ARTIFICIAL INTELLIGENCE

&

MACHINE LEARNING

Lecture Notes

B.Tech 3rd Year

Prepared By

Sonali Kar

Assistant Professor

Dept. of Computer Science & Engineering

Balasore college of Engineering & Technology

NH - 16, Sergarh, Balasore, 756060 - Odisha, India

6th Semester

Artificial Intelligence & Machine Learning

Module-I: (12 hours)

INTRODUCTION –The Foundations of Artificial Intelligence; - INTELLIGENT AGENTS – Agents and Environments, Good Behaviour: The Concept of Rationality, the Nature of Environments, the Structure of Agents, SOLVING PROBLEMS BY SEARCH – Problem-Solving Agents, Formulating problems, Searching for Solutions, Uninformed Search Strategies, Breadth-first search, Depth-first search, Searching with Partial Information, Informed (Heuristic) Search Strategies, Greedy best-first search, A* Search, CSP, Means-End-Analysis.

Module-II: (12 hours)
ADVERSARIAL SEARCH – Games, The Mini-Max algorithm, optimal decisions in multiplayer games, Alpha-Beta Pruning, Evaluation functions, Cutting off search, LOGICAL AGENTS –Knowledge-Based agents, Logic, Propositional Logic, Reasoning Patterns in Propositional Logic,Resolution, Forward and Backward chaining - FIRST ORDER LOGIC – Syntax and Semantics of First-Order Logic, Using First-Order Logic , Knowledge Engineering in First-Order Logic -INFERENCE IN FIRST ORDER LOGIC – Propositional vs. First-Order Inference, Unification and Lifting, Forward Chaining, Backward Chaining, Resolution

Module-III: (6 hours)
UNCERTAINTY – Acting under Uncertainty, Basic Probability Notation, The Axioms of Probability,Inference Using Full Joint Distributions, Independence, Bayes' Rule and its Use, PROBABILISTICREASONING – Representing Knowledge in an Uncertain Domain, The Semantics of BayesianNetworks, Efficient Representation of Conditional Distribution, Exact Inference in BayesianNetworks, Approximate Inference in Bayesian Networks

Module-IV: (10 hours)
LEARNING METHODS – Statistical Learning, Learning with Complete Data, Learning with HiddenVariables, Rote Learning, Learning by Taking Advice, Learning in Problem-solving, learningfrom Examples: Induction, Explanation-based Learning, Discovery, Analogy, FormalLearning Theory,Neural Net Learning and Genetic Learning. Expert Systems: Representingand Using DomainKnowledge, Expert System Shells, Explanation, Knowledge Acquisition.

Books:
[1] Elaine Rich, Kevin Knight, & Shivashankar B Nair, Artificial Intelligence, McGraw Hill,3rd ed.,2009
[2] Stuart Russell, Peter Norvig, Artificial Intelligence -A Modern Approach, 4/e, Pearson, 2003.
[3] Nils J Nilsson, Artificial Intelligence: A New Synthesis, Morgan Kaufmann Publications,2000
[4] Introduction to Artificial Intelligence & Expert Systems, Dan W Patterson, PHI.,2010
[5] S Kaushik, Artificial Intelligence, Cengage Learning, 1st ed.2011

Digital Learning Resources:
Course Name: Artificial Intelligence Search Methods For Problem
Solving Course Link:
https://swayam.gov.in/nd1_noc20_cs81/preview
Course Instructor: Prof. D. Khemani, IIT Madras
Course Name: Fundamentals of Artificial Intelligence
Course Link:

https://swayam.gov.in/nd1_noc20_me88/preview
Course Instructor: Prof. S. M. Hazarika, IIT Guwahati
Course Name: Introduction to Machine Learning
Course Link:
https://nptel.ac.in/courses/106/105/106105152 Course
Instructor: Prof. S. Sarkar, IIT Kharagpur
Course Name: Machine Learning
Course Link:
https://nptel.ac.in/courses/106/106/106106202 Course
Instructor: Prof. Carl Gustaf Jansson, IIT Madras

Module-1 **Lecture-1**

Introduction

Learning Objective:

1. Introduction

1.1 What is Artificial Intelligence?
1.2 Why we need AI?
1.3 What is intelligence in AI?
1.4 Advantages and Disadvantages of AI
1.5 Applications of AI

1. Introduction

1.1 What is Artificial Intelligence?

- It is the branch of computer science that emphasizes the development of intelligence machines, thinking and working like humans and able to make decisions. It is also known as Machine Intelligence.
- According to the father of Artificial Intelligence, John McCarthy, it is "The science and engineering of making intelligent machines, especially intelligent computer programs".
- Artificial Intelligence is a way of making a computer, a computer-controlled robot, or a software think intelligently, in the similar manner the intelligent humans think.
- AI is accomplished by studying how human brain thinks and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

1.2 Why we need AI?

- To create expert systems: The systems which exhibit intelligent behaviour with the capability to learn, demonstrate, and explain and advice its users.
- To implement human intelligence in machines: Creating systems that understand, think, learn and behave like humans. Helping machines find solutions to complex problems like humans do and applying them as algorithms in a computer friendly manner.

1.3 What is intelligence in AI?

- The ability of a system to calculate, perceive relationships and analogies, learn from experience, store and retrieve information from memory, solve problems, use natural language fluently, classify and adapt new situations.
- The Intelligence is intangible.
- It is composed of

a) Reasoning
b) Learning
c) Problem solving
d) Perception
e) Linguistic intelligence

a) **Reasoning:** It is the set of processes that enables us to provide basis for judgement, making decisions, and prediction.
b) **Learning:** It is the activity of gaining knowledge or skill by studying, practising, being taught, or experiencing something. Learning enhances the awareness of the subjects of the study.
c) **Problem solving:** Problem solving also includes decision making, which is the process of selecting the best suitable alternative out of multiple alternatives to reach the desired goal are available.
d) **Perception**: It is the process of acquiring, interpreting, selecting, and organizing sensory information.

e) **Linguistic Intelligence**: It is one's ability to use, comprehend, speak, and write the verbal and written language.

1.4 Advantages and Disadvantages of AI

Advantages:

- **High Accuracy with less error:** AI machines or systems are prone to less errors and high accuracy as it takes decisions as per pre-experience or information.
- **High-Speed:** AI systems can be of very high-speed and fast-decision making.
- **High reliability:** AI machines are highly reliable and can perform the same action multiple times with high accuracy.
- **Useful for risky areas:** AI machines can be helpful in situations such as defusing a bomb, exploring the ocean floor, where to employ a human can be risky.
- **Digital Assistant:** AI can be very useful to provide digital assistant to the users such as AI technology is currently used by various E-commerce websites to show the products as per customer requirement.
- **Useful as a public utility:** AI can be very useful for public utilities such as a self-driving car which can make our journey safer and hassle-free, facial recognition for security purpose, Natural language processing to communicate with the human in human-language, etc.

Disadvantages:

- **High Cost:** The hardware and software requirement of AI is very costly as it requires lots of maintenance to meet current world requirements.
- **Can't think out of the box:** Even we are making smarter machines with AI, but still they cannot work out of the box, as the robot will only do that work for which they are trained, or programmed.
- **Unemployment**
- **No feelings and emotions:** AI machines can be an outstanding performer, but still it does not have the feeling so it cannot make any kind of emotional attachment with human, and may sometime be harmful for users if the proper care is not taken.
- **Increase dependency on machines:** With the increment of technology, people are getting more dependent on devices and hence they are losing their mental capabilities.
- **No Original Creativity:** As humans are so creative and can imagine some new ideas but still AI machines cannot beat this power of human intelligence and cannot be creative and imaginative.

1.5 Applications of AI

i. Gaming

ii. Natural language processing

iii. Expert systems

iv. Speech Recognition

v. Handwriting Recognition

vi. Intelligent robots

vii. Computer vision etc

Module-1 **Lecture-2**

Learning Objective:

2. Agents in Artificial Intelligence
2.1 Agent
2.2 Intelligent Agent

2. Agents in Artificial Intelligence

2.1 Agent

- An agent is anything that can perceive its environment through sensors and acts upon that environment through actuators.
- An agent can be:

Human agent: A human agent has eyes, ears, and other organs which work for sensors and hand, legs, vocal tract and other body parts work as actuators.
Robotic Agent: A robotic agent can have cameras, infrared range finder, NLP for sensors and various motors for actuators.
Software Agent: Software agent can have keystrokes, file contents, which act as sensors and display on the screen, files etc act as actuators.

Before moving forward, we should first know about sensors, effectors, and actuators.

Sensor: Sensor is a device which detects the change in the environment and sends the information to other electronic
Devices: An agent observes its environment through sensors.
Actuators: Actuators are the component of machines that converts energy into motion. The actuators are only responsible for moving and controlling a system. An actuator can be an electric motor, gears, break etc.
Effectors: Effectors are the devices which affect the environment. Effectors can be legs, wheels, arms, fingers and display screen.

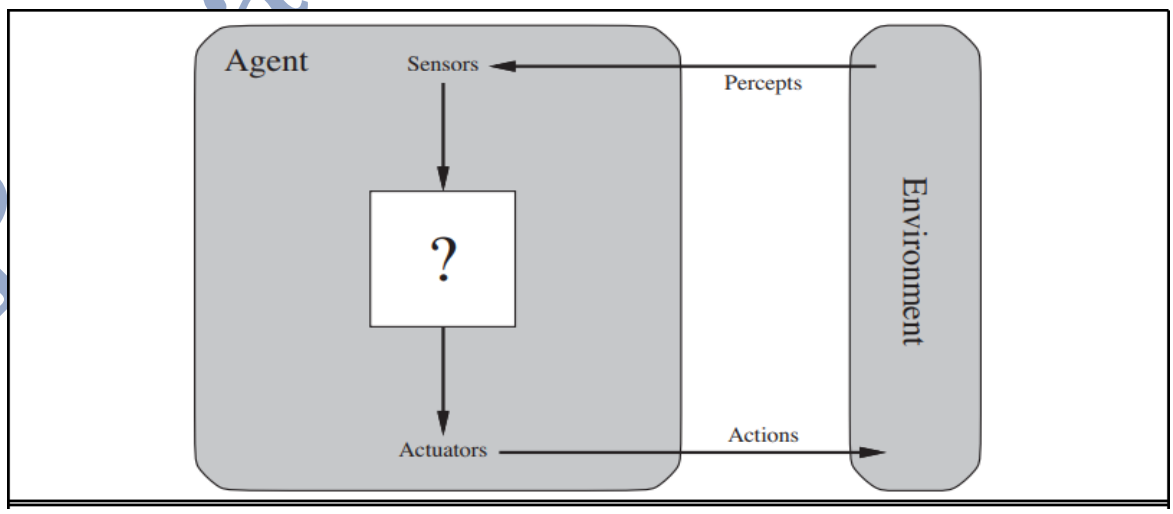


Fig: Agents interact with environments through sensors and actuators.

2.2 Intelligent Agent

- An intelligent agent is an autonomous entity which acts upon an environment using sensors and actuators for achieving goals.
- An intelligent agent may learn from the environment to achieve their goals. An intelligent agent is also called as a rational agent which is one that does the right thing.
- An intelligent agent can transform perception into actions rationally.

Following are the main four rules for an AI agent:

Rule 1: An AI agent must have the ability to perceive the environment.
Rule 2: The observation must be used to make decisions.
Rule 3: Decision should result in an action.
Rule 4: The action taken by an AI agent must be a rational action.

Learning Objective:

3. Structure of an Agent

3. Structure of an Agent

The structure of an intelligent agent is a combination of architecture and agent program.
It can be viewed as:

Agent = Architecture + Agent program

Architecture: Architecture is machinery that an AI agent executes on.
Agent program: The agent function (f) for an artificial agent will be implemented by an agent program. An agent's behaviour is described by the agent function that maps any given percept sequence to an action.
The agent function f maps from percept histories to actions:

$$f: P^* \rightarrow A$$

The part of the agent taking an action on the environment is called an actuator.

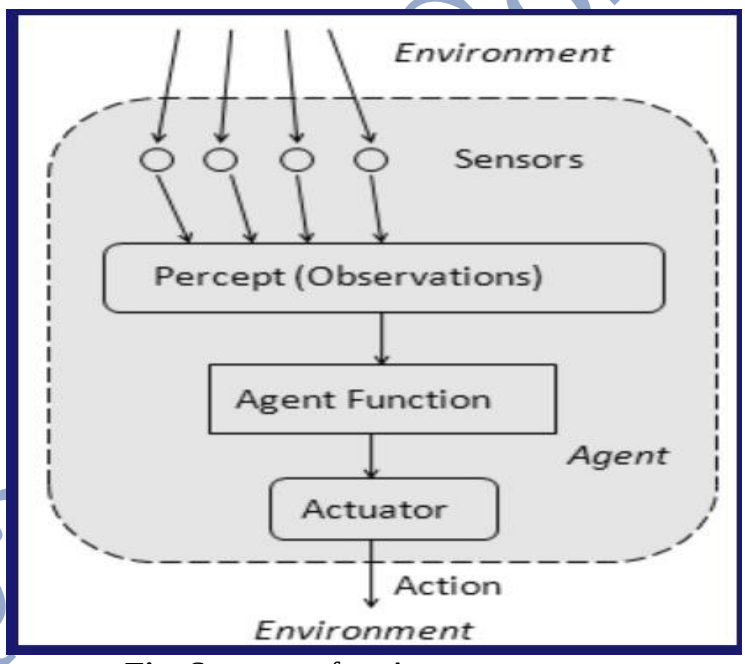
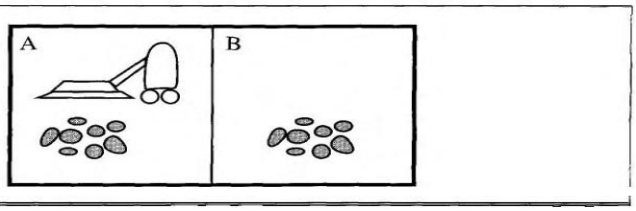


Fig: Structure of an Agent

Example:

Simple example-the vacuum-cleaner world: 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 below figure. An agent program that implements it which is mentioned below.



A vacuum-cleaner world with just two locations.

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
[A,Clean],[A,Clean],[A,Clean]	*Right*
[A,Clean],[A,Clean],[A,Dirty]	Suck

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

if *status* = *Dirty* **then return** *Suck*
else if *location* = *A* **then return** *Right*
else if *location* = *B* **then return** *Left*

The agent program for a simple reflex agent in the two-state vacuum environment. This program implements the agent function tabulated in the above figure.

Module-1 **Lecture-4**

Learning Objective:

4. Agent Environments

4.1 Features of Environment

4. Agent Environments

- An environment is everything in the world which surrounds the agent, but it is not a part of an agent itself.
- An environment can be described as a situation in which an agent is present.
- The environment is where agent lives, operate and provide the agent with something to sense and act upon it.

4.1 Features of Environment

An environment can have various features from the point of view of an agent.

1. Fully observable vs Partially Observable
2. Static vs Dynamic
3. Discrete vs Continuous
4. Deterministic vs Stochastic
5. Single-agent vs Multi-agent
6. Episodic vs sequential
7. Known vs Unknown
8. Accessible vs Inaccessible

1. Fully observable vs Partially Observable:

- If an agent sensor can sense or access the complete state of an environment at each point of time then it is a fully observable environment, else it is partially observable.
- A fully observable environment is easy as there is no need to maintain the internal state to keep track history of the world.
- An agent with no sensors in all environments then such an environment is called as unobservable.

2. Static vs Dynamic:

- If an environment does not undergo any change especially when an agent is busy in performing a specific task, then the environment is said to be static otherwise it dynamic.
- Taxi driving is an example of a dynamic environment whereas Crossword puzzles are an example of a static environment.

3. Discrete vs Continuous:

- If in an environment there are a finite number of percepts and actions that can be performed within it, then such an environment is called a discrete environment else it is called continuous environment.
- A chess game comes under discrete environment as there is a finite number of moves that can be performed.
- A self-driving car is an example of a continuous environment.

4. Deterministic vs Stochastic:

- If an agent's current state and selected action can completely determine the next state of the environment, then such environment is called a deterministic environment.
- A stochastic environment is random in nature and cannot be determined completely by an agent.

5. Single-agent vs Multi-agent:

- If only one agent is involved in an environment, and operating by itself then such an environment is called single agent environment.
- However, if multiple agents are operating in an environment, then such an environment is called a multi-agent environment.
- The agent design problems in the multi-agent environment are different from single agent environment.

6. Episodic vs Sequential:

- In an episodic environment, there is a series of one-shot actions, and only the current percept is required for the action.
- In episodic the agent's experience divided in to atomic episodes.
- Next episode not dependent on actions taken in previous episode.
- However, in Sequential environment (non episodic), an agent requires memory of past actions to determine the next best actions.

7. Known vs Unknown:

- Known and unknown are not actually a feature of an environment, but it is an agent's state of knowledge to perform an action.
- In a known environment, the results for all actions are known to the agent. While in unknown environment, agent needs to learn how it works in order to perform an action.

8. Accessible vs Inaccessible:

- If an agent can obtain complete and accurate information about the state's environment, then such an environment is called an Accessible environment else it is called inaccessible.
- An empty room whose state can be defined by its temperature is an example of an accessible environment.
- Information about an event on earth is an example of Inaccessible environment

Learning Objective:

5. Good Behaviour: The Concept of Rationality

5.1 Rational Agent

5.2 Rationality

5.3 The Nature of Environments

5. Good Behaviour: The Concept of Rationality

5.1 Rational Agent

- A rational agent is an agent which has clear preference, models uncertainty, and acts in a way to maximize its performance measure with all possible actions.
- A rational agent is said to perform the right things.
- AI is about creating rational agents to use for game theory and decision theory for various real-world scenarios.
- For an AI agent, the rational action is most important because in AI reinforcement learning algorithm, for each best possible action, agent gets the positive reward and for each wrong action, an agent gets a negative reward.

