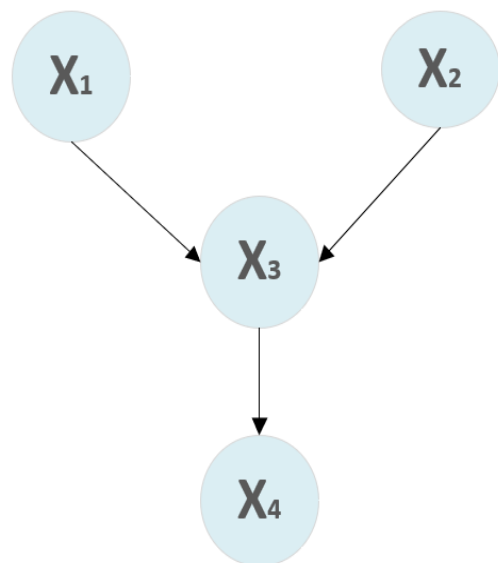
# Bayesian Networks

- 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.**

A **Bayesian network graph (Directed Acyclic 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  $X_1$ ,  $X_2$ ,  $X_3$  and  $X_4$  are random variables represented by the nodes of the network graph.

➤ If we are considering node  $X_3$ , which is connected with node  $X_1$  by a directed arrow, then node  $X_1$  is called the parent of Node  $X_3$ .

➤ Node  $X_4$  is independent of node  $X_1$ .

# Conditional Probability Tables- CPTs

- The conditional probability tables in the network give the probabilities for the value of the random variables depending on the combination of values for the parent nodes.
- Each row must be sum to 1.
- All variables are Boolean, and therefore, the probability of a true value is  $p$ , the probability of false value must be  $1-p$ .
- A table for a boolean variable with  $k$ -parents contains  $2^k$  independently specifiable probabilities.
- A variable with no parents has only one row, representing the prior probabilities of each possible values of the variable.

# Joint Probability Distribution

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



➤ If we have variables A,B,C,D then the probabilities of a different combination of their variables are known as Joint probability distribution.

➤ A directed graph G with four vertices A,B,C and D. If  $P(x_A, x_B, x_C, x_D)$  factorizes with respect to G, then we must have

$$P(x_A, x_B, x_C, x_D) = P(x_A)P(x_B|x_A)P(x_C|x_B)P(x_D|x_C)$$

# Bayesian Network- Burglar Alarm

- You have installed a new burglar-alarm at home.
- It is fairly reliable at detecting a burglary, but also responds on occasion to minor earthquakes.
- You have also have two neighbors, David and Sophia, who have promised to call you at work when they hear the alarm.
- David always calls when he hears the alarm, but sometimes confuses the telephone ringing with the alarm and calls then, too.
- Sophia, on the other hand, likes loud music and misses the alarm altogether.
- Given the evidence of who has or has not called, we would like to estimate the probability of a burglary.

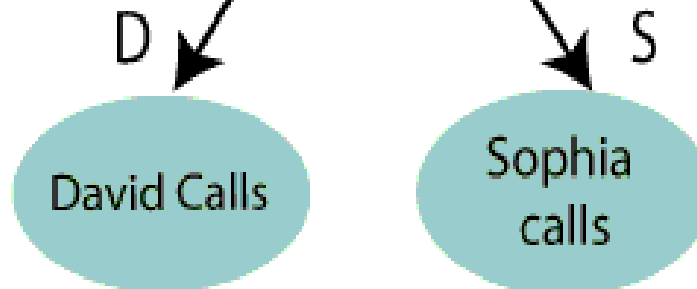
T	0.002
F	0.998



T	0.001
F	0.999

B	E	P(A=T)	P(A=F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P(D=T)	P(D=F)
T	0.91	0.09
F	0.05	0.95



A	P(S=T)	P(S=F)
T	0.75	0.25
F	0.02	0.98

# 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.



# Problem:

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:

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

- From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

$$\begin{aligned} P(S, D, A, \neg B, \neg E) &= P(S|A) * P(D|A) * P(A|\neg B \wedge \neg E) * P(\neg B) * P(\neg E). \\ &= 0.75 * 0.91 * 0.001 * 0.998 * 0.999 \\ &= \mathbf{0.00068045}. \end{aligned}$$

Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

# Inference in Bayesian Networks

## **Purpose:**

- Probabilistic Inference System is to compute Posterior Probability Distribution for a set of query variables given some observed events.
- That is, some assignment of values to a set of evidence variables.

# Inference in Bayesian Networks

## Notations:

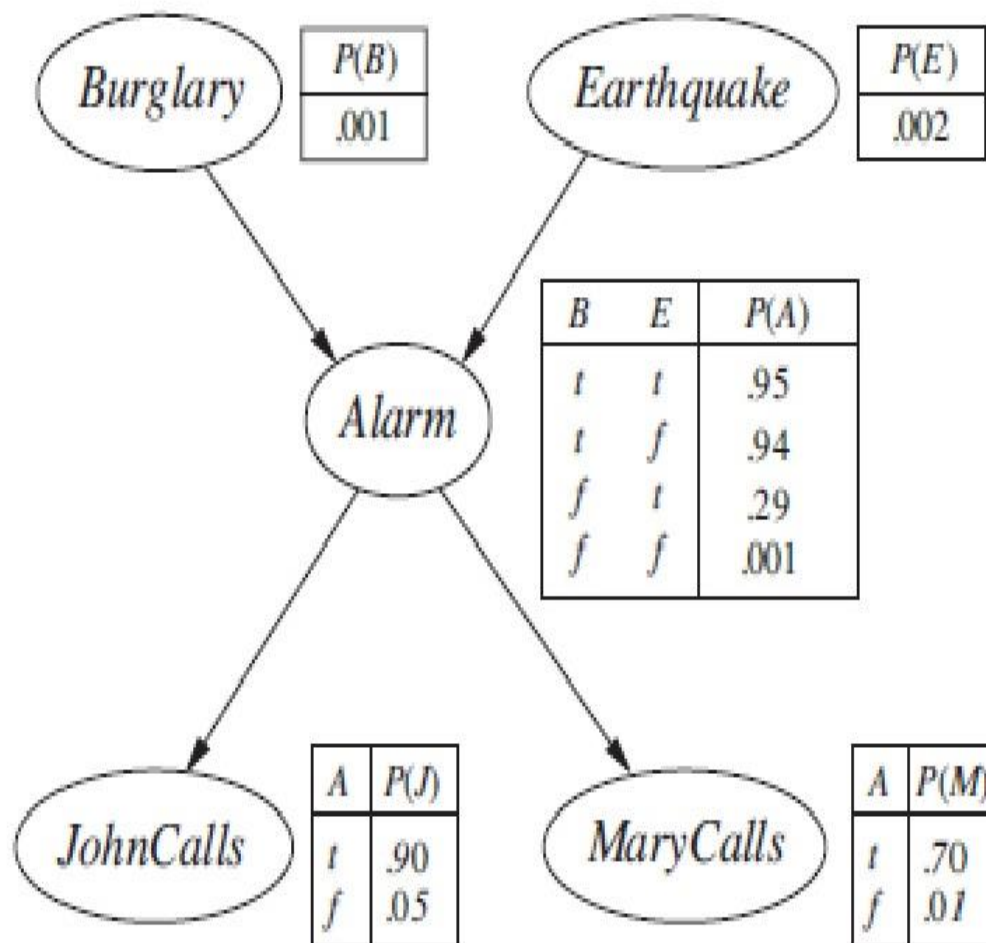
- **X** - Denotes the query variable.
- **E** - Set of Evidence variables
- **e** - Particular observed event
- **Y** - Non evidence, non query variable  $Y_1, \dots, Y_n$  (Hidden variables)
- The complete set of variables  $X = \{X\} \cup E \cup Y$
- A typical query ask for the Posterier Probability Distribution  $P\{X|e\}$

- In the burglary network, we might observe the event in which

JohnCalls = true and MaryCalls = true

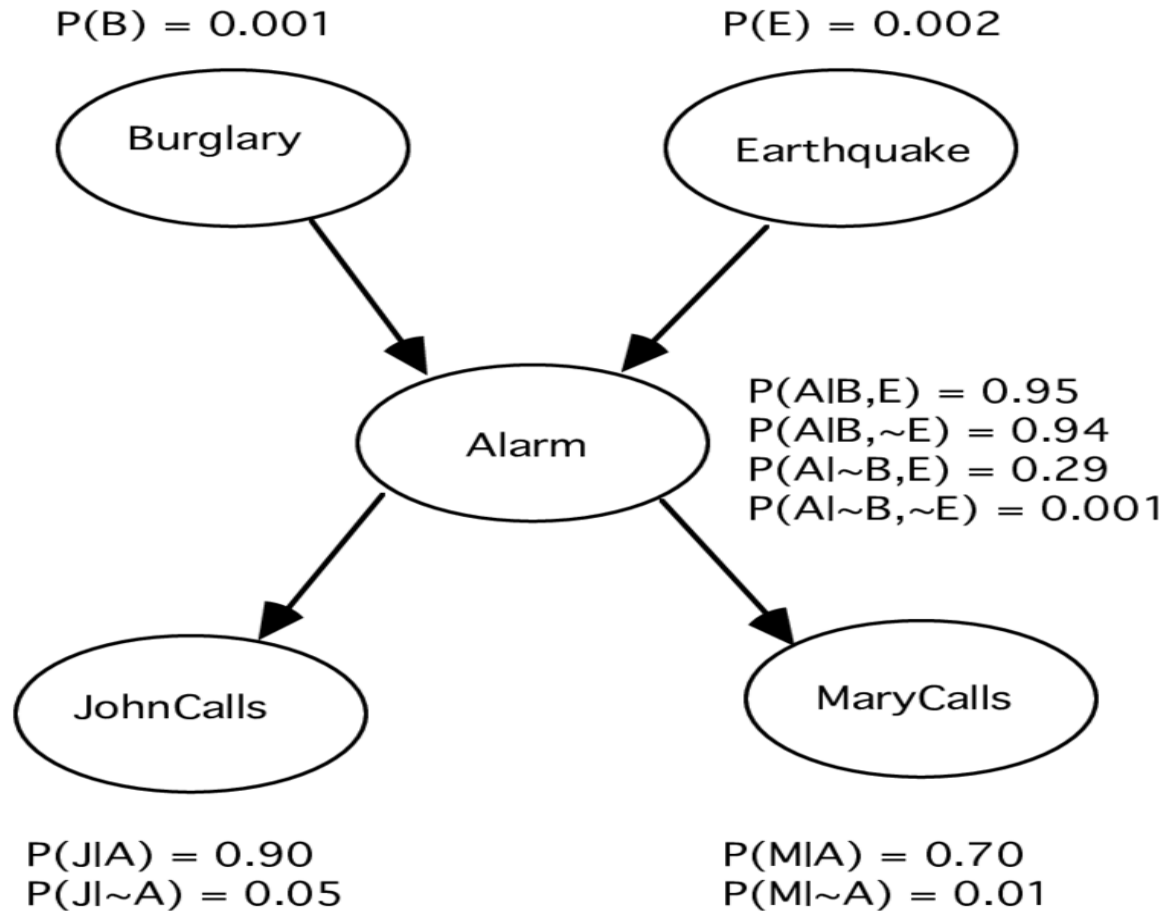
- We could then ask for, say, the probability that a burglary has occurred:

$$P(\text{Burglary} \mid \text{JohnCalls} = \text{true}, \text{MaryCalls} = \text{true}) = ?$$



**Figure 14.2** A typical Bayesian network, showing both the topology and the conditional probability tables (CPTs). In the CPTs, the letters *B*, *E*, *A*, *J*, and *M* stand for *Burglary*, *Earthquake*, *Alarm*, *JohnCalls*, and *MaryCalls*, respectively.

$$P(B \mid J=\text{true}, M=\text{true})$$



# Types of Inferences:

- Inference by Enumeration  
(inference by listing or recording all variables)
- Inference by Variable Elimination  
(inference by variable removal)

# Inference by Enumeration

- Any conditional probability can be computed by summing terms from the full joint distribution.
- More specifically, a query  $P(X | e)$  can be answered using equation:

$$P(X | e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y)$$

where  $\alpha$  is normalized constant

$X$  – Query variable

$e$  - event

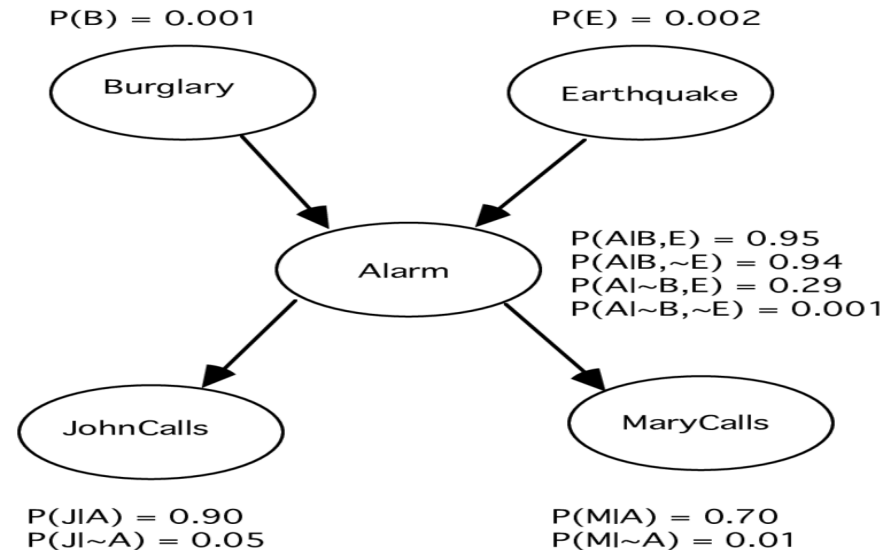
$y$  – number of terms (hidden variables)



# Inference by Enumeration...

- Consider  $P(\text{Burglary} \mid \text{JohnCalls} = \text{true}, \text{MaryCalls} = \text{true})$
- Burglary - query variable (X)
- JohnCalls - Evidence variable 1 (E1)
- MaryCalls - Evidence variable 2 (E2)
- The hidden variable of this query are earthquake and alarm.

# Inference by Enumeration...



$$P(B \mid j, m) = P(B, j, m) / P(j, m)$$

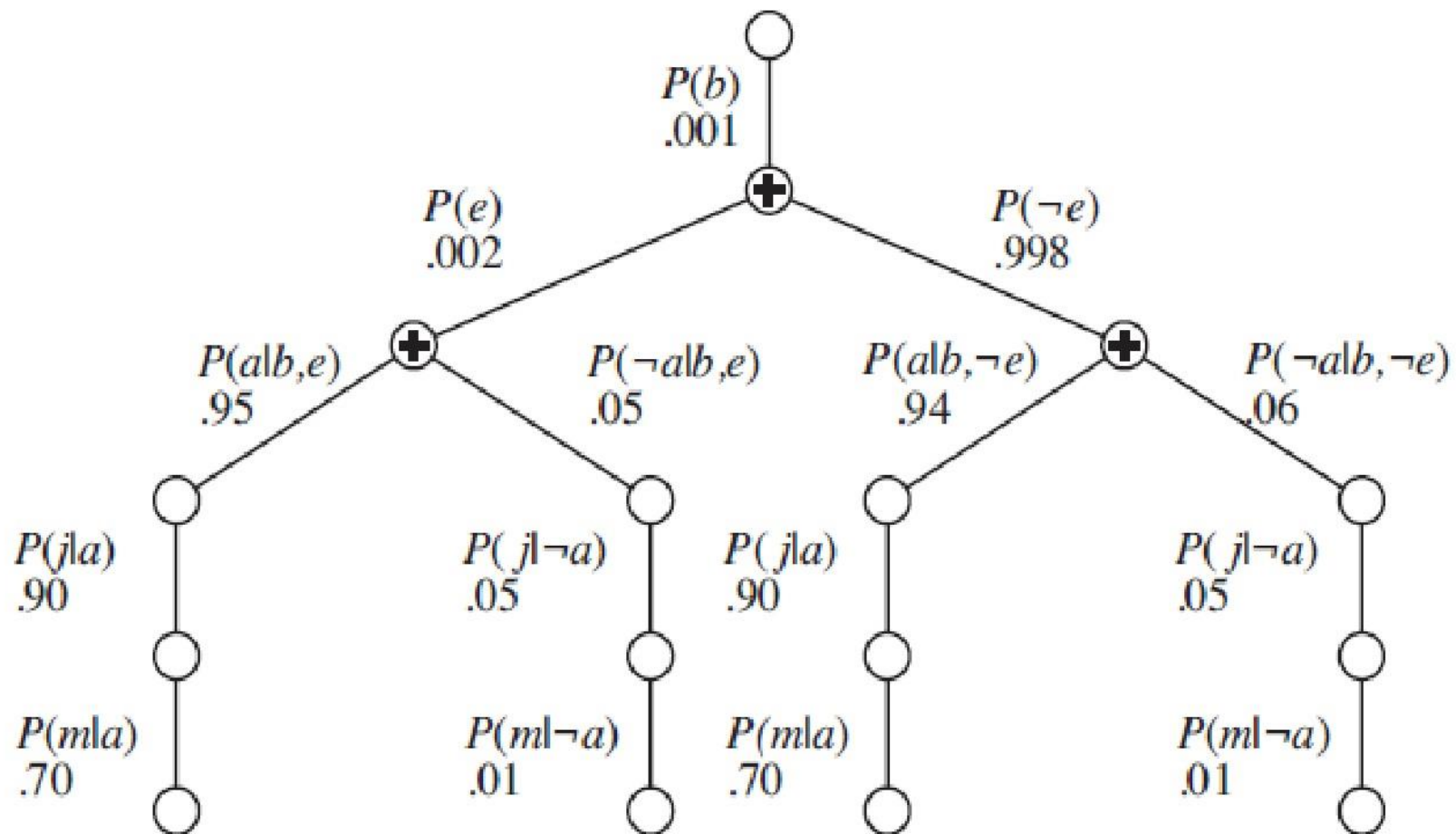
$$P(B \mid j, m) = \alpha P(B, j, m)$$

$$P(B \mid j, m) = \alpha \sum_{E, A} P(B, E, A, j, m)$$

$$P(B \mid j, m) = \alpha \sum_{E, A} P(B)P(E)P(A|E, B)P(j|A)P(m|A)$$

$$P(B \mid j, m) = \alpha P(B) \sum_E P(E) \sum_A P(A|E, B)P(j|A)P(m|A)$$

$$P(\mathbf{B} \mid \mathbf{j}, \mathbf{m}) = \alpha P(\mathbf{B}) \Sigma_{\mathbf{E}} P(\mathbf{E}) \Sigma_{\mathbf{A}} P(\mathbf{A} \mid \mathbf{B}, \mathbf{E}) P(\mathbf{j} \mid \mathbf{A}) P(\mathbf{m} \mid \mathbf{A})$$

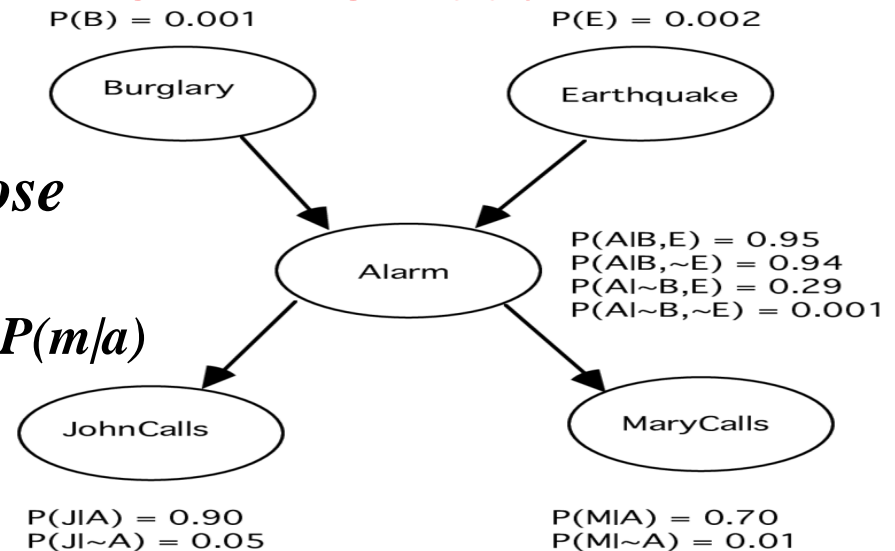


# Inference by Enumeration...

*Let us consider for simplicity purpose*

*burglary = true*

$$P(b \mid j, m) = \alpha P(b) \sum_e P(e) \sum_a P(a/e, b) P(j/a) P(m/a)$$



This expression can be evaluated by looping through the variables in order, multiplying CPT entries as we go. For each summation, we also need to loop over the variable's possible values. Using the numbers from above figure, we obtain  $P(b \mid j, m) = \alpha \times 0.00059224$ . The corresponding computation for  $\neg b$  yields  $\alpha \times 0.0014919$ ; hence,

$$P(B \mid j, m) = \alpha (0.00059224, 0.0014919) \approx 0.284, 0.716 .$$

That is, the chance of a burglary, given calls from both neighbors, is about 28%.

