

ARTIFICIAL INTELLIGENCE

4 YEAR SEM-1 BTECH MECHANICAL ENGINEERING (R18A1205)



UNIT-1

UNIT - I Introduction: AI problems, Agents and Environments, Structure of Agents, Problem Solving Agents

Basic Search Strategies: Problem Spaces, Uninformed Search (Breadth-First, Depth-First Search, Depth-first with Iterative Deepening), Heuristic Search (Hill Climbing, Generic Best-First, A*), Constraint Satisfaction (Backtracking, Local Search)

Introduction:

- AI is the universal field of computer science.
 - It is one of the fascinating technology.
 - Greatest scope in future
 - It has a tendency to cause a machine to work as a human.
-
- Artificial-----” man made”
 - Intelligence----- the ability of making artificial thing----”thinking power”

AI definition:

- It is a branch of cs by which we can create an intelligent machine, which can behave like a human and think like a human and able to make decisions.
- With AI, you do not need to pre-program a machine to do some work.
- In spite you can program a machine, which can work with own intelligence
- There are two ideas in the definition.

✓ **Intelligence** [Ability to understand, think & learn]

✓ **Artificial device** [Non Natural]

What is AI

- Artificial Intelligence (AI) is a branch of Science which deals with helping machines find solutions to complex problems in a more human-like fashion.
- This generally involves borrowing characteristics from human intelligence, and applying them as algorithms in a computer friendly way.
- A more or less flexible or efficient approach can be taken depending on the requirements established, which influences how artificial the intelligent behavior appears

Structure Programming Vs AI

Structured Programming	Artificial Intelligence
<p>A program without AI can answer the "specific" questions it is meant to answer</p>	<p>AI answers any question belonging to its "generic" type.</p>
<p>If you modify a structural program, its entire structure changes.</p>	<p>AI programs are all about modifications. They keep absorbing the information provided to them as stimuli for future referencing like the human brain.</p>

Artificial intelligence can be viewed from a variety of perspectives.

- From the perspective of **intelligence** artificial intelligence is making machines "intelligent" -- acting as we would expect people to act.
- The inability to distinguish computer responses from human responses is called the Turing test.
- Intelligence requires knowledge
- Expert problem solving - restricting domain to allow including significant relevant knowledge

- From a **business** perspective AI is a set of very powerful tools, and methodologies for using those tools to solve business problems.
- From a **programming** perspective, AI includes the study of symbolic programming, problem solving, and search.
- Typically AI programs focus on symbols rather than numeric processing.
- Problem solving - achieve goals.
- Search - seldom access a solution directly. Search may include a variety of techniques.

Why AI:

With the help of AI,

- We can create such amazing software's ---- a device which can solve real-world problems accurately and easily.
- We can create our personal virtual assistances.
- We can build such robots which can work in a environment where survival of human can be at a risk .
- AI opens path for new technologies, new devices and new opportunities.

Goals:

- Replicate human intelligence
- Solve-knowledge intensive tasks
- An intelligent connection of perception and action.
- Building a machine which can perform tasks that requires human intelligence
 - Providing theorem/algorithm
 - Plan some surgical operation
 - Playing chess
 - Driving car in traffic
- Creating some system which can exhibit intelligent behaviours.

History of AI

- 1943: Early Beginnings
Boolean Circuit model of Brains
- 1950: Turing
Turing's computing Machinery and Intelligence
- 1956: Birth of AI
Dartmouth Conference: Artificial Intelligence name Adopted
- 1955-1965: Great Enthusiasm
GPS Solver [General Problem Solver]

History of AI

- **1966:** Reality Dawns
Realization that many AI problems are intraceable
- **1969-1985:** Adding Domain Knowledge
Development of **knowledge based systems**
Success of **rule based expert systems**
- **1986:** Rise of Machine Learning
Neural Networks return to popularity
- **1990:** Role of **Uncertainty**
Bayesian networks as a **knowledge representation framework**
- **1995:** AI as Science[Integration of learning, reasoning, knowledge representation, AI methods used in vision, language and data mining.

Applications of AI

- **Gaming** – AI plays important role for machine to think of large number of possible positions based on deep knowledge in strategic games.
- **Natural Language Processing** – Interact with the computer that understands natural language spoken by humans
- **Expert Systems** – Machine or software provide explanation and advice to the users.

Applications of AI

- **Vision Systems** – Systems understand, explain, and describe visual input on the computer
- **Speech Recognition** – There are some AI based speech recognition systems have ability to hear and express as sentences and understand their meanings while a person talks to it.
- **Handwriting Recognition** – The handwriting recognition software reads the text written on paper and recognize the shapes of the letters and convert it into editable text.
- **Intelligent Robots** – Robots are able to perform the instructions given by a human.

- **Thinking Humanly:** The Cognitive modeling Approach
- We can say that given program thinks like a human, we must have some way of determining how humans think. i.e., we need to get inside the actual working of human minds.
- There are 3 ways to do this;

- **Through Introspection**

- Trying to catch our own Thoughts as they go by*

- **Through psychological experiments**

- Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as computer programs.

- If the programs input/output and timing behaviour matches human behaviour that is the evidence.

- **Brain Imaging**

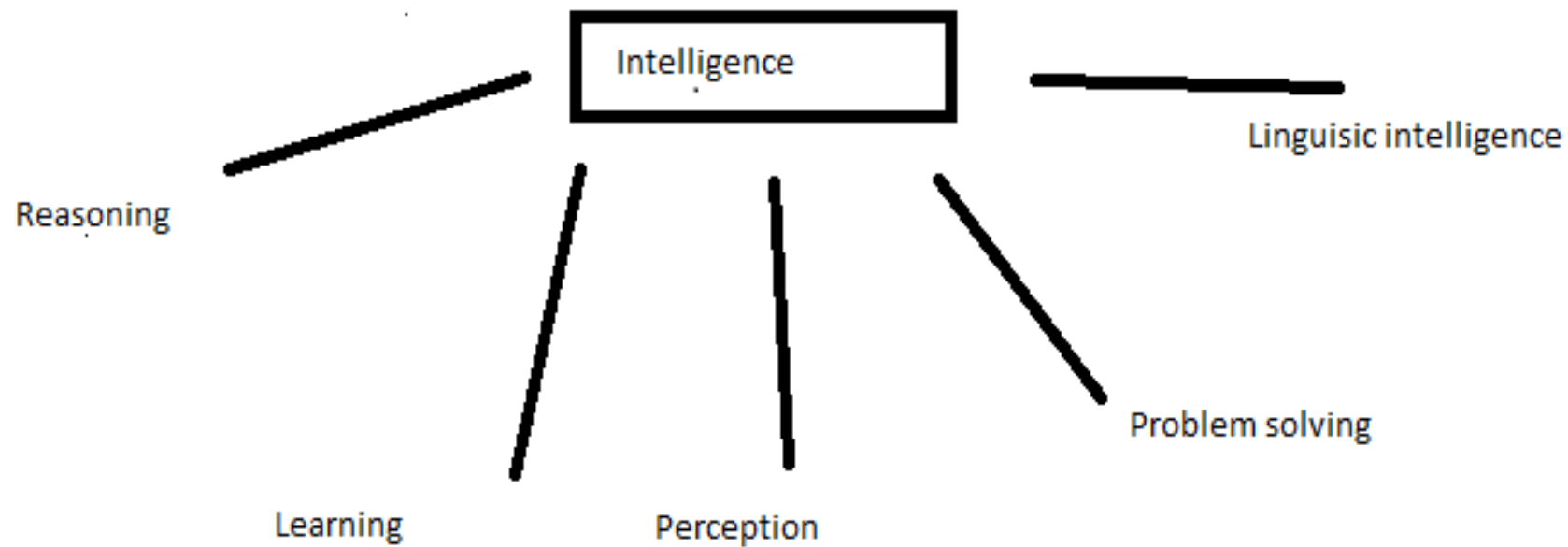
- Observing the brain in action

- **Acting Humanly:** The Turing test Approach
- The Turing Test, proposed by Alan Turing (1950), was designed to provide a satisfactory operational definition of intelligence.
- Here the computer is asking some questions by a human interrogator.
- The computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or not.

Acting Humanly

- The computer would need to possess the following capabilities:
 - **Natural language processing:** Enable it to communicate successfully in English.
 - **Knowledge representation:** Store what it knows or hears.
 - **Automated reasoning:** Use the stored information to answer questions and to draw new conclusions.
 - **Machine learning:** To adapt to new circumstances and to detect and extrapolate patterns.
 - **Computer vision:** To perceive objects.
 - **Robotics:** To manipulate objects and move about.

AI is composed of:



- **Reasoning:** Set of process that enable us to provide logical thinking
- **Learning:** It is an activity of gaining knowledge
- **Perception:** It is a process of acquiring, interpreting and selecting and even organizing the sensor information.
- **Problem solving:** It is the process of working through the details of a problem to reach the solution.
- **Linguistic intelligence:** It is one's ability to use comprehend speak and write the verbal and written languages.

Importance of AI

- Game Playing
- Speech Recognition
- Understanding Natural Language
- Computer Vision(3D)
- Expert Systems(medical)
- Heuristic Classification(fraud detection)

The applications of AI

Consumer Marketing

- Have you ever used any kind of credit/ATM/store card while shopping?
- if so, you have very likely been “input” to an AI algorithm
- All of this information is recorded digitally
- Companies like Nielsen gather this information weekly and search for patterns
 - – general changes in consumer behavior
 - – tracking responses to new products
 - – identifying customer segments: targeted marketing, e.g., they find out that consumers with sports cars who buy textbooks respond well to offers of new credit cards.
- Algorithms (“data mining”) search data for patterns based on mathematical theories of learning

Applications of AI

Identification Technologies

- ID cards e.g., ATM cards
- can be a nuisance and security risk: cards can be lost, stolen, passwords forgotten, etc
- Biometric Identification, walk up to a locked door
 - – Camera
 - – Fingerprint device
 - – Microphone
 - – Computer uses biometric signature for identification
 - – Face, eyes, fingerprints, voice pattern
 - – This works by comparing data from person at door with stored library
 - – Learning algorithms can learn the matching process by analyzing a large library database off-line, can improve its performance.

Applications of AI

Intrusion Detection

- Computer security
- - we each have specific patterns of computer use times of day, lengths of sessions, command used, sequence of commands, etc
- – would like to learn the “signature” of each authorized user
- – can identify non-authorized users o How can the program automatically identify users?
- – record user’s commands and time intervals
- – characterize the patterns for each user
- – model the variability in these patterns
- – classify (online) any new user by similarity to stored patterns

Applications of AI

Machine Translation

- Language problems in international business – e.g., at a meeting of Japanese, Korean, Vietnamese and Swedish investors, no common language
- – If you are shipping your software manuals to 127 countries, the solution is ; hire translators to translate
- – would be much cheaper if a machine could do this!
- How hard is automated translation
- – very difficult!
- – e.g., English to Russian
- – not only must the words be translated, but their meaning also!

Intelligent Agent:

- Agents in AI:

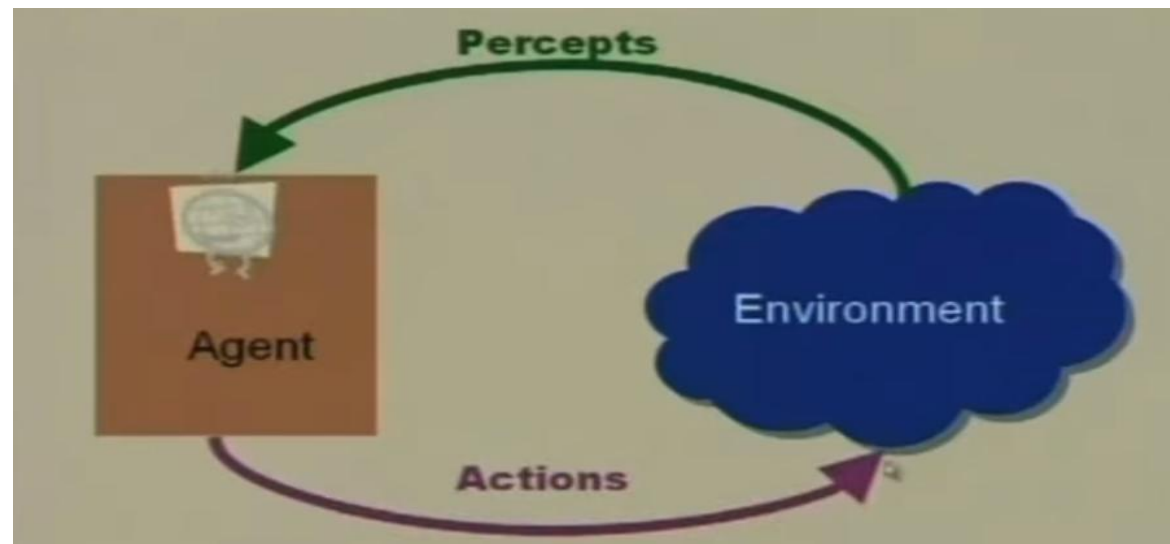
Study of relational agents and its environment. i.e., agents sense the environment through sensors and act on their environment through actuators.

Ex: automatic self-driving car

AI agent can have mental properties like:

Knowledge,

Belief and intention etc...



Intelligent Agent's:

Must Sense

Must Act

Must be Autonomous(to some extent)

Must be rational

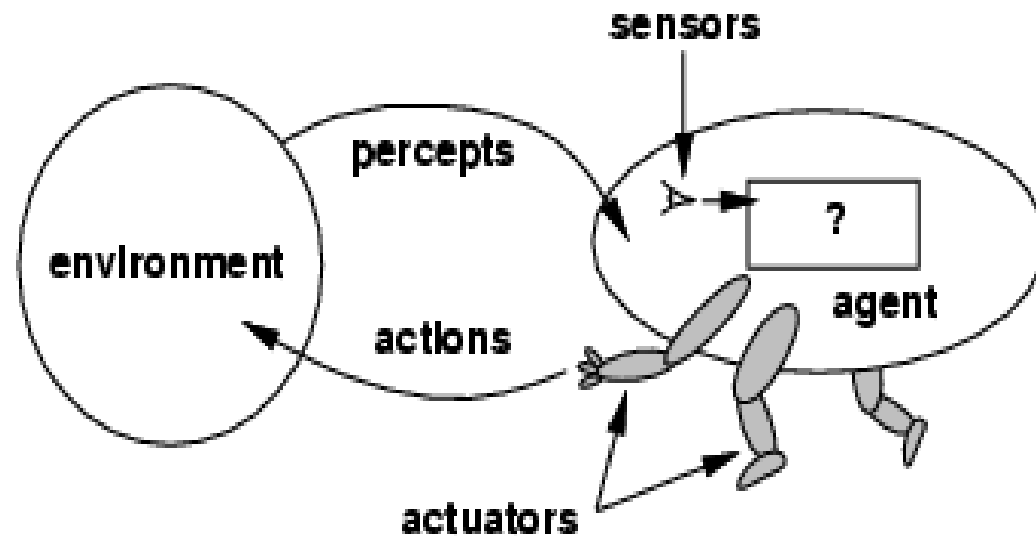


Fig 2.1: Agents and Environments

What is an agent?

- An agent can be anything, that perceive environment through sensors and act up on that environment through actuators.
- **Perceiving-----thinking-----acting**
- Generally agent can be of 3 types:

Human agent

Robotic agent

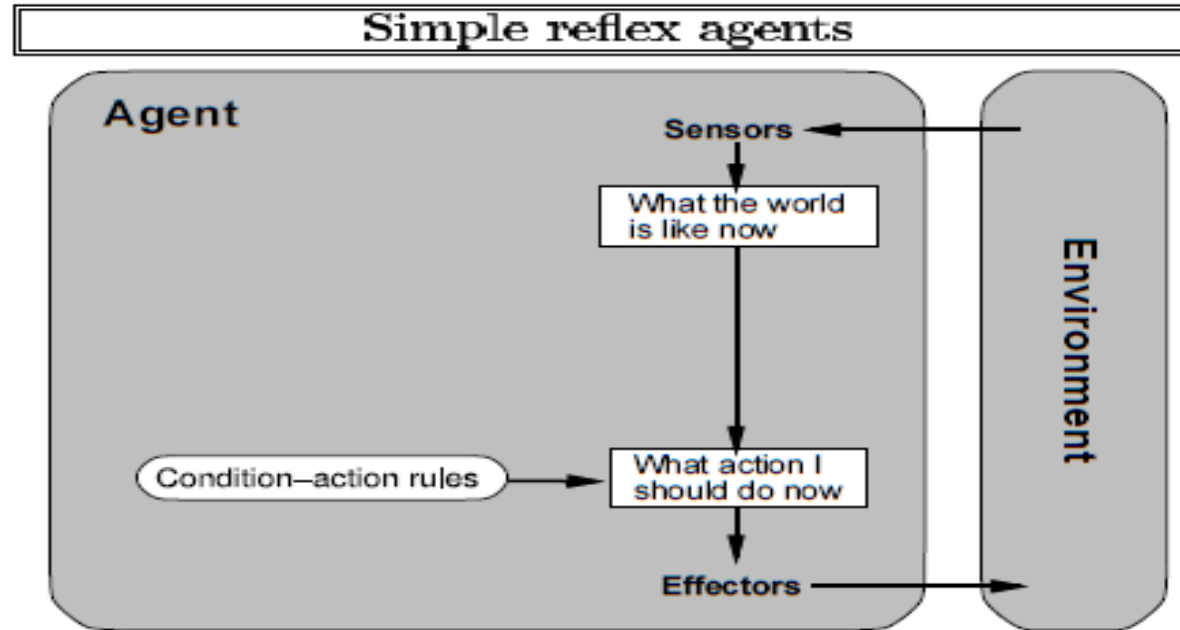
Software agent ---- keystrokes (python –F5)

- **Sensor:** It is a device which detects the change in environment and sends the information to the other electronic device.
- **Actuators:** It is a part or component of machine that converts energy into motion
- **Effectors:** The device which effects the environment.

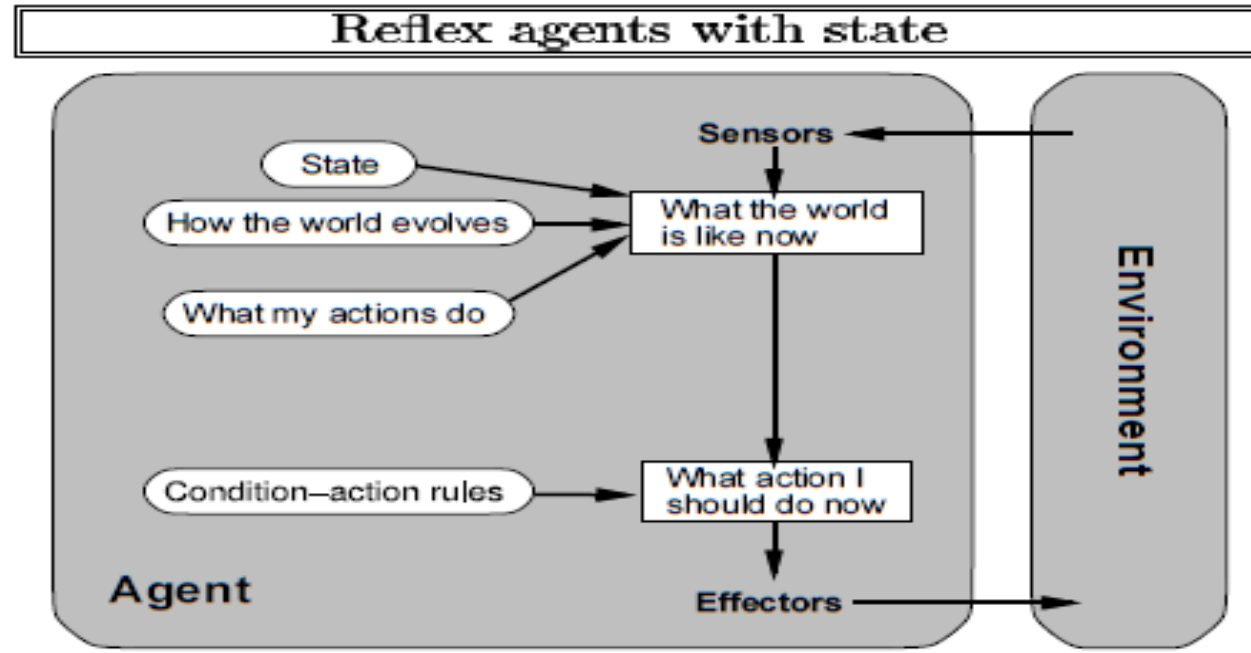
Different types of AI agents:

- Agent can be grouped into 5 classes based on their degree of perceived intelligence and capacity.
 1. Simple reflex agent
 2. Model based reflex agent
 3. Goal based agent
 4. Utility based agent
 5. Learning agent

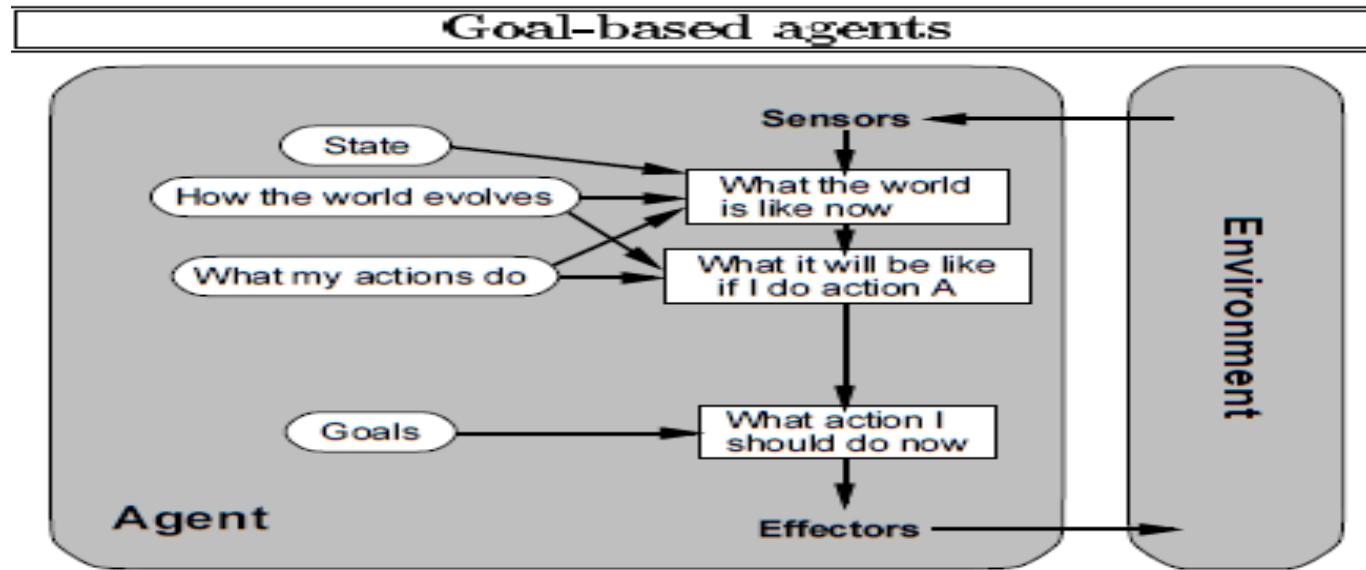
- **Simple reflex agent:** This agent works only on the principle of current perception. Works based on **condition-action rule**



- **Model based reflex agent:** It works by finding a rule whose condition matches the current situation. It can handle actually observable environments.

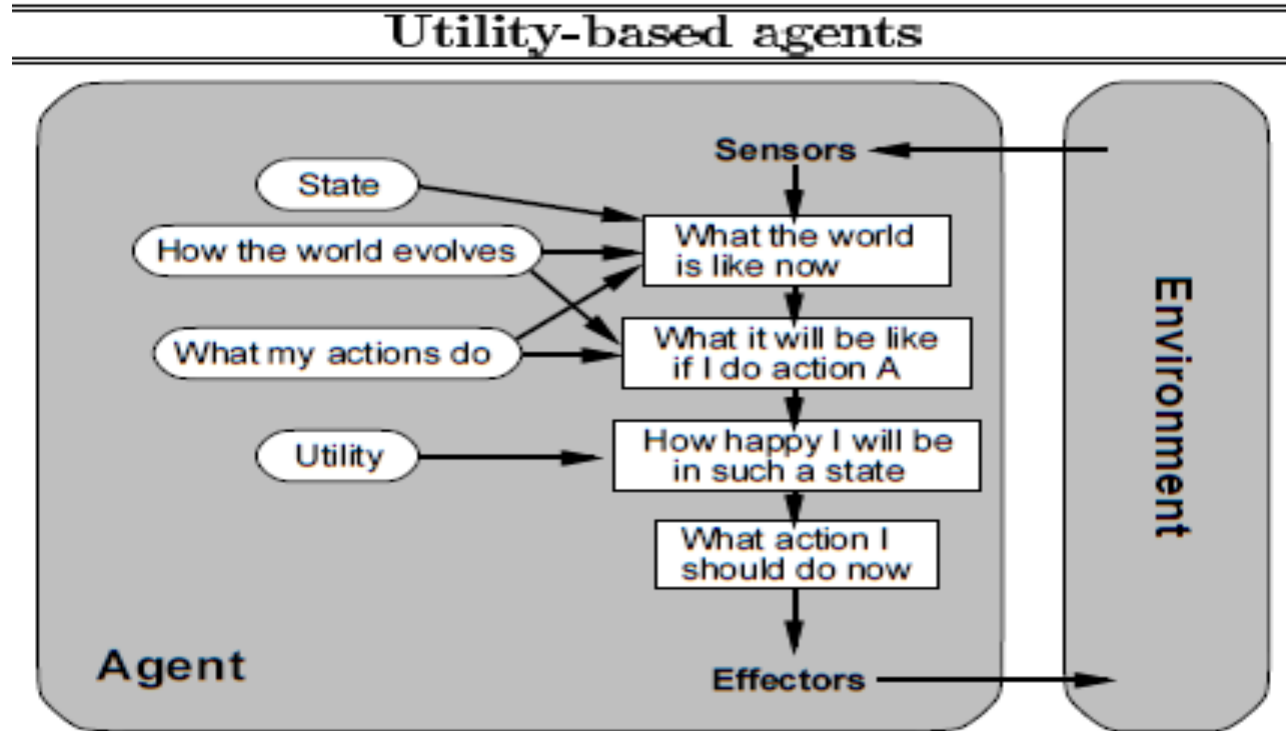


- **Goal based agent:** It mainly focus on reaching the goal set and the decision took by the agent is based on how far it is currently from the goal and desired state.



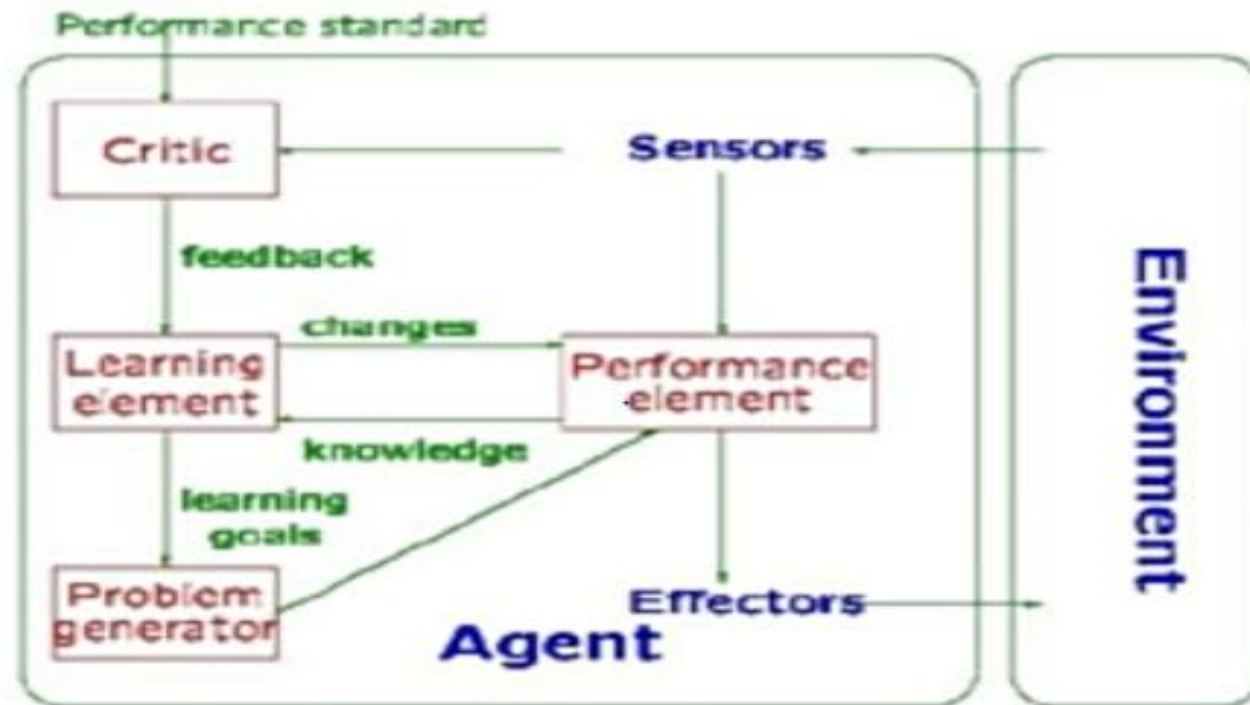
- **Utility based agent:** It is most similar to Goal-based agent but provides an extra component of utility measurement which makes them different by providing a measure of success at a given state.

It is useful when there are multiple possible alternatives and agent has to choose.



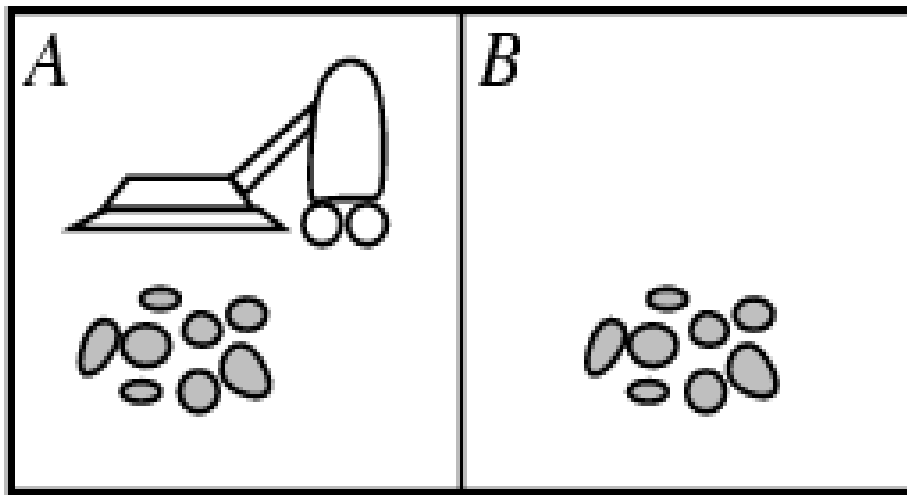
- **Learning agent:** It learns from past experience.
It starts with basic knowledge and able to act, automatic adaptability through learning.

- Learning agent mainly focus on four conceptual components:
 1. Learning elements
 2. Critic elements
 3. Performance element
 4. Problem generator



- **example-the vacuum-cleaner world shown in Fig**

This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck, otherwise move to the other square. A partial tabulation of this agent function is shown in



1.Vaccum cleaner

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

2. Agent Function

- The REFLEX-VACCUUM-AGENT program is invoked for each new percept (location, status) and returns **an** action each time.
- **Function** REFLEX-VACCUUM-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**

- ***Task environments:***

- We must think about task environments, which are essentially the "problems" to which rational agents are the "solutions."

- ***Specifying the task environment (PEAS)***

The rationality of the simple vacuum-cleaner agent, needs specification of

- The performance measure
- The environment
- The agent's actuators and sensors.

PEAS:

All these are grouped together under the heading of the *task environment*. We call this the **PEAS** (*Performance, Environment, Actuators, and Sensors*) description. In designing an agent, the first step must always be to specify the task environment as fully as possible.

Agent Type	Performance Measure	Environments	Actuators	Sensors
Taxi driver	Safe: fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, Signal, horn, display	Cameras, sonar, Speedometer, GPS, Odometer, engine sensors, keyboards, accelerometer

- ***Properties of task environments:***

- Fully observable **vs.** partially observable
- Deterministic **vs.** stochastic
- Episodic **vs.** sequential
- Static **vs.** dynamic
- Discrete **vs.** continuous
- Single agent **vs.** Multiagent

Problem Solving Using Search

STATE
SPACE
SEARCH

Problem Solving by Search

- **Problem searching:**

In general searching refers to finding information for one needs.

Searching is most commonly used technique of problem solving in AI.

- Generally to build a system, to solve a problem what we need:

1. Define (Initial situation)
2. Analyzing (techniques)
3. Isolate and represent
4. Choose the best solution
5. Implementation

This is also called as **problem space** which defines the the various components that go into creating a resolution for a problem and also includes the above 5 points as stages of problem space.

- For example problem solving in games, “**SUDOKU PUZZLE**”

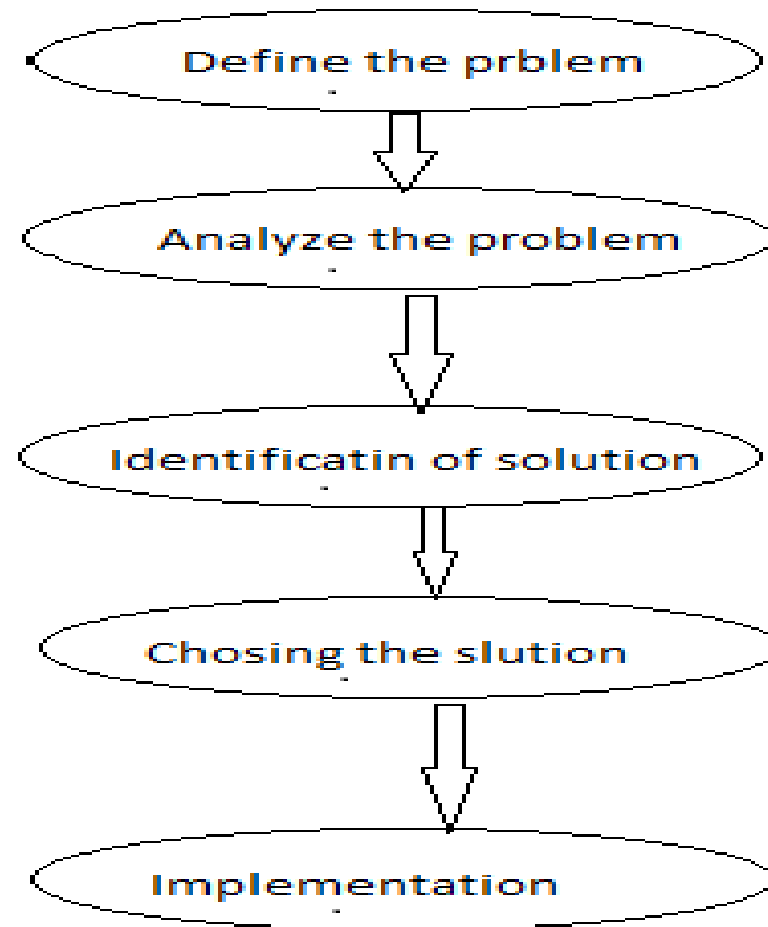
It is done by building an AI system.

To do this first we define the problem statement and generating the solution and keeping the condition in mind.

- Some of the problem solving techniques which helps AI are:

1. Chess
2. Travelling sales man problem
3. N-Queen problem

Flow chart:



Problem-Solving Agents: Introduction

- *In which we see how an agent can find a sequence of actions that achieves its goals, when no single action will do.*
- The method of solving problem through AI involves the process of defining the search space, deciding start and goal states and then finding the path from start state to goal state through search space.
- State **space search** is a process used in the field of computer science, including **artificial intelligence(AI)**, in which successive configurations or states of an instance are considered, with the goal of finding a goal state with a desired property.

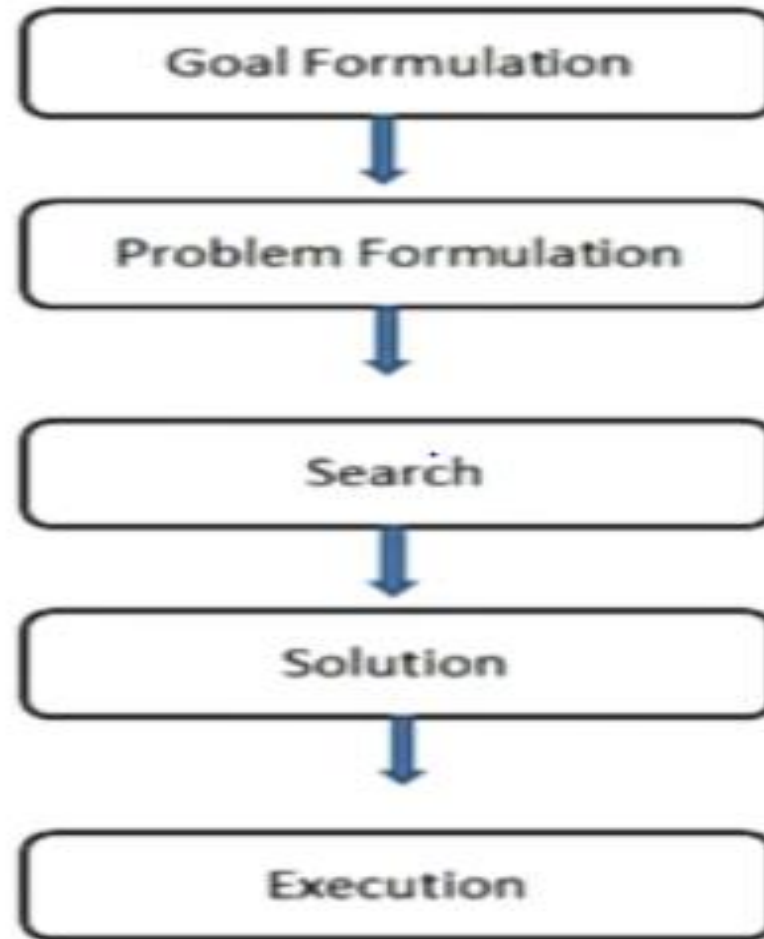
- This is a one-kind of goal-based agent (use structured representation) called a problem-solving agent or rational agent.
- Problem-solving agent use atomic representations, states of the world are considered as wholes, with no internal structure visible to the problem solving algorithms.

The major difference between intelligent and problem solving agent is:

- Intelligent agent maximize the performance
- Problem-solving agents find sequence of actions.

Ex: shortest route path algorithm

Functionality of Problem-Solving Agents:



- **Goal Formulation:**

Problem-solving is about having a goal we want to reach (Ex: we want to travel from A ----- E)

- **Problem Formulation:**

A problem formulation is about deciding what actions and states to consider.

- **Search:**

The process of finding the a sequence is called search.

- **Solution and execute:**

Once the solution is found from different aspects through search recommendation , that sequence of actions will help to carry out the execution.

- Problem-solving agent now simply designed as:

Formulate-----search-----execute

A problem can be defined formally by 4 components:

1.Initial state

- it is the state from which our agents start solving the problem

2.State description

a description of the possible **actions** available to the agent, it is common to describe it by means of a **successor function**, given state x then $SUCCESSOR-FN(x)$ returns a set of ordered pairs $\langle action, successor \rangle$ where *action* is a legal action from state x and *successor* is the state in which we can be by applying *action*.

The initial state and the successor function together defined what is called **state space** which is the set of all possible states reachable from the initial state {e.i: $in(A)$, $in(B)$, $in(C)$, $in(D)$, $in(E)$ }.

3.Goal test

- we should be able to decide whether the current state is a *goal state* {e.i: is the current state is in(E)?}.

4.Path cost

a function that assigns a numeric value to each path, each step we take in solving the problem should be somehow weighted, so If I travel from A to E our agent will pass by many cities, the cost to travel between two consecutive cities should have some cost measure, {e.i: Traveling from 'A' to 'B' costs 20 km or it can be typed as $c(A, 20, B)$ }.

A solution to a problem is path from the initial state to a goal state, and **solution quality** is measured by the path cost, and the **optimal solution** has the lowest path cost among all possible solutions.

Goal Directed Agent

- A goal directed agent needs to achieve certain goals.
- Many problems can be represented as a set of states and a set of rules of how one state is transformed to another
- The agent must choose a sequence of actions to achieve the desired goal.

Definitions

Each state is an abstract representation of the agent's environment. It is an abstraction that denotes a configuration of the agent.

- **Initial state** : The description of the starting configuration of the agent
- An **action/ operator** takes the agent from one state to another state. A state can have a number of successor states.
- A *plan* is a sequence of actions.

Definitions

Each state is an abstract representation of the agent's environment. It is an abstraction that denotes a configuration of the agent.

- Initial state : The description of the starting configuration of the agent
- An action/ operator takes the agent from one state to another state. A state can have a number of successor states.
- A *plan* is a sequence of actions.

Definitions

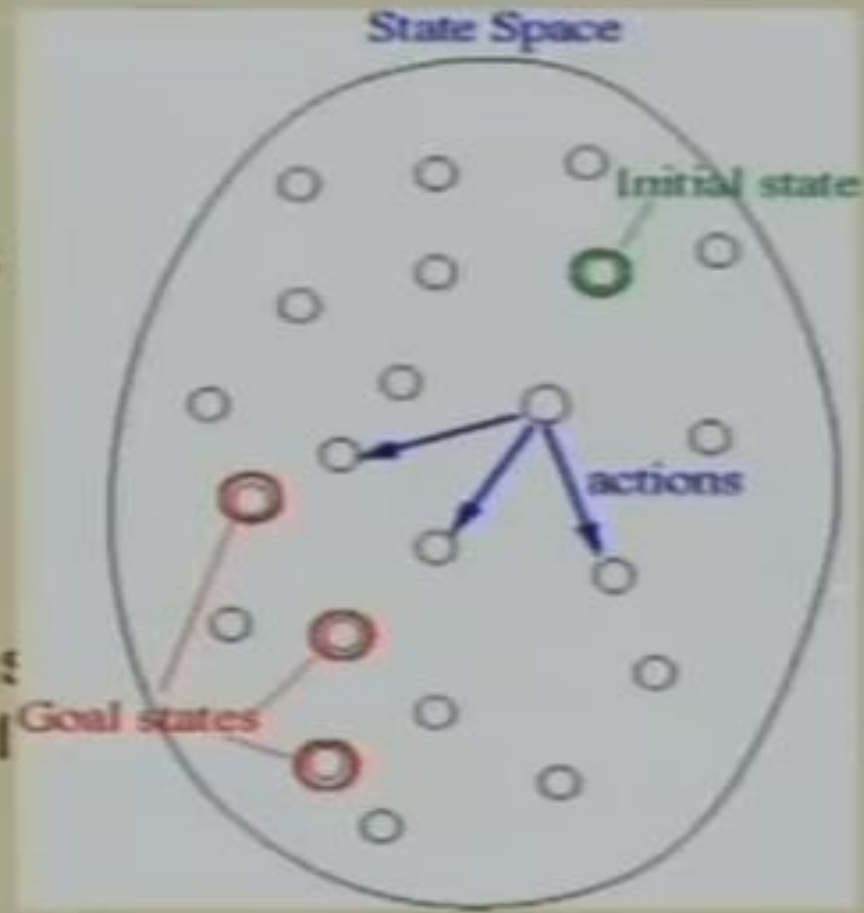
- A *goal* is a description of a set of desirable states of the world. Goal states are often specified by a goal test which any goal state must satisfy.
- Path cost : path \rightarrow positive number
Usually path cost = sum of step costs

Definitions

- *Problem formulation* means choosing a relevant set of states to consider, and a feasible set of operators for moving from one state to another.
- *Search* is the process of *imagining* sequences of operators applied to the initial state, and checking which sequence reaches a goal state.

Search Problem

- S : the full set of states
- s_0 : the initial state
- $A: S \rightarrow S$ set of operators
- G : the set of final states. $G \subseteq S$
- Search problem :
Find a sequence of actions which transforms the agent from the initial state to a goal state $g \in G$.



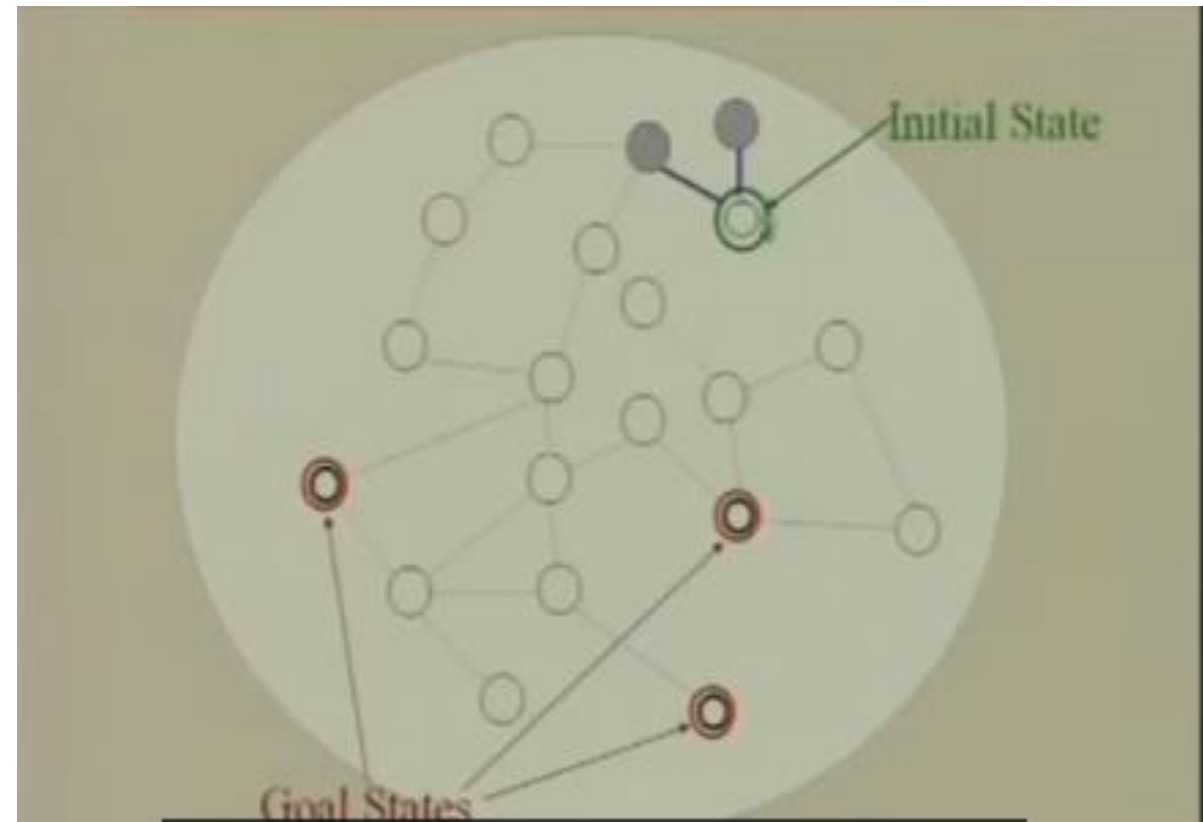
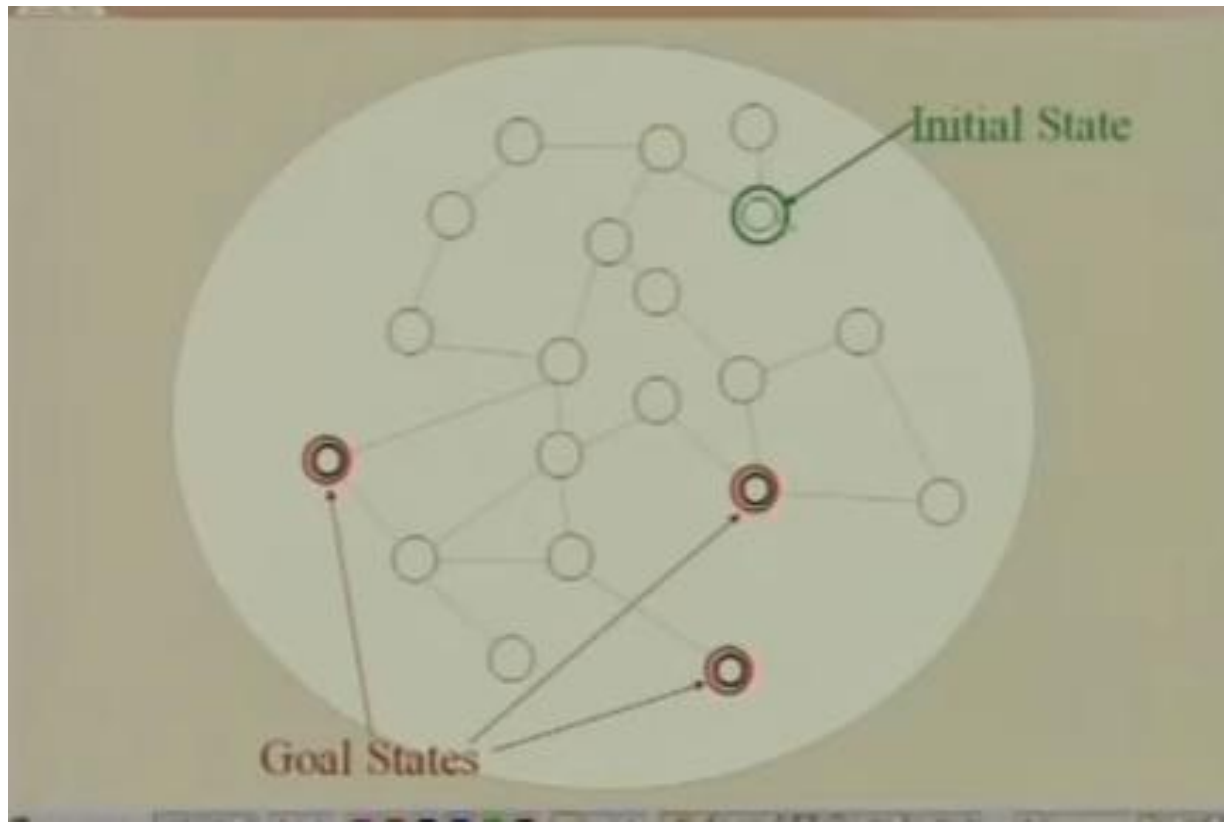
Search Problem

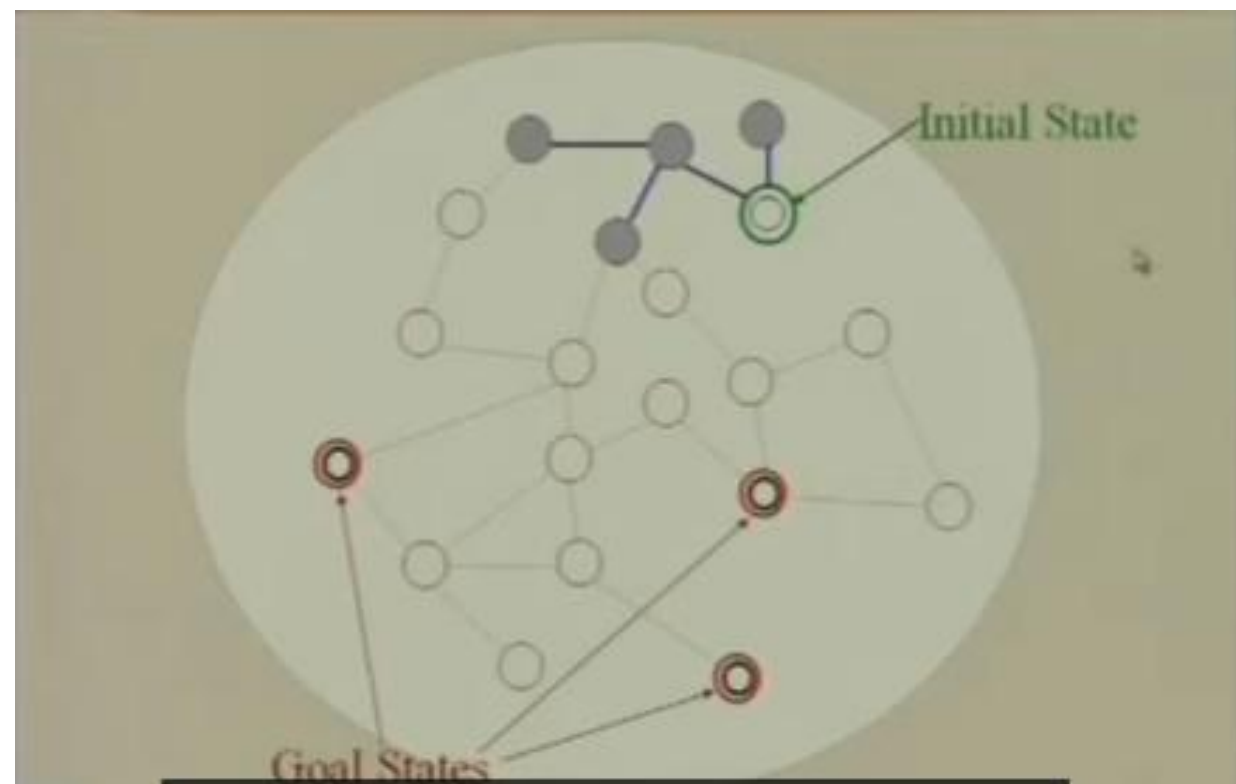
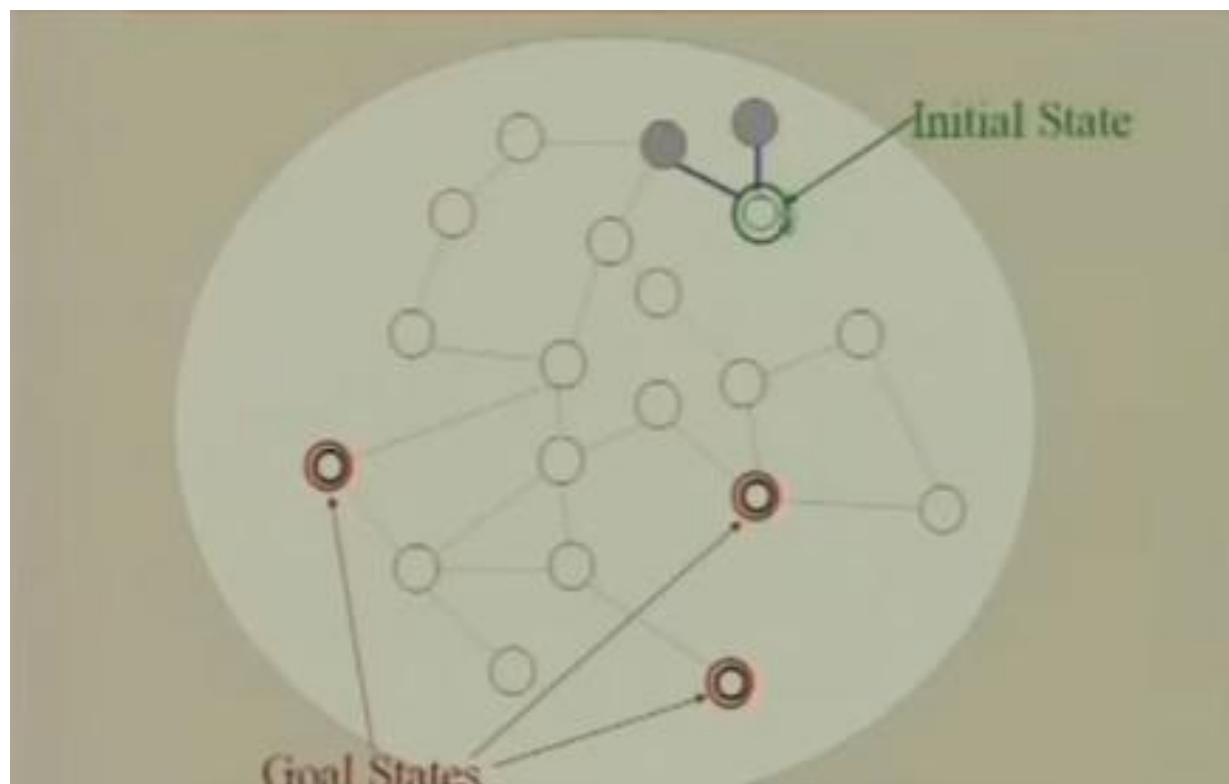
- The search problem consists of finding a solution plan, which is a path from the current state to the goal state.
- Representing search problems
 - ▶ A search problem is represented using a directed graph.
 - ◆ The states are represented as nodes.
 - ◆ The allowed actions are represented as arcs.