5.2 Rationality

- Rationality is concerned with expected actions and results depending upon what the agent has perceived. Performing actions with the aim of obtaining useful information is an important part of rationality.
- The rationality of an agent is measured by its performance measure. Rationality can be judged on the basis of following points:
    - The performance measures, which determine the degree of success.
    - The agent's prior knowledge about the environment.
    - The actions that the agent can perform.
    - Agent's Percept Sequence till now.

This leads to a definition of a rational agent:

- For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.
- A rational agent always performs the right action, where the right action means the action that causes the agent to be most successful in the given percept sequence.
- The problem the agent solves is characterized by Performance Measure, Environment, Actuators, and Sensors (PEAS).

5.3 The Nature of Environments

- To design a rational agent, we must specify the task environment.
- The performance measure, the environment, and the agent's actuators and sensors are grouped as the task environment, and called as PEAS (Performance measure, Environment, Actuators, Sensors).

Task Environment: PEAS for self-driving cars:

Let's suppose a self-driving car then PEAS representation will be:

Performance Measures: Safety, time, legal drive, comfort

Environment: Roads, other vehicles, road signs, pedestrian

Actuators: Steering, accelerator, brake, signal, horn

Sensors: Camera, GPS, speedometer, odometer, accelerometer, sonar.

PEAS for Medical Diagnose:

Performance Measures: Healthy patient, Minimized cost

Environment: Patient, Hospital, Staffs

Actuators: Tests, Treatements

Sensors: Keyboard (Entry of symptoms)

PEAS for Vacuum Cleaner:

Performance Measures: Cleanness, Efficiency, Battery life, Security

Environment: Room, Table, Wood floor, Carpet

Actuators: Wheels, Brushes, Vacuum Extractor

Sensors: Camera, Dirt detection sensor, Cliff sensor, Bump Sensor, Infrared Wall Sensor

Learning Objective:

6. Types of Agents

6.1 Simple Reflex Agents

6.2 Model-Based Reflex Agents

6.3 Goal-Based Agents

6. Types of Agents

In artificial intelligence, agents are entities that sense their surroundings and act to accomplish predetermined objectives. From basic reactive reactions to complex decision-making, these agents display a wide range of behaviours and skills.

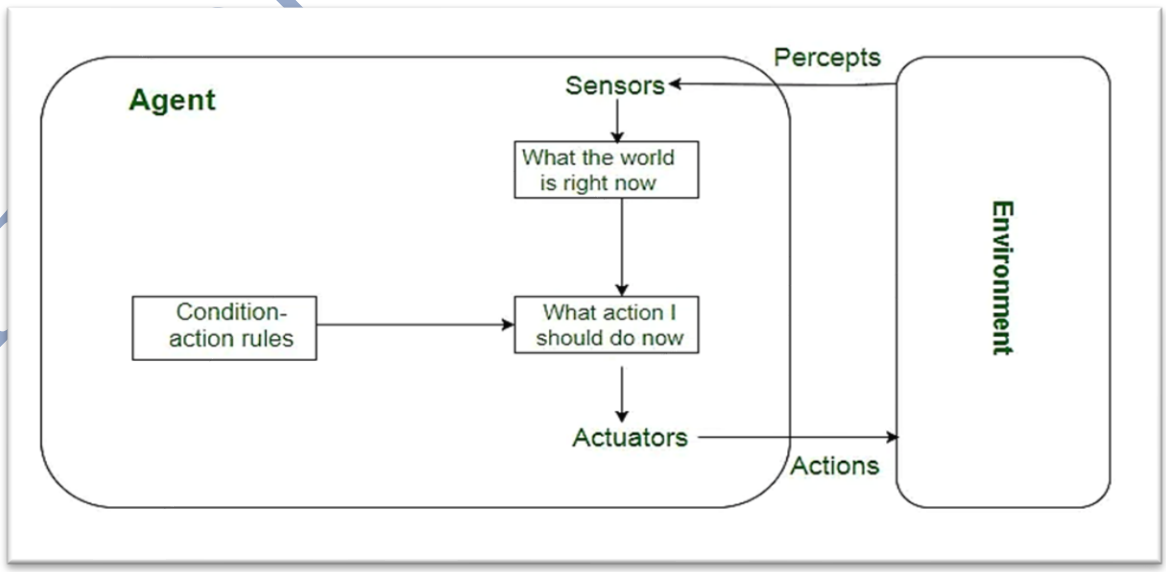
Agents can be grouped into 5 classes based on their degree of perceived intelligence and capability. These are given below :

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

6.1 Simple Reflex Agents:

The Simple reflex agents are the simplest agents. These agents take decisions on the basis of the current percepts and ignore the rest of the percept history. They have no internal state or memory and respond instantly to the current situation.

Example:- An automatic door sensor is a simple reflex agent. When the sensor detects movement near the door, it triggers the mechanism to open. The rule is: if movement is detected near the door, then open the door. It does not consider any additional context, such as who is approaching or the time of day, and will always open whenever movement is sensed.



6.2 Model-Based Reflex Agents:

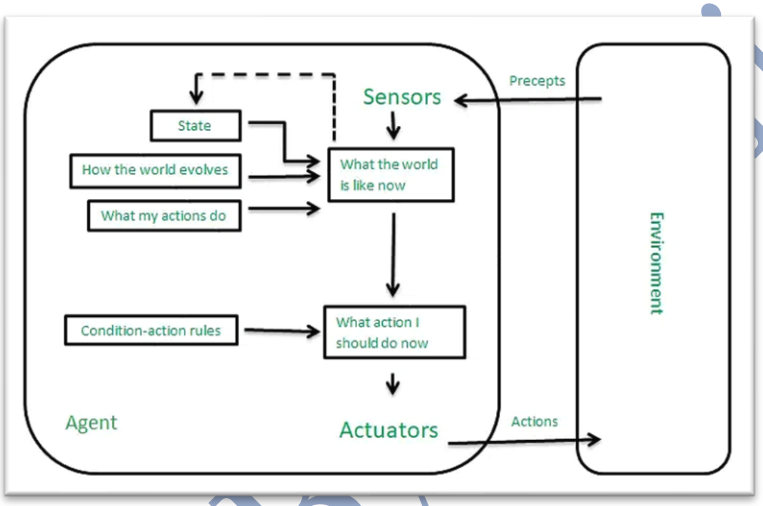
Model-based agents are more sophisticated than simple reflex agents. These agents are capable of tracking the situation and working in a partially observable environment.

A model-based agent has two important factors:

a) Model: It is knowledge about "how things happen in the world," so it is called a Model-based agent.
b) Internal State: It is a representation of the current state based on percept history.

These agents have the model, "which is knowledge of the world" and based on the model they perform actions.

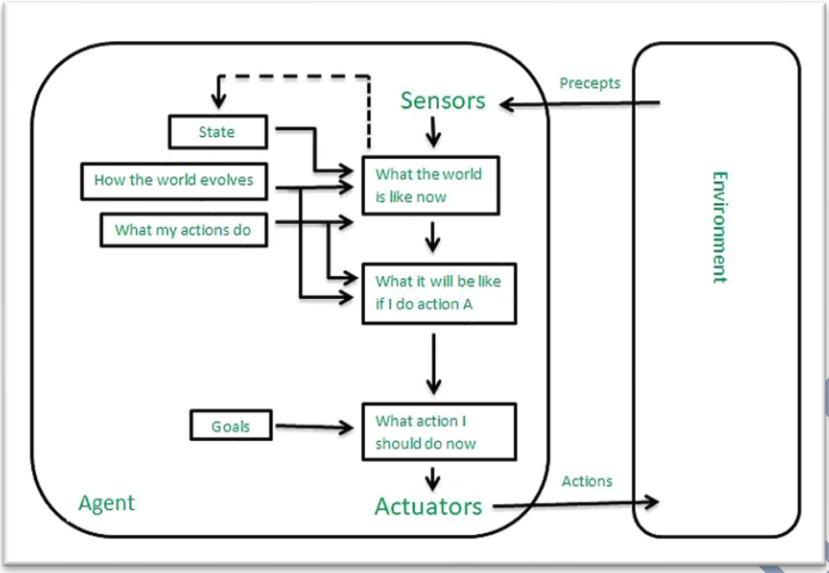
Example:- A vacuum cleaner like the **Roomba** one that maps a room and remembers obstacles like furniture. It ensures cleaning without repeatedly bumping into the same spots.



6.3 Goal-Based Agents

Goal-based agents have predefined objectives or goals that they aim to achieve. By combining descriptions of goals and models of the environment, these agents plan to achieve different objectives, like reaching particular destinations. They use search and planning methods to create sequences of actions that enhance decision-making in order to achieve goals. Goal-based agents differ from reflex agents by including forward-thinking and future-oriented decision-making processes.

Example: A delivery robot tasked with delivering packages to specific locations. It analyzes its current position, destination, available routes, and obstacles to plan an optimal path towards delivering the package.



Learning Objective:

6.4 Utility-Based Agents

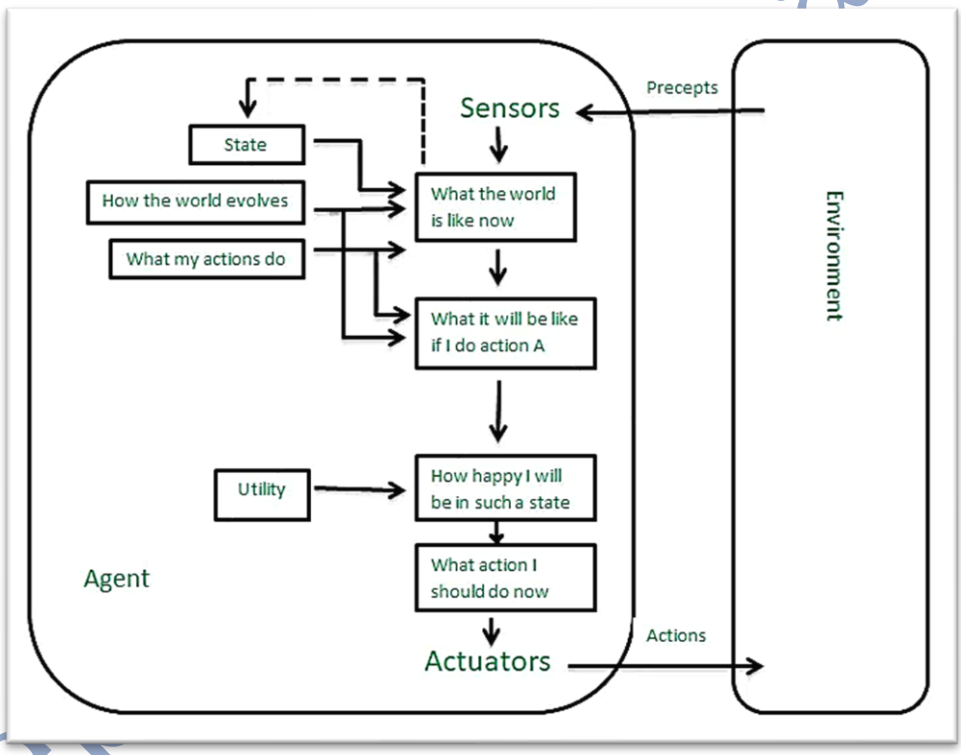
6.5 Learning Agents

6.4 Utility-Based Agents

These agents are comparable to goal-based agents, but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.

The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.

Example: An investment advisor algorithm suggests investment options by considering factors such as potential returns, risk tolerance, and liquidity requirements, with the goal of maximizing the investor's long-term financial satisfaction.



6.5 Learning Agents

In artificial intelligence, a learning agent is an agent that possesses the ability to learn from its past experiences. It starts to act with basic knowledge and then able to act and adapt automatically through learning.

A learning agent has mainly four conceptual components, which are:

Learning element: It is responsible for making improvements by learning from environment

Critic: Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.

Performance element: It is responsible for selecting external action

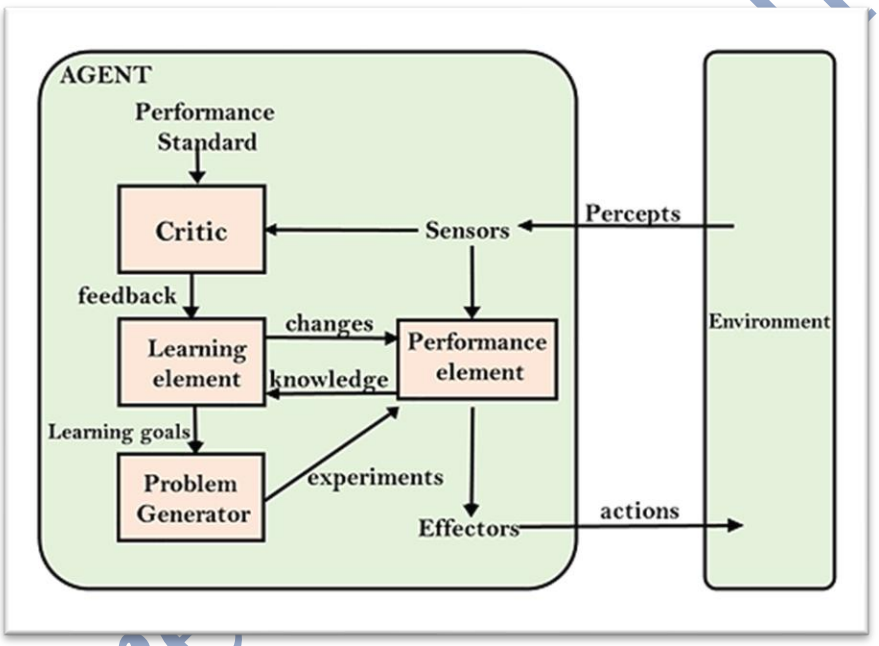
Problem generator: This component is responsible for suggesting actions that will lead to new and informative experiences.

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

Examples:

Chatbots*:* AIML is frequently used to develop chatbots that can simulate conversation with users. These chatbots use pattern matching to respond to user inputs. A classic example is the A.L.I.C.E (Artificial Linguistic Internet Computer Entity) chatbot, which learns from user interactions to provide more accurate and helpful responses.

Recommendation Systems: AIML can also be used to create recommendation systems. These agents analyze user preferences and behaviors to suggest products, services, or content. For instance, an online shopping website might use an AIML-based learning agent to recommend items to customers based on their browsing history and purchase patterns.



Module-1 Lecture-8

Learning Objective:

7. Solving Problems By Search

7.1 Problem-Solving Agents

7.2 Formulating problems

7.3 Searching for Solutions

7.1 Problem-Solving Agents:

A problem is a set of information that the agent will utilize to make decisions. A problem-solving refers to a state where we wish to reach to a definite goal from a present state or condition.

According to computer science, problem-solving is a part of artificial intelligence, which includes various approaches including heuristics and algorithms.

There are some following steps which require to solve a problem:

a. **Goal Formulation:** It is the first step in problem solving and based on the current situation and the agent's performance measure.
b. **Problem Formulation:** It is the process of deciding what actions and states to consider, given a goal.

The process of looking for a sequence of actions that reaches the goal is called **search**.

A search algorithm takes a problem as input and returns a solution in the form of an action sequence. Once a solution is found, the actions it recommends can be carried out. This is called the **execution phase.**

Thus, we have a simple "formulate, search, execute" design for the agent. After formulating a goal and a problem to solve, the agent calls a search procedure to solve it.

It then uses the solution to guide its actions, doing whatever the solution recommends as the next thing to do typically, the first action of the sequence and then removing that step from the sequence. Once the solution has been executed, the agent will formulate a new goal.

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

Problem solving components

A problem can be defined formally by five components

i. **Initial state:** The first component that describes the problem is the initial state that the agent starts in.
ii. **Action:** A description of the possible actions available to the agent. Given a particular state s, ACTIONS(s) returns the set of actions that can be executed in s.
iii. **Transition Model:** A description of what each action does; the formal name for this is the transition model, specified by a function RESULT(s, a) that returns the state that results from doing action a in state s. We also use the term successor to refer to any state reachable from a given state by a single action.
Together, the initial state, actions, and transition model implicitly define the state space of the problem. The state space forms a directed network or graph in which the nodes are states and the links between nodes are actions. A path in the state space is a sequence of states connected by a sequence of actions
iv. **Goal Test:** It determines whether a given state is a goal state.
v. **Path Cost:** A path cost function that assigns a numeric cost to each path. The problem-solving agent chooses a cost function that reflects its own performance measure.

Example problems: 8-puzzle,8-queens problem, The travelling salesman problem etc.

7.2 Formulating problems

The formulation is reasonable, but it is still a model, an abstract mathematical description and not the real thing.