# Inference by Variable Elimination

- The enumeration algorithm can be improved substantially by eliminating repeated calculations.
- The idea is simple: do the calculation once and save the results for later use. This is a form of dynamic programming.
- Variable elimination works by evaluating expressions, from the previous equation (derived in inference by enumeration)

$$P(b \mid j, m) = \alpha P(b) \Sigma_e P(e) \Sigma_a P(a \mid e, b) P(j \mid a) P(m \mid a)$$

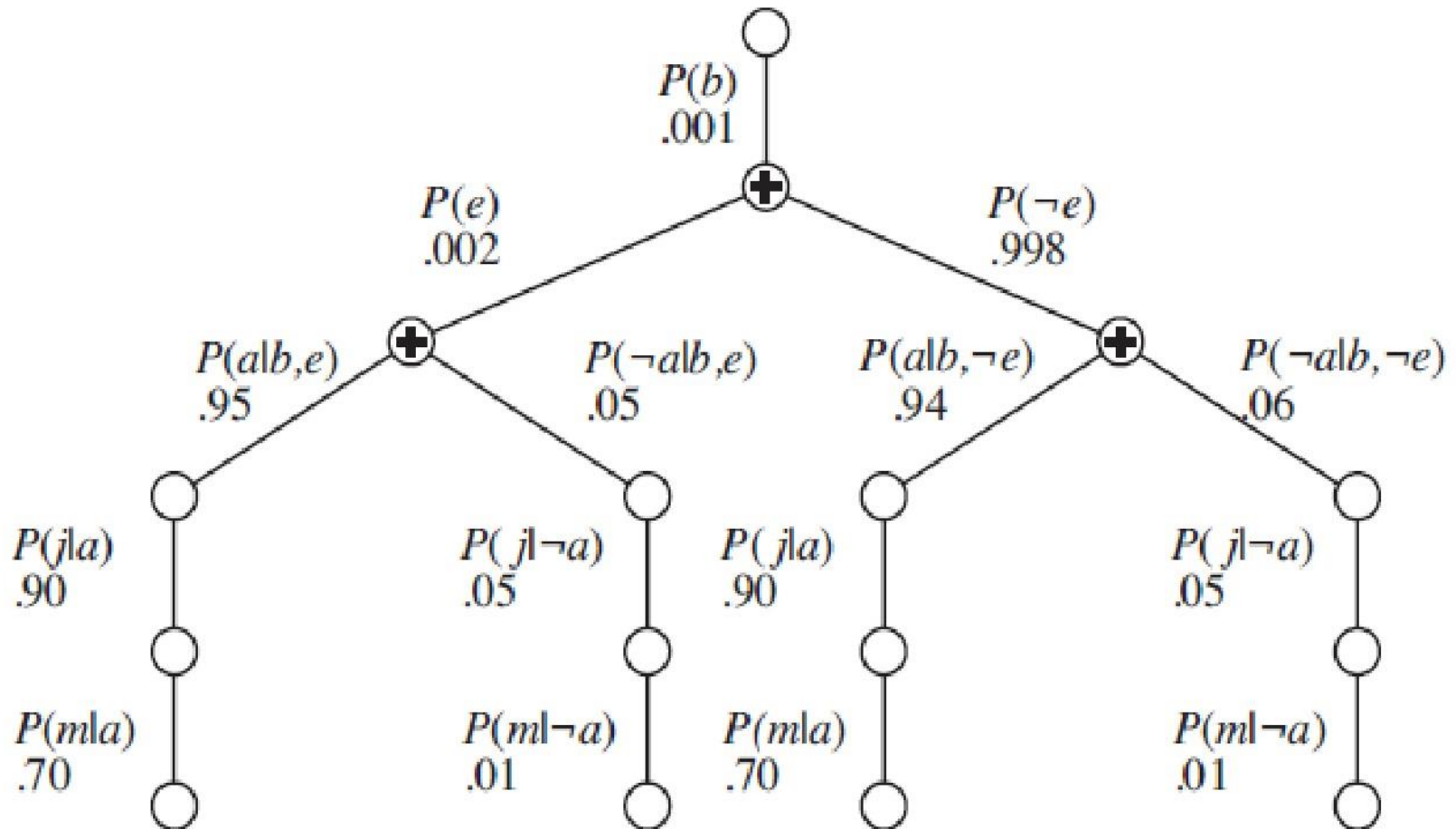
# Inference by Variable Elimination...

- Intermediate results are stored, and summations over each variable are done only for those portions of the expression that depend on the variable.
- Let us illustrate this process for the burglary network.
- We evaluate the below expression in such a way that we annotated each part of the expression with the name of the associated variable; these parts are called factors

$$P(B / j, m) = \alpha P(B) \Sigma_E P(E) \Sigma_A P(A/E, B) P(j/A) P(m/A)$$

## Variable Elimination

$$P(\mathbf{B} \mid \mathbf{j}, \mathbf{m}) = \alpha P(\mathbf{B}) \Sigma_{\mathbf{E}} P(\mathbf{E}) \Sigma_{\mathbf{A}} P(\mathbf{A} \mid \mathbf{B}, \mathbf{E}) P(\mathbf{j} \mid \mathbf{A}) P(\mathbf{m} \mid \mathbf{A})$$



# Inference by Variable Elimination...

$$\propto P(B) \Sigma_E P(E) \Sigma_A P(A/E, B) P(j/A) P(m/A)$$

P(J   A)	
A	0.90 (0.1)
-A	0.05 (0.95)

P(M   A)	
A	0.70 (0.30)
-A	0.01 (0.99)

P(j A)P(m A)	
A	0.9 * 0.7
-A	0.05 * 0.01



# Inference by Variable Elimination...

$$\propto P(B) \Sigma_E P(E) \Sigma_A P(A/E, B) f_1(A)$$

P(J   A)	
A	0.90 (0.1)
-A	0.05 (0.95)

P(M   A)	
A	0.70 (0.30)
-A	0.01 (0.99)

f1 (A)	
A	0.63
-A	0.0005

# Inference by Variable Elimination...

$$\propto P(B) \Sigma_E P(E) \Sigma_A P(A/E, B) f1(A)$$

f1 (A)	
A	0.63
-A	0.0005

P(A   E,B)	
e , b	0.95 (0.05)
e , -b	0.29 (0.71)
-e , b	0.94(0.06)
-e , -b	0.001(0.999)

$\Sigma_A P(A   E,B) f1(A)$	
e , b	$0.95 * 0.63 + 0.05 * 0.0005$
e , -b	$0.29 * 0.63 + 0.71 * 0.0005$
-e , b	$0.94 * 0.63 + 0.06 * 0.0005$
-e , -b	$0.001 * 0.63 + 0.999 * 0.0005$

# Inference by Variable Elimination...

$$\propto P(B) \Sigma_E P(E) f_2(E, B)$$

f1 (A)	
A	0.63
-A	0.0005

P(A E,B)	
e , b	0.95 (0.05)
e , -b	0.29 (0.71)
-e , b	0.94(0.06)
-e , -b	0.001(0.999)

f2(E,B)	
e , b	0.60
e , -b	0.18
-e , b	0.59
-e , -b	0.001

# Inference by Variable Elimination...

$$\propto P(B) \sum_E P(E) f_2(E, B)$$

$P(E=T)$     $P(E=F)$   
 0.002   0.998

$P(B=T)$     $P(B=F)$   
 0.001   0.999

$f_2(E, B)$	
e , b	0.60
e , -b	0.18
-e , b	0.59
-e , -b	0.001

$P(B) \sum_E P(E) f_2(E, B)$	
b	$0.60 * 0.002 * 0.001 + 0.59 * 0.998 * 0.001$
-b	$0.18 * 0.002 * 0.999 + 0.001 * 0.998 * 0.999$

# Inference by Variable Elimination...

$\propto f_3(B)$

$P(E=T) \quad P(E=F)$   
0.002    0.998

$P(B=T) \quad P(B=F)$   
0.001    0.999

<b>f2(E,B)</b>	
e , b	0.60
e , -b	0.18
-e , b	0.59
-e , -b	0.001

<b>f3(B)</b>	
b	0.0006
-b	0.0013

# Inference by Variable Elimination...

$$\propto f_3(B) \rightarrow P(B \mid j, m)$$

<b>f<sub>3</sub>(B)</b>	
b	0.0006
-b	0.0013

$$N = 0.0006 + 0.0013 = 0.0019$$

<b>P(B   j, m)</b>	
b	0.32
-b	0.68

That is, the chance of a burglary, given calls from both neighbors, is about 32%.

# WHAT IS FUZZY LOGIC?

- Definition of fuzzy
  - Fuzzy – “not clear, distinct, or precise; blurred”
- Definition of fuzzy logic
  - A form of knowledge representation suitable for notions that cannot be defined precisely, but which depend upon their contexts.
  - So we cannot decide in real life that the given problem or statement is either true or false.
  - At that time, this concept provides many values between the true and false and gives the flexibility to find the best solution to that problem.

# WHAT IS FUZZY LOGIC?

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 –

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

CERTAINLY YES
POSSIBLY YES
CANNOT SAY
POSSIBLY NO
CERTAINLY NO



# Implementation

- It can be implemented in systems with various sizes and capabilities ranging from small micro-controllers to large, networked, workstation-based control systems.
- It can be implemented in hardware, software, or a combination of both.

## 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.

# TRADITIONAL REPRESENTATION OF LOGIC



Slow

Speed = 0



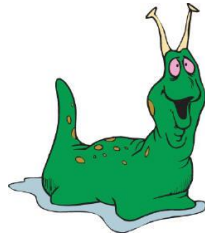
Fast

Speed = 1

```
bool speed;  
get the speed  
if ( speed == 0) {  
    // speed is slow  
}  
else {  
    // speed is fast  
}
```

# FUZZY LOGIC REPRESENTATION

- For every problem must represent in terms of fuzzy sets.
- What are fuzzy sets?



Slowest

[ 0.0 – 0.25 ]



Slow

[ 0.25 – 0.50 ]



Fast

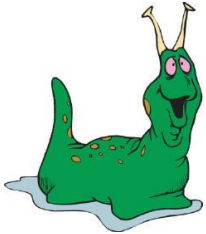
[ 0.50 – 0.75 ]



Fastest

[ 0.75 – 1.00 ]

# FUZZY LOGIC REPRESENTATION CONT.



Slowest

Slow

Fast

Fastest

```
float speed;  
get the speed  
if ((speed >= 0.0)&&(speed < 0.25)) {  
    // speed is slowest  
}  
else if ((speed >= 0.25)&&(speed < 0.5))  
{  
    // speed is slow  
}  
else if ((speed >= 0.5)&&(speed < 0.75))  
{  
    // speed is fast  
}  
else // speed >= 0.75 && speed < 1.0  
{  
    // speed is fastest  
}
```

# Fuzzy Set

- The set theory of classical is the subset of Fuzzy set theory. Fuzzy logic is based on this theory, which is a generalization of the classical theory of set (i.e., crisp set) introduced by Zadeh in 1965.
- A fuzzy set is a collection of values which exist between 0 and 1.
- Fuzzy sets are denoted or represented by the tilde ( $\sim$ ) character. The sets of Fuzzy theory were introduced in 1965 by Lofti A. Zadeh and Dieter Klaua.
- In the fuzzy set, the partial membership also exists. This theory released as an extension of classical set theory.

# Fuzzy Set

This theory is denoted mathematically as A fuzzy set ( $\tilde{A}$ ) is a pair of U and M, where U is the Universe of discourse and M is the membership function which takes on values in the interval [ 0, 1 ]. The universe of discourse (U) is also denoted by  $\Omega$  or X.

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) | x \in X\}$$

# Operations on Fuzzy Set

Given  $\tilde{A}$  and  $B$  are the two fuzzy sets, and  $X$  be the universe of discourse with the following respective member functions:

$$\mu_{\tilde{A}}(x) \text{ and } \mu_{\tilde{B}}(x)$$

The operations of Fuzzy set are as follows:

**1. Union Operation:** The union operation of a fuzzy set is defined by:

$$\mu_{A \cup B}(x) = \max (\mu_A(x), \mu_B(x))$$

## Example:

Let's suppose  $A$  is a set which contains following elements:

$$A = \{(X_1, 0.6), (X_2, 0.2), (X_3, 1), (X_4, 0.4)\}$$

And,  $B$  is a set which contains following elements:

$$B = \{(X_1, 0.1), (X_2, 0.8), (X_3, 0), (X_4, 0.9)\}$$

then,

$$A \cup B = \{(X_1, 0.6), (X_2, 0.8), (X_3, 1), (X_4, 0.9)\}$$

# Operations on Fuzzy Set

**2. Intersection Operation:** The intersection operation of fuzzy set is defined by:

$$\mu_{A \cap B}(x) = \min (\mu_A(x), \mu_B(x))$$

## **Example:**

Let's suppose A is a set which contains following elements:

$$A = \{(X_1, 0.3), (X_2, 0.7), (X_3, 0.5), (X_4, 0.1)\}$$

and, B is a set which contains following elements:

$$B = \{(X_1, 0.8), (X_2, 0.2), (X_3, 0.4), (X_4, 0.9)\}$$

then,

$$A \cap B = \{(X_1, 0.3), (X_2, 0.2), (X_3, 0.4), (X_4, 0.1)\}$$



# Operations on Fuzzy Set

**3. Complement Operation:** The complement operation of fuzzy set is defined by:

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x),$$

**Example:**

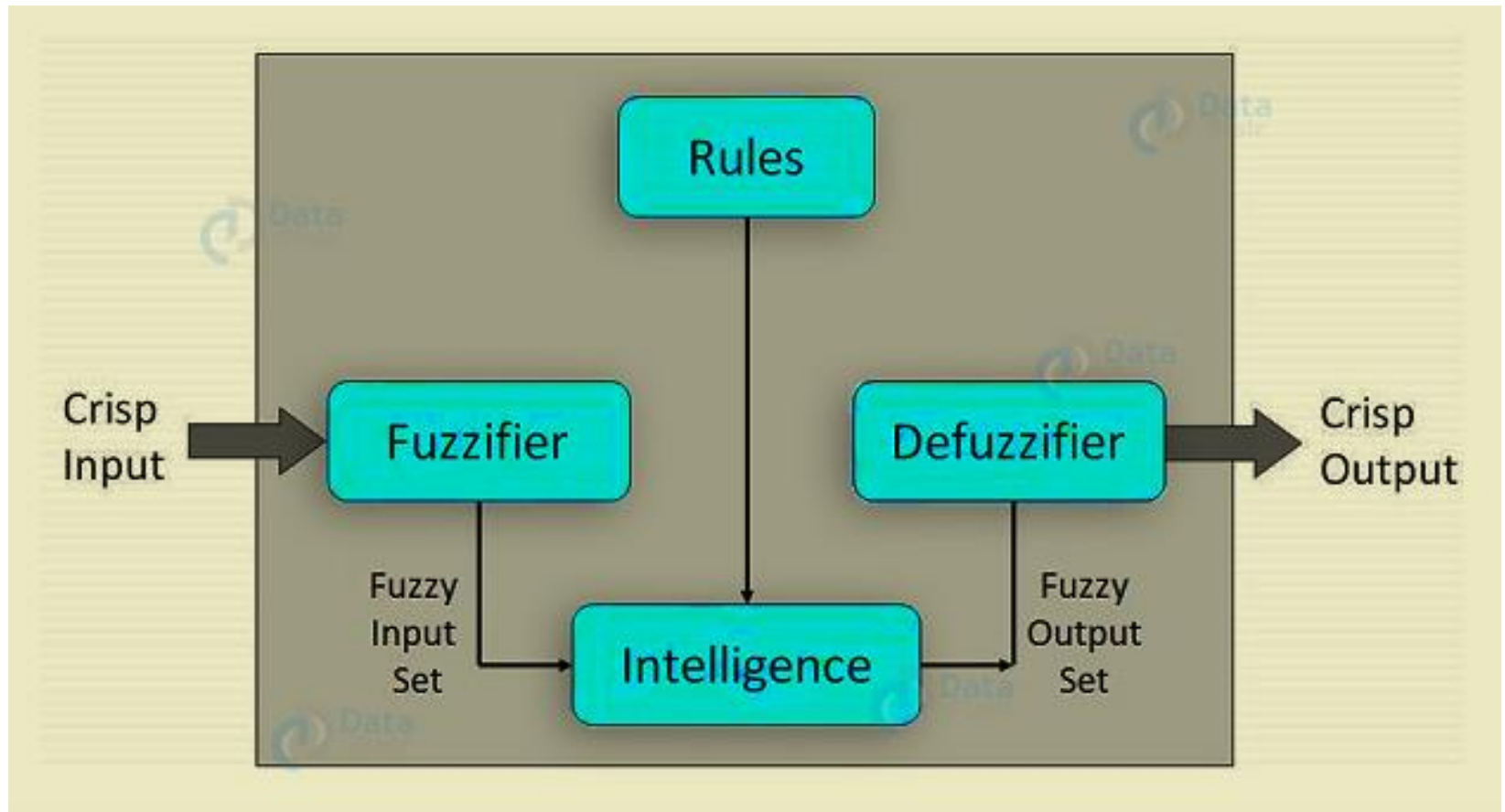
Let's suppose A is a set which contains following elements:

$$A = \{(X_1, 0.3), (X_2, 0.8), (X_3, 0.5), (X_4, 0.1)\}$$

then,

$$\bar{A} = \{(X_1, 0.7), (X_2, 0.2), (X_3, 0.5), (X_4, 0.9)\}$$

# Fuzzy Logic Systems Architecture



# Fuzzy Logic Systems Architecture

Fuzzy Logic architecture has four main parts as shown in the diagram:

## **Fuzzification:**

- Fuzzification step helps to convert inputs. It allows you to convert, crisp numbers into fuzzy sets. Crisp inputs measured by sensors and passed into the control system for further processing. This component divides the input signals into following five states in any Fuzzy Logic system:
  - Large Positive (LP)
  - Medium Positive (MP)
  - Small (S)
  - Medium Negative (MN)
  - Large negative (LN)

# Fuzzy Logic Systems Architecture

## **Rule Base:**

- It contains all the rules and the if-then conditions offered by the experts to control the decision-making system. The recent update in fuzzy theory provides various methods for the design and tuning of fuzzy controllers. This updates significantly reduce the number of the fuzzy set of rules.

## **Inference Engine:**

- It helps you to determines the degree of match between fuzzy input and the rules. Based on the % match, it determines which rules need implment according to the given input field. After this, the applied rules are combined to develop the control actions.

# Fuzzy Logic Systems Architecture

## **Defuzzification:**

- At last the Defuzzification process is performed to convert the fuzzy sets into a crisp value. There are many types of techniques available, so you need to select it which is best suited when it is used with an expert system.

# Fuzzy logic algorithm

1) Initialization process:

- Define the linguistic variables.
- Construct the fuzzy logic membership functions that define the meaning or values of the input and output terms used in the rules.
- Construct the rule base (Break down the control problem into a series of IF X AND Y, THEN Z rules based on the fuzzy logic rules).

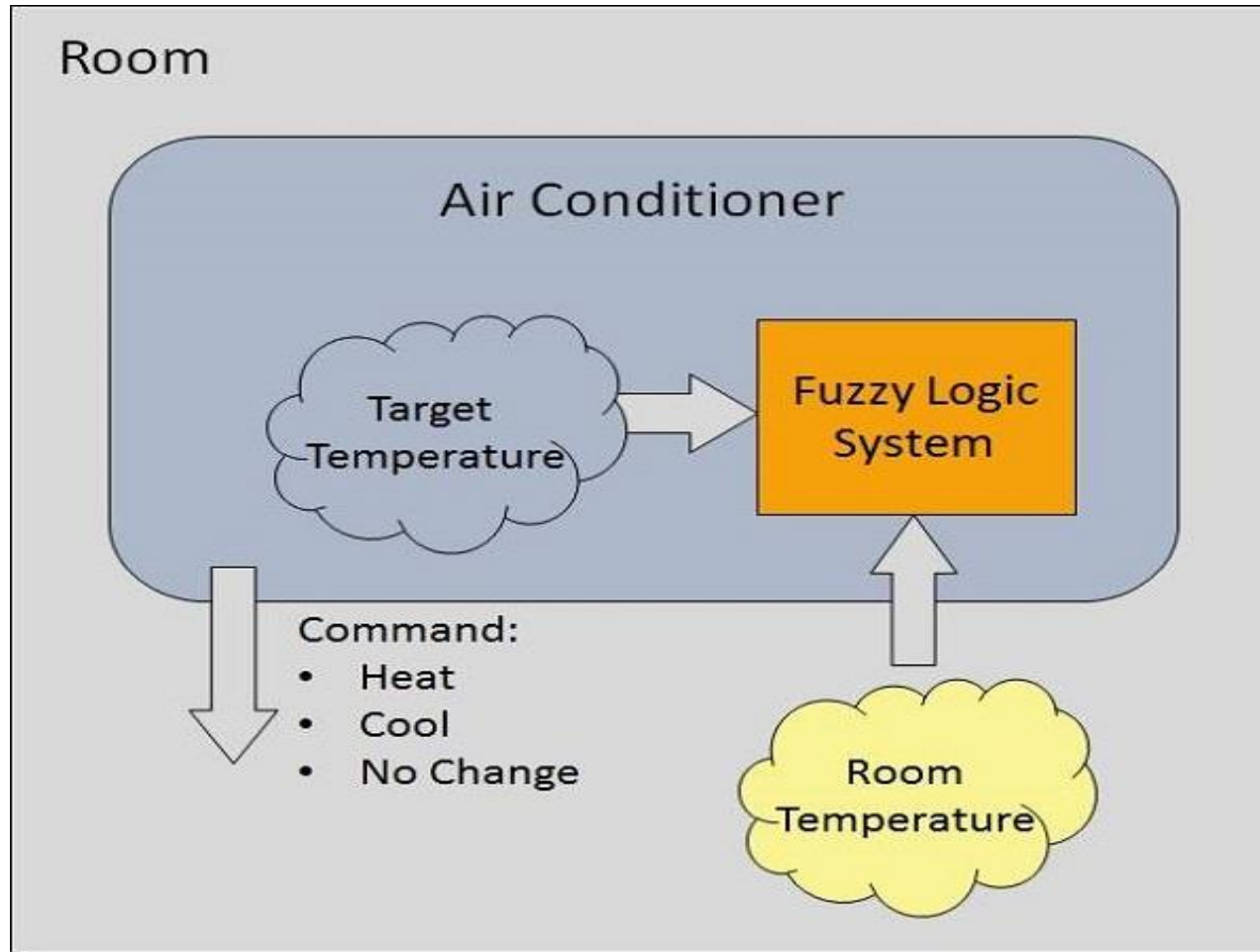
2) Convert crisp input data to fuzzy values using the membership functions (fuzzification).

3) Evaluate the rules in the rule base (inference).

4) Combine the results of each rule (inference).

5) Convert the output data to non-fuzzy values (defuzzification).

# Example: Air conditioner system controlled by a FLS



# Example: Air conditioner system controlled by a FLS

- The system adjusts the temperature of the room according to the current temperature of the room and the target value. The fuzzy engine periodically compares the room temperature and the target temperature, and produces a command to heat or cool the room.