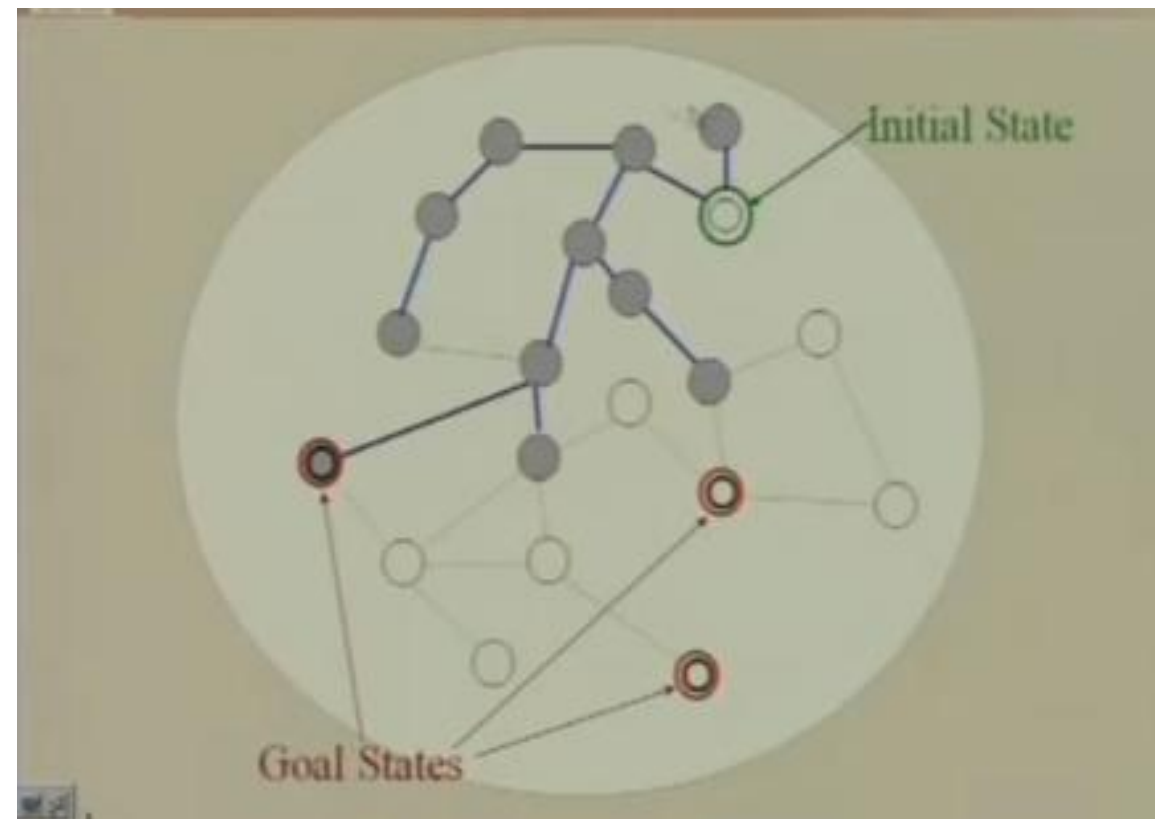
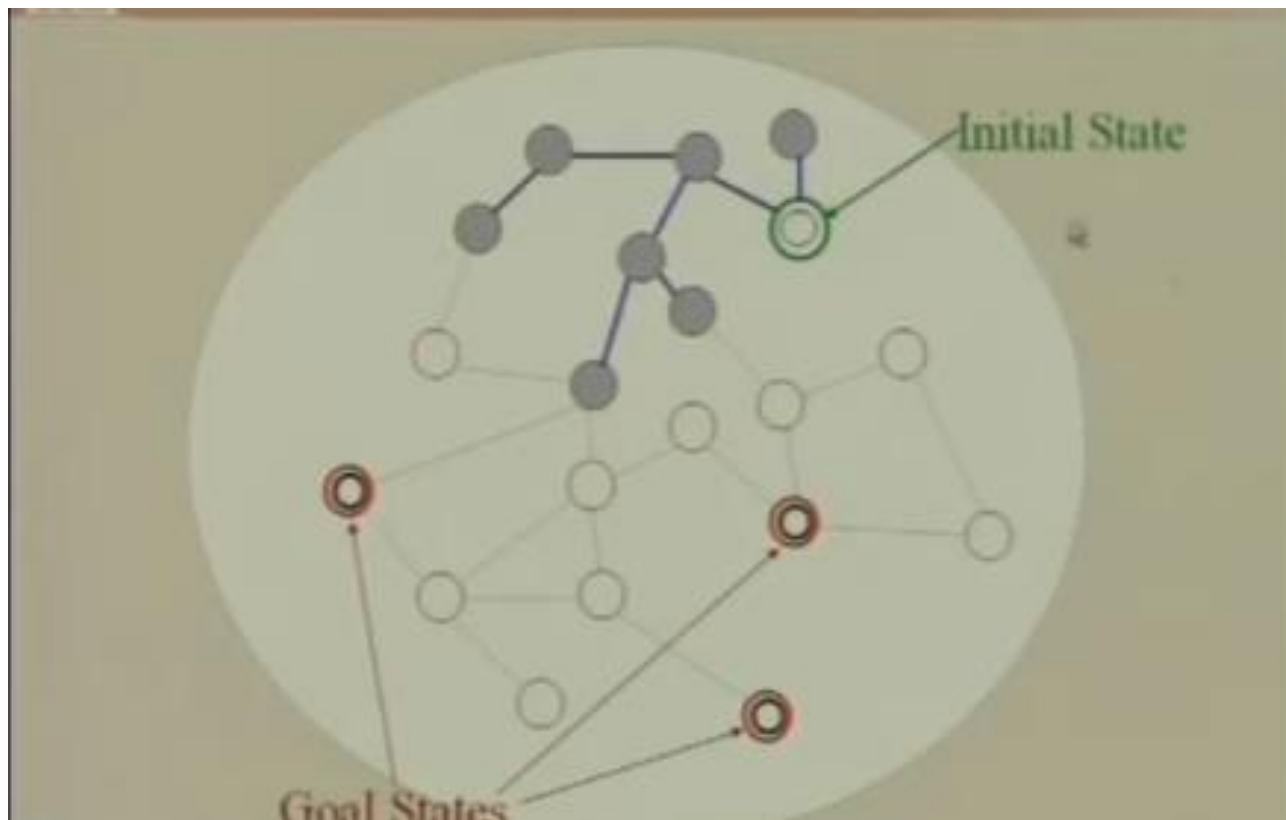
Searching Process

- Check the current state
- Execute allowable actions to move to the next state
- Check if the new state is a solution state
 - ▶ If it is not, the new state becomes the current state and the process is repeated until a solution is found or the state space is exhausted.

State Space







Pegs and Disks

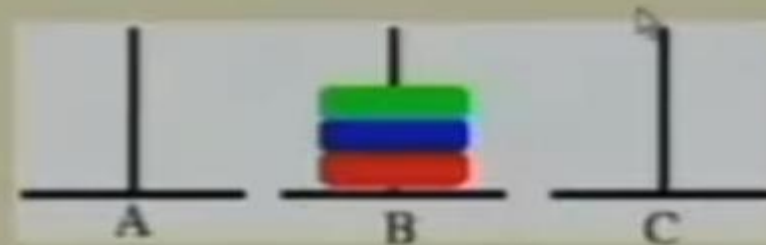
- Consider the following problem. We have 3 pegs and 3 disks.

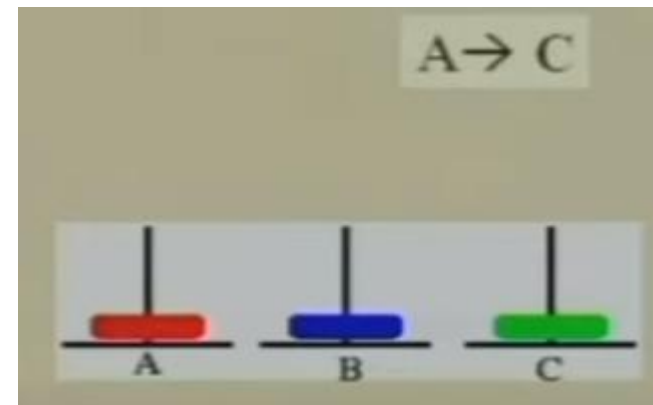
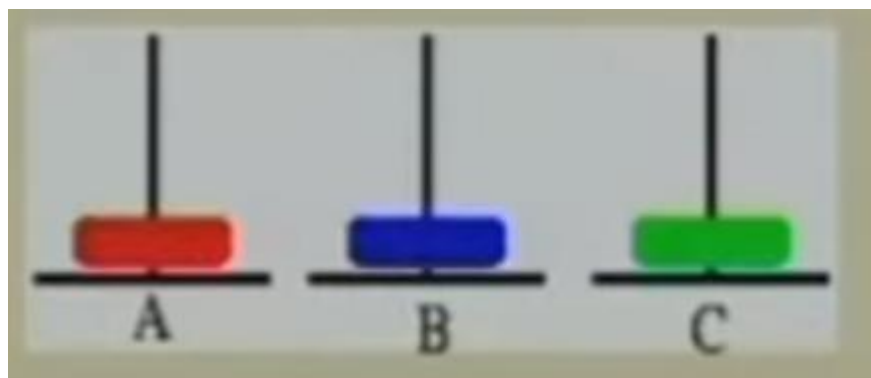
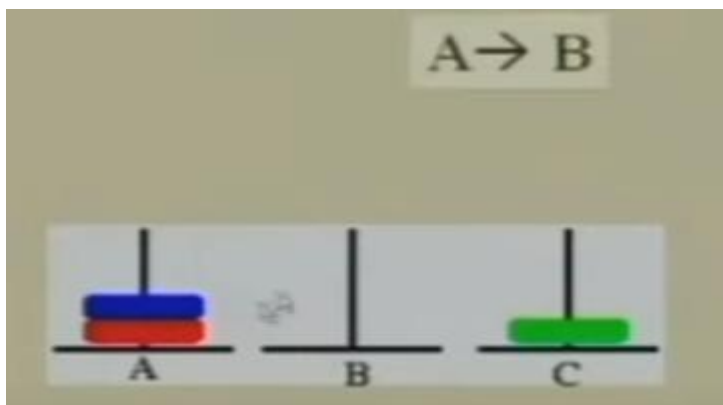
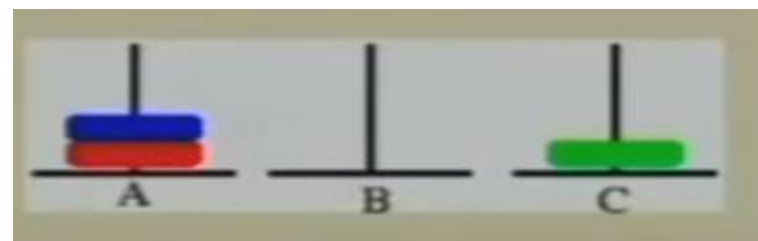
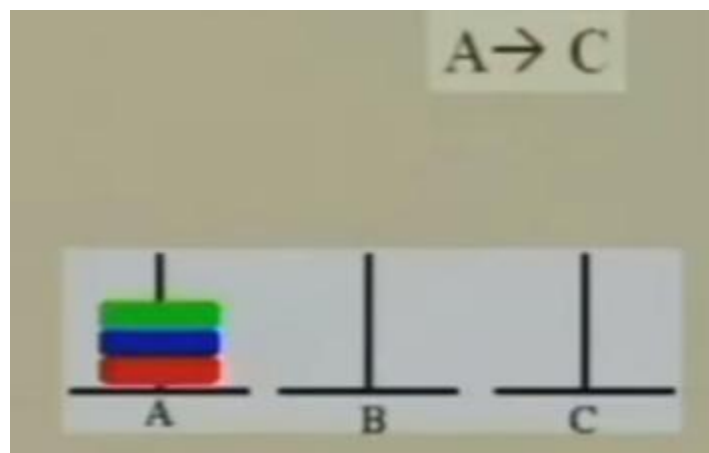
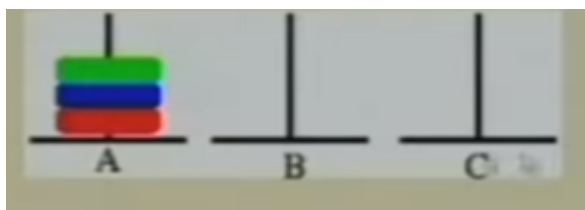
- Operators: one may move the topmost disk on any needle to the topmost position to any other needle

Initial state

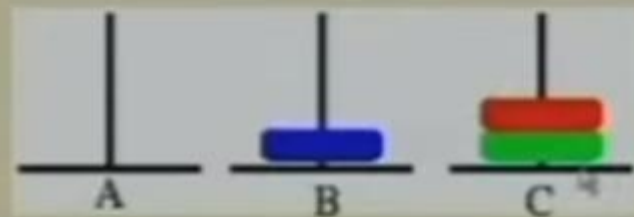


Goal configuration

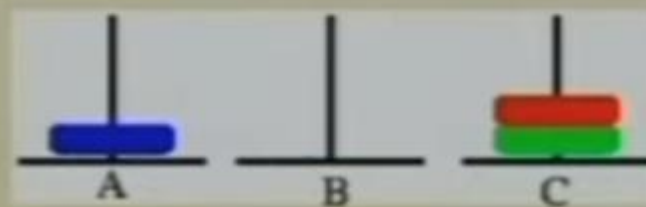




$B \rightarrow A$



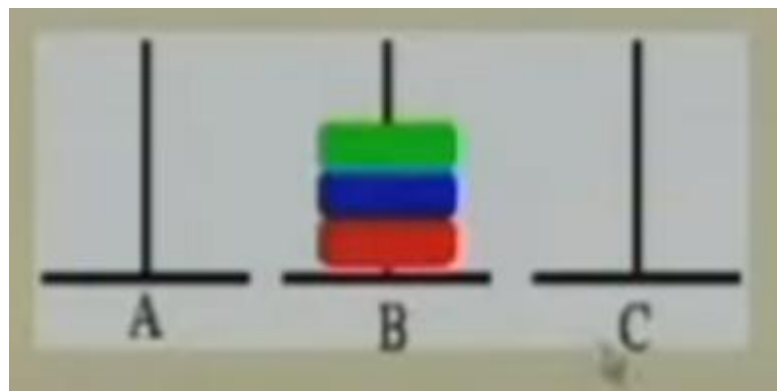
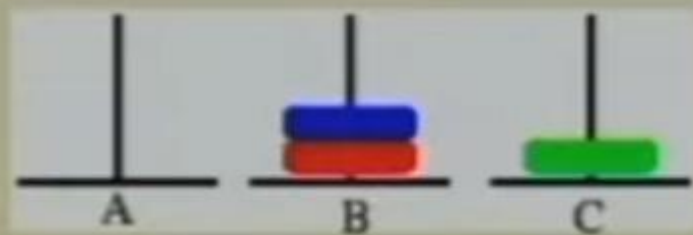
$C \rightarrow B$



$A \rightarrow B$

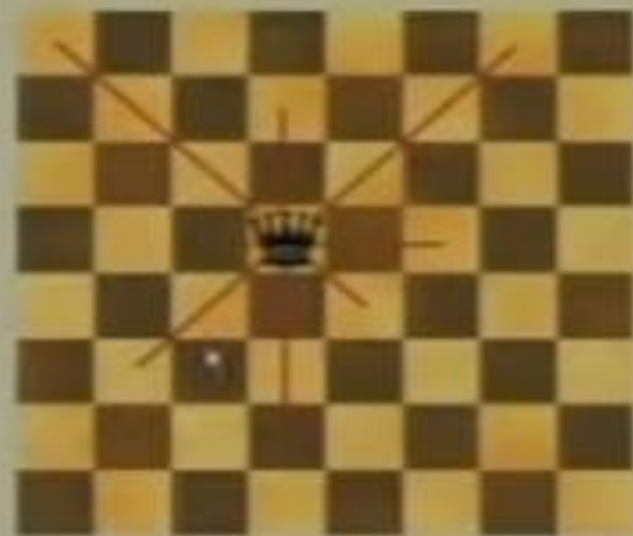
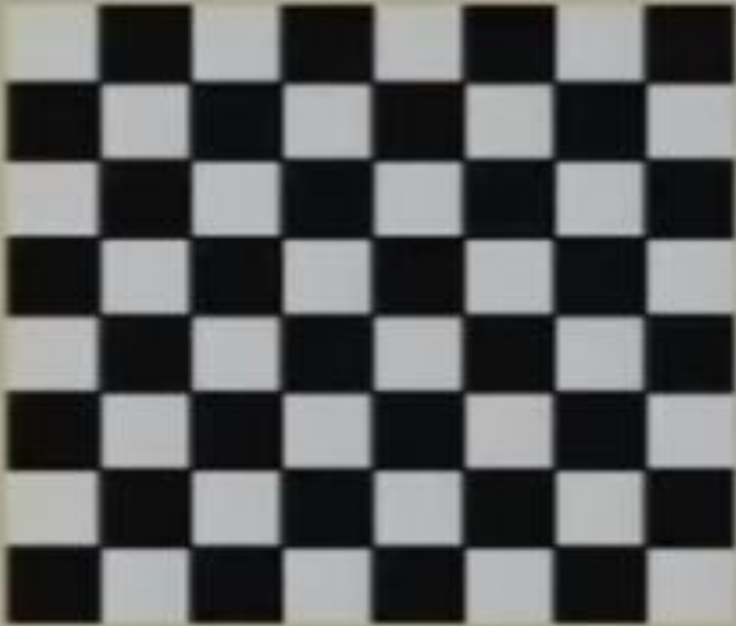


$C \rightarrow B$

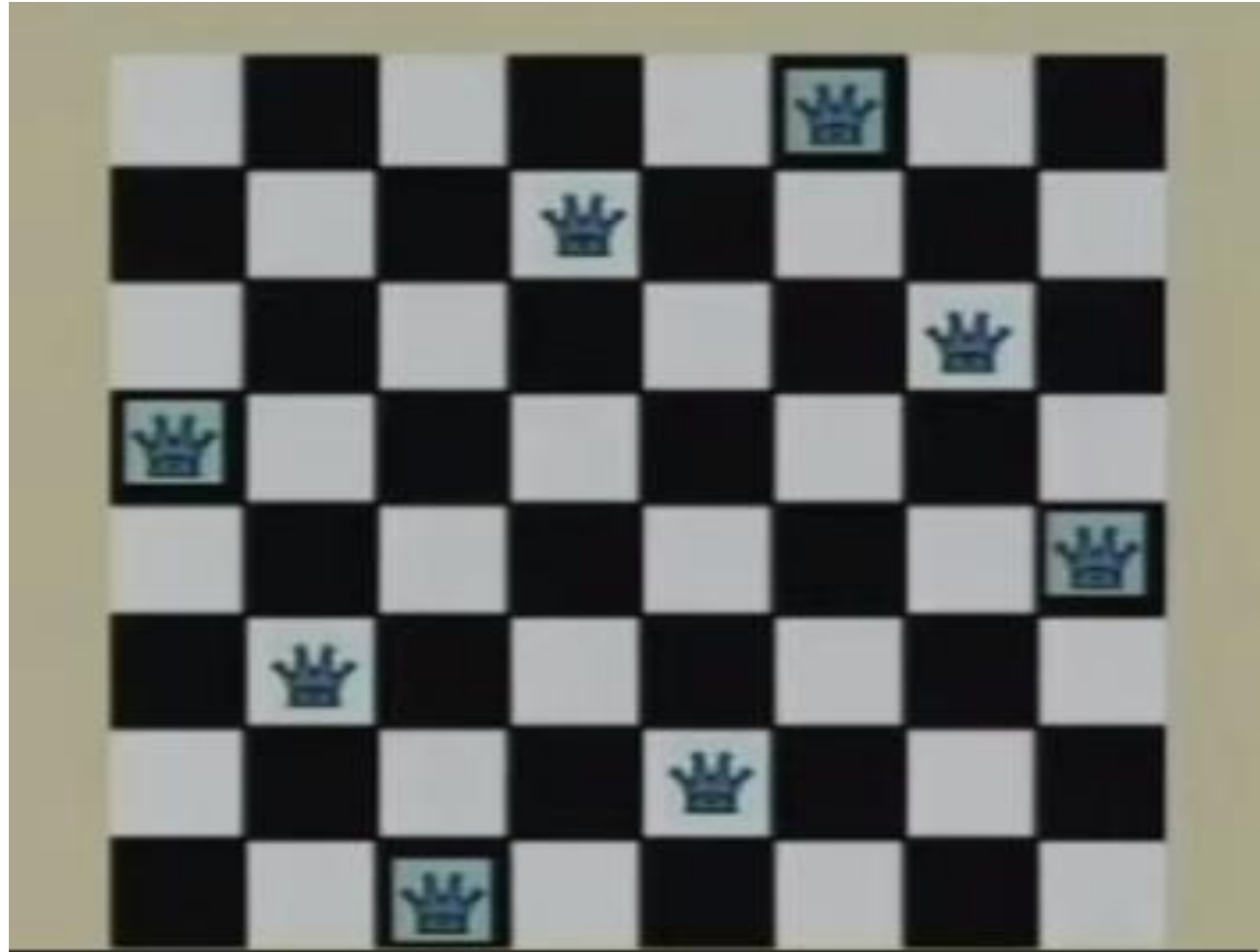


8 Queens

Place 8 queens on a chessboard so that no two queens are in the same row, column or diagonal

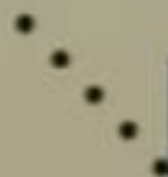


8 Queens Solution



N Queens Problem Formulation 1

- States : Any arrangement of 0 to 8 queens on the board
- Initial state : 0 queens on the board
- Successor function : Add a queen in any square
- Goal test : 8 queens on the board, none are attacked



64 successors

N Queens Problem Formulation 2

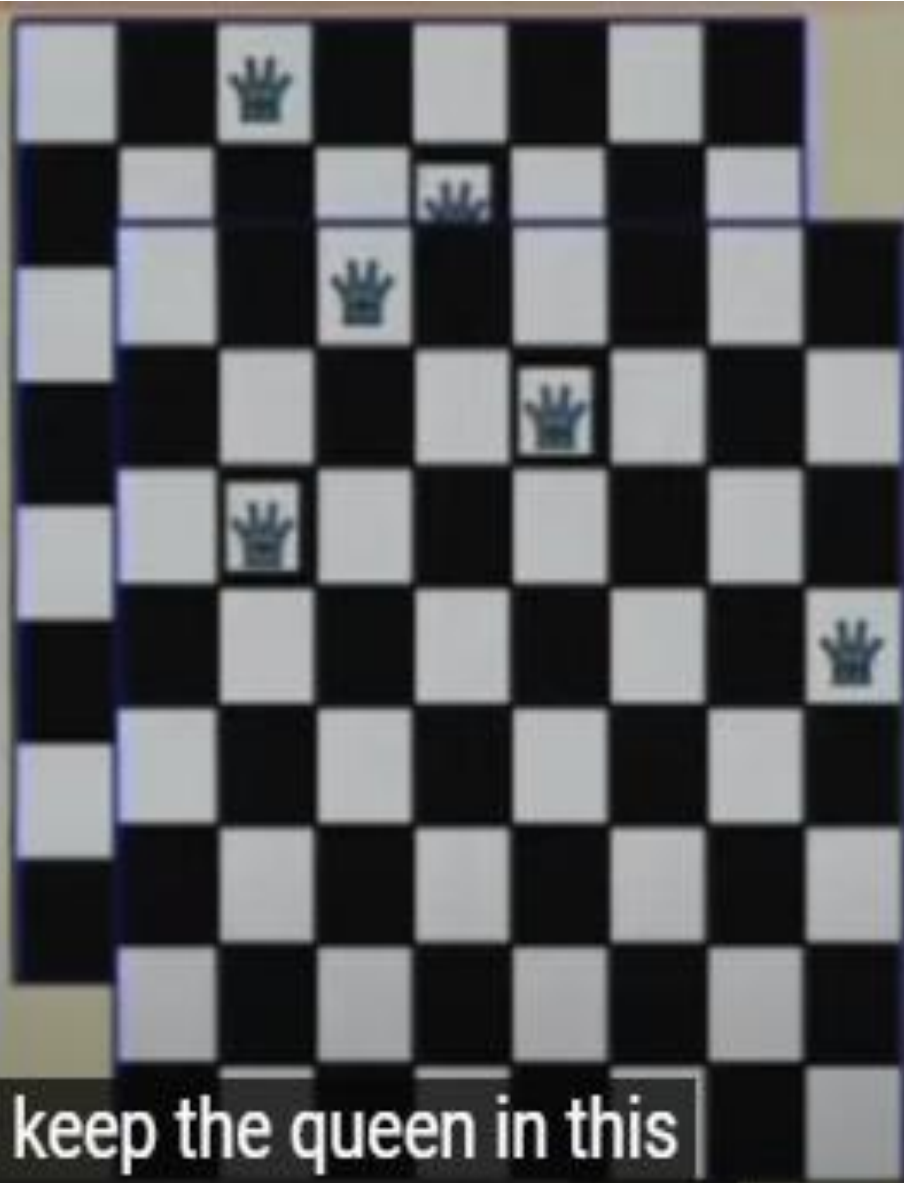
- States : Any arrangement of 8 queens on the board
- Initial state : All queens are at column 1
- Successor function : Change the position of any one queen
- Goal test : 8 queens on the board, none are attacked





N Queens Problem Formulation 3

- States : Any arrangement of k queens in the first k rows such that none are attacked
- Initial state : 0 queens on the board
- Successor function : Add a queen to the $(k+1)$ th row so that none are attacked.
- Goal test : 8 queens on the board, none are attacked



by this one, we can keep the queen in this
position, we c Subtitles/closed captions (c) e, we

Other Examples:



Vacuum World

- **Initial state:**
 - Our vacuum can be in any state of the 8 states shown in the picture.
- **State description:**
 - Successor function generates legal states resulting from applying the three actions {Left, Right, and Suck}.
 - The states space is shown in the picture, there are 8 world states.
- **Goal test:**
 - Checks whether all squares are clean.
- **Path cost:**
 - Each step costs 1, so the path cost is the sum of steps in the path.

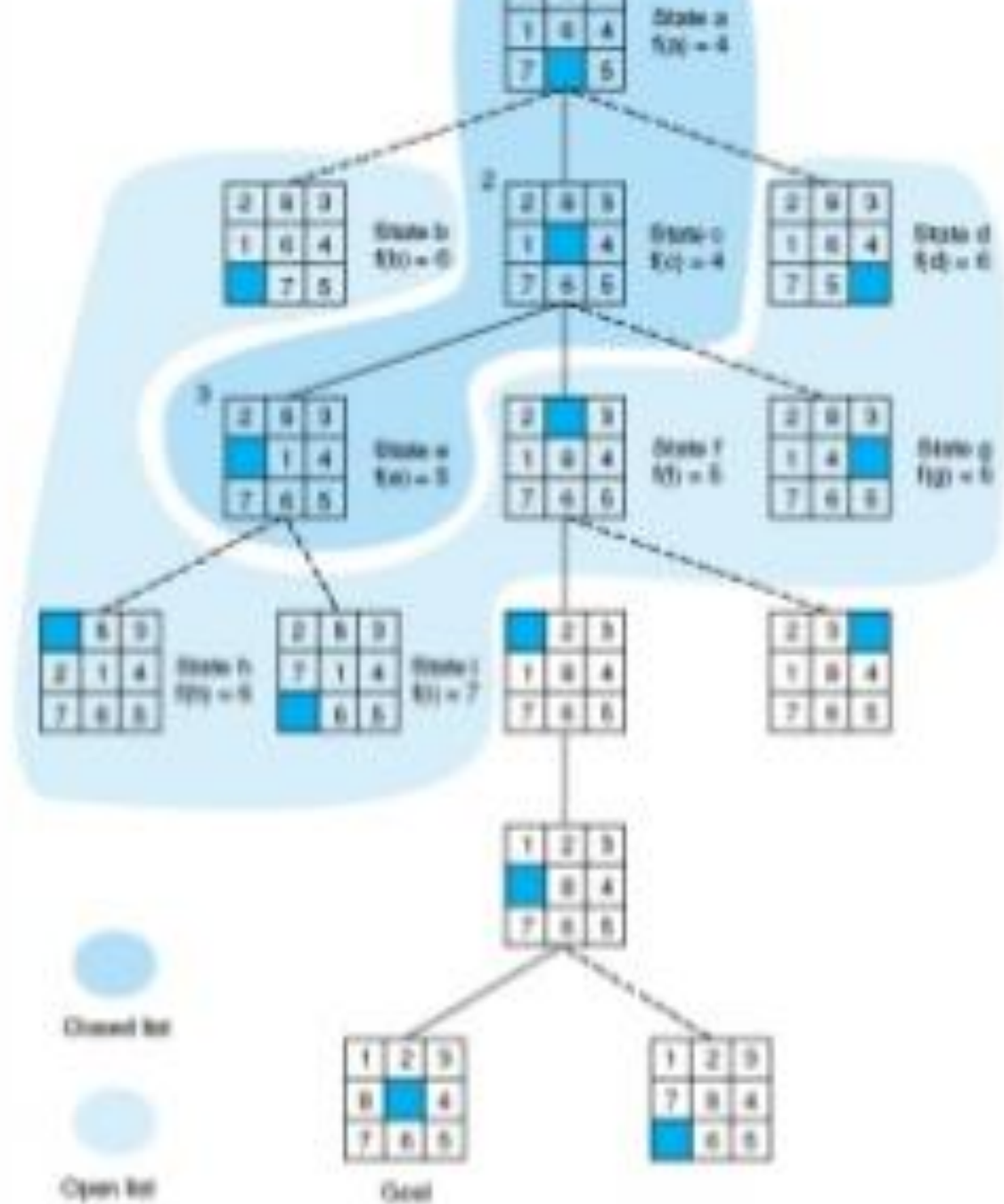
8-Puzzle

- Initial state:
 - Our board can be in any state resulting from making it in any configuration.
- State description:
 - Successor function generates legal states resulting from applying the three actions {move blank Up, Down, Left, or Right}.
 - State description specifies the location of each of the eight tiles and the blank.
- Goal test:
 - Checks whether the states matches the goal configured in the goal state shown in the picture.
- Path cost:
 - Each step costs 1, so the path cost is the sum of steps in the path.

Start State

Problems Solving Agent/Answer Solution

Goal State





Real World Problems

■ Airline Travelling Problem

■ States:

- Each is represented by a location and the current time.

■ Initial State:

- This is specified by the problem.

■ Successor Function:

- This returns the states resulting from taking any scheduled flight, leaving later than the current time plus the within airport transit time, from the current airport to another.

■ Goal Test:

- Are we at the destination by some pre-specified time?

■ Path Cost:

- This depends on the monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of air place, frequent-flyer mileage awards and so on.

Different types of search strategies:

- **Search strategies can be :**

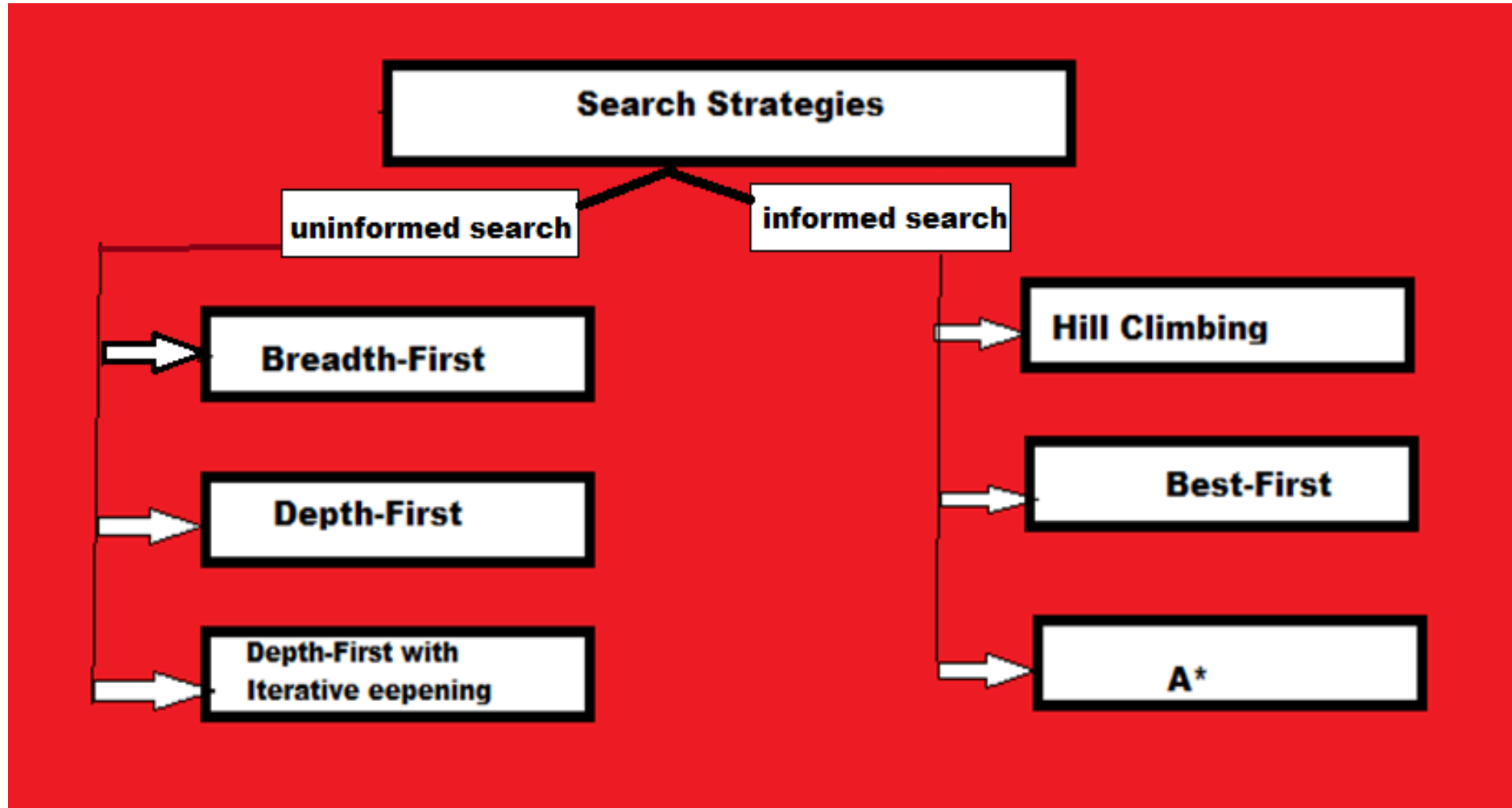
Uninformed search strategies

Informed search strategies

- **Uninformed search strategies:** These are also called as Blind search. These search strategies use only the information available in the problem definition. **(This includes Breadth-first search, Uniform cost search, Depth-first search, Iterative deepening depth-first search and Bidirectional Search)**

Informed search strategies: These are also called as Heuristic search. These search strategies use other domain specific information. **(This includes Hill climbing, Best-first, Greedy Search and A* Search)**

Search strategies classification:



Difference between uninformed and informed search:

Un -Informed Search

1. Nodes in the state are searched mechanically, until the goal is reached or time limit is over / failure occurs.
 2. Info about goal state may not be given
 3. Blind grouping is done
 4. Search efficiency is low.
 5. Practical limits on storage available for blind methods.
 6. Impractical to solve very large problems.
 7. Best solution can be achieved.
- E.g : DFS , BFS , Branch & Bound , Iterative Deepening ...etc.

Informed Search

1. More info. About initial state & operators is available . Search time is less.
 2. Some info. About goal is always given.
 3. Based on heuristic methods
 4. Searching is fast
 5. Less computation required
 6. Can handle large search problems
 7. Mostly a good enough solution is accepted as optimal solution.
- E.g: Best first search , A* , AO * , hill climbing...etc

Search Algorithm Terminologies:

- Search
 - Search problem may have three factors: search space, start state, goal state
- Search tree
- Actions
- Transition model
- Path cost
- solution
- Optimal solution

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.

Definition of search problem:

- a *search problem* is defined by:
 - a *state space* (i.e., an initial state or set of initial states and a set of operators)
 - a *set of goal states* (listed explicitly or given implicitly by means of a property that can be applied to a state to determine if it is a goal state)
- a *solution* is a path in the state space from an initial state to a goal state

Exploring the state space:

- *search* is the process of exploring the state space to find a solution
- exploration starts from the initial state
- the search procedure applies operators to the initial state to generate one or more new states which are hopefully nearer to a solution
- the search procedure is then applied recursively to the newly generated states
- the procedure terminates when either a solution is found, or no operators can be applied to any of the current states

Search Trees:

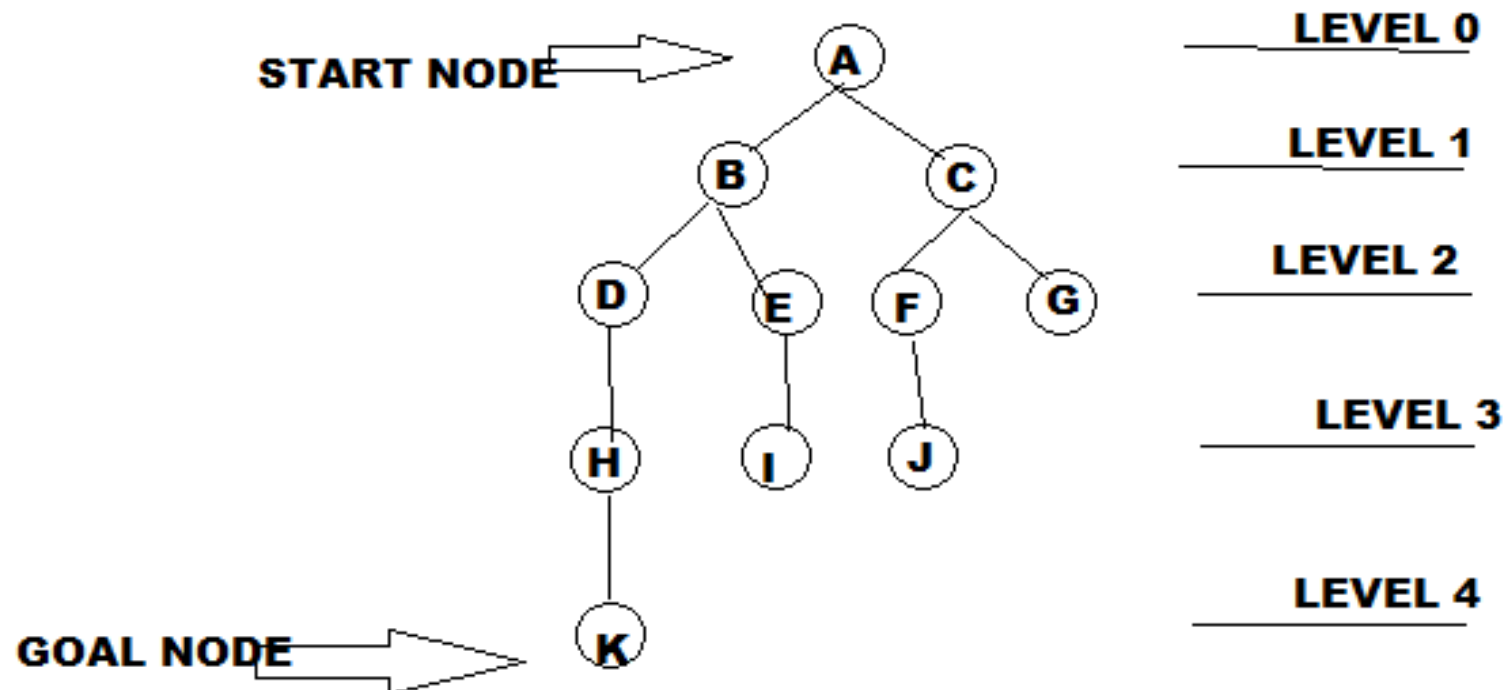
- the part of the state space that has been explored by a search procedure can be represented as a *search tree*
- nodes in the search tree represent *paths* from the initial state (i.e., partial solutions) and edges represent operator applications
- the process of generating the children of a node by applying operators is called *expanding* the node
- the branching factor of a search tree is the average number of children of each non-leaf node
- if the branching factor is b , the number of nodes at depth d is b^d

Breadth-First search (BFS):

- proceeds level by level down the search tree
- first explores all paths of length 1 from the root node, then all paths of length 2, length 3 etc.
- starting from the root node (initial state) explores all children of the root node, left to right
- if no solution is found, expands the first (leftmost) child of the root node, then expands the second node at depth 1 and so on ...

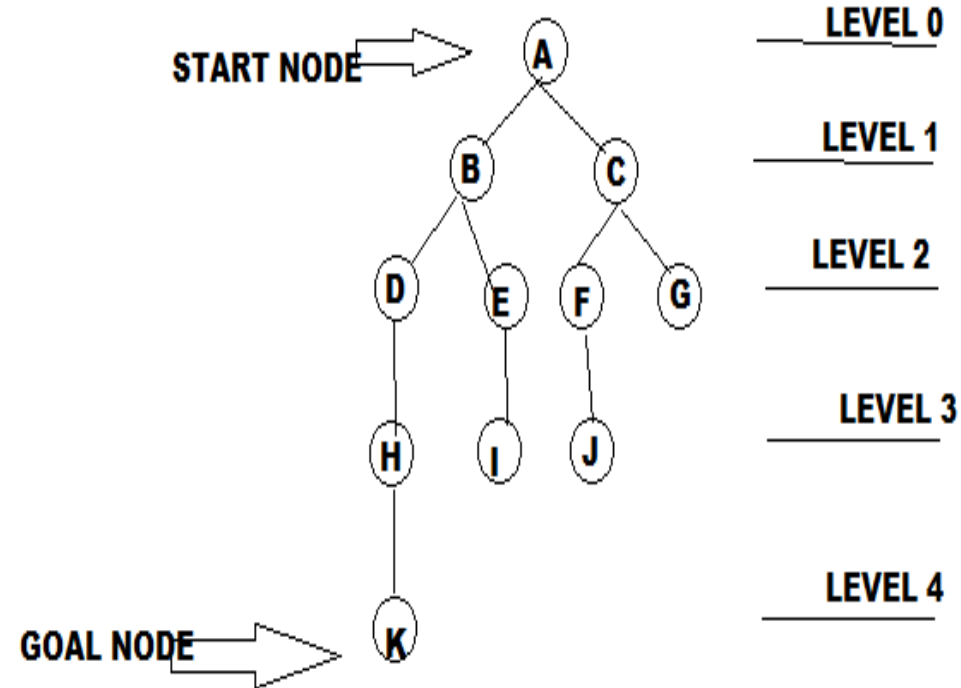
- It is the most common search strategy for traversing a tree or graph
- This algorithm searches in breadth-wise in a tree/graph, so it is called as breadth-first search
- BFS algorithm starts searching from the root node of the tree and expand all nodes (successor nodes) at the current state before moving to the next node of next level
- BFS is implemented using FIFO queue DS

Example:



- Let us see how the BFS is following the FIFO order (FIRST IN FIRST OUT):
- A -----start node (check the possibilities for node A) **i.e A is first entered into queue**

~~A~~
~~BC~~
~~CDE~~
~~DEFG~~
~~EFGH~~
~~FGHI~~
~~GHIJ~~
~~HIJ~~
~~IJK~~
~~JK~~
 K -----**GOAL NODE**



- Time complexity: This is obtained by the no of nodes traverse in BFS until the shallowest little depth
- Let d is the depth of shallowest
- Let b is the node at every state

$$\text{Then } T(b) = B + B^2 + \dots + B^d = O(B^d)$$

Space complexity: It is given by memory complexity i.e., the memory required to store each node

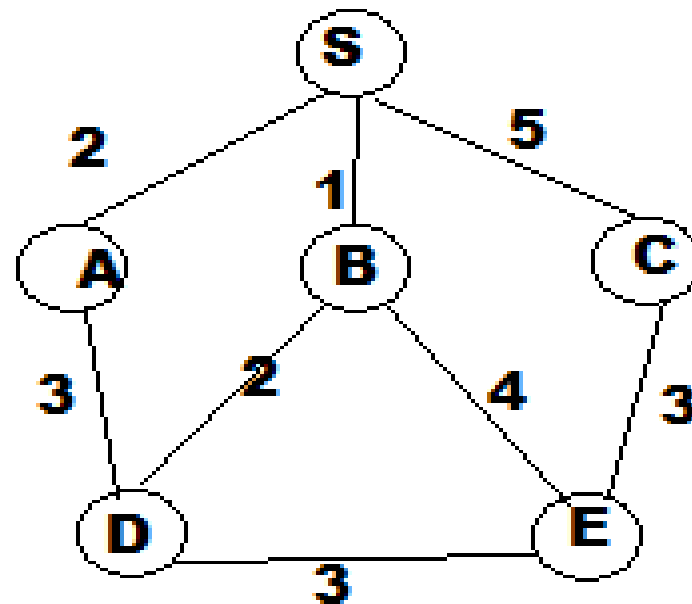
$$\text{Then } s(b) = O(B^d)$$

Completeness: BFS is complete

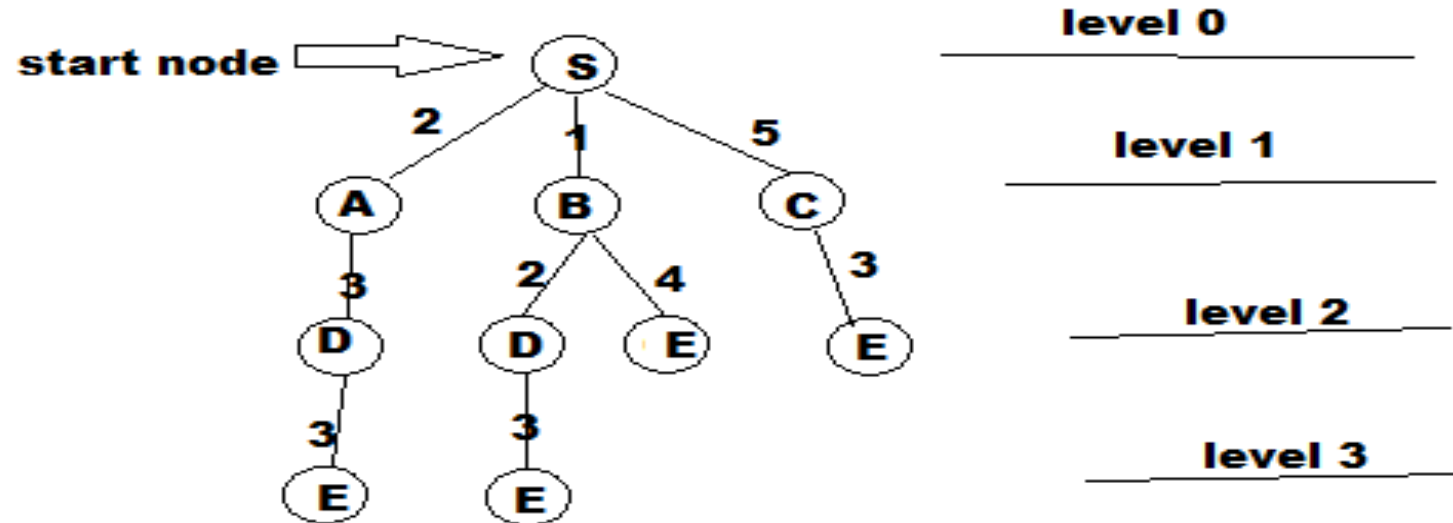
Optimal: Yes it is optimal

Example BFS:

- Find the route path from S to E using BFS



- Expand all possibilities from each and every node.
- After expanding with all possibilities the below one is the tree representation with start node A and final node E.
- Based on the path cost the path should be SBE (or) SCE



- Let us see how the BFS is following the FIFO order (FIRST IN FIRST OUT):
- S -----start node (check the possibilities for node S) i.e S is first entered into queue

~~S~~

~~ABC~~

~~BCD~~

~~CDDE~~

~~DDEE~~

~~DEEE~~

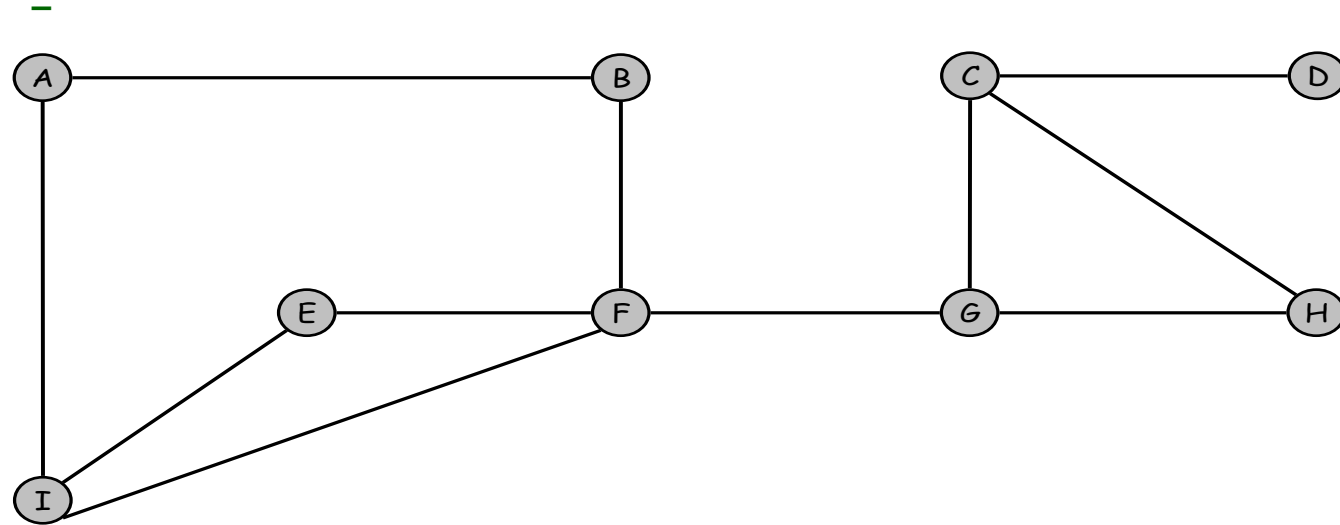
~~EEEE~~

~~EEE~~

~~EE~~

E -----goal node reached

Breadth First Search



front



FIFO Queue

Depth-First search

- DFS may be recursive or non recursive algorithm.
- It is a recursive algorithm for traversing a tree / graph
- In this it starts from the root node and follows each path of its greatest depth node before moving to next path. That's the reason to call it as DFS.
- So it travels from top to bottom direction but not like BFS.
- DFS uses **stack DS (LIFO)** for its implementation.
- The process is similar to BFS algorithm.

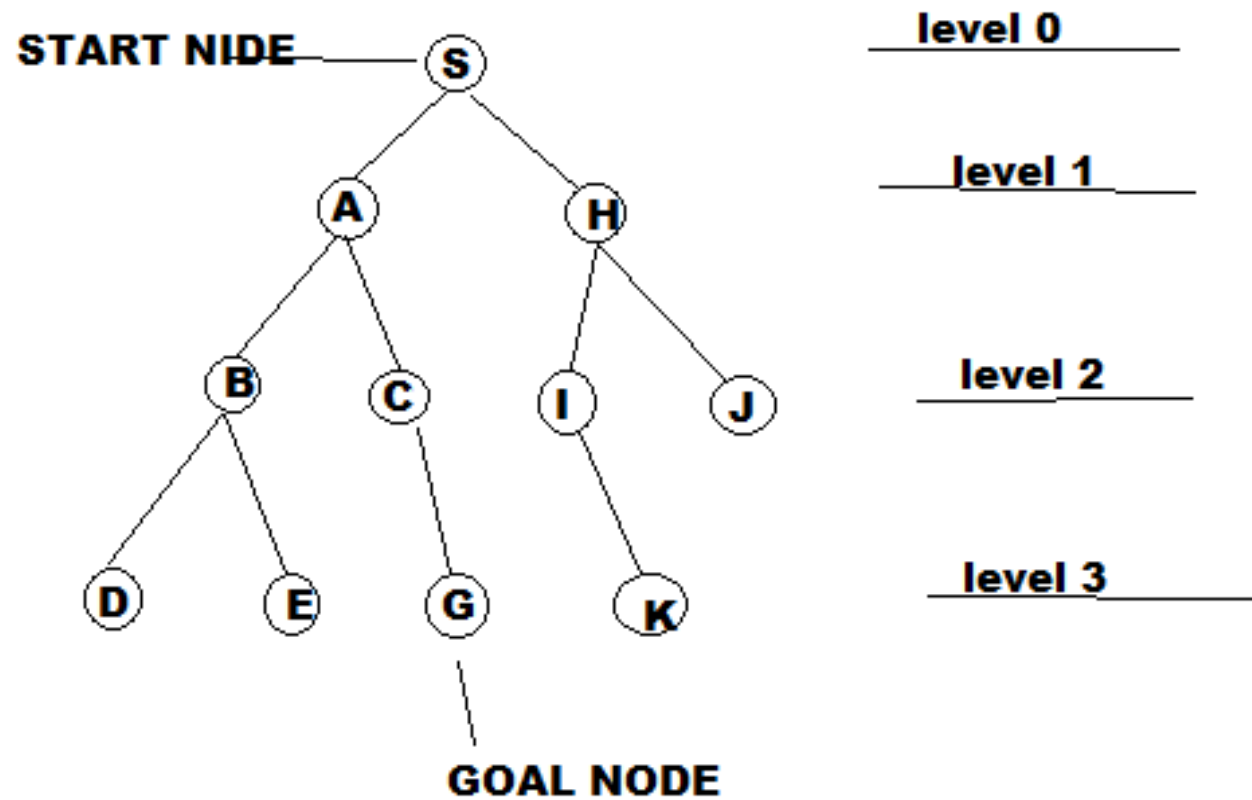
- Advantages:

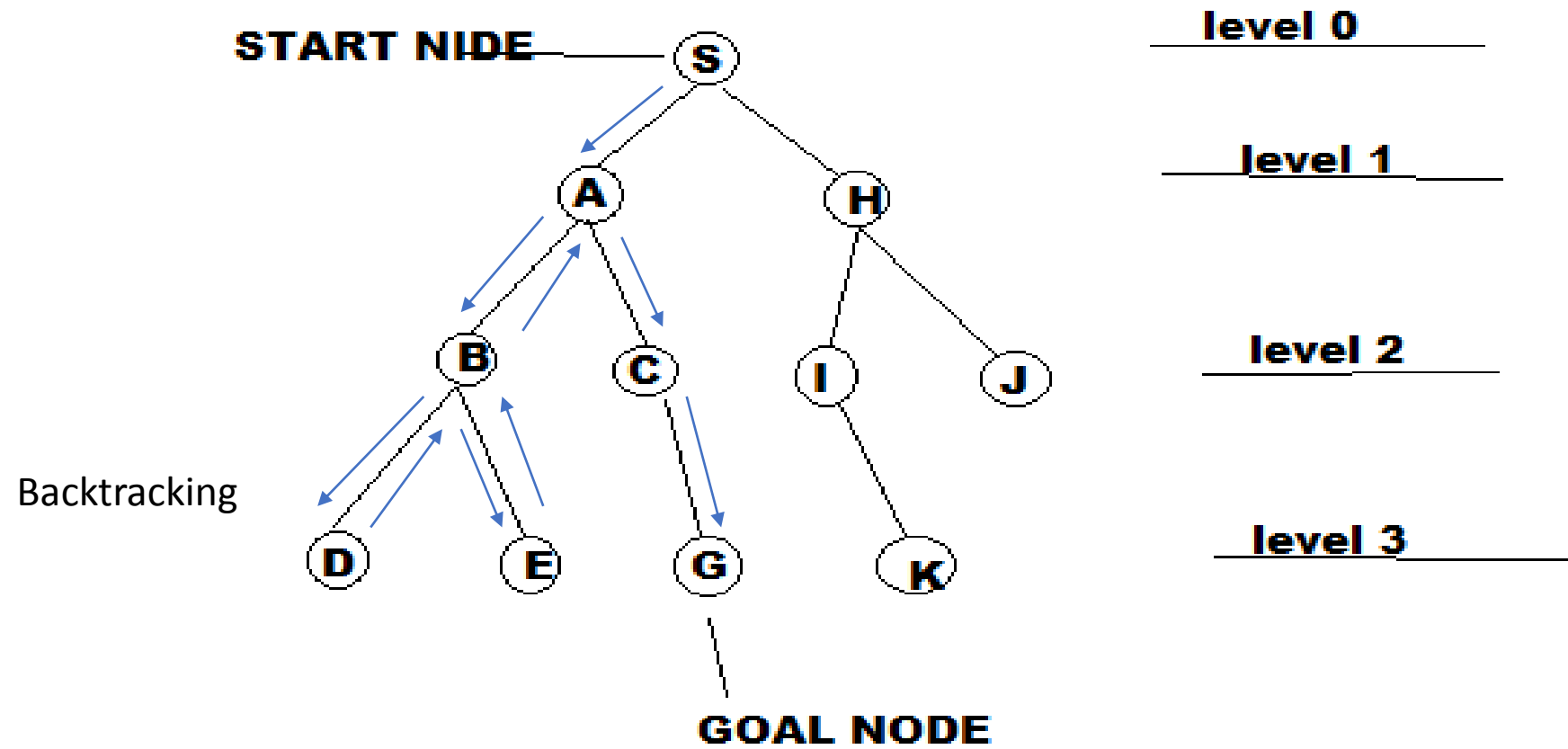
1. It requires less memory as it only needs to store a stack of the nodes on the path from root node to current node.
2. It takes less time to reach to the goal node than BFS alg

Disadvantages:

1. There is a chance of re-occurring of many states and there is no guarantee of finding the solution.
2. DFS alg goes for deep down searching and sometimes it may go to infinite loop

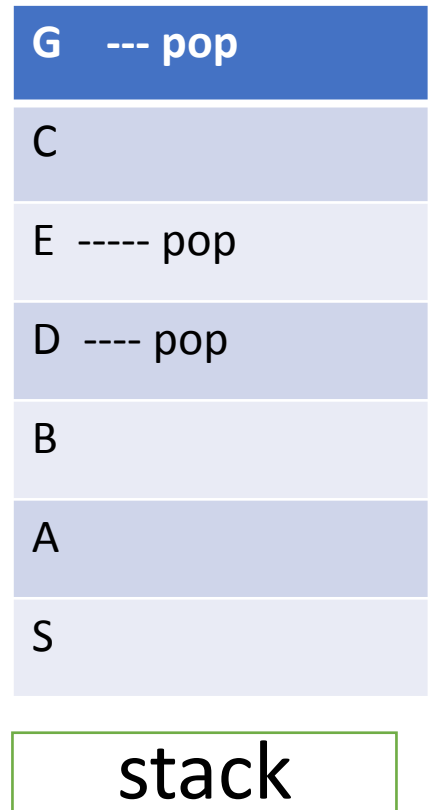
Example: DFS





- It starts searching from the root nodes.
- So starts traversing from root node 'S'.
- The visited nodes will be pushed into the stack by checking their successor nodes (i.e A and H)
- Then it traverse to A (visited and successor nodes are B & C)
- Traverse from A ----- B (visited and successor nodes are D&E)
- Traverse from B ----- D (visited and no successor nodes)
- Since there are no expanded nodes so it will be popped from the stack and starts backtracking.