The process of removing detail from a representation is called ***abstraction.***

7.3 Searching for Solutions

- A solution is an action sequence, so search algorithms work by considering various possible action sequences.
- The possible action sequences starting at the initial state form a search tree with the initial state at the root; the branches are actions and the nodes correspond to states in the state space of the problem.
- Expanding the current state; that is, applying each legal action to the current state, thereby generating a new set of states. The current state is the parent node, newly generated states are child nodes.
- Leaf node is a node with no children in the tree. The set of all leaf nodes available for expansion at any given point is called the **frontier.**
- The process of expanding nodes on the frontier continues until either a solution is found or there are no more states to expand.
- Search algorithms all share this basic structure they vary primarily according to how they choose which state to expand next is called as **search strategy.**

Module-1 **Lecture-9**

Learning Objective:

8. Searching

8.1 Uniformed Search

8.1.1 Breadth-first Search

8. Searching

- In AIML searching is the process of finding a solution or a path from an initial state to a goal state, usually within a search space.
- A search space is essentially a set of all possible states or configurations of a system or environment. In AI, search is a key technique used for problems such as puzzle solving, pathfinding, decision making, and optimization.
- Searching allows an agent or algorithm to explore various possible solutions in an intelligent manner and find the optimal or desired solution.
- There are different types of search algorithms, mainly categorized into:
  a) Uninformed Search (Blind Search)
  b) Informed Search (Heuristic Search)

8.1.1 Uninformed Search (Blind Search):

- The Uninformed Search is also called as Blind Search do not have any additional information about the goal beyond the initial state. These can also do is generate successors and distinguish a goal state from a non-goal state.
- They explore the search space systematically without any heuristics to guide the search.
- There present different types of uninformed search algorithms, they are
  1) Breadth-first search
  2) Depth-first search
  3) Uniform-cost search
  4) Depth-limited search
  5) Iterative deepening depth-first search
  6) Bidirectional search

8.1.1 Breadth-first Search

- Breadth-first search is the most common search strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.
- Here all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.
- Breadth-first search is an instance of the general graph-search algorithm in which the shallowest unexpanded node is chosen for expansion. This is achieved very simply by using a FIFO queue for the frontier. The new nodes go to the back of the queue, and old nodes, which are shallower than the new nodes, get expanded first. There is one slight tweak on the general graph-search algorithm, which is that the goal test is applied to each node when it is generated rather than when it is selected for expansion.
- Thus, breadth-first search always has the shallowest path to every node on the frontier.

Algorithm:

Step 1: Place the starting node on the queue.

Step 2: If the queue is empty, return failure and stop.

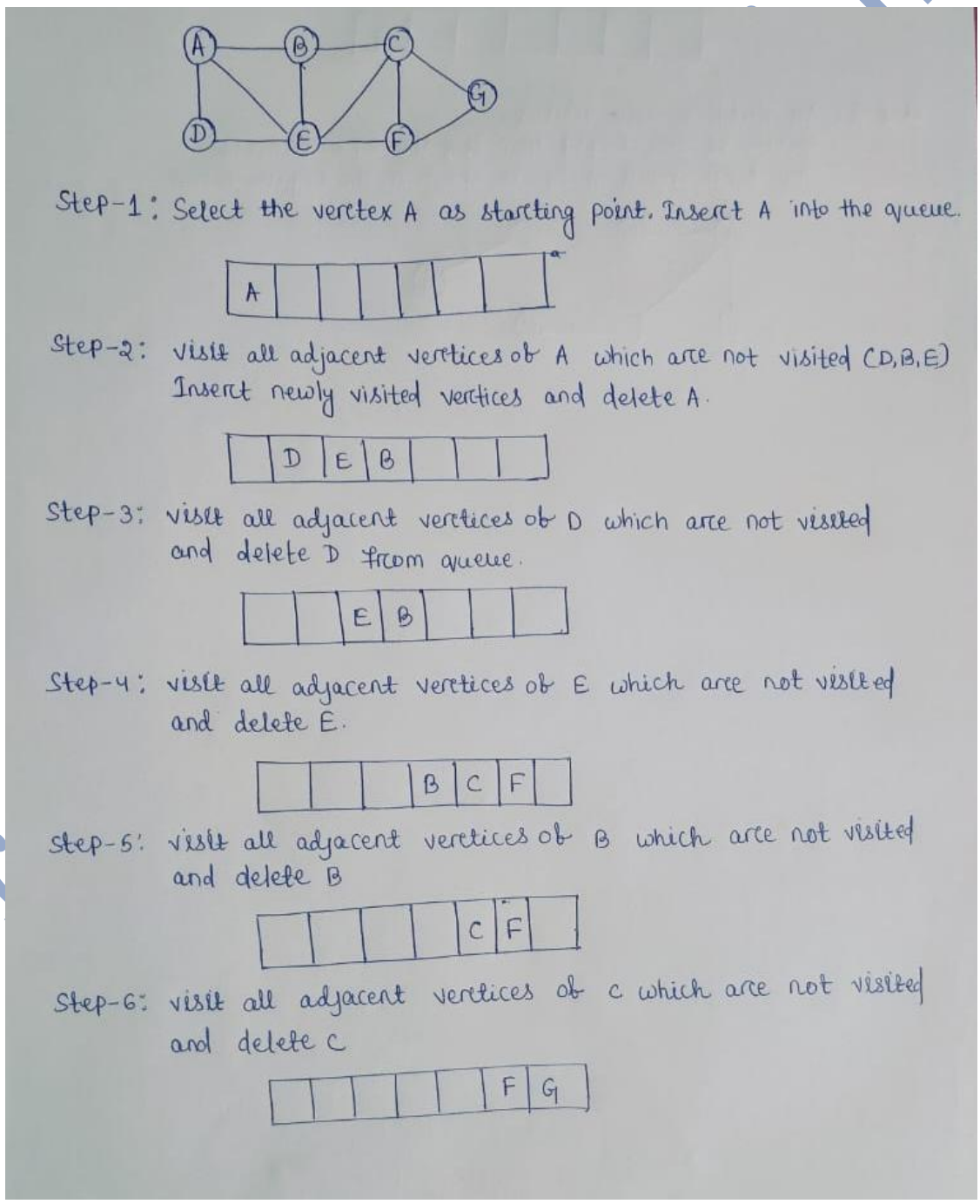
Step 3: If the first element on the queue is a goal node return success and stop. Otherwise,

Step 4: Remove and expand the first element from the queue and place all the children at the end of the queue in any order.

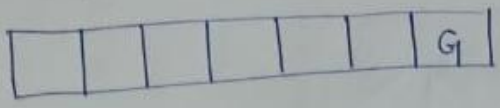
Step 5: If queue is empty Go to Step 6 else go to step 3.

Step 6: Exit.

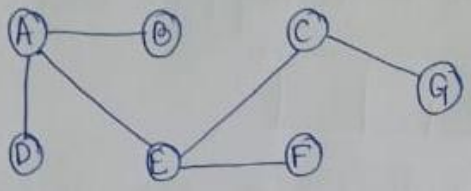
Example:



Step-7: visit all adjacent vertices of F which are not visited and delete F



Step-8: All vertices are visited.
Queue became Empty. Now stop the BFS process.
The final result of BFS is shown in the figure below



Time Complexity: The Time Complexity of BFS algorithm is O (b^d).

Space Complexity: The Space Complexity of BFS algorithm is O (b^d).

Advantages:

- **Simplicity**: This algorithm is easy to understand and implement using a queue.
- **Systematic Exploration**: Explores all nodes level by level, ensuring no node is missed within the same depth before moving deeper.
- **Wide Range of Applications**: BFS is versatile, applied in areas like web crawling, social network analysis, and AI-based problem-solving.

Disadvantages:

- **High Memory Usage**: BFS requires storing all nodes at the current level in memory, which can grow significantly in large or densely connected graphs.
- **Slow for Deep Solutions**: If the solution lies deep in the graph, BFS can become inefficient as it explores all shallower nodes first.

Module-1 **Lecture-10**

Learning Objective:

8. Searching

8.1 Uniformed Search

8.1.2 Depth-first Search

8.1.2 Depth-first Search

- Depth First Search (DFS) algorithm is a recursive algorithm for searching all the vertices of a graph or tree data structure. This algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.
- DFS uses a stack data structure for its implementation

Algorithm:

Step 1: PUSH the starting node into the stack.

Step 2: If the stack is empty then stops and return failure.

Step 3: If the top node of the stack is the goal node, then stop and return success.

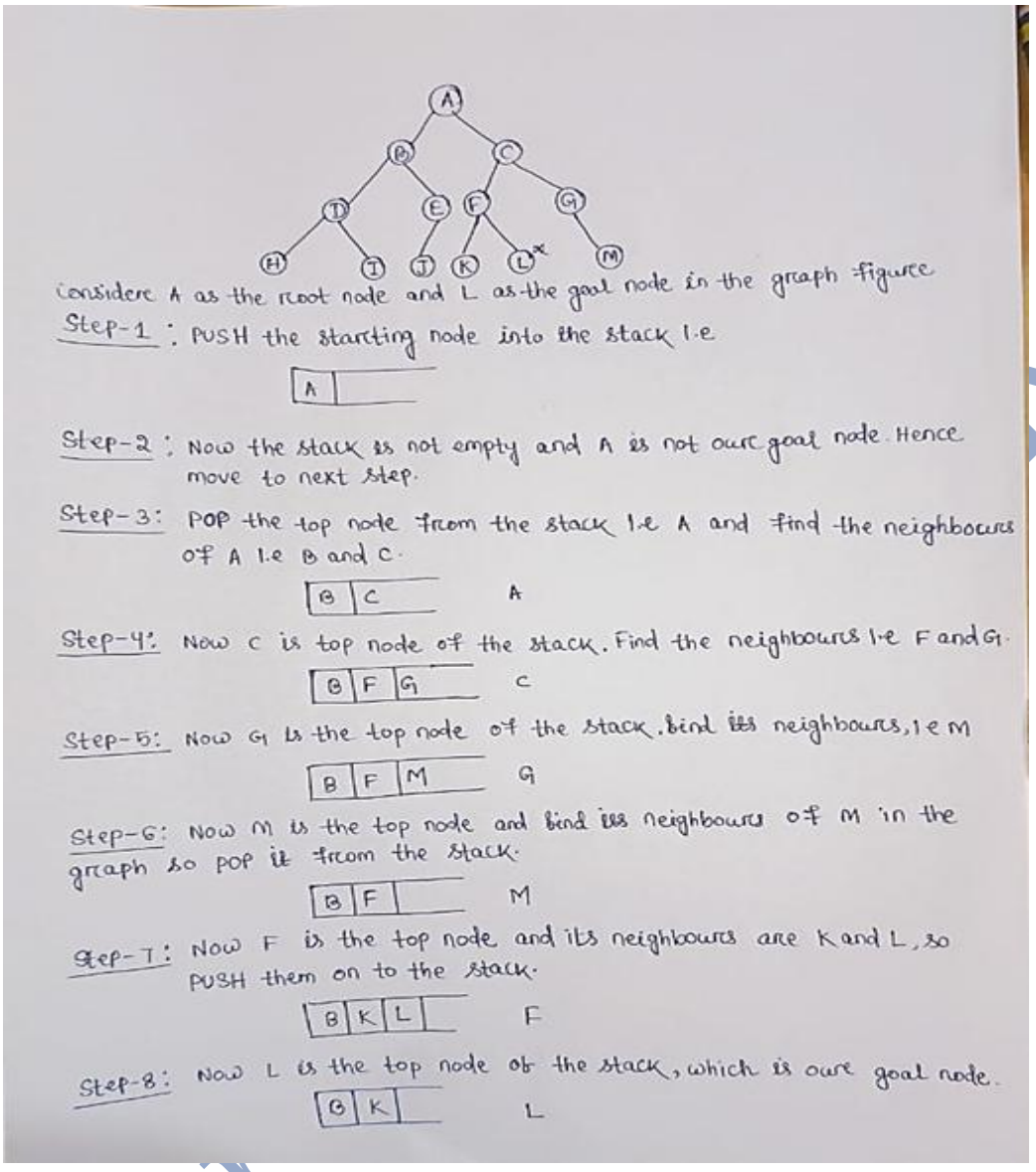
Step 4: Else POP the top node from the stack and process it. Find all its neighbours that are in ready state and PUSH them into the stack in any order.

Step 5: If stack is empty Go to step 6 else Go to step 3.

Step 6: Exit

Example:

Let us take an example for implementing DFS algorithm.



Time Complexity: The Time Complexity of DFS algorithm is O (b^d).

Space Complexity: The Space Complexity of DFS algorithm is O (b^d).

Advantages:

- DFS consumes very less memory space.
- It will reach the goal node in a less time period than BFS if it traverses in a right path.
- It may find a solution without examining much of the search because we may get the desired solution in the very first go.

Disadvantages:

- It is possible that many states keep reoccurring. There is no guarantee of finding the goal node.
- Sometimes the states may also enter into infinite loops**.**

Module-1 **Lecture-11**

Learning Objective:

9. Searching

9.1 Informed(Heuristic) Search

9.1.1 Greedy best-first Search

9.1.2 A* Search

9.1 Informed(Heuristic) Search

- Informed search algorithms are a type of search algorithm that uses heuristic functions to guide the search process.
- A heuristic function is a function that maps from problem state descriptions to measure of desirability usually represented as number. The purpose of heuristic function is to guide the search process in the most profitable directions by suggesting which path to follow first when more than is available.
- Generally a term heuristic is used for any advice that is effective but is not guaranteed to work in every case. For example in case of travelling sales man (TSP) problem we are using a heuristic to calculate the nearest neighbour. Heuristic is a method that provides a better guess about the correct choice to make at any junction that would be achieved by random guessing. This technique is useful in solving though problems which could not be solved in any other way. Solutions take an infinite time to compute.
- There are different types of informed search techniques are present
  Greedy Best-first Search/ Best first Search
  A* Search

9.1.1 Greedy best-first Search/ Best first Search

- Best first search is an instance of graph search algorithm in which a node is selected for expansion based on evaluation function f (n). Traditionally, the node which is the lowest evaluation is selected for the explanation because the evaluation measures distance to the goal.
- Best first search can be implemented within general search frame work via a priority queue, a data structure that will maintain the fringe in ascending order of f values.
- It is the combination of depth first and breadth first search algorithm.
- Best first search algorithm is often referred greedy algorithm this is because they quickly attack the most desirable path as soon as its heuristic weight becomes the most desirable.

Algorithm:

Step 1: Place the starting node or root node into the queue.

Step 2: If the queue is empty, then stop and return failure.

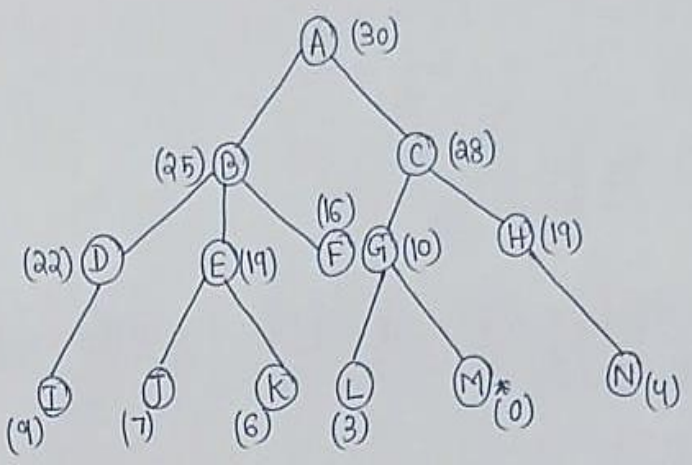
Step 3: If the first element of the queue is our goal node, then stop and return success.

Step 4: Else, remove the first element from the queue. Expand it and compute the estimated goal distance for each child. Place the children in the queue in ascending order to the goal distance.

Step 5: Go to step-3

Step 6: Exit.

Example:



Step-1 :

Considere the node A as our root node. So the first element of the queue is A which is not our goal node, so remove it from the queue and find it's neighbour that are to inserted in ascending order.

A	

Step-2 :

The neighbours of A are B and C. They will be inserted into the queue in ascending order.

B	C		A

Step-3 :

Now B is on the FRONT end of the queue. So calculate the neighbours of B that are maintaining a least distance from the root.

F	E	D	C	B

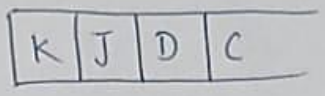
Step-4 :

Now the node F is on the queue. But as it has no further children so remove it from the queue and processed further.

E	D	C	F

Step-5

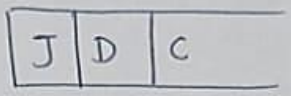
Now E is the FRONT end. So the children of E are J and K. Insert them into the queue in ascending order.



E

Step-6

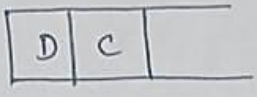
Now K is the FRONT end and as it has no further children, so remove it and proceed further.



K

Step-7

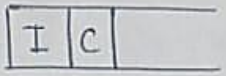
Also J has no corresponding children. So remove it and proceed further.



J

Step-8

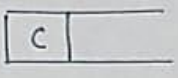
Now D is on the FRONT end and calculates the children of D and put it into the queue.



D

Step-9

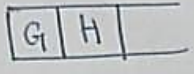
Now I is the front ~~end~~ node and it has no children. So processed further after removing this node from the queue



I

Step-10

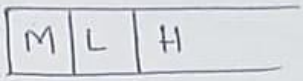
Now C is the FRONT node. So calculate the neighbours of C that are to be inserted in ascending order into the queue.



C

Step-11

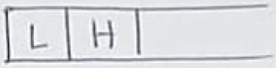
Now remove G from the queue and calculate its neighbour that is to insert in ascending order into the queue.



G

Step-12

Now M is the FRONT node of the queue which is our goal node. So Stop here and exit.



M

Time Complexity: The worst case time complexity of Greedy best first search is $O(b^m)$.

Space Complexity: The worst case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.

Advantage:

- It is more efficient than that of BFS and DFS.
- Time complexity of Best first search is much less than Breadth first search.

Disadvantages:

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

Learning Objective:

9. Searching

9.1 Informed(Heuristic) Search

9.1.1 Greedy best-first Search

9.1.2 A* Search

9.1.2 A* Search

- A* is a powerful graph traversal and pathfinding algorithm widely used in artificial intelligence and computer science. This algorithm is a specialization of best-first search.
- It is mainly used to find the shortest path between two nodes in a graph, given the estimated cost of getting from the current node to the destination node.
- A* requires heuristic function to evaluate the cost of path that passes through the particular state. This algorithm is complete if the branching factor is finite and every action has fixed cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. It can be defined by following formula.