## **Define linguistic variables and terms**

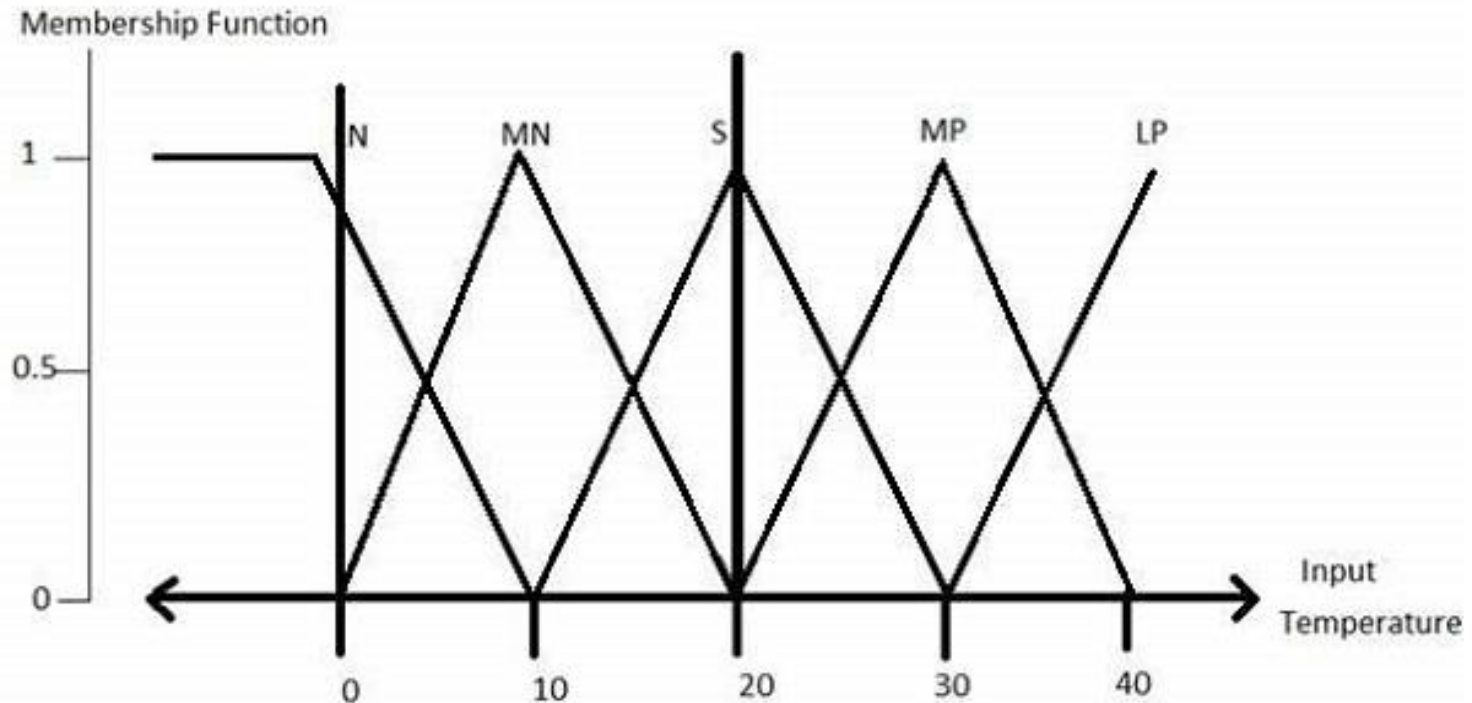
- Linguistic variables are input and output variables in the form of simple words or sentences. For room temperature, cold, warm, hot, etc., are linguistic terms.
- Temperature (t) = {too-cold, cold, warm, hot, too hot}
- Every member of this set is a linguistic term and it can cover some portion of overall temperature values.



# Example: Air conditioner system controlled by a FLS

**Construct membership functions for them**

The membership functions of temperature variable are as shown –



# Example: Air conditioner system controlled by a FLS

## Construct knowledge base rules

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

RoomTemp. /Target	Too Cold	Cold	Warm	Hot	Too Hot
Too Cold	No_Change	Heat	Heat	Heat	Heat
Cold	Cool	No_Change	Heat	Heat	Heat
Warm	Cool	Cool	No_Change	Heat	Heat
Hot	Cool	Cool	Cool	No_Change	Heat
Too Hot	Cool	Cool	Cool	Cool	No_Change

# Example: Air conditioner system controlled by a FLS

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

For Air Conditioner example, the following rules can be used:

- 1) IF (temperature is cold OR too-cold) AND (target is warm) THEN command is heat.
- 2) IF (temperature is hot OR too-hot) AND (target is warm) THEN command is cool.
- 3) IF (temperature is warm) AND (target is warm) THEN command is nochange.

# Example: Air conditioner system controlled by a FLS

**Defuzzification:** The result is a fuzzy value and should be defuzzified to obtain a crisp output. This is the purpose of the defuzzifier component of a FLS. Defuzzification is performed according to the membership function of the output variable. o This defuzzification is not part of the 'mathematical fuzzy logic' and various strategies are possible.

The mostly-used algorithms for defuzzification are listed.

- 1) Finding the center of gravity.
- 2) Finding the center of gravity for singletons.
- 3) Finding the average mean.
- 4) Finding the left most maximum.
- 5) Finding the right most maximum.

# Membership Function

- As this function allows you to quantify linguistic term. Also, represent a fuzzy set graphically. Although, MF for a fuzzy set  $A$  on the universe of discourse. That  $X$  is defined as  $\mu_A: X \rightarrow [0,1]$ .
- In this function, between a value of 0 and 1, each element of  $X$  is mapped. We can define it as the degree of membership. Also, it quantifies the degree of membership of the element. That is in  $X$  to the fuzzy set  $A$ .

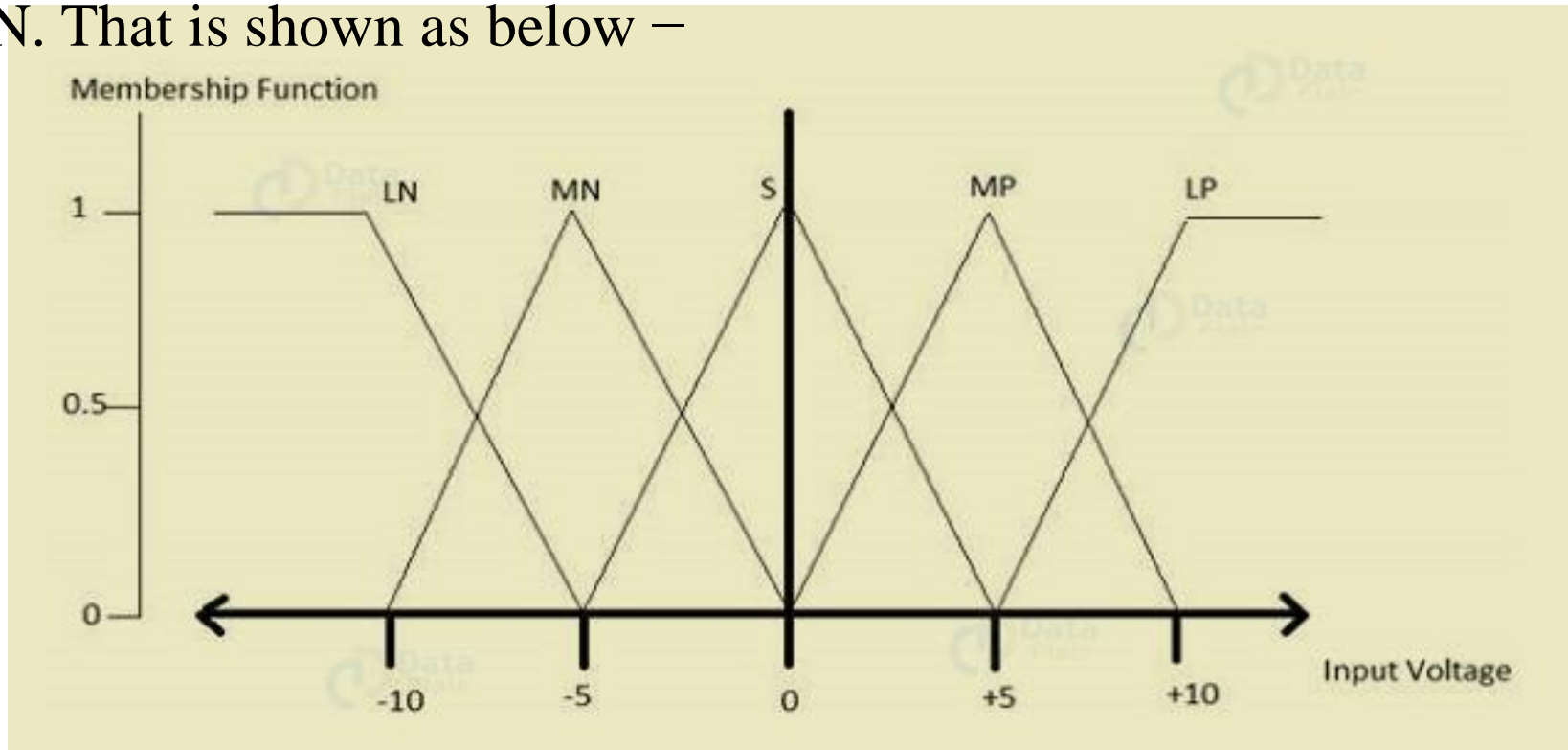
**x-axis**— It represents the universe of discourse.

**y-axis** — It represents the degrees of membership in the  $[0, 1]$  interval.

# Membership Function

We can apply different membership functions to fuzzify a numerical value. Also, we use simple functions as complex. As they do not add more precision in the output.

We can define all membership functions for LP, MP, S, MN, and LN. That is shown as below –

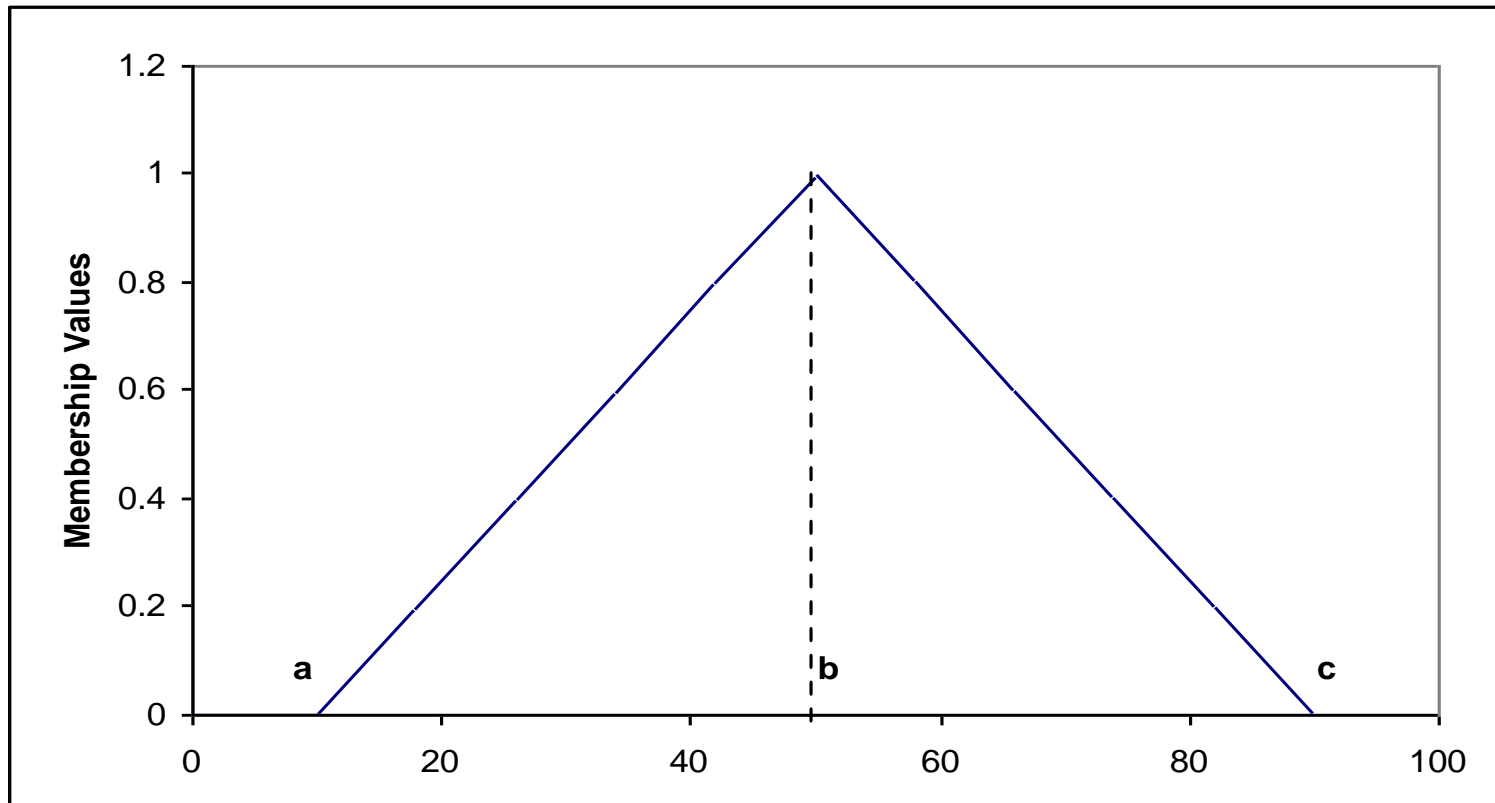


*Fuzzy Logic System – Membership Function*

# Triangular membership functions

$$\mu_F(x, a, b, c) = \begin{cases} 0, & \text{if } x < a \\ (x - a) / (b - a), & \text{if } a \leq x \leq b \\ (c - x) / (c - b), & \text{if } b \leq x \leq c \\ 0, & \text{if } c < x \end{cases}$$

# Cont...



**Figure Triangular Function**



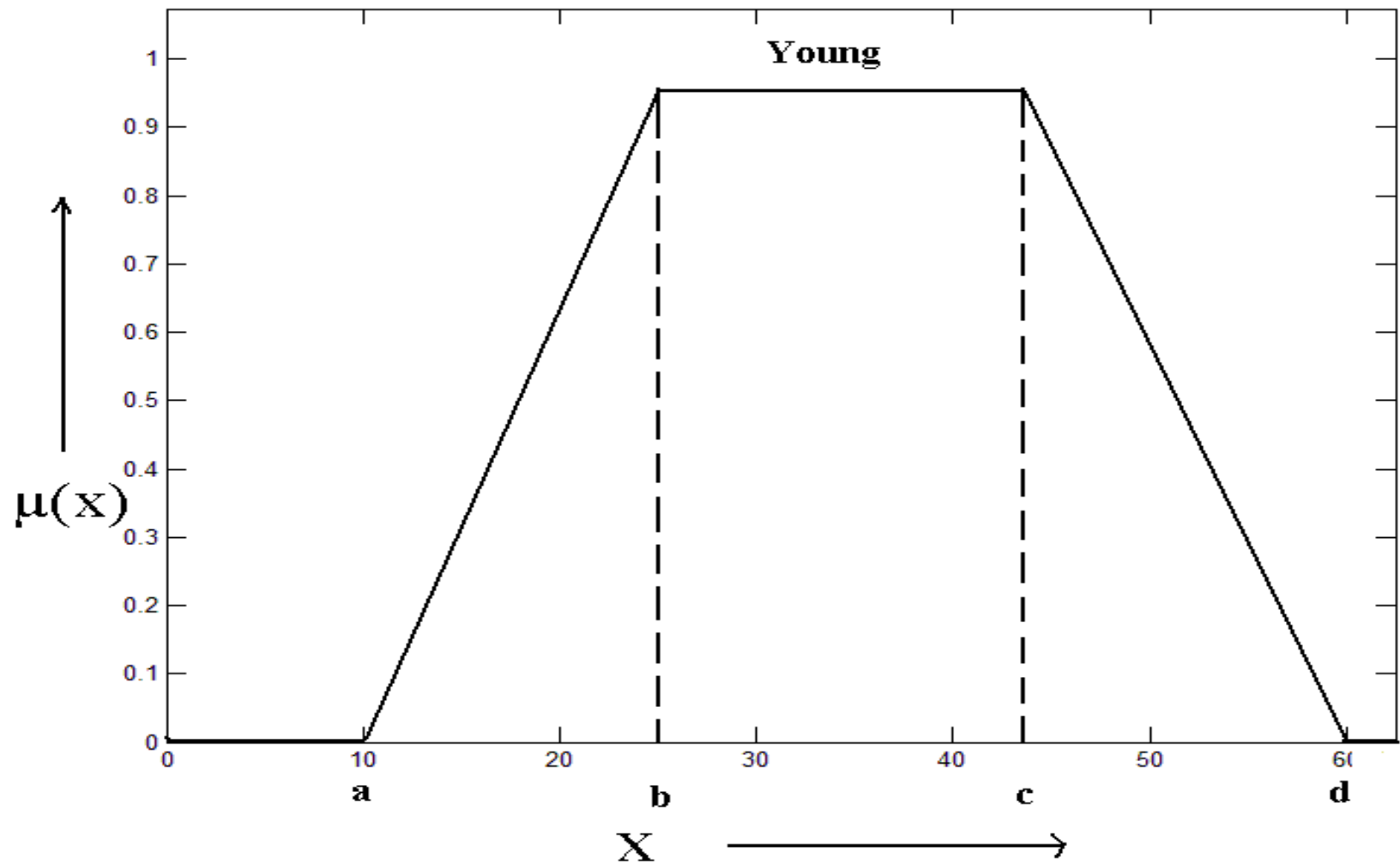
# Trapezoidal membership function

$$\mu_F(x, a, b, c, d) = \begin{cases} 0, & \text{if } x < a \\ (x - a) / (b - a), & \text{if } a \leq x \leq b \\ 1, & \text{if } b < x < c \\ (d - x) / (d - c), & \text{if } c \leq x \leq d \\ 0, & \text{if } d < x \end{cases}$$

# Trapezoidal membership function

$$\mu_F(x, a, b, c, d) = \begin{cases} 0, & \text{if } x < a \\ (x - a) / (b - a), & \text{if } a \leq x \leq b \\ 1, & \text{if } b < x < c \\ (d - x) / (d - c), & \text{if } c \leq x \leq d \\ 0, & \text{if } d < x \end{cases}$$

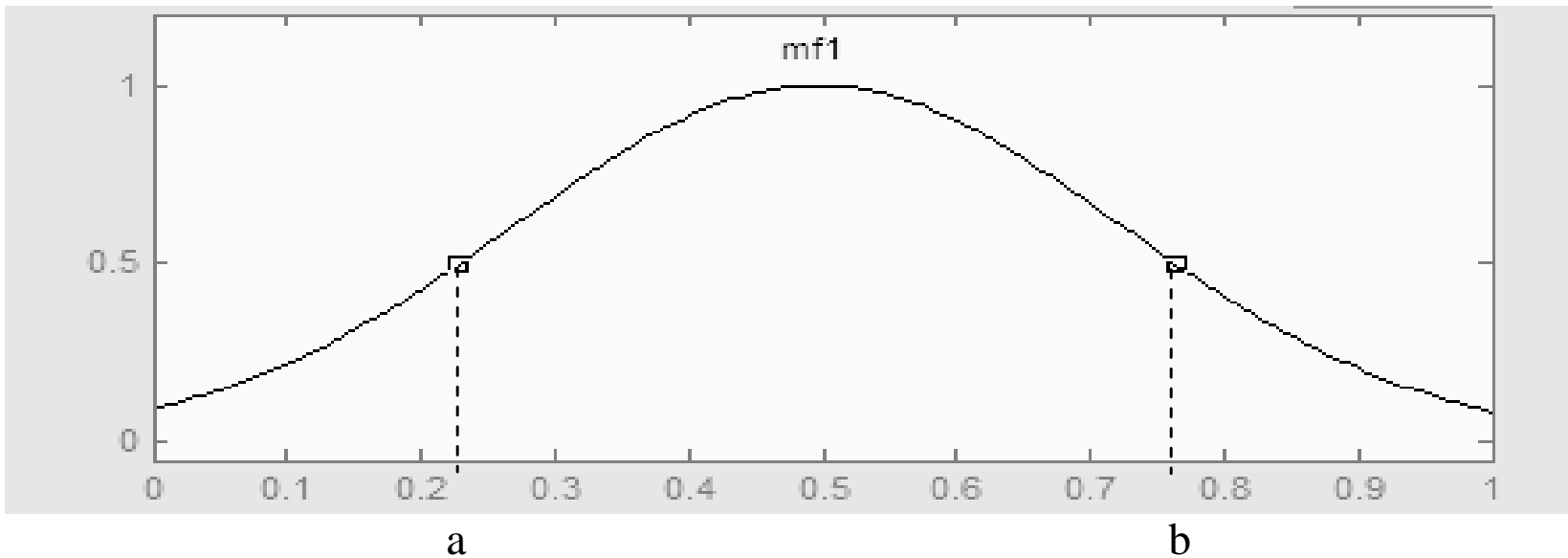
# Cont...



# Gaussian membership function

$$\mu(x, a, b) = e^{\frac{-(x-b)^2}{2a^2}}$$

The graph given in Fig. 10.6 is for parameters  $a = 0.22$ ,  $b = 0.78$



**Figure Gaussian Membership Function**

# Applications of Fuzzy Logic

Following are the different application areas where the Fuzzy Logic concept is widely used:

- It is used in **Businesses** for decision-making support system.
- It is used in **Automotive systems** for controlling the traffic and speed, and for improving the efficiency of automatic transmissions. **Automotive systems** also use the shift scheduling method for automatic transmissions.
- This concept is also used in the **Defence** in various areas. Defence mainly uses the Fuzzy logic systems for underwater target recognition and the automatic target recognition of thermal infrared images.
- It is also widely used in the **Pattern Recognition and Classification** in the form of Fuzzy logic-based recognition and handwriting recognition. It is also used in the searching of fuzzy images.

# Applications of Fuzzy Logic

- Fuzzy logic systems also used in **Securities**.
- It is also used in **microwave oven** for setting the power and cooking strategy.
- This technique is also used in the area of **modern control systems** such as expert systems.
- **Finance** is also another application where this concept is used for predicting the stock market, and for managing the funds.
- It is also used for controlling the brakes.
- It is also used in the **industries of chemicals** for controlling the pH, and chemical distillation process.
- It is also used in the **industries of manufacturing** for the optimization of milk and cheese production.
- It is also used in the vacuum cleaners, and the timings of washing machines.
- It is also used in heaters, air conditioners, and humidifiers.