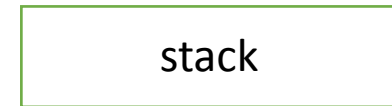
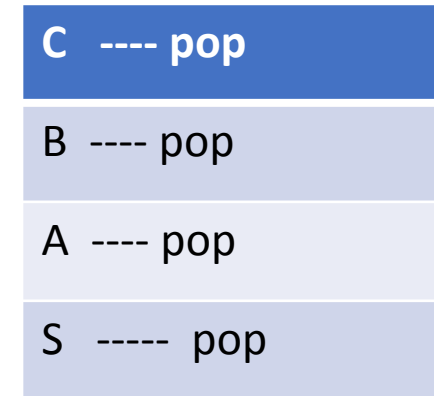
Now traverse back to “B”, since it is already visited check it successor node i.e “E” (visited and no successor nodes)



- Since there are no expanded nodes so it will be popped from the stack and starts backtracking.
- Now traverse back to “B” then to “A”, but is visited and check for successor nodes which are not visited.
- “A” has successor node “C” (visited and successor nodes are G)
- Traverse from C --- G (visited and no successor nodes.)
- “G” is the goal node and stops searching when reaches goal node
- Since there are no expanded nodes so it will be popped from the stack

- The DFS is not done because in stack we have some nodes,
When all the stack elements are popped out
i.e., stack should be empty then only we can say that
DFS is terminated.

- Before popping out check the node with its successors
Whether all visited or not. If visited pop out one by one
Until the stack is empty



Output: SABDECG

- **Completeness:** Yes, Complete with in finite state space.
- **Time complexity:** $T(n) = 1 + n^2 + n^3 + \dots + n^m$

Hence $T(n) = O(n^m)$

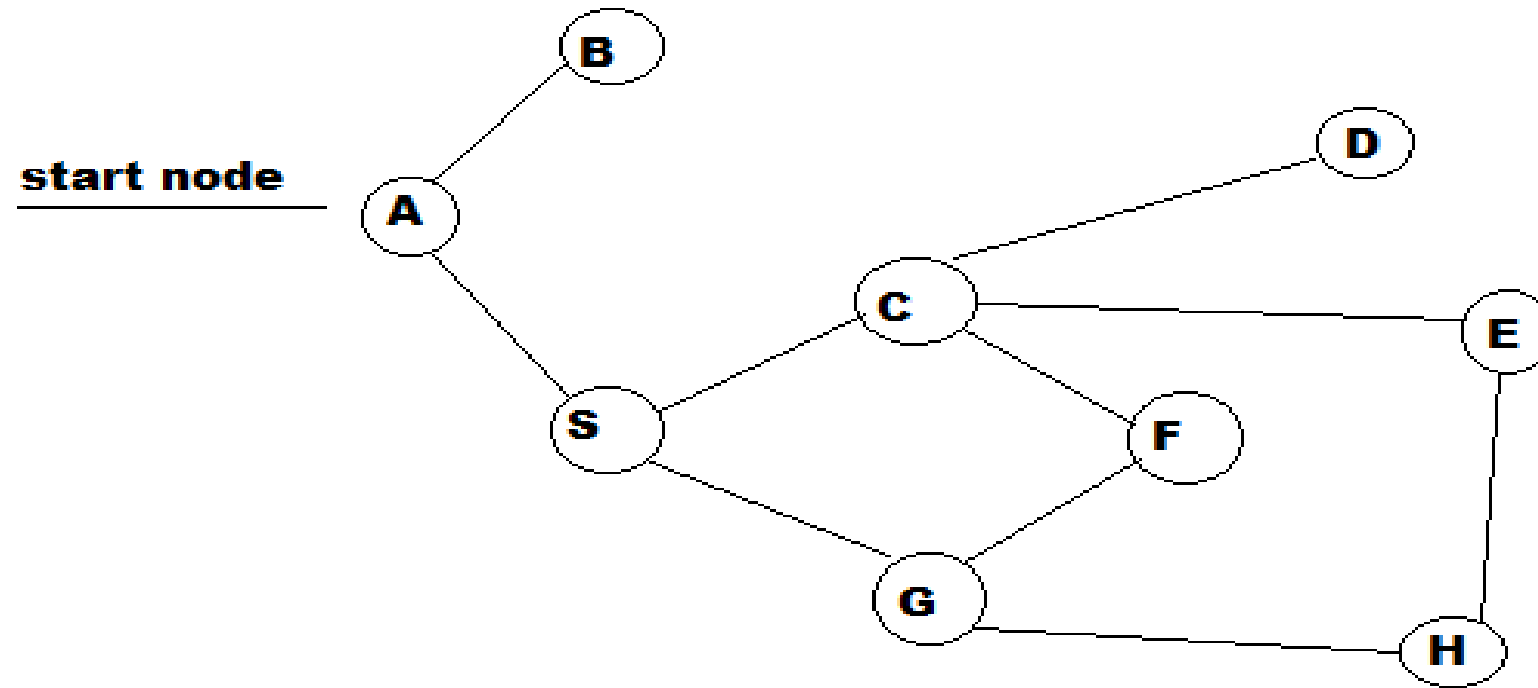
m ----- max depth of any node

- **Space complexity:** $O(b^m)$

b ----- is level of the tree

- **Optimal:** non-optimal (due o infinite loops)

Example 2:



- Start node is “A”
- A-----B,S (possible nodes from A)
- Traverse from A --- B
- Traverse from A --- S (C,G possible nodes from S)
- Traverse from S ---- C (D,E,F possible nodes from C)
- Traverse from C ----- D (no possible nodes, just pop it out)
- Traverse from C ----- E (H possible node from E)
- Traverse from E ----- H (G possible node from H)
- Traverse from H ----- G (F possible node from G)
- Traverse from G ----- F ()

Output : ABSCDEHGF

F
G
H
E
D ----- pop
C
S
B ----- pop
A

stack

F ---- POP
G --- pop
H ---- pop
E ----- pop
C ----- pop
S ----- pop
A ---- pop

DLS: Depth Limited search

- It is similar to DFS with a predetermined limit
- It also solves the problem of infinite path (DFS)
- Always treats the depth has no successor nodes
- DLS can be terminated in 2 conditions:

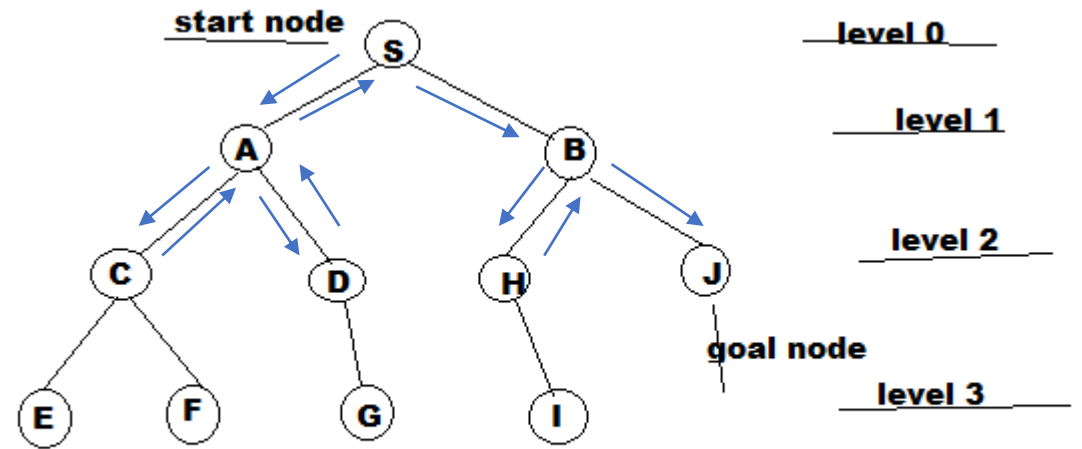
One is during standard failure value

Second one is cut-off failure value

Advantages:

Memory efficient

- Disadvantages:
- Incompleteness and not optimal
- In the given diagram consider my limit is level -2
- If my goal is found or not found in the limit DLS terminates



Time complexity is: $O(b^L)$

L-----limit

Space complexity is: $O(b * L)$

Of course in the above diagram we are following the concept of DFS but not moving beyond the search . i.e., it avoids infinite path in DLS which is n disadvantage of DFS

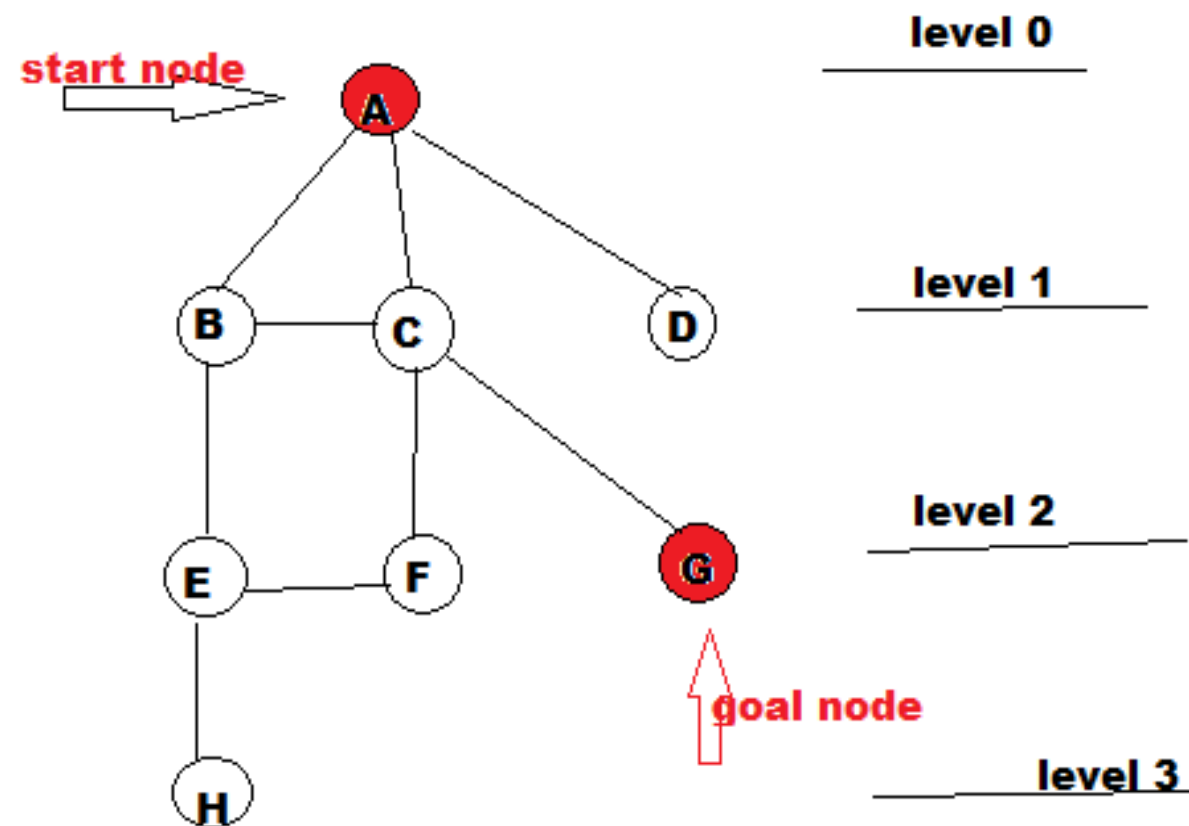
Iterative deepening depth-first search:

- Iterative deepening is the preferred uninformed search method. When the search space is large and the depth of the solution is not known.
- It is often used in combination with DFS, that finds the best depth-limit. This can be done by gradually increasing the limit i.e., First ---0, then 1 and so on..... until a goal is found.
- Iterative deepening combines the benefits of DFS and BFS.

- **Description:**

- . It is a search strategy resulting when you combine BFS and DFS, thus combining the advantages of each strategy, taking the completeness and optimality of BFS and the modest memory requirements of DFS.
- . IDS works by looking for the best search depth d , thus starting with depth limit 0 and make a BFS and if the search failed it increase the depth limit by 1 and try a BFS again with depth 1 and so on – first $d = 0$, then 1 then 2 and so on – until a depth d is reached where a goal is found.

Example



At depth-limit ---0

- “A” root node is visited (open node)
- Iterate the depth level, so level=1
- The possible nodes from ‘A’ are B,C,D
- In level1 “B” (current node) has adjacent node “C” and “E” but “C” is in the same level1 ,”E” is not because it limit exceeds.
- Now the current node is “C”
- In level1 “C” (current node) has adjacent node “B” (already visited) and “F” , “G” is not in the same level1 because it limit exceeds.

Algorithm:

```

    procedure IDDFS(root)
        for depth from 0 to  $\infty$ 
            found  $\leftarrow$  DLS(root, depth)
            if found  $\neq$  null
                return found

procedure DLS(node, depth)
    if depth = 0 and node is a goal
        return node
    else if depth > 0
        foreach child of node
            found  $\leftarrow$  DLS(child, depth-1)
            if found  $\neq$  null
                return found
    return null
```

Performance Measure:

Completeness: IDS is like BFS, is complete when the branching factor b is finite.

Optimality: IDS is also like BFS optimal when the steps are of the same cost.

Time Complexity: $N(\text{IDS}) = (b)d + (d - 1)b^2 + (d - 2)b^3 + \dots + (2)b^{d-1} + (1)b^d = O(b^d)$

If this search were to be done with BFS, the total number of generated nodes in the worst case will be like:

$$N(\text{BFS}) = b + b^2 + b^3 + b^4 + \dots + b^d + (b^{d+1} - b) = O(b^{d+1})$$

If we consider a realistic numbers, and use $b = 10$ and $d = 5$, then number of generated nodes in BFS and IDS will be like

$$N(\text{IDS}) = 50 + 400 + 3000 + 20000 + 100000 = 123450$$

$$N(\text{BFS}) = 10 + 100 + 1000 + 10000 + 100000 + 999990 = 1111100$$

BFS generates like 9 time nodes to those generated with IDS.

- Space Complexity:
 - o IDS is like DFS in its space complexity, taking $O(bd)$ of memory.

Conclusion:

- We can conclude that IDS is a hybrid search strategy between BFS and DFS inheriting their advantages.
- IDS is faster than BFS and DFS.
- It is said that “IDS is the preferred uniformed search method when there is a large search space and the depth of the solution is not known”.

Informed search strategies:

- A Heuristic technique helps in solving problems, even though there is no guarantee that it will never lead in the wrong direction. There are heuristics of every general applicability as well as domain specific. The strategies are general purpose heuristics. In order to use them in a specific domain they are coupled with some domain specific heuristics. There are two major ways in which domain - specific, heuristic information can be incorporated into rule-based search procedure.
- A heuristic function is a function that maps from problem state description to measures desirability, usually represented as number weights. The value of a heuristic function at a given node in the search process gives a good estimate of that node being on the desired path to solution.

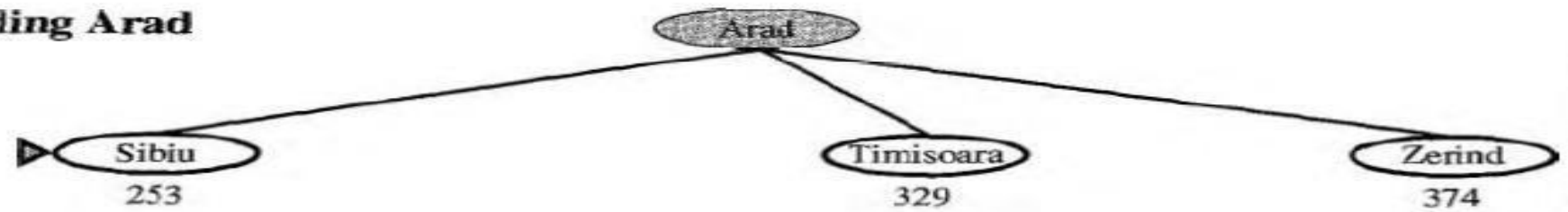
- **Greedy Best First Search**

- Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function:
- Taking the example of Route-finding problems in Romania, the goal is to reach Bucharest starting from the city Arad. We need to know the straight-line distances to Bucharest from various cities
- For example, the initial state is In (Arad), and the straight line distance heuristic h_{SLD} (In (Arad)) is found to be 366. Using the straight-line distance heuristic h_{SLD} , the goal state can be reached faster.

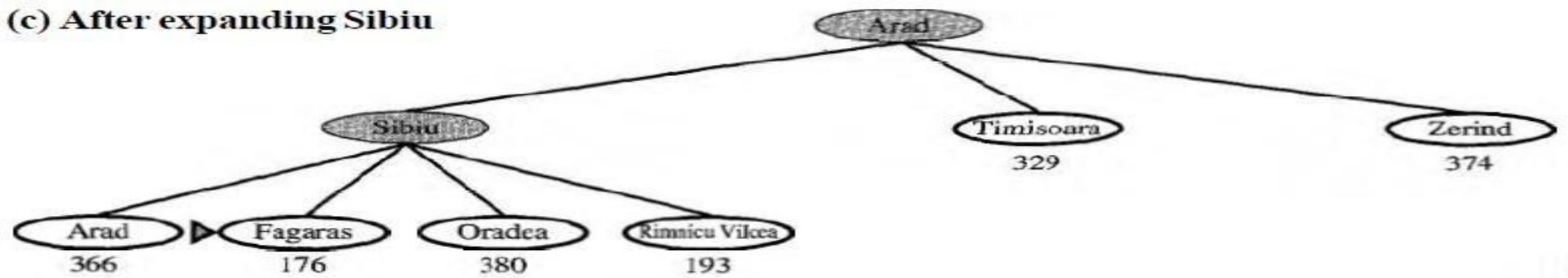
(a) The initial state



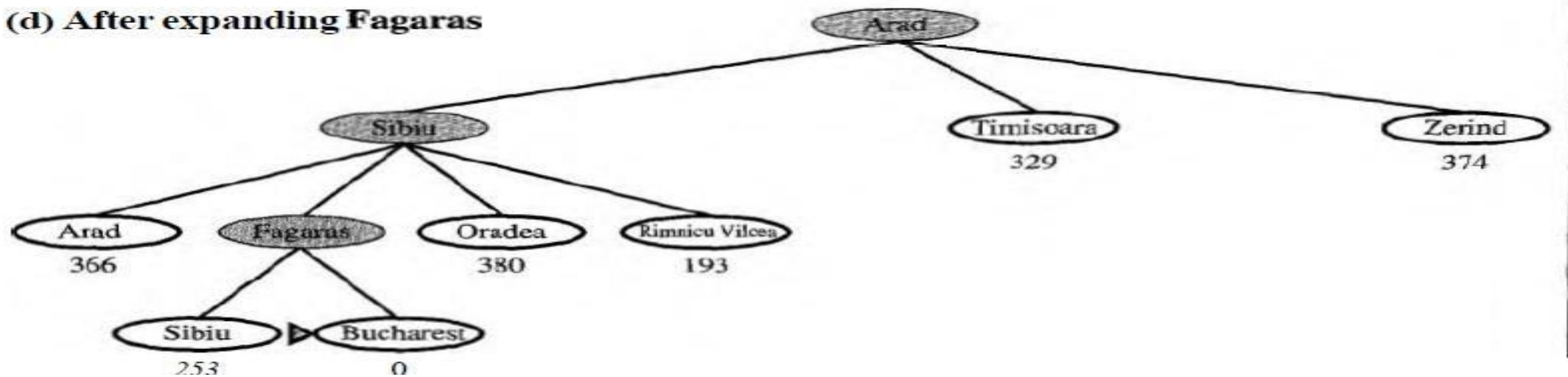
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



Evaluation Criterion of Greedy Search

Complete: NO [can get stuck in loops, e.g., Complete in finite space with repeated-state checking]

Time Complexity: $O(b^m)$ [but a good heuristic can give dramatic improvement]

Space Complexity: $O(b^m)$ [keeps all nodes in memory]

Optimal: NO

Greedy best-first search is not optimal, and it is incomplete. The worst-case time and space complexity is $O(b^m)$, where m is the maximum depth of the search space.

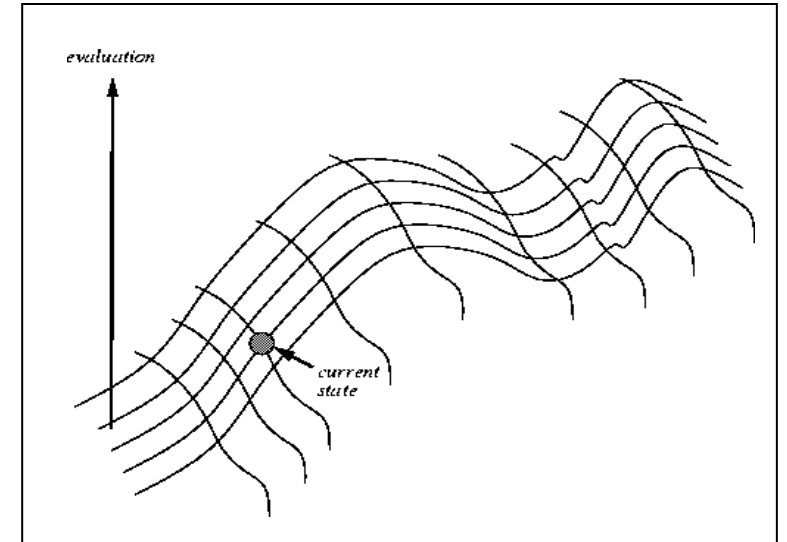
- **HILL CLIMBING PROCEDURE:**

- *Hill Climbing Algorithm*

- We will assume we are trying to maximize a function. That is, we are trying to find a point in the search space that is better than all the others. And by "better" we mean that the evaluation is higher. We might also say that the solution is of better quality than all the others.
- The idea behind hill climbing is as follows.

1. Pick a random point in the search space.
2. Consider all the neighbors of the current state.
3. Choose the neighbor with the best quality and move to that state.
4. Repeat 2 thru 4 until all the neighboring states are of lower quality.
5. Return the current state as the solution state.

- **Algorithm:**
 - **Function** HILL-CLIMBING(*Problem*) **returns** a solution state
Inputs: *Problem*, problem
 - Local variables: *Current*, a node
 - *Next*, a node
 - *Current* = MAKE-NODE(INITIAL-STATE[*Problem*])
 - **Loop do**
 - *Next* = a highest-valued successor of *Current*
 - **If** VALUE[*Next*] < VALUE[*Current*] **then return** *Current*
 - Current* = *Next*
 - **End**



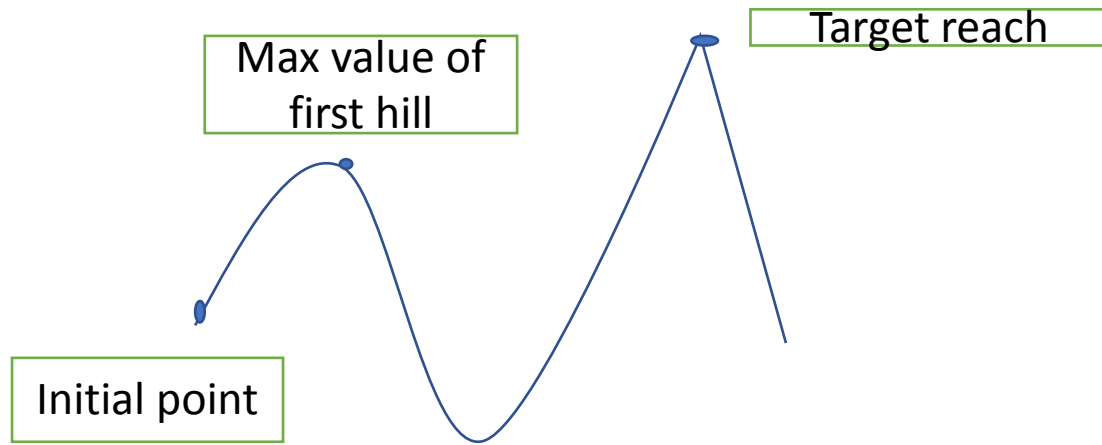
- You should note that this algorithm does not maintain a search tree. It only returns a final solution. Also, if two neighbors have the same evaluation and they are both the best quality, then the algorithm will choose between them at random.

Problems with Hill Climbing

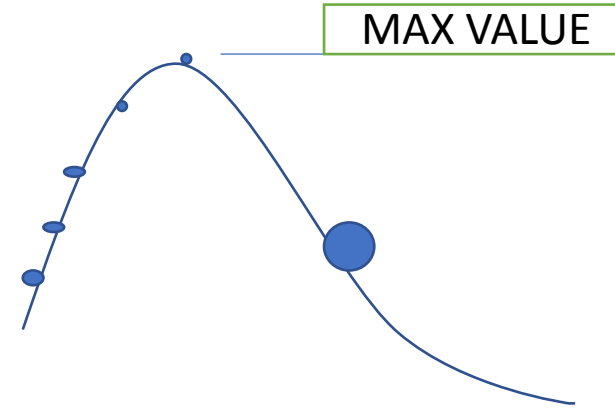
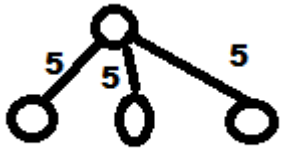
- The main problem with hill climbing (which is also sometimes called *gradient descent*) is that we are
- Not guaranteed to find the best solution. In fact, we are not offered any guarantees about the solution. It could be abysmally bad.
- You can see that we will eventually reach a state that has no better neighbours but there are better solutions elsewhere in the search space. The problem we have just described is called a *local maxima*.

- Example for hill climbing

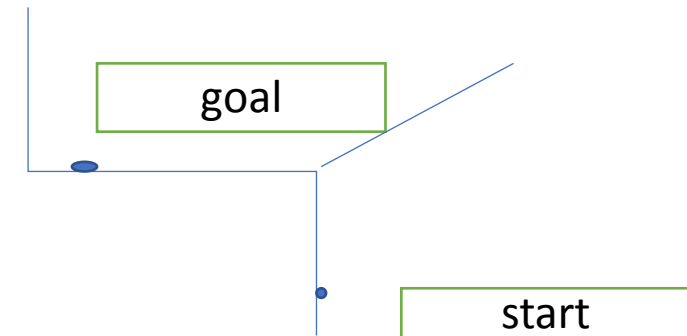
1. Local max problem



2. plateau/Flat max



3. Ridge



Best First Search:

A combination of depth first and breadth first searches.

Depth first is good because a solution can be found without computing all nodes and breadth first is good because it does not get trapped in dead ends.

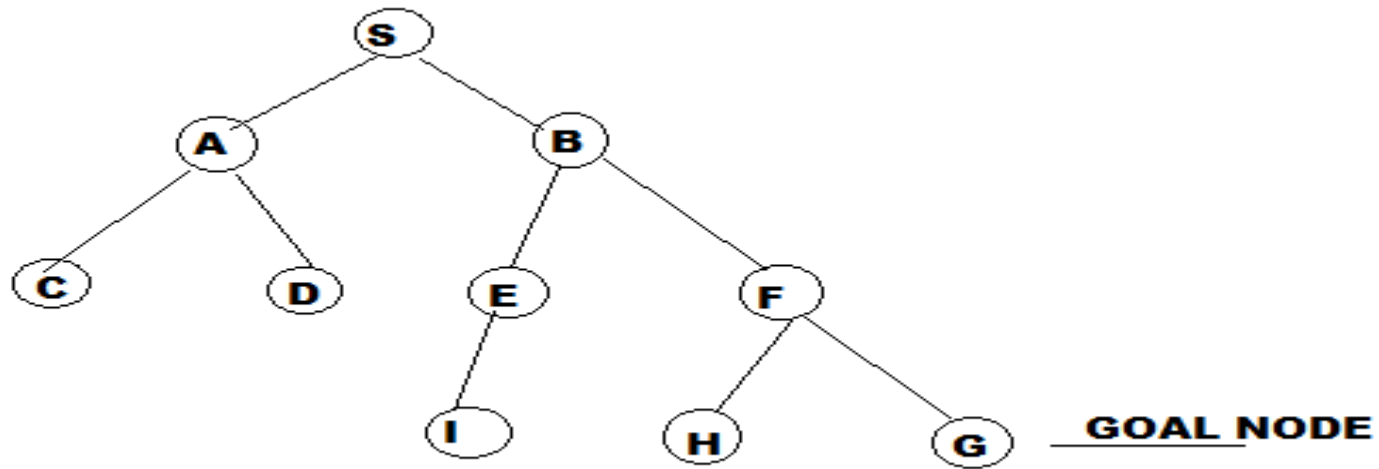
The best first search allows us to switch between paths thus gaining the benefit of both approaches. At each step the most promising node is chosen. If one of the nodes chosen generates nodes that are less promising it is possible to choose another at the same level and in effect the search changes from depth to breadth. If on analysis these are no better than this previously unexpanded node and branch is not forgotten and the search method reverts to the

OPEN is a priority queue of nodes that have been evaluated by the heuristic function but which have not yet been expanded into successors. The most promising nodes are at the front.

CLOSED are nodes that have already been generated and these nodes must be stored because a graph is being used in preference to a tree.

- **Algorithm:**

- Start with OPEN holding the initial state
- Until a goal is found or there are no nodes left on open do.
 - Pick the best node on OPEN
 - Generate its successors
 - For each successor Do
 - If it has not been generated before ,evaluate it ,add it to OPEN and record its parent
 - If it has been generated before change the parent if this new path is better and in that case update the cost of getting to any successor nodes.
 - If a goal is found or no more nodes left in OPEN, quit, else return to 2.



NODE	H(n)
S	13
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
G	0

- OPEN ----- nodes which are not used
- CLOSE ----- which are already evaluated

- Initial node ----s
- Open[A,B], close[s]
- Open [A], CLOSE[S,B]
- OPEN[A,E], CLOSE [S,B,F]
- OPEN[A,E,H], CLOSE[S,B,F,G]

• OUTPUT: S---B---F----G

Time complexity=space complexity $O(bm)$

M---- max depth of search space

Properties:

1. It is not optimal.
2. It is incomplete because it can start down an infinite path and never return to try other possibilities.
3. The worst-case time complexity for greedy search is $O(bm)$, where m is the maximum depth of the search space.
4. Because greedy search retains all nodes in memory, its space complexity is the same as its time complexity

A* search algorithm

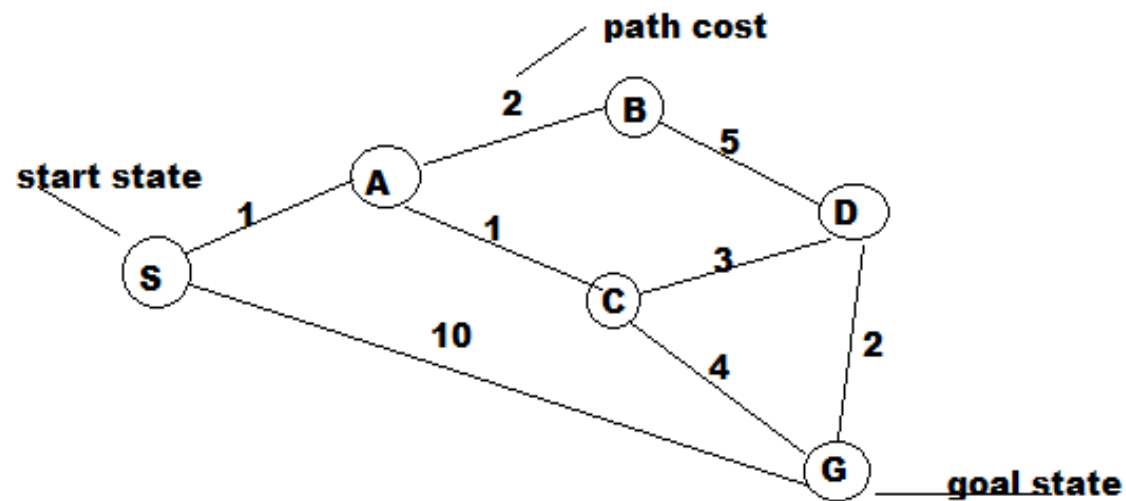
- The Best First algorithm is a simplified form of the A* algorithm.
- The A* search algorithm (pronounced "Ay-star") is a tree search algorithm that finds a path from a given initial node to a given goal node (or one passing a given goal test). It employs a "heuristic estimate" which ranks each node by an estimate of the best route that goes through that node. It visits the nodes in order of this heuristic estimate.
- Similar to greedy best-first search but is more accurate because A* takes into account the nodes that have already been traversed.

- A* search algorithm finds the shortest path through the search space using the heuristic function i.e., $h(n)$
- It uses $h(n)$ and cost to reach the node 'n' from the start state i.e., $g(n)$
- To find the value of the particular path then,

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

- This algorithm expands less search tree and provides optimal results faster
- It is similar to uniform cost search (gives path cost from one node to another node i.e., uses $g(n)$)
- A* uses search heuristic as well as the cost to reach the node. So combine both cost as:

- $F(n) = g(n) + h(n)$
 - $F(n)$ is the function which is also called as fitness number
- $F(n)$ = Estimated cost of cheapest solution
- $g(n)$ = cost to reach node 'n' from start state
- $h(n)$ = cost to reach from node 'n' to goal node



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

Algorithm:

1. **Initialize** : Set $OPEN = (S)$; $CLOSED = ()$
 $g(s) = 0$, $f(s) = h(s)$
2. **Fail** : If $OPEN = ()$, Terminate and fail.
3. **Select** : select the minimum cost state, n , from $OPEN$,
save n in $CLOSED$
4. **Terminate** : If $n \in G$, Terminate with success and return $f(n)$
5. **Expand** : for each successor, m , of n
 - a) If $m \in [OPEN \cup CLOSED]$
Set $g(m) = g(n) + c(n, m)$
Set $f(m) = g(m) + h(m)$
Insert m in $OPEN$
 - b) If $m \in [OPEN \cup CLOSED]$ |

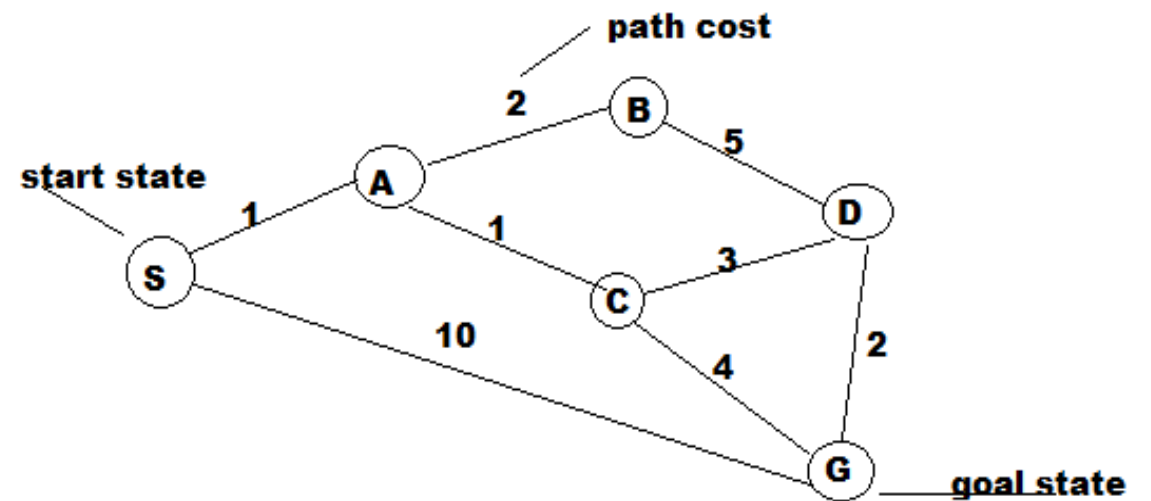
Set $g(m) = \min \{ \underline{g(m)} , g(n) + c(n, m) \}$

Set $f(m) = g(m) + h(m)$

If $f(m)$ has decreased and $m \in \text{CLOSED}$

Move m to OPEN.

- Initial state 'S'
- S-----A = $F(n) = g(n) + h(n)$
 $F(n) = 1 + 3 = 4$
- S-----G = $F(n) = g(n) + h(n)$
 $F(n) = 10 + 0 = 10$



- From “s” we have received $F(n) = 4$ & 10
- Compare 2 values and consider the lowest value as the current path and put highest value in hold state.
- Like wise we have to repeat till the goal state reach and stops searching
- After reaching the goal state check the hold state values are greater than the goal state value or not, if yes then consider that lowest path cost to reach the goal

- $S \rightarrow A \rightarrow B = F(n) = g(n) + h(n)$
 $F(n) = 3 + 4 = 7$

- $S \rightarrow A \rightarrow C = F(n) = g(n) + h(n)$
 $F(n) = 2 + 2 = 4$

- $S \rightarrow A \rightarrow C \rightarrow D = F(n) = g(n) + h(n)$
 $F(n) = 5 + 6 = 11$

- $S \rightarrow A \rightarrow C \rightarrow G = F(n) = g(n) + h(n)$
 $F(n) = 6 + 0 = 6$

The cost is 6 to reach the goal

- Output: S---A---C---G

- Advantages:

Best algorithm than other

Optimal & complete

Can also solve complex problems

- Disadvantages:

Always not guaranteed to produce shortest path

It is not practical for various large-scale problems

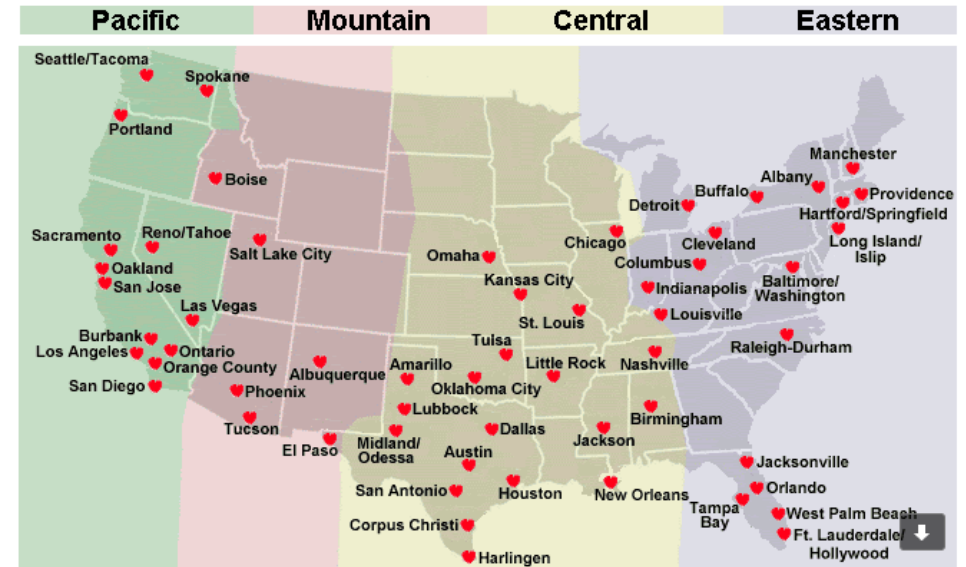
UNIT - II

- Advanced Search: Constructing Search Trees, Stochastic Search, A* Search Implementation, Minimax Search, Alpha-Beta Pruning
- Basic Knowledge Representation and Reasoning: Propositional Logic, First-Order Logic, Forward Chaining and Backward Chaining, Introduction to Probabilistic Reasoning, Bayes Theorem

Constructing Search Trees

The construction of search tree will find a potential solution to unknown questions by looking at the available data in an organized manner, one piece (or "node") at a time.

- Suppose one is writing a program for an airline to find a way for a customer to get from City A to City B given a list of all of the possible flights between any two cities. The program will need to go through the list and determine which flight (or connecting series of flights) that the customer should take. This is implemented using a search tree. City A (where the customer currently is located) will be the initial piece of information, or "initial node." The search tree will consist of this initial node at the top of the tree. Next, the program would look at all of the flights beginning with City A. These flights would show cities that the customer could reach in just one flight, and so these cities (each city a node itself) are on the next level, below the initial node. Now, each of these cities has flights as well, which would be on the third level, and the tree would continue so forth.



- Choosing a search strategy (i.e. Breadth-first or Depth-first) will tell the program how to progress down the search tree, or hierarchy of nodes that has been built from the information. There are clearly many different ways that one can get from City A to City B, so the search strategy is the key to determining which path will be selected.
- Search trees, with search strategies, can also be found in finding solutions in puzzle-solving or simple games such as "The Eight Queens Problem" where one tries to place eight queens in locations on a chessboard such that none of them can move to the space of another.

Therefore:

- **Create search trees to solve Artificial Intelligence questions. Use each piece of information as a node and construct a hierarchy of nodes based on how the individual pieces of information connect together so that one can go along a path through a series of nodes in order to get from one to another.**
- Use a SEARCH STRATEGIES to determine the way of traveling down the hierarchy of the search tree. HEURISTICS can be used to give weights or "costs" to the different paths between nodes and help in the strategy decisions of which path to ultimately choose

Stochastic Search

- Adversarial search, or game-tree search, is a technique for analyzing an adversarial game in order to try to determine who can win the game and what moves the players should make in order to win. Adversarial search is one of the oldest topics in Artificial Intelligence. The original ideas for adversarial search were developed by Shannon in 1950 and independently by Turing in 1951, in the context of the game of chess—and their ideas still form the basis for the techniques used today

Game playing:

- Humans have always been fascinated with games.
- Games challenge our ability to think.
- simple games:
Tic-Tac-Toe, Nim, Kalah, 8 Puzzle
- More complex games:
checkers, chess, bridge, Go

Computers playing games

- Checkers – beat the human world champion
- Chess – many very good programs. Deep Blue beat the human world champion
- Backgammon – at the level of a top few humans
- Othello – play much better than the best human players
- Go – not good

- Good example for Stochastic search is Game playing.

- It is an interesting topic because:

one require intelligence

Logical thinking

Rational mind

Searching algorithms

- At the same time, we don't know what the opponent choice and ideas
- It is a multi agent environment
- Game playing works on consistent values (utility factor)
- It is a space search so we use BFS and DFS
- In this depth is called **ply**
- How much the big is game tree the same amount of search is done

Game Theory

- Two players
- The players alternate
- Focus on games of perfect information
- Zero sum game

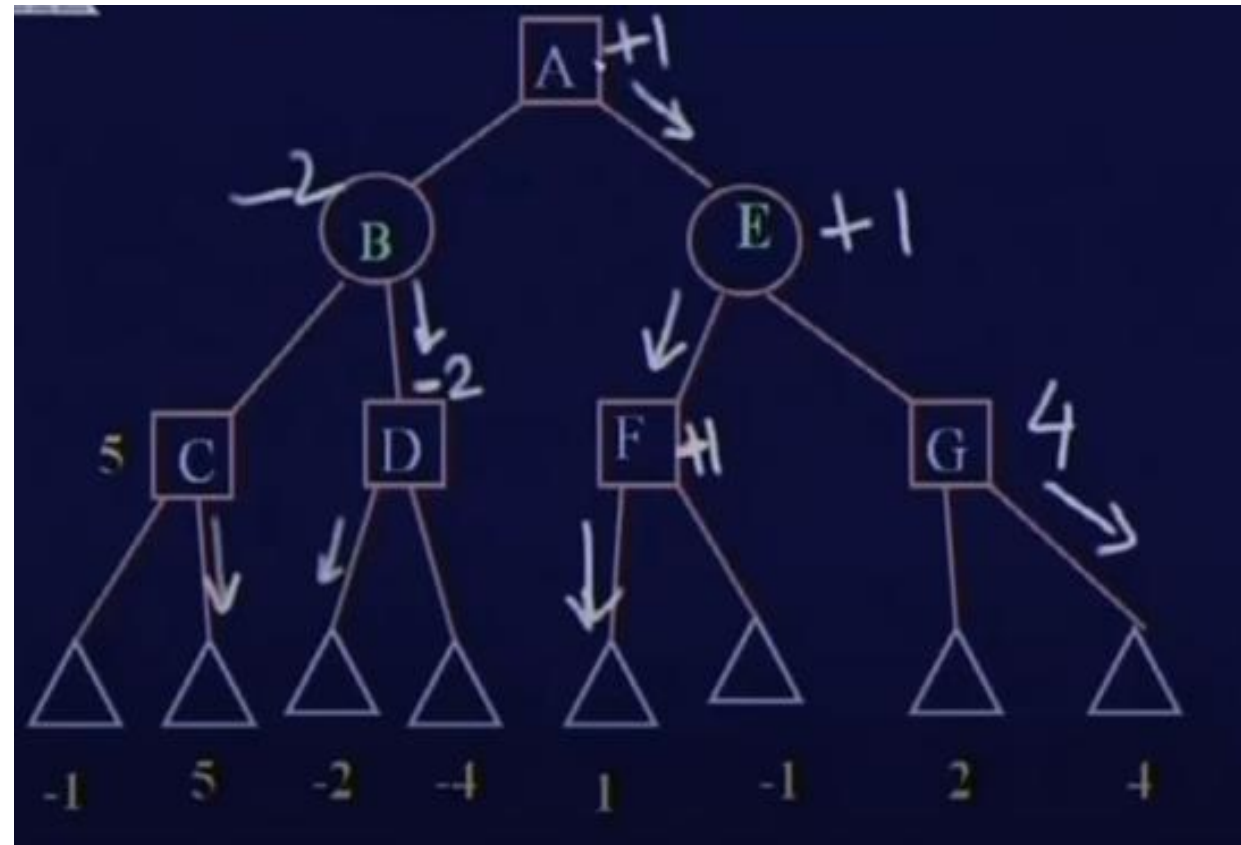
2-Person Games:

- **Players:** We call them Max and Min.
- **Initial State:** Includes board position and whose turn it is.
- **Operators:** These correspond to legal moves.
- **Terminal Test:** A test applied to a board position which determines whether the game is over. In chess, for example, this would be a checkmate or stalemate situation.
- **Utility Function:** A function which assigns a numeric value to a terminal state. For example, in chess the outcome is win (+1), lose (-1) or draw (0). Note that by convention, we always measure utility relative to Max.

Basic Strategy

- Grow a search tree
- Only one player can move at each turn
- Assume we can assign a payoff to each final position – called a utility.
- We can propagate values backwards from final positions
- Assume the opponent always makes moves worst for us
- Pick best moves on own turn

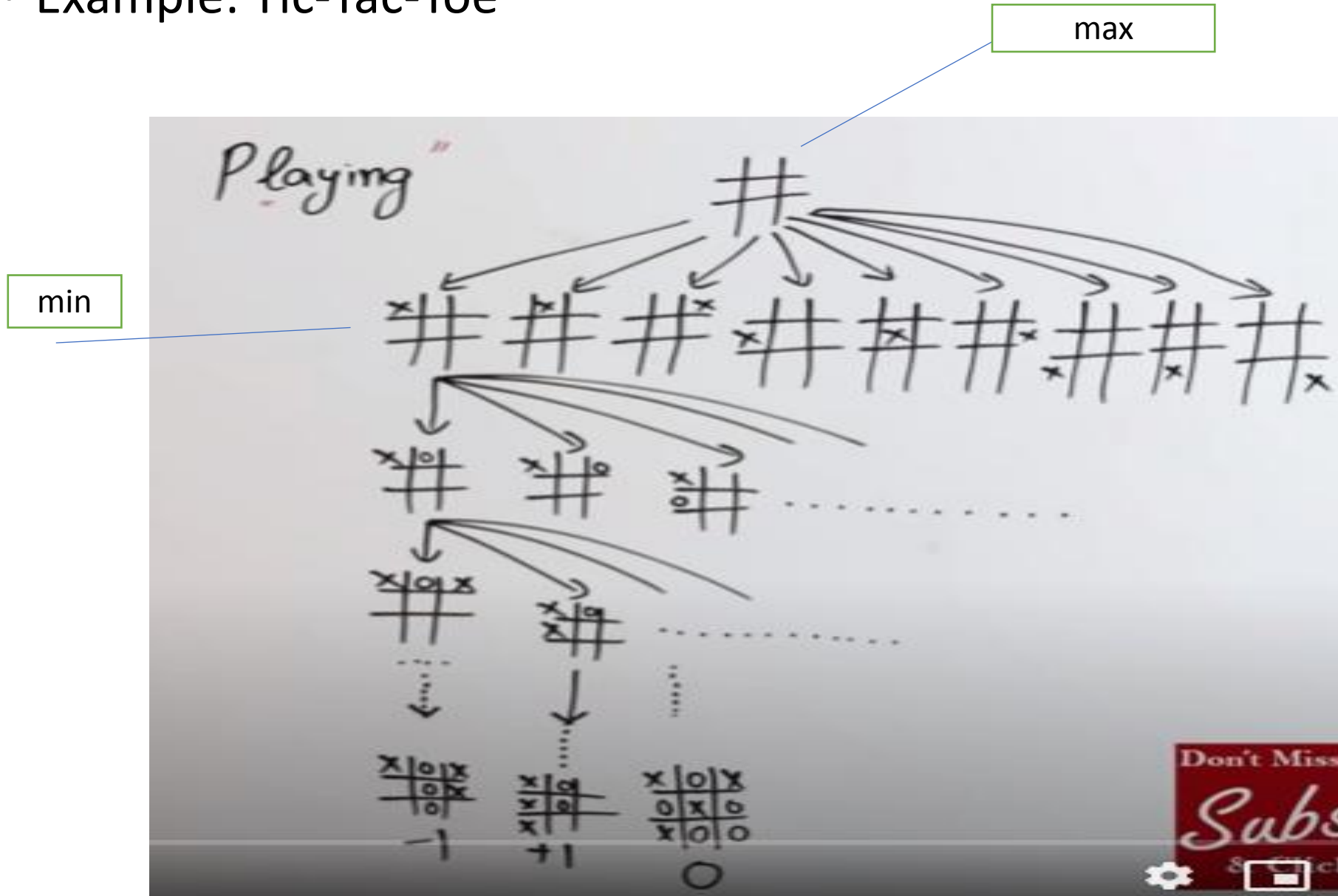
- We have utility values -1,5,-2,-4,1,-1,2,4
 - Now it is our turn to move
- we choose best value or max value
- The next turn is opponent will always choose which is worst for Us (i.e 1st player)



Game tree

- Successive nodes represent positions where different players must move. We call the nodes MAX or MIN nodes depending of who is the player that must move at that node.
- A game tree could be infinite.
- The ply of a node is the number of moves needed to reach that node (i.e. arcs from the root of the tree). The *ply of a tree* is the maximum of the plies of its nodes.

- Example: Tic-Tac-Toe



Brute-Force Search

- We begin considering a purely brute-force approach to game playing.
- This will only be feasible for small games, but provides a basis for further discussions.

Minimax

- Minimax theorem - Every two-person zero-sum game is a forced win for one player, or a forced draw for either player, in principle these optimal minimax strategies can be computed.
- Performing this algorithm on tic-tac-toe results in the root being labeled a draw.