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

Where,

f(n): The actual cost path from the start state to the goal state.

g(n): The actual cost path from the start state to the current state.

h(n): The actual cost path from the current state to goal state.

Algorithm:

Step-1: Place the starting node in the OPEN list.

Step-2: If OPEN list is empty, then stop and return failure.

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

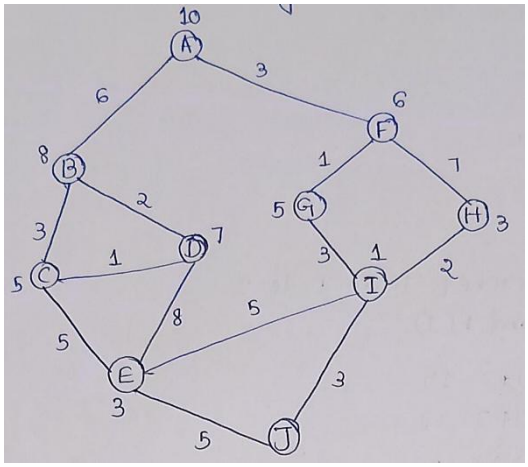
Step-4: Expand node n and generate all of its successors, and put n into the closed list. For every successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into OPEN list.

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

Step 6: Return to Step 2.

Example:-

Find the most cost effective path to reach from start state A to final state J using A* Algorithm.



Ans:-

The numbers written on edges represent the distance between the nodes.

The numbers written on nodes represent the heuristic value.

Stap-1:

We start with node A. Node B and Node F can be reached from node A.

Here we calculate f(B) and f(F) by using A* Algorithm.

f(B) = 6+8=14

f(F) = 3+6=9

As f(F)<F(B), we go to node F

Path A ➟ F

Step-2:

Node G and node H can be reached from node F.

Here we calculates f(G) and f(H)

f(G) = (3+1)+5= 9

f(H) = (3+7)+3= 13

As f(G)<f(H), we go to node G.

Path A ➟ F ➟ G

Step-3:

Node I can be reached from node G.

Here we calculate f(I)

f(I)= (3+1+3)+1= 8

Here we go to node I

Path A ➞ F ➞ G ➞ I

Step-4:

Node E,H and J can be reached from node I.

We calculate f(E),f(H) and f(J)

f(E) = (3+1+3+5)+3=15

f(H) = (3+1+3+2)+3=12

f(J) = (3+1+3+3)+0 =10

As f(J) is least, go to node J.

Path A ➞ F ➞ G ➞ I ➞ J

Time Complexity: The time complexity of A* search algorithm is O (b^d).

Space Complexity: The space complexity of A* search algorithm is O (b^d).

Advantages:

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

Disadvantages:

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

Module-1 Lecture-13

Learning Objective:

10. Constraint Satisfaction Problems (CSP)

10.1 Crypt Arithmetic Problem

10. Constraint Satisfaction Problems (CSP)

Constraint Satisfaction Problems (CSP) play a crucial role in artificial intelligence (AI) as Iit solves various problems that require decision-making under certain constraints. CSPs represent a class of problems where the goal is to find a solution that satisfies a set of constraints. These problems are commonly encountered in fields like scheduling, planning, resource allocation, and configuration.

A constraint satisfaction problem consists of three components, X, D, and C:

X is a set of variables, {X1,...,Xn}.

D is a set of domains, {D1,...,Dn}, one for each variable.

C is a set of constraints that specify allowable combinations of value.

Popular Problems with CSP

The following problems are some of the popular problems that can be solved using CSP:

1. Crypt Arithmetic Problem (Coding alphabets to numbers.)
2. n-Queens Problem
3. Sudoku
4. Map Coloring
5. Crossword

10.1 Crypt Arithmetic Problem

Cryptarithmetic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacement either with alphabets or other symbols. In cryptarithmetic problem, the digits (0-9) get substituted by some possible alphabets or symbols. The task in cryptarithmetic problem is to substitute each digit with an alphabet to get the result arithmetically correct.

Rules:

The rules or constraints on a cryptarithmetic problem are as follows:

- There should be a unique digit to be replaced with a unique alphabet.
- No two letters have same value.
- The result should satisfy the predefined arithmetic rules, i.e., 2+2 =4, nothing else.
- Digits should be from 0-9 only.
- There should be only one carry forward, while performing the addition operation on a problem.
- The problem can be solved from both sides, i.e., lefthand side (L.H.S), or righthand side (R.H.S)

Example:

Let's understand the crypt arithmetic problem as well as it's constraints better with the help of an example:

```
   T O
+  G O
 ------
 O U T
```

Step-1:

These alphabets are replaced by numbers such that all the constraints are satisfied.So initially we have all blank spaces.

We start from left most side, there we have left most symbol is : O

It is the letter which is generated by carrying.

So carry generated can be only one, so we have O=1.

When we are doing addition of n letters & result of adding

of letters is n+1, then resulted letter value is always 1 as carry.

LETTER	DIGIT
T	
O	1
G	
U	

Step-2:

Next we have T+G=U & O+O=T

We will go for O+O=T first.

We have O=1, so O+O=1+1=2(T)

LETTER	DIGIT
T	2
O	1
G	
U	

Step-3:

Next we have T+G=U

We T=2, so 2+G=U

Now here we know U must generate carry so 2+G must be 10 or greater that 10 means we must add such number in 2 so, that we can get carry generated(or we can add 10 or more than 10).

We have first option (if we consider G=9, i.e 2+G as 2+9(G)=11, here we get U=1)

But we can't chose U=1 as 1 is already assigned to O.

We have second option (if we consider G=8, i.e 2+Gas 2+8(G)=10, here we get U=0)

Which can be chosen & then we can tally the answer as follows:

```
  2 1
+ 8 1
--------
1 0 2
```

LETTER	DIGIT
T	2
O	1
G	8
U	0

Module-1 Lecture-14

Learning Objective:
11. Means-End-Analysis

11. Means-End-Analysis:

- A collection of search strategies that can reason either forward or backward but for a problem one direction or the other must be chosen, but a mixture of the two directions is appropriate for solving a complex and large problem.
- Such a mixed strategy, make it possible that first to solve the major part of a problem and then go back and solve the small problems arise during combining the big parts of the problem. Such a technique is called Means-Ends Analysis.
- Means-Ends Analysis is problem-solving techniques used in Artificial intelligence for limiting search in AI programs.
- It is a mixture of Backward and forward search technique.
- The means end analysis process centers around the detection of dfifferences between the current state and the goal state.

How means-ends analysis Works:

The means-ends analysis process can be applied recursively for a problem. It is a strategy to control search in problem-solving.

Following are the main Steps which describe the working of MEA techniques for solving a problem.

1. First, evaluate the difference between Initial State and final State.

2. Select the various operators which can be applied for each difference.

3. Apply the operator at each difference, which reduces the difference between the current state and goal state.

Operator Subgoaling:

In the Mean end analysis process, we detect the differences between the current state and goal state. Once these differences occur, then we can apply an operator to reduce the differences. But sometimes it is possible that an operator cannot be applied to the current state. So we create the sub problem of the current state, in which operator can be applied, such type of backward chaining in which operators are selected, and then sub goals are set up to establish the preconditions of the operator is called Operator Subgoaling.

Algorithm of Means-Ends Analysis

Step 1: Compare CURRENT to GOAL, if there are no differences between them then return.

Step-2: Otherwise, select the most important difference and reduce it by doing the following steps until success or failure occurs:

a) Select a new operator O which is applicable for the current difference, and if there is no such operator, then signal failure.

b) Attempt to apply operator O to CURRENT. Make a description of two states.

i) O-START, a state in which O's preconditions are satisfied.

ii) O-RESULT, the state that would result if O were applied In O-START.

c) If

(First-Part ⟵ MEA (CURRENT, O-START))

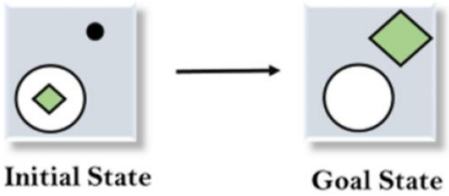
And

(LAST-Part ⟵ MEA (O-Result, GOAL)),

are successful, then signal Success and return the result of combining FIRST-PART, O, and LAST-PART.

Example:

Let's take an example where we know the initial state and goal state as given below. In this problem, we need to get the goal state by finding differences between the initial state and goal state and applying operators.

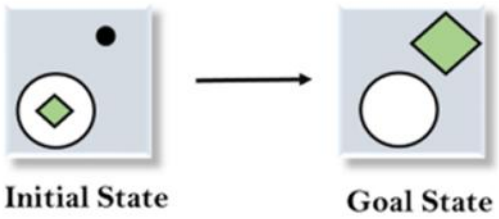


Solution:

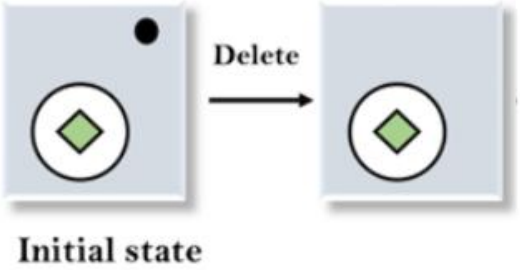
To solve the above problem, we will first find the differences between initial states and goal states, and for each difference, we will generate a new state and will apply the operators. The operators we have for this problem are:

- Move
- Delete
- Expand

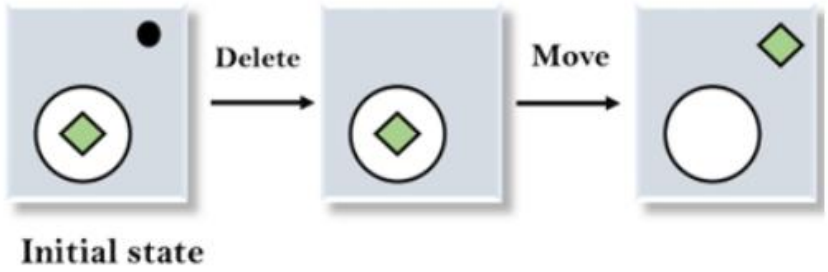
1. Evaluating the initial state: In the first step, we will evaluate the initial state and will compare the initial and Goal state to find the differences between both states.



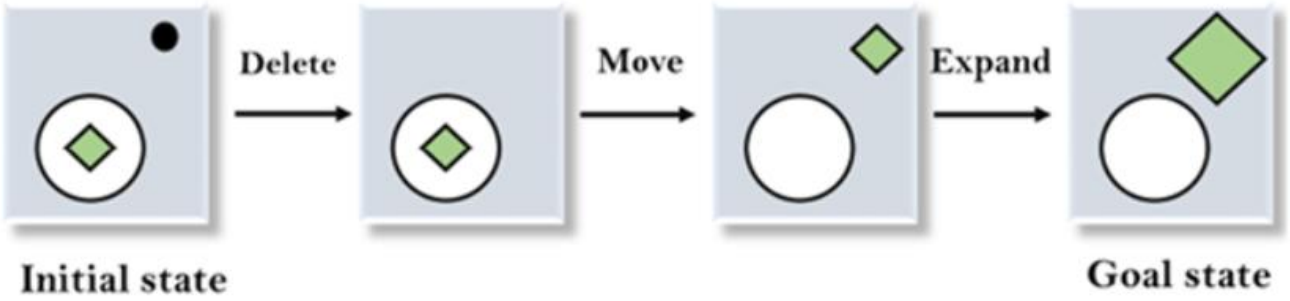
2. Applying Delete operator: As we can check the first difference is that in goal state there is no dot symbol which is present in the initial state, so, first we will apply the Delete operator to remove this dot.



3. Applying Move Operator: After applying the Delete operator, the new state occurs which we will again compare with goal state. After comparing these states, there is another difference that is the square is outside the circle, so, we will apply the Move Operator.



4. Applying Expand Operator: Now a new state is generated in the third step, and we will compare this state with the goal state. After comparing the states there is still one difference which is the size of the square, so, we will apply Expand operator, and finally, it will generate the goal state.



Module-2 **Lecture-15**

Learning Objective:

12 Adversarial Search

12.1 Game Playing

12.2 Game Tree

12 Adversarial Search:

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

12.1 Game Playing

- Game playing is an important domain of AI.
- Games do not require much knowledge, the only knowledge we need to provide is the rules, legal moves and the conditions of winning or losing the game.
- The most common search techniques in game playing are :
  (i) Mini Max algorithm
  (ii)Alpha Beta Pruning

12.2 Game Tree

A game tree is a tree where nodes of the tree are the game states and edges of the tree are the moves by players. Game tree involves initial state, action function/successor function, and result Function/utility function.

Optimal Decisions in Games:

We will consider games with two players, whom we will call MAX and MIN. MAX moves first, and then they take turns moving until the game is over. At the end of the game, points are awarded to the winning player and penalties are given to the loser. A game can be formally defined as a kind of search problem with the following components:

- The initial state, which includes the board position and identifies the player to move
- A successor function, which returns a list of (move, state) pairs, each indicating a legal move and the resulting state.
- A terminal test, which determines when the game is over. States where the game has ended are called terminal states.
- A utility function (also called an objective function or payoff function), which gives a numeric value for the terminal states. In chess, the outcome is a win, loss, or draw, with values +1,-1, or 0.

The initial state and the legal moves for each side define the game tree for the game. The below figure shows part of the game tree for tic-tac-toe (noughts and crosses). From the initial state, MAX has nine possible moves. Play alternates between MAX's placing an X and MIN'S placing an O until reach leaf nodes corresponding to terminal states such that one player has three in a row or all the squares are filled. The number on each leaf node indicates the utility value of the terminal state from the point of view of MAX: high values are assumed to be good for MAX and bad for MIN (which is how the players get their names). It is MAX's job to use the search tree (particularly the utility of terminal states) to determine the best move.

Example: Tic-Tac-Toe game tree: The following figure is showing part of the game-tree for tic-tac-toe game. Following are some key points of the game:

- There are two players MAX and MIN.
- Players have an alternate turn and start with MAX.
- MAX maximizes the result of the game tree.
- MIN minimizes the result.

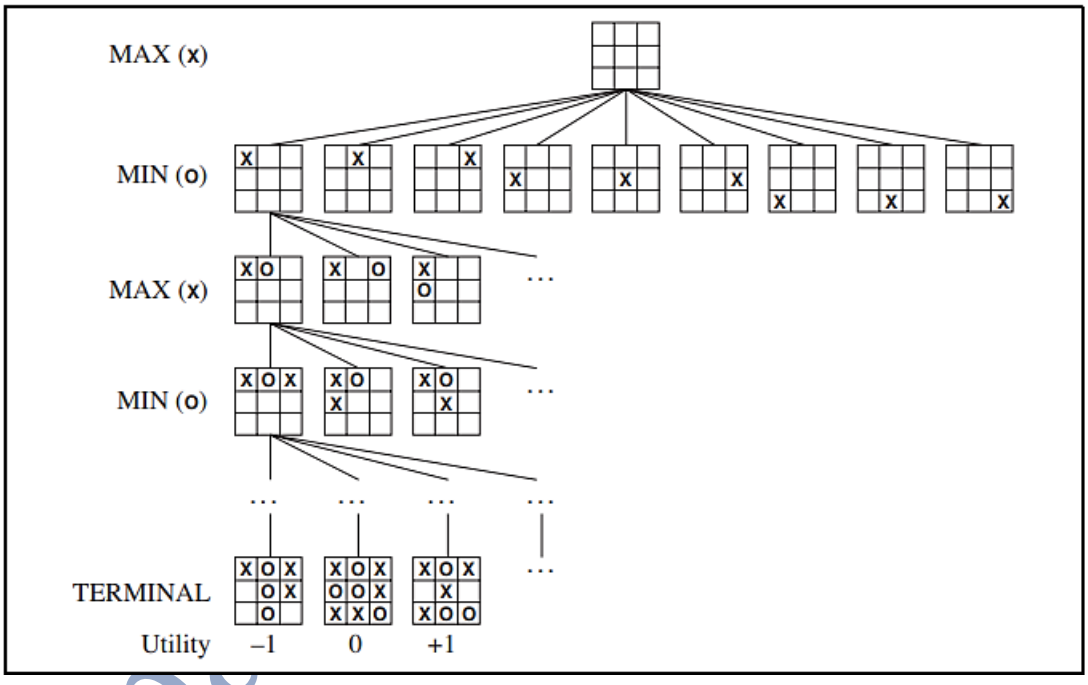


Fig:- Game Tree of Tic-Tac-Toe game

Explanation:

- From the initial state, MAX has 9 possible moves as he starts first. MAX place x and MIN place o, and both players play alternatively until we reach a leaf node where one player has three in a row or all squares are filled.
- Both players will compute each node, minimax, the minimax value which is the best achievable utility against an optimal adversary.
- Suppose both the players are well aware of the tic-tac-toe and playing the best play. Each player is doing his best to prevent another one from winning. MIN is acting against Max in the game.
- So in the game tree, we have a layer of Max, a layer of MIN, and each layer is called as Ply. Max place x, then MIN puts o to prevent Max from winning, and this game continues until the terminal node.

- In this either MIN wins, MAX wins, or it's a draw. This game-tree is the whole search space of possibilities that MIN and MAX are playing tic-tac-toe and taking turns alternately.