# Utility Theory and utility functions

- Decision theory, in its simplest form, deals with choosing among actions based on the desirability of their immediate outcomes
- If agent may not know the current state and define  $\text{RESULT}(a)$  as a random variable whose values are the possible outcome states. The probability of outcome  $s$ , given evidence observations  $e$ , is written

$$P(\text{RESULT}(a) = s \mid a, e)$$

where the  $a$  on the right-hand side of the conditioning bar stands for the event that action  $a$  is executed

- The agent's preferences are captured by a utility function,  $U(s)$ , which assigns a single number to express the desirability of a state.

- The expected utility of an action given the evidence,  $EU(a|e)$ , is just the average utility value of the outcomes, weighted by the probability that the outcome occurs:

$$EU(a|e) = \sum_{s'} P(\text{RESULT}(a) = s' \mid a, e) U(s')$$

The principle of maximum expected utility (MEU) says that a rational agent should choose the action that maximizes the agent's expected utility:

$$\text{action} = \underset{a}{\operatorname{argmax}} EU(a|e)$$

In a sense, the MEU principle could be seen as defining all of AI. All an intelligent agent has to do is calculate the various quantities, maximize utility over its actions, and away it goes.



# Basis of Utility Theory

- Intuitively, the principle of Maximum Expected Utility (MEU) seems like a reasonable way to make decisions, but it is by no means obvious that it is the only rational way.
- Why should maximizing the average utility be so special?
- What's wrong with an agent that maximizes the weighted sum of the cubes of the possible utilities, or tries to minimize the worst possible loss?
- Could an agent act rationally just by expressing preferences between states, without giving them numeric values?
- Finally, why should a utility function with the required properties exist at all?

# Constraints on rational preferences

- These questions can be answered by writing down some constraints on the preferences that a rational agent should have and then showing that the MEU principle can be derived from the constraints

$A \succ B$  the agent prefers A over B.

$A \sim B$  the agent is indifferent between A and B.

$A \succeq B$  the agent prefers A over B or is indifferent between them.

We can think of the set of outcomes for each action as a lottery—think of each action as a ticket. A lottery L with possible outcomes  $S_1, \dots, S_n$  that occur with probabilities  $p_1, \dots, p_n$  is written

$L = [p_1, S_1; p_2, S_2; \dots p_n, S_n]$  .

# Constraints on rational preferences

- In general, each outcome  $S_i$  of a lottery can be either an atomic state or another lottery. The primary issue for utility theory is to understand how preferences between complex lotteries are related to preferences between the underlying states in those lotteries.

To address this issue we list six constraints that we require any reasonable preference relation to obey:

- **Orderability:** Given any two lotteries, a rational agent must either prefer one to the other or else rate the two as equally preferable. That is, the agent cannot avoid deciding.

Exactly one of  $(A \succ B)$ ,  $(B \succ A)$ , or  $(A \sim B)$  holds.

# Constraints on rational preferences

- **Transitivity:** Given any three lotteries, if an agent prefers A to B and prefers B to C, then the agent must prefer A to C.

$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$

- **Continuity:** If some lottery B is between A and C in preference, then there is some probability p for which the rational agent will be indifferent between getting B for sure and the lottery that yields A with probability p and C with probability 1 – p.

$$A \succ B \succ C \Rightarrow \exists p [p, A; 1 - p, C] \sim B .$$

- **Substitutability:** If an agent is indifferent between two lotteries A and B, then the agent is indifferent between two more complex lotteries that are the same except that B is substituted for A in one of them. This holds regardless of the probabilities and the other outcome(s) in the lotteries.

$$A \sim B \Rightarrow [p, A; 1 - p, C] \sim [p, B; 1 - p, C] .$$

This also holds if we substitute  $\succ$  for  $\sim$  in this axiom.

# Constraints on rational preferences

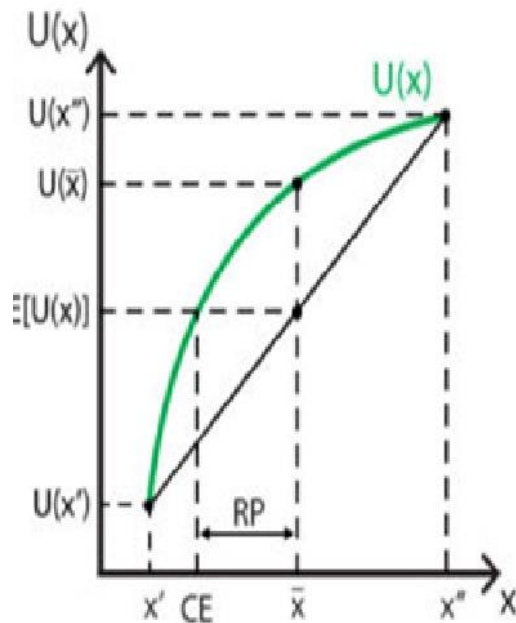
- **Monotonicity:** Suppose two lotteries have the same two possible outcomes, A and B. If an agent prefers A to B, then the agent must prefer the lottery that has a higher probability for A (and vice versa).

$$A \succ B \Rightarrow (p > q \Leftrightarrow [p, A; 1 - p, B] \succ [q, A; 1 - q, B])$$

- **Decomposability:** Compound lotteries can be reduced to simpler ones using the laws of probability. This has been called the “no fun in gambling” rule because it says that two consecutive lotteries can be compressed into a single equivalent lottery.

$$[p, A; 1 - p, [q, B; 1 - q, C]] \sim [p, A; (1 - p)q, B; (1 - p)(1 - q), C] .$$

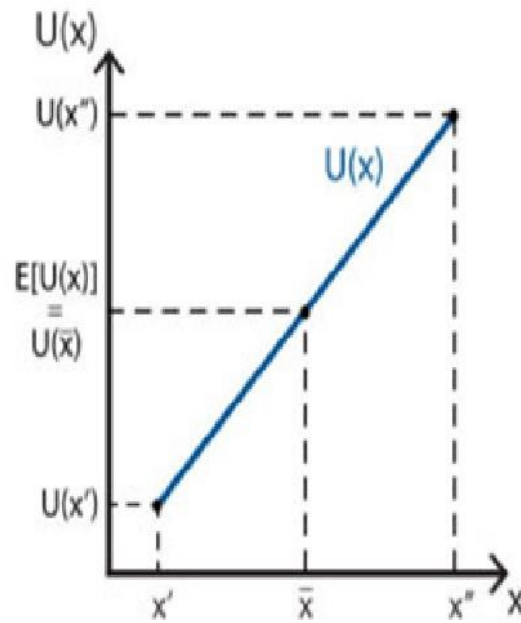
# Expected Utilities



Risk averse individual

$$E[U(x)] < U(\bar{x})$$

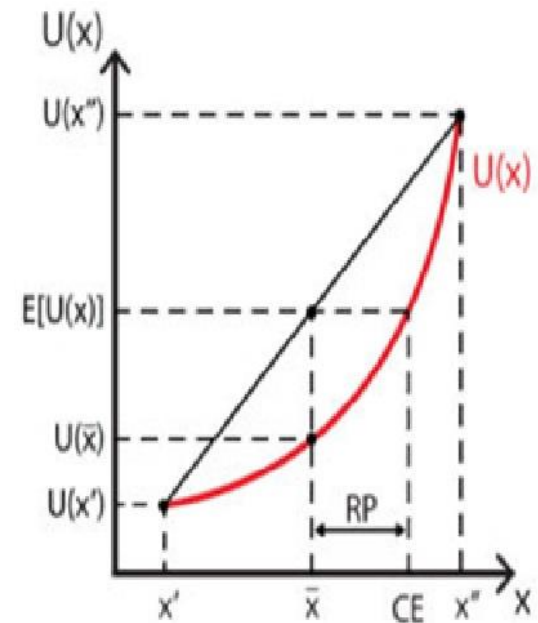
$$CE < \bar{x}$$



Risk neutral individual

$$E[U(x)] = U(\bar{x})$$

$$CE = \bar{x}$$



Risk loving individual

$$E[U(x)] > U(\bar{x})$$

$$CE > \bar{x}$$

# UNIT-4

Learning

# **Machine Learning Paradigms**



# What is learning?

- “Learning denotes changes in a system that ... enable a system to do the same task more efficiently the next time.”  
–Herbert Simon
- “Learning is constructing or modifying representations of what is being experienced.”  
–Ryszard Michalski
- “Learning is making useful changes in our minds.” –  
Marvin Minsky

# Paradigms in Machine Learning

- A paradigm, as most of us know, is a set of ideas, assumptions and values held by an entity and they shape the way that entity interacts with their environment.
- For machine learning, this translates into the set of policies and assumptions inherited by a machine learning algorithm which dictate how it interacts with both the data inputs and the user.

# Machine Learning

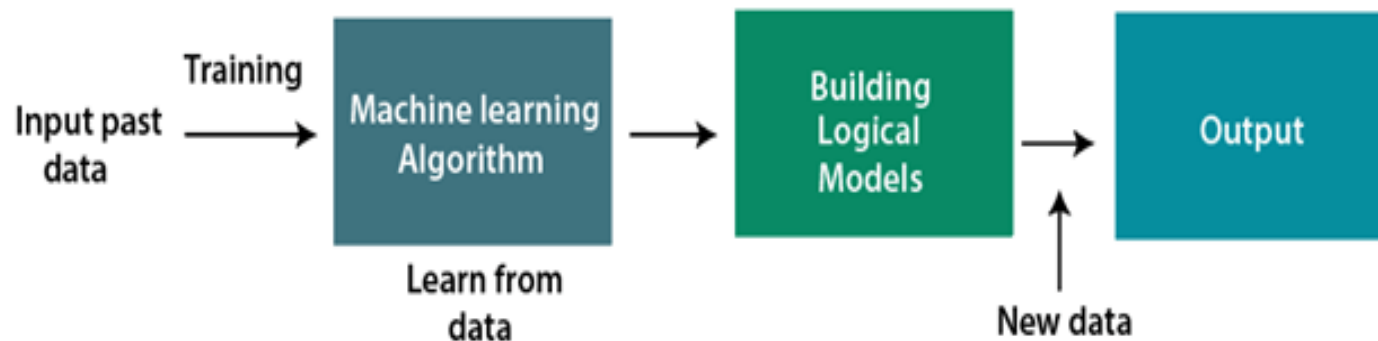
- Machine Learning is said as a subset of **artificial intelligence** that is mainly concerned with the development of algorithms which allow a computer to learn from the data and past experiences on their own. The term machine learning was first introduced by **Arthur Samuel** in **1959**. We can define it in a summarized way as:
- **Machine learning enables a machine to automatically learn from data, improve performance from experiences, and predict things without being explicitly programmed.**

# Machine Learning

- With the help of sample historical data, which is known as **training data**, machine learning algorithms build a **mathematical model** that helps in making predictions or decisions without being explicitly programmed. Machine learning brings computer science and statistics together for creating predictive models. Machine learning constructs or uses the algorithms that learn from historical data. The more we will provide the information, the higher will be the performance.
- **A machine has the ability to learn if it can improve its performance by gaining more data.**

# How does Machine Learning work

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



# Classification of Machine Learning

At a broad level, machine learning can be classified into three types:

- **Supervised learning**
- **Unsupervised learning**
- **Reinforcement learning**

# Supervised Learning

- Supervised learning is a type of machine learning method in which we provide sample labeled data to the machine learning system in order to train it, and on that basis, it predicts the output.
- The system creates a model using labeled data to understand the datasets and learn about each data, once the training and processing are done then we test the model by providing a sample data to check whether it is predicting the exact output or not.
- The goal of supervised learning is to map input data with the output data. The supervised learning is based on supervision, and it is the same as when a student learns things in the supervision of the teacher. The example of supervised learning is **spam filtering**.

# Supervised Learning

Supervised learning can be grouped further in two categories of algorithms:

- **Classification**
- **Regression**

## **Classification:**

- Classification is a process of categorizing a given set of data into classes, It can be performed on both structured or unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories.



# Classification

- The classification predictive modeling is the task of approximating the mapping function from input variables to discrete output variables. The main goal is to identify which class/category the new data will fall into.
- Heart disease detection can be identified as a classification problem, this is a binary classification since there can be only two classes i.e has heart disease or does not have heart disease. The classifier, in this case, needs training data to understand how the given input variables are related to the class. And once the classifier is trained accurately, it can be used to detect whether heart disease is there or not for a particular patient.

# Classification

- Since classification is a type of supervised learning, even the targets are also provided with the input data. Let us get familiar with the classification in machine learning terminologies.

**Examples of supervised machine learning algorithms for classification are:**

- Decision Tree Classifiers
- Support Vector Machines
- Naive Bayes Classifiers
- K Nearest Neighbor
- Artificial Neural Networks

# Regression

- The regression algorithms attempt to estimate the mapping function ( $f$ ) from the input variables ( $x$ ) to numerical or continuous output variables ( $y$ ). Now, the output variable could be a real value, which can be an integer or a floating point value. Therefore, the regression prediction problems are usually quantities or sizes.
- For example, if you are provided with a dataset about houses, and you are asked to predict their prices, that is a regression task because the price will be a continuous output.

## **Examples of supervised machine learning algorithms for regression:**

- Linear Regression
- Logistic Regression
- Regression Decision Trees
- Artificial Neural Networks

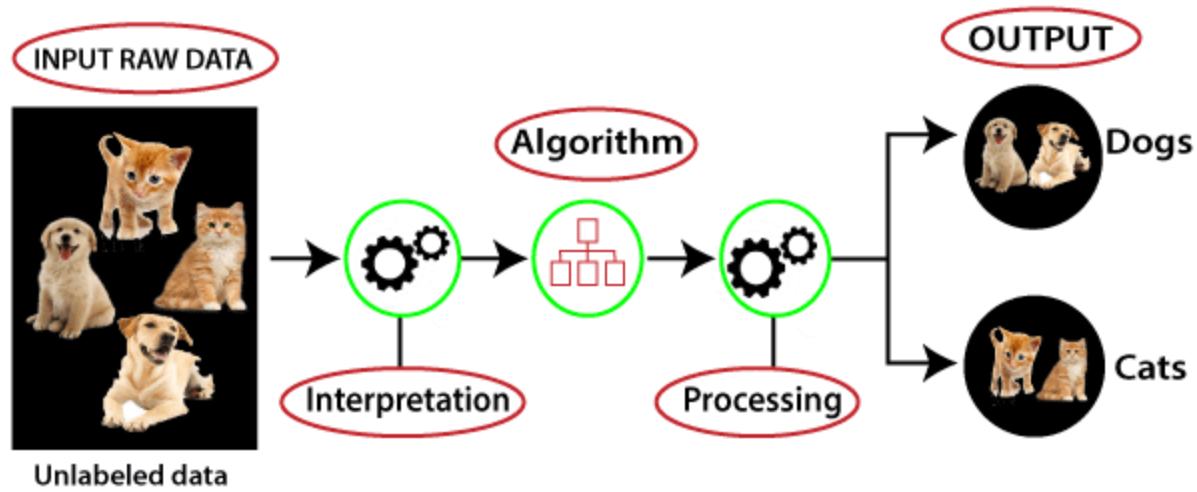
# Unsupervised Learning

- As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as:
- Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.
- Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data.

# Unsupervised Learning

- The goal of unsupervised learning is to **find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.**
- **Example:** Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will perform this task by clustering the image dataset into the groups according to similarities between images.

# Unsupervised Learning



**Working of Unsupervised Learning**

# Unsupervised Learning

- The unsupervised learning algorithm can be further categorized into two types of problems:
- **Clustering:** Clustering is a method of grouping the objects into clusters such that objects with most similarities remain into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.
- **Association:** An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.

# Unsupervised Learning

Below is the list of some popular unsupervised learning algorithms:

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



# Inductive learning

- Inductive Learning Algorithm (ILA) is an iterative and inductive machine learning algorithm which is used for generating a set of a classification rule, which produces rules of the form “IF-THEN”, for a set of examples, producing rules at each iteration and appending to the set of rules.
- In this first input  $x$ , (the verified value) given to a function  $f$  and the output is  $f(x)$ .
- Then we can give different set of inputs (raw inputs) to the same function  $f$ , and verify the output  $f(x)$ .
- By using the outputs we generate (learn) the rules.

# Basic Idea:

- There are basically two methods for knowledge extraction firstly from domain experts and then with machine learning.
- For a very large amount of data, the domain experts are not very useful and reliable. So we move towards the machine learning approach for this work.
- To use machine learning One method is to replicate the experts logic in the form of algorithms but this work is very tedious, time taking and expensive.
- So we move towards the inductive algorithms which itself generate the strategy for performing a task and need not instruct separately at each step.

# Inductive learning

- Inductive learning also known as discovery learning, is a process where the learner discovers rules by observing examples.
- We can often work out rules for ourselves by observing examples. If there is a pattern; then record it.
- We then apply the rule in different situations to see if it works.
- With inductive language learning, tasks are designed specifically to guide the learner and assist them in discovering a rule.

# Inductive learning

- Inductive learning: System tries to make a “general rule” from a set of observed instances.

- Example:

Mango  $\rightarrow$  f(Mango)  $\rightarrow$  sweet (e1)

Banana  $\rightarrow$  f(Banana)  $\rightarrow$  sweet (e2)

.....

Fruits  $\rightarrow$  f(Fruits)  $\rightarrow$  sweet (general rule)

# Example

- Suppose an example set having attributes - Place type, weather, location, decision and seven examples.
- Our task is to generate a set of rules that under what condition what is the decision.

Example no.	Place type	weather	location	decision
I )	hilly	winter	kullu	Yes
II )	mountain	windy	Mumbai	No
III )	mountain	windy	Shimla	Yes
IV )	beach	windy	Mumbai	No
V )	beach	warm	goa	Yes
VI )	beach	windy	goa	No
VII )	beach	warm	Shimla	Yes

### Subset 1:

s.no	place type	weather	location	decision
1	hilly	winter	kullu	Yes
2	mountain	windy	Shimla	Yes
3	beach	warm	goa	Yes
4	beach	warm	Shimla	Yes

### Subset 2:

s.no	place type	weather	location	decision
5	mountain	windy	Mumbai	No
6	beach	windy	Mumbai	No
7	beach	windy	goa	No

- **at iteration 1**

row 3 & 4 column **weather** is selected and row 3 & 4 are marked.  
the rule is added to R IF weather is warm then a decision is yes.

- **at iteration 2**

row 1 column **place type** is selected and row 1 is marked.  
the rule is added to R IF place type is hilly then the decision is yes.

- **at iteration 3**

row 2 column **location** is selected and row 2 is marked.  
the rule is added to R IF location is Shimla then the decision is yes.

- **at iteration 4**

row 5&6 column **location** is selected and row 5&6 are marked.  
the rule is added to R IF location is Mumbai then a decision is no.

- **at iteration 5**

row 7 column **place type & the weather** is selected and row 7 is marked.  
rule is added to R IF place type is beach AND weather is windy then the decision is no.

**finally we get the rule set :-**

## **Rule Set**

**Rule 1:** IF the weather is warm THEN the decision is yes.

**Rule 2:** IF place type is hilly THEN the decision is yes.

**Rule 3:** IF location is Shimla THEN the decision is yes.

**Rule 4:** IF location is Mumbai THEN the decision is no.

**Rule 5:** IF place type is beach AND the weather is windy  
THEN the decision is no.



# 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.

# Decision Trees



An example of a decision tree can be explained using above binary tree. Let's say you want to predict whether a person is fit given their information like age, eating habit, and physical activity, etc. The decision nodes here are questions like **'What's the age?'**, **'Does he exercise?'**, **'Does he eat a lot of pizzas?'** And the leaves, which are outcomes like either 'fit', or 'unfit'. In this case this was a binary classification problem (a yes no type problem).

# Decision Trees

There are two main types of Decision Trees:

## **Classification trees** (Yes/No types)

- What we've seen above is an example of classification tree, where the outcome was a variable like 'fit' or 'unfit'. Here the decision variable is **Categorical**.

## **Regression trees** (Continuous data types)

- Here the decision or the outcome variable is **Continuous**, e.g. a number like 123.
- **Working** Now that we know what a Decision Tree is, we'll see how it works internally.
- There are many algorithms out there which construct Decision Trees, but one of the best is called as **ID3 Algorithm**. ID3 Stands for **Iterative Dichotomiser 3**.

# Decision Trees

- ID3 algorithm is a **classification algorithm** that follows a **greedy approach** of building a decision tree by selecting a best attribute that yields **maximum Information Gain (IG)** or **minimum Entropy (H)**.
- **Entropy:** Entropy is the measures of **impurity, disorder** or **uncertainty** in a bunch of examples.
- Entropy controls how a Decision Tree decides to **split** the data. It actually effects how a **Decision Tree** draws its boundaries.
- **The Equation of Entropy:**

$$Entropy = - \sum p(X) \log p(X)$$



here  $p(x)$  is a fraction of examples in a given class

# Decision Trees

- **Information gain (IG):** It measures how much “information” a feature gives us about the class. It is also called as Kullback-Leibler divergence

## Why it matter ?

- Information gain is the main key that is used by Decision Tree Algorithms to construct a Decision Tree.
- Decision Trees algorithm will always tries to maximize Information gain.
- An attribute with highest Information gain will tested/split first.
- The Equation of Information gain:

$$\text{Information gain} = \text{entropy (parent)} - [\text{weightes average}] * \text{entropy (children)}$$

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

**Entropy calculations:** Entropy of the current state. In the above example, we can see in total there are 5 No's and 9 Yes's.

$$Entropy(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

$$Entropy(S) = -\left(\frac{9}{14}\right) \log_2 \left(\frac{9}{14}\right) - \left(\frac{5}{14}\right) \log_2 \left(\frac{5}{14}\right) \\ = 0.940$$

Remember that the Entropy is 0 if all members belong to the same class, and 1 when half of them belong to one class and other half belong to other class that is perfect randomness.

$$Entropy(S_{strong}) = -\left(\frac{3}{6}\right) \log_2 \left(\frac{3}{6}\right) - \left(\frac{3}{6}\right) \log_2 \left(\frac{3}{6}\right) \\ = 1.000$$

In the above equation, out of 6 Strong examples, we have **3 examples where the outcome was 'Yes' for Play Golf** and **3 where we had 'No' for Play Golf**.