- Game playing majorly works on 2 algorithms:

1 Minimax search

2 alpha Beta pruning

Minimax search

mainly works with the help of backtracking algorithm

Best move strategy is used

Max will try to maximize (best move)

Min will try to minimize (Worst move)

Minimax Search

1. Generate the whole game tree.
2. Apply the utility function to leaf nodes to get their values.
3. Use the utility of nodes at level n to derive the utility of nodes at level $n-1$.
4. Continue backing up values towards the root (one layer at a time).
5. Eventually the backed up values reach the top of the tree, at which point Max chooses the move that yields the highest value. This is called the minimax decision because it maximises the utility for Max on the assumption that Min will play perfectly to minimise it.

function MINIMAX(N)

begin

if N is a leaf then

return the estimated score of this leaf

else

Let N_1, N_2, \dots, N_m be the successors of N;

if N is a Min node then

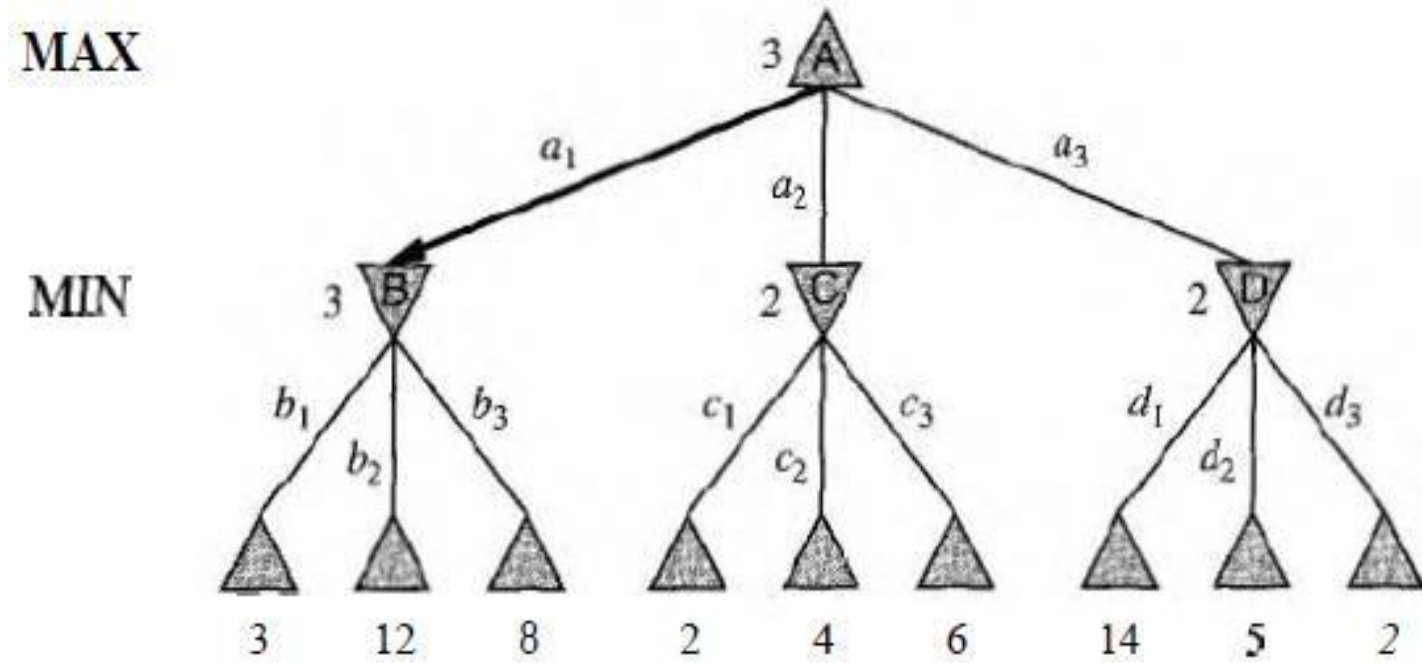
return $\min\{\text{MINIMAX}(N_1), \dots, \text{MINIMAX}(N_m)\}$

else

return $\max\{\text{MINIMAX}(N_1), \dots, \text{MINIMAX}(N_m)\}$

end

Basic Example 1



Example 2:

- Let us assume $\text{Max} = -\infty$ and $\text{Min} = \infty$

Max move

$(-\infty, 3) \rightarrow 3$

$(3, 2) \rightarrow 2$

$(2, -1) \rightarrow -1$

$(-\infty, 1) \rightarrow 1$

$(1, 0) \rightarrow 0$

$(0, 2) \rightarrow 0$

$(-\infty, 5) \rightarrow 5$

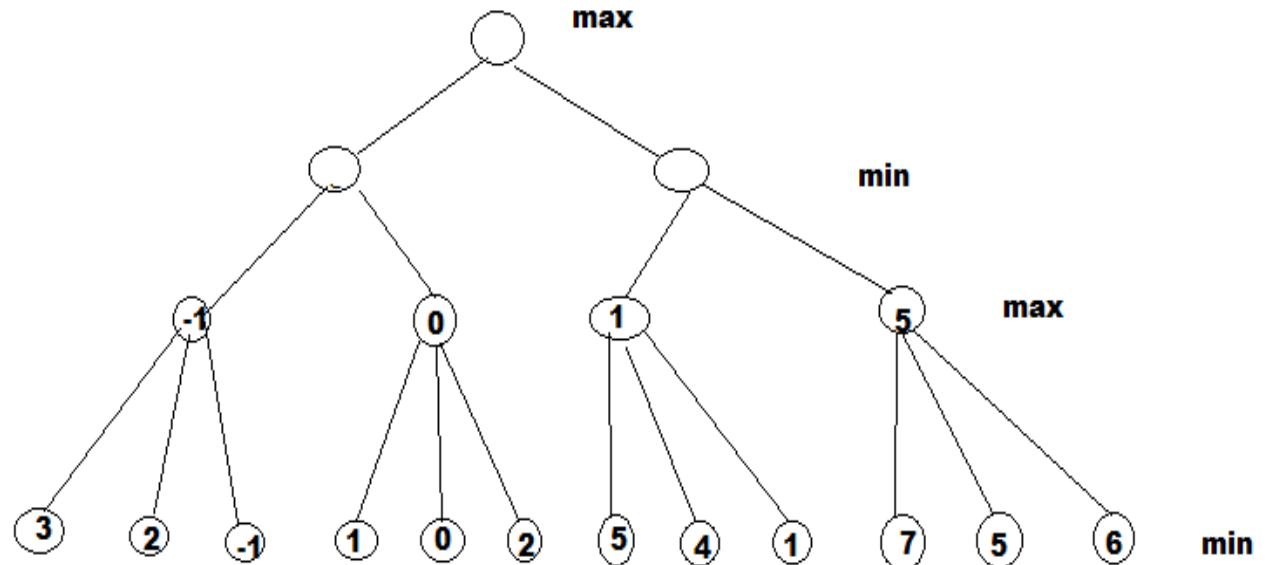
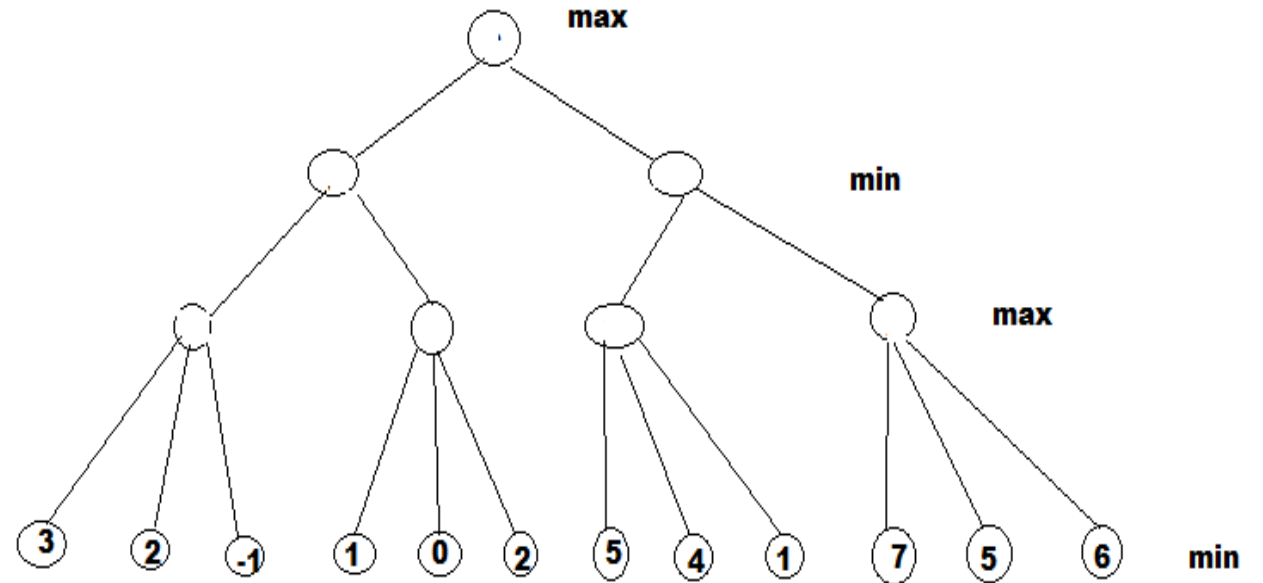
$(5, 4) \rightarrow 4$

$(4, 1) \rightarrow 1$

$(-\infty, 7) \rightarrow \text{min } 7$

$(7, 5) \rightarrow \text{min } 5$

$(5, 6) \rightarrow 5$



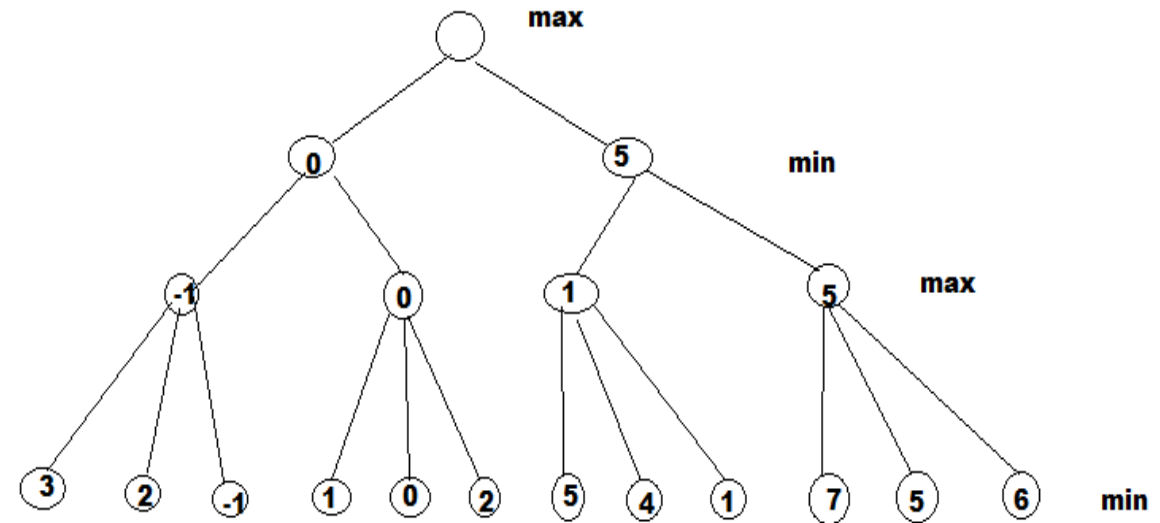
Min move

$(-\infty, -1) \text{ --- } -1$

$(-1, 0) \text{ ---- } 0$

$(-\infty, 1) \text{ ---- } 1$

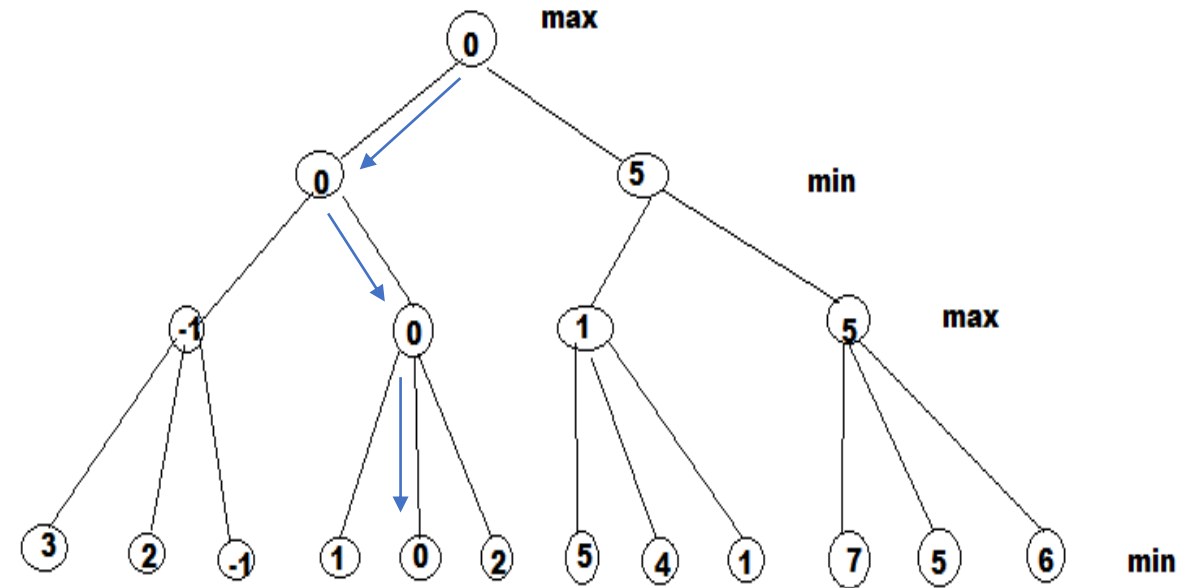
$(1, 5) \text{ ----- } 5$



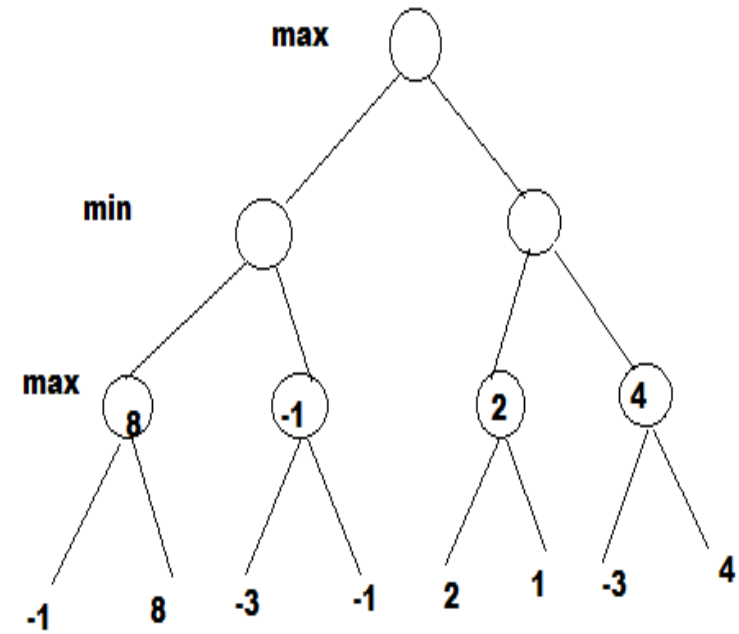
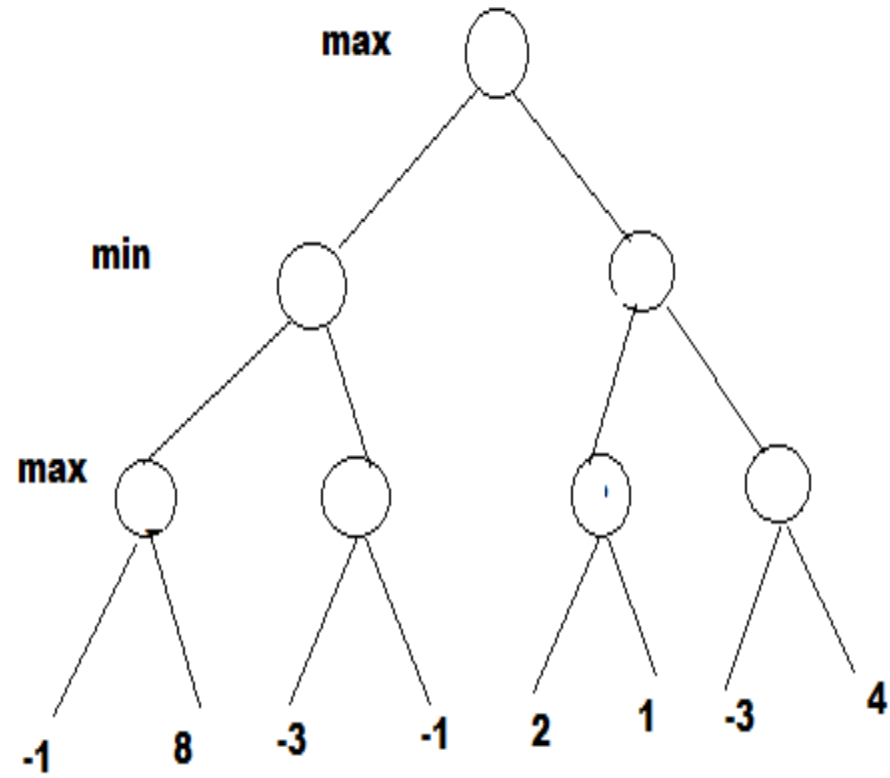
Max Move

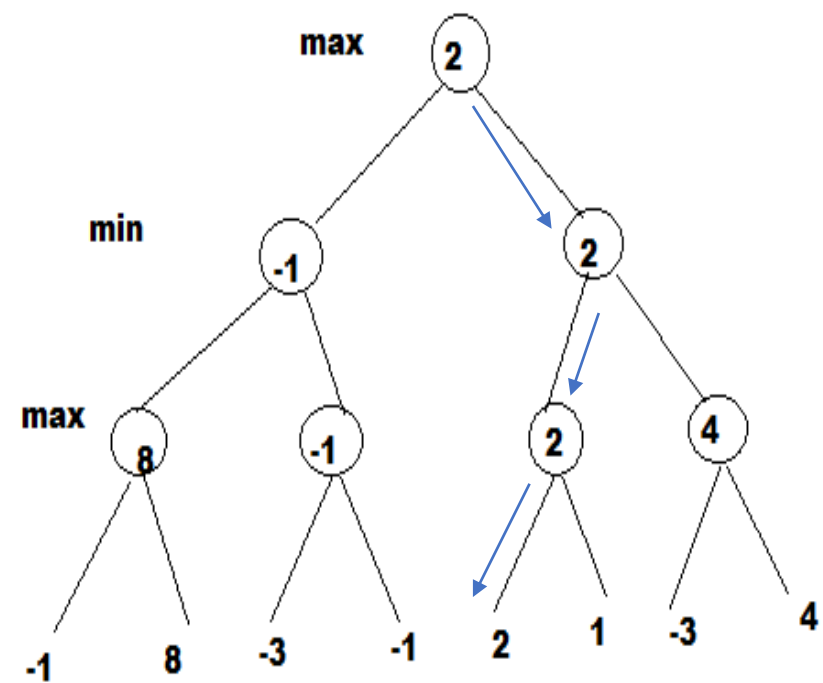
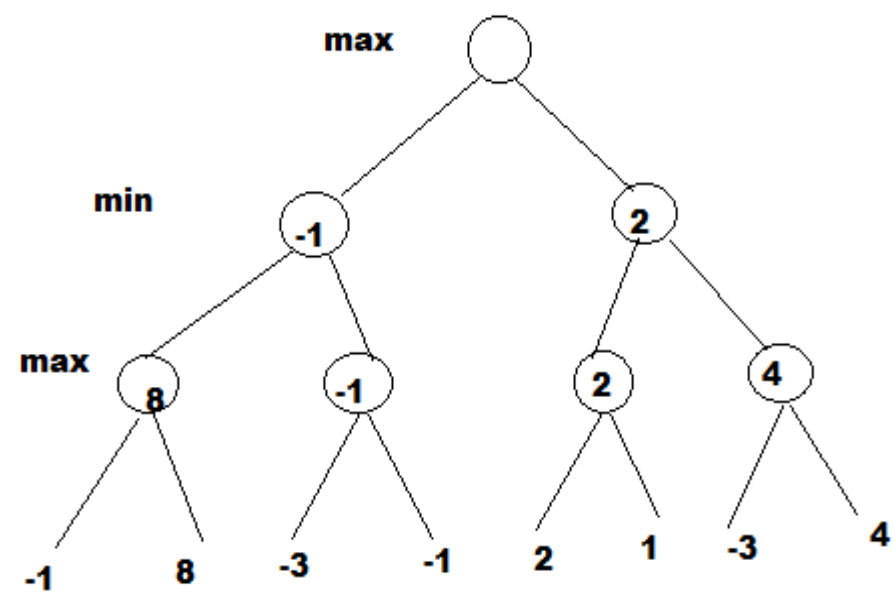
$(-\infty, 0) \text{ ---- } 0$

$(0, 5) \text{ ----- } 0$



Example:3





Example for Tic-Tac-Toe (using min-max alg)

draw = 0
win = 1

Initial state

Consider,
max = X
min = O

max state

min state

max

(draw)

win

win

win

win (draw)

0	0	x
x		0
		x

} Initial state

Consider,
 $\text{max} = X$
 $\text{min} = 0$

max
state

min
Stake

max 10

0	0	x
x	x	0
0	x	x

(draw)

৩১৭

WZT

৯১৭

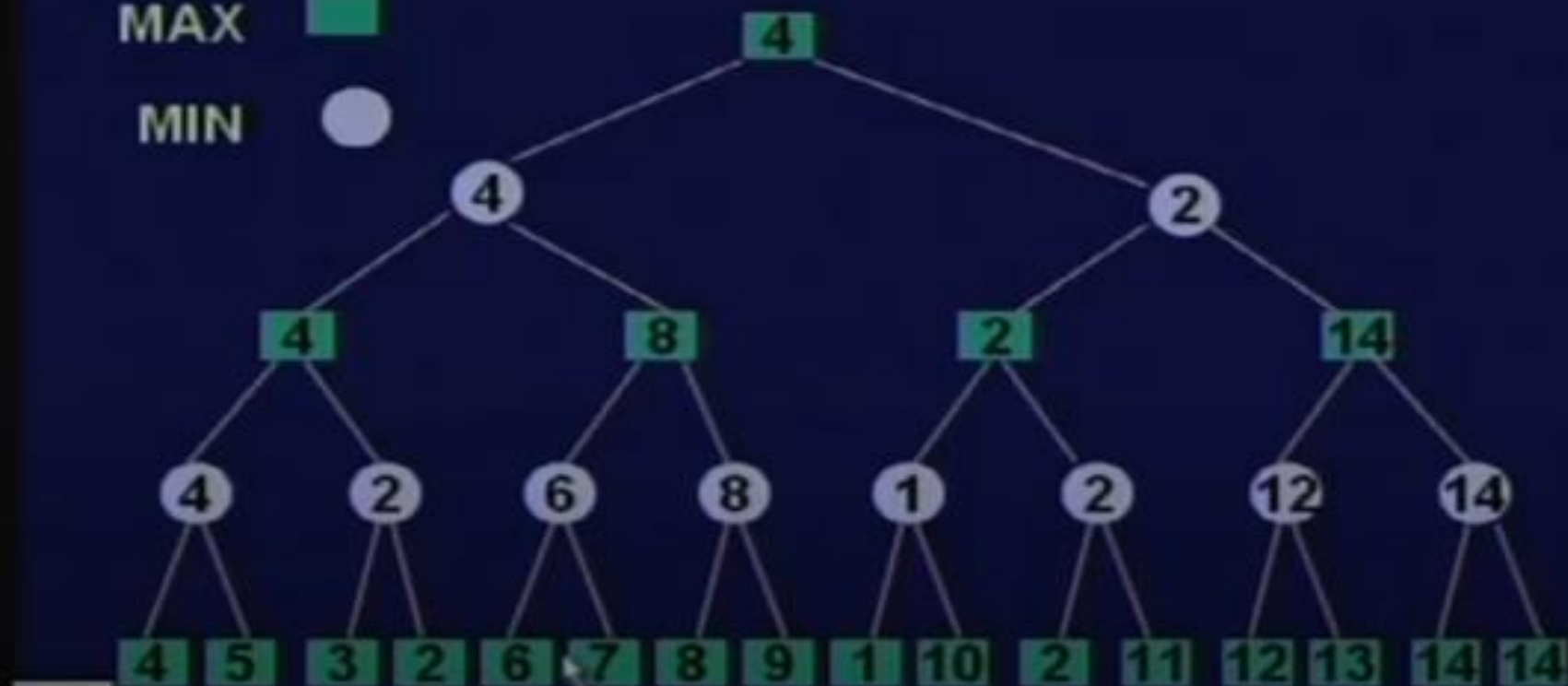
win (draw)

Minimax Tree example

MAX



MIN



- Properties of minimax:
 - Complete : Yes (if tree is finite)
 - Optimal : Yes (against an optimal opponent)
 - Time complexity : $O(bm)$
 - Space complexity : $O(bm)$ (depth-first exploration)
 - For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
 - exact solution completely infeasible.

Limitations

- – Not always feasible to traverse entire tree
- – Time limitations

Heuristic Minimax Search

- We search to some fixed cutoff (search horizon).
- We estimate the merit of positions at the frontier by some heuristic static evaluation function when the final outcome has not yet been determined.
- We propagate these values up.

Modified Heuristic MiniMax function

```
function MINIMAX(n)
begin
  if terminal-test (n) then
    return payoff (n)
  else
    if n is a MAX node then  $v = \infty$  else  $v = -\infty$ 
    for all  $m \in \text{successors}(n)$ 
      if n is a MAX node
         $v = \max(v, \text{MINIMAX}(m))$ 
      else  $v = \min(v, \text{MINIMAX}(m))$ 
    end
```

Heuristic Evaluation Functions

- Given a heuristic static evaluation function, it is straightforward to write a program to play a game.
- From any given position, we simply generate all the legal moves, apply our static evaluator to the position resulting from each move, and then move to the position with the largest or smallest evaluation, depending if we are MIN/MAX

Alpha-Beta pruning

- In minimax algorithm we explore all nodes in the given search tree, exploration is more
- So now we can develop a algorithm where we can reduce the expansion of nodes that is alpha beta pruning
- In this **pruning** means we are going to ignore the next branches.
- It is a method of cut off search by exploring less no of nodes.
- In simple words alpha beta pruning is defined as cutting the parts of the tree which we don't need.

Alpha-Beta search

- Similar to branch and bound
- We cut off search when we cannot do better than the best so far.
- To implement this we will manipulate alpha and beta values, and store them on internal nodes in the search tree

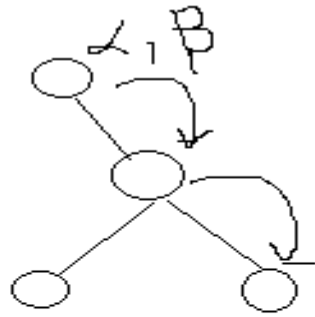
Alpha values

- At a Max node we will store an alpha value
 - A *lower bound* on the exact minimax score
 - the true value might be $\geq \alpha$
 - if we know Min can choose moves with score $< \alpha$
 - then Min will never choose to let Max go to a node where the score will be α or more

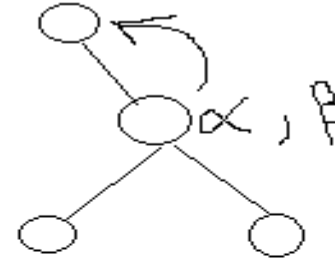
Beta values

- At a Min node
 - the beta value is a lower bound on the exact minimax score
 - the true value might be $\leq \beta$
 - if we know Max can choose moves with score $> \beta$
 - then Max will never choose to let Min go to a node where the score will be β or less

- Basically 2 values
- α and β
- Alpha= max value, in worst case alpha =-inf
- Beta=min value, in worst case beta=inf
- At max level always the value of **alpha will be changed** and beta value remains constant
- At min level always the value of **beta will be changed** and alpha value remains constant
- Never take any value to the upside of the tree i.e.,



**correct way to
values of alpha and
beta**

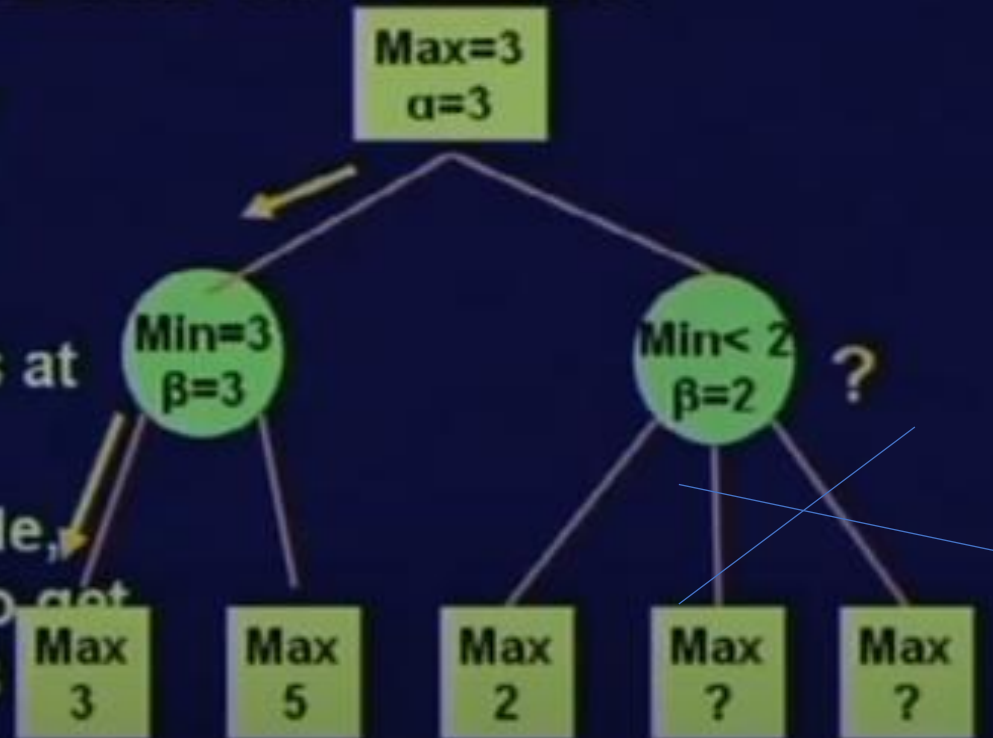


**wrong way to take
values of alpha and
beta**

- Always need to check the condition at each node whether
- Alpha \geq Beta
- If it is greater then we can prone the next path to travel so that we can reduce the expansion of nodes and should know the alpha cut-off and beta cut-off

Alpha Beta in Action

- Why can we cut off search?
- $\beta = 2 < \alpha = 3$
where the α value is at an ancestor node
- At the ancestor node, Max had a choice to get a score of at least 3
- Max is not going to move right to let Min guarantee a score of 2



Alpha and Beta values

- At Max node, if an ancestor Min node has $\beta < \alpha$
 - Min's best play must never let Max move to this node
 - therefore this node is irrelevant
 - if $\beta = \alpha$, Min can do as well without letting Max get here
 - so again we need not continue

Alpha-Beta Pruning Rule

- Search can be discontinued at a node if:
 - ① – It is a Max node and
 - the alpha value is \geq the beta of any Min ancestor
 - this is *beta cutoff*
 - Or it is a Min node and
 - the beta value is \leq the alpha of any Max ancestor
 - this is *alpha cutoff*

Calculating Alpha-Beta values

- Alpha-Beta calculations similar to Minimax, but the pruning rule cuts down search
- Final backed up value of node
 - might be the minimax value
 - or might be an approximation where search cut off
 - *less* than the minimax value at a Max node
 - *more* than the minimax value at a Min node
 - we don't need to know the true value

MAX

MAXIMIN (n, alpha, beta)

IF n is at the search depth, RETURN $V(n)$

FOR each child m of n

 value = MINIMAX (m, alpha, beta)

 IF value > alpha , alpha = value

 IF alpha >= beta , return alpha

RETURN alpha

MIN

MINIMAX (n, alpha, beta)

IF n is at the search depth, RETURN $V(n)$

FOR each child m of n

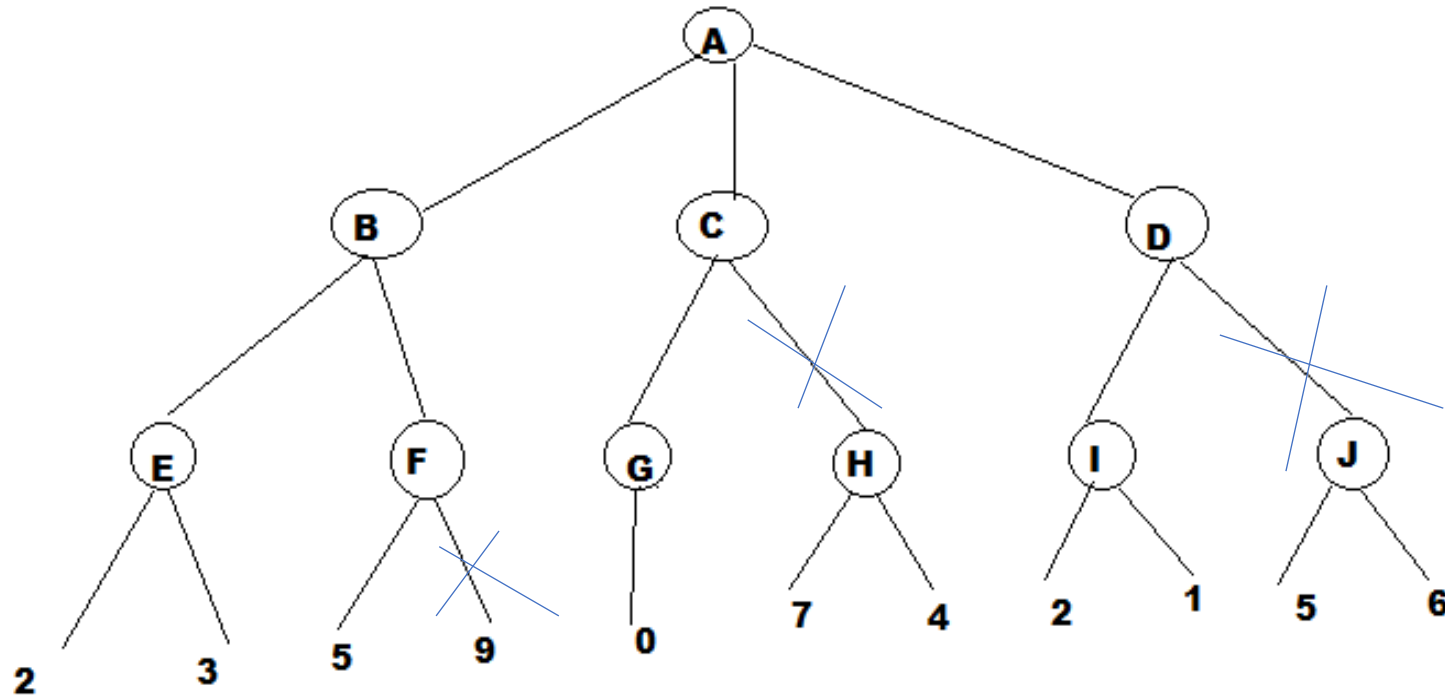
 value = MAXIMIN (m, alpha, beta)

 IF value < beta , beta = value

 IF beta <= alpha, return ~~alpha~~ beta

RETURN beta

Example for alpha beta pruning



max

min

max

max α

min β

α $-\infty$
 β ∞

- After terminal nodes or leaf nodes we have max level

- **At max:**

Initial value of alpha = -inf (changes) and beta = inf (remains constant)

$\text{Max}(-\text{inf}, 2) = 2$

$\text{Max}(2, 3) = 3$

At node E the value is 3 (greater than equal to 3)

- **At min:**

Next upper node is B (less than equal to 3)

But if you want to evaluate next part then initially $\alpha = -\text{inf}$ and $\beta = \text{inf}$

But now, $\alpha = -\text{inf}$

$\beta = 3$ (changes at min)

At node B the value is 3 (less than equal to 3)

- At max:

To max node 'F' we can take the values of 'B' node

Now $\alpha = -\infty$

$\beta = 3$

$\text{Max}(-\infty, 5) = 5$

Now $\alpha = 5$ which is going to change at 'F' node (greater than equal 5)
and $\beta = 3$ (remains constant at max level)

Now check the condition,

yes $\alpha \geq \beta$

Since condition is true prune the next path

- Now again check at B $\min(3,5) = 3$ (less than equal)

- At 'A' –max

Alpha=-inf (≥ 3)

Beta=inf

At 'c' – min

Alpha=3

Beta=inf

At 'G' ---max

Alpha =3

Beta=inf

But 'G' has only one possible , so no need to compare (G=0)

- At 'C' ---- min

Alpha=3

Beta=0

$\alpha \geq \beta$ ----- True ---- prone

- At 'I' node --- max

Initially alpha=3 and beta=inf

Now at 'I' alpha =3 and beta=inf

Now I=3

At 'D' node ----- min D=3

Initially alpha=3 and beta=inf

Now at 'D' node alpha=3 and beta=2

$\alpha \geq \beta$ ----- True ---- prone

- Best case of alpha-beta is $O(b^{d/2})$
- D -----depth of the node (ply)
- $O(b^{d/2})$ is higher than $O(b^d)$

Basic Knowledge Representation and Reasoning:

- **Intelligence**----- Machine requires intelligence to perform a task i.e., ability to use knowledge.
- The important factor of intelligence is **Knowledge**
- **How to represent knowledge**
- **How to give Knowledge to machine**
- **How to store on to machine**
- **Reasoning** ----- It is defined in different ways like Processing of knowledge, capability of thinking, to analyze, to have valid conclusion.

Knowledge and Intelligence

- Does knowledge have any role in demonstrating intelligent behavior?



How can we represent knowledge in a machine?

- We need a language to represent domain knowledge
- There must be a method to use this knowledge
- Inference Mechanism
- Syntax and Semantics of a language
 - Laughs (Tom) == ??
 - Likes (Sunita, Aditi) == ??

- Syntax : Grammatical of a language
- Semantics : meaning or sentence of a language

If there is no proper or good representation than that yields to the above two forms i.e., syntax and semantics

Inference mechanism reads the environment and interprets the knowledge that is represented in suitable language and act accordingly.

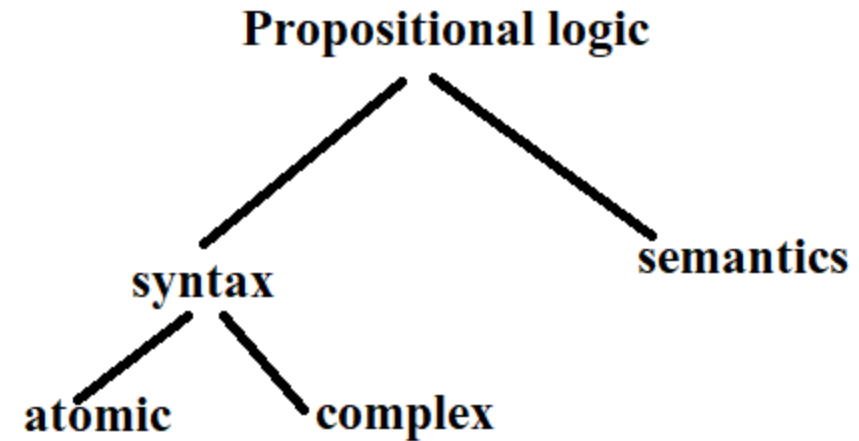
In simple words it requires a

language and

Method to use language

- So There must be a method or technique to represent the knowledge that is called logic (propositional logic and first-order predicate logic)
- It is one of the Knowledge representation

- Propositional logic is one of the simplest method of Knowledge representation.
- The term proposition simply defined like writing a sentence in terms of English language, programming or mathematical language.
- It is a language with some concrete rules which deals with propositions and has no ambiguity in representation.
- It consist of precisely ,
- The atomic and complex are the 2 representations of syntax and semantics

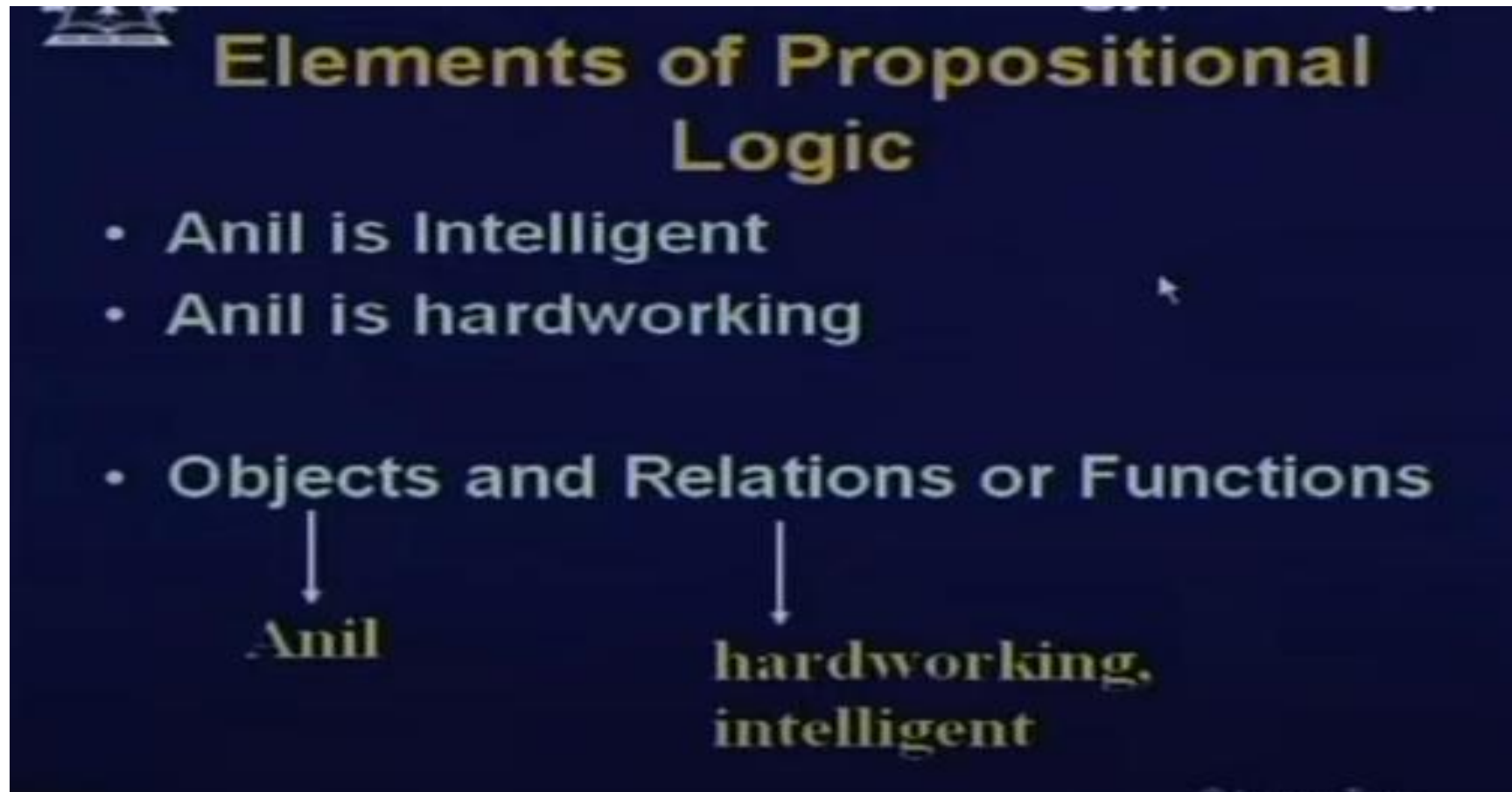


Logic is a Formal Language






- Propositional Logic

- Anil is Intelligent
 - Anil is hardworking
 - If Anil is Intelligent and Anil is hardworking Then Anil scores high marks
- Propositions
-
- The diagram illustrates that the three statements listed are propositions. Three white arrows point from the statements to the word 'Propositions'. The first arrow points from 'Anil is Intelligent', the second from 'Anil is hardworking', and the third from 'If Anil is Intelligent and Anil is hardworking Then Anil scores high marks'.

- This representation will support the inference and can be translated into logics using syntax and semantics.
- Syntax: well defined sentence in the language and should have the proper structure
- Semantics: Defines the truth of meaning of sentence



- Propositional logic is the declarative statement either in terms of **TRUE/FALSE**
- Connectives**

Word	Symbol	Example
Not		 A
And	\wedge	$A \wedge B$
OR		$A \vee B$
Implies		$A \longrightarrow B$
If and only if		$A \longleftrightarrow B$

Procedure to derive Truth Value

- Truth Table

P	Q	$P \wedge Q$
F	F	F
F	T	F
T	F	F
T	T	T

P	Q	$P \vee Q$
F	F	F
F	T	T
T	F	T
T	T	T

P	Q	$P \rightarrow Q$
F	F	T
F	T	T
T	F	F
T	T	T

P	Q	$P \leftrightarrow Q$
T	T	T
T	F	F
F	T	F
F	F	T

P	$\neg P$
T	F
F	T

Towards the Syntax

- Let P stand for Intelligent (Anil)
- Let Q stand for Hardworking (Anil)
- What does $P \wedge Q$ (P and Q) mean?
- What does $P \vee Q$ (P or Q) mean?
- $P \wedge Q$, $P \vee Q$ are compound propositions

Syntactic Elements of Propositional Logic

- Vocabulary
 - A set of propositional symbols (P,Q,R etc.) each of which can be True or False
 - Set of logical operators
 - \wedge (AND), \vee (OR), \neg (NOT), \rightarrow (Implies)
 - Often parenthesis () is used for grouping
 - There are two special symbols
TRUE (T) and FALSE (F) – these are logical constants

Example wffs

$\Box P$

$\Box \text{True}$

$\Box P \wedge Q$

$\Box (P \vee Q) \rightarrow R$

$\Box (P \wedge Q) \vee R \rightarrow S$

$\Box \neg (P \vee Q)$

$\Box \neg (P \vee Q) \rightarrow R \wedge S$

Implication \rightarrow

- $P \rightarrow Q$
- If P is true then Q is true
- If it rains then the roads are wet

P	Q	$P \rightarrow Q$
T	T	T
F	T	T
T	F	F
F	F	T

If the roads are wet than it rains ?????? ----- meaning less

there might be some other reasons to have the road wet.

Equivalence \Leftrightarrow

- $P \Leftrightarrow Q$


- Example ?

- If two sides of a triangle are equal then two base angles of the triangle are equal
- Can be represented as two sentences

P	Q	$P \leftrightarrow Q$
T	T	T
F	T	F
T	F	F
F	F	T

I go to mall if I have to do shopping





So how do we get the meaning?

- Remember: Sentences can be compound propositions
- Interpret each atomic proposition in the same world
- Assign Truth values to each interpretation
- Compute the truth value of the compound proposition

Validity of a sentence

- If a propositional sentence is true under all possible interpretation, it is **VALID**
- Tautology
 $P \vee \neg P$ is always true

$$\neg P \vee Q \rightarrow P \wedge Q$$

P	Q	$\neg P$	$\neg P \vee Q$	$P \wedge Q$	$\neg P \vee Q \rightarrow P \wedge Q$
F	F	T	T	F	F
F	T	T	T	F	F
T	F	F	F	F	T
T	T	F	T	T	T

Example 1

- A ----- It is hot
- B ----- It is humidity
- C ----- It is raining

Condition for propositional logic: write the propositional statements

If it is humid **then** it is hot ----- $B \rightarrow A$

If it is hot and humid then it is not raining ----- $(A \wedge B) \rightarrow \sim C$

Example 2:

P----You can access the internet from campus

Q---- you are CSE students

R---- you are freshman

You can access the internet from campus only if you are CSE students or you are not a freshman ----- $P \rightarrow (Q \vee \sim R)$



Limitation of Propositional Logic

- Consider the following argument
 - All dogs are faithful
 - Tommy is a dog
 - Therefore, Tommy is faithful
- How to represent and infer this in propositional logic?



Limitation of Propositional Logic

p is all dogs are faithful

q is Tommy is a dog

- But $p \wedge q \Rightarrow$ Tommy is faithful ??
- No! we cannot infer this in Propositional logic



More scenarios

- Tom is a hardworking student

Hardworking (Tom)

- Tom is an intelligent student

Intelligent (Tom)

- If Tom is hardworking and Tom is intelligent Then Tom scores high marks

Hardworking (Tom) \wedge Intelligent (Tom) \rightarrow
Scores_High_Marks (Tom)



What about John and Jill ?

If we could write instead

All students who are hardworking and
intelligent scores high marks !!!

For all x such that x is a student and x is
intelligent and x is hardworking then x
scores high marks

The Problem of Infinite Model

- In general, propositional logic can deal with only a finite number of propositions.
- If there are only three dogs Tommy, Jimmy and Laika, then

T : Tommy is faithful

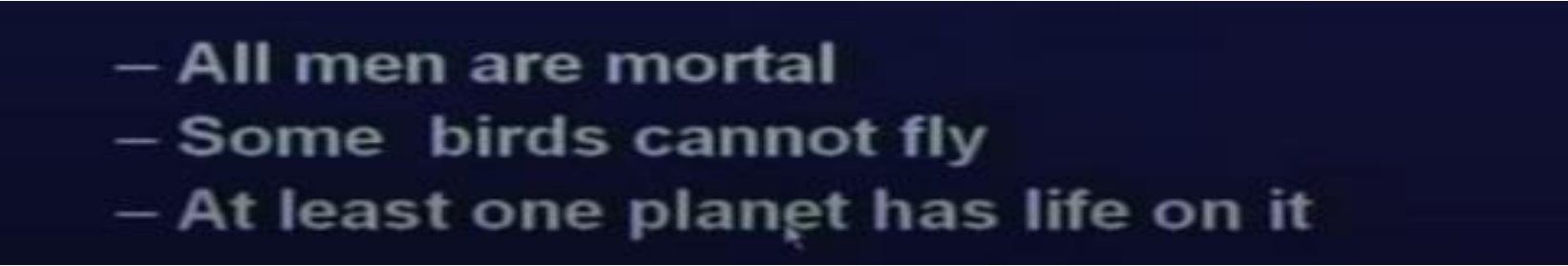
J : Jimmy is faithful,

L : Laika is faithful

All dogs are faithful $\Leftrightarrow T \wedge J \wedge L$

First-Order Logic

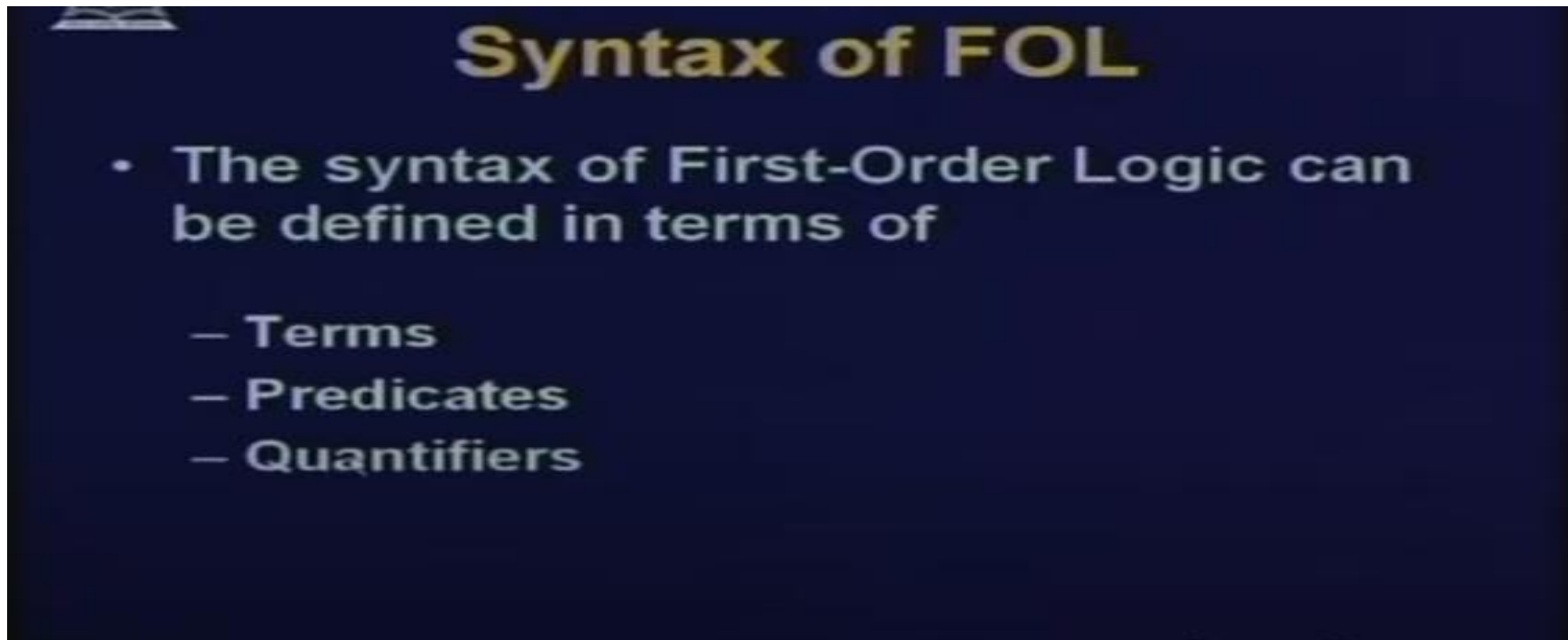
- it is the another way of representing the knowledge in AI, considered to be an extension of PL (proposition logic)
- FOL is also known as predicate logic
- FOL is defined as a powerful language that develops information about the object more easy way and also can express the relationship between their objects and also infer arguments in infinite models like



- All men are mortal
- Some birds cannot fly
- At least one planet has life on it

- Simply says it assumes objects, relations and functions

- FOL is also a natural language which has two parts : syntax and semantics
- The basic syntax elements of: constants (terms), variables (terms), predicates, Functions (terms), Connectives, equality, Quantifiers



Syntax of FOL

- The syntax of First-Order Logic can be defined in terms of
 - Terms
 - Predicates
 - Quantifiers

- terms

- Tommy is a term
- Men is a term

A term denotes some object other than true or false.

Tommy is a dog

All men are mortal

- A functional term of arity n takes n objects of type W_1 to W_n as inputs and returns an object of type W .

$f(w_1, w_2, \dots, w_n)$

$plus(3,4) = 7$

functional term

constant terms

- **Predicates:**

- These are the sentences which are joined from a predicate symbol followed by parenthesis () with sequence of terms.
- The representation is

Predicate(term1,term2,.....term n)

- Predicates are like functions except that their return type is true or false
- Example:
 - $gt(x,y)$ is true iff $x > y$
 - Here gt is a predicate symbol that takes two arguments of type natural number
 - $gt(3,4)$ is a valid predicate but $gt(3,-4)$ is not



Types of Predicates

- A predicate with no variable is a proposition
 - Tommy is a dog
- A predicate with one variable is called a property
 - $dog(x)$ is true iff x is a dog.
 - mortal (y) is true iff y is mortal



Formulation of Predicates

- Let $P(x,y,...)$ and $Q(x,y,...)$ are two predicates.
- Then so are

$$P \vee Q$$

$$P \wedge Q$$

$$\neg P$$

$$P \Rightarrow Q$$

- Example:
- Hari and Raghu are brothers -----> brother(Hari, Raghu)
- Tommy is a dog --->dog(Tommy)

Predicate Examples

- If x is a man then x is mortal
 $\text{man}(x) \Rightarrow \text{mortal}(x)$
 $\neg \text{man}(x) \vee \text{mortal}(x)$
- If n is a natural number, then n is either even or odd.
 $\text{Natural}(n) \Rightarrow \text{even}(n) \vee \text{odd}(n)$

- Quantifiers:

- It is a language element which generates quantification.
- These are the symbols that permits to determine or identify the range and scope of the variable in the logic expression.

- There are two basic quantifiers in FOL
 - \forall “For all” – Universal quantifier
 - \exists “There exists” – Existential quantifier

last 1 week.



$$\forall x \ P(x)$$

$$\exists x \ \underline{\underline{Q(x)}}$$

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

- Universal quantifier is a symbol of logical representation which specify the statement with in a range is true for everything.
- In universal quantifiers we use implification as a symbol (\longrightarrow)

Example: If x is a variable then $\forall x$ is read as different ways with in a range

- Existential quantifier will express the statement with in its scope is true for at least one instance of sometimes.
- In Existential quantifier we use “V” , “ \wedge ”

Example: If x is a variable then $\exists (x)$ is read as different ways with in some scope of existence.