In a given game tree, the optimal strategy can be determined from the minimax value of each node, which can be written as MINIMAX(n). MAX prefer to move to a state of maximum value and MIN prefer to move to a state of minimum value then:

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Learning Objective:
13. Mini-Max Algorithm

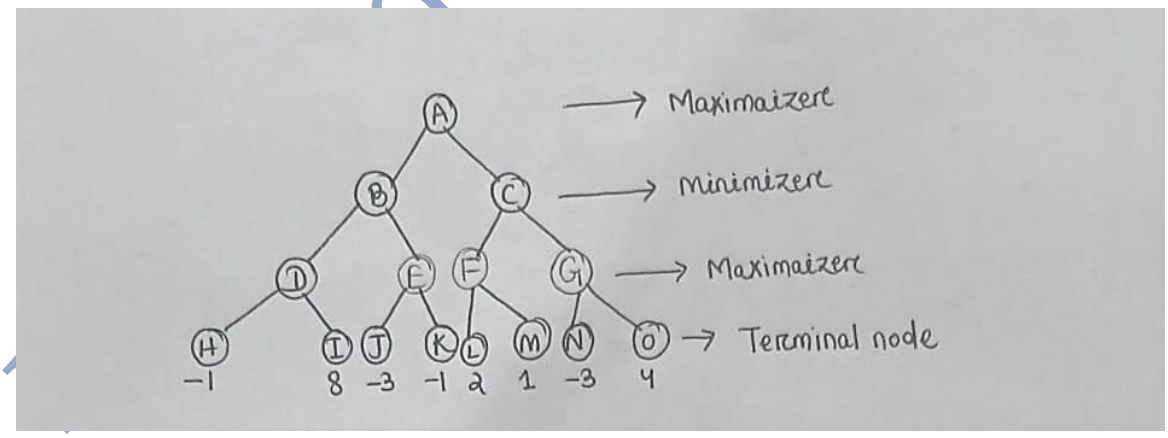
13. Mini-Max Algorithm

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally. This algorithm uses recursion to search through the game-tree.
- This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Here both the Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Working of Mini-Max Algorithm:

Step-1: In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states.

In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst case initial value = -∞, and minimizer will take next turn which has worst-case initial value = +∞.



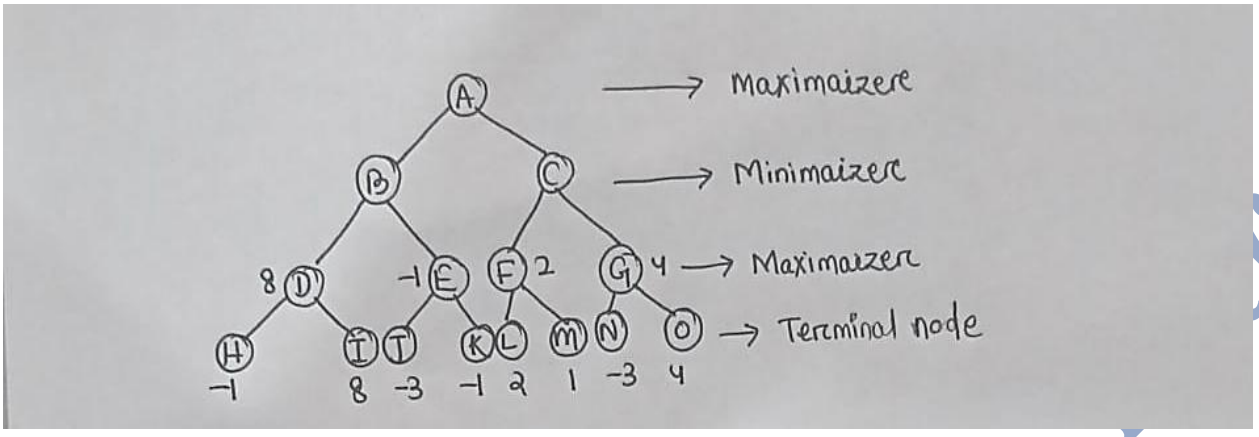
Step 2: Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

For node D max(-1,-∞) => max(-1,8)= 8

For Node E max(-3, -∞) => max(-3, -1)= -1

For Node F max(2, -∞) => max(2,1) = 2

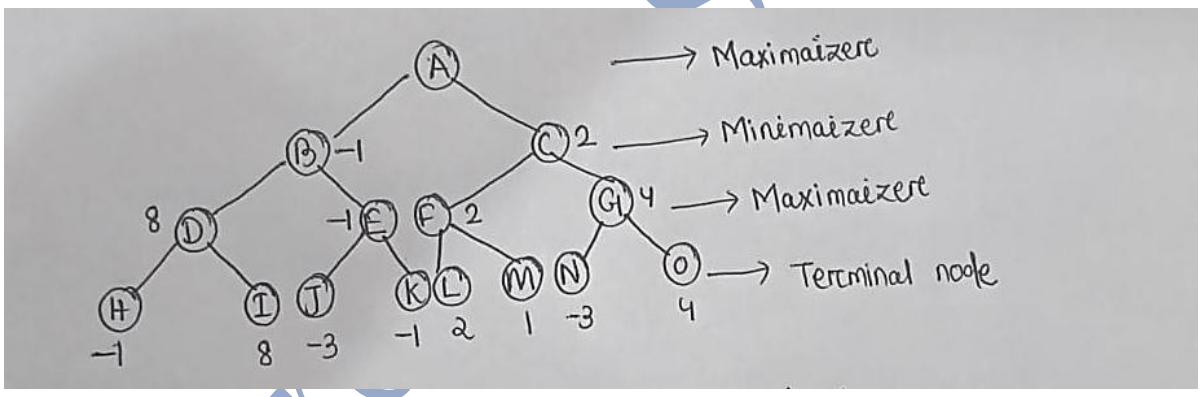
For node G max(-3, -∞) = max(-3, 4) = 4



Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

For node B min(8,-1) = -1 or for node B min(8,+∞)=>min(8,-1)=-1

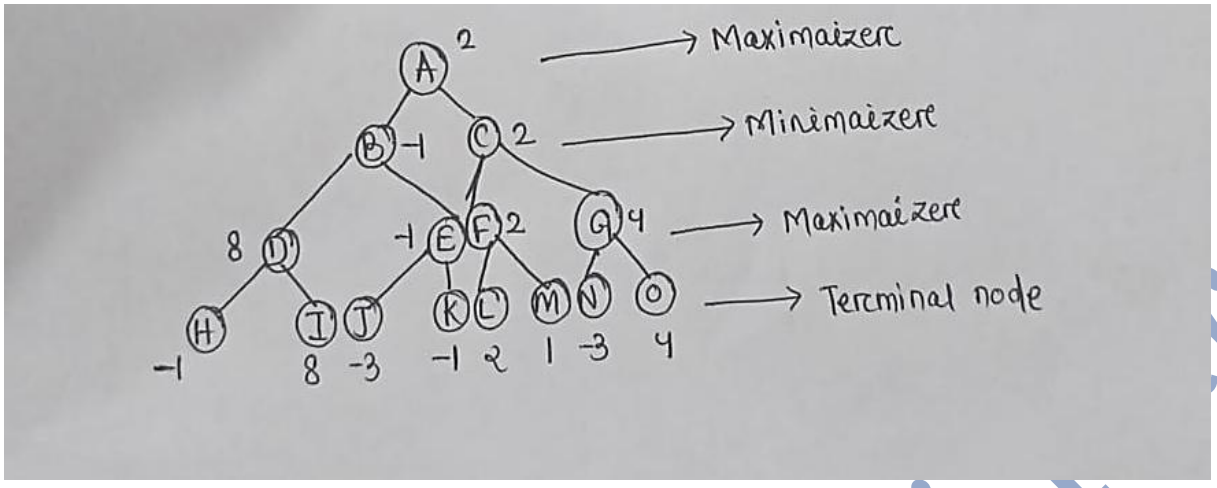
For node C min (2, 4) = 2 or for node C min(2,+∞)=>min(2,4)=-2



Step 4: Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node.

In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

For node A max(-1, 2)= 2 or node A max(-1,-∞)=>max(-1,2)=2



That was the complete workflow of the minimax two player game

Time complexity- As it performs DFS for the game-tree, so the time complexity of Mini-Max algorithm is O(bm), where b is branching factor of the game-tree, and m is the maximum depth of the tree.

Space Complexity- Space complexity of Mini-max algorithm is also similar to DFS which is O(bm).

Learning Objective:
14. Optimal decisions in multiplayer games

14. Optimal decisions in multiplayer games

- Many popular games allow more than two players. Let us examine how to extend the minimax idea to multiplayer games. This is straightforward from the technical viewpoint, but raises some interesting new conceptual issues.
- First, we need to replace the single value for each node with a vector of values. For example, in a three-player game with players A, B, and C, a vector (vA, vB, vC) is associated with each node.
- For terminal states, this vector gives the utility of the state from each player's viewpoint. (In two-player, zero-sum games, the two-element vector can be reduced to a single value because the values are always opposite.) The simplest way to implement this is to have the UTILITY function return a vector of utilities.
- Now we have to consider non terminal states. Consider the node marked X in the game tree shown in the figure below.
- In that state, player C chooses what to do. The two choices lead to terminal states with utility vectors vA =1, vB =2, vC =6 and vA =4, vB =2, vC =3 . Since 6 is bigger than 3, C should choose the first move. This means that if state X is reached, subsequent play will lead to a terminal state with utilities vA =1, vB =2, vC =6 . Hence, the backed-up value of X is this vector.
- The backed-up value of a node n is always the utility vector of the successor state with the highest value for the player choosing at n.
- Anyone who plays multiplayer games, such as Diplomacy, quickly becomes aware that much more is going on than in two-player games. Multiplayer games usually involve alliances, whether formal or informal, among the players. Alliances are made and broken as the game proceeds.
- For Example suppose A and B are in weak positions and C is in a stronger position. Then it is often optimal for both A and B to attack C rather than each other, lest C destroy each of them individually. In this way, collaboration emerges from purely selfish behavior. As soon as C weakens under the joint onslaught, the alliance loses its value, and either A or B could violate the agreement. In some cases, explicit alliances merely make concrete what would have happened anyway. In other cases, a social stigma attaches to breaking an alliance, so players must balance the immediate advantage of breaking an alliance against the long-term disadvantage of being perceived as untrustworthy.
- If the game is not zero-sum, then collaboration can also occur with just two players. Suppose, for example, that there is a terminal state with utilities vA =1000, vB =1000 and that 1000 is the highest possible utility

for each player. Then the optimal strategy is for both players to do everything possible to reach this state—that is, the players will automatically cooperate to achieve a mutually desirable goal.

`

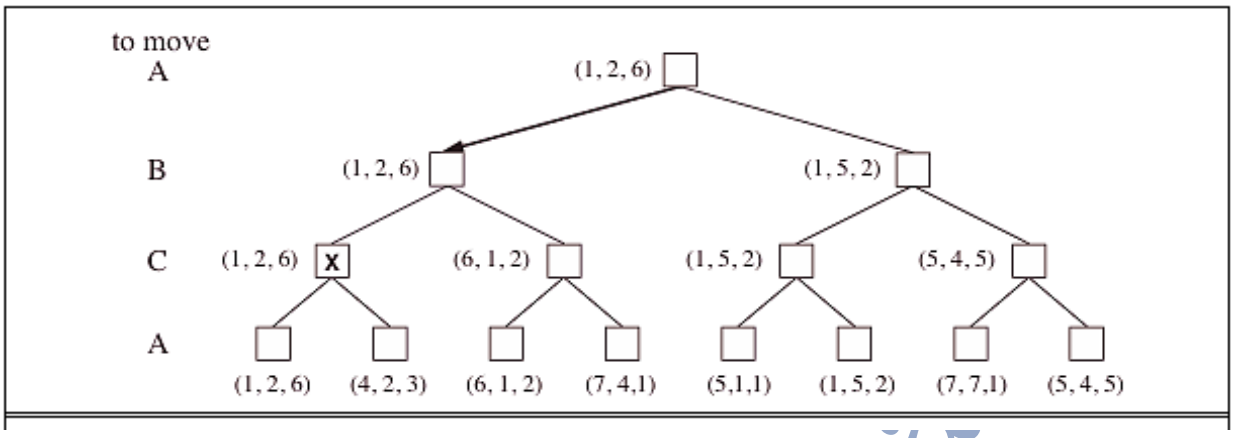


Fig:- The first three plies of a game tree with three players (A, B, C). Each node is labelled with values from the viewpoint of each player. The best move is marked at the root.

Learning Objective:
15. Alpha-Beta Pruning

15. Alpha-Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches (reducing the size of the search tree) in the game tree which need not be searched because there already exists a better move available.
- Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning. This involves two threshold parameter Alpha and beta for future expansion, so it is called alpha-beta pruning. It is also called as Alpha-Beta Algorithm.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:

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

  Beta: The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is $+\infty$.
- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is **$\alpha >= \beta$**

Key points about alpha-beta pruning:

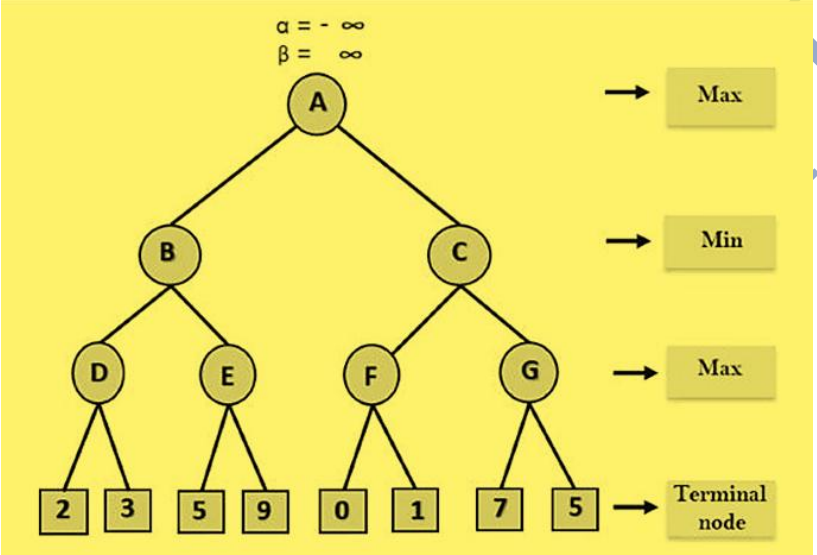
- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While back tracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.

- We will only pass the alpha, beta values to the child nodes.

Working of Alpha-Beta Pruning:

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

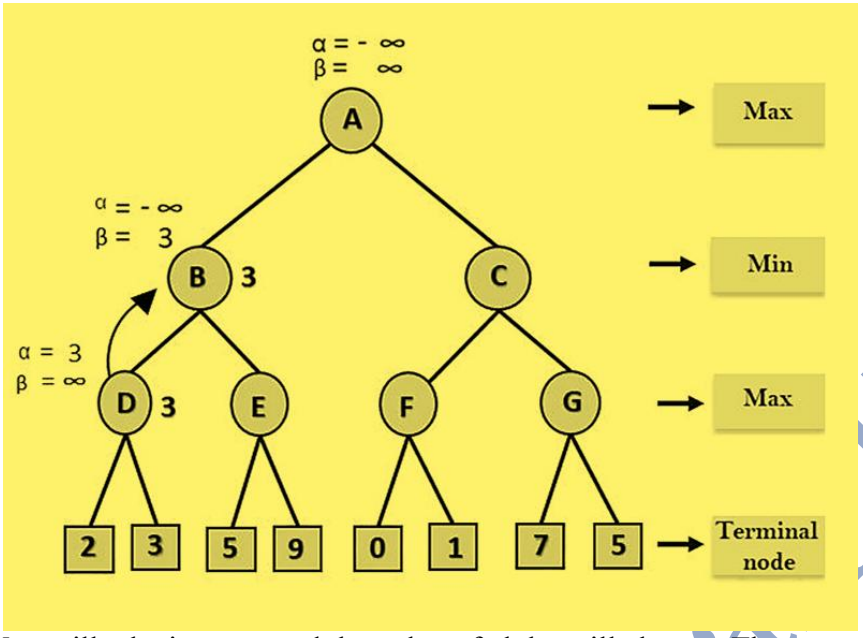
Step 1: At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



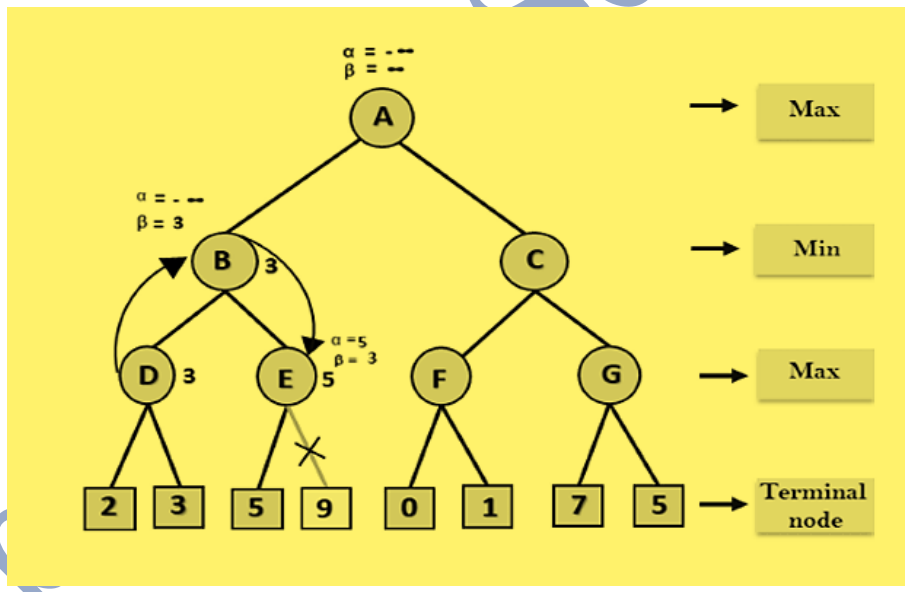
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

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

In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= ∞, and β= 3 will also be passed.