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

## Information Gain (IG) Calculation:

$$IG(S, Wind) = H(S) - \sum_{i=0}^n P(x) * H(x)$$

where 'x' are the possible values for an attribute. Here, attribute 'Wind' takes two possible values in the sample data, hence  $x = \{Weak, Strong\}$  We'll have to calculate:

1.  $H(S_{weak})$
2.  $H(S_{strong})$
3.  $P(S_{weak})$
4.  $P(S_{strong})$
5.  $H(S) = 0.94$  which we had already calculated in the previous example

Amongst all the 14 examples we have **8** places where the wind is weak and **6** where the wind is Strong.

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

## Information Gain (IG) Calculation:

$$\begin{aligned}
 P(S_{weak}) &= \frac{\text{Number of Weak}}{\text{Total}} \\
 &= \frac{8}{14} \\
 P(S_{strong}) &= \frac{\text{Number of Strong}}{\text{Total}} \\
 &= \frac{6}{14}
 \end{aligned}$$

Now out of the 8 Weak examples, 6 of them were 'Yes' for Play Golf and 2 of them were 'No' for 'Play Golf'. So, we

$$\begin{aligned}
 Entropy(S_{weak}) &= -\left(\frac{6}{8}\right)\log_2\left(\frac{6}{8}\right) - \left(\frac{2}{8}\right)\log_2\left(\frac{2}{8}\right) \\
 &= 0.811
 \end{aligned}$$

We already calculated  $Entropy(S_{strong})=1$

$$\begin{aligned}
 IG(S, Wind) &= H(S) - \sum_{i=0}^n P(x) * H(x) \\
 IG(S, Wind) &= H(S) - P(S_{weak}) * H(S_{weak}) - P(S_{strong}) * H(S_{strong}) \\
 &= 0.940 - \left(\frac{8}{14}\right)(0.811) - \left(\frac{6}{14}\right)(1.00) \\
 &= 0.048
 \end{aligned}$$



# Draw a decision tree for the given data set using ID3 algorithm

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

In the given example there are four attributes {outlook, temperature, humidity, wing} and there is class which contains binary values i.e., yes or no.

We need to calculate information gain for each attribute so that we can decide which attribute will be taken as a root node for drawing a decision tree.

So for calculating Information gain we need to calculate Entropy values.

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute : Outlook

Values(Outlook) = Sunny, Overcast, Rain

$$S = [9+, 5-]$$

$$Entropy(S) = -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14} = 0.94$$

$$S_{Sunny} \leftarrow [2+, 3-]$$

$$Entropy(S_{Sunny}) = -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} = 0.971$$

$$S_{Overcast} \leftarrow [4+, 0-]$$

$$Entropy(S_{Overcast}) = -\frac{4}{4} \log_2 \frac{4}{4} - \frac{0}{4} \log_2 \frac{0}{4} = 0$$

$$S_{Rain} \leftarrow [3+, 2-]$$

$$Entropy(S_{Rain}) = -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} = 0.971$$

$$Gain(S, Outlook) = Entropy(S) - \sum_{v \in \{Sunny, Overcast, Rain\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Outlook)$$

$$= Entropy(S) - \frac{5}{14} Entropy(S_{Sunny}) - \frac{4}{14} Entropy(S_{Overcast}) - \frac{5}{14} Entropy(S_{Rain})$$

$$Gain(S, Outlook) = 0.94 - \frac{5}{14} 0.971 - \frac{4}{14} 0 - \frac{5}{14} 0.971 = 0.2464$$

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute : Temperature

Values(Temperature) = Hot, Mild, Cool

$$S = [9+, 5-] \quad Entropy(S) = -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14} = 0.94$$

$$S_{Hot} \leftarrow [2+, 2-] \quad Entropy(S_{Hot}) = -\frac{2}{4} \log_2 \frac{2}{4} - \frac{2}{4} \log_2 \frac{2}{4} = 1.0$$

$$S_{Mild} \leftarrow [4+, 2-] \quad Entropy(S_{Mild}) = -\frac{4}{6} \log_2 \frac{4}{6} - \frac{2}{6} \log_2 \frac{2}{6} = 0.9183$$

$$S_{Cool} \leftarrow [3+, 1-] \quad Entropy(S_{Cool}) = -\frac{3}{4} \log_2 \frac{3}{4} - \frac{1}{4} \log_2 \frac{1}{4} = 0.8113$$

$$Gain(S, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Temp)$$

$$= Entropy(S) - \frac{4}{14} Entropy(S_{Hot}) - \frac{6}{14} Entropy(S_{Mild})$$

$$- \frac{4}{14} Entropy(S_{Cool})$$

$$\begin{aligned} Gain(S, Temperature) &= 0.94 - (4/14)(1.0) \\ &\quad - (6/14)(0.9183) \\ &\quad - (4/14)(0.8113) \\ &= 0.0289 \end{aligned}$$

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute : Humidity

Values(Humidity) = High, Normal

$$S = [9+, 5-] \quad Entropy(S) = -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14} = 0.94$$

$$S_{High} \leftarrow [3+, 4-] \quad Entropy(S_{High}) = -\frac{3}{7} \log_2 \frac{3}{7} - \frac{4}{7} \log_2 \frac{4}{7} = 0.9852$$

$$S_{Normal} \leftarrow [6+, 1-] \quad Entropy(S_{Normal}) = -\frac{6}{7} \log_2 \frac{6}{7} - \frac{1}{7} \log_2 \frac{1}{7} = 0.5916$$

$$Gain(S, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Humidity)$$

$$= Entropy(S) - \frac{7}{14} Entropy(S_{High}) - \frac{7}{14} Entropy(S_{Normal})$$

$$Gain(S, Humidity) = 0.94 - \frac{7}{14} 0.9852 - \frac{7}{14} 0.5916 = 0.1516$$

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute : Wind

Values(Wind) = Strong, Weak

$$S = [9+, 5-] \quad Entropy(S) = -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14} = 0.94$$

$$S_{Strong} \leftarrow [3+, 3-] \quad Entropy(S_{Strong}) = 1.0$$

$$S_{Weak} \leftarrow [6+, 2-] \quad Entropy(S_{Weak}) = -\frac{6}{8} \log_2 \frac{6}{8} - \frac{2}{8} \log_2 \frac{2}{8} = 0.8113$$

$$Gain(S, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Wind) = Entropy(S) - \frac{6}{14} Entropy(S_{Strong}) - \frac{8}{14} Entropy(S_{Weak})$$

$$Gain(S, Wind) = 0.94 - \frac{6}{14} 1.0 - \frac{8}{14} 0.8113 = 0.0478$$

- We calculating information gain for all attributes:

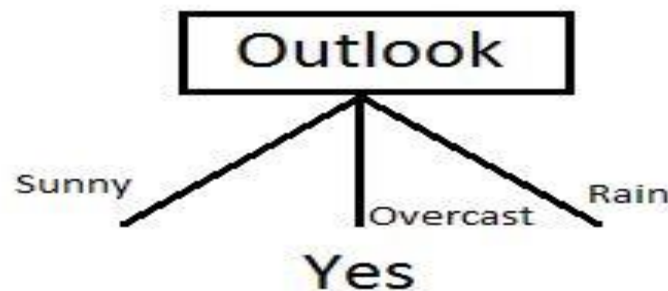
$$\text{Gain}(S, \text{Outlook}) = 0.2464,$$

$$\text{Gain}(S, \text{Temperature}) = 0.0289$$

$$\text{Gain}(S, \text{Humidity}) = 0.1516$$

$$\text{Gain}(S, \text{Wind}) = 0.0478$$

- We can clearly see that  $\text{IG}(S, \text{Outlook})$  has the highest information gain of 0.246, hence we chose **Outlook attribute as the root node**. At this point, the decision tree looks like.



- Here we observe that whenever the outlook is Overcast, Play Golf is always 'Yes', it's no coincidence by any chance, the simple tree resulted because of **the highest information gain is given by the attribute Outlook**.
- Now how do we proceed from this point? We can simply apply **recursion**, you might want to look at the algorithm steps described earlier.
- Now that we've used Outlook, we've got three of them remaining Humidity, Temperature, and Wind. And, we had three possible values of Outlook: Sunny, Overcast, Rain.
- Where the Overcast node already ended up having leaf node 'Yes', so we're left with two subtrees to compute: Sunny and Rain.



Attribute : Temperature

Values(Temperature) = Hot, Mild, Cool

Day	Temperature	Humidity	Wind	Play Golf
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

$$S_{Sunny} = [2+, 3-] \quad Entropy(S_{Sunny}) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.97$$

$$S_{Hot} \leftarrow [0+, 2-] \quad Entropy(S_{Hot}) = 0.0$$

$$S_{Mild} \leftarrow [1+, 1-] \quad Entropy(S_{Mild}) = 1.0$$

$$S_{Cool} \leftarrow [1+, 0-] \quad Entropy(S_{Cool}) = 0.0$$

$$Gain(S_{Sunny}, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Temp)$$

$$= Entropy(S) - \frac{2}{5}Entropy(S_{Hot}) - \frac{2}{5}Entropy(S_{Mild})$$

$$- \frac{1}{5}Entropy(S_{Cool})$$

$$Gain(S_{Sunny}, Temp) = 0.97 - \frac{2}{5}0.0 - \frac{2}{5}1 - \frac{1}{5}0.0 = 0.570$$

Day	Temperature	Humidity	Wind	Play Golf
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

Attribute : Humidity

Values(Humidity) = High, Normal

$$S_{Sunny} = [2+, 3-] \quad Entropy(S) = -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} = 0.97$$

$$S_{high} \leftarrow [0+, 3-] \quad Entropy(S_{high}) = 0.0$$

$$S_{Normal} \leftarrow [2+, 0-] \quad Entropy(S_{Normal}) = 0.0$$

$$Gain(S_{Sunny}, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Humidity) = Entropy(S) - \frac{3}{5} Entropy(S_{High}) - \frac{2}{5} Entropy(S_{Normal})$$

$$Gain(S_{Sunny}, Humidity) = 0.97 - \frac{3}{5} 0.0 - \frac{2}{5} 0.0 = 0.97$$

Attribute : Wind

Values(Wind) = Strong, Weak

Day	Temperature	Humidity	Wind	Play Golf
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

$$S_{Sunny} = [2+, 3-]$$

$$Entropy(S) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.97$$

$$S_{Strong} \leftarrow [1+, 1-]$$

$$Entropy(S_{Strong}) = 1.0$$

$$S_{Weak} \leftarrow [1+, 2-]$$

$$Entropy(S_{Weak}) = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0.9183$$

$$Gain(S_{Sunny}, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

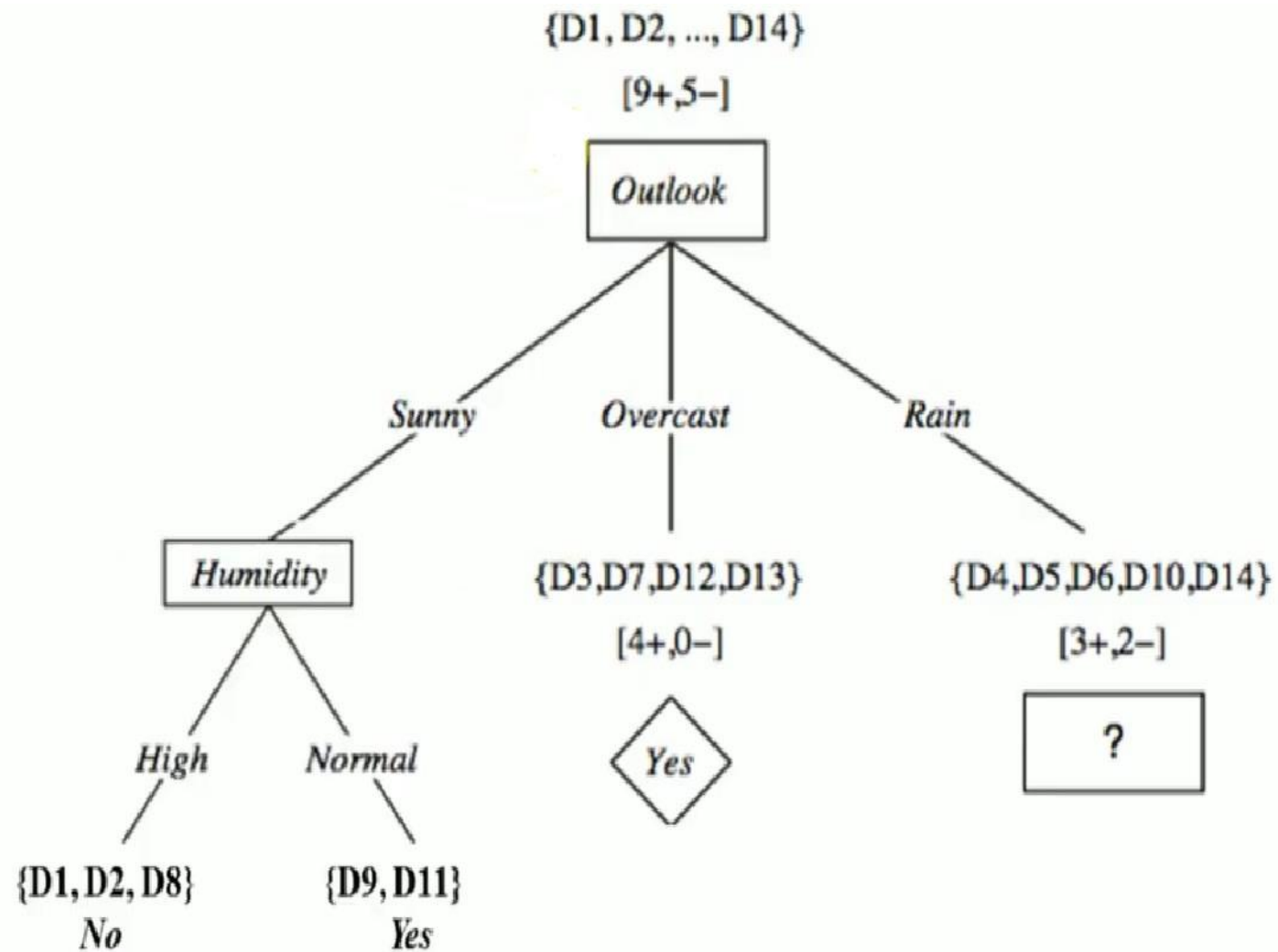
$$Gain(S_{Sunny}, Wind) = Entropy(S) - \frac{2}{5} Entropy(S_{Strong}) - \frac{3}{5} Entropy(S_{Weak})$$

$$Gain(S_{sunny}, Wind) = 0.97 - \frac{2}{5} 1.0 - \frac{3}{5} 0.918 = 0.0192$$

$$\textit{Gain}(S_{\textit{sunny}}, \textit{Temperature}) = 0.570$$

$$\textit{Gain}(S_{\textit{sunny}}, \textit{Humidity}) = 0.97$$

$$\textit{Gain}(S_{\textit{sunny}}, \textit{Wind}) = 0.0192$$



Attribute : Temperature

Values(Temperature) = Hot, Mild, Cool

Day	Temperature	Humidity	Wind	Play Golf
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
D10	Mild	Normal	Weak	Yes
D14	Mild	High	Strong	No

$$S_{Rain} = [3+, 2-]$$

$$Entropy(S_{Sunny}) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.97$$

$$S_{Hot} \leftarrow [0+, 0-]$$

$$Entropy(S_{Hot}) = 0.0$$

$$S_{Mild} \leftarrow [2+, 1-]$$

$$Entropy(S_{Mild}) = -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} = 0.9183$$

$$S_{Cool} \leftarrow [1+, 1-]$$

$$Entropy(S_{Cool}) = 1.0$$

$$Gain(S_{Rain}, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Temp)$$

$$= Entropy(S) - \frac{0}{5} Entropy(S_{Hot}) - \frac{3}{5} Entropy(S_{Mild})$$

$$- \frac{2}{5} Entropy(S_{Cool})$$

$$Gain(S_{Rain}, Temp) = 0.97 - \frac{0}{5} 0.0 - \frac{3}{5} 0.918 - \frac{2}{5} 1.0 = 0.0192$$

Day	Temperature	Humidity	Wind	Play Golf
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
D10	Mild	Normal	Weak	Yes
D14	Mild	High	Strong	No

Attribute : Humidity

Values(Humidity) = High, Normal

$$S_{Rain} = [3+, 2-]$$

$$Entropy(S_{Sunny}) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.97$$

$$S_{High} \leftarrow [1+, 1-]$$

$$Entropy(S_{High}) = 1.0$$

$$S_{Normal} \leftarrow [2+, 1-]$$

$$Entropy(S_{Normal}) = -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} = 0.9183$$

$$Gain(S_{Rain}, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Humidity) = Entropy(S) - \frac{2}{5} Entropy(S_{High}) - \frac{3}{5} Entropy(S_{Normal})$$

$$Gain(S_{Rain}, Humidity) = 0.97 - \frac{2}{5} 1.0 - \frac{3}{5} 0.918 = 0.0192$$

Day	Temperature	Humidity	Wind	Play Golf
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
D10	Mild	Normal	Weak	Yes
D14	Mild	High	Strong	No

Attribute : Wind

Values(Wind) = Strong, Weak

$$S_{Rain} = [3+, 2-]$$

$$Entropy(S_{Sunny}) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.97$$

$$S_{Strong} \leftarrow [0+, 2-]$$

$$Entropy(S_{Strong}) = 0.0$$

$$S_{Weak} \leftarrow [3+, 0-]$$

$$Entropy(S_{Weak}) = 0.0$$

$$Gain(S_{Rain}, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Wind) = Entropy(S) - \frac{2}{5} Entropy(S_{Strong}) - \frac{3}{5} Entropy(S_{Weak})$$

$$Gain(S_{Rain}, Wind) = 0.97 - \frac{2}{5} 0.0 - \frac{3}{5} 0.0 = 0.97$$

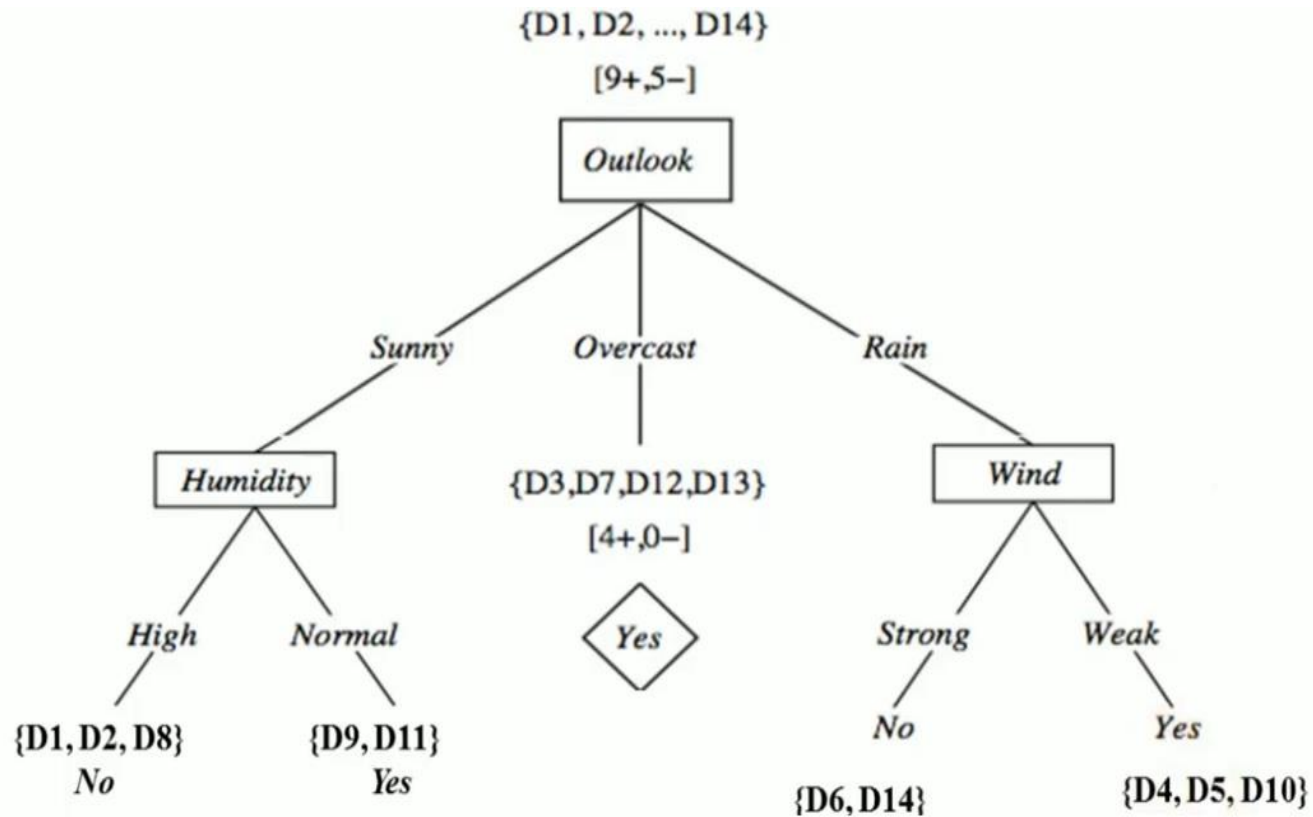


$$\textit{Gain}(S_{rain}, \textit{Temperature}) = 0.0192$$

$$\textit{Gain}(S_{rain}, \textit{Humidity}) = 0.0192$$

$$\textit{Gain}(S_{rain}, \textit{Wind}) = 0.97$$

# Decision Tree



# Neural Networks

- Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial neurons.
- Neuron in ANNs tend to have fewer connections than biological neurons.
- Each neuron in ANN receives a number of inputs.
- An activation function is applied to these inputs which results in activation level of neuron (output value of the neuron).
- Knowledge about the learning task is given in the form of examples called training examples.