Universal Quantifiers

- All birds cannot fly

fly (x) : x can fly

bird (x) : x is a bird

$$\neg(\forall x (\text{bird}(x) \Rightarrow \text{fly}(x)))$$



Existential Quantifiers

- At least one planet has life on it
Planet (x) : x is a planet
haslife (x) : x has life on it
 $\exists x (\text{planet}(x) \wedge \text{haslife}(x))$

Examples:

① All man drink coffee

↓

∀ let x is a variable

↓

x_1, x_2, \dots, x_n

i.e., x_1 drink coffee $\wedge x_2 \dots \wedge x_n$

So we can represent as

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

② All birds fly

↓

∀

fly (bird)

↓

Predicate

↓

term

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

③ Every man respects his parent

\forall $\exists x$ $\exists y$

$\forall x \text{ man}(x) \longrightarrow \text{respects}(x, \text{Parent})$

④ Some boys are intelligent

\exists Consider 'some' as 'x'

$x_1 \vee x_2 \vee x_3 \vee \dots \vee x_n$

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

↓

There are some boys (x) who is intelligent

Example:

- John likes all kind of food
- Apple and vegetables are food
- Anything anyone eats and not killed his food
- Anil eats peanut and still alive
- Harry eats everything that anil eats
- John like peanuts

Convert all the above statements into predicate logic (or) FOL?

Solution:

$\forall x \text{ food}(x) \rightarrow \text{likes}(\text{John}, x)$

$\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetable})$

$\forall x \forall y \text{ eats}(x, y) \wedge \sim \text{killed}(x) \rightarrow \text{food}(y)$

$\text{eats}(\text{Anil}, \text{peanuts}) \wedge \text{alive}(\text{Anil})$

$\forall x \text{ eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$

$\text{likes}(\text{John}, \text{peanuts})$

Inference in FOL:

- This is used to deduce new facts or sentences from existing sentences
- By understanding FOL inference rules let us understand some basic terminology used:

Substitution:

It is a fundamental operation performed on terms and formulae's

Equality:

It not only uses predicate ,terms for creating or making atomic sentences but also uses equality

Example:

Brother(john)=smith

$\sim(x=y)$ equivalent to $x \neq y$

Inference rules for quantifiers

- Universal Generalization
- Universal Instantiation
- Existential Instantiation
- Existential Introduction

- Universal Generalization:

It is a valid inference rule which states that if premises $p(c)$ is true for any arbitrary element 'C' in the universe of discourse, then $\forall (x) p(x)$ can be represented as,

$$\frac{P(c)}{\forall (x) p(x)}$$

- Universal Instantiation:

It is also called as universal elimination, can be applied multiple times to add new sentence.

$$\frac{\forall (x) p(x)}{P(c)}$$

- Existential Instantiation:

It is also called as existential elimination which can be applied only once to replace the existential sentence.

$$\frac{\exists x p(x)}{p(c)}$$

- Existential Introduction:

It states that if there is some element 'c' in the universe of discourse, which has a property 'p'

$$\frac{P(c)}{\exists x p(x)}$$

Resolution in FOL:

- It is defined as a theorem proven technique which proves by contradiction.
- It is used when there are various statements, then read to prove conclusion of those statements.
- Unification is the key concept which proves the conclusion of those statements.
- Resolution is a single inference rule which can be efficiently operated on Horn clause and definite clause (conjunction normal form and clausal form)

Clause: Disjunction of literals

Conjunctive Normal Form: A sentence represented as a conjunction of clauses said to be CNF

Steps for resolution:

- Conversion of Sentence into FOL
- Convert FOL statement into CNF
- Negate the statement which needs to prove (by contradiction)
- Draw resolution graph (unification).

Example:

John likes all kind of food

Apple and vegetables are food

Anything anyone eats and not killed his food

Anil eats peanut and still alive

Harry eats everything that anil eats

Prove by resolution that --- John like peanuts

Step 1: Convert the given into FOL

Solution:

$\forall x \text{ food}(x) \rightarrow \text{likes}(\text{John}, x)$

$\neg \text{food}(\text{Apple}) \wedge \text{food}(\text{vegetable})$

$\forall x \forall y \text{ eats}(x, y) \wedge \neg \text{killed}(x) \rightarrow \text{food}(y)$

$\text{eats}(\text{Anil}, \text{peanuts}) \wedge \text{alive}(\text{Anil})$

$\forall x \text{ eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$

$\text{likes}(\text{John}, \text{peanuts})$

- Here I am adding some predicates to show equality

Solution:

$\forall x \text{ food}(x) \rightarrow \text{likes}(\text{John}, x)$

$\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetable})$

$\forall x \forall y \text{ eats}(x, y) \wedge \sim \text{killed}(x) \rightarrow \text{food}(y)$

$\text{eats}(\text{Anil}, \text{peanuts}) \wedge \text{alive}(\text{Anil})$

$\forall x \text{ eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$

$\text{likes}(\text{John}, \text{peanuts})$

Here I am adding some predicates:

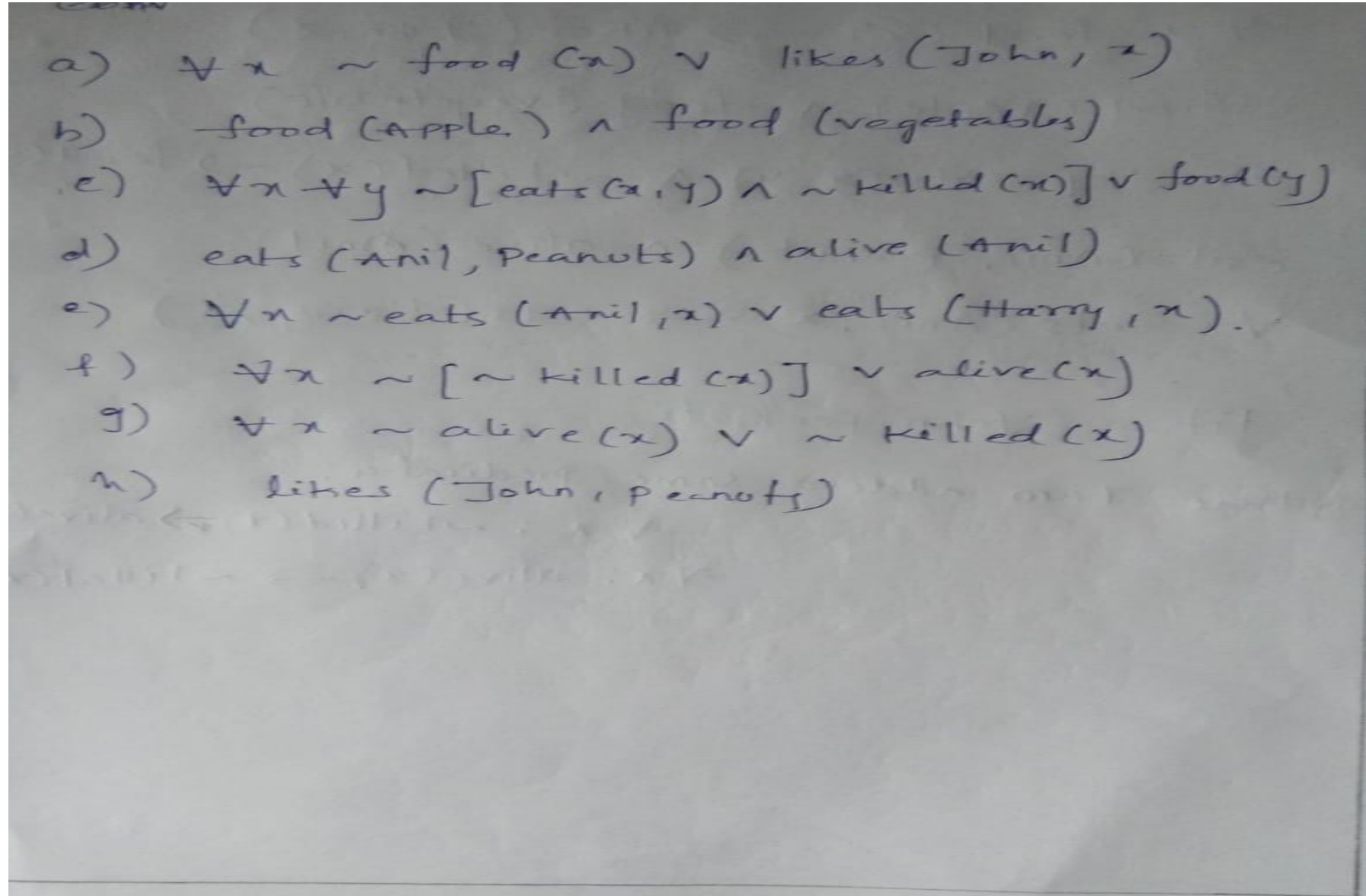
$\forall x : \sim \text{killed}(x) \Rightarrow \text{alive}(x)$

$\forall x : \text{alive}(x) \rightarrow \sim \text{killed}(x)$

- Step 2: Conversion of FOL into CNF

$$P \rightarrow Q = \neg P \vee Q$$

(i) Eliminate all implications and rewrite

- 
- a) $\forall x \sim \text{food}(x) \vee \text{likes}(\text{John}, x)$
- b) $\neg \text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
- c) $\forall x \forall y \sim [\text{eats}(x, y) \wedge \sim \text{killed}(x)] \vee \text{food}(y)$
- d) $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
- e) $\forall x \sim \text{eats}(\text{Anil}, x) \vee \text{eats}(\text{Harry}, x)$
- f) $\forall x \sim [\sim \text{killed}(x)] \vee \text{alive}(x)$
- g) $\forall x \sim \text{alive}(x) \vee \sim \text{killed}(x)$
- h) $\text{likes}(\text{John}, \text{peanuts})$

(ii) Move negation inside and rewrite

- a) $\forall x \sim \text{food}(x) \vee \text{likes}(\text{John}, x)$
- b) $\text{food}(\text{apple}) \wedge \text{food}(\text{vegetables})$
- c) $\forall x \forall y \sim \text{eats}(x, y) \vee \text{killed}(x) \vee \text{food}(y)$
- d) $\text{eats}(\text{Anil}, \text{peanuts}) \wedge \text{alive}(\text{Anil})$
- e) $\forall x \sim \text{eats}(\text{Anil}, x) \vee \text{eats}(\text{Harry}, x)$
- f) $\forall x \sim \text{killed}(x) \vee \text{alive}(x)$
- g) $\forall x \sim \text{alive}(x) \vee \sim \text{killed}(x)$
- h) $\text{likes}(\text{John}, \text{peanuts})$

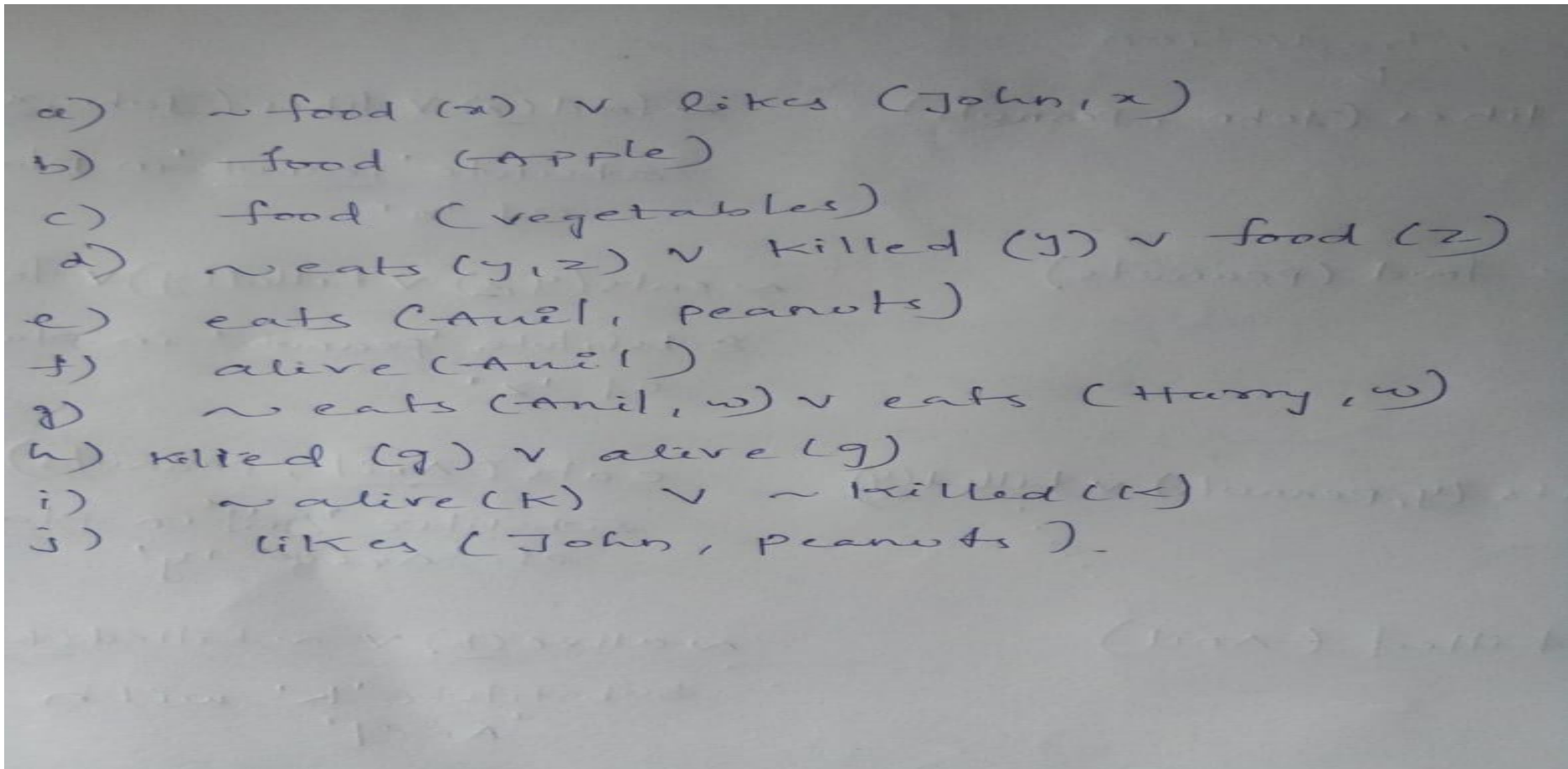
- (iii) Rename or standardize variable

- a) $\forall x \sim \text{food}(x) \vee \text{likes}(\text{John}, x)$
- b) $\text{food}(\text{Apple}) \wedge \text{food}(\text{Vegetables})$
- c) $\forall y \forall z \sim \text{eats}(y, z) \vee \text{killed}(y) \wedge \text{food}(z)$
- d) $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
- e) $\forall w \sim \text{eats}(\text{Anil}, w) \vee \text{eats}(\text{Harry}, w)$
- f) $\forall g \sim \text{killed}(g) \vee \text{alive}(g)$
- g) $\forall k \sim \text{alive}(k) \vee \sim \text{killed}(k)$
- h) $\text{likes}(\text{John}, \text{Peanuts})$

- (iv) Eliminate any Existential instantiation quantifiers in the statements:

No problem because in any of statement there is no existential quantifier

- (v) Drop universal quantifiers



Step3: Negate the statement to be proved

In this we consider or apply negation to the conclusion statement i.e., it is represented as

$\sim \text{likes}(\text{john}, \text{peanuts})$

Step4: Draw resolution graphs (unification)

In this we will solve problem by resolution tree using substitution, so the resolution graph is given as

Contradiction

↓

$\sim \text{likes}(\text{John}, \text{Peanuts})$

$\sim \text{food}(\underline{x}) \vee \text{likes}(\text{John}, \underline{x})$

Substitute 'peanuts' in place
of 'x'

$\sim \text{food}(\text{peanuts})$

$\sim \text{eats}(\underline{y}, \underline{z}) \vee \text{killed}(\underline{y}) \vee \text{food}(\underline{z})$

Substitute 'peanuts' in place
of 'z'

$\sim \text{eats}(\underline{y}, \text{peanuts}) \vee \text{killed}(\underline{y})$

$\text{eats}(\underline{\text{Anil}}, \text{peanuts})$

Substitute 'Anil' in place
of 'Aneg' 'y'

$\text{killed}(\underline{\text{Anil}})$

$\sim \text{alive}(\underline{k}) \vee \sim \text{killed}(\underline{k})$

Substitute 'k' with
'Anil'

$\sim \text{alive}(\text{Anil})$

$\text{alive}(\text{Anil})$

{ }

Hence proved.

Forward chaining and Backward chaining

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

Inference engine:

- The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts. The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:
 1. Forward chaining
 2. Backward chaining
- Horn Clause and Definite clause:

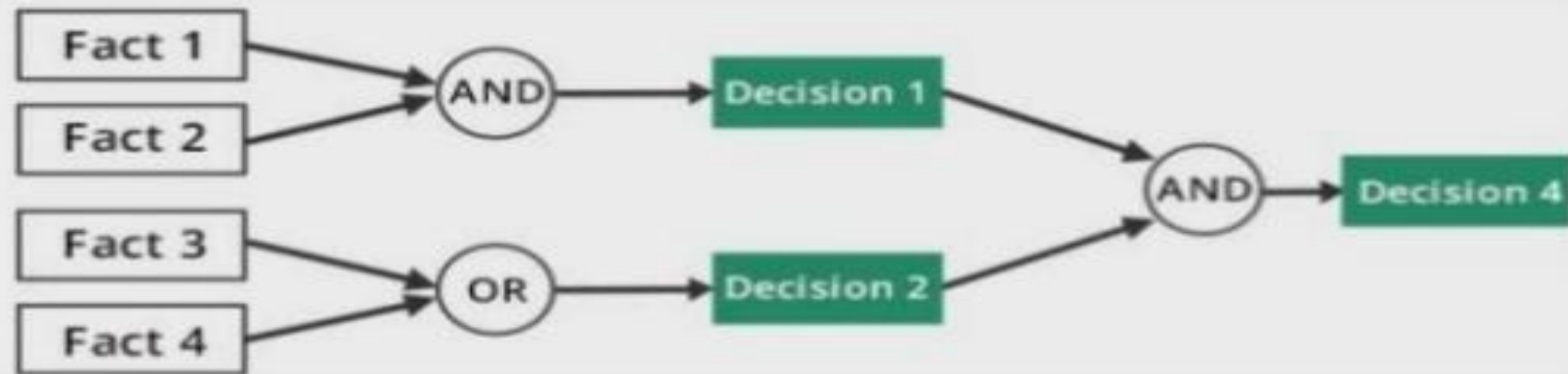
- Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.
- The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Forward Chaining

It is a strategy of an expert system to answer the question, "**What can happen next?**"

Here, the inference engine follows the chain of conditions and derivations and finally deduces the outcome. It considers **all** the facts and rules, and sorts them before concluding to a solution.

This strategy is followed for working on conclusion, result, or effect. For example, prediction of share market status as an effect of changes in interest rates.



In Forward chaining, the system starts from a set of facts, and a set of rules, and tries to find a way of using those rules and facts to deduce a conclusion or come up with a suitable course of action.

This is known as **data-driven reasoning** because **the reasoning starts from** a set of data and ends up at the goal, which is the conclusion.

Forward chaining mechanism

When applying forward chaining, the first step is to take the facts in the fact database and see if any combination of these matches all the antecedents of one of the rules in the rule database. When all the antecedents of a rule are matched by facts in the database, then this rule is **triggered**.

Usually, when a rule is triggered, it is then **fired**, which means its conclusion is added to the facts database. If the conclusion of the rule that has fired is an action or a recommendation, then the system may cause that action to take place or the recommendation to be made.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

- Consider the following example:

Rule1: If A and C then F

Rule2: If A and E then G

Rule3: IF b then E

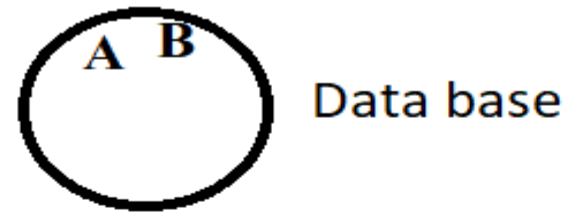
Rule4: If G then D

Problem : Prove IF A and B are true then D is true (IF A and B then D)

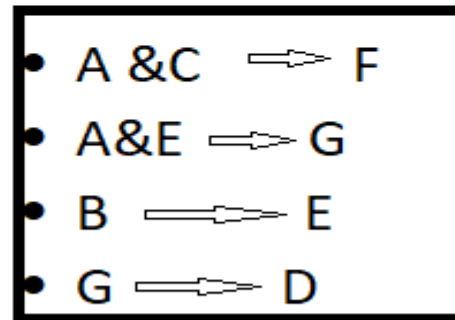
In Data base we have A and B

In Knowledge base :

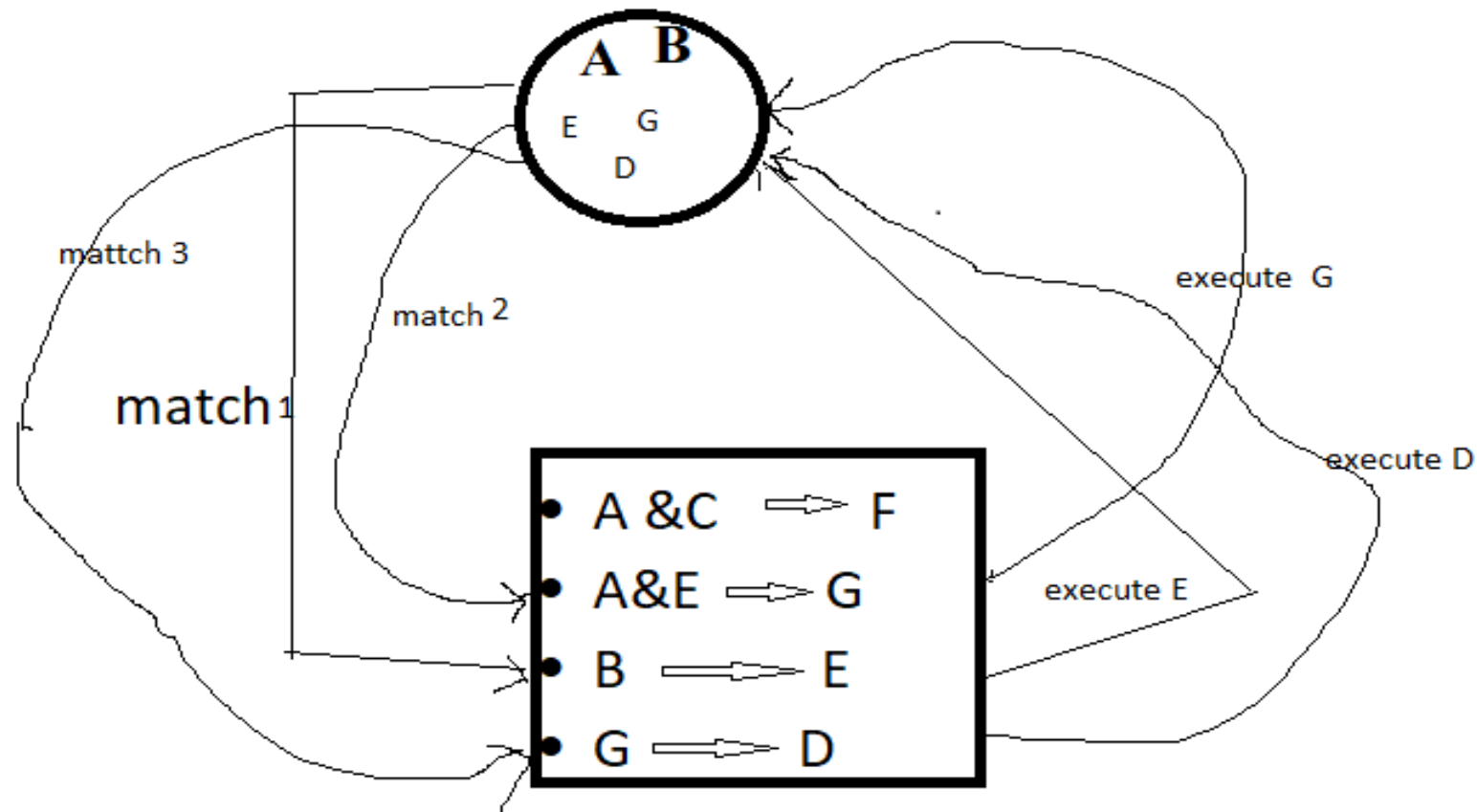
- $A \ \& \ C \longrightarrow F$
- $A \ \& \ E \longrightarrow G$
- $B \longrightarrow E$
- $G \longrightarrow D$



Knowledge base



Finally reached from initial states A & B to goal state D



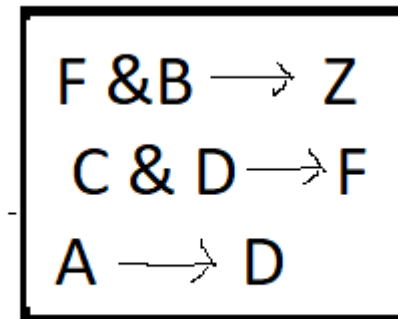
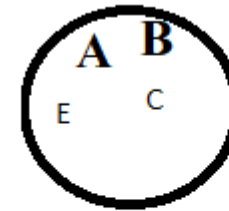
Example 2

- Goal state : Z
- Facts/Data base : A,B,E,C
- Rules:

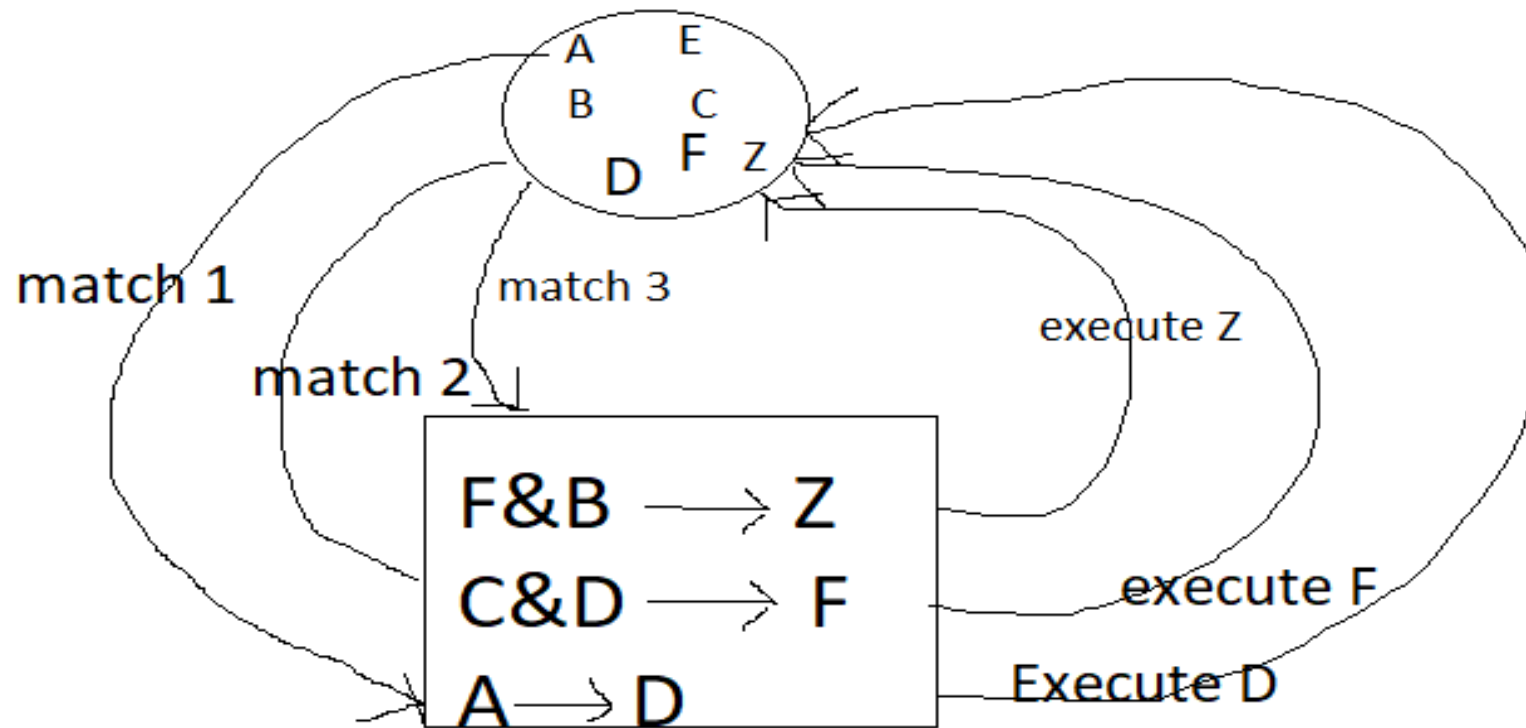
Rule1: $F \ \& \ B \longrightarrow Z$

Rule2: $C \ \& \ D \longrightarrow F$

Rule 3: $A \longrightarrow D$



Finally reached conclusion state 'Z'

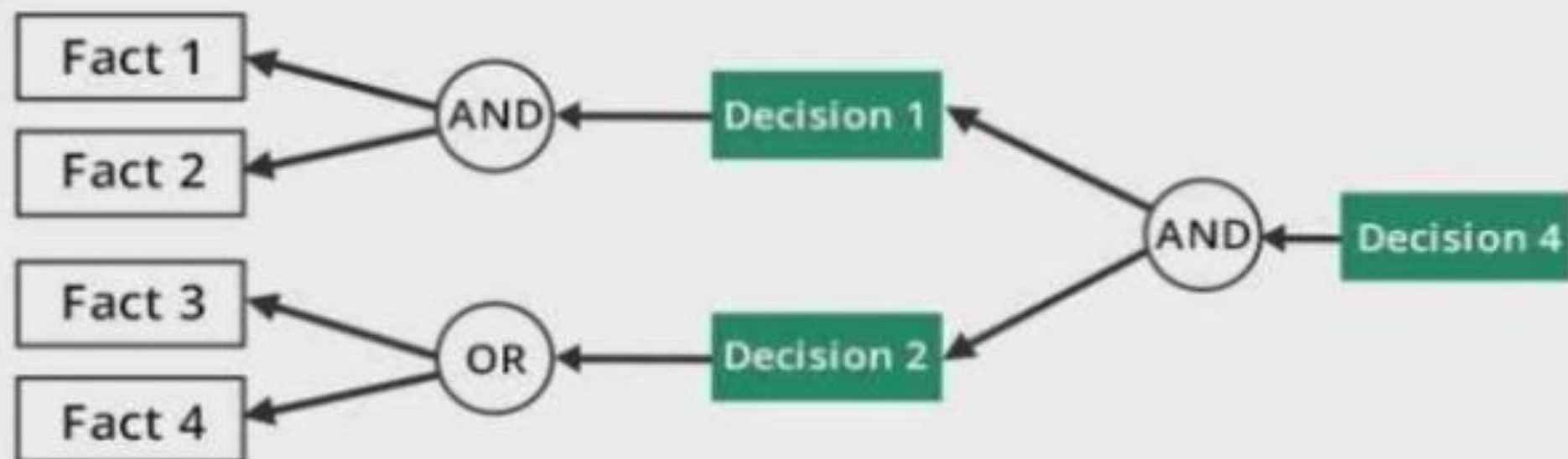


- **Backward-chaining** is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.
- Backward chaining is a ***goal driven*** method of deriving a particular goal from a given knowledge base and set of inference rules
- Inference rules are applied by matching the goal of the search to the consequents of the relations stored in the knowledge base

Backward Chaining

With this strategy, an expert system finds out the answer to the question, **"Why this happened?"**

On the basis of what has already happened, the interface engine tries to find out which conditions could have happened in the past for this result. This strategy is followed for finding out cause or reason. For example, diagnosis of blood cancer in humans.



- In other words

In backward chaining, we start from a conclusion, which is the hypothesis we wish to prove, and we aim to show how that conclusion can be reached from the rules and facts in the database.

The conclusion we are aiming to prove is called a **goal**, and so reasoning in this way is known as **goal-driven reasoning**.

Backward chaining mechanism

Backward chaining starts with the goal state, which is the set of conditions the agent wishes to achieve in carrying out its plan. It now examines this state and sees what actions could lead to it. For example, if the goal state involves a block being on a table, then one possible action would be to place that block on the table. This action might not be possible from the start state, and so further actions need to be added before this action in order to reach it from the start state. In this way, a plan can be formulated starting from the goal and working back toward the start state.

Properties of backward chaining:

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

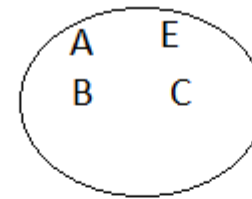
Example:

- Goal state: Z
- Facts: A,E,B,C
- Rules:

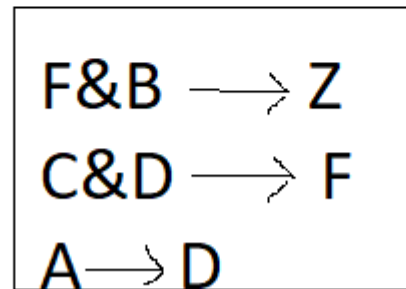
Rule1: $F \& B \rightarrow Z$

Rule2: $C \& D \rightarrow F$

Rule3: $A \rightarrow D$

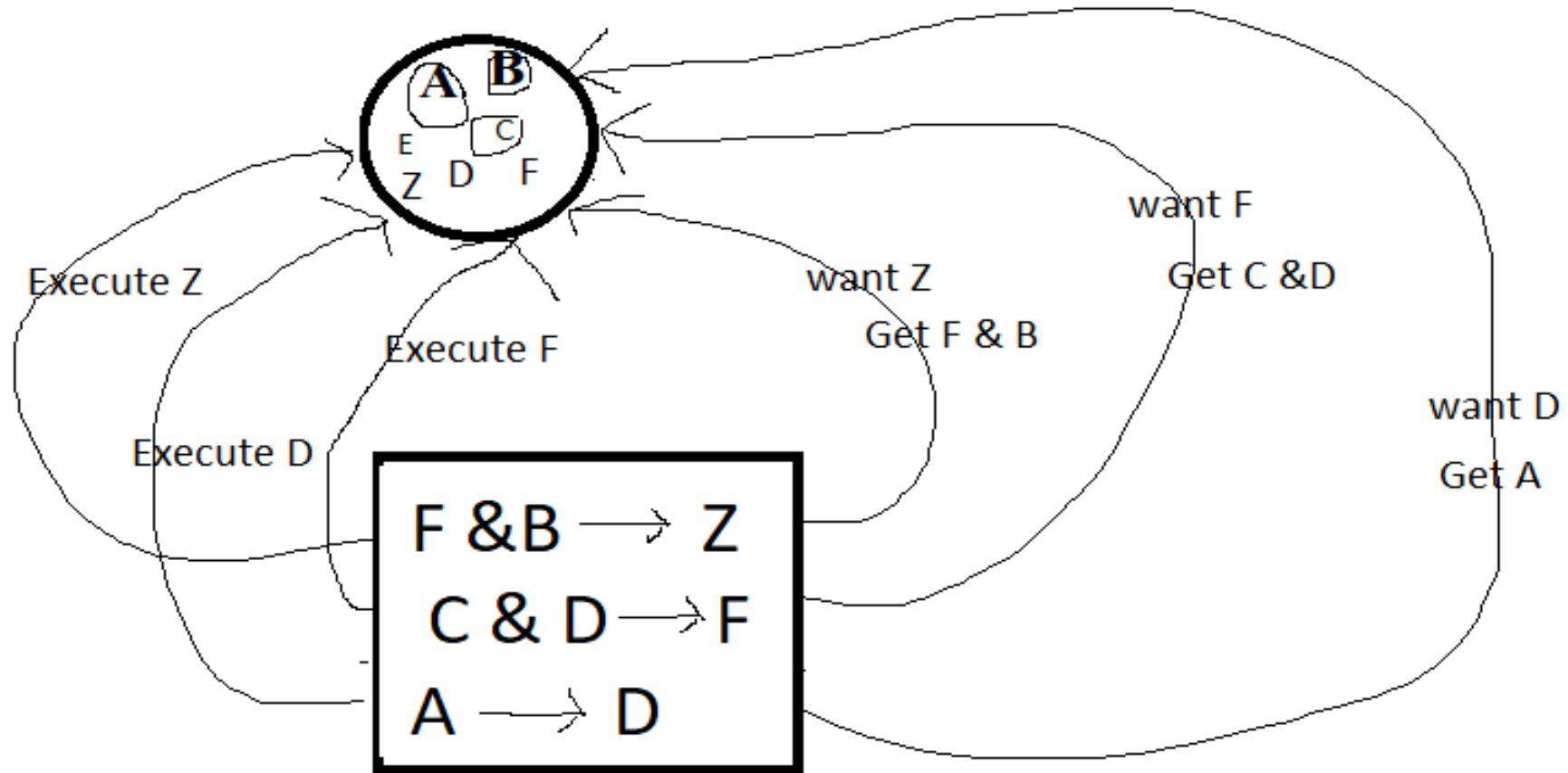


Facts/data base



Knowledge base

- Finally reached “Z” and added into the facts/data base



S. No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
8.	It operates in the forward direction.	It operates in the backward direction.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.

Introduction to probabilistic reasoning

- Reasoning:

- ✎ Reasoning is the process by which we use the knowledge we have to draw conclusions or infer something new about a domain of interest.
- ✎ Without the ability to reason, we are simple recalling information. (Infact this is the difference between a standard database system and a knowledge-based expert system. Unlike the expert system, the database has no reasoning facilities)

Probability

- Probabilities are used to compute the truth of given statement, written as numbers between 0 and 1, that describes how likely an event is to occur.
- 0 indicates impossibility and 1 indicates certainly.
 - 1. Tossing a coin 2. Trolling a dice
- Probability based reasoning
 - understanding from knowledge
 - how much of uncertainty present in that event.

- *Probability provides a way of **summarizing** the uncertainty, that comes from our **laziness** and **ignorance**.*
- Toothache problem - an 80% chance , a probability of 0.8 that the patient has a cavity if he or she has a toothache.
- The 80% summarizes those cases, but both toothache and cavity are unconnected.
- The missing 20% summarizes, all other possible causes of toothache, that we are **too lazy** or **ignorant to confirm** or **deny**.

- Probabilities between 0 and 1 correspond to intermediate degrees of belief in the **truth of the sentence**.
- The sentence itself is in *fact* either **true or false**.
- It is important to note that a **degree of belief** is different from a degree of truth.
- A probability of 0.8 does not mean "80% true" but rather an 80% degree of belief-that is, a fairly strong expectation.
- Thus, probability theory makes the same **ontological commitment** as logic-namely, that facts either do or do not hold in the world.
- Degree of truth, as opposed to degree of belief, is the subject of **fuzzy logic**

- In probability theory, a sentence such as
- "The probability that the patient has a cavity is 0.8",
- is about the agent's beliefs, not directly about the world.
- These percepts create the **evidence**, which are based on probability statements.

- All probability statements must indicate the evidence with respect to that probability is being assessed.
- If an agent receives new percepts, its probability assessments are updated to reflect the new evidence.

Random Variable

- Referring to a "part" of the world, whose "status" is initially unknown
- We will use lowercase for the names of values
 - $P(a) = 1 - P(\neg a)$
- Tossing coin : $P(h) = 1 - P(\neg h) : (0.5 = 1 - 0.5)$
- Rolling dice : $P(n) = 1 - P(\neg n) : (0.16 = 1 - 0.84)$

Types of random variables

- Boolean random variables
 - Cavity domain (true,false), if Cavity = true then cavity, or
 - if Cavity = false then \neg cavity
- Discrete random variables – countable domain
 - *Weather* might be (sunny, rainy, cloudy, snow)
- Continuous random variables – finite set real numbers with equal intervals e.g. interval(0.1)

Atomic events

- This is useful in understanding the foundations of probability theory.

Atomic events...

- Atomic events have some important properties
- They are **mutually exclusive** -at most one can actually be the case.
- The set of all possible atomic events is **exhaustive** – at least one must be the case.
- Any particular atomic event entails the truth or falsehood of every proposition, whether simple or complex
- Any proposition is **logically equivalent** to the disjunction of all atomic events that required the **truth of proposition**.

Prior Probability

- The **unconditional** or **prior probability** associated with a proposition a , is the **degree of belief** according to the absence of any other information;
- it is written as $P(a)$.
- For example, if the prior probability that one have a cavity is 0.1, then we would write
 - $P(\text{Cavity} = \text{true}) = 0.1$ or $P(\text{cavity}) = 0.1$.
- It is important to remember that $P(a)$ can be used only when **there is no other information**.

Posterior probability: Which is calculated after all information or informed in prior is taken into he account. Posterior probability is the combination of prior probability and new information

Conditional Probability

- The **conditional** or **posterior** probabilities notation is $P(a|b)$,
- where a and b are any proposition.
- This is read as "the probability of a , given that *all* we know is b ."
- For example, $P(\text{cavity} | \text{toothache}) = 0.8$
- if a patient is observed to have a toothache and no other information is yet available, then the probability of the patient's having a cavity will be 0.8.

Conditional probabilities...

- Conditional probabilities can be defined in terms of unconditional probabilities.
- The equation is

$$P(a|b) = \frac{P(a \wedge b)}{P(b)}$$

- whenever $P(b) > 0$.
- This equation can also be written as
- $P(a \wedge b) = P(a/b) P(b)$ which is called the **product rule**.

- **Example: rolling a dice (6)**
- Sample space(σ)= {1,2,3,4,5,6}
- Events: even {2,4,6} and odd {1,3,5}

$$P(6)=1/6$$

$$P(\text{even}) = 3/6 = 1/2$$

$$P(\text{odd}) = 3/6 = 1/2$$

Conditional probability : What is the probability of getting 6 when it is an even no?

$$P(a/b) = \frac{p(a \cap b)}{p(b)} = \frac{1/6}{1/2}$$

Sol:1/3

Basic Axioms of Probability

- All probabilities are between 0 and 1. For any proposition a ,

$$0 \leq P(a) \leq 1$$

- Necessarily true (i.e., valid) propositions have probability 1, and necessarily false (i.e., unsatisfiable) propositions have probability 0.

$$P(\text{true}) = 1 \qquad P(\text{false}) = 0 .$$

- The probability of a disjunction is given by

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b) .$$

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 An important branch of applied statistics called Bayes Analysis can be developed out of conditional probability. It is possible given the outcome of the second event in a sequence of two events to determine the probability of various possibilities for the first event.

Bayes Theorem

Probability – Bayes' Rule

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

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

- From conditional prob:

$$P(A|B) = p(A \cap B) / p(B) \text{ ---- } P(A|B) \cdot p(B) = p(A \cap B) \text{ -----eq1}$$

$$P(B|A) = p(B \cap A) / p(A) \text{ ---- } P(B|A) \cdot p(A) = p(B \cap A) \text{ -----eq2}$$

$$\text{eq1} = \text{eq2}$$

- $P(A|B) \cdot p(B) = P(B|A) \cdot p(A)$ from this get $P(A|B)$ & $P(B|A)$

$$\left. \begin{array}{l} P(A|B) = P(B|A) \cdot p(A) / p(B) \\ P(B|A) = P(A|B) \cdot p(B) / p(A) \end{array} \right\} \text{ Bayes' rule}$$

- Calculate Hypothesis for flu based on symptoms: GIVEN

$P(A)$: symptom of flu ---- 0.00001

$P(B|A)$: prob of symptoms gives flu ---- 0.95

$P(B)$: your symptoms of flu = 0.01

- $P(A|B) = p(B|A) * p(A) / p(B)$

$$0.95 * 0.0001 / 0.01 = 0.00095 \text{ (<1 in thousand people)}$$

Example for conditional probability given:

$P(C)$ ---70%----children like chocolates (0.7)

$p(C \text{ and } S)$ ----35%---children likes both chocolates and strawberry (0.35)

- **Calculate conditional prob who likes chocolates also likes strawberry**

$$P(S|C) = p(C \wedge S) / p(C)$$

$$0.35 / 0.7 = 0.5 \text{ (50\% --- who likes chocolates also likes strawberry)}$$

- **Example – How to buy a used car**

- I am thinking of buying a used car at Honest Ed's. In order to make an informed decision, I look up the records in an auto magazine of the car type I am interested in and find that unfortunately 30% have faulty transmissions.
- To get more information on this particular car at Honest Ed's I hire a mechanic who can make a shrewd guess on the basis of a quick drive around the block. Of course, he isn't always right but he does have an excellent record. Of all the faulty cars he has examined in the past, he correctly pronounced 90% "faulty". In other words, he wrongly pronounced only 10% "ok".
- He has almost as good a record in judging good cars. He has correctly pronounced 80% "ok", while he wrongly pronounced only 20% "faulty".
- "faulty" describes the mechanics opinion.
- faulty with no quotation marks describes the actual state of the car.

- **Example – How to buy a used car**

- What is the chance that the car I'm thinking of buying has a faulty transmission:

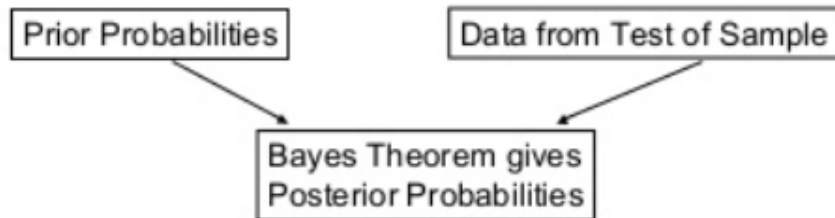
1. Before I hire the mechanic?
2. If the mechanic pronounces it "faulty"?
3. If the mechanic pronounces it "ok"?

The Logic of Bayes Theorem

Bayes Theorem gives us a way of calculating $P(A|B)$ from a knowledge of $P(B|A)$.

The point of Bayes may be stated more generally:

Prior probabilities combined with some sort of information such as a test or sample yield posterior probabilities.



Posterior Probability

Law of Total Probability

If a sample space can be partitioned into k mutually exclusive and exhaustive events

$$A_1, A_2, A_3, \dots, A_k$$

i.e. $S = A_1 \cup A_2 \cup A_3 \dots \cup A_k$

Then for any event E :

$$P(E) = P(A_1)P(E|A_1) + P(A_2)P(E|A_2) \dots + P(A_k)P(E|A_k)$$

Proof:

$$\begin{aligned} E &= E \cap S \\ &= E \cap (A_1 \cup A_2 \cup \dots \cup A_k) \\ &= (E \cap A_1) \cup (E \cap A_2) \cup \dots \cup (E \cap A_k) \end{aligned}$$

Since these are mutually exclusive

$$\begin{aligned} P(E) &= P(E \cap A_1) + P(E \cap A_2) + \dots + P(E \cap A_k) \\ &= P(A_1)P(E|A_1) + P(A_2)P(E|A_2) \dots + P(A_k)P(E|A_k) \end{aligned}$$

Bayes Theorem

If a sample space can be partitioned into k mutually exclusive and exhaustive events

$$A_1, A_2, A_3, \dots, A_k$$

$$S = A_1 \cup A_2 \cup A_3 \dots \cup A_k$$

Then for any event E : $P(A_i | E) = \frac{P(A_i) P(E|A_i)}{P(E)}$

where

$$P(E) = P(A_1)P(E|A_1) + P(A_2)P(E|A_2) \dots + P(A_k)P(E|A_k)$$

Proof:

For any i , $1 \leq i \leq k$

$$= E \cap A_i = A_i \cap E$$

$$= P(E) P(A_i|E) = P(A_i) P(E|A_i)$$

Prior Probability

Posterior Probability

$$P(A_i | E) = \frac{P(A_i) P(E|A_i)}{P(E)}$$

Conditional Probability

Relating Bayes To The Car Example

Prior probabilities

The initial probabilities before any testing are called the prior probabilities.

Posterior probabilities

The probabilities after testing are called the posterior probabilities.

Note that the sum of the branches from each node sum to 1.

Assignment:

Example

An insurance company runs three different offices , A, B and C. 30% of the company's employees work in Office A, 20% in Office B and 50% in Office C.

10% of the staff in Office A are managers, 20% of the staff in Office B are managers and 5% from Office C are managers.

Offices	A	B	C
Proportion Employees	.3	.2	.5
Proportion Managers	.1	.2	.05

- (a) What is the total proportion of managers in the company?
- (b) If a member of staff, chosen at random, turns out to be a manager, what is the probability that she works in Office A?

UNIT - III

Advanced Knowledge Representation and Reasoning:
Knowledge Representation Issues, Nonmonotonic
Reasoning, Other Knowledge Representation Schemes
Reasoning Under Uncertainty: Basic probability, Acting
Under Uncertainty, Bayes' Rule, Representing
Knowledge in an Uncertain Domain, Bayesian Networks

- Introduction:

Human beings are good at understanding, reasoning, and interpreting knowledge and using this knowledge, they could be able to perform various actions in the real world.

But,

How do machines perform the same?

How it helps machine in reasoning and interpretation?

KNOWLEDGE REPRESENTATION AND REASONING

Knowledge representation in AI describes the representation of any knowledge.

Knowledge representation also known as KR/KRR represents information from the real world from a computer to understand and then utilize this knowledge to solve complex real life problems like communication with human beings in natural language.

Different Kinds of Knowledge Need to Represent the Following Things:

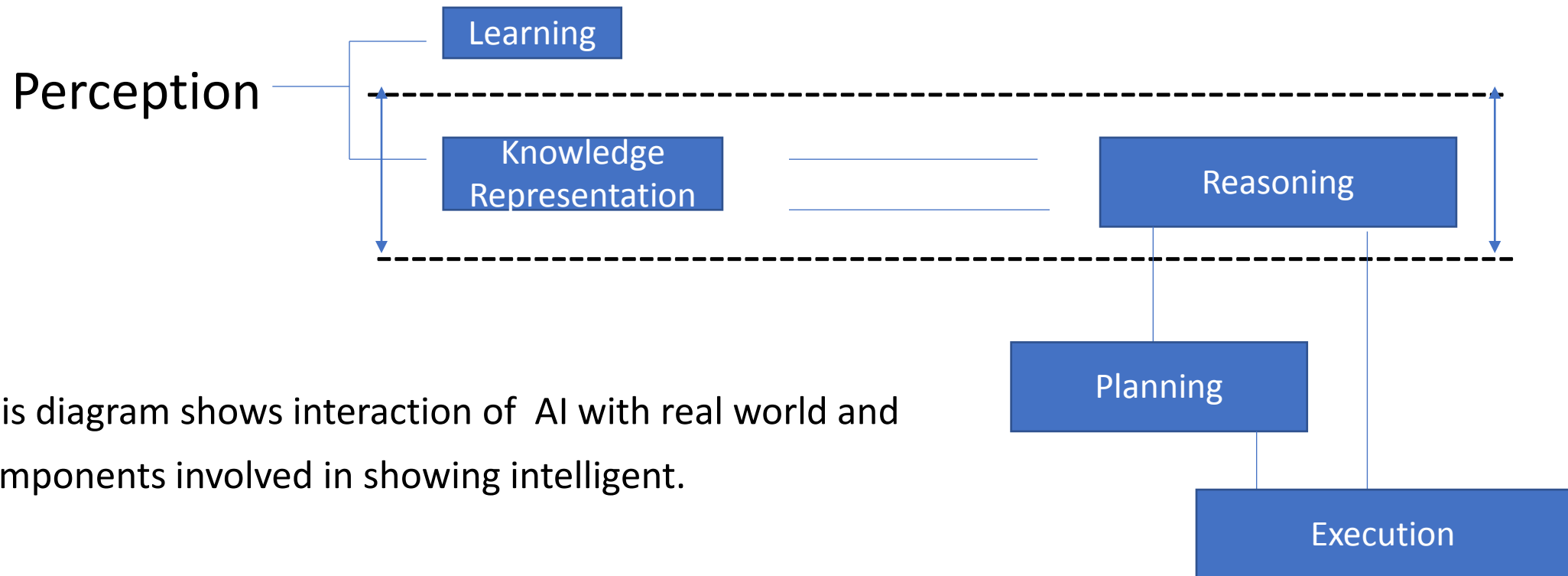
Objects, events, performance, facts, metaknowledge, and knowledgebase.

Different types of knowledge:

- **Declarative knowledge:** This includes concepts, facts, and objects.
- **Structural knowledge:** It defines basic problem solving knowledge that describes the relationship between the concepts and objects.
- **Procedural knowledge:** It is responsible for knowing how to define something and includes rules, strategies, and procedures.
- **Meta knowledge:** Defines the knowledge about other types of knowledge or data.
- **Heuristic knowledge:** This represents some experts knowledge in the field or subject.

Cycle of Knowledge representation

Artificial Intelligent system usually consist of various concepts to display intelligent behaviour.

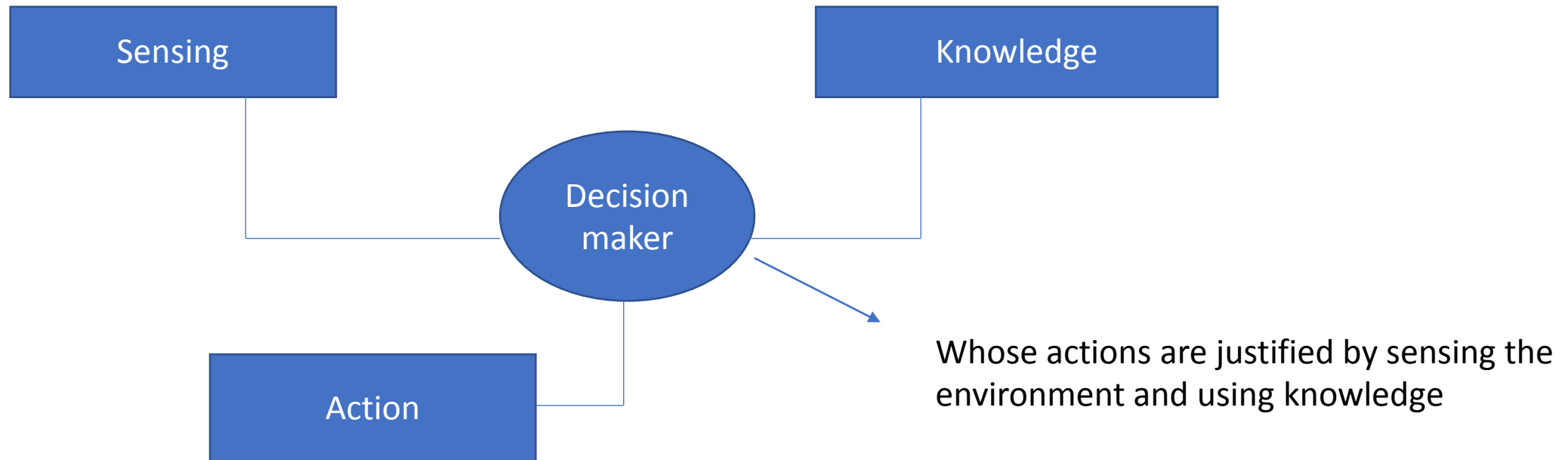


This diagram shows interaction of AI with real world and components involved in showing intelligent.

- **PERCEPTION:** It retrieves the data or information from the environment and finds the source of noises and also checks if the AI was damaged by anything and also sends to respond when any sense has been detected.
- **LEARNING:** Learns from the captured data by perception, focuses on self improvement to learn new things, requires knowledge acquisition, past searches, and inferences.
- **KNOWLEDGE REPRESENTATION:** Human likes intelligent in machine. Simply, it represents all about understanding intelligent.
- **REASONING:** Instead of understanding knowledge or building from bottom-up, the main goal is to understand intelligent behaviour from top-down and focuses on what an intelligent system need to know in order to behave intelligently.
- **PLANNING & EXECUTION:** This depend on the analysis of knowledge representation and reasoning. (Planning includes initial state and finding the every conditions and facts)

Relationship Between Knowledge and Intelligence

- Knowledge plays a vital role in the real world.



- But assume what happens if we remove knowledge part. (It could not be able to display any intelligent behaviour)

Techniques (other knowledge representation schemes)

- **4 techniques:**

1. Logic representation

2. Semantic network programming

- IS-A relation (inheritance)
- Kind of relation

3. Frame representation

4. Production rules

Requirements of Knowledge Representation:

- 1. Representation Accuracy:** It is defined as that it should represent all kind of required knowledge.
- 2. Inferential Adequacy:** It is described as that it should be able to manipulate the representation structure to produce new knowledge corresponding to the existing structure.
- 3. Inferential Efficiency:** It is the ability to direct the inferential knowledge mechanism into the most productive system.
- 4. Acquisitional Efficiency:** It is the ability to acquire new knowledge easily using automatic methods.