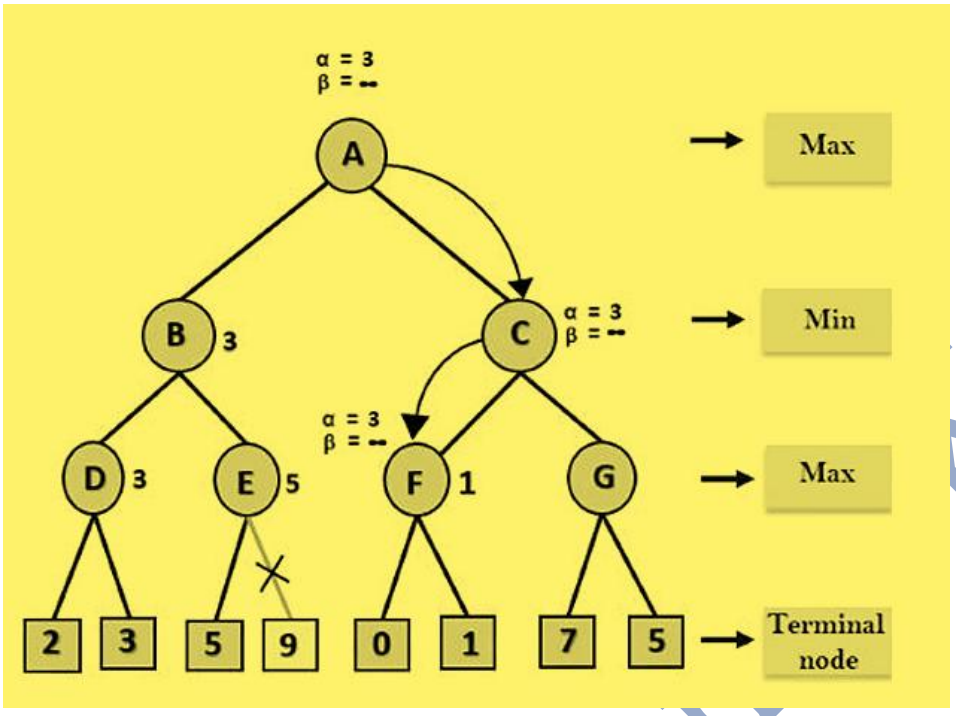
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will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



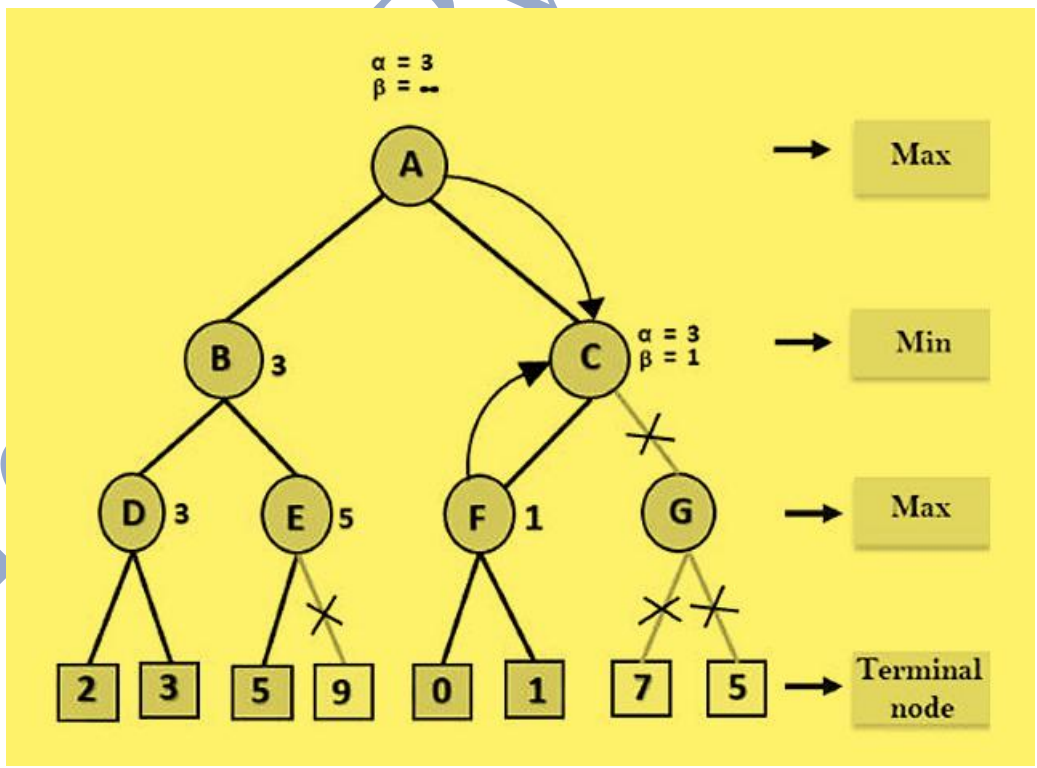
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

At node C, α=3 and β= +∞, and the same values will be passed on to node F.

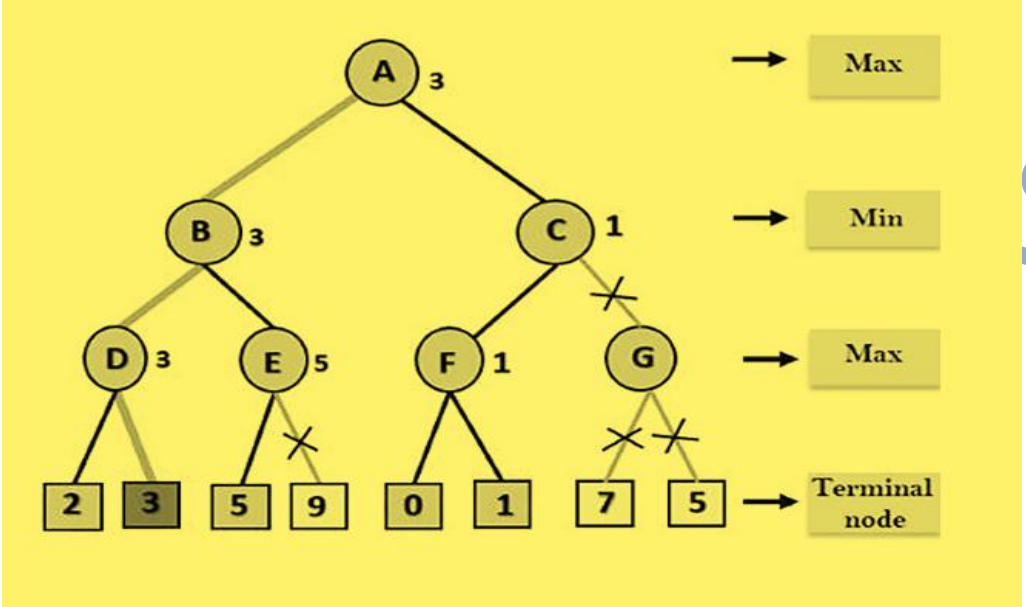
Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



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



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



Module-2 Lecture-19

Learning Objective:
16. Logical Agent

17. Knowledge based Agent

16. Logical Agent:

A logical agent in Artificial Intelligence is an intelligent agent that makes decisions based on logical reasoning. It uses knowledge representation and inference mechanisms to derive conclusions from available information. Logical agents operate using propositional logic or first-order logic (FOL) to make deductions and solve problems systematically.

17. Knowledge based Agent

- The central component of a knowledge-based agent is its knowledge base, or KB. A knowledge base is a set of sentences. Each sentence is expressed in a language called a knowledge representation language and represents some assertion about the world. Sometimes we dignify a sentence with the name axiom, when the sentence is taken as given without being derived from other sentences.
- There must be a way to add new sentences to the knowledge base and a way to query what is known. The standard names for these operations are TELL and ASK, respectively. Both operations may involve inference that is, deriving new sentences from old.
- Inference must obey the requirement that when one ASKs a question of the knowledge base, the answer should follow from what has been told to the knowledge base previously.
- The agent maintains a knowledge base, KB, which may initially contain some background knowledge.
- Each time the agent program is called, it does three things.
    - First, it TELLs the knowledge base what it perceives.
    - Second, it ASKs the knowledge base what action it should perform. In the process of answering this query, extensive reasoning may be done about the current state of the world, about the outcomes of possible action sequences, and so on.
    - Third, the agent program TELLs the knowledge base which action was chosen, and the agent executes the action.
- The details of the representation language are hidden inside three functions that implement the interface between the sensors and actuators on one side and the core representation and reasoning system on the other.
- The functions are discussed below:
    a. **MAKE-PERCEPT-SENTENCE()**: This function returns a sentence which tells the perceived information by the agent at a given time.
    b. **MAKE-ACTION-QUERY()**: This function returns a sentence which tells what action the agent must take at the current time.
    c. **MAKE-ACTION-SENTENCE()**: This function returns a sentence which tells an action is selected as well as executed.

Various levels of knowledge-based agent:

1. Knowledge level:

Knowledge level is the first level of knowledge-based agent, and in this level, we need to specify what the agent knows, and what the agent goals are. With these specifications, we can fix its behavior. For example, suppose an automated taxi agent needs to go from a station A to station B, and he knows the way from A to B, so this comes at the knowledge level.

2. Implementation level:

This is the physical representation of logic and knowledge. At the implementation level agent perform actions as per logical and knowledge level. At this level, an automated taxi agent actually implement his knowledge and logic so that he can reach to the destination.

Module-2 **Lecture-20**

Learning Objective:

18. Logic

18.1 Propositional Logic

18.1.1 Syntax of propositional logic

18.1.2 Logical Connectives

18.1.3 Truth Table

18. Logic:

In AIML logic is the formal and structured approach to reasoning that allows machines (computers, robots, or systems) to make decisions, solve problems, or draw conclusions based on a set of rules, facts, or knowledge.

Logic helps machines perform reasoning tasks like humans do by following certain principles of logic (such as true/false, and/or/not, etc.).

Types of Logic in AIML:

There are two main types of logic used in AIML, these are

1. Propositional Logic (PL)
2. First-Order Logic (FOL) or Predicate Logic

18.1 Propositional Logic (PL) :

- Propositional Logic is also known as Boolean Logic, is the simplest form of logic used in Artificial Intelligence (AI) to express facts, statements, and conditions.
- It uses propositions (statements) that can be either True (T) or False (F).

Example of Propositional Logic:

a) It is Saturday.

b) The Sun rises from West (False proposition)

c) 13+2= 67(False proposition)

d) 5 is a prime number.

Rules for Writing Propositional Logic:

- A proposition is a declarative statement that can either be True (T) or False (F) but not both.
- Propositional Logic uses five main logical connectives to connect statements.The connectives are: NOT(Negation),AND(Conjuction),OR(Disjunction),IMPLIES and BICONDITIONAL.
- Every propositional logic statement must be clear and unambiguous.
- When combining two or more propositions, always use parentheses to avoid confusion.
- When combining propositions, you must always follow the truth table to evaluate the logic.
- Avoid writing statements that contradict each other.

- The implication represents a cause-effect relationship. Always ensure the cause happens before the effect.
  Example (P → Q)
- Always write complex propositional logic in standard form:

18.1.1 Syntax of Propositional Logic:

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

a) Atomic Propositions /Atomic Sentence
b) Complex propositions/Complex Sentence

a) Atomic Proposition/Atomic Sentence:

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

Example:

5+2 is 7, it is an atomic proposition as it is a **true** fact.

"The Sun is cold" is also a proposition as it is a **false** fact.

b) Complex propositions/Complex Sentence:

Complex propositions are constructed by combining simpler or atomic propositions, using parentheses and logical connectives.

Example:

"It is raining today, and street is wet."

"Ankit is a doctor, and his clinic is in Mumbai."

18.1.2 Logical Connectives:

Logical connectives are used to connect two simpler propositions or represent a sentence logically. We can create complex propositions with the help of logical connectives.

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

1. **NOT(Negation):** A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.
   Example: It is raining.

   P=It is raining.

   ¬ P
2. **AND(Conjunction):** A sentence which has ∧ connective such as, P ∧ Q is called a conjunction.

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

   P= Rohan is intelligent,

   Q= Rohan is hardworking. → P∧ Q.

3. **OR(Disjunction):** A sentence which has ∨ connective, such as P ∨ Q is called disjunction, where P and Q are the propositions.

   Example: "Ritika is a doctor or Engineer",

   Here P= Ritika is Doctor.

   Q= Ritika is Engineer, so we can write it as P ∨ Q.

4. **IMPLIES(Implication):** A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as

   If it is raining, then the street is wet.

   Let P= It is raining,

   Q= Street is wet, so it is represented as P → Q

5. **IF AND ONLY IF(Biconditional):** A sentence such as P↔ Q is a Biconditional sentence, example: An angle is right if and only if it measures 90 degree.

   Example:

   Let P= An angle is right

   Q= An angle is measures 90 degree

   It can be represented as P ↔ Q.

Following is the summarized table for Logical Connectives:

Word	Technical Term	Symbol	Meaning	Example
NOT	NEGATION	¬	Reverses the truth value	¬A → True if A is False
AND	CONJUNCTION	∧	True if both operands are true	(A ∧ B) → True only if A and B are both True
OR	DISJUNCTION	∨	True if at least one operand is true	(A ∨ B) → True if A or B (or both) are True
IMPLIES	IMPLICATION	→	True unless the first operand is true and the second is false	(A → B) → False only if A is True and B is False
IF AND ONLY IF	BICONDITIONAL	↔	True if both operands have the same truth value	(A ↔ B) → True if A and B are both True or both False

18.1.3 Truth Table:

A **truth table** is a table used in logic and Boolean algebra to show all possible truth values of logical expressions based on their inputs. It lists all possible combinations of truth values for variables and shows the result of applying logical operators.

For Negation:

P	¬P
T	F

F	T

For Conjunction:

P	Q	P ∧ Q
T	T	T
T	F	F
F	T	F
F	F	F

For Disjunction:

P	Q	P ∨ Q
T	T	T
T	F	T
F	T	T
F	F	F

For Implication:

P	Q	P → Q
T	T	T
T	F	F
F	T	T
F	F	T

For Biconditional:

P	Q	P ↔Q
T	T	T
T	F	F
F	T	F
F	F	T

Truth table with three propositions:

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

P	Q	R	(P ∨ Q) ∧ R
TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	FALSE
FALSE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE

Module-2 **Lecture-21**

Learning Objective:

18. Logic

18.1 Propositional Logic

18.1.4 Precedence of Logical Connectives

18.1.5 Evaluation Rules

18.1.6 Logical Equivalence

18.1.7 Equivalence Laws

18.1.8 Limitations of Propositional logic

18.1.9 Translate English sentences into Propositional Logic

18.1.4 Precedence of Logical Connectives:

The precedence of logical connectives determines the order in which operations are evaluated in a logical expression, similar to operator precedence in arithmetic.

18.1.5 Evaluation Rules:

- Operators with higher precedence are evaluated first unless parentheses dictate otherwise.
- Parentheses override precedence, ensuring that the enclosed operations are computed first.

Precedence Order (Highest to Lowest)

Precedence	Operator	Name
1 (Highest)	¬	NOT (Negation)
2	∧	AND (Conjunction)
3	∨	OR (Disjunction)
4	→	IMPLICATION (If-Then)
5 (Lowest)	↔	BICONDITIONAL (If and Only If)

18.1.6 Logical Equivalence:

Logical equivalence means that two logical expressions always produce the same truth values for all possible inputs. If two statements A and B are logically equivalent, we write:

$$A \cong B$$

This means that A and B have the same truth table.

Example:

1. **Prove ¬(A ∨ B) ≅ (¬A ∧ ¬B)**

Ans:- ¬(A ∨ B) ≅ (¬A ∧ ¬B)

This states that NOT (A OR B) is logically equivalent to (NOT A AND NOT B).

A	B	A∨B	¬(A∨B)	¬A	¬B	A ∧ ¬B
T	T	T	F	F	F	F
T	F	T	F	F	T	F
F	T	T	F	T	F	F
F	F	F	T	T	T	T

Since the columns for ¬(A∨B)¬(A∨B) and ¬A∧¬B¬A∧¬B are identical, the two expressions are logically equivalent.

Tautologies:

A proposition P is a tautology if it is true under all circumstances. It means it contains the only T in the final column of its truth table.

Example: Prove that the statement (P⟶Q) ↔(~Q⟶~P) is a tautology.

P	Q	P→Q	~Q	~P	~Q⟶~P	(P→Q)↔(~Q⟶~P)
T	T	T	F	F	T	T
T	F	F	T	F	F	T
F	T	T	F	T	T	T
F	F	T	T	T	T	T

Contradiction:

A statement that is always false is known as a contradiction.

Example: Show that the statement P ∧~P is a contradiction.

P	~P	P ∧~P
T	F	F
F	T	F

18.1.7 Equivalence Laws:

Equivalence Laws or Relations are used to reduce or simplify a given well formed formula or to derive a new formula from the existing formula. These laws can be verified using the truth table approach.

Some of the important equivalence laws are given below.