# Contd..

- An Artificial Neural Network is specified by:
  - **neuron model**: the information processing unit of the NN,
  - **an architecture**: a set of neurons and links connecting neurons. Each link has a weight,
  - **a learning algorithm**: used for training the NN by modifying the weights in order to model a particular learning task correctly on the training examples.
- The aim is to obtain a NN that is trained and generalizes well.
- It should behaves correctly on new instances of the learning task.

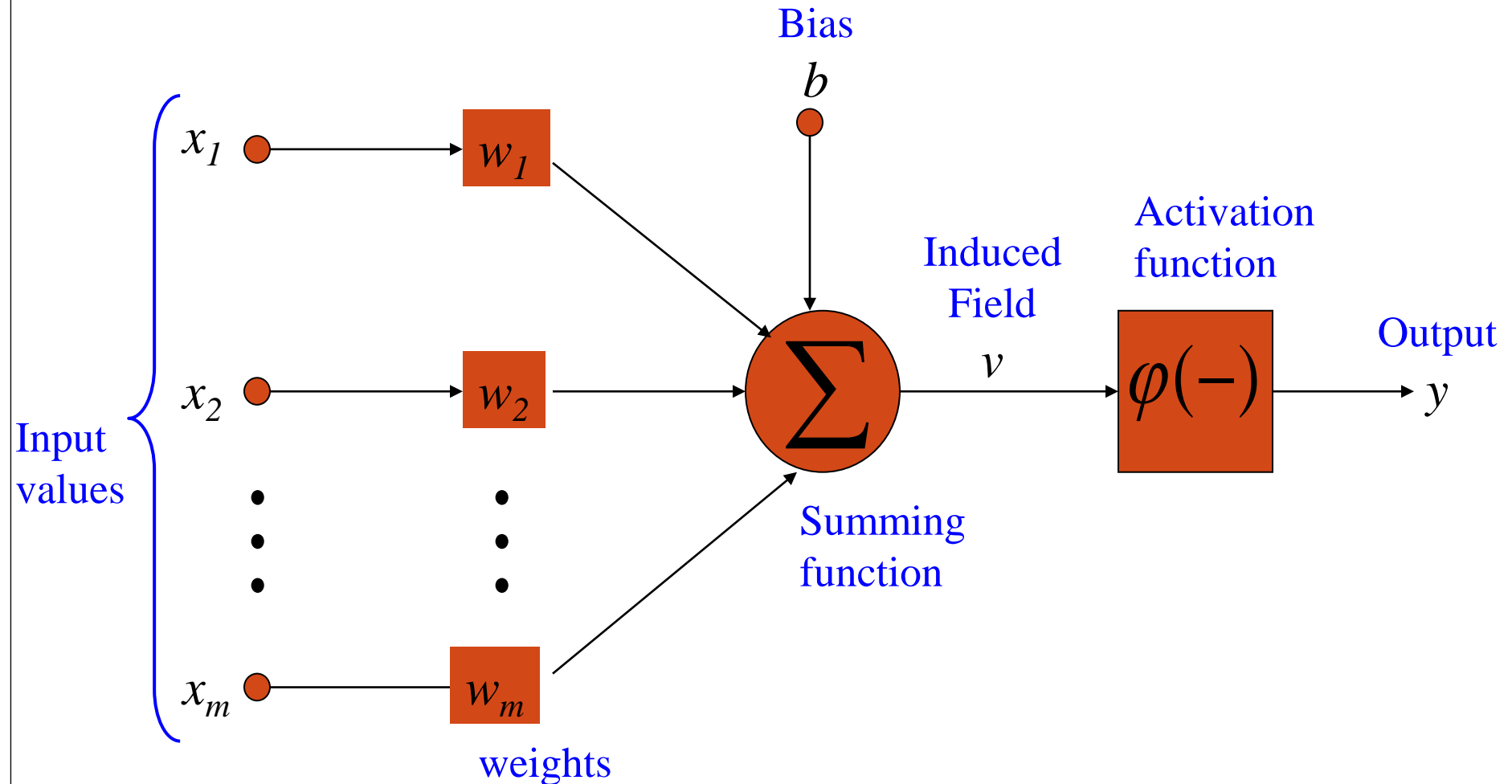
# Neuron

- The neuron is the basic information processing unit of a NN. It consists of:
  - 1 A set of **links**, describing the neuron inputs, with **weights**  $W_1, W_2, \dots, W_m$
  - 2 An **adder** function (linear combiner) for computing the weighted sum of the inputs:  
(real numbers)
  - 3 **Activation function**  $\varphi$  for limiting the amplitude of the neuron output. Here 'b' denotes bias.

$$u = \sum_{j=1}^m w_j x_j$$

$$y = \varphi(u + b)$$

# The Neuron Diagram



# Bias of a Neuron

- The bias  $b$  has the effect of applying a **transformation** to the weighted sum  $u$

$$v = u + b$$

- The bias is an external parameter of the neuron. It can be modeled by adding an extra input.
- $v$  is called **induced field** of the neuron

$$v = \sum_{j=0}^m w_j x_j$$

$$w_0 = b$$

# Neural Networks Activation Functions

- Activation functions are mathematical equations that determine the output of a neural network.
- The function is attached to each neuron in the network, and determines whether it should be activated (“fired”) or not, based on whether each neuron’s input is relevant for the model’s prediction.
- Activation functions also help normalize the output of each neuron to a range between 1 and 0 or between -1 and 1.
- An additional aspect of activation functions is that they must be computationally efficient because they are calculated across thousands or even millions of neurons for each data sample.



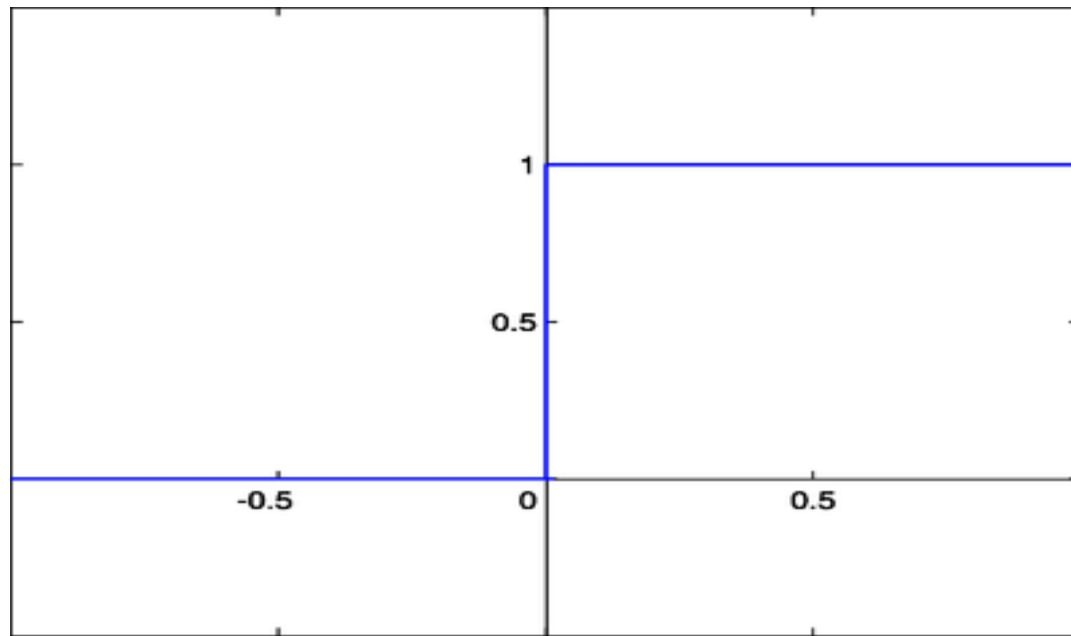
## Contd..

- Modern neural networks use a technique called backpropagation to train the model, which places an increased computational strain on the activation function, and its derivative function.
- It's just a thing function that you use to get the output of node. It is also known as Transfer Function.
- It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).

# Step Function

- A step function is a function like that used by the original Perceptron.
- The output is a certain value,  $A_1$ , if the input sum is above a certain threshold and  $A_0$  if the input sum is below a certain threshold.
- The values used by the Perceptron were  $A_1 = 1$  and  $A_0 = 0$ .

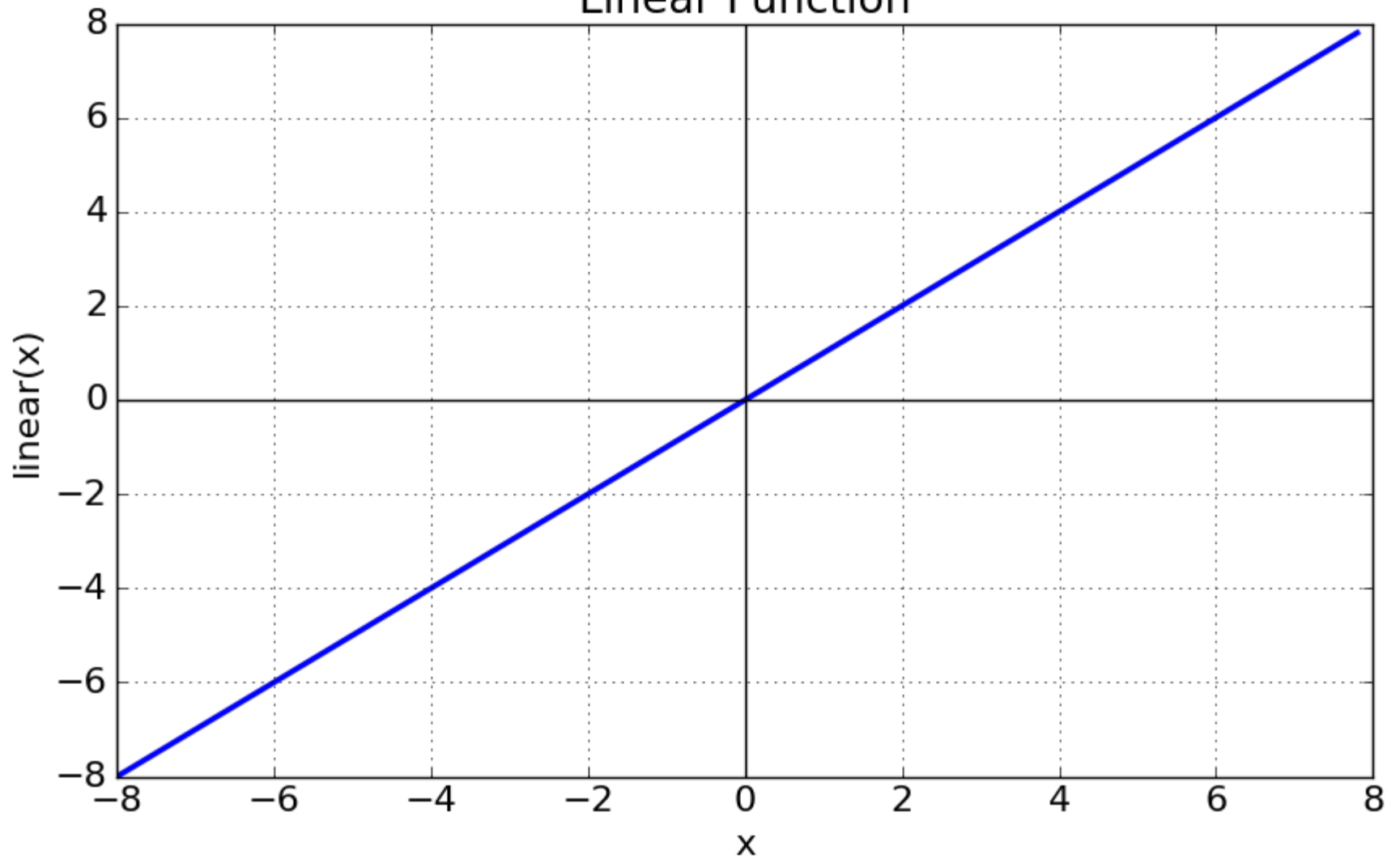
# Step Function



# Linear or Identity Activation Function

- As you can see the function is a line or linear. Therefore, the output of the functions will not be confined between any range.
- **Equation** :  $f(x) = x$
- **Range** : (-infinity to infinity)
- It doesn't help with the complexity or various parameters of usual data that is fed to the neural networks.

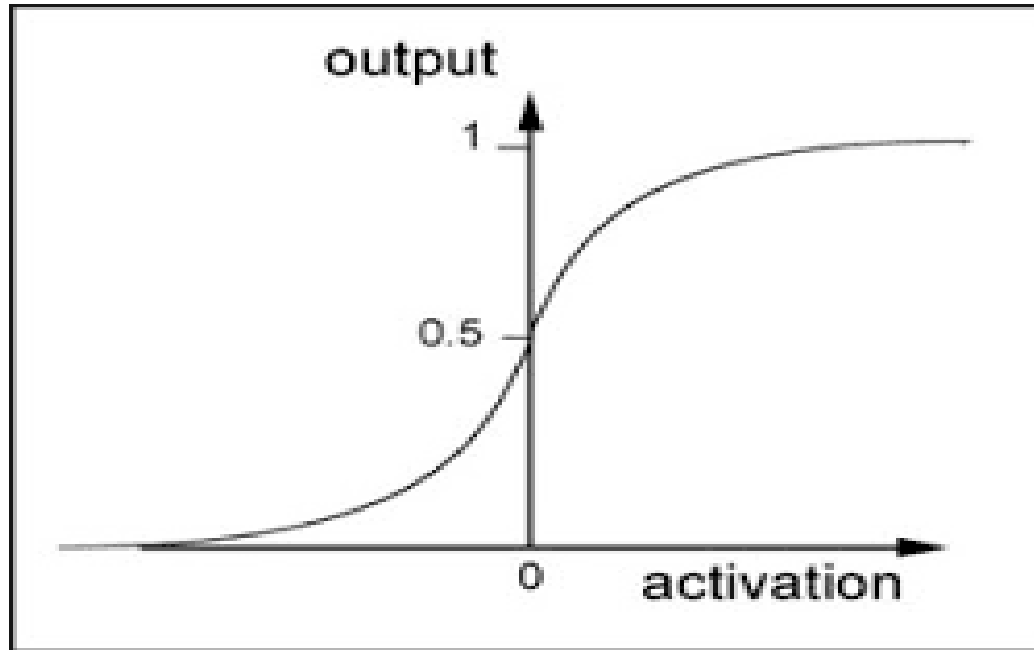
## Linear Function



# Sigmoid or Logistic Activation Function

- The sigmoid function is an activation function where it scales the values between 0 and 1 by applying a threshold.
- The above equation represents a sigmoid function. When we apply the weighted sum in the place of  $X$ , the values are scaled in between 0 and 1.
- The beauty of an exponent is that the value never reaches zero nor exceed 1 in the above equation.
- The large negative numbers are scaled towards 0 and large positive numbers are scaled towards 1.

## Sigmoid Function

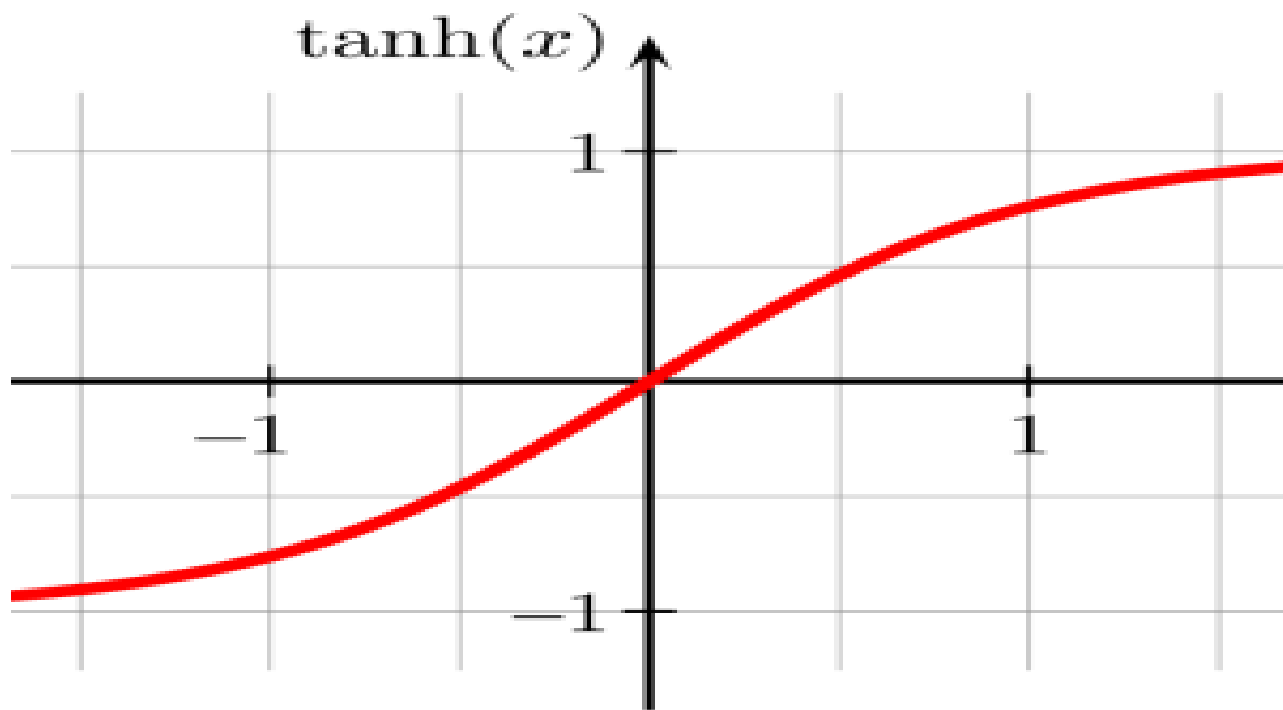


$$f(x) = \frac{1}{1 + e^{-(x)}}$$

# Tanh or Hyperbolic Tangent Function

- The Tanh function is an activation function which re scales the values between -1 and 1 by applying a threshold just like a sigmoid function.
- The advantage i.e the values of a tanh is zero centered which helps the next neuron during propagating.
- When we apply the weighted sum of the inputs in the  $\tanh(x)$ , it re scales the values between -1 and 1. .
- The large negative numbers are scaled towards -1 and large positive numbers are scaled towards 1.





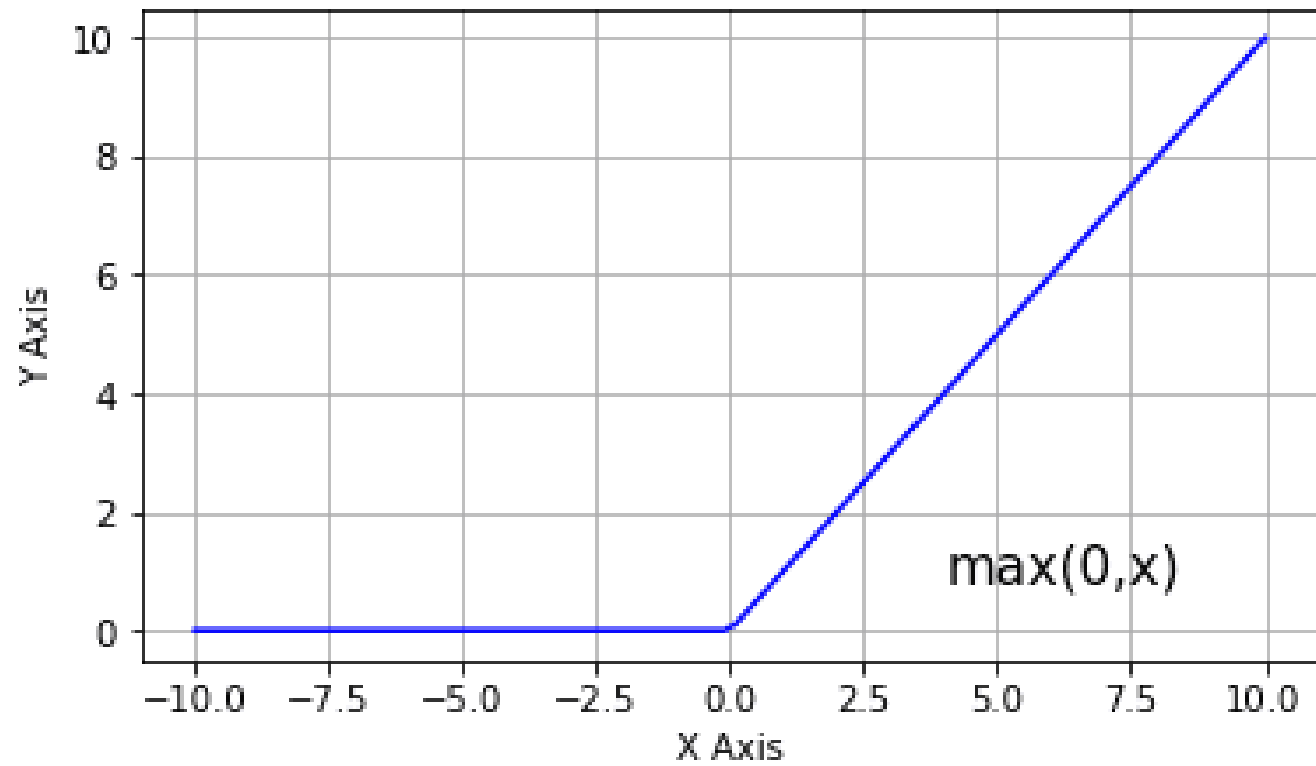
# ReLU(Rectified Linear Unit) :

- This is one of the most widely used activation function.
- The benefits of ReLU is the sparsity, it allows only values which are positive and negative values are not passed which will speed up the process and it will negate or bring down possibility of occurrence of a dead neuron.

$$f(x) = (0, \max)$$

- This function will allow only the maximum values to pass during the front propagation .
- The draw backs of ReLU is when the gradient hits zero for the negative values, it does not converge towards the minima which will result in a dead neuron while back propagation.