Knowledge Representation Issues

- The main goal is to facilitate the knowledge.
- The following are the some of the issues where represents the knowledge.
 1. Important attributes.
 2. Relationship among attributes.
 3. Choosing granularity.
 4. Set of objects.
 5. Finding the structure.

Important Attributes:

- In this any attributes of object that may occur in every problem domain?
- There might be a chance of occurring 2 attributes namely
 1. Instance.
 2. Is-of

The above 2 attributes are important to represent the knowledge because they help in inheritance.

Relationship Among Attributes:

This type of attributes may suspect that is there any important relation between different attributes of objects?

There are four properties define to have relation between different attribute of objects:

1. Inverse
2. Existence in an hierarchy

This defines in terms of generalization and specialization.

3. Reasoning for a value.
4. Single value attribute.

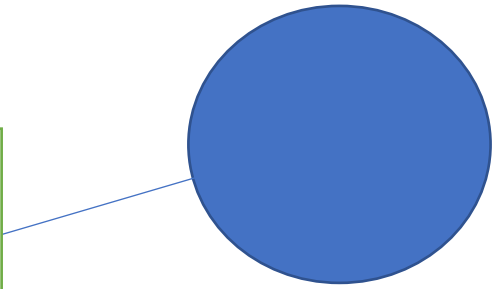
Choosing Granularity:

In this we define up to what level the knowledge can be represented and what are the primitives?

Set of Objects

In this we define how set of objects can be represented?

Using universal quantifiers objects can be represented.



If the data
doesn't exist in
the knowledge

Finding the Structure:

It is defined how to search for relevant data structure or according to particular situation, fetch or get the knowledge.

Example:

In the knowledge base if one of the data that does not exist, then how we are going to find the structure to add the data or to get the data.

So, generally in order to find the structure of any data it includes:

1. How to perform an initial selection.
2. How to fill the appropriate details in current situation.
3. How to find better structure.
4. When to create and remember new structure.

Types of Reasoning

In artificial intelligence, reasoning can be divided into the following categories:

- Deductive reasoning
- Inductive reasoning
- Abductive reasoning
- Common Sense Reasoning
- Monotonic Reasoning
- Non-monotonic Reasoning



1. Deductive reasoning:

Deductive reasoning is deducing new information from logically related known information. It is the form of valid reasoning, which means the argument's conclusion must be true when the premises are true.

Deductive reasoning is a type of propositional logic in AI, and it requires various rules and facts. It is sometimes referred to as top-down reasoning, and contradictory to inductive reasoning.

In deductive reasoning, the truth of the premises guarantees the truth of the conclusion.

Deductive reasoning mostly starts from the general premises to the specific conclusion, which can be explained as below example.

Example:

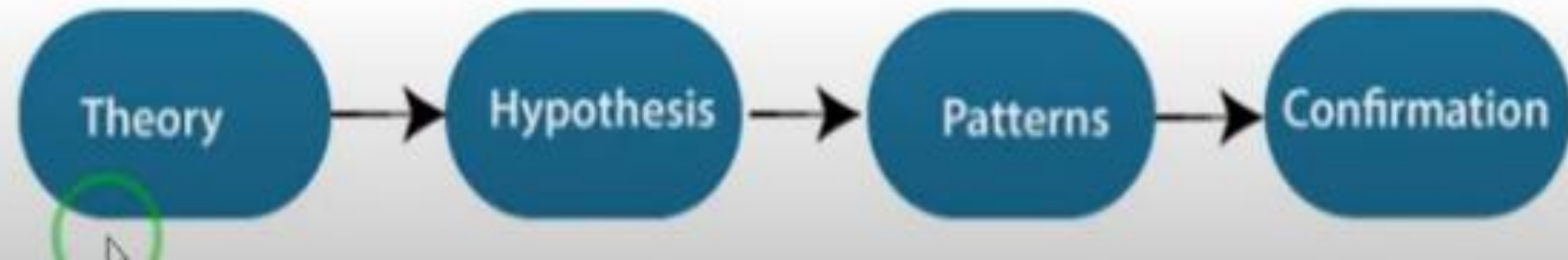
Premise-1: All the human eats veggies



Premise-2: Suresh is human.

Conclusion: Suresh eats veggies.

The general process of deductive reasoning is given below:



Inductive reasoning

Inductive reasoning is a form of reasoning to arrive at a conclusion using limited sets of facts by the process of generalization. It starts with the series of specific facts or data and reaches to a general statement or conclusion.

Inductive reasoning is a type of propositional logic, which is also known as cause-effect reasoning or bottom-up reasoning.

In inductive reasoning, we use historical data or various premises to generate a generic rule, for which premises support the conclusion.

In inductive reasoning, premises provide probable supports to the conclusion, so the truth of premises does not guarantee the truth of the conclusion.

Example:

Premise: All of the pigeons we have seen in the zoo are white.

Conclusion: Therefore, we can expect all the pigeons to be white.



3. Abductive reasoning:

Abductive reasoning is a form of logical reasoning which starts with single or multiple observations then seeks to find the most likely explanation or conclusion for the observation.

Abductive reasoning is an extension of deductive reasoning, but in abductive reasoning, the premises do not guarantee the conclusion.

Implication: Cricket ground is wet if it is raining

Axiom: Cricket ground is wet.

Conclusion It is raining.

4. Common Sense Reasoning

Common sense reasoning is an informal form of reasoning, which can be gained through experiences.

Common Sense reasoning simulates the human ability to make presumptions about events which occurs on every day.

It relies on good judgment rather than exact logic and operates on **heuristic knowledge** and **heuristic rules**.

Example:

1. **One person can be at one place at a time.**
2. **If I put my hand in a fire, then it will burn.**

The above two statements are the examples of common sense reasoning which a human mind can easily understand and assume.

5. Monotonic Reasoning:

In monotonic reasoning, once the conclusion is taken, then it will remain the same even if we add some other information to existing information in our knowledge base. In monotonic reasoning, adding knowledge does not decrease the set of prepositions that can be derived.

To solve monotonic problems, we can derive the valid conclusion from the available facts only, and it will not be affected by new facts.

Monotonic reasoning is not useful for the real-time systems, as in real time, facts get changed, so we cannot use monotonic reasoning.

Monotonic reasoning is used in conventional reasoning systems, and a logic-based system is monotonic.

Any theorem proving is an example of monotonic reasoning.

Example:

- **Earth revolves around the Sun.**

It is a true fact, and it cannot be changed even if we add another sentence in knowledge base like, "The moon revolves around the earth" Or "Earth is not round," etc.

3

Advantages of Monotonic Reasoning:

- In monotonic reasoning, each old proof will always remain valid.
- If we deduce some facts from available facts, then it will remain valid for always.

Disadvantages of Monotonic Reasoning:

- We cannot represent the real world scenarios using Monotonic reasoning.
- Hypothesis knowledge cannot be expressed with monotonic reasoning, which means facts should be true.
- Since we can only derive conclusions from the old proofs, so new knowledge from the real world cannot be added.

6. Non-Monotonic Reasoning:

In Non-monotonic reasoning, some conclusions may be invalidated if we add some more information to our knowledge base.

Logic will be said as non-monotonic if some conclusions can be invalidated by adding more knowledge into our knowledge base.

Non-monotonic reasoning deals with incomplete and uncertain models.

"Human perceptions for various things in daily life, "is a general example of non-monotonic reasoning.

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

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

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

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

Advantages of Non-monotonic reasoning:

- For real-world systems such as Robot navigation, we can use non-monotonic reasoning.
- In Non-monotonic reasoning, we can choose probabilistic facts or can make assumptions.

Disadvantages of Non-monotonic Reasoning:

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

Other Knowledge Representation Schemes

- **Logic representation (prepositional and predicate)**

LR is a language with some definite rules which deal with proposition and has no ambiguity in representation. It represents a conclusion based on various conditions and place down some important common rules and also it consists of precisely defined syntax and semantics which supports the sound inference.

Each sentence can be translated into sentence using syntax and semantics.

- **Semantic Network**

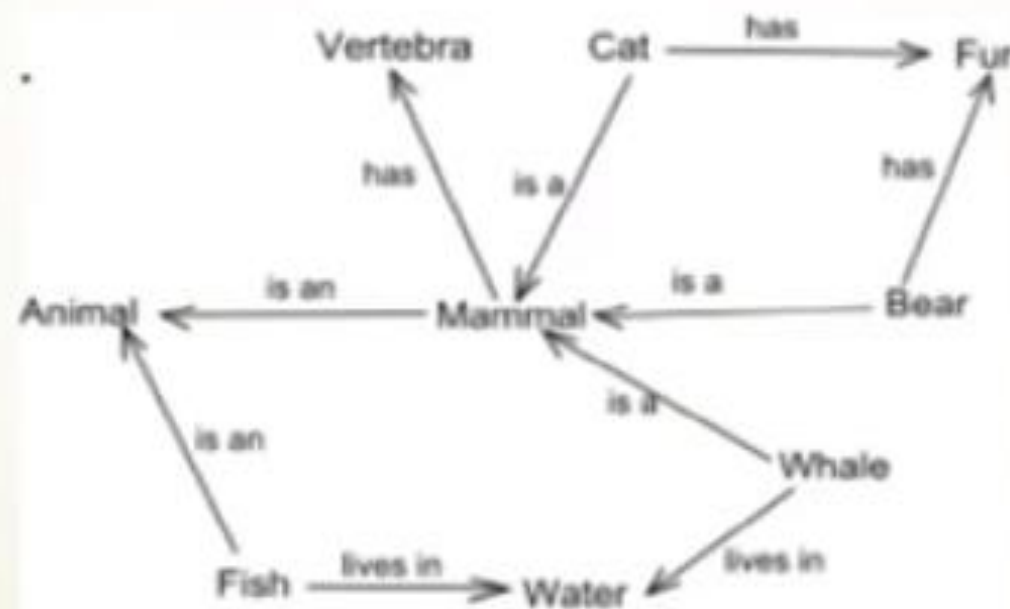
When do you say a particular sentence or statement is meaningful?

- A sentence is meaningful when we can really understand it and map it to some of the known concepts of the real world in which we live or see or can visualize or realize.
- Knowledge can be represented as a network of different concepts.
- If we consider semantic net as a knowledge representation method or scheme then it must also have some particular inference mechanism by which it can utilize this representation to infer new things to answer different questions.

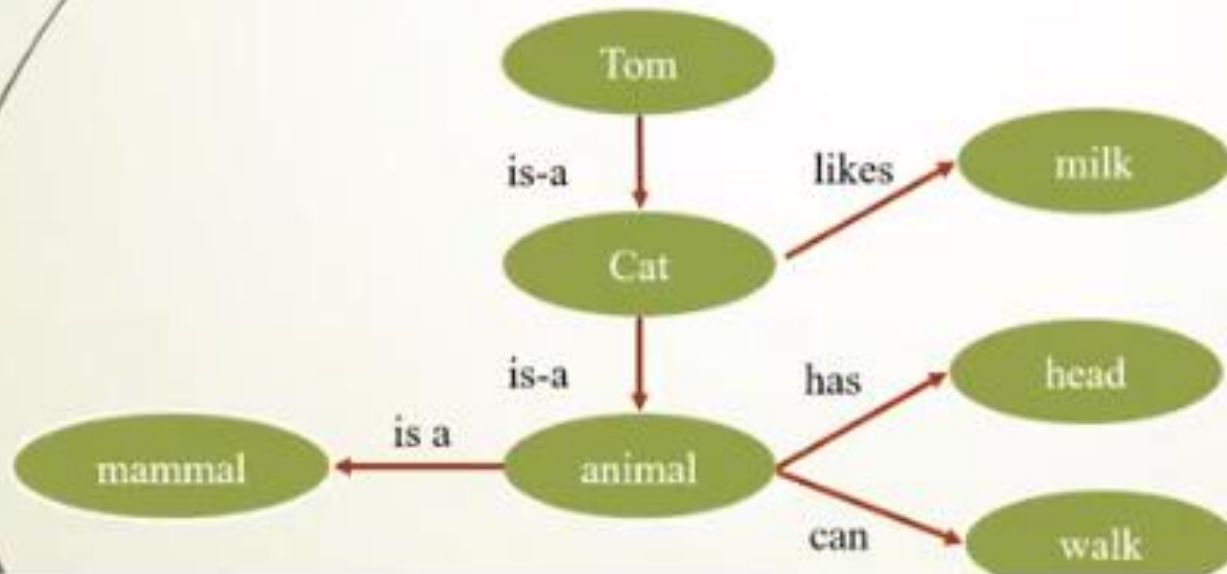
- First introduced by Quillian in the late-60s

M. Ross Quillian. "Semantic Memories", In M. M. Minsky, editor, *Semantic Information Processing*, pages 216-270. Cambridge, MA: MIT Press, 1968

- Developed a computational model which represented "concepts" as *Hierarchical networks*

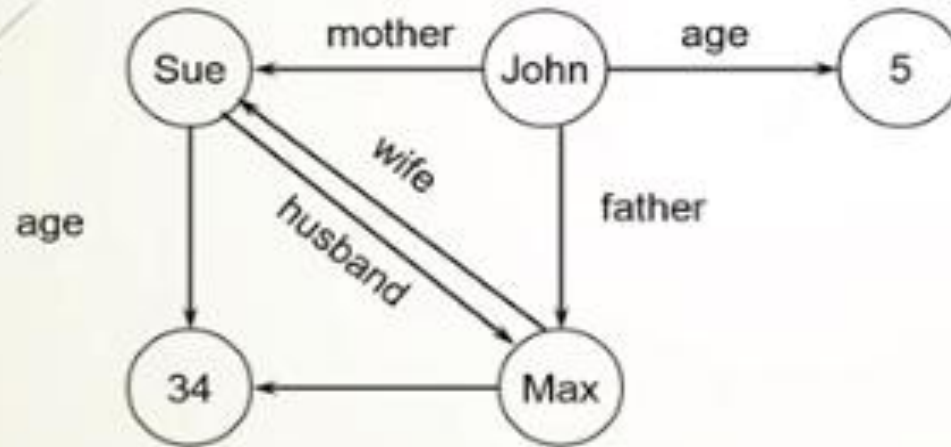


- The idea is that we can **store** our **knowledge** in the form of a **graph**, with **nodes** representing objects in the world, and **arcs** representing relationships between those objects.
- concepts can be represented as hierarchies of inter connected concept nodes (e.g. mammal, cat, tom)
- A concept may have a number of associated attributes at a given level.



Nodes and Arcs

- Nodes represent objects: name, attribute or attribute value
- Arcs define binary relations which hold between objects denoted by the nodes.



mother (john, sue)
age (john, 5)
wife (sue, max)
age (max, 34)
...

Components of semantic network

Semantic network representation consists of 4 parts

- ▀ **Lexical**: which symbols are allowed in the representation's vocabulary
- ▀ **Structural**: describes constraints on how the symbols can be arranged
- ▀ **Procedural**: specifies the access procedures (to create, modify, answer questions)
- ▀ **Semantic**: establishes the way of associating the meaning

nodes – denoting objects

links – denoting relations
between objects

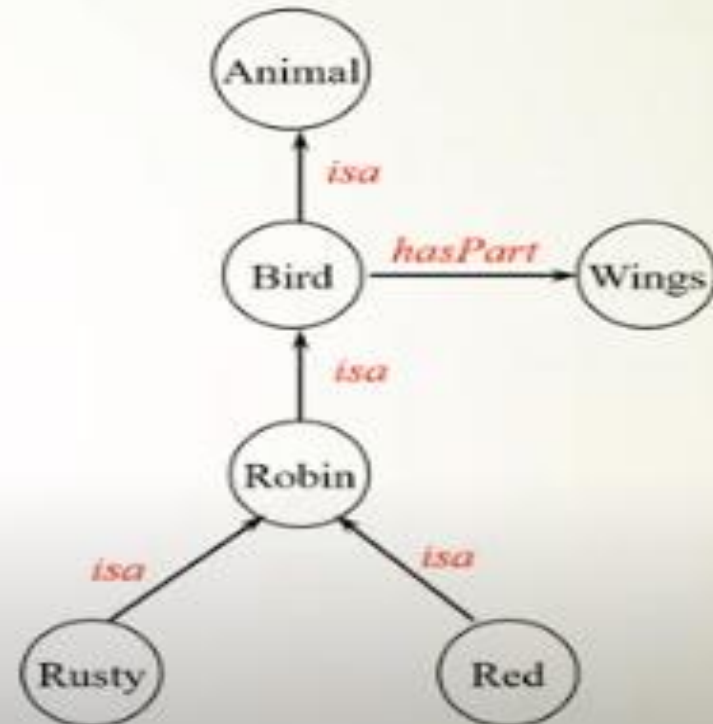
labels – denoting particular
objects and relations

- In simple words semantic network representation works as an alternate for predicate logic for known representations.
- In this we can represent the knowledge in the form of graphical networks.
- **Consists of 2 types of relations:**

IS-A relation (inheritance)

Other kind of relation (Non binary relation)

- Inheritance is one of the main kind of reasoning done in semantic nets
- The **ISA** (is a) relation is often used to link a class and its superclass.
- Some links (e.g. **haspart**) are inherited along **ISA** paths



- Frame Representation

- Variance to semantic network
- I
- A popular method to represent the facts in Expert System
- The data structure represent the knowledge about the concept or object.
- Resemble the record and structure in data structure.
- Provide a natural way in representing the structured knowledge.

Basic Frame Design Structure

I

Frame Name :

CLASS :

Properties

Prop1	Value1
Prop2	Value2
:	:
:	:

Example:

Basic Frame Design Structure

Frame Name : Tweety

CLASS : Bird

Properties

Color	Yellow
Eats	Worms
No-Wings	1
Flies	False
Hungry	Unknown
Activity	Unknown
Lives	Cage

- Provide ways to manage the knowledge in slot that contains characteristic and attribute
- Consists of 2 basic elements
 - **slot** – set of attributes to describe the object that represent the frame
 - **facet** – ^Isubslot to describe knowledge or action for the attribute in the slot.

- Need to document clearly the information related to the model
- Provide a way to limit the value for the attribute
- Provide information modularity – system expansion and maintenance

Facet Types :

- **VALUE** – value for slot: example: colour slot, the value is either blue, red or yellow
- **TYPE** – data/value that assign to the slot, example number of wings must be in numeric.
- **DEFAULT** – the initial value for the slot, example the default value for the number of wings is two unless stated otherwise.
- **CONSTRAINT/RANGE**- the range for the value, example 0- 100.

Example for Frame of car

Value, Range and Default

Frame: CAR

Specialization of: LAND VEHICLE

Body: Steel

Windows: glass

Fuel Remaining: ₁

Range:

Default: none

(empty, 1/4 tank, 1/2 tank, full)

VALUE

CONSTRAINT
/RANGE

DEFAULT

- **Instance frame:** Instance frame inherits information from class frame. We can give unique properties for instance.
- Inherits the behavior: one instance also inherits action

Class frame

Frame Name :

CLASS :

Properties

Color	Unknown
Eats	Worms
No-Wings	2
Flies	True
Hungry	Unknown
Activity	Unknown

Instance frame

Frame Name :

CLASS :

Properties

Color	Yellow
Eats	Worms
No-Wings	2
Flies	if-needed: check no-wings
Hungry	Unknown
Activity	if-changed: hungry=TRUE value=make noise
Lives	Cage

- **Production rules:**

A production system is based on a set of rules about behavior. These rules are a basic representation found helpful in expert systems, automated planning, and action selection. It also provides some form of artificial intelligence.

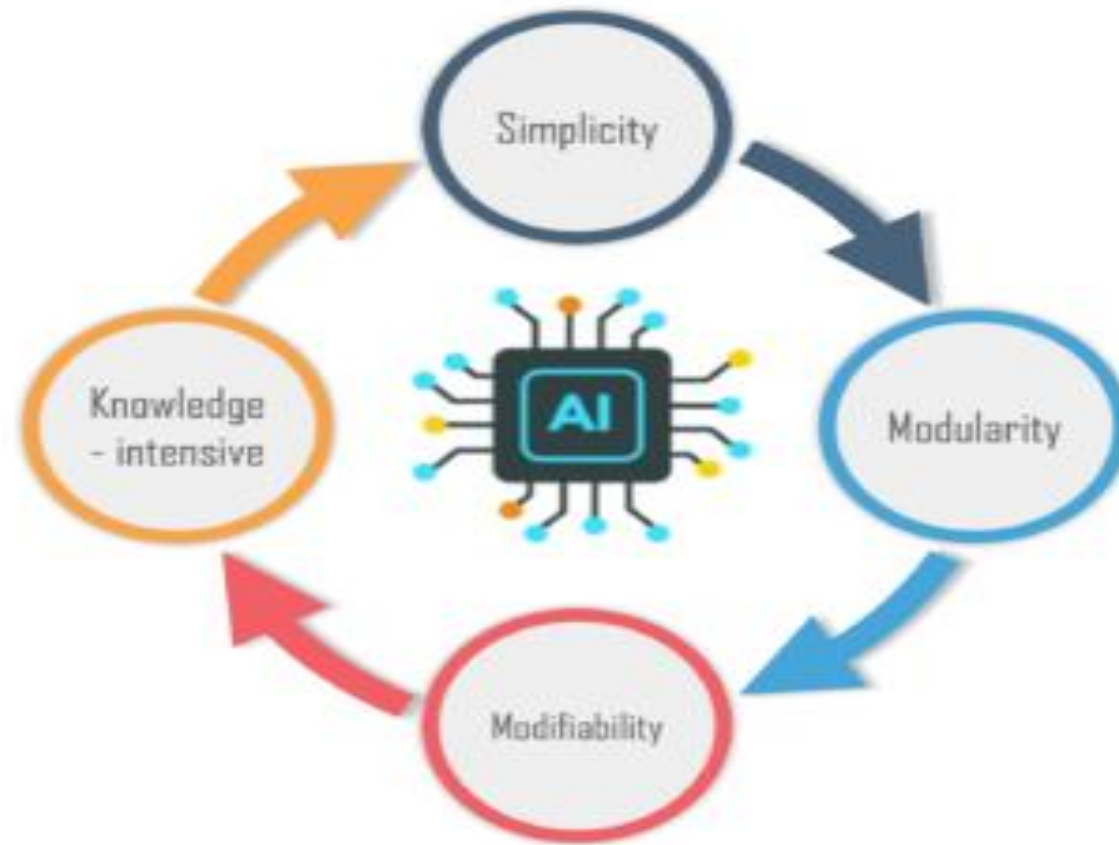
What is Production System?

Production system or production rule system is a computer program typically used to provide some form of artificial intelligence, which consists primarily of a set of rules about behavior but it also includes the mechanism necessary to follow those rules as the system responds to states of the world.

Components of Production System

- The major components of Production System in Artificial Intelligence are:
- **Global Database:** The global database is the central data structure used by the production system in Artificial Intelligence.
- **Set of Production Rules:** The production rules operate on the global database. Each rule usually has a precondition that is either satisfied or not by the global database. If the precondition is satisfied, the rule is usually be applied. The application of the rule changes the database.
- **A Control System:** The control system then chooses which applicable rule should be applied and ceases computation when a termination condition on the database is satisfied. If multiple rules are to fire at the same time, the control system resolves the conflicts.

Features of Production System



The main features of the production system include:

1. **Simplicity:** The structure of each sentence in a production system is unique and uniform as they use the “IF-THEN” structure. This structure provides simplicity in knowledge representation. This feature of the production system improves the readability of production rules.
2. **Modularity:** This means the production rule code the knowledge available in discrete pieces. Information can be treated as a collection of independent facts which may be added or deleted from the system with essentially no deleterious side effects.
3. **Modifiability:** This means the facility for modifying rules. It allows the development of production rules in a skeletal form first and then it is accurate to suit a specific application.
4. **Knowledge-intensive:** The knowledge base of the production system stores pure knowledge. This part does not contain any type of control or programming information. Each production rule is normally written as an English sentence; the problem of semantics is solved by the very structure of the representation.

Control/Search Strategies

- How would you decide which rule to apply while searching for a solution for any problem? There are certain requirements for a good control strategy that you need to keep in mind, such as:
- The first requirement for a good control strategy is that it should **cause motion**.
- The second requirement for a good control strategy is that it should be **systematic**.
- Finally, it must be **efficient** in order to find a good answer.

Production System Rules

Production System rules can be classified as:

- **Deductive Inference Rules**
- **Abductive Inference Rules**
- You can represent the knowledge in a production system as a set of rules along with a control system and database. It can be written as:
- ***If(Condition) Then (Condition)***
- The production rules are also known as condition-action, antecedent-consequent, pattern-action, situation-response, feedback-result pairs.

Classes of Production System

There are four major classes of Production System in Artificial Intelligence:

- Monotonic Production System
- Partially Commutative Production System
- Non-Monotonic Production Systems
- Commutative Systems

Reasoning under uncertainty

Reasoning under Uncertainty

- **Probability theory** provides the **basis for our treatment of systems that reason under uncertainty.**
- **Utility Theory** provides **ways and means of weighing up the desirability of goals** and the likelihood of achieving them.
 - Actions are no longer certain to achieve goals.
- **Probability theory and utility theory** put together constitute decision theory; **take decisions within uncertain domain.**
 - Build rational agents for uncertain worlds.

- Basics of probability theory, including the representation language for uncertain beliefs.
 - Acting Under Uncertainty
 - Rational Decisions
 - Basic Probability Notation
 - Bayes' Rule
- Belief networks, a powerful tool for representing and reasoning with uncertain knowledge.

- Within a logical-agent approach, **agents almost never have access to the whole truth** about their environment.
 - Some **sentences can be ascertained directly from the agent's percepts**, and others can be **inferred from current and previous percepts** together with knowledge about the environment.
 - However, **for almost every case, there will be important questions to which the agent cannot find a categorical answer.**
- The **agent must therefore act under uncertainty.**
- Uncertainty can also arise because of **incompleteness and incorrectness in the agent's understanding** of the properties of the environment.

Handling uncertain knowledge

Trying to use first-order logic to cope with complex domain like medical diagnosis fails for three main reasons:

- 1. Laziness:** It is too much work to list the complete set of antecedents or consequents needed to ensure an exception less rule, and too hard to use the enormous rules that result.
- 2. Theoretical ignorance:** Expertise of the area may not be sufficient to have complete theory for the domain.
- 3. Practical ignorance:** Even if we know all the rules, we may be uncertain about particular cases because all the necessary tests have not or cannot be run.

- **Agent's knowledge can at best provide only a degree of belief** in the relevant sentences.
 - True for medical domain, as well as most other judgmental domains: law, business, design, automobile repair, gardening, dating, and so on
- Dealing with **degrees of belief is through probability theory**, which assigns a numerical degree of belief between 0 and 1 to sentences.
- **Probability provides a way of summarizing the uncertainty** that comes from our laziness and ignorance.

Uncertainty and rational decisions

- To make such choices, an agent must first have preferences, between possible outcomes of the plans.

- Use the utility theory to represent and reason with "preference"

- 1. Preference:** options, choices, what is more preferred.

- 2. Outcome:** Completely specified state.

- 3. Utility Theory:** "The quality of being useful" - theory says that every state has a degree of usefulness, or utility, to an agent and that the agent will prefer states with higher utility.

Basic probability notation

- Notation for **describing degrees of belief.**
 - Formal language for representing and reasoning with uncertain knowledge.
- The version of probability theory we present uses an extension of **propositional logic for its sentences.**
- The **dependence on experience** is reflected in the syntactic distinction between
 - **prior probability statements**, which apply before any evidence is obtained, and
 - **conditional probability statements**, which include the evidence explicitly.

□ Sample Space: The **set of all possible worlds i.e., all possible outcomes** is referred to as sample space.

□ Notation

■ Ω - Sample Space

■ ω - An element in the sample space

■ φ - An event or a proposition

□ An event φ is a subset of sample space Ω : $\varphi \subseteq \Omega$

Example: Two Dice adding up to 11 is an event

$$\varphi = \{ (5,6) , (6,5) \}$$

Probability model

- A probability model associates a numerical probability $P(\omega)$ with each possible world:
 - Every possible world must have a probability between 0 and 1
 - Total probability of the set of all possible worlds is 1
- Unconditional Probability
 - Unconditional Probability is when you **don't consider any other information** except for the object in question.
 - Example – Two dices – red and blue; consider only one – red.
- Conditional Probability
 - In Conditional Probability **we have evidence i.e., extra information** already revealed.
 - Example – Rolling two dices – one is a 6; Sum cannot be 5! ✓

Prior probability

Use notation **$P(A)$** for the **unconditional or prior probability** that proposition A is true.

For example, if *Fever* denotes the proposition that a particular patient has a fever,

✓ $P(\text{Fever}) = 0.1$

$P(A)$ can only be used when there is no other information. As soon as some new information B is known, we have to reason with the conditional probability of A given B instead of $P(A)$.

means that **in the absence of any other information**, the agent will assign a probability of 0.1 (a 10% chance) to the event of the patient having a fever.

Random variables

- The proposition that is the subject of a probability statement can be represented by a proposition symbol, as in the $P(A)$ example.
- **Propositions can also include equalities involving random variables.**
- Every Random Variable has a domain - a set of possible values that it can take.
 - For example, let's say we have the random variable Total that calculates the sum of two dice:
 - Then the domain is the set $\{2, \dots, 12\}$
 - A Boolean random variable has the domain $\{\text{True}, \text{False}\}$

- For propositions involving random variables; For example, if we are **concerned about the random variable Weather**, we might have
 - ✓ $P(\text{Weather}=\text{Sunny}) = 0.7$
 - ✓ $P(\text{Weather}=\text{Cloudy}) = 0.08$
- Can view **proposition symbols as random variables** as well, if we assume that they have a domain [true,false).
 - For example: Expression ✓ $P(\text{Fever})$ can be viewed as shorthand for ✓ $P(\text{Fever} = \text{true})$.
 - Similarly, ✓ $P(\neg \text{Fever})$ is shorthand for $P(\text{Fever} = \text{false})$.

Probability Distribution

- ❑ A probability distribution is when we want to talk about **all the possible values of a random variable**. Usually **indicated by a bold P**.
- ❑ A **Discrete Random Variable** is a random variable that takes a **finite number of distinct values**.
For example,
An expression such as $P(\text{Weather})$, denotes a vector of values for the probabilities of each individual state of the weather.
For example, we would write
 $\mathbf{P}(\text{Weather}) = (0.7, 0.2, 0.08, 0.02)$
- ❑ This statement defines a **probability distribution**.

Conditional probability

□ Conditional or posterior probabilities is expressed with the notation $P(A|B)$.

■ This is read as "the probability of A given that all we know is B."

Once the agent has obtained some evidence concerning the previously unknown propositions making up the domain, prior probabilities are no longer applicable.

□ For example

$$P(\text{Cavity}|\text{Toothache}) = 0.8$$

■ Indicates that if a patient is observed to have a toothache, and no other information is yet available, then the probability of the patient having a cavity will be 0.8.

- $P(X | Y)$ is a two-dimensional table giving the values of $P(X=x_i|Y = y_j)$ for each possible i, j .
- Conditional probabilities can be **defined in terms of unconditional probabilities.**

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

- This **equation can also be written as follows, which is called the product rule.**

The product rule is perhaps easier to remember: it comes from the fact that for A and B to be true, we need B to be true, then A to be true given B .

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

Axioms of probability

To define properly the semantics of statements in probability theory, we will need to describe how probabilities and logical connectives interact.

1. All probabilities are between 0 and 1.

$$0 < P(A) < 1$$

2. Necessarily true (i.e., valid) propositions have probability 1, and necessarily false (i.e., unsatisfiable) propositions have probability 0.

$$\checkmark P(\text{True}) = 1; \checkmark P(\text{False}) = 0$$

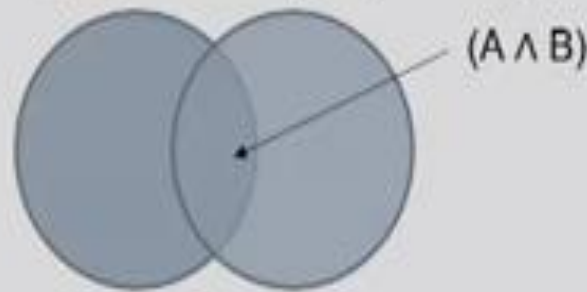
3. The probability of a disjunction is given by

$$\checkmark P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

To define properly the semantics of statements in probability theory, we will need to describe how probabilities and logical connectives interact.

3. The probability of a disjunction is given by

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$



The figure depicts each proposition as a set, which can be thought of as the set of all possible worlds in which the proposition is true.

The **total probability of $(A \vee B)$** is seen to be the **sum of the probabilities assigned to A and B** , but with **$P(A \wedge B)$ subtracted out so that those cases are not counted twice.**

□ From these three axioms, we can **derive all other properties of probabilities**.

For example,

If we let B be $\neg A$ in the last axiom, we obtain an expression for the **probability of the negation of a proposition** in terms of the probability of the proposition itself:

$$\checkmark P(A \vee \neg A) = P(A) + P(\neg A) - \checkmark P(A \wedge \neg A)$$

$$P(\text{True}) = P(A) + P(\neg A) - \checkmark P(\text{False})$$

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

$$\checkmark P(\neg A) = 1 - P(A)$$

Joint probability distribution

- Joint probability distribution **completely specifies an agent's probability assignments to all propositions** in the domain (both simple and complex).
- The joint probability distribution assigns probabilities to all possible atomic events.
 - An n-dimensional table with a value in every cell giving the probability of that specific state occurring.

✓

	Toothache	\neg Toothache
Cavity	0.04	0.06
\neg Cavity	0.01	0.89

□ Adding across a row or column gives the unconditional probability of a variable,

■ $P(\text{Cavity}) = 0.06 + 0.04 = 0.10.$

■ $P(\text{Cavity} \vee \text{Toothache}) = 0.04 + 0.01 + 0.06 = 0.11$

□ Conditional probabilities can be found from the joint,

■
$$P(\text{Cavity}|\text{Toothache}) = \frac{P(\text{Cavity} \wedge \text{Toothache})}{P(\text{Toothache})} = \frac{0.04}{0.04+0.01} = 0.8$$

Bayes' Rule

- Recall the **two forms of the product rules**

- ✓ $P(A \wedge B) = P(A|B) P(B)$

- ✓ $P(A \wedge B) = P(B|A) P(A)$

- Equating the two right-hand sides and dividing by $P(A)$, we get

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

- This **equation is known as Bayes' rule** (also Bayes' law or Bayes' theorem). This simple equation underlies all modern AI systems for probabilistic inference.

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

- Bayes' rule requires three terms
 - two prior probabilities and
 - a conditional probabilityto compute the fourth an conditional probability.
- In practice, Bayes' rule is useful, **we have good probability estimates for these three quantities** and need to compute the fourth.

Applying Bayes' rule: Example

A doctor knows that the disease **✓meningitis causes the patient to have a ✓stiff neck, say, ✓50% of the time.** The doctor also knows some unconditional facts: the **prior probability of a patient having meningitis is 1/50,000,** and the **prior probability of any patient having a stiff neck is 1/20.**

S be the proposition that the patient has a stiff neck

M be the proposition that the patient has meningitis.

$$P(S|M) = 0.5$$

$$✓P(M) = \frac{1}{50000} = 0.00002$$

$$✓P(S) = \frac{1}{20} = 0.05$$

$$P(M|S) = \frac{P(S|M)P(M)}{P(S)} = 0.0002$$

Notice that even though a stiff neck is strongly indicated by meningitis (probability 0.5), the probability of meningitis in the patient remains small.

This is because the prior on stiff necks is much higher than that for meningitis.

What is the probability of the disease that he has a stiff neck?

Using Bayes' rule: combining evidence

Suppose we have two conditional probabilities relating to cavities:

$$P(\text{Cavity}|\text{Toothache}) = 0.8$$

$$P(\text{Cavity}|\text{Catch}) = 0.95$$

What can a dentist conclude if her nasty steel probe catches in the aching tooth of a patient?

✓
$$P(\text{Cavity}|\text{Toothache} \wedge \text{Catch}) = \frac{P(\text{Toothache} \wedge \text{Catch}|\text{Cavity}) P(\text{Cavity})}{P(\text{Toothache} \wedge \text{Catch})}$$

Although it seems feasible to estimate conditional probabilities for n different individual variables, it is a daunting task to come up with numbers for n^2 pairs of variables.

- Application of Bayes' rule - simplified to a form that requires fewer probabilities in order to produce a result.

- The process of Bayesian updating incorporates evidence one piece at a time, modifying the previously held belief in the unknown variable.

$$P(\text{Cavity}|\text{Toothache}) = \frac{P(\text{Toothache}|\text{Cavity}) P(\text{Cavity})}{P(\text{Toothache})}$$

- When Catch is observed, we can apply Bayes' rule with **Toothache as the constant conditioning context.**

$$P(\text{Cavity}|\text{Toothache} \wedge \text{Catch}) = \frac{P(\text{Cavity}|\text{Toothache}) P(\text{Catch}|\text{Toothache} \wedge \text{Cavity})}{P(\text{Catch}|\text{Toothache})}$$

- In Bayesian updating, as new piece of evidence is observed, the belief in the unknown variable is multiplied by a factor that depends on the new evidence.

is not going to change the probability that the cavity is causing a toothache.

- Exploit **conditional independence** of Toothache and Catch given Cavity.
 - Given conditional independence, we can simplify the equation for updating.
- Combining many pieces of evidence may require assessing a large number of conditional probabilities.
- **Conditional independence** brought about by direct causal relationships in the domain **allows Bayesian updating to work effectively** even with multiple pieces of evidence.

Representing Knowledge in an uncertain domain

- There are various ways of representing uncertainty. Here we consider three different approaches, representing three different areas of uncertainty:
 1. **Probability theory**
 2. **Fuzzy logic**
 3. **Truth maintenance System**

Probability theory:

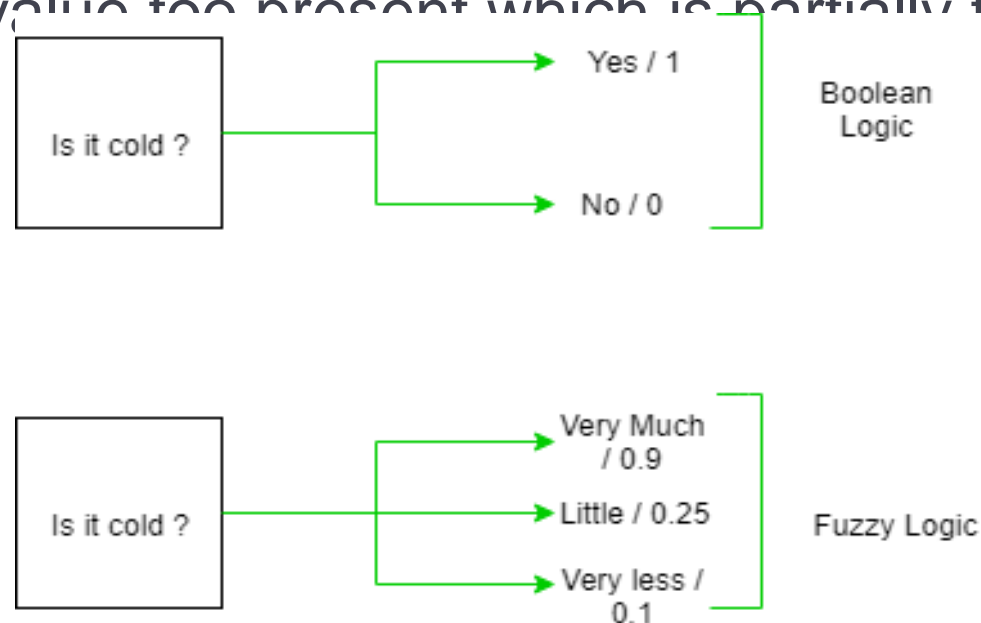
- Probabilistic assertions and queries are not usually about particular possible worlds, but about sets of them.
 - In probability theory, the set of all possible worlds is called the sample space. The Greek letter Ω (uppercase omega) is used to refer to the sample space, and ω (lowercase omega) refers to elements of the space, that is, particular possible worlds.
 - A fully specified probability model associates a numerical probability $P(\omega)$ with each possible world. The basic axioms of probability theory say that every possible world has a probability between 0 and 1 and that the total probability of the set of possible worlds is 1:
 - $0 \leq P(\omega) \leq 1$ for every ω and $\omega \in \Omega$
 - $P(\Omega) = 1$
- i. If a coin is flipped there is an equal chance of it landing on head side or tail side, consider H_1 is for heads and H_2 for tails. This scenario is expressed as $P(H_1)=0.5$ and $P(H_2)=0.5$.
- ii. The probability of 1st and 2nd toss both landing on heads is $0.5*0.5=0.25$.
- iii. We can write this as $P(H_1 \wedge H_2)=0.25$ and in general two independent events P and Q , $P(P \wedge Q)=P(P)*P(Q)$.

Fuzzy logic:

- In the existing expert systems, uncertainty is dealt with through a combination of predicate logic and probability-based methods.
- An alternative approach to the management of uncertainty is based on the use of fuzzy logic, which is the logic underlying approximate or, equivalently, fuzzy reasoning.
- e.g., most, many, few, not very many, almost all, infrequently, about 0.8, etc.
- In this way, fuzzy logic subsumes both predicate logic and probability theory, and makes it possible to deal with different types of uncertainty within a single conceptual framework.

The term **fuzzy** refers to things which are not clear or are vague. In the real world many times we encounter a situation when we can't determine whether the state is true or false, their fuzzy logic provides a very valuable flexibility for reasoning. In this way, we can consider the inaccuracies and uncertainties of any situation.

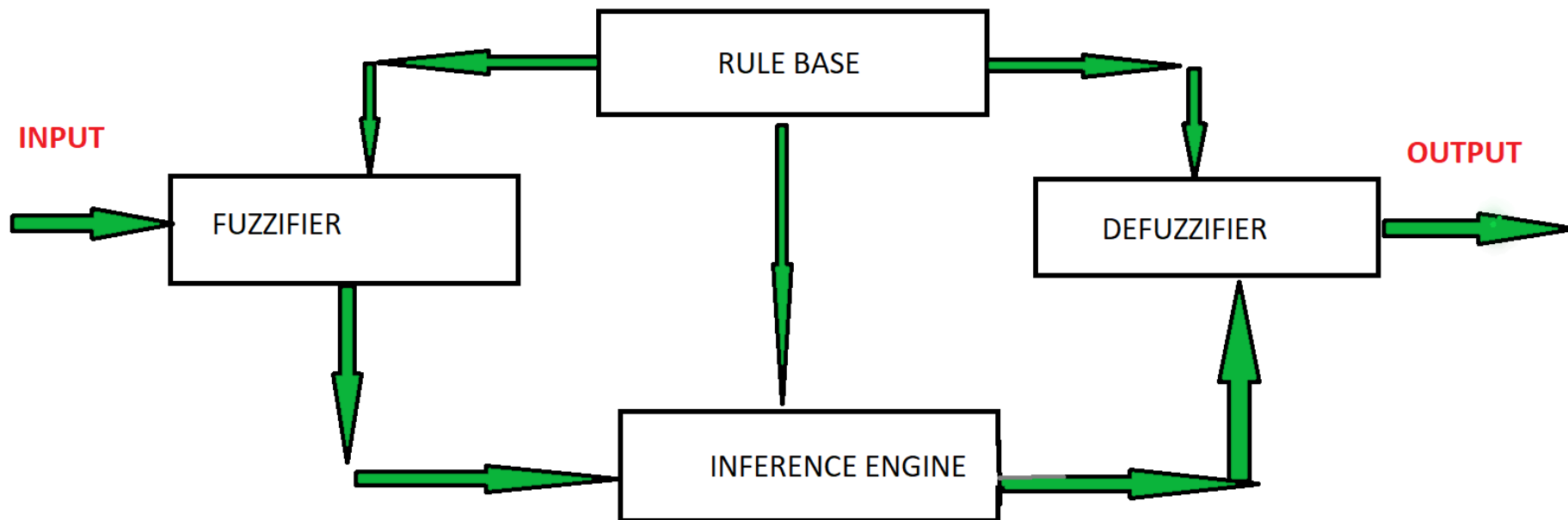
In boolean system truth value, 1.0 represents absolute truth value and 0.0 represents absolute false value. But in the fuzzy system, there is no logic for absolute truth and absolute false value. But in fuzzy logic, there is intermediate value too present which is partially true and partially false.



ARCHITECTURE

Its Architecture contains four parts :

- **RULE BASE:** It contains the set of rules and the IF-THEN conditions provided by the experts to govern the decision making system, on the basis of linguistic information. Recent developments in fuzzy theory offer several effective methods for the design and tuning of fuzzy controllers. Most of these developments reduce the number of fuzzy rules.
- **FUZZIFICATION:** It is used to convert inputs i.e. crisp numbers into fuzzy sets. Crisp inputs are basically the exact inputs measured by sensors and passed into the control system for processing, such as temperature, pressure, rpm's, etc.
- **INFERENCE ENGINE:** It determines the matching degree of the current fuzzy input with respect to each rule and decides which rules are to be fired according to the input field. Next, the fired rules are combined to form the control actions.
- **DEFUZZIFICATION:** It is used to convert the fuzzy sets obtained by inference engine into a crisp value. There are several defuzzification methods available and the best suited one is used with a specific expert system to reduce the error.



FUZZY LOGIC ARCHITECTURE

Truth maintenance System:

- To choose their actions, reasoning programs must be able to make assumptions and subsequently revise their beliefs when discoveries contradict these assumptions.
- The Truth Maintenance System (TMS) is a problem solver subsystem for performing these functions by recording and maintaining the reasons for program beliefs. Such recorded reasons are useful in constructing explanations of program actions and in guiding the course of action of a problem solver
- TMS are another form of knowledge representation which is best visualized in terms of graphs.
- It stores the latest truth value of any predicate. The system is developed with the idea that truthfulness of a predicate can change with time, as new knowledge is added or exiting knowledge is updated.
- It keeps a record showing which items of knowledge is currently believed or disbelieved.

Bayesian Networks:

- **Representing knowledge in an uncertain domain:**

- 1) Joint probability distribution

- 2) Bayes' rules allows unknown probabilities to be computed from known

What is Bayesian network (belief network)?

A Bayesian network falls under the category of probabilistic graphical modelling (PGM) technique that is used to compute uncertainties by using the concept of probabilities

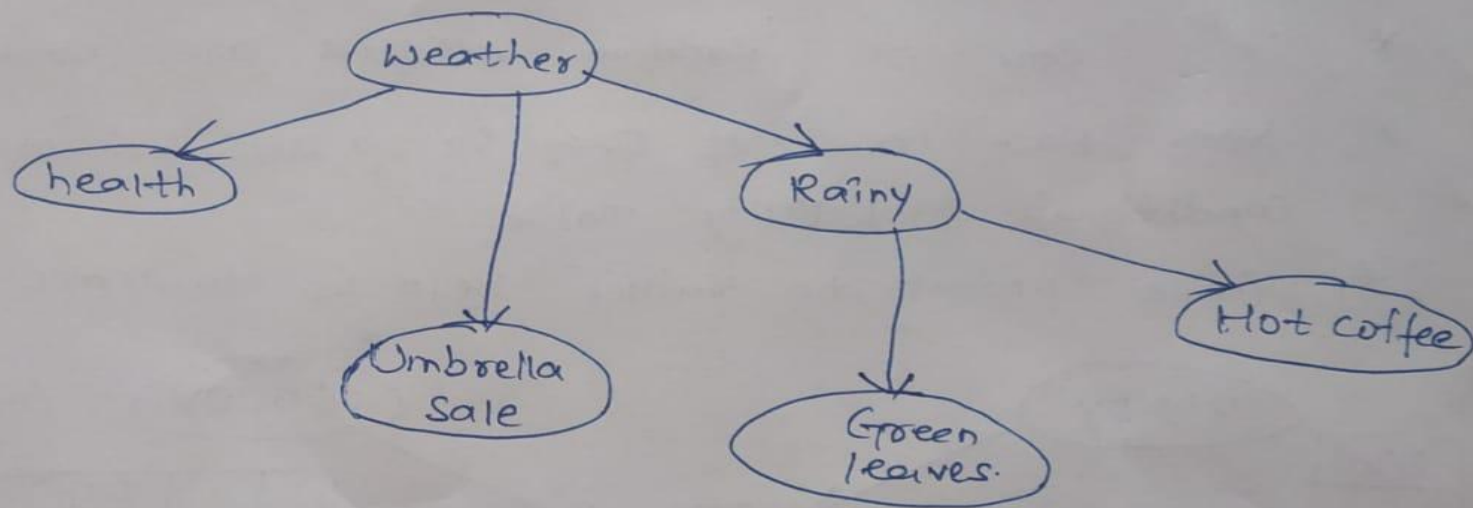
Bayesian Belief Network (BBN)

- A BBN is a probabilistic Graphical Model that represents conditional dependencies between Random Variables through a Directed Acyclic Graph (DAG).

- * The DAG, ^{graph} consists of nodes and arcs.
- * The nodes represent variables, which can be discrete (or) Continuous.
- * The arcs represent causal relationships between variables

An example:

An example:



* BBN's enable us to model and reason about uncertainty.

* BBN's represent the Joint probability Distribution.

Two types of probabilities comes to our Rescue.

- ① Joint probability
- ② conditional probability.

Joint Probability:

→ Joint Probability

→ conditional probability

$$P(\text{weather, health, Umbrella sale}) = P(\text{weather}) \times P(\text{health} | \text{weather}) \times P(\text{Umbrella sale} | \text{weather})$$

Generalising

$$P(x_1, x_2, x_3 \dots x_n) = \prod_{i=1}^n P(x_i | \text{Parent}(x_i))$$

denotes Belief

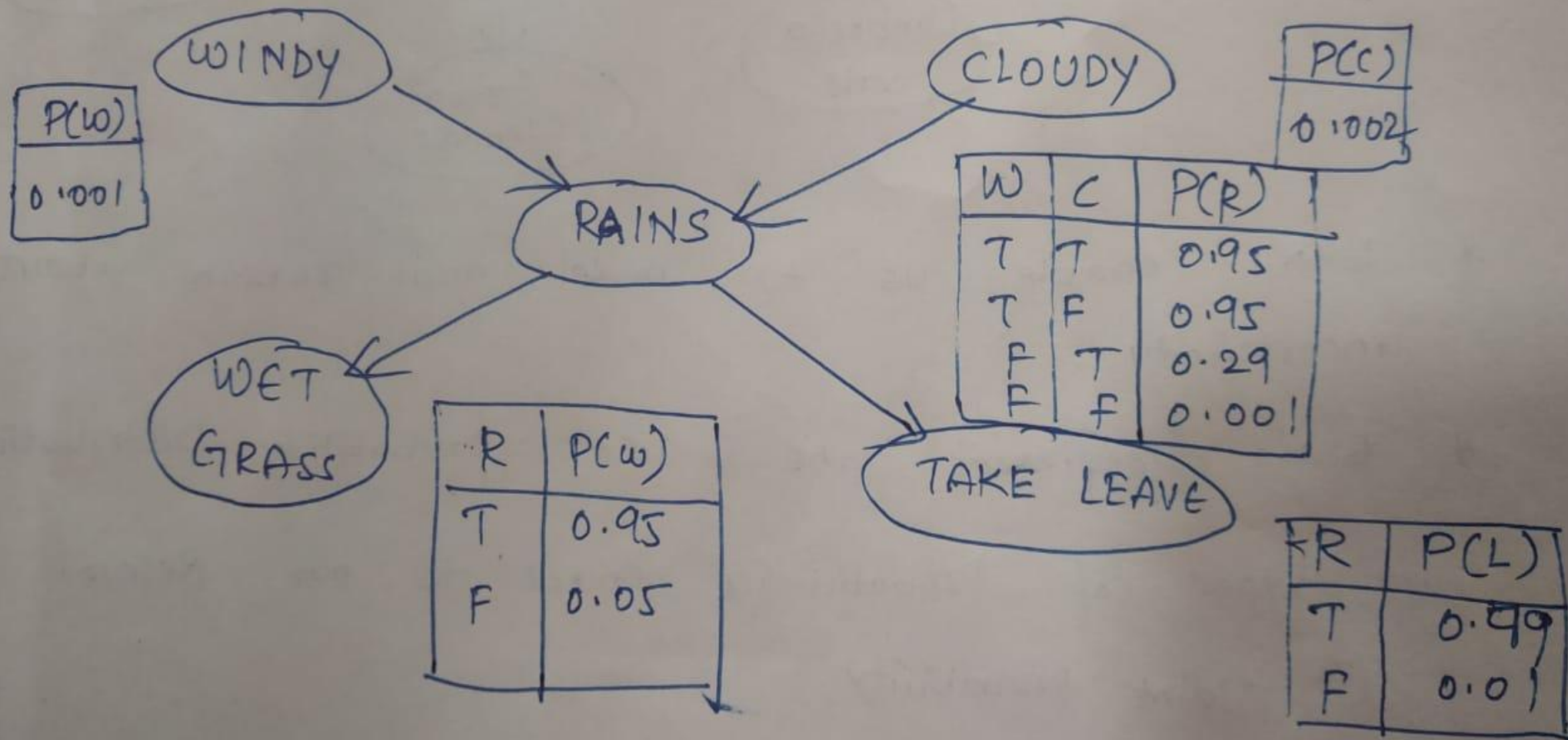
So, what these probabilities emit & how are they going to help i.e., flow

* These probabilities can help make an inference.

* Let's say the Random Variables are boolean

* ~~Let's~~ Each node of DAG is associated with a Conditional Probability Table.

* These probability values help us to make an inference



REDMI NOTE 5 PRO
MI DUAL CAMERA

Let's find the probability of having a wet grass?

$$P(w) = P(w|R) * P(R) + P(w|\bar{R}) * P(\bar{R}) \quad \text{--- ①}$$

$$= 0.9 * P(R) + 0.05 * P(\bar{R})$$

$$P(R) = P(R|w, C) * P(w \wedge C) + P(R|\bar{w}C) * P(\sim w \wedge C) + \\ P(R|w\bar{C}) * P(w \wedge \sim C) + P(R|\bar{w}\bar{C}) * P(\sim w \wedge \sim C) \quad \text{--- ②}$$

$$= 0.95 * 0.001 * 0.002 + 0.29 * 0.999 * 0.002 +$$

$$0.95 * 0.001 * 0.998 + 0.001 * 0.999 * 0.998$$

$$\boxed{P(R) = 0.00252}$$

$$P(\bar{R}) = P(\bar{R}|w, c) * P(w \wedge c) + P(\bar{R}|\bar{w}c) * P(\bar{w} \wedge c) + \\ P(\bar{R}|w\bar{c}) * P(w \wedge \bar{c}) + P(\bar{R}|\bar{w}\bar{c}) * P(\bar{w} \wedge \bar{c}) \quad \text{--- (3)}$$

$$(1-0.95) * 0.001 * 0.002 + (1-0.29) * 0.999 * 0.002$$

$$+ (1-0.95) * 0.001 * 0.998 + (1-0.001) * 0.999 *$$

$$0.998$$

$$\boxed{P(\bar{R}) = 0.99744}$$

$$P(w) = 0.9 * P(R) + 0.05 * P(\bar{R})$$

$$= 0.9 * 0.00252 + 0.05 * 0.99744.$$

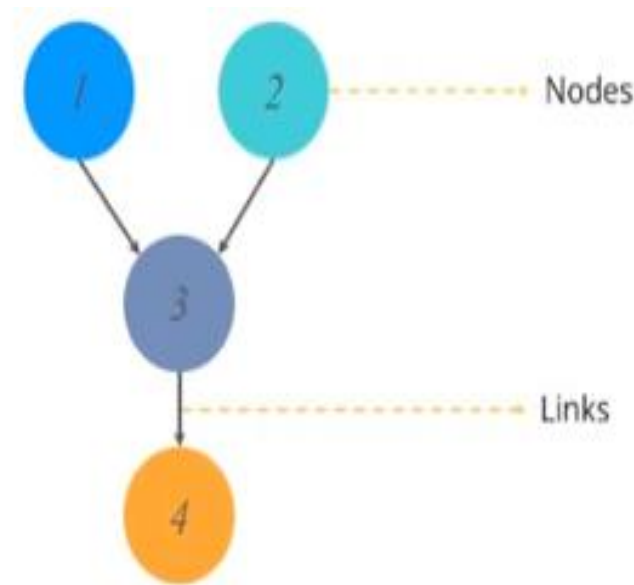
$$P(w) = 0.0521$$

- They are used to module uncertainties by using DAG

What is DAG (Directed Acyclic Graph)

A DAG is used to represent Belief/Bayesian/casual networks and like any other statistical graphs.