Sl. No	Name of Relation	Equivalence Relations
1.	Commutative Law	A ∨ B ≅ B ∨ A A ∧ B ≅ B ∧ A
2.	Associative Law	A ∨ (B ∨ C) ≅ (A ∨ B) ∨ C A ∧ (B∧C) ≅ (A ∧ B) ∧C
3.	Double Negation Law	¬(¬A) ≅ A
4.	Distributive Laws	A ∨ (B ∧ C) ≅ (A ∨ B) ∧ (A ∨ C) A ∧ (B ∨ C) ≅ (A ∧ B) ∨ (A ∧ C)
5.	De Morgan's Laws	¬(A ∨ B) ≅¬A ∧ ¬B ¬(A ∧ B) ≅¬A ∨ ¬B
6.	Absorption Laws	A ∨ (A ∧ B) ≅ A A ∧ (A ∨ B) ≅ A A ∨ (¬A ∧ B) ≅ A ∨ B A ∧ (¬A ∨ B) ≅ A ∧ B

7.	**Idempotence Law**	$A \vee A \cong A$ $A \wedge A \cong A$
8.	**Excluded Middle Law**	$A \vee \neg A \cong T(True)$
9.	**Contradiction Law**	$A \wedge \neg A \cong F(False)$
10.	**Commonly Used Equivalence Relations**	$A \vee F \cong A$ $A \vee T \cong T$ $A \wedge T \cong A$ $A \wedge F \cong F$ $A \rightarrow B \cong \neg A \vee B$ $A \leftrightarrow B \cong (A \rightarrow B) \wedge (B \rightarrow A)$ $\cong (A \wedge B) \vee \neg A \wedge \neg B$

18.1.8 Limitations of Propositional logic:

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

Example:

a. All the girls are intelligent.

b. Some apples are sweet.

II. Propositional logic has limited expressive power.

III. In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

18.1.9 Translate English sentences into Propositional Logic

Example:

a. Let p = It is raining

b. Let q = Mary is sick

c. Let t = Bob stayed up late last night

d. Let r = Paris is the capital of France

e. Let s = John is a loud-mouth

Translating Negation

a. It isn't raining
¬p
b. It is not the case that Mary isn't sick
¬ ¬q
c. Paris is not the capital of France
¬ r
d. John is in no way a loud-mouth
¬ s
e. Bob did not stay up late last night
¬ t

Translating Conjunction

a. It is raining and Mary is sick
(p ∧ q)
b. Bob stayed up late last night and John is a loud-mouth
(t ∧ s)
c. Paris isn't the capital of France and It isn't raining
(¬r ∧ ¬p)
d. John is a loud-mouth but Mary isn't sick
(s ∧ ¬q)
e. It is not the case that it is raining and Mary is sick
translation 1: It is not the case that both it is raining and Mary is sick
¬(p ∧ q)
translation 2: Mary is sick and it is not the case that it is raining
(¬p ∧ q)

Translating Disjunction

a. It is raining or Mary is sick
(p ∨ q)
b. Paris is the capital of France and it is raining or John is a loud-mouth
((r ∧ p) ∨ s)
or (r ∧ (p ∨ s))
c. Mary is sick or Mary isn't sick
(q ∨ ¬q)
d. John is a loud-mouth or Mary is sick or it is raining
((s ∨ q) ∨ p)
or (s ∨ (q ∨ p))
e. It is not the case that Mary is sick or Bob stayed up late last night
¬(q ∨ t)

Translating Implication

a. If it is raining, then Mary is sick
(p → q)

b. It is raining, when John is a loud-mouth
$(s \rightarrow p)$
c. Mary is sick and it is raining implies that Bob stayed up late last night
$((q \wedge p) \rightarrow t)$
d. It is not the case that if it is raining then John isn't a loud-mouth
$\neg(p \rightarrow \neg s)$

Translating Equivalence or Biconditional Statement

a. It is raining if and only if Mary is sick
$(p \leftrightarrow q)$
b. If Mary is sick then it is raining, and vice versa
$((p \rightarrow q) \wedge (q \rightarrow p))$
or $(p \leftrightarrow q)$
c. It is raining is equivalent to John is a loud-mouth
$(p \leftrightarrow s)$
d. It is raining is not equivalent to John is a loud-mouth
$\neg(p \leftrightarrow s)$

Learning Objective:

19. Resolution in Propositional Logic

19. Resolution in Propositional Logic

In mathematical logic and automated theorem proving, resolution is a rule of inference leading to a refutation-complete theorem-proving technique for sentences in propositional logic and first-order logic.

The resolution rule in propositional logic is a single valid inference rule that produces a new clause implied by two clauses containing complementary literals. A literal is a propositional variable or the negation of a propositional variable. Two literals are said to be complements if one is the negation of the other.

In propositional logic, the procedure for producing a proof by resolution of proposition P with respect to a set of axioms F is in the following.

Algorithm:

1. Convert all the propositions of F to a clause form.
2. Negate P and convert the result to clause form. Add it to the set of clauses obtained in step 1.
3. Repeat until either a contradiction is found or no progress can be made.
   a) Select two clauses. Call these the parent clause.
   b) Resolve them together. The resulting clause, called resolvent, will be the disjunction of all of the literals of both of the parent clauses with the following exception:
   If there are any pairs of literals L and ¬L such that one of the parent clauses contains L and the other contains ¬L , then select one such pair and eliminate both L and ¬L from the resolvent.
   c) If the resolvent is the empty clause, then a contradiction has been found. If it is not then add it to the set of clauses available to the procedure.

Example:

Suppose we are given the axioms below and want to prove R.

Given Axioms	Converted to Clause Form	Proposition
P	P	----------1
$(P \wedge Q) \rightarrow R$	$\neg P \vee \neg Q \vee R$	----------2
$(S \vee T) \rightarrow Q$	$\neg S \vee Q$	----------3
	$\neg T \vee Q$	----------4
T	T	----------5

- First we negate R, Producing ¬R, Which is already in clause form. Then we begin selecting pair of clauses to resolve together. Although any pair of clauses can be resolved, only those pairs that contain complementary literals will produce a resolvent that is likely to lead to the goal of producing the empty

clause. Here we begin by resolving with the clause ¬R Since that is one of the clauses that must be involved in the contradiction we are trying to find.

- One way of viewing the resolution process is that it takes a set of clauses that are all assumed to be true and based on information provided by the others, it generates new clauses that represent restrictions on the way each of those original clauses can be made true.
- A contradiction occurs when a clause becomes so restricted that there is no way it can be true. This is indicated by the generation of the empty clause.
- Here in order for proposition 2 to be true. One of three things must be true: ¬P, ¬Q or R. But we are assuming that ¬R is true. Given that the only way for proposition 2 to be true is for one of two things to be true : ¬P or ¬Q. That is what the first resolvent clause says.
- But proposition 1 says that P is true , which means that ¬P can't be true, which leaves only one way for proposition 2 to be true, namely for ¬Q to be true. Proposition 4 can be true if either ¬T or Q is true. But since we now know that ¬Q must be true, the Only Way for Proposition 4 to be true is for ¬T to be true.
- But proposition 5 says that T is true . Thus there is no way for all of these clauses to be tryue in a single interpretation. This is indicated by the empty clause.

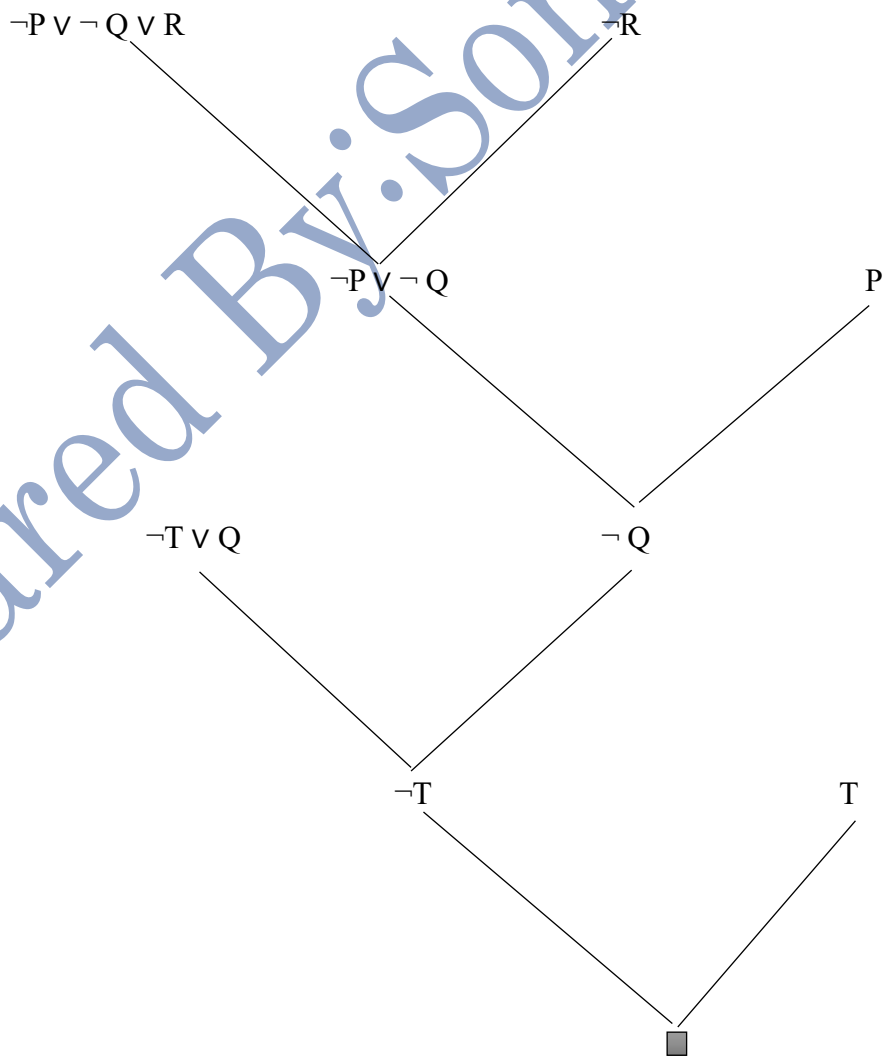


Fig:- Resolution in Propositional Logic

Module-2 Lecture-23

Learning Objective:

20. Forward Chaining & Backward Chaining

Sl. 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.

Learning Objective:

21. First-Order Logic

21.1 Syntax of First-Order logic:

21. First-Order Logic:

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic. FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as Predicate logic or First-order predicate logic. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:

  Objects: A, B, people, numbers, colors, wars, theories, squares etc.

  Relations: It can be unary relation such as: red, round, is adjacent, or n-any relation such as: the sister of, brother of, has color, comes between

  Function: Father of, best friend, third inning of, end of etc.
- As a natural language, first-order logic also has two main parts:

  a) Syntax

  b) Semantics

21.1 Syntax of First-Order Logic:

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

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

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as Predicate (term1, term2, ..., term n).
- Example:
    - Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).

      Chinky is a cat: => cat (Chinky).

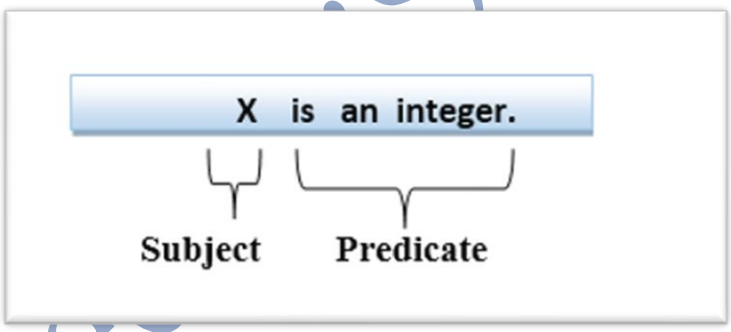
Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.
- First-order logic statements can be divided into two parts:

  Subject: Subject is the main part of the statement.

  Predicate: A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

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

  (i) Universal Quantifier, (for all, everyone, everything)

  (ii) Existential quantifier, (for some, at least one).

(i) Universal Quantifier:

- Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.
- The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

- In universal quantifier we use implication "→".

If x is a variable, then ∀x is read as:

For all x

For each x

For every x.

(ii) Existential Quantifier:

- Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.
- It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.
- In Existential quantifier we always use AND or Conjunction symbol (∧).

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

There exists a 'x.'

For some 'x.'

For at least one 'x.'

Note:

- The main connective for universal quantifier ∀ is implication →.
- The main connective for existential quantifier ∃ is and ∧.

Properties of Quantifiers:

- In universal quantifier, ∀x∀y is similar to ∀y∀x.
- In Existential quantifier, ∃x∃y is similar to ∃y∃x.
- ∃x∀y is not similar to ∀y∃x.

Example:

Some Examples of FOL using quantifier:

1. All birds fly.
In this question the predicate is "fly(bird)."
And since there are all birds who fly so it will be represented as follows.

∀x bird(x) →fly(x).

2. Every man respects his parent.
In this question, the predicate is "respect(x, y)," where x=man, and y= parent.
Since there is every man so will use ∀, and it will be represented as follows:

∀x man(x) → respects (x, parent).

3. Some boys play cricket.

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

∃x boys(x) → play(x, cricket).

4. Not all students like both Mathematics and Science.

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

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

¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

5. Only one student failed in Mathematics.

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

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

∃(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)].

Free and Bound Variables:

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

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

Example: ∀x ∃(y)[P (x, y, z)], where z is a free variable.

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

Example: ∀x [A (x) B(y)], here x and y are the bound variables.

Learning Objective:

22. Knowledge Engineering in First Order Logic

22. Knowledge Engineering in First Order Logic:

Knowledge Engineering:

The process of constructing a knowledge-base in first-order logic is called as knowledge- engineering. In knowledge- engineering, someone who investigates a particular domain, learns important concept of that domain, and generates a formal representation of the objects, is known as knowledge engineer.

The knowledge-engineering process:

An engineering term is used when we are talking about any project. Therefore, knowledge engineering over a project involves the below described steps:

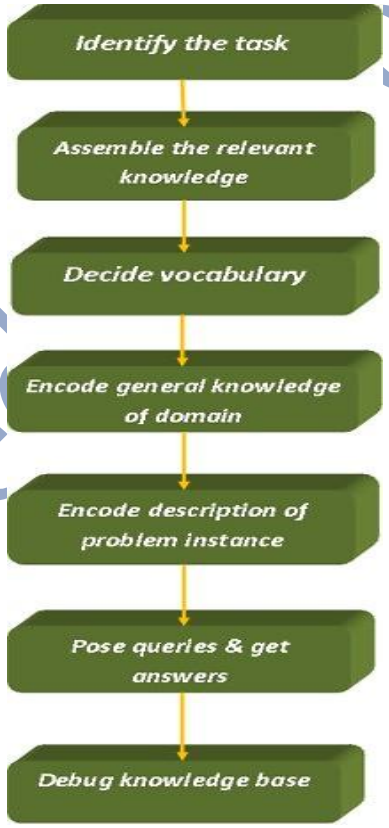


Fig: Knowledge Engineering Process

Identify the task: The knowledge engineer must delineate the range of questions that the knowledge base will support and the kinds of facts that will be available for each specific problem instance. For example, does the knowledge base need to be able to choose actions or is it required to answer questions only about the contents of the environment? Will the sensor facts include the current location? The task will identify the knowledge requirement needed to connect the problem instance with the answers.

Assemble the relevant knowledge: A knowledge engineer should be an expert in the domain. If not, he should work with the real experts to extract their knowledge. This concept is known as Knowledge Acquisition.

Here, we do not represent the knowledge formally. But to understand the scope of the knowledge base and also to understand the working of the domain.

Decide on a vocabulary of constants, predicates, and functions: Translating important domain-level concepts into logical level names. This involves many questions of knowledge engineering style. Like programming style ,this can have a significant impact on the eventual success of the project.

Here, the knowledge engineer asks questions like:

- What are the elements which should be represented as objects?
- What functions should be chosen?

After satisfying all the choices, the vocabulary is decided. It is known as the **Ontology of the domain**. Ontology determines the type of things that exists but does not determine their specific properties and interrelationships.

Encode general knowledge about the domain: In this step, the knowledge engineer writes down the axioms for all the chosen vocabulary terms. This pins down(to the extent possible) the meaning of the terms, enabling the expert to check the content. Often, this step reveals misconceptions or gaps occur between the vocabulary terms that must be fixed by returning to step3 and iterating through the process.

Encode description of the specific problem instance: We write the simple atomic sentences for the selected vocabulary terms. We encode the chosen problem instances.

Pose queries to the inference procedure and get answers: It is the testing step. We apply the inference procedure on those axioms and problem-specific facts which we want to know.

Debug the knowledge base: It is the last step of the knowledge engineering process where the knowledge engineer debugs all the errors.

Learning Objective:

23. Inference in First Order Logic

23. Inference in First Order Logic

Inference in FOL can be achieved using:

- Inference rules
- Forward chaining and Backward chaining
- Resolution
- Unification

Inference in First-Order Logic (FOL) is the process of deducing new facts or conclusions from given facts or premises. To understand FOL inference, let's first clarify some key terminologies:

1. Substitution in FOL:

Substitution is a core operation in inference, allowing us to replace variables with constants, terms, or other variables. It is essential for applying inference rules like unification and resolution.

Notation: If we write F[a/x], it means we substitute the variable x with the constant a in the formula F.

Example: If we have the formula Loves(x, Mary) and apply the substitution {John/x}, it results in Loves(John, Mary).

Substitution is particularly complex when dealing with **quantifiers** (∀ and ∃). We must be careful not to change the meaning of a formula by substituting variables bound by quantifiers.

2. Equality in FOL:

In addition to predicates and terms, FOL also includes equality (=) to specify that two terms refer to the same object.

Example:

- Brother(John) = Smith
  → This means that John's brother is the same person as Smith.
- Father(Peter) ≠ Robert
  → This means that Peter's father is not Robert.

We can also use negation to express inequality:

¬(x = y) is equivalent to x ≠ y, meaning that x and y are different objects.

Equality allows us to form stronger logical statements, ensuring that different representations of the same entity are treated as identical when reasoning in FOL.