ReLU Activation Function

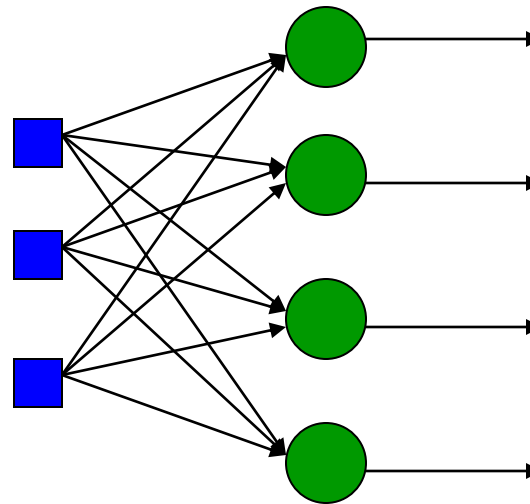


# Network Architectures

- Three different classes of network architectures
  - single-layer feed-forward
  - multi-layer feed-forward
  - recurrent
- The architecture of a neural network is linked with the learning algorithm used to train

# Single Layer Feed-forward

*Input layer  
of  
source nodes*



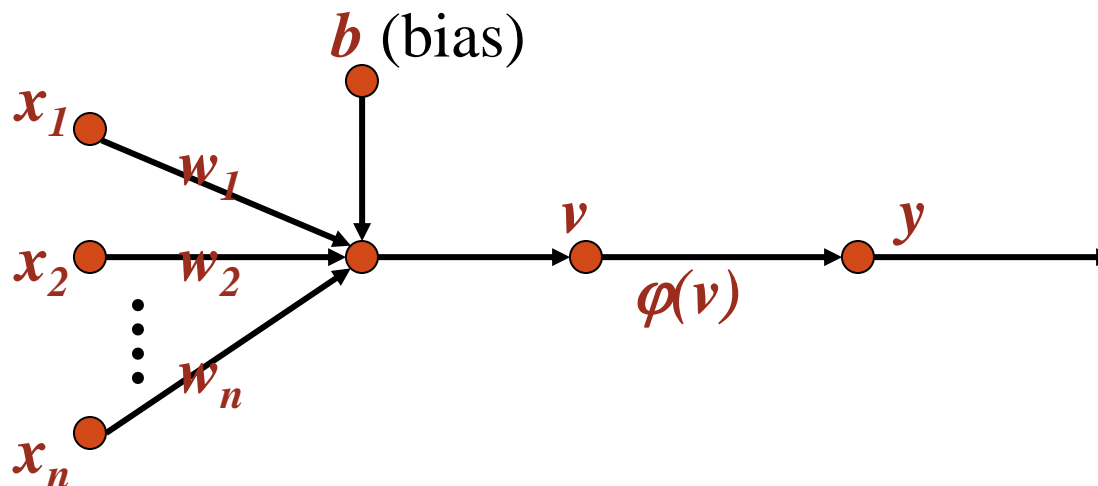
*Output layer  
of  
neurons*

# Perceptron: Neuron Model

## (Special form of single layer feed forward)

- The perceptron was first proposed by Rosenblatt (1958) is a simple neuron that is used to classify its input into one of two categories.
- A perceptron uses a **step function** that returns +1 if weighted sum of its input  $\geq 0$  and -1 otherwise

$$\varphi(v) = \begin{cases} +1 & \text{if } v \geq 0 \\ -1 & \text{if } v < 0 \end{cases}$$



# Perceptron for Classification

- The perceptron is used for binary classification.
- First train a perceptron for a classification task.
  - Find suitable weights in such a way that the training examples are correctly classified.
  - Geometrically try to find a hyper-plane that separates the examples of the two classes.
- The perceptron can only model linearly separable classes.
- When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.
- Given training examples of classes  $C_1$ ,  $C_2$  train the perceptron in such a way that :
  - If the output of the perceptron is 1 then the input is assigned to class  $C_1$
  - If the output is 0 then the input is assigned to  $C_2$

# Learning Process for Perceptron

- Initially assign random weights to inputs between -0.5 and +0.5
- Training data is presented to perceptron and its output is observed.
- If output is incorrect, the weights are adjusted accordingly using following formula.

$$w_i \leftarrow w_i + (a * x_i * e), \text{ where 'e' is error produced and 'a' } (-1 < a < 1) \text{ is learning rate}$$

- ‘a’ is defined as 0 if output is correct, it is +ve, if output is too low and –ve, if output is too high.
- Once the modification to weights has taken place, the next piece of training data is used in the same way.
- Once all the training data have been applied, the process starts again until all the weights are correct and all errors are zero.
- Each iteration of this process is known as an epoch.



# Example: Perceptron to learn OR function

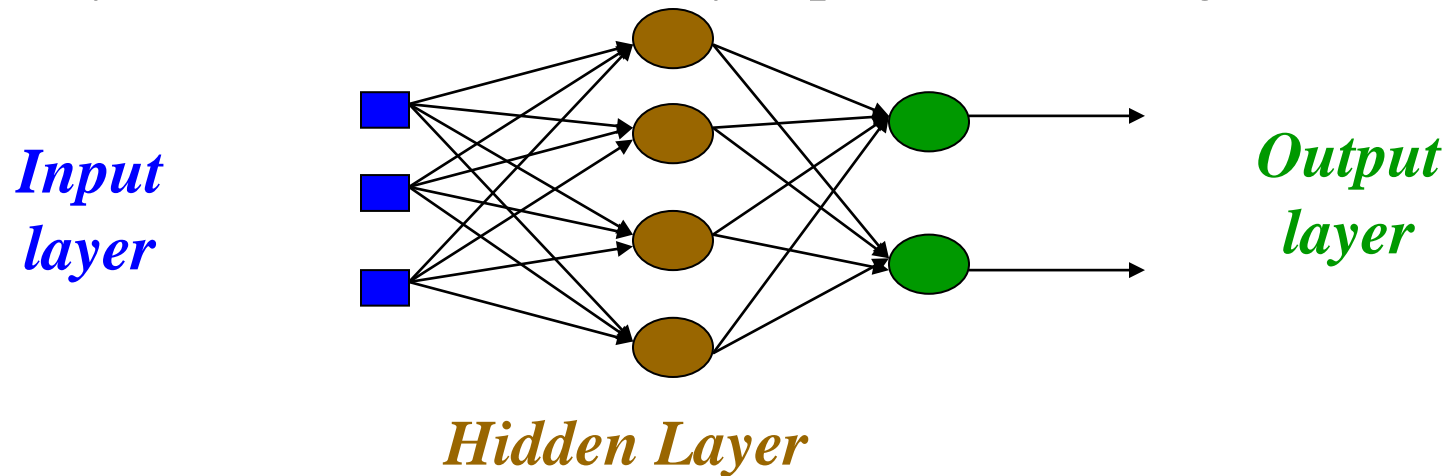
- Initially consider  $w_1 = -0.2$  and  $w_2 = 0.4$
- Training data say,  $x_1 = 0$  and  $x_2 = 0$ , output is 0.
- Compute  $y = \text{Step}(w_1 * x_1 + w_2 * x_2) = 0$ . Output is correct so weights are not changed.
- For training data  $x_1=0$  and  $x_2 = 1$ , output is 1
- Compute  $y = \text{Step}(w_1 * x_1 + w_2 * x_2) = 0.4 = 1$ . Output is correct so weights are not changed.
- Next training data  $x_1=1$  and  $x_2 = 0$  and output is 1
- Compute  $y = \text{Step}(w_1 * x_1 + w_2 * x_2) = -0.2 = 0$ . Output is incorrect, hence weights are to be changed.
- Assume  $a = 0.2$  and error  $e=1$   
 **$w_i = w_i + (a * x_i * e)$  gives  $w_1 = 0$  and  $w_2 = 0.4$**
- With these weights, test the remaining test data.
- Repeat the process till we get stable result.

# Perceptron: Limitations

- The perceptron can only model linearly separable functions,
  - those functions which can be drawn in 2-dim graph and single straight line separates values in two part.
- Boolean functions given below are linearly separable:
  - AND
  - OR
  - COMPLEMENT
- It cannot model XOR function as it is non linearly separable.
  - When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.

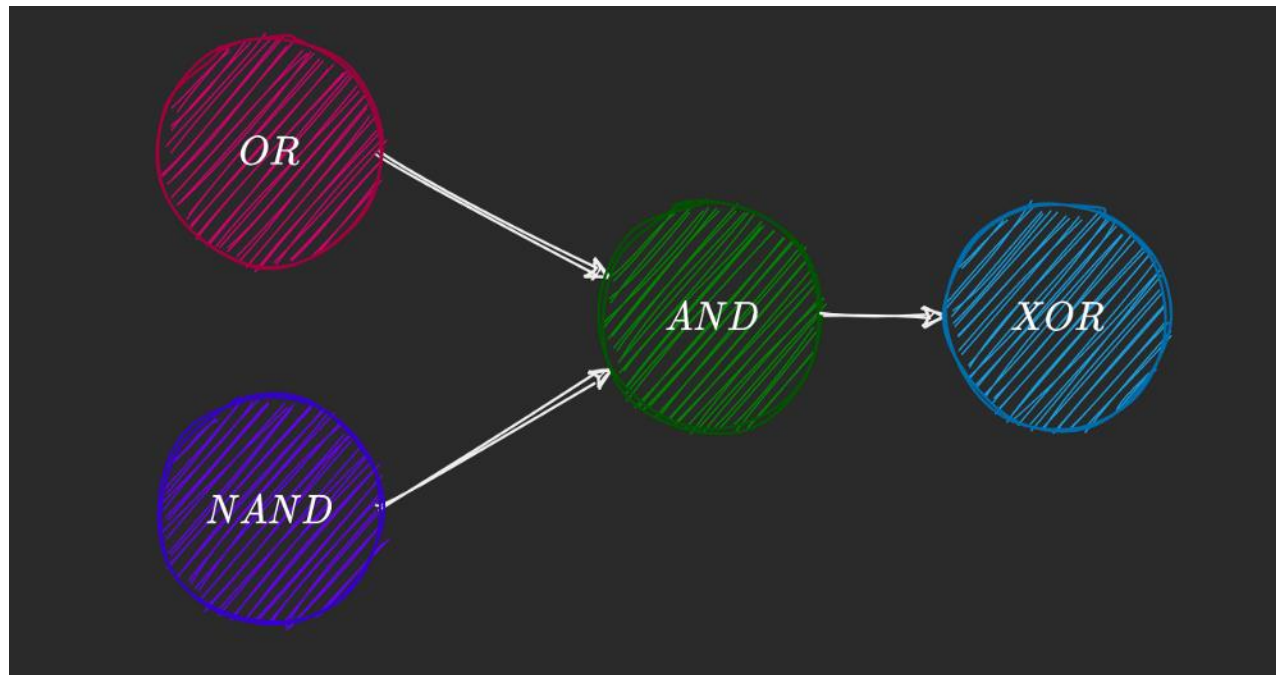
# Multi layer feed-forward NN (FFNN)

- FFNN is a more general network architecture, where there are hidden layers between input and output layers.
- Hidden nodes do not directly receive inputs nor send outputs to the external environment.
- FFNNs overcome the limitation of single-layer NN.
- They can handle non-linearly separable learning tasks.



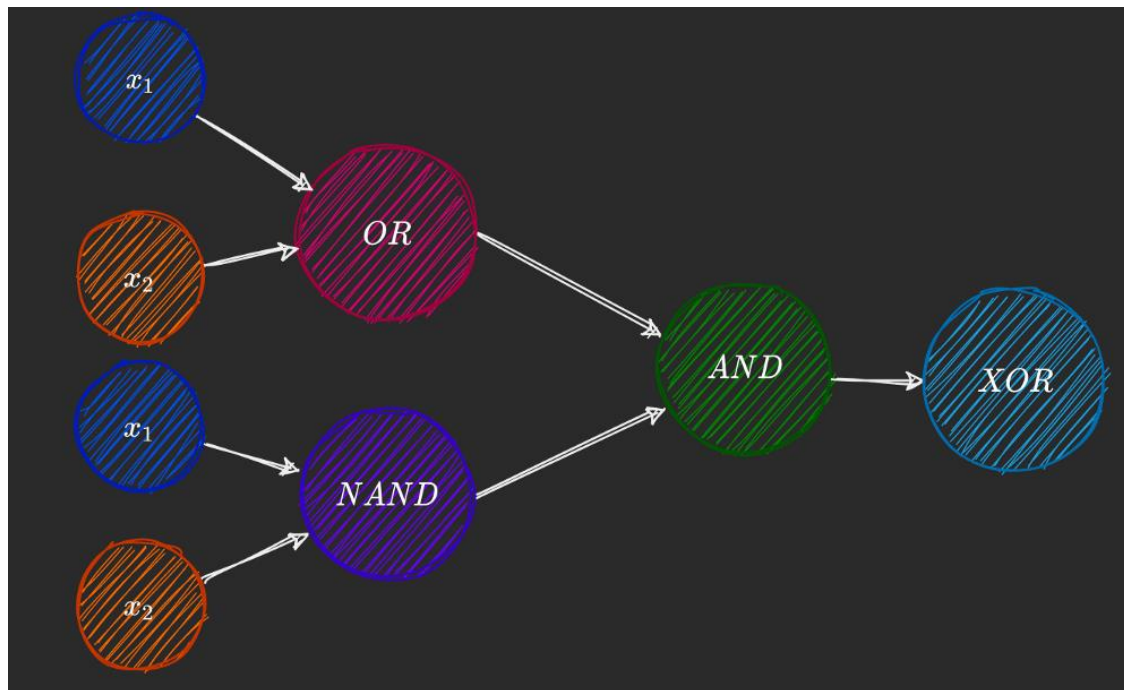
3-4-2 Network

# FFNN for XOR



# FFNN for XOR

- The ANN for XOR has two hidden nodes that realizes this non-linear separation and uses the sign (step) activation function.
- Arrows from input nodes to two hidden nodes indicate the directions of the weight vectors.
- The output node is used to combine the outputs of the two hidden nodes.



Inputs		Output of Hidden Nodes		Output Node AND ( $H_1, H_2$ )	$X_1$ XOR $X_2$
$X_1$	$X_2$	$H_1$ (OR)	$H_2$ (NAND)		
0	0	0	1	0	0
0	1	1	1	1	1
1	0	1	1	1	1
1	1	1	0	0	0

Since we are representing two input states by 0 (false) and 1 (true), we will take two hidden states  $H_1$  and  $H_2$  for performing OR ,NAND operation and the result of that will act as input for output node to perform AND operation which results ultimately in XOR operation .

# FFNN NEURON MODEL

- The classical learning algorithm of FFNN is based on the gradient descent method.
- For this reason the activation function used in FFNN are continuous functions of the weights, differentiable everywhere.
- The activation function for node  $i$  may be defined as a simple form of the **sigmoid function** in the following manner:

$$\varphi(V_i) = \frac{1}{1 + e^{(-V_i)}}$$

where  $A > 0$ ,  $V_i = \sum W_{ij} * Y_j$ , such that  $W_{ij}$  is a weight of the link from node  $i$  to node  $j$  and  $Y_j$  is the output of node  $j$ .

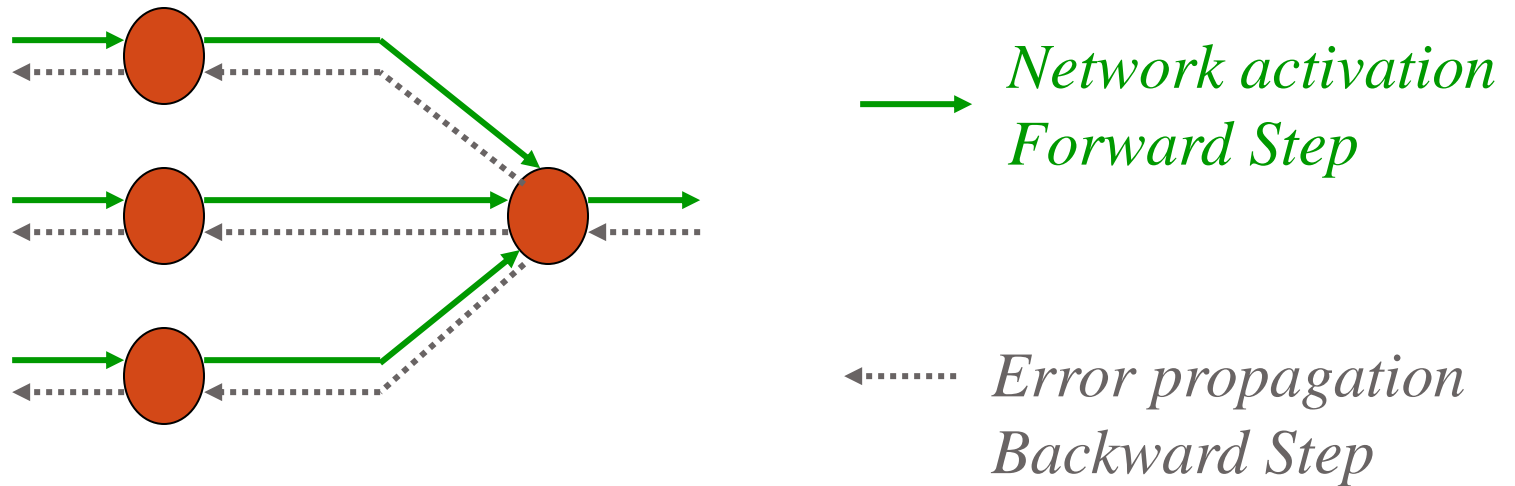
# Training Algorithm: Backpropagation

- The Backpropagation algorithm learns in the same way as single perceptron.
- It searches for weight values that minimize the total error of the network over the set of training examples (training set).
- Backpropagation consists of the repeated application of the following two passes:
  - **Forward pass:** In this step, the network is activated on one example and the error of (each neuron of) the output layer is computed.
  - **Backward pass:** in this step the network error is used for updating the weights. The error is propagated backwards from the output layer through the network layer by layer. This is done by recursively computing the local gradient of each neuron.



# Backpropagation

- Back-propagation training algorithm



- Backpropagation adjusts the weights of the NN in order to minimize the network total mean squared error.

## Contd..

- Consider a network of three layers.
- Let us use  $i$  to represent nodes in input layer,  $j$  to represent nodes in hidden layer and  $k$  represent nodes in output layer.
- $w_{ij}$  refers to weight of connection between a node in input layer and node in hidden layer.
- The following equation is used to derive the output value  $Y_j$  of node  $j$

$$Y_j = \frac{1}{1 + e^{-X_j}}$$

where,  $X_j = \sum x_i \cdot w_{ij} - \theta_j$ ,  $1 \leq i \leq n$ ;  $n$  is the number of inputs to node  $j$ , and  $\theta_j$  is threshold for node  $j$

# Total Mean Squared Error

- **The error function** For simplicity, we'll use the Mean Squared Error function. For the first output, the error is the correct output value minus the actual output of the neural network:

$$0.5 - 0.735 = -0.235$$

- For the second output:

$$0.5 - 0.455 = 0.045$$

Now we'll calculate the Mean Squared Error:

$$\text{MSE}(\text{O1}) = \frac{1}{2} (-0.235)^2 = 0.027$$

$$\text{MSE}(\text{O2}) = \frac{1}{2} (0.045)^2 = 0.001$$

The Total Error is the sum of the two errors:

$$\text{Total Error} = 0.0276 + 0.001 = \mathbf{0.0286}$$

This is the number we need to minimize with backpropagation.

# Weight Update Rule

- The Backprop weight update rule is based on the gradient descent method:
  - It takes a step in the direction yielding the maximum decrease of the network error E.
  - This direction is the opposite of the gradient of E.
- Iteration of the Backpropagation algorithm is usually terminated when the sum of squares of errors of the output values for all training data in an epoch is less than some threshold such as 0.01

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

# Backprop learning algorithm (incremental-mode)

n=1;

initialize **weights** randomly;

**while** (stopping criterion not satisfied or n < max\_iterations)

**for** each example (**x**,*d*)

    - run the network with input *x* and compute the output *y*

    - update the weights in backward order starting from those of the output layer:

$$w_{ji} = w_{ji} + \Delta w_{ji}$$

    with  $\Delta w_{ji}$  computed using the (generalized) Delta rule

**end-for**

  n = n+1;

**end-while;**

# Stopping criteria

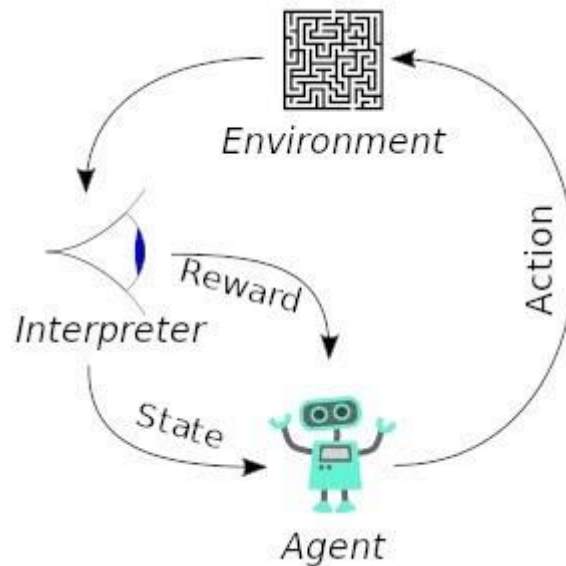
- Total mean squared error change:
  - Back-prop is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small (in the range  $[0.1, 0.01]$ ).
- Generalization based criterion:
  - After each epoch, the NN is tested for generalization.
  - If the generalization performance is adequate then stop.
  - If this stopping criterion is used then the part of the training set used for testing the network generalization will not be used for updating the weights.

# Reinforcement Learning

- Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation.
- It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation.
- Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task.
- In the absence of a training dataset, it is bound to learn from its experience.