It contains nodes and links which shows relation between nodes.

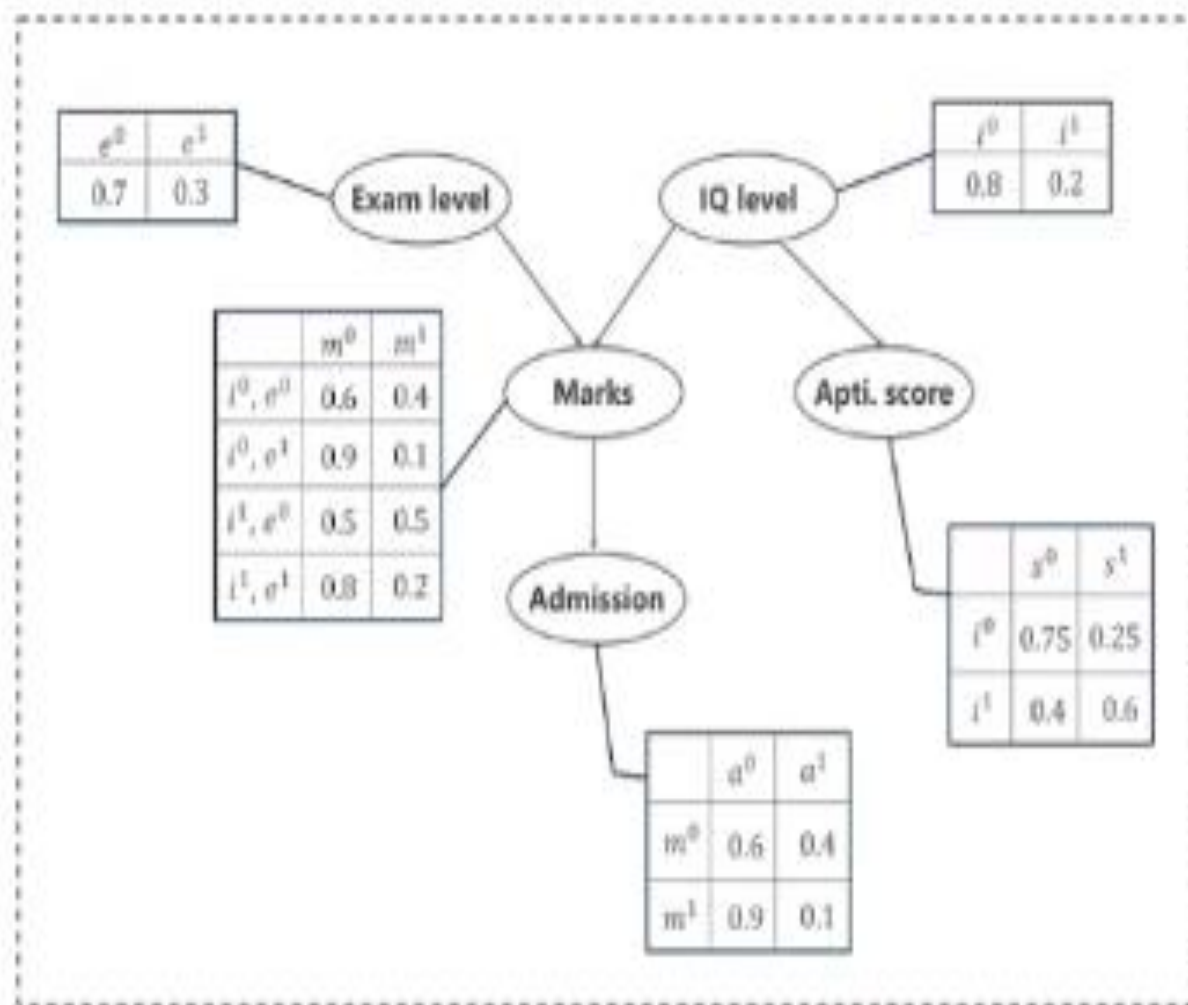


A DAG models the uncertainty of an event occurring based on the Conditional Probability Distribution (CPD) of each random variable

Create a Bayesian Network that will model the marks (m) of a student on his examination.

The marks will depend on:

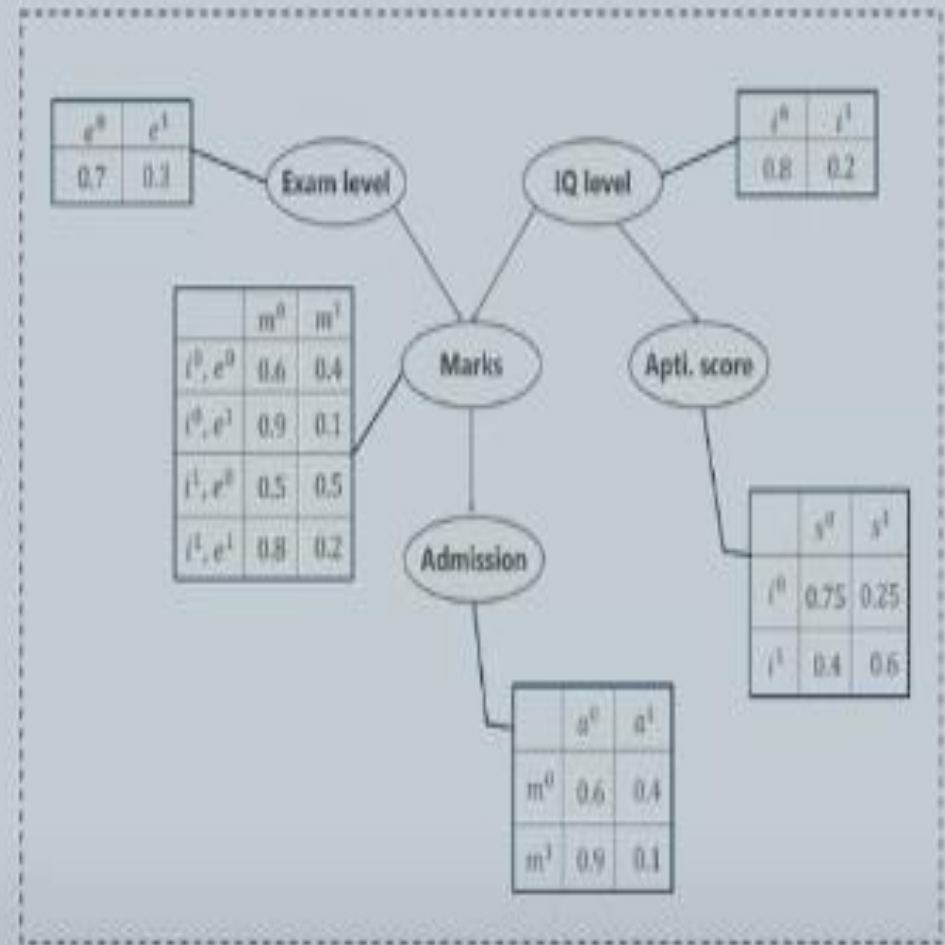
- **Exam level** (e): (difficult, easy)
- **IQ of the student** (i): (high, low)
- Marks \rightarrow **admitted** (a) to a university
- The IQ \rightarrow **aptitude score** (s) of the student

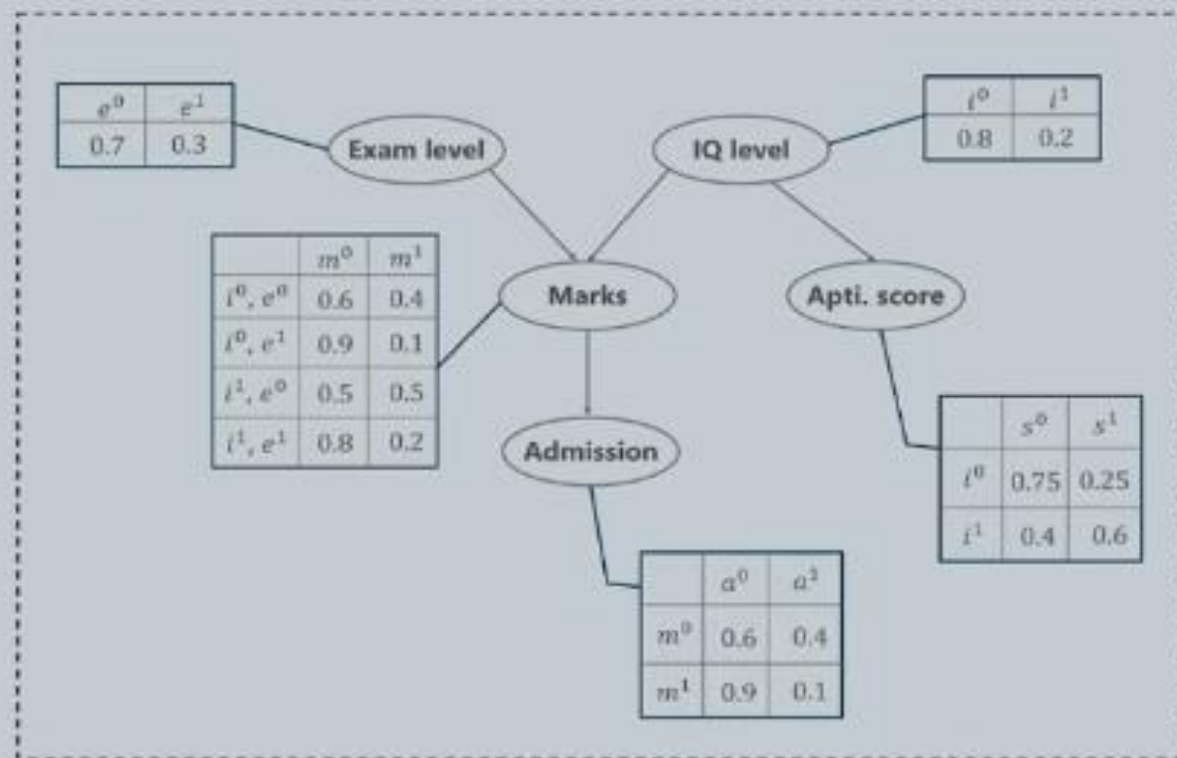


Factorizing Joint Probability Distribution:

$$p(a, m, i, e, s) = p(a \mid m) p(m \mid i, e) p(i) p(e) p(s \mid i)$$

- $p(a \mid m)$: CP of student admit \rightarrow marks
- $p(m \mid i, d)$: CP of the student's marks \rightarrow (IQ & exam level)
- $p(i)$: Probability \rightarrow IQ level
- $p(d)$: Probability \rightarrow exam level
- $p(s \mid i)$ CP of aptitude scores \rightarrow IQ level





The probability of a random variable depends on his parents. Therefore, we can formulate Bayesian Networks as:

$$P(X_1, \dots, X_n) = \prod_{i=1}^n p(X_i \mid \text{Parents}(X_i))$$

Bayesian network applications



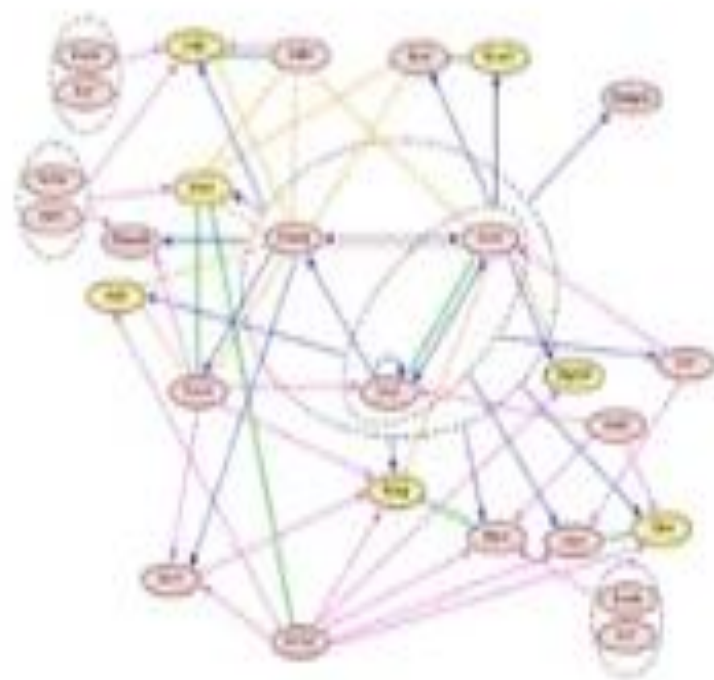
Disease Diagnosis



Optimized Web Search



Spam Filtering



Gene Regulatory Networks

Example – Lung Cancer Diagnosis

A patient has been suffering from shortness of breath (called ✓**dyspnoea**) and visits the doctor, worried that he has lung ✓**cancer**. The doctor knows that other diseases, such as tuberculosis and bronchitis are possible causes, as well as lung cancer. She also knows that other relevant information includes whether or not the patient is a ✓**smoker** (increasing the chances of cancer and bronchitis) and what sort of air **pollution** he has been exposed to. A positive **X-Ray** would indicate either TB or lung cancer.

Nodes and values

What are the nodes to represent and what values can they take, or what state can they be in? For now we will consider only nodes that take discrete values. The values should be both mutually exclusive and exhaustive.

Nodes can be discrete or continuous

- **Boolean nodes** – represent propositions taking binary values

Example: *Cancer* node represents proposition "*the patient has cancer*"

- **Ordered values**

Example: *Pollution* node with values *low, medium, high*

- **Integral values**

Example: *Age* with possible values 1-120

Preliminary choices: Nodes and values

Node	Type	Values
Pollution	Binary	$\{low, high\}$
Smoker	Boolean	$\{T, F\}$
Cancer	Boolean	$\{T, F\}$
Dyspnoea	Boolean	$\{T, F\}$
Xray	Binary	$\{pos, neg\}$

Modeling choices are to be made. For example, **an alternative to representing a patient's exact age might be to clump patients into different age groups**, such as {baby, child, adolescent, young, old}.

The trick is to **choose values that represent the domain efficiently**, but with enough detail to perform the reasoning required.

Bayesian network structure

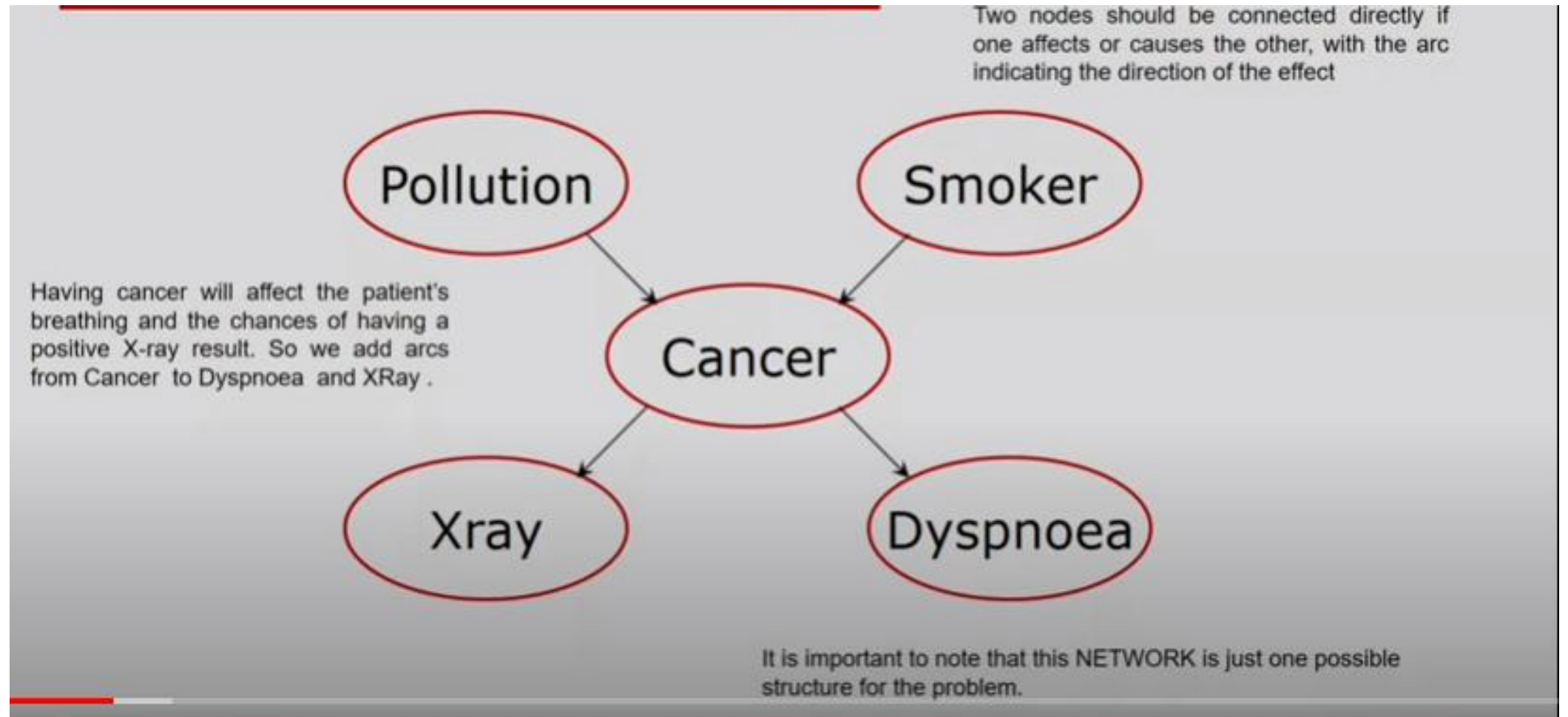
The **structure, or topology, of the network** should **capture qualitative relationships** between variables.

In particular, **two nodes should be connected directly if one affects or causes the other**, with the arc indicating the direction of the effect.

For Example

In the Example being discussed; What factors affect a patient's chance of having cancer? If the answer is "Pollution and smoking," then we should add arcs from Pollution and Smoker to Cancer.

Lung cancer diagnosis:



CPT: (conditional probability table)

After specifying topology, must **specify the CPT for each discrete node**

☐ Each row contains the conditional probability of each node value for each possible combination of values in its parent nodes.

Once the topology of the BN is specified, the next step is to quantify the relationships between connected nodes. Done by specifying a conditional probability distribution for each node. As we are only considering discrete variables at this stage, this takes the form of a conditional probability table

☐ Each row must sum to 1.

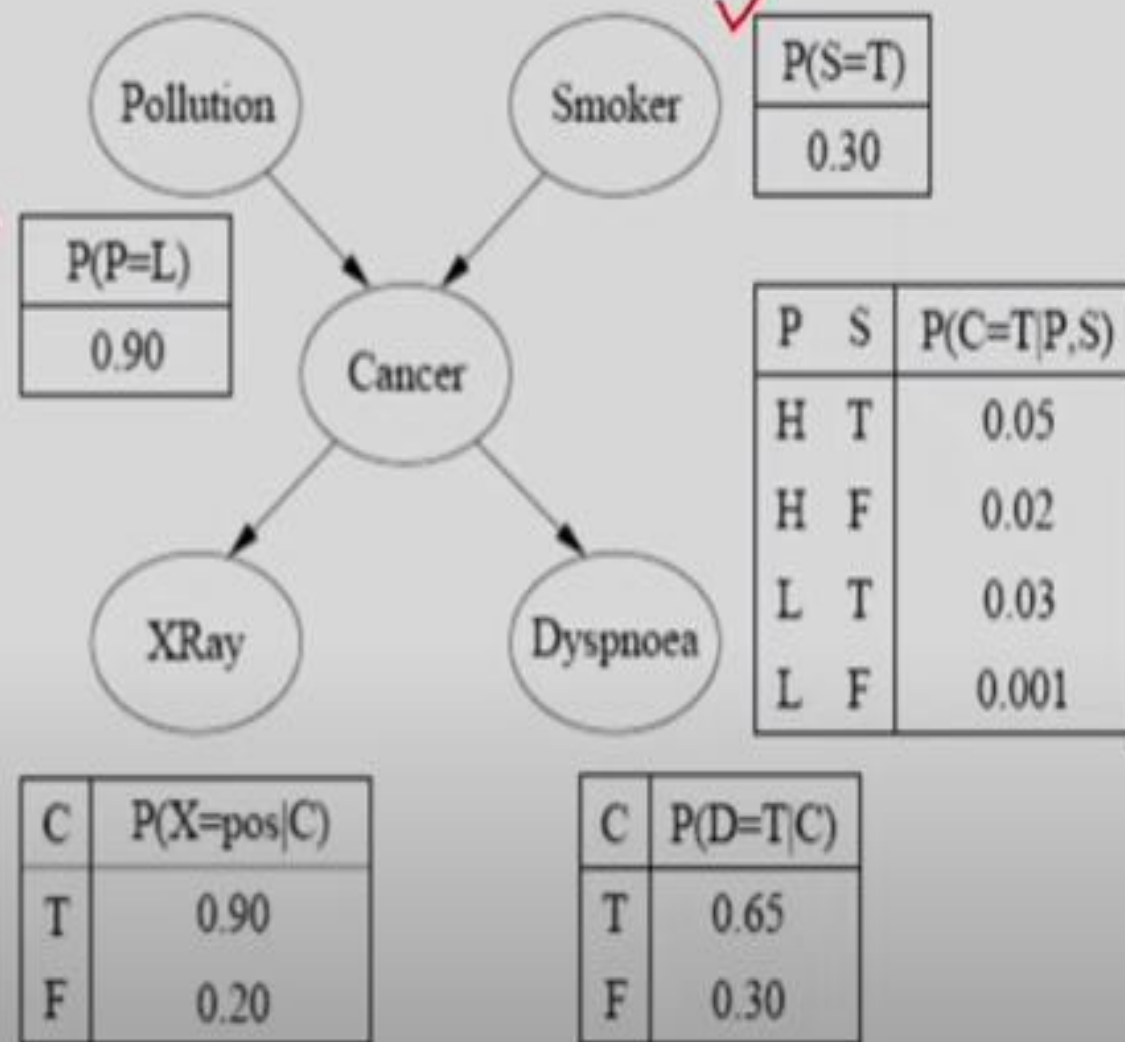
☐ A CPT for a Boolean variable with n Boolean parents contains 2^{n+1} probabilities.

☐ A node with no parents has one row (its prior probabilities).

Root nodes also have an associated CPT, although it is degenerate, containing only one row representing its prior probabilities

For each node we need to look at all the possible combinations of values of those parent nodes. Each such combination is called an **instantiation** of the parent set.

For each distinct instantiation of parent node values, we need to specify the probability that the child will take each of its values

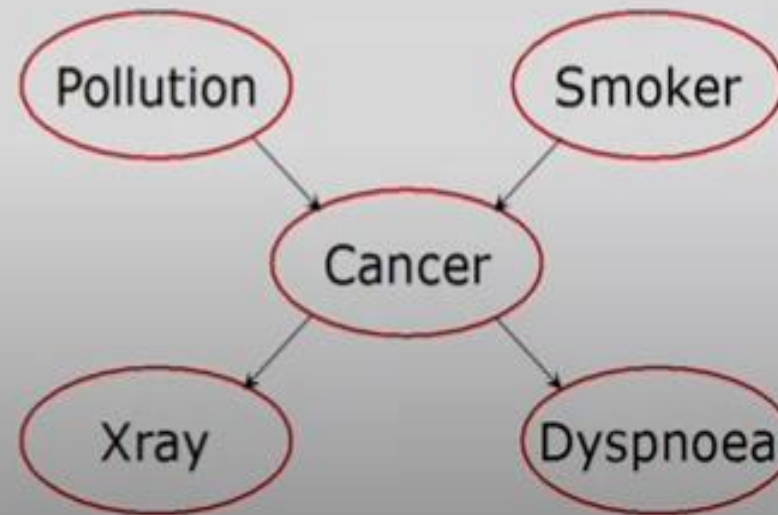


If a node has many parents or if the parents can take a large number of values, the CPT can get very large!

The Markov Property

- Modeling with Bayesian Networks requires the assumption of the **Markov Property**:
 - *There are no direct dependencies in the system being modelled which are not already explicitly shown via arcs.*

Example: smoking can influence dyspnoea only through causing cancer



- Bayesian networks which have the Markov property are also called **Independence-maps** (or, I-maps for short), since every independence suggested by the lack of an arc is real in the system.
- It is not generally required that the arcs in a BN correspond to real dependencies in the system.
 - The CPTs may be parameterized in such a way as to nullify any dependence.
 - Every fully-connected Bayesian network can represent, any joint probability distribution over the variables being modeled

-
- We shall prefer minimal models and, in particular, minimal I-maps, which are I-maps such that the deletion of any arc violates I-mapness by implying a non-existent independence in the system.
 - If, in fact, **every arc in a Bayesian network happens to correspond to a direct dependence** in the system, then the Bayesian Network is said to be a **Dependence-map** (or, D-map for short).
 - A Bayesian network which is both an I-map and a D-map is said to be a **perfect map**.

Conditional independence

The relationship between **conditional independence** and **Bayesian network structure** is important for understanding how Bayesian networks work.

1. Causal Chains
2. Common Causes
3. Common Effects - Conditional dependence.
4. D-separation

UNIT-IV

Learning:

What Is Learning? Rote Learning, Learning by Taking Advice, Learning in Problem Solving, Learning from Examples -- Winston's Learning Program, Decision Trees.

What do you mean by learning ?

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.



Learn from experience



Learn from data

What is Machine Learning?

- Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.
- Machine learning focuses on the development of computer programs that can access data and use it learn for themselves. The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide.
- The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Definition

A computer program is said to learn from experience with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**. (Mitchell 1997)

This means :

Given : A task **T**

A performance measure **P**

Some experience **E** with the task

Goal: Generalize the experience in a way that allows to improve your performance on the task.

Why do machines need learning:

- To understand and improve efficiency of machine learning
- Discover new things or structures that are unknown to human
- Fill in skeletal/incomplete about a domain

Advantages:

Skill refinements

Knowledge acquisition

Different kinds of learning:

Supervised learning: we get correct answers for each training instance

Supervised learning involves learning a function from examples

of its inputs and outputs

Unsupervised learning: we don't know anything. . .

Unsupervised learning involves learning patterns in the input when no specific output values are supplied

Reinforcement learning: we get occasional rewards

In reinforcement learning the agent must learn from reinforcement (reward, less exact feedback than in supervised learning)

Paradigms of Machine Learning

- **Rote Learning:** Learning by memorization; One-to-one mapping from

Inputs to stored representation; Association-based storage and retrieval.

- **Induction:** Learning from examples; A form of supervised learning, uses specific examples to reach general conclusions; Concepts are learned from sets of labeled instances.

- **Clustering:** Discovering similar group; Unsupervised, Inductive learning in which natural classes are found for data instances, as well as ways of classifying them.

- **Analogy:** Determine correspondence between two different representations that come from Inductive learning in which a system transfers knowledge from one database into another database of a different domain.

■ **Discovery:** Learning without the help from a teacher; Learning is both inductive and deductive. It is deductive if it proves theorems and discovers concepts about those theorems. It is Inductive when it raises conjectures (guess). It is unsupervised, specific goal not given.

■ **Genetic Algorithms:** Inspired by natural evolution; In the natural world, the organisms that are poorly suited for an environment die off, while those well-suited for it prosper. Genetic algorithms search the Space of individuals for good candidates. The "goodness" of an individual is measured by some fitness function. Search takes place in parallel, with many individuals in each generation.

Reinforcement: Learning from feedback (+ve or -ve reward) given at end of a sequence of steps. Unlike supervised learning, the Reinforcement learning takes place in an environment where the Agent cannot directly compare the results of its action to a desired result. Instead, it is given some reward or punishment that relates to its actions. It may win or lose a game, or be told it has made a good move or a poor one. The job of reinforcement learning is to find a successful function using these rewards.

Different methods of learning

The following are the different methods of learning:

- Rote Learning
- Learning by Taking Advice
- Learning in Problem Solving
- Learning from Examples
 - (Winston's program and decision trees)

Other:

- Learning with micro operators
- Explanation based learning
- Formal based learning

Rote Learning

Rote Learning is basically memorization.

- Saving knowledge so it can be used again.
- Retrieval is the only problem.
- No repeated computation, inference or query is necessary.

A simple example of rote learning is caching

- Store computed values (or large piece of data)
- Recall this information when required by computation.
- Significant time savings can be achieved.
- Many AI programs (as well as more general ones) have used caching very effectively

Example:

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$

But next time if you give, 6!

It will compute $5! * 6$ (but not $6 * 5 * 4 * 3 * 2 * 1$)

- It avoids understanding the inner complexity but focuses on **memorizing** the materials. So that it can be recalled exactly the way it was read
- **Learning by something** i.e., repeating----In this “over and over again” method is used.

Ex: how we learn or memorize the lyrics of songs

- Rote learning is basically memorization:
 1. Saving knowledge, so it can be used again
 2. Retrieval is the only problem
 3. No repeated computation, inference or query necessary

Ex: cache memory/caching

Learning by taking advice

- A computer can do very little work without program
- In the form of programming we give instructions to the computer or system
- When programming, we write series of programs, a rudimental kind of learning takes place
- After being programmed the computer can do something which it could not do it before
- Interpreter or compiler is needed to intervene to change the teacher's instructions into code that machine can execute directly
- In simple words
Request----interpret---operative-----integrate-----evaluate.

Learning in Problem Solving

- No teacher advices but learning can be happened through experience

Ex: To find gcd of 2 numbers

- It does not involve in increase of knowledge but focus on just method of using knowledge
- Learning by parameter adjustment

Ex: Samsung checker

If the above example includes the computation then assume

$$C_1t_1 + C_2t_2 + C_3t_3 + \dots + C_nt_n$$

where, c ----- weights

t ----- features

- Every pattern or feature is generally combined with weights, but the **problem is how to join weights and features?**
- Many problems rely on evaluation procedure that combines information from several sources into a single summary statistics.
- Pattern classification often combines with weights
- It is difficult to know how much weight has to be attached to each feature
- One way to find is through experience
- Other method is we use the outcome to adjust the weights for factor in an evaluation time. (because feature weights may get changed)

Learning from Examples

- **This involves the process of learning by example** -- where a system tries to induce a general rule from a set of observed instances.
- **This involves classification** -- assigning, to a particular input, the name of a class to which it belongs. Classification is important to many problem solving tasks.
- **A learning system has to be capable of evolving its own class descriptions:**
 - Initial class definitions may not be adequate.
 - The world may not be well understood or rapidly changing.
- The task of constructing class definitions is called induction (learning by induction) or concept learning

- The techniques used for constructing class definitions (or concept learning) are:
 - Winston's learning program
 - Version spaces
 - Decision trees
- In this we classify the situation

Ex: classifying numbers 0----9, 11-----99,100-----999 (number of digits)
- It is the simplest form (straight forward recognition task)
- Before classification can be done, the classes which it will be using must be defined.
- Isolate set of features relevant to the task domain. There are many methods to isolate.

Method 1: Define each class and each class has a scaling function.

Example: $C_1t_1 + C_2t_2 + C_3t_3 + \dots + C_nt_n$

Task: Weather prediction, where

t1---- rainfall

t2---- humidity

t3 ---- cloudy

t4 ---- temperature and so on.....

Method 2: Define each class as a structure compared to those features.

Task: Identify part of animal

Each part of the animal can be stored as a structure with various features representing such as color, length, feathers and so on.....

Designing a Learning System: An Example

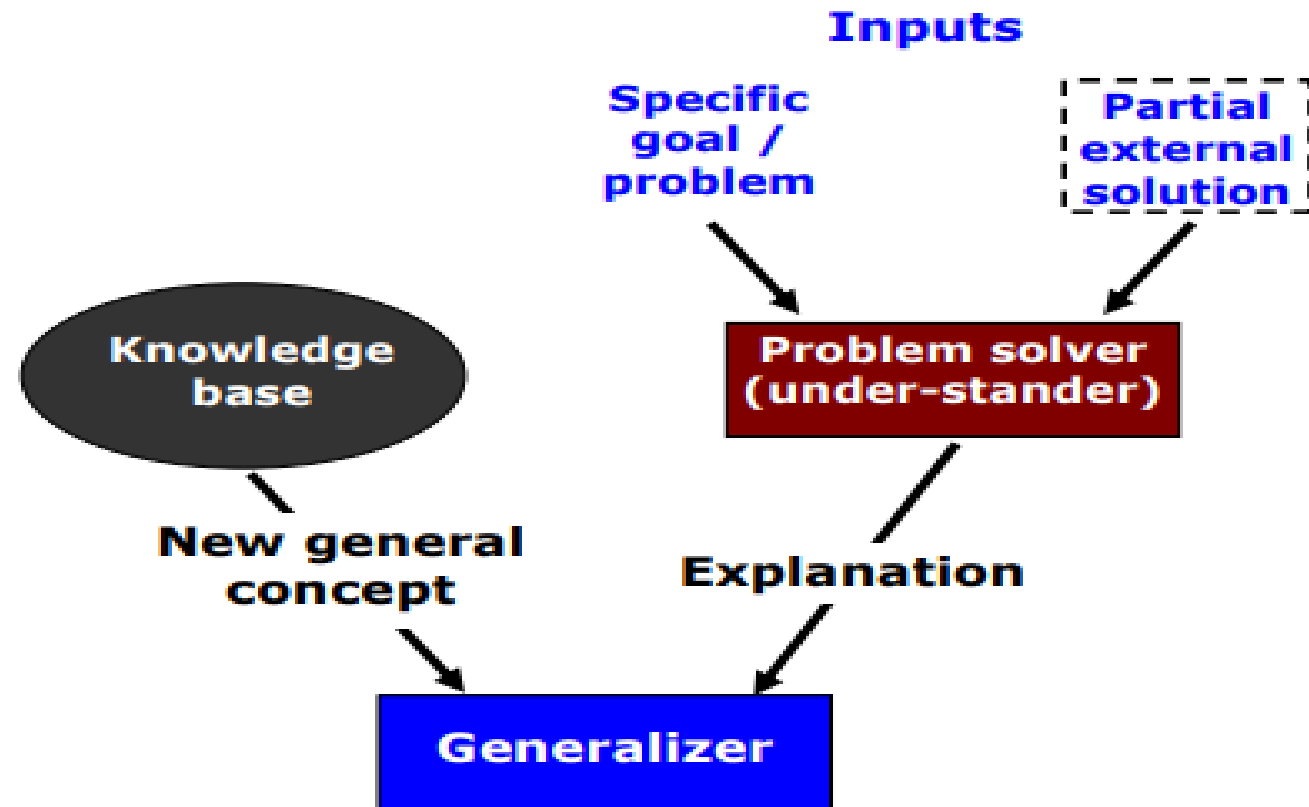
1. Problem Description
2. Choosing the Training Experience
3. Choosing the Target Function
4. Choosing a Representation for the Target Function
5. Choosing a Function Approximation Algorithm
6. Final Design

Explanation-based Learning (EBL)

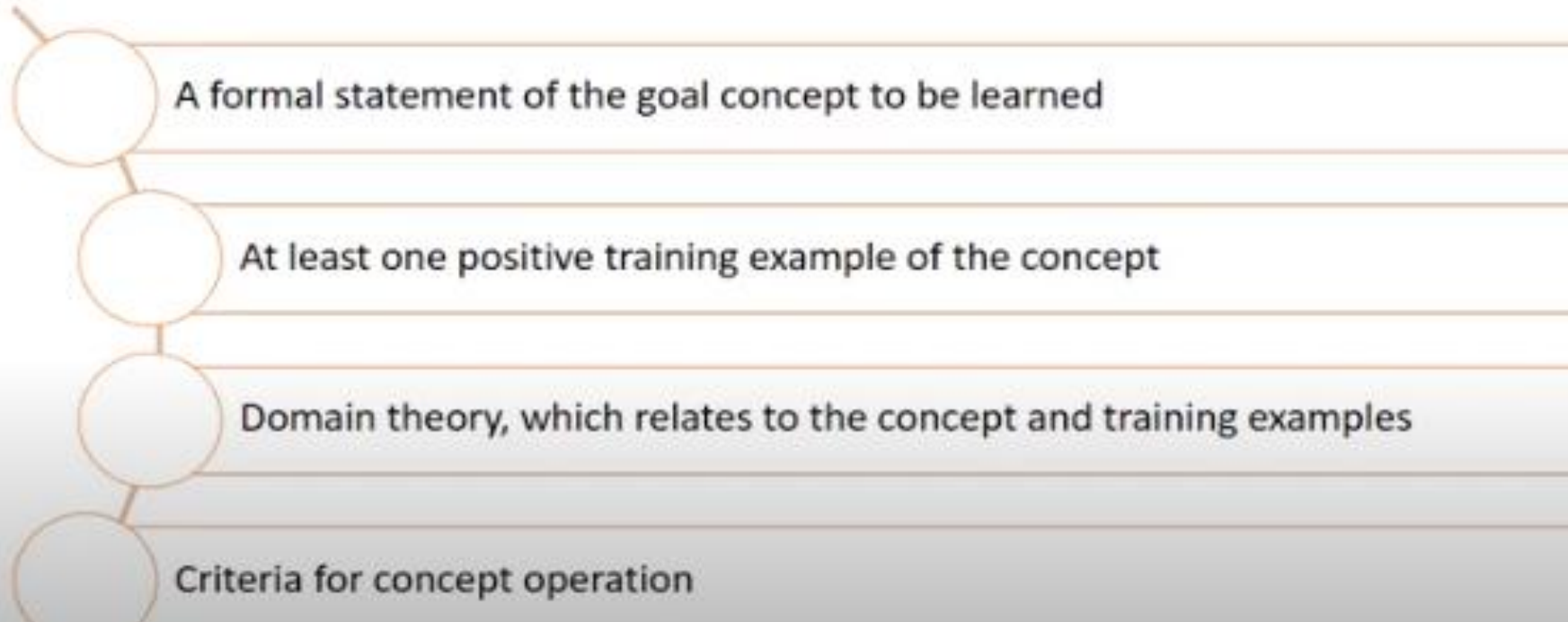
- An Explanation-based Learning (**EBL**) system accepts an example (i.e. a training example) and explains **what it learns from the example**.
- **Explanation based generalization (EBG)** is an algorithm for explanation based learning, described in Mitchell et al. (1986).
- The objective of EBL is to formulate a **generalized explanation of the goal concept**

- It **has two steps**
 - first, explain method
 - secondly, generalize method.
- During the first step, **the domain theory is used to prune away** all the unimportant aspects of training examples **with respect to the goal concept.**
- The **second step** is to generalize the explanation as far as possible while still describing the goal concept

The overall architecture of the EBL learning method.



EBL requirements- In EBL four kind of information must be known to learners in advance.



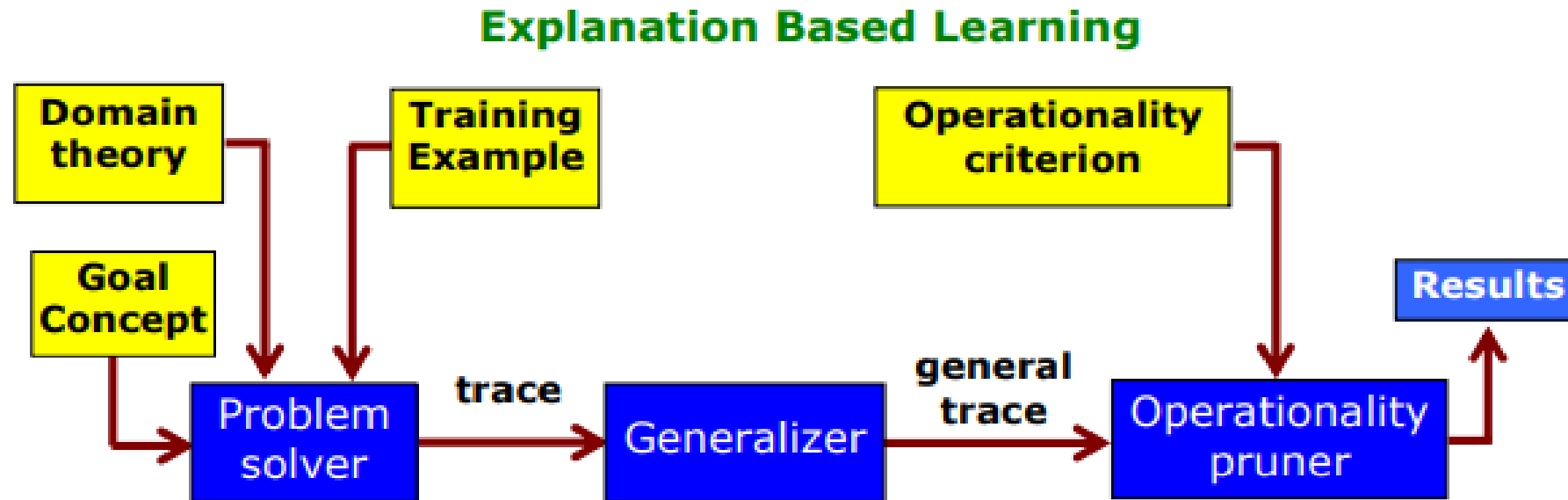
Working

- Given
 - Goal (e.g., some predicate calculus statement)
 - Situation Description (facts)
 - Domain Theory (inference rules)
 - Operationality Criterion
- Use **problem solver to justify**, using the rules, **the goal** in terms of the facts.
- Generalize **the justification as much as** possible.
- The Operationality criterion **states which other terms** can appear in the generalized result.



EBL System Schematic

The schematic below shows explanation based learning.



Input to EBL :

The blocks color yellow are external to EBL.

They are the 4 different kinds of input that EBL algorithm accepts.

Example

- Consider the problem of learning the **concept bucket**.
- We want to generalize from a single example of a bucket.
- At first collect the following informations.

1. **Input Examples:**

Owner (object, X) \wedge has part (object, Y) \wedge is(object, Deep) \wedge Color (Object, Green)
 \wedge ... (Where Y is any thin material)

2. **Domain Knowledge:**

is (a, Deep) \wedge has part (a, b) \wedge is a(b, handle) \rightarrow liftable (a)

has part (a, b) \wedge is a (b, Bottom) \wedge is (b, flat) \rightarrow Stable (a)

has part (a, b) \wedge is a (b, Y) \wedge is (b, Upward – pointing) \rightarrow Open – vessel (a)

3. Goal: Bucket

B is a bucket if B is lift able, stable and open-vessel.

4. Description of Concept: These are expressed in purely structural forms like Deep, Flat, rounded etc.

4. **Description of Concept:** These are expressed in purely structural forms like Deep, Flat, rounded etc.

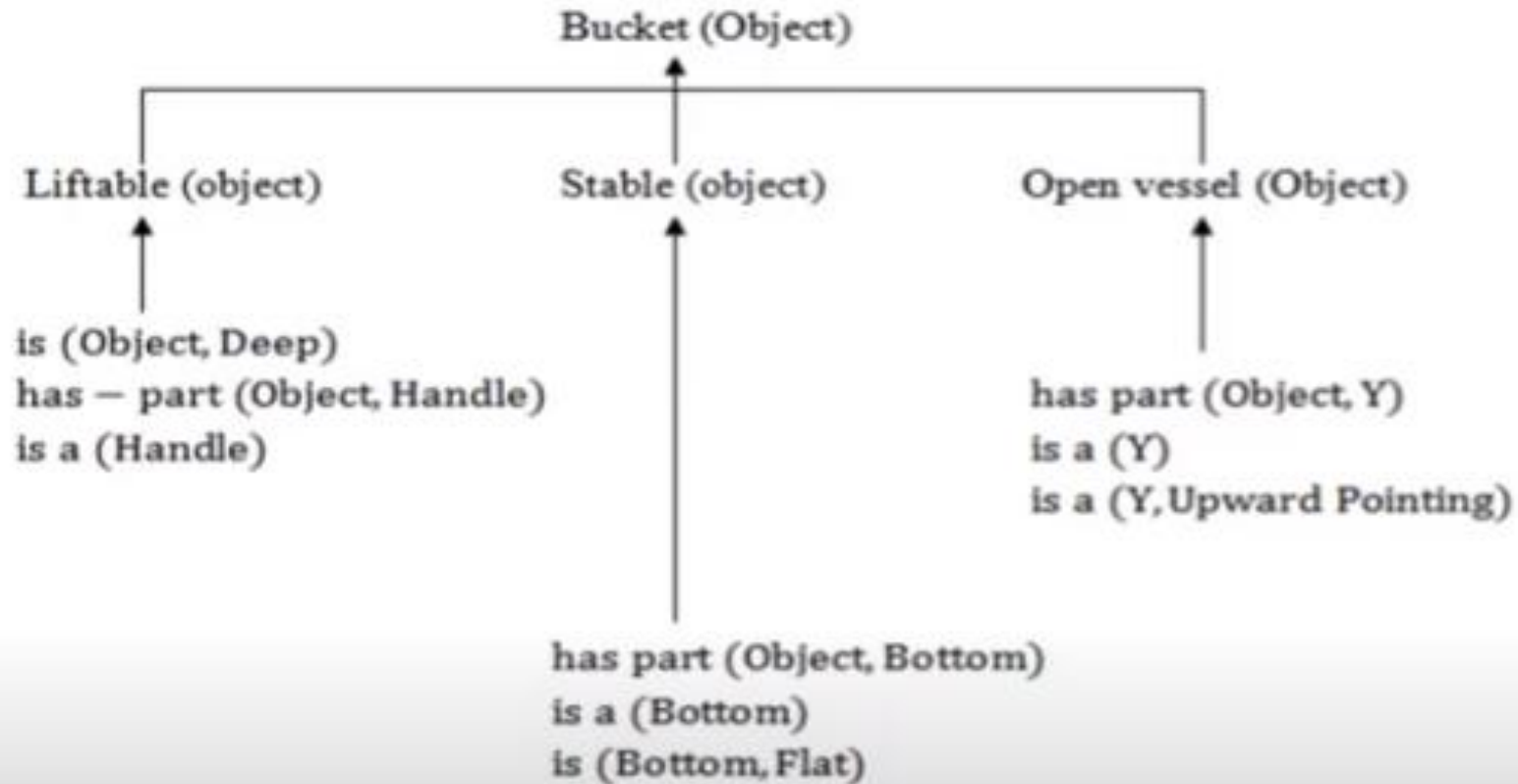


Figure An explanation of BUCKET Object

Winston's Learning Program

Winston (1975) described a **Blocks World Learning program**. This program operated in a simple blocks domain. The goal is to construct representation of the definition of concepts in the blocks domain.

Example : Concepts such a "house".

- Start with input, a line drawing of a blocks world structure.
It learned Concepts **House, Tent, Arch** as :
brick (rectangular block) with a **wedge** (triangular block) suitably placed on top of it, **tent** – as 2 wedges touching side by side, or an **arch** – as 2 non-touching bricks supporting a third wedge or brick.
- The program for Each concept is learned through near miss. A near miss is an object that is not an instance of the concept but a very similar to such instances.
- The program uses procedures to analyze the drawing and construct a semantic net representation.

Winston's learning program


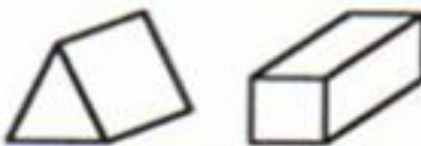




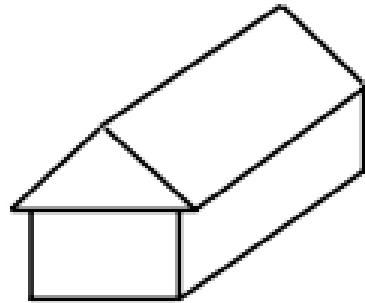
	Concept	Near Miss
House		
Tent		
Arch		

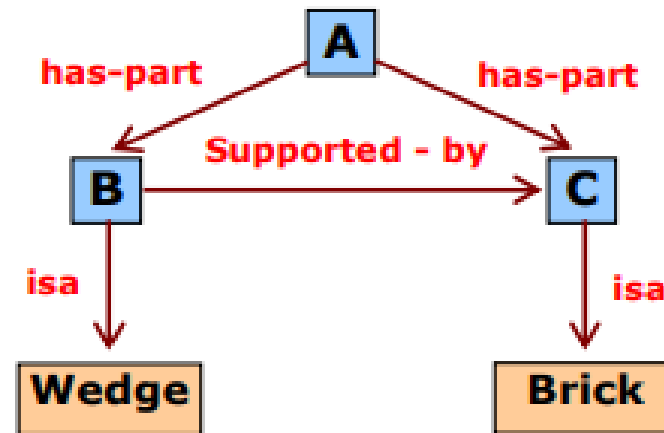
Figure 17.2: Some Blocks World Concepts

- An example of such an structural for the house is shown below.

Object - house



Semantic net



- Node **A** represents entire structure, which is composed of two parts : node **B**, a Wedge, and node **C**, a Brick.
Links in network include supported-by, has-part, and isa.

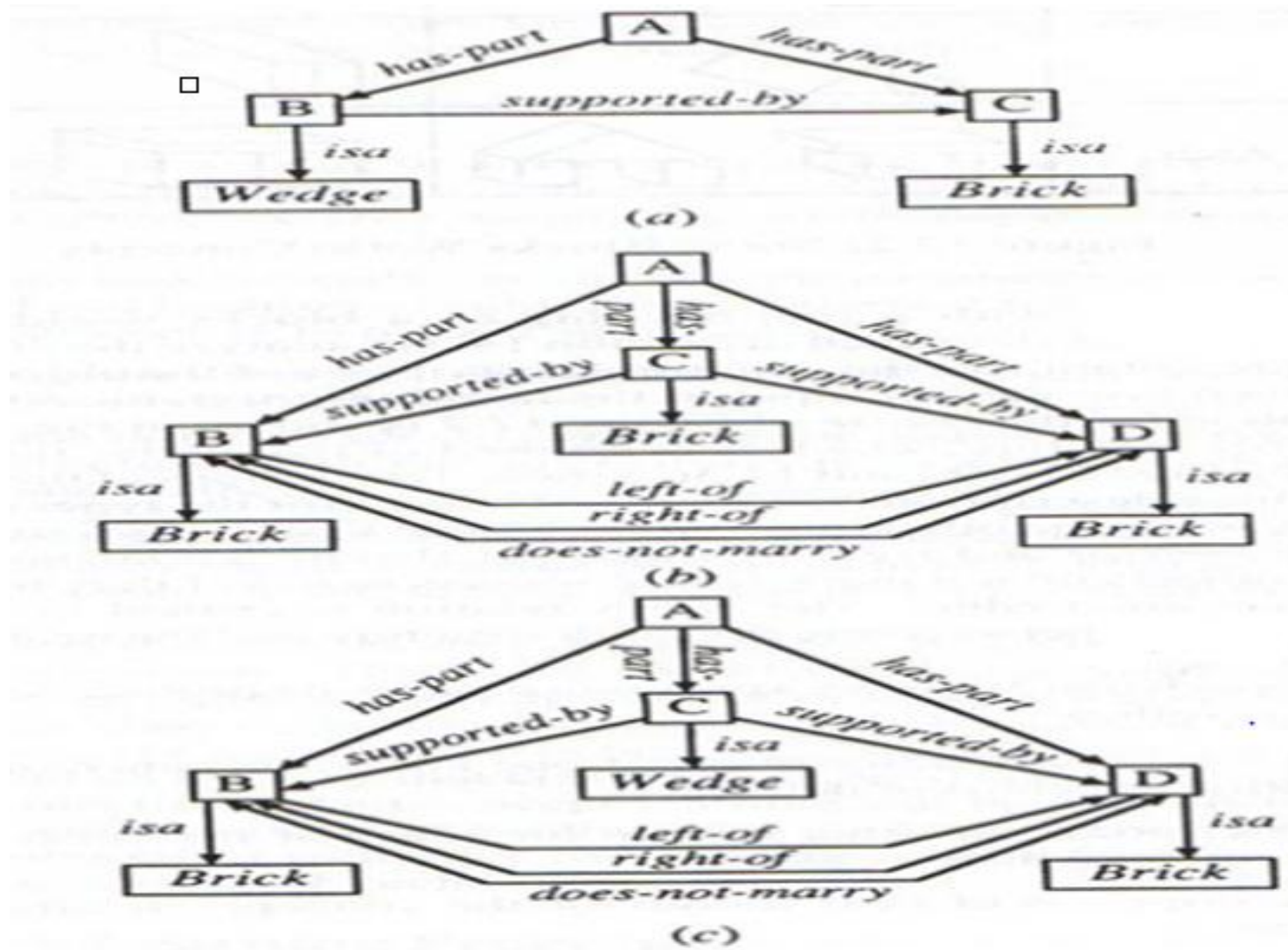


Fig: Structural Descriptions

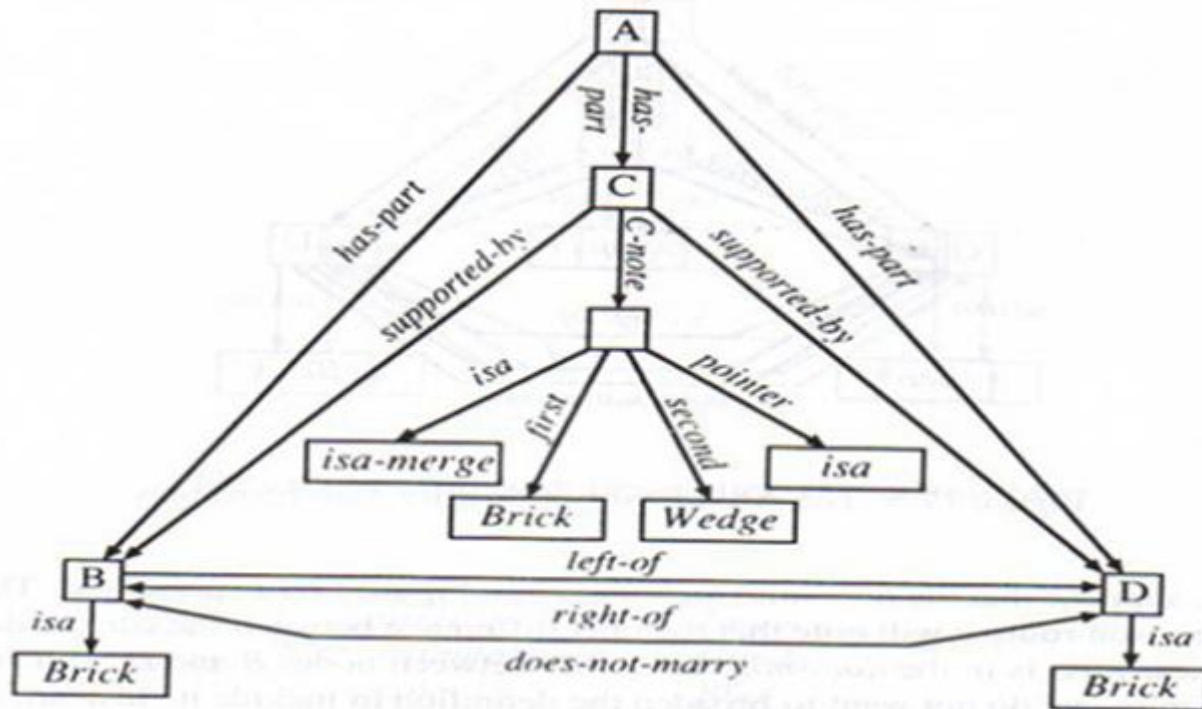


Fig: The comparison of two arches

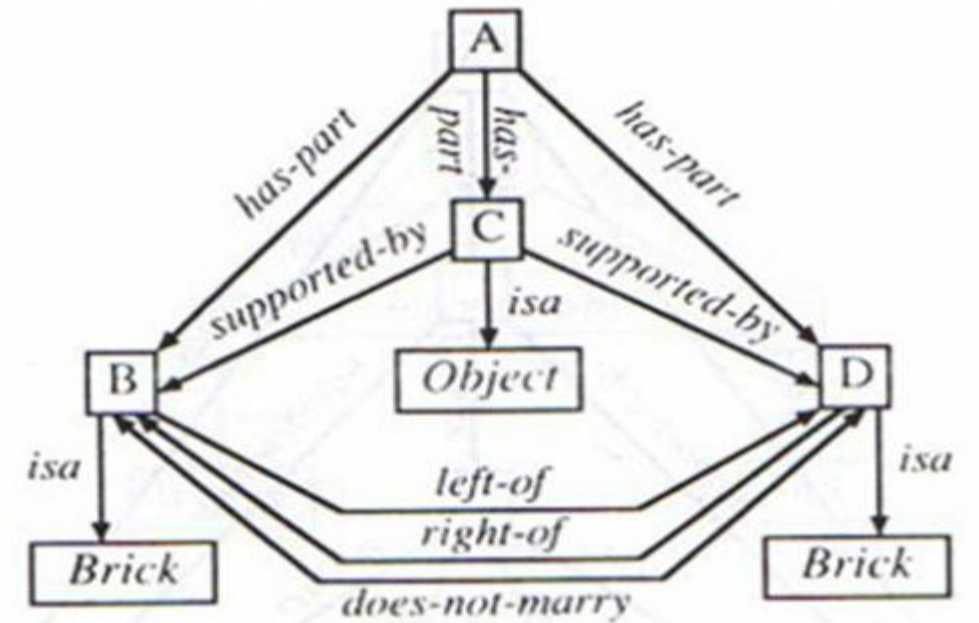


Fig: The Arch description after two examples

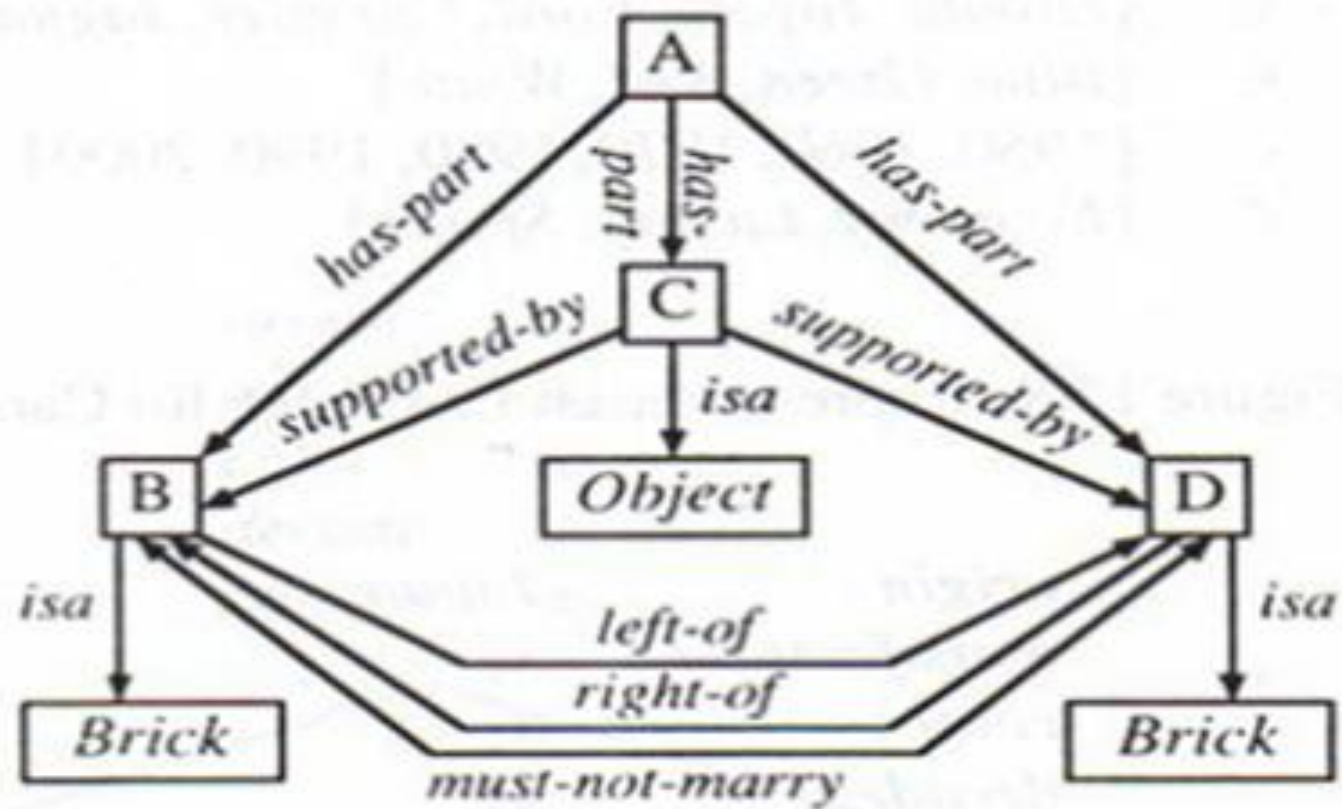


Fig: The Arch description after a near Miss

- Winston's program followed 3 basic steps in concept formulation:
 1. Select one known instance of the concept.
Call this the **concept definition**.
 2. Examine definitions of other known instance of the concept.
Generalize the definition to include them.
 3. Examine descriptions of **near misses**.
Restrict the definition to **exclude** these.
- Both steps 2 and 3 of this procedure rely heavily on comparison process by which similarities and differences between structures can be detected.
- Winston's program can be similarly applied to learn other concepts such as "ARCH".

Decision Trees.

Decision trees are powerful tools for classification and prediction.

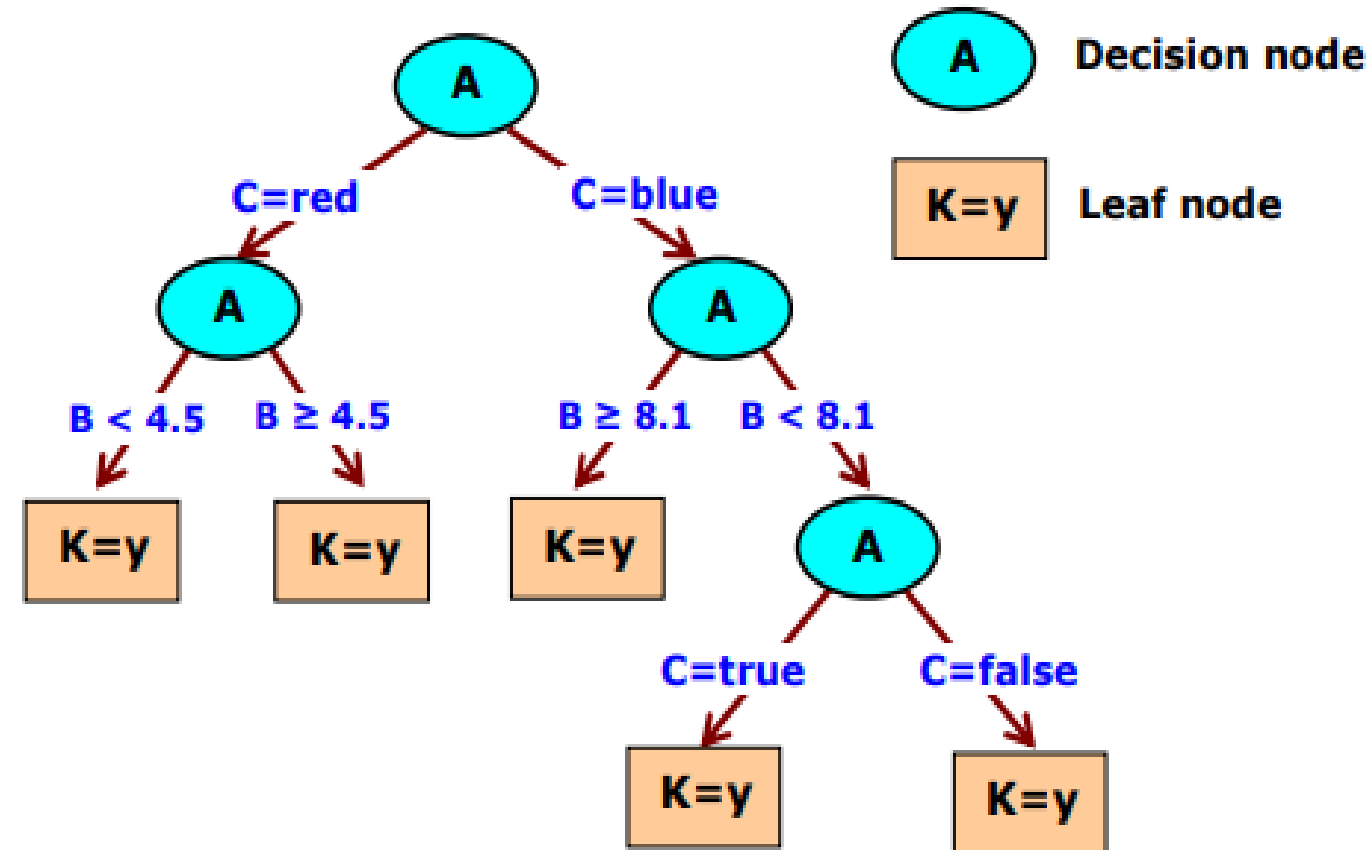
Decision trees represent rules. Rules are easily expressed so that humans can understand them or even directly use in a database access language like SQL so that records falling into a particular category may be retrieved.

- **Description**

- Decision tree is a classifier in the form of a tree structure where each node is either a leaf or decision node.
 - ‡ **leaf node** – indicates the target attribute (class) values of examples.
 - ‡ **decision node** – specify test to be carried on an attribute-value.

- Decision tree is a typical inductive approach to learn knowledge on classification. The conditions are :
 - ‡ **Attribute-value description:** Object or case must be expressible as a fixed collection of properties or attributes having discrete values.
 - ‡ **Predefined classes :** Categories to which examples are to be assigned must already be defined (ie supervised data).
 - ‡ Discrete classes: Classes must be sharply delineated; continuous classes broken up into vague categories as "hard", "flexible", "soft".
 - ‡ Sufficient data: Enough training cases to distinguish valid patterns.

- **Example :** A simple decision tree



- Dichotomiser: Defined in terms of dividing into two which are completely opposite.

Algorithm to generate a Decision Tree

A decision tree program constructs a decision tree **T** from a set of training cases. The advantage of a decision tree learning is that a program, rather than a knowledge engineer, elicits knowledge from an expert.

ID3 Algorithm (Iterative Dichotomiser 3)

ID3 is based on the Concept Learning System (CLS) algorithm, developed J. Ross Quinlan, at the University of Sydney in 1975.

■ **Description**

- ✦ ID3 builds a decision tree from a set of "examples".
The "examples" have several attributes.
They belong to a class (yes or no).
- ✦ Leaf nodes contain the class name.
- ✦ Decision node is an attribute test .

■ Attribute Selection

How does ID3 decide which attribute is the best?

There are different criteria to select which attribute will become a test attribute in a given branch of a tree. A well known criteria is information gain. It uses a log function with a base of 2, to determine the number of bits necessary to represent a piece of information.

‡ **Entropy** measures information content in an attribute.

Shannon defines information entropy in terms of a discrete random variable X , with possible states (or outcomes) $x_1 \dots x_n$ as:

$$H(X) = \sum_{i=1}^n p(x_i) \bullet \log_2 (1/p(x_i)) = - \sum_{i=1}^n p(x_i) \bullet \log_2 p(x_i)$$

where $p(x_i) = P(X = x_i)$ is the probability of the i^{th} outcome of X .

‡ **Information gain** of an attribute X with respect to class attribute Y .

Let Y and X are discrete variables that take values in $\{y_1 \dots y_i \dots y_k\}$ and $\{x_1 \dots x_j \dots x_l\}$.

The information gain of a given attribute X with respect to the class attribute Y is the reduction in uncertainty about the value of Y when we know the value of X , is called $I(Y; X)$.

Uncertainty

Uncertainty about the value of **Y** is measured by its **entropy**, **H(Y)**.

uncertainty about the value of **Y** when we know the value of **X** is given by the **conditional entropy** of **Y** given **X**, i.e., **H(Y |X)**.

Information Gain of a given attribute **X** with respect to the class attribute **Y** is given by : **I(Y ; X) = H(Y) - H(Y |X)**. where

H(Y) = $-\sum_{yi=1}^k p(yi) \bullet \log_2 p(yi)$ is the initial entropy in **Y**,

H(Y |X) = $\sum_{xj=1}^l p(xj) \bullet H(Y |xj)$ is conditional entropy of **Y**

given **X** where **H(Y |xj)** = $-\sum_{yi=1}^k p(yi | xj) \bullet \log_2 p(yi | xj)$ is

the entropy in **Y** given a particular answer **x_j**.

■ Example

Decide if the weather is amenable to play Golf or Baseball.

‡ **Training Data** Collected are Over 2 weeks

Day	Outlook	Temp	Humidity	wind	Play
1	sunny	85	85	week	no
2	sunny	80	90	strong	no
3	cloudy	83	78	week	yes
4	rainy	70	96	week	yes
5	rainy	68	80	week	yes
6	rainy	65	70	strong	no
7	cloudy	64	65	strong	yes
8	sunny	72	95	week	no
9	sunny	69	70	week	yes
10	rainy	75	80	week	yes
11	sunny	75	70	strong	yes
12	cloudy	72	90	strong	yes
13	cloudy	81	75	week	yes
14	rainy	71	85	strong	no

Learning set

In the above example, two attributes, the Temperature and Humidity have continuous ranges. ID3 requires them to be **discrete** like **hot, medium, cold, high, normal**. Table below indicates the acceptable values.

Attribute		Possible Values		
Outlook	Sunny	Cloudy	Rainy	
Temperature	Hot	Medium	Cold	
Humidity	High	Normal		
Wind	Strong	Weak		
Class	play	no play		
Decision	n (negative)	p (positive)		

Assign discrete values to the Attributes

Partition the continuous attribute values to make them discrete, following the key mentioned below.

Temperature : Hot (H) 80 to 85 Medium (M) 70 to 75 Cold (C) 64 to 69

Humidity : High (H) 81 to 96 Normal (N) 65 to 80

Class : Yes (Y) play No (N) no play

Day	Outlook	Temp		Humidity		Wind	Class (play)
1	Sunny	85	Hot	85	High	week	no
2	Sunny	80	Hot	90	High	strong	no
3	Cloudy	83	Hot	78	High	week	yes
4	Rainy	70	Medium	96	High	week	yes
5	Rainy	68	Cold	80	Normal	week	yes
6	Rainy	65	Cold	70	Normal	strong	no
7	Cloudy	64	Cold	65	Normal	strong	yes
8	Sunny	72	Medium	95	High	week	no
9	Sunny	69	Cold	70	Normal	week	yes
10	Rainy	75	Medium	80	Normal	week	yes
11	Sunny	75	Medium	70	Normal	strong	yes
12	Cloudy	72	Medium	90	High	strong	yes
13	Cloudy	81	Hot	75	Normal	week	yes
14	Rainy	71	Medium	85	High	strong	no

- **Finally in simple words:**

Calculate the entry of every attribute using the data set “S” i.e.,

$$\text{Entropy}(S) = \sum_{i=1}^n p(x_i) \bullet \log_2 (1/p(x_i)) = - \sum_{i=1}^n p(x_i) \bullet \log_2 p(x_i)$$

Put the set “S” into subset using the attribute for which the resulting entropy (after splitting) is minimum or (or equivalently gain is max) i.e.,

$$\text{Gain}(S,A) = \text{Entropy}(x) - \sum_{i=1}^n [p(S/A) \cdot \text{Entropy}(S/A)]$$

Where, A ---- Attribute

S/A ---- S is given by A

- Make a decision tree node containing that attribute
- Recurse on subsets using remaining attributes.

Example:

To go for outing/Not based on weather forecasting?

Attributes:

Outlook

Temperature

Humidity

Wind

Play/class/Decision

Step 1: Calculate Entropy for Decision:

- Play or decision column consist of 14 instances and includes two labels “yes” and “no”
- Decision labelled “yes”=9
- Decision labelled “no”=5

$$\begin{aligned}\text{Entropy (S/Decision)} &= - (9/14) \text{Log}_2 (9/14) - (5/14) \text{Log}_2 (5/14) \\ &= 0.940\end{aligned}$$

◇ **Step 02 : Attribute Outlook**

Outlook value can be sunny, cloudy or rainy.

Outlook = sunny is of occurrences 5

Outlook = cloudy is of occurrences 4

Outlook = rainy is of occurrences 5

Outlook = sunny, 2 of the examples are 'yes' and 3 are 'no'

Outlook = cloudy, 4 of the examples are 'yes' and 0 are 'no'

Outlook = rainy, 3 of the examples are 'yes' and 2 are 'no'

$$\text{Entropy}(S_{\text{sunny}}) = - (2/5) \times \log_2(2/5) - (3/5) \times \log_2(3/5) = 0.970950$$

$$\text{Entropy}(S_{\text{cloudy}}) = - (4/4) \times \log_2(4/4) - (0/4) \times \log_2(0/4) = 0$$

$$\text{Entropy}(S_{\text{rainy}}) = - (3/5) \times \log_2(3/5) - (2/5) \times \log_2(2/5) = 0.970950$$

$$\begin{aligned} \text{Gain}(S, \text{Outlook}) &= \text{Entropy}(S) - (5/14) \times \text{Entropy}(S_{\text{sunny}}) \\ &\quad - (4/14) \times \text{Entropy}(S_{\text{cloudy}}) \\ &\quad - (5/14) \times \text{Entropy}(S_{\text{rainy}}) \\ &= 0.940 - (5/14) \times 0.97095059 - (4/14) \times 0 \\ &\quad - (5/14) \times 0.97095059 \\ &= 0.940 - 0.34676 - 0 - 0.34676 \\ &= 0.246 \end{aligned}$$

◆ **Step 03 : Attribute Temperature**

Temp value can be hot, medium or cold.

Temp = hot is of occurrences 4

Temp = medium is of occurrences 6

Temp = cold is of occurrences 4

Temp = hot, 2 of the examples are 'yes' and 2 are 'no'

Temp = medium, 4 of the examples are 'yes' and 2 are 'no'

Temp = cold, 3 of the examples are 'yes' and 1 are 'no'

$$\text{Entropy(Shot)} = - (2/4) \times \log_2(2/4) - (2/4) \times \log_2(2/4) = -0.999999999$$

$$\text{Entropy(Smedium)} = - (4/6) \times \log_2(4/6) - (2/6) \times \log_2(2/6) = -0.91829583$$

$$\text{Entropy(Scold)} = - (3/4) \times \log_2(3/4) - (1/4) \times \log_2(1/4) = -0.81127812$$

$$\begin{aligned} \text{Gain(S, Temp)} &= \text{Entropy(S)} - (4/14) \times \text{Entropy(Shot)} \\ &\quad - (6/14) \times \text{Entropy(Smedium)} \\ &\quad - (4/14) \times \text{Entropy(Scold)} \\ &= 0.940 - (4/14) \times 0.99999 - (6/14) \times 0.91829583 \\ &\quad - (4/14) \times 0.81127812 \\ &= 0.940 - 0.2857142 - 0.393555 - 0.2317937 \\ &= 0.0289366072 \end{aligned}$$

♦ Step 04 : Attribute Humidity

Humidity value can be high, normal.

Humidity = high is of occurrences 7

Humidity = normal is of occurrences 7

Humidity = high, 3 of the examples are 'yes' and 4 are 'no'

Humidity = normal, 6 of the examples are 'yes' and 1 are 'no'

$$\text{Entropy}(S_{\text{high}}) = - (3/7) \times \log_2(3/7) - (4/7) \times \log_2(4/7) = -0.9852281$$

$$\text{Entropy}(S_{\text{normal}}) = - (6/7) \times \log_2(6/7) - (1/7) \times \log_2(1/7) = -0.5916727$$

$$\begin{aligned}\text{Gain}(S, \text{Humidity}) &= \text{Entropy}(S) - (7/14) \times \text{Entropy}(S_{\text{high}}) \\ &\quad - (7/14) \times \text{Entropy}(S_{\text{normal}}) \\ &= 0.940 - (7/14) \times 0.9852281 - (7/14) \times 0.5916727 \\ &= 0.940 - 0.49261405 - 0.29583635 \\ &= 0.1515496\end{aligned}$$

Step 5: Calculate wind factor on decision:

$$\text{Gain}(S, W \text{ (wind)}) = \text{Entropy}(S) - \sum_{i=1}^n p(S/W) * \text{Entropy}(S/W)$$

But, Wind has 2 factors “weak” and “strong”

$$= \text{Entropy}(S) - [p(S/w=\text{weak}) * \text{Entropy}(S/w=\text{weak})] - [p(S/s=\text{strong}) * \text{Entropy}(S/s=\text{strong})]$$

8 occurrences of **wind = weak** and

6 occurrences of **wind = strong**.

For **wind = weak**, 6 of the examples are **YES** and 2 are **NO**.

For **wind = strong**, 3 of the examples are **YES** and 3 are **NO**.

- **Weak wind factor on Decision:**

Entropy (D/w=weak)= $-(6/8) \bullet \log_2(6/8) - (2/8) \bullet \log_2(2/8) = 0.811$

Entropy (D/s=strong)= $-(3/6) \bullet \log_2(3/6) - (3/6) \bullet \log_2(3/6) = 1.00$

Substitute in **Gain(S,W (wind))**= $0.940 - [(8/14) \bullet 0.811] - [(6/14) \bullet 1]$
= 0.048

◇ **Step 06 : Summary of results are**

Entropy(S) = 0.940

Gain(S, Outlook) = **0.246**

Gain(S, Temp) = 0.0289366072

Gain(S, Humidity) = 0.1515496

Gain(S, Wind) = 0.048

◇ **Step 07 : Find which attribute is the root node.**

Gain(S, Outlook) = 0.246 is highest.

Therefore "Outlook" attribute is the decision attribute in the root node.

"Outlook" as root node has three possible values - **sunny, cloudy, rain.**

◇ **Step 08 : Find which attribute is next decision node.**

Outlook has three possible values,

so root node has three branches (sunny, cloudy, rain).

$S_{\text{sunny}} = \{D1, D2, D8, D9, D11\} = 5$ "examples". "D" represent "Days".

With outlook = sunny

$\text{Gain}(S_{\text{sunny}}, \text{Humidity}) = \mathbf{0.970}$

$\text{Gain}(S_{\text{sunny}}, \text{Temp}) = 0.570$

$\text{Gain}(S_{\text{sunny}}, \text{Wind}) = 0.019$

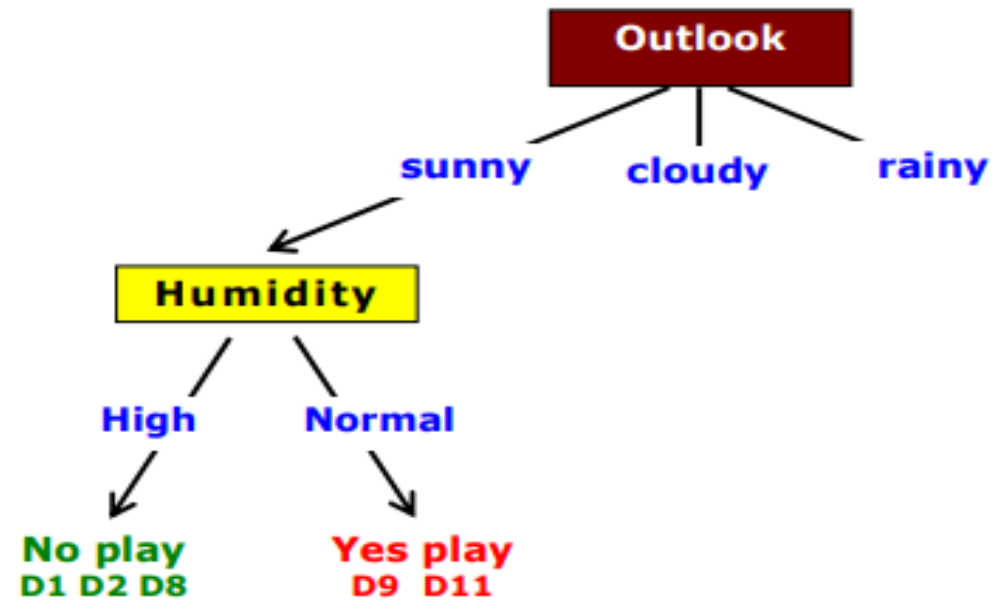
Humidity has the highest gain 0.970 is next decision node.

Humidity is having 2 factors, High and normal

D1,D2,D8=No

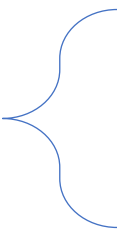
D9,D11=yes

Hence if the outlook is sunny , outing can happen on D9 and D11



Step 9: Cloudy outlook on decision

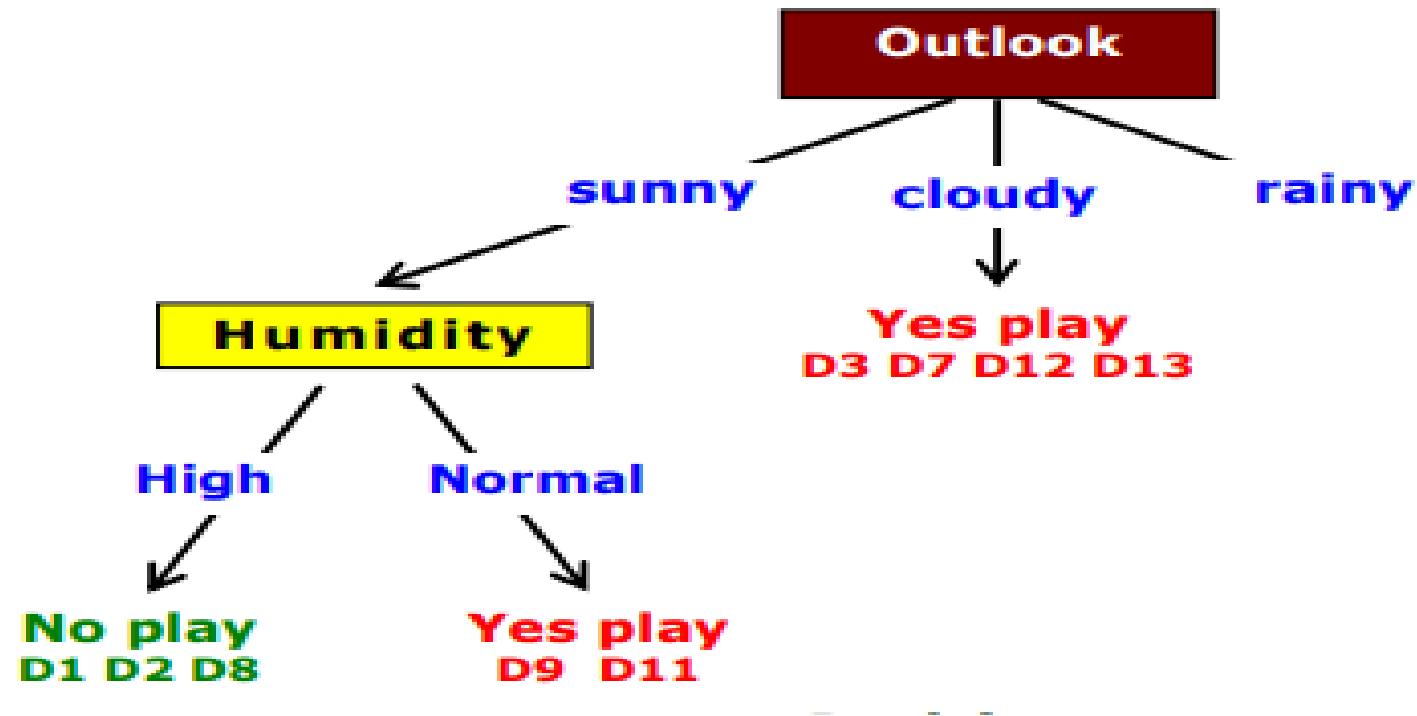
Decision will always be “yes” if outlook were “cloudy”



S.No	Day	Outlook	Temp		Humidity		Wind	Play
1	11	sunny	75	Medium	70	Normal	strong	yes
2	2	sunny	80	Hot	90	High	strong	no
3	1	sunny	85	Hot	85	High	week	no
4	8	sunny	72	Medium	95	High	week	no
5	9	sunny	69	Cold	70	Normal	week	yes
6	12	cloudy	72	Medium	90	High	strong	yes
7	3	cloudy	83	Hot	78	High	week	yes
8	7	cloudy	64	Cold	65	Normal	strong	yes
9	13	cloudy	81	Hot	75	Normal	week	yes
10	14	rainy	71	Medium	85	High	strong	no
11	6	rainy	65	Cold	70	Normal	strong	no
12	10	rainy	75	Medium	80	Normal	week	yes
13	5	rainy	68	Cold	80	Normal	week	yes
14	4	rainy	70	Medium	96	High	week	yes

- D3, D7 ,D12,D13=yes

Hence if the outlook is sunny , outing can happen on D3, D7 ,D12 and D13



- **Step 10: Rainy outlook on decision**

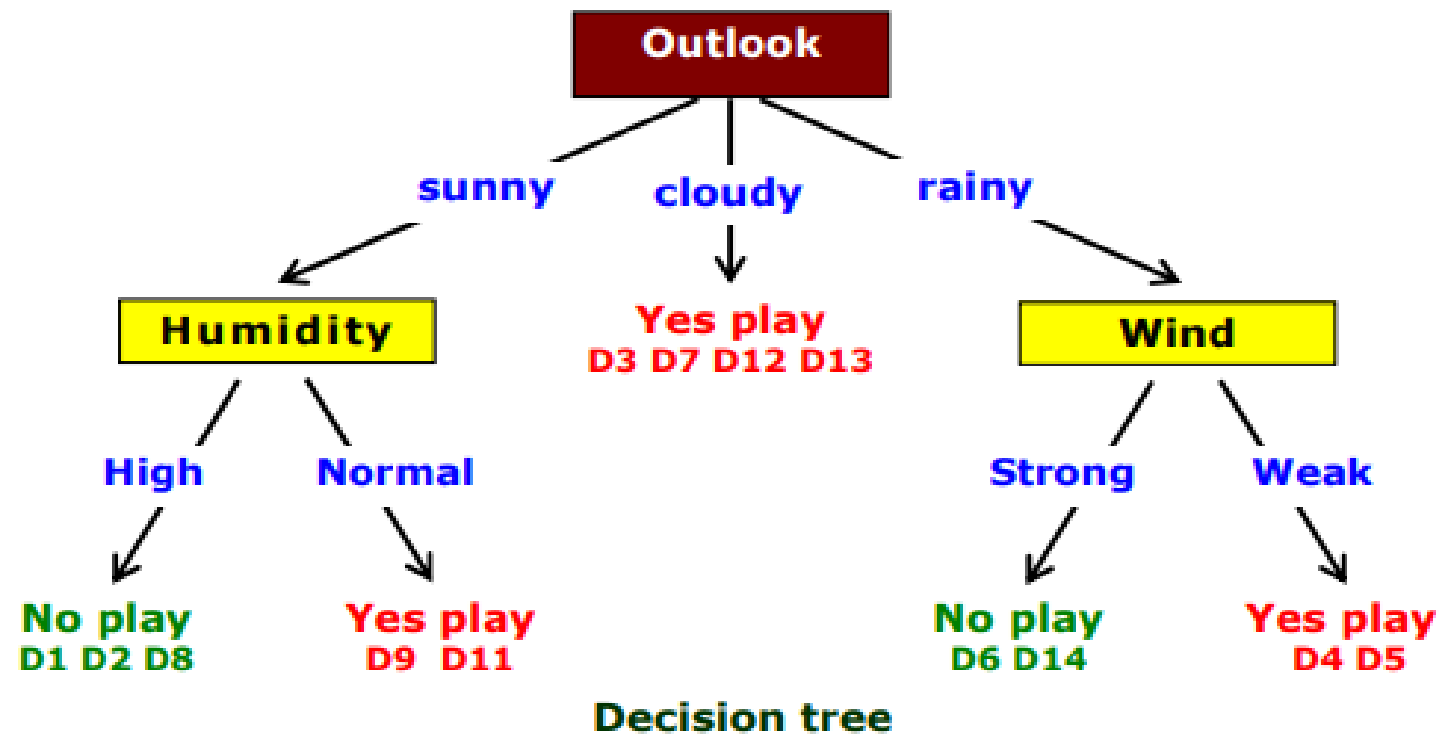
Gain (Rainy, Temperature)

Gain (Rainy, Humidity)

Gain (Rainy, wind) ----- **correct decision**

- **Wind Produces the highest score if outlook were rainy, because if you compare the other 2 attributes with play/decision there are conflicts.**
- Wind is having 2 factors, Strong and weak
D6,D14=No
D4, D5, D10=yes
Hence if the outlook is rainy , outing can happen on D4, D5 and D10

- **Note:** This process gets completed only if all days data are classified perfectly or run out of attributes.
- Thus the classification tree built using ID3 algorithm is shown below. It tells if weather was amenable to play?



UNIT - V

Expert Systems:

Representing and Using Domain Knowledge,
Shell, Explanation, Knowledge Acquisition.

What is Expert System?

- **Expert System** is an interactive and reliable computer-based decision-making system which uses both facts and heuristics to solve complex decision-making problems. It is considered at the highest level of human intelligence and expertise. The purpose of an expert system is to solve the most complex issues in a specific domain.
- The Expert System in AI can resolve many issues which generally would require a human expert. It is based on knowledge acquired from an expert. Artificial Intelligence and Expert Systems are capable of expressing and reasoning about some domain of knowledge. Expert systems were the predecessor of the current day artificial intelligence, deep learning and machine learning systems.

What are Expert Systems?

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.

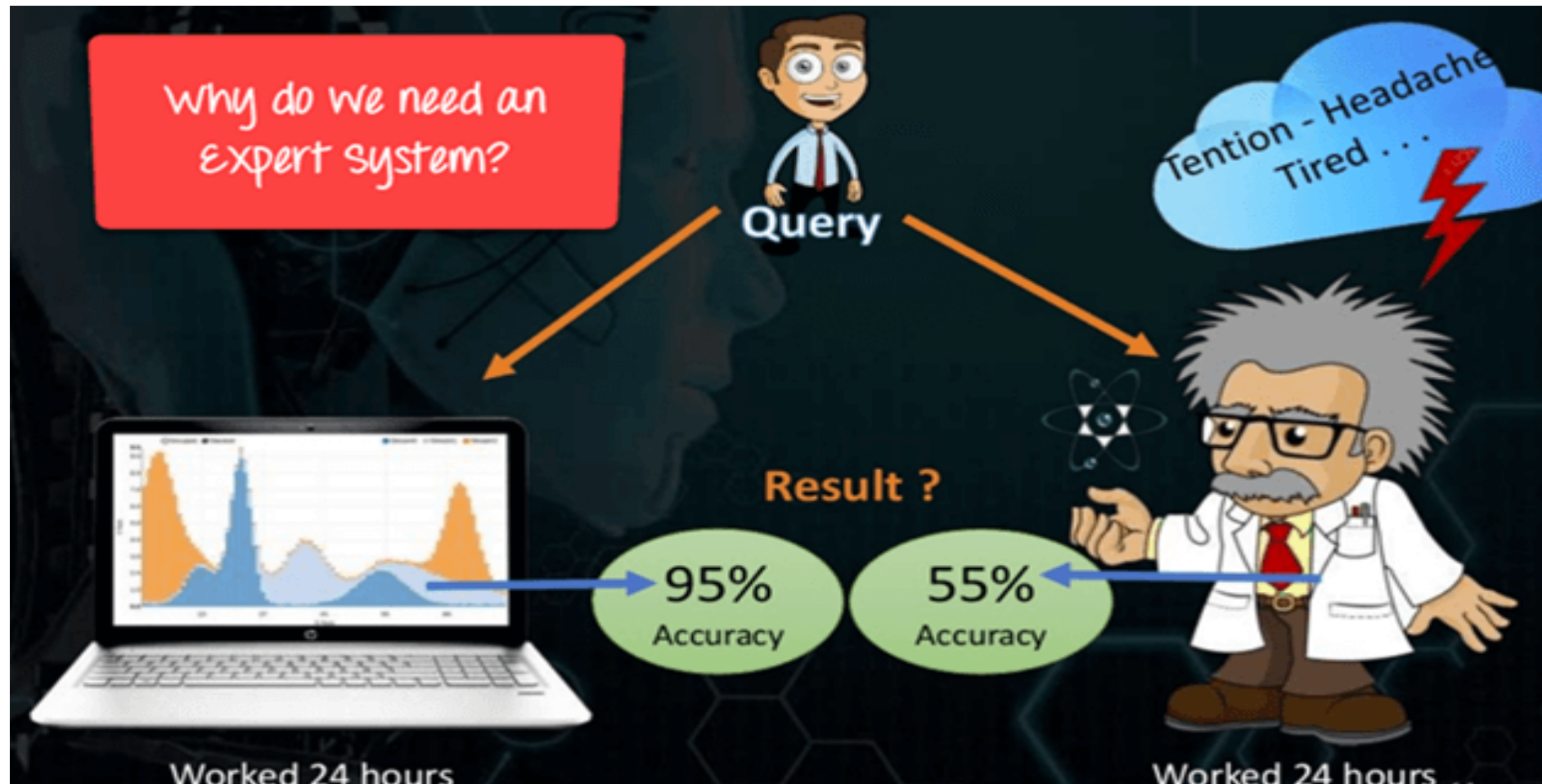
Characteristics of Expert Systems

- **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.

- **Right on Time Reaction:** An Expert System in Artificial Intelligence interacts in a very reasonable period of time with the user. The total time must be less than the time taken by an expert to get the most accurate solution for the same problem.
- **Flexible:** It is vital that it remains flexible as it the is possessed by an Expert system.
- **Effective Mechanism:** Expert System in Artificial Intelligence must have an efficient mechanism to administer the compilation of the existing knowledge in it.
- **Capable of handling challenging decision & problems:** An expert system is capable of handling challenging decision problems and delivering solutions.

Three common methods of knowledge representation evolved over the years are IF-THEN rules, Semantic networks and Frames. (already discussed in previous classes)

Why Expert systems are required?



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

They are incapable of –

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

Examples of the Expert System:

Below are some popular examples of the Expert System:

- **DENDRAL:** It was an artificial intelligence project that was made as a chemical analysis expert system. It was used in organic chemistry to detect unknown organic molecules with the help of their mass spectra and knowledge base of chemistry.
- **MYCIN:** It was one of the earliest backward chaining expert systems that was designed to find the bacteria causing infections like bacteraemia and meningitis. It was also used for the recommendation of antibiotics and the diagnosis of blood clotting diseases.
- **PXDES:** It is an expert system that is used to determine the type and level of lung cancer. To determine the disease, it takes a picture from the upper body, which looks like the shadow. This shadow identifies the type and degree of harm.
- **CaDeT:** The CaDet expert system is a diagnostic support system that can detect cancer at early stages.

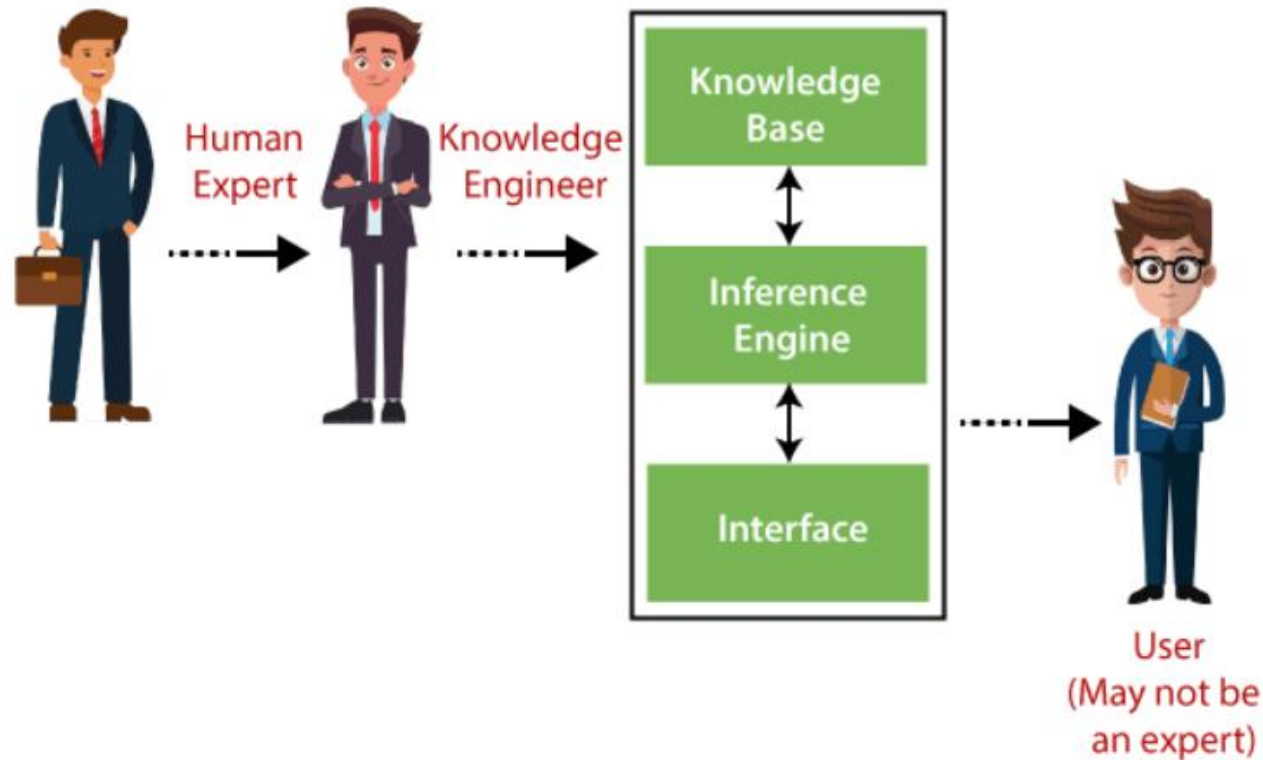
The process of Building An Expert Systems

- Determining the characteristics of the problem
- Knowledge engineer and domain expert work in coherence to define the problem
- The knowledge engineer translates the knowledge into a computer-understandable language. He designs an inference engine, a reasoning structure, which can use knowledge when needed.
- Knowledge Expert also determines how to integrate the use of uncertain knowledge in the reasoning process and what type of explanation would be useful.

Components of Expert System

An expert system mainly consists of three components:

- **User Interface**
- **Inference Engine**
- **Knowledge Base**



1. 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.**

2. 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**.
- **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.

3. Knowledge Base

- The knowledgebase is a type of storage that stores knowledge acquired from the different experts of the particular domain. It is considered as big storage of knowledge. The more the knowledge base, the more precise will be the Expert System.
- It is similar to a database that contains information and rules of a particular domain or subject.
- One can also view the knowledge base as collections of objects and their attributes. **Such as a Lion is an object and its attributes are it is a mammal, it is not a domestic animal, etc.**

Components of Knowledge Base

- **Factual Knowledge:** The knowledge which is based on facts and accepted by knowledge engineers comes under factual knowledge.
- **Heuristic Knowledge:** This knowledge is based on practice, the ability to guess, evaluation, and experiences.
- **Knowledge Representation:** It is used to formalize the knowledge stored in the knowledge base using the If-else rules.
- **Knowledge Acquisitions:** It is the process of extracting, organizing, and structuring the domain knowledge, specifying the rules to acquire the knowledge from various experts, and store that knowledge into the knowledge base.



Knowledge Extraction Process

Participant in Expert Systems Development

Participant	Role
Domain Expert	He is a person or group whose expertise and knowledge is taken to develop an expert system.
Knowledge Engineer	Knowledge engineer is a technical person who integrates knowledge into computer systems.
End User	It is a person or group of people who are using the expert system to get to get advice which will not be provided by the expert.

Conventional System vs. Expert System

Conventional System	Expert System
Knowledge and processing are combined in one unit.	Knowledge database and the processing mechanism are two separate components.
The programme does not make errors (Unless error in programming).	The Expert System may make a mistake.
The system is operational only when fully developed.	The expert system is optimized on an ongoing basis and can be launched with a small number of rules.
Step by step execution according to fixed algorithms is required.	Execution is done logically & heuristically.
It needs full information.	It can be functional with sufficient or insufficient information.

Human expert vs. Expert System

Human Expert	Artificial Expertise
Perishable	Permanent
Difficult to Transfer	Transferable
Difficult to Document	Easy to Document
Unpredictable	Consistent
Expensive	Cost effective System

Why Expert System?

Why Expert
System



No emotion

High Efficiency

Expertise in a domain

No Memory limitation

Regular updates improve the
performance

High Security

Considers all facts

- **No memory Limitations:** It can store as much data as required and can memorize it at the time of its application. But for human experts, there are some limitations to memorize all things at every time.
- **High Efficiency:** If the knowledge base is updated with the correct knowledge, then it provides a highly efficient output, which may not be possible for a human.
- **Expertise in a domain:** There are lots of human experts in each domain, and they all have different skills, different experiences, and different skills, so it is not easy to get a final output for the query. But if we put the knowledge gained from human experts into the expert system, then it provides an efficient output by mixing all the facts and knowledge
- **Not affected by emotions:** These systems are not affected by human emotions such as fatigue, anger, depression, anxiety, etc.. Hence the performance remains constant.
- **High security:** These systems provide high security to resolve any query.
- **Considers all the facts:** To respond to any query, it checks and considers all the available facts and provides the result accordingly. But it is possible that a human expert may not consider some facts due to any reason.
- **Regular updates improve the performance:** If there is an issue in the result provided by the expert systems, we can improve the performance of the system by updating the knowledge base.

Applications of Expert System:

- The following table shows where ES can be applied.

Application	Description
Design Domain	Camera lens design, automobile design.
Medical Domain	Diagnosis Systems to deduce cause of disease from observed data, conduction medical operations on humans.
Monitoring Systems	Comparing data continuously with observed system or with prescribed behavior such as leakage monitoring in long petroleum pipeline.
Process Control Systems	Controlling a physical process based on monitoring.
Knowledge Domain	Finding out faults in vehicles, computers.
Finance/Commerce	Detection of possible fraud, suspicious transactions, stock market trading, Airline scheduling, cargo scheduling.

Advantages of Expert System

- These systems are highly reproducible.
- They can be used for risky places where the human presence is not safe.
- Error possibilities are less if the KB contains correct knowledge.
- The performance of these systems remains steady as it is not affected by emotions, tension, or fatigue.
- They provide a very high speed to respond to a particular query.

Limitations of Expert System

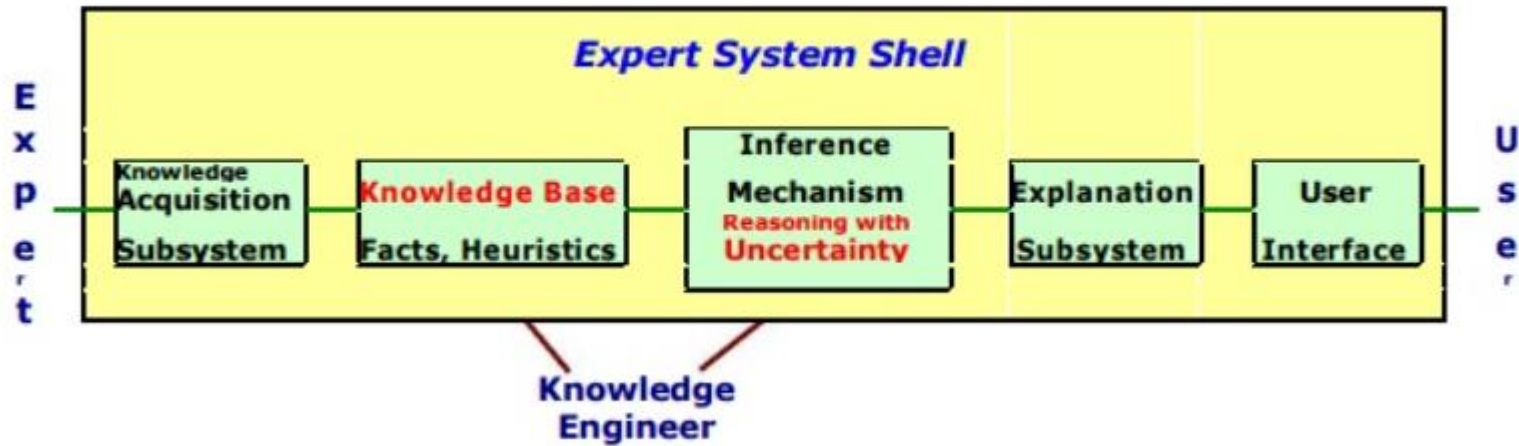
- The response of the expert system may get wrong if the knowledge base contains the wrong information.
- Like a human being, it cannot produce a creative output for different scenarios.
- Its maintenance and development costs are very high.
- Knowledge acquisition for designing is much difficult.
- For each domain, we require a specific ES, which is one of the big limitations.
- It cannot learn from itself and hence requires manual updates.

Expert System Shells

- An Expert system shell is a software development environment. It contains the basic components of expert systems. A shell is associated with a prescribed method for building applications by configuring and instantiating these components.

Shell components and description:

- The generic components of a shell : the knowledge acquisition, the knowledge Base, the reasoning, the explanation and the user interface are shown below. The knowledge base and reasoning engine are the core components.



All these components are described in the next slide.

- **Knowledge Base:**

A store of factual and heuristic knowledge. Expert system tool provides one or more knowledge representation schemes for expressing knowledge about the application domain. Some tools use both Frames (objects) and IF-THEN rules. In PROLOG the knowledge is represented as logical statements.

■ Reasoning Engine

Inference mechanisms for manipulating the symbolic information and knowledge in the knowledge base form a line of reasoning in solving a problem.

The inference mechanism can range from simple modus ponens backward chaining of IF-THEN rules to Case-Based reasoning.

Knowledge Acquisition subsystem

A subsystem to help experts in build knowledge bases. However, collecting knowledge, needed to solve problems and build the knowledge base, is the biggest bottleneck in building expert systems.

Explanation subsystem

A subsystem that explains the system's actions. The explanation can range from how the final or intermediate solutions were arrived at justifying the need for additional data.

User Interface

A means of communication with the user. The user interface is generally not a part of the expert system technology. It was not given much attention in the past. However, the user interface can make a critical difference in the perceived utility of an Expert system.

Explanation - Expert System

- Most expert systems have explanation facilities that allow the user to ask questions - why and how it reached some conclusion.

Explanation

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

Example :

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

System Does the car start ?

User No.

System Does the engine turn over ?

User Yes.

System Do you smell gas ?

User Yes

System Recommendation - Wait 5 minutes and try again.

User Why ?

System I used the rule:

If not start, and engine_turn_over, and smell_gas

Then recommendation is 'Wait 5 minutes and try again'

User

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

Types of Explanation

There are four types of explanations commonly used in expert systems.

Rule trace reports on the progress of a consultation;

‡ Explanation of how the system reached to the given conclusion;

‡ Explanation of why the system did not give any conclusion.

Explanation of why the system is asking a question;

knowledge acquisition

Introduction

- Knowledge acquisition is the process of extracting, structuring and organizing knowledge from one source, usually human experts, so it can be used in software such as an ES. This is often the major obstacle in building an ES. There are three main topic areas central to knowledge acquisition that require consideration in all ES projects. First, the domain must be evaluated to determine if the type of knowledge in the domain is suitable for an ES. Second, the source of expertise must be identified and evaluated to ensure that the specific level of knowledge required by the project is provided. Third, if the major source of expertise is a person, the specific knowledge acquisition techniques and participants need to be identified.

Theoretical Considerations

- An ES attempts to replicate in software the reasoning/pattern-recognition abilities of human experts who are distinctive because of their particular knowledge and specialized intelligence. ES should be heuristic and readily distinguishable from algorithmic programs and databases. Further, ES should be based on expert knowledge, not just competent or skillful behavior.

- **Domains**

Several domain features are frequently listed for consideration in determining whether an ES is appropriate for a particular problem domain. Several of these caveats relate directly to knowledge acquisition. First, bona fide experts, people with generally acknowledged expertise in the domain, must exist. Second, there must be general consensus among experts about the accuracy of solutions in a domain. Third, experts in the domain must be able to communicate the details of their problem solving methods. Fourth, the domain should be narrow and well defined and solutions within the domain must not require common sense.

- **Experts**

Although an ES knowledge base can be developed from a range of sources such as textbooks, manuals and simulation models, the knowledge at the core of a well developed ES comes from human experts. Although multiple experts can be used, the ideal ES should be based on the knowledge of a single expert. In light of the pivotal role of the expert, caveats for choosing a domain expert are not surprising. First, the expert should agree with the goals of the project. Second, the expert should be cooperative and easy to work with. Third, good verbal communication skills are needed. Fourth, the expert must be willing and able to make the required time commitment (there must also be adequate administrative/managerial support for this too).

- **Knowledge Acquisition Technique**

At the heart of the process is the interview. The heuristic model of the domain is usually extracted through a series of intense, systematic interviews, usually extending over a period of many months. Note that this assumes the expert and the knowledge engineer are not the same person. It is generally best that the expert and the knowledge engineer not be the same person since the deeper the experts' knowledge, the less able they are in describing their logic. Furthermore, in their efforts to describe their procedures, experts tend to rationalize their knowledge and this can be misleading.

General suggestions about the knowledge acquisition process are summarized in rough chronological order below:

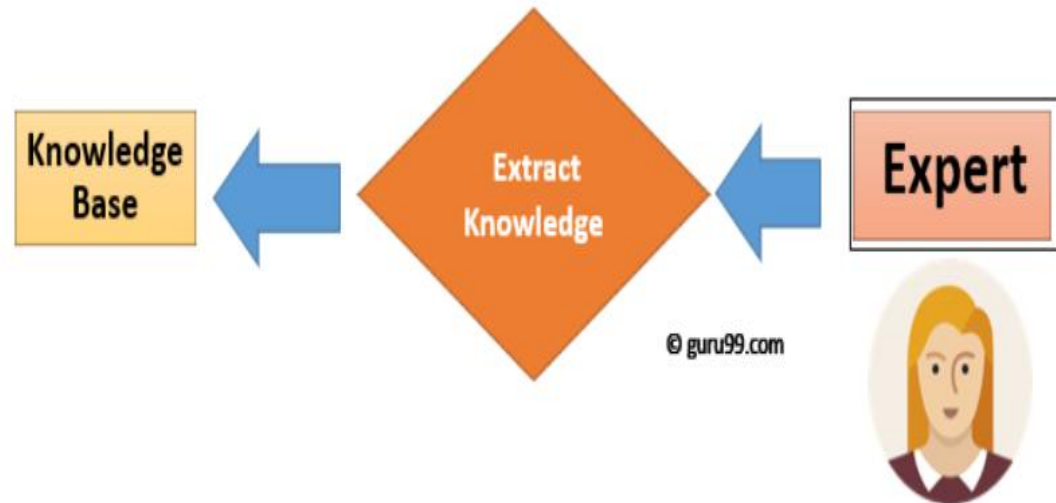
- Observe the person solving real problems.
- Through discussions, identify the kinds of data, knowledge and procedures required to solve different types of problems.
- Build scenarios with the expert that can be associated with different problem types.
- Have the expert solve a series of problems verbally and ask the rationale behind each step.
- Develop rules based on the interviews and solve the problems with them.
- Have the expert review the rules and the general problem solving procedure.
- Compare the responses of outside experts to a set of scenarios obtained from the project's expert and the ES.

Note that most of these procedures require a close working relationship between the knowledge engineer and the expert.

Practical Considerations

The preceding section provided an idealized version of how ES projects might be conducted. In most instances, the above suggestions are considered and modified to suit the particular project. The remainder of this section will describe a range of knowledge acquisition techniques that have been successfully used in the development of ES.

- **Operational Goals**
- **Pre-training**
- **Knowledge Document**
- **Scenarios**
- **Interviews**



Knowledge Extraction Process