# Learning from rewards

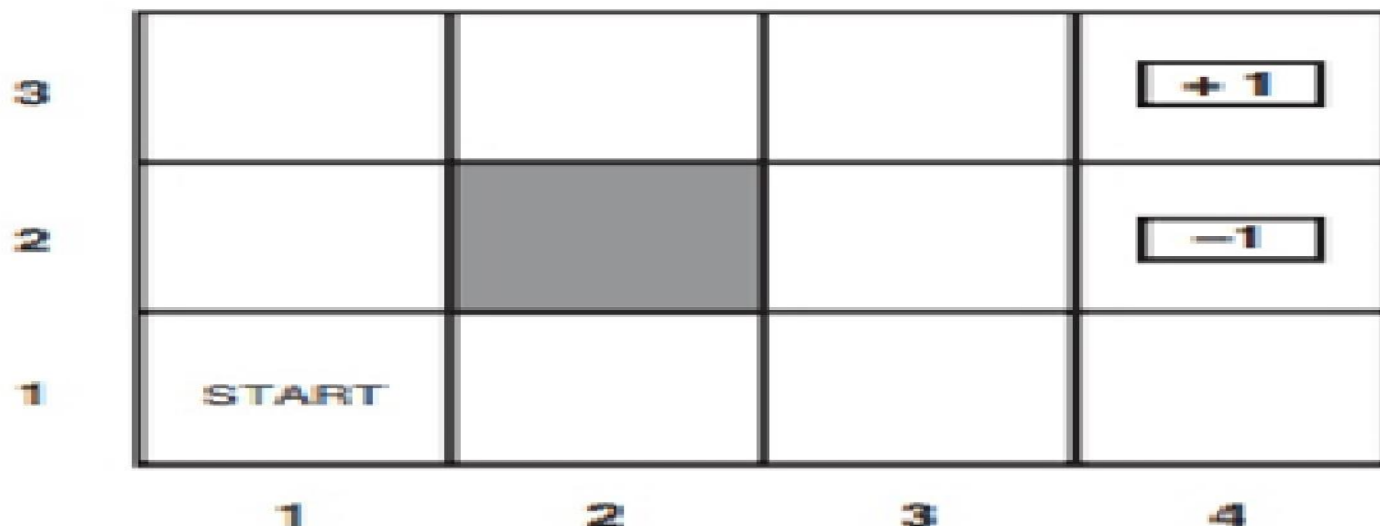
- In **Reinforcement Learning (RL)**, agents are trained on a **reward** and **punishment** mechanism. The **agent** is rewarded for correct moves and punished for the wrong ones. In doing so, the agent tries to minimize wrong moves and maximize the right ones.





# Learning from rewards

**Example:** The problem is as follows: We have an agent and a reward, with an hurdle (2,2) in between. The agent is supposed to find the best possible path to reach the reward.



The agent learns by trying all the possible paths and then choosing the path which gives him the reward with the least hurdles. Each right step will give the agent a reward and each wrong step will subtract the reward of the agent. The total reward will be calculated when it reaches the final reward that is at the state where an agent gets +1 reward

# Steps in Reinforcement Learning

- Input: The input should be an initial state from which the model will start
- Output: There are many possible output as there are variety of solution to a particular problem
- Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
- The model keeps continues to learn.
- The best solution is decided based on the maximum reward.

- **Policy:** It is a mapping of an action to every possible state in the system (sequence of states).
- **Optimal Policy:** A policy which maximizes the long term reward.

# Active and Passive Reinforcement Learning

- Both active and passive reinforcement learning are types of Reinforcement Learning.
- In case of passive reinforcement learning, the agent's policy is fixed which means that it is told what to do.
- In contrast to this, in active reinforcement learning, an agent needs to decide what to do as there's no fixed policy that it can act on.
- Therefore, the goal of a passive reinforcement learning agent is to execute a fixed policy (sequence of actions) and evaluate it while that of an active reinforcement learning agent is to act and learn an optimal policy.

# Passive Reinforcement Learning Techniques

- In this kind of RL, agent assume that the agent's policy  $\pi(s)$  is fixed.
- Agent is therefore bound to do what the policy dictates, although the outcomes of agent actions are probabilistic.
- The agent may watch what is happening, so the agent knows what states the agent is reaching and what rewards the agent gets there.

## Techniques:

1. Direct utility estimation
2. Adaptive dynamic programming
3. Temporal difference learning

# Active Reinforcement Learning Techniques

- In this kind of RL agent , it assume that the agent's policy  $\pi(s)$  is not fixed.
- Agent is therefore not bound on existing policy and tries to act and find an Optimal policy for calculating and maximizing the overall reward value.

## Techniques:

1. Q-Learning
2. ADP with exploration function

# Applications of Reinforcement Learning

- Robotics for industrial automation.
- Business strategy planning
- Machine learning and data processing
- It helps you to create training systems that provide custom instruction and materials according to the requirement of students.
- Aircraft control and robot motion control

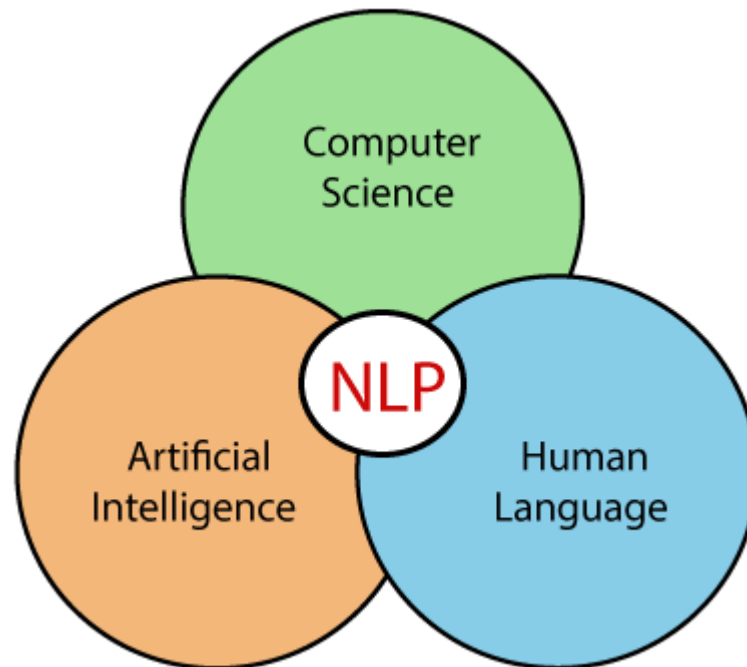
# UNIT-5



# **Natural Language Processing (NLP)**

# What is NLP?

- *Natural Language Processing or NLP is a field of Artificial Intelligence that gives the machines the ability to read, understand and derive meaning from human languages.*



# Why NLP?

- Everything we express (either verbally or in written) carries huge amounts of information.
- The topic we choose, our tone, our selection of words, everything adds some type of information that can be interpreted and value extracted from it.
- In theory, we can understand and even predict human behavior using that information.
- But there is a problem: one person may generate hundreds or thousands of words in a declaration, each sentence with its corresponding complexity.
- If you want to scale and analyze several hundreds, thousands or millions of people or declarations in a given geography, then the situation is unmanageable.

# Why NLP?

- Data generated from conversations, declarations or even tweets are examples of unstructured data.
- **Unstructured data** doesn't fit neatly into the traditional row and column structure of relational databases, and represent the vast majority of data available in the actual world.
- It is messy and hard to manipulate. Nevertheless, thanks to the advances in disciplines like machine learning a big revolution is going on regarding this topic.

# Applications of NLP

- Chat box
- Speech recognition
- Machine Translation
- Spell Checking
- Keyword Searching
- Information extraction
- Advertisement Matching

# Components of NLP

- **Natural Language Understanding (NLU)**

Taking some spoken / typed sentence and working out what it means

- **Natural Language Generation (NLG)**

Taking some formal representation of what you want to say and working out a way to express it in a natural (human) language (e.g., English)

# Components of NLP

- **Natural Language Understanding (NLU)**
  - Mapping the given input in the natural language into a useful representation
  - Different levels of analysis required:
    - Morphological analysis
    - Syntactic Analysis
    - Semantic Analysis
    - Discourse Analysis

# Components of NLP

- **Natural Language Generation (NLG)**
  - Producing some output in the natural language from some internal representation
  - Different levels of synthesis required:
    - Deep planning (what to say)
    - Syntactic Generation
- **NL Understanding is much harder than NL Generation. But still both of them are hard.**



# Building an NLP pipeline

There are the following steps to build an NLP pipeline

- Tokenization
- Stemming
- Lemmatization
- POS Tags
- Name Entity Recognition
- Chunking

# Tokenization

- Cutting the big sentences into small tokens
- Example: Today is Wednesday

“Today”

“is”

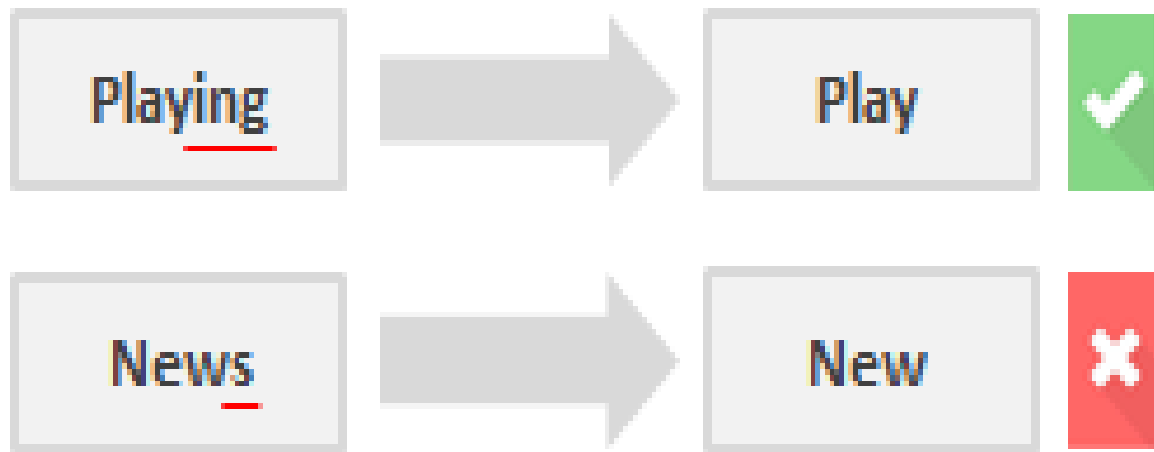
“Wednesday”

# Stemming

- Refers to the process of slicing the end or the beginning of words with the intention of removing affixes (lexical additions to the root of the word).
- Normalize words into its base or root forms.
- For example, celebrates, celebrated and celebrating, all these words are originated with a single root word "celebrate."

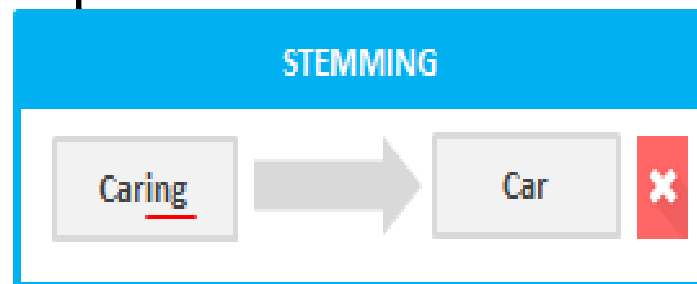
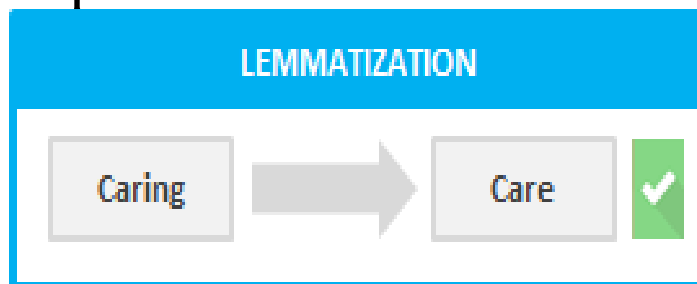
# Problem in Stemming

- The big problem with stemming is that sometimes it produces the root word which may not have any meaning.



# Lemmatization

- Lemmatization is quite similar to the Stemming but it overcomes the limitation of Stemming.
- It is used to group different inflected forms of the word, called Lemma.
- The main difference between Stemming and lemmatization is that it produces the root word, which has a meaning.
- Output of Lemmatization is a proper word.



# POS Tags

- POS stands for Parts of Speech Tags.
- It includes Noun, verb, adverb, and Adjective.
- It indicates that how a word functions with its meaning as well as grammatically within the sentences.
- A word has one or more parts of speech based on the context in which it is used.
- **Example: "Google"** something on the Internet.
- In the above example, Google is used as a verb, although it is a proper noun.

# Name Entity Recognition

- Named Entity Recognition (NER) is the process of detecting the named entity such as person name, organization name, location or Monetary value.
- Example: HelpingHands founder Arhaan lists his Bangalore penthouse for 20 Million rupees.

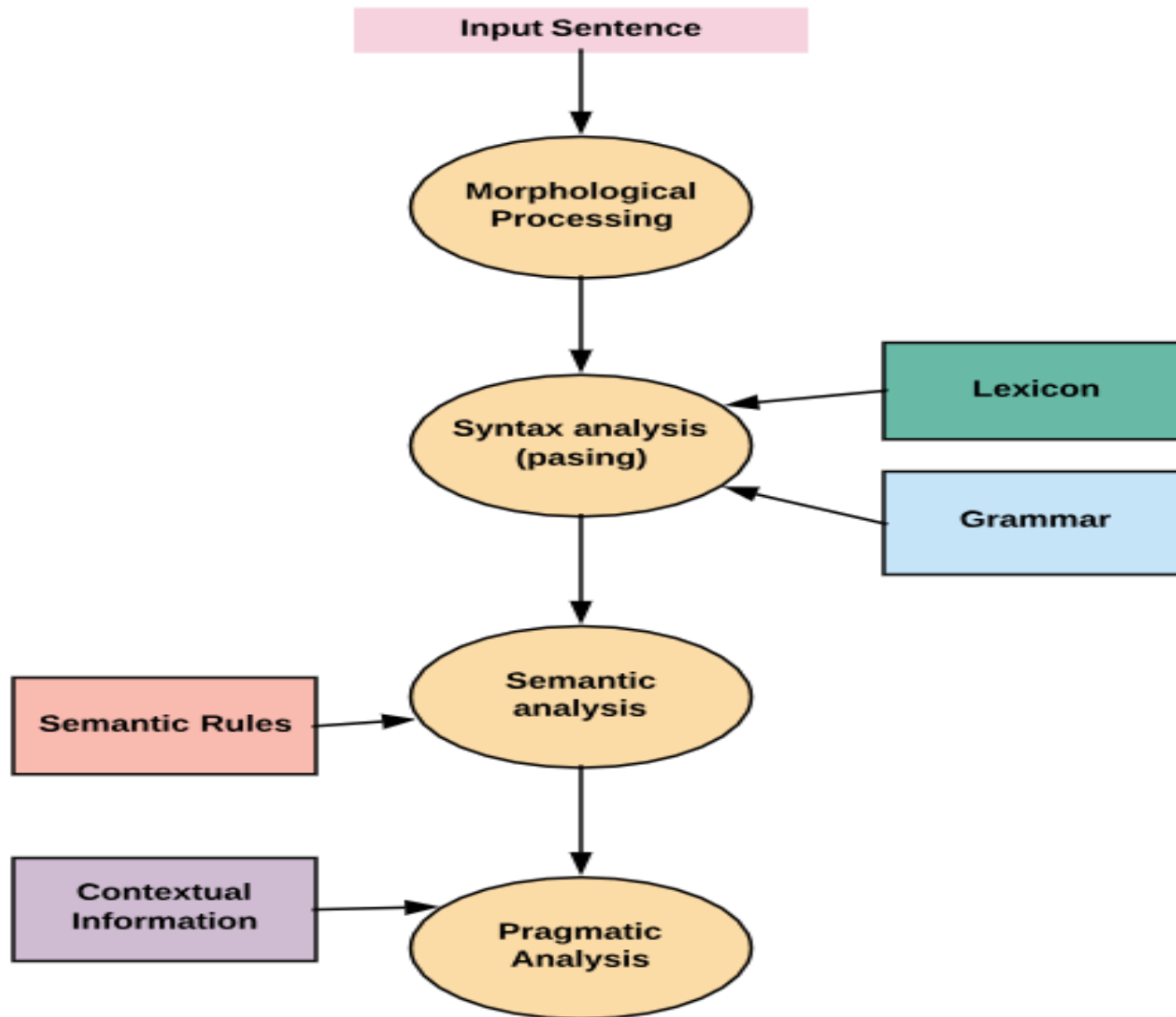
# Chunking

- Chunking is used to collect the individual piece of information and grouping them into bigger pieces of sentences.
- This help getting insight and meaningful information from the text.



# Phases of NLP

- Morphological and Lexical Analysis
- Syntactic Analysis
- Semantic Analysis
- Discourse Integration
- Pragmatic Analysis



# Morphological and Lexical Analysis

- The lexicon of a language is its vocabulary that includes its words and expressions
- Morphology depicts analyzing, identifying and description of structure of words
- Lexical analysis involves dividing a text into paragraphs and the sentences

# Syntactic Analysis

- Syntax concerns the proper ordering of words and its effects on meaning
- This involves analysis of the words in a sentence to depict the grammatical structure of the sentence.
- The words are transformed into structure that shows how the words are related to each other.
- Eg: School went to Raju

This would be rejected by English syntactic analyzer

# Semantic Analysis

- Semantic concerns the (literal) meaning of words, phrases and sentences.
- This abstracts the dictionary meaning or the exact meaning from context.
- The structures which are created by the syntactic analyzer are assigned meaning.
- Eg: “Hot Ice cream”  
It is rejected as it does not give any sense

# Discourse Integration

- Sense of the context
- The meaning of any single sentence depends upon the sentences that precedes it and also invokes the meaning of the sentences that follow it.
- Eg: The word “there” in the sentence “He wants to go there” depends upon the prior discourse context.

# Pragmatic Analysis

- Pragmatic concerns the overall communicative and social context and its effects on interpretation.
- It means abstracting or deriving the purposeful use of the language in situations.
- Importantly those aspects of language which require world knowledge.
- The main focus is on what was said is reinterpreted on what it actually means
- Eg: “close the window” should have been interpreted as a request rather than an order.

# Natural Language Generation (NLG)

- Natural Language Generation is the process of constructing natural language outputs from non-linguistic inputs.
- NLG can be viewed as the reverse process of NL understanding.
- A NLG system contains:
  - **Discourse Planner**  
What will be generated, which sentences
  - **Surface Realizer**  
realizes a sentence from its internal representation
  - **Lexical selection**  
selecting the correct words describing the concepts



# Why NLP is difficult?

- NLP is difficult because Ambiguity and Uncertainty exist in the language.

## Ambiguity

There are the following three ambiguity -

### Lexical Ambiguity

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

**Example:** Jack is looking for a **match**.

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

# Why NLP is difficult?

## Syntactic Ambiguity

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

**Example:** I saw the boy with the binocular.

- In the above example, did I have the binoculars? Or did the boy have the binoculars?

## Referential Ambiguity

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

**Example:** John met Jack. He said, "I am hungry."

- In the above sentence, you do not know that who is hungry, either John or Jack.

# Grammars and Parsing

- It is the part for implementation aspects of syntactic Analysis
- There are a number of algorithms researchers have developed for syntactic analysis, but we consider only the following simple methods –

Context-Free Grammar

Top-Down Parser

Bottom-Up Parser

# Grammars and Parsing

Syntactic categories (common denotations) in NLP

np - noun phrase

vp - verb phrase

s - sentence

det - determiner (article)

n - noun

tv - transitive verb (takes an object)

iv - intransitive verb

prep - preposition

pp - prepositional phrase

adj - adjective

# Context Free Grammar

- A *context-free grammar (CFG)* is a list of rules that define the set of all well-formed sentences in a language. Each rule has a left-hand side, which identifies a syntactic category, and a right-hand side, which defines its alternative component parts, reading from left to right.
- E.g., the rule  $s \rightarrow np\ vp$  means that "a sentence is defined as a noun phrase followed by a verb phrase."

# Parsing

- **Parsing** in **NLP** is the **process** of determining the syntactic structure of a text (parse tree) by analyzing its constituent words based on an underlying grammar (of the language).

Figure 1. A grammar and a parse tree for "the giraffe dreams".

s → np vp

np → det n

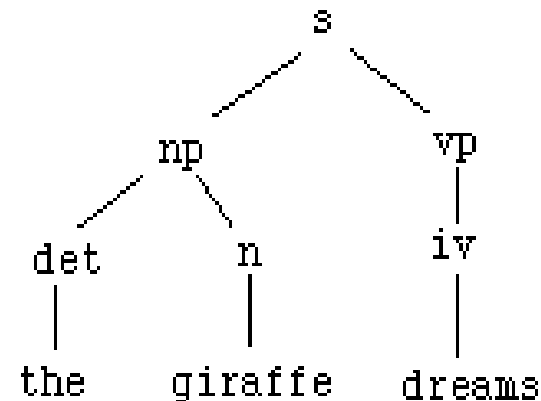
vp → tv np  
→ iv

det → the  
→ a  
→ an

n → giraffe  
→ apple

iv → dreams

tv → eats  
→ dreams



- A *sentence* in the language defined by a CFG is a series of words that can be derived by systematically applying the rules, beginning with a rule that has *s* on its left-hand side.
- A *parse* of the sentence is a series of rule applications in which a syntactic category is replaced by the right-hand side of a rule that has that category on its left-hand side, and the final rule application yields the sentence itself.
- E.g., a parse of the sentence "the giraffe dreams" is:  

$$s \Rightarrow np\ vp \Rightarrow det\ n\ vp \Rightarrow the\ n\ vp \Rightarrow the\ giraffe\ vp \Rightarrow the\ giraffe\ iv \Rightarrow the\ giraffe\ dreams$$
- A convenient way to describe a parse is to show its *parse tree*, which is simply a graphical display of the parse. Figure 1 shows a parse tree for the sentence "the giraffe dreams".
- Note that the root of every sub tree has a grammatical category that appears on the left-hand side of a rule, and the children of that root are identical to the elements on the right-hand side of that rule.

# Top-Down Parser

- Here, the parser starts with the S symbol and attempts to rewrite it into a sequence of *terminal symbols* that matches the classes of the words in the input sentence until it consists entirely of terminal symbols.
- These are then checked with the input sentence to see if it matched. If not, the process is started over again with a different set of rules. This is repeated until a specific rule is found which describes the structure of the sentence.



# Bottom-Up Parser

- Looks at words in input string first, checks / assigns their category(ies), and tries to combine them into acceptable structures in the grammar.
- Involves scanning the derivation so far for sub-strings which match the right-hand-side of grammar / production rules and using the rule that would show their derivation from the nonterminal symbol of that rule
- Finally it ends to S symbol.