Inference Rules for Quantifiers in First-Order Logic (FOL):

Inference rules in FOL help us derive new logical conclusions from given statements. The key inference rules involving quantifiers are:

1. Universal Instantiation (UI)
2. Existential Instantiation(EI)

1. Universal Instantiation (UI)

- Universal instantiation is also called as universal elimination. It is a valid inference rule. It can be applied multiple times to add new sentences.
- The new KB is logically equivalent to the previous KB. As per UI, we can infer any sentence obtained by substituting a ground term for the variable.
- The UI rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from ∀ x P(x) for any object in the universe of discourse.
- It can be represented as

$$\frac{\forall x\, P(x)}{P(c)}.$$

Example:1

IF "Every person like ice-cream"=> ∀x P(x) so we can infer that

"John likes ice-cream" => P(c)

Example:2

Let's take a famous example,

"All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL:

∀x king(x) ∧ greedy (x) → Evil (x),

So from this information, we can infer any of the following statements using Universal Instantiation:

King(John) ∧ Greedy (John) → Evil (John),

King(Richard) ∧ Greedy (Richard) → Evil (Richard),

King(Father(John)) ∧ Greedy (Father(John)) → Evil (Father(John)),

2. Existential Instantiation (EI):

- Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.
- It can be applied only once to replace the existential sentence.
- The new KB is not logically equivalent to old KB, but it will be satisfiable if old KB was satisfiable.

- This rule states that one can infer P(c) from the formula given in the form of ∃x P(x) for a new constant symbol c.
- The restriction with this rule is that c used in the rule must be a new term for which P(c) is true.
- It can be represented as

$$\frac{\exists x\, P(x)}{\therefore\ P(c)}$$

Example:

- From the given sentence: ∃x Crown(x) ∧ OnHead(x, John),
- So we can infer: Crown(K) ∧ OnHead(K, John), as long as K does not appear in the knowledge base.
- The above used K is a constant symbol, which is called Skolem constant.
- The Existential instantiation is a special case of Skolemization process.

Generalized Modus Ponens Rule:

- For the inference process in FOL, we have a single inference rule which is called Generalized Modus Ponens. It is lifted version of Modus ponens.
- Generalized Modus Ponens can be summarized as, " P implies Q and P is asserted to be true, therefore Q must be True."
- According to Modus Ponens, for atomic sentences pi, pi', q. Where there is a substitution θ such that SUBST

(θ, pi',) = SUBST (θ, pi), it can be represented as:

$$\frac{p1', p2', \ldots, pn', (p1 \wedge p2 \wedge \ldots \wedge pn \Rightarrow q)}{SUBST(\theta, q)}$$

Example:

We will use this rule for Kings are evil, so we will find some x such that x is king, and x is greedy so we can infer that x is evil.

Here let say, p1' is king(John) p1 is king(x)

p2' is Greedy(y) p2 is Greedy(x)

θ is {x/John, y/John} q is evil(x)

SUBST(θ,q) is evil(John)

Learning Objective:

24. Propositional vs. Predicate Logic

24. Propositional vs. Predicate Logic

Feature	Propositional Logic	Predicate Logic
Definition	Deals with declarative statements (propositions) that have a definite truth value (true/false).	Uses variables, objects, and relations to express logical statements with a specified domain.
Complexity	Simple, uses Boolean logic.	More expressive, extends propositional logic with predicates and quantification.
Truth Value	Each proposition has a fixed truth value (true or false).	The truth value of a predicate depends on the values of variables.
Scope Analysis	Not performed.	Uses quantifiers to analyze scope (∀ for all, ∃ for existence, ∃! for exactly one).
Logical Operators	Uses standard logical connectives: Negation (¬), Conjunction (∧), Disjunction (∨), Exclusive OR (⊕), Implication (⇒), Bi-Conditional (⇔).	Extends propositional logic by adding quantifiers.
Representation	Generalized representation of logical statements.	More specialized and expressive.
Handling of Entities	Cannot handle sets of entities.	Deals with sets of entities using quantifiers.

Learning Objective:
25. Unification and Lifting

25. Unification and Lifting:

- Lifted inference rules require finding substitutions that make different logical expressions look identical. This process is called unification.
- Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
- It takes two literals as input and makes them identical using substitution.
- Let Ψ1 and Ψ2 be two atomic sentences and σ be a unifier such that, $\Psi1\sigma = \Psi2\sigma$, then it can be expressed as UNIFY (Ψ1, Ψ2).

  Example: Find the MGU for Unify {King(x), King (John)}

  Let Ψ1 = King(x), Ψ2 = King (John)

  Substitution θ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical.

- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).
- Unification is a key component of all first-order inference algorithms.
- It returns fail if the expressions do not match with each other.
- The substitution variables are called Most General Unifier or MGU.

  E.g. Let's say there are two different expressions, P(x, y), and P(a, f(z)).

  In this example, we need to make both above statements identical to each other. For this, we will perform the substitution.

  P(x, y)..... (i)

  P(a, f(z))..... (ii)

  Substitute x with a, and y with f(z) in the first expression, and it will be represented as a/x and f(z)/y. With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: [a/x, f(z)/y].

Conditions for Unification:

Following are some basic conditions for unification:

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.

- Number of Arguments in both expressions must be identical.
- Unification will fail if there are two similar variables present in the same expression.
- For each pair of the following atomic sentences find the most general unifier (If exist).

Examples:

1.UNIFY(knows(Richard, x), knows(Richard, John))

Here, Ψ1 = knows(Richard, x), and Ψ2 = knows(Richard, John)

S0 => { knows(Richard, x); knows(Richard, John)}

SUBST θ= {John/x}

S1 => { knows(Richard, John); knows(Richard, John)}, Successfully Unified.

Unifier: {John/x}.

2. Find the MGU of {p(f(a), g(Y)) and p(X, X)}

Sol: S0 => Here, Ψ1 = p(f(a), g(Y)), and Ψ2 = p(X, X)

SUBST θ= {f(a) / X}

S1 => Ψ1 = p(f(a), g(Y)), and Ψ2 = p(f(a), f(a))

SUBST θ= {f(a) / g(y)}, Unification failed.

Unification is not possible for these expressions.

3. Find the MGU of UNIFY(prime (11), prime(y))

Here, Ψ1 = {prime(11) , and Ψ2 = prime(y)}

S0 => {prime(11) , prime(y)}

SUBST θ= {11/y}

S1 => {prime(11) , prime(11)}, Successfully unified.

Unifier: {11/y}.

4. Find the MGU of {p (X, X), and p (Z, f(Z))}

Here, Ψ1 = {p (X, X), and Ψ2 = p (Z, f(Z))

S0 => {p (X, X), p (Z, f(Z))}

SUBST θ= {X/Z}

S1 => {p (Z, Z), p (Z, f(Z))}

SUBST θ= {f(Z) / Z}, Unification Failed.

Hence, unification is not possible for these expressions.

Module-2 Lecture-29

Learning Objective:
26. Forward Chaining

26. Forward Chaining

Horn Clause and Definite clause:

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

Definite clause: A clause which is a disjunction of literals with exactly one positive literal is known as a definite clause or strict horn clause.

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

Example: (¬ p V ¬ q V k). It has only one positive literal k.

It is equivalent to p ∧ q → k.

Consider the following example which we will solve in both approaches:

Forward Chaining:

Example:

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

Prove that "Robert is criminal."

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

Facts Conversion into FOL:

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

  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ------(1)

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

  Owns(A, T1) ----------(2)

  Missile(T1) ----------(3)

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

  ∀p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ----------(4)

- Missiles are weapons.

  Missile(p) → Weapons (p) ----------(5)

- Enemy of America is known as hostile.

  Enemy(p, America) →Hostile(p) ----------(6)

- Country A is an enemy of America.

  Enemy (A, America) ----------(7)

- Robert is American

  American(Robert). ----------(8)

Forward chaining proof:

Step-1: In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1). All these facts will be represented as below.

Step-2:

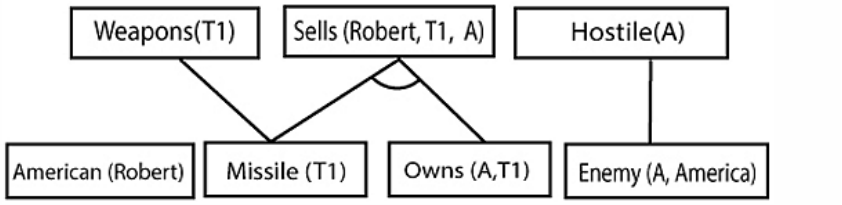
At the second step, we will see those facts which infer from available facts and with satisfied premises.

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

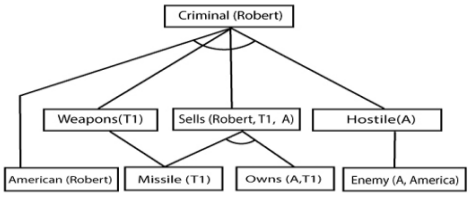
Rule-(2) and (3) are already added.

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

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



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



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

Learning Objective:
27. Backward Chaining

27. Backward Chaining:

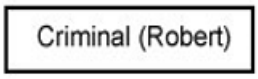
In backward-chaining, we will use the same above example, and will rewrite all the rules.

American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ----------(1)
Owns(A, T1) ----------(2)
Missile(T1) ----------(3)
∀p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) --------(4)
Missile(p) → Weapons (p) ----------(5)
Enemy(p, America) →Hostile(p) ---------(6)
Enemy (A, America) ----------(7)
American(Robert). ----------(8)

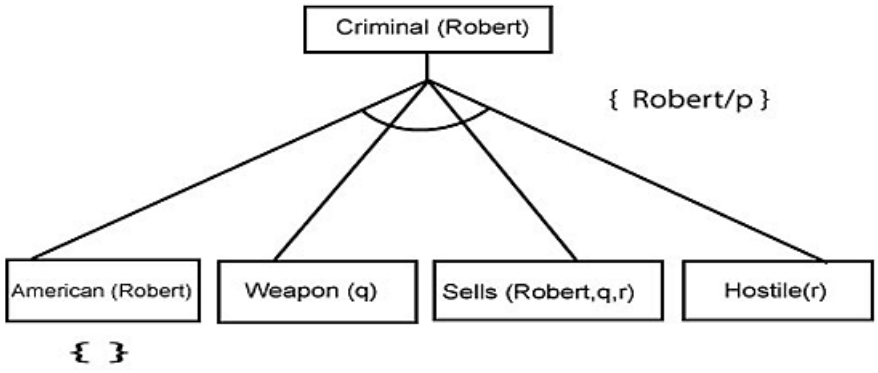
Backward-Chaining Proof:

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

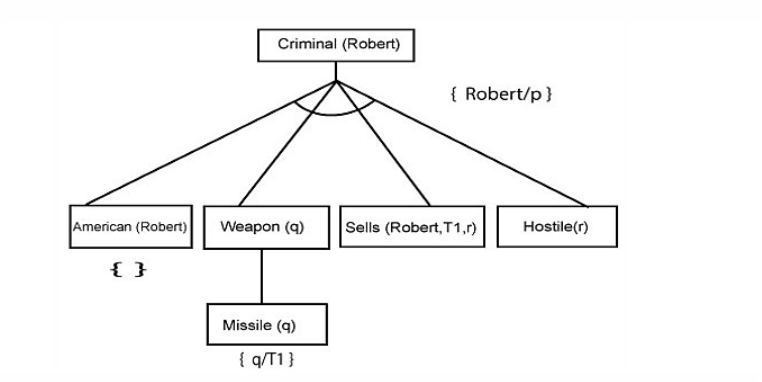
Step-1: At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.



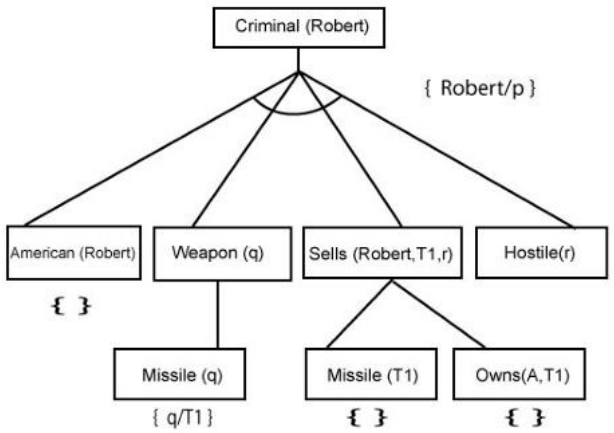
Step-2: At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert. Here we can see American (Robert) is a fact, so it is proved here.



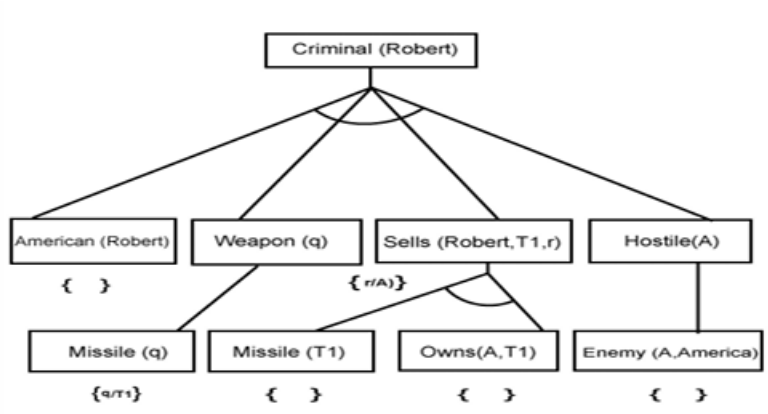
Step-3: At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



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



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



Learning Objective:
28. Resolution in FOL

28. Resolution in FOL

Resolution method in FOPL is an uplifted version of the propositional resolution method.

In FOPL, the process to apply the resolution method is as follows:

- Conversion of facts into first order logic.
- Convert the given axiom into CNF, i.e., a conjunction of clauses. Each clause should be dis-junction of literals.
- Apply negation on the goal given.
- Use literals which are required and prove it.
- Draw resolution graph/tree. Unlike propositional logic, FOPL literals are complementary if one unifies with the negation of another literal.

For Example:

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

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

{Bird(F(x)) V ¬Kills(G(x),x)}

The rule applied on the following example is called Binary Resolution because it only resolves exactly two literals.

Conjunctive Normal Form

There are following steps used to convert into CNF:

- **Eliminate all implication (→) and biconditional.**
  (P ⇔ Q) ==> (P → Q) ∧ (Q → P)
  P → Q => ¬P ∨ Q
- **Move negation (¬)inwards**
  De-Morgan's Law.

  ¬ (P ∨ Q) = ¬P ∧ ¬Q
  ¬(P ∧ Q)= ¬ P ∨ ¬Q
- **Double negation elimination**

  ¬ (¬P)=P

¬∀x: A becomes ∃x: ¬A and,

¬∃x: A becomes ∀x: ¬A

It means that the universal quantifier becomes existential quantifier and vice-versa.

- Rename variables or standardize variables.
- Eliminate existential instantiation quantifier by elimination.

In this step, we will eliminate existential quantifier ∃, and this process is known as Skolemization.

- Drop Universal quantifiers.
- Use distributive laws(V over ^)
- Eliminate AND (^)/conjunction symbols separating the expression in clauses.
- A ^ B splits the entire clause in to two separate clauses i.e. A and B .
- (A∨ B) ∧ c splits the entire clause into two separate clauses A∨ B and C.
- (A∧B) ∨ C splits the clause into two clauses i.e. A∨C and B∨C. [(A∧B) ∨ C =(A∨C)^(A∨B)]

Example:

a) John likes all kind of food.
b) Apple and vegetable are food.
c) Anything anyone eats and not killed is food.
d) Anil eats peanuts and still alive.
e) Harry eats everything that Anil eats.

Prove by resolution that:

f) John likes peanuts.

Step-1: Conversion of Facts into FOL

In the first step we will convert all the given statements into its first order logic.

a. ∀x: food(x) → likes(John, x)
b. food(Apple) ∧ food(vegetables)
c. ∀x ∀y: eats(x, y) ∧ ¬ killed(x) → food(y)
d. eats (Anil, Peanuts) ∧ alive(Anil).
e. ∀x : eats(Anil, x) → eats(Harry, x)
f. ∀x: ¬ killed(x) → alive(x) } **added predicates.**
g. ∀x: alive(x) →¬ killed(x) }
h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.

- **Eliminate all implication (→) and rewrite.**

a. ∀ x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀ x ∀ y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀ x ¬ eats(Anil, x) V eats(Harry, x)

f. ∀ x¬ [¬ killed(x)] V alive(x)

g. ∀ x ¬ alive(x) V ¬ killed(x)

h. likes(John, Peanuts).

- **Move negation (¬) inwards and rewrite**

a. ∀ x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀ x ∀ y ¬ eats(x, y) V killed(x) V food(y)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀ x ¬ eats(Anil, x) V eats(Harry, x)

f. ∀ x ¬killed(x)] V alive(x)

g. ∀ x ¬ alive(x) V ¬ killed(x)

h. likes(John, Peanuts).

- **Rename variables or standardize variables**

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀w¬ eats(Anil, w) V eats(Harry, w)

f. ∀g ¬killed(g)] V alive(g)

g. ∀k ¬ alive(k) V ¬ killed(k)

h. likes(John, Peanuts).

- **Eliminate existential instantiation quantifier by elimination**

In this step, we will eliminate existential quantifier and this process is known as Skolemization. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.

- **Drop Universal quantifiers.**

In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.

a. ¬ food(x) V likes(John, x)

b. food(Apple)

c. food(vegetables)

d. ¬ eats(y, z) V killed(y) V food(z)

e. eats (Anil, Peanuts)

f. alive(Anil)

g. ¬ eats(Anil, w) V eats(Harry, w)

h. killed(g) V alive(g)

i. ¬ alive(k) V ¬ killed(k)

j. likes(John, Peanuts).

Note: Statements "food(Apple) Λ food(vegetables)" and "eats (Anil, Peanuts) Λ alive(Anil)" can be written in two separate statements.

- **Distribute conjunction Λ over disjunction ¬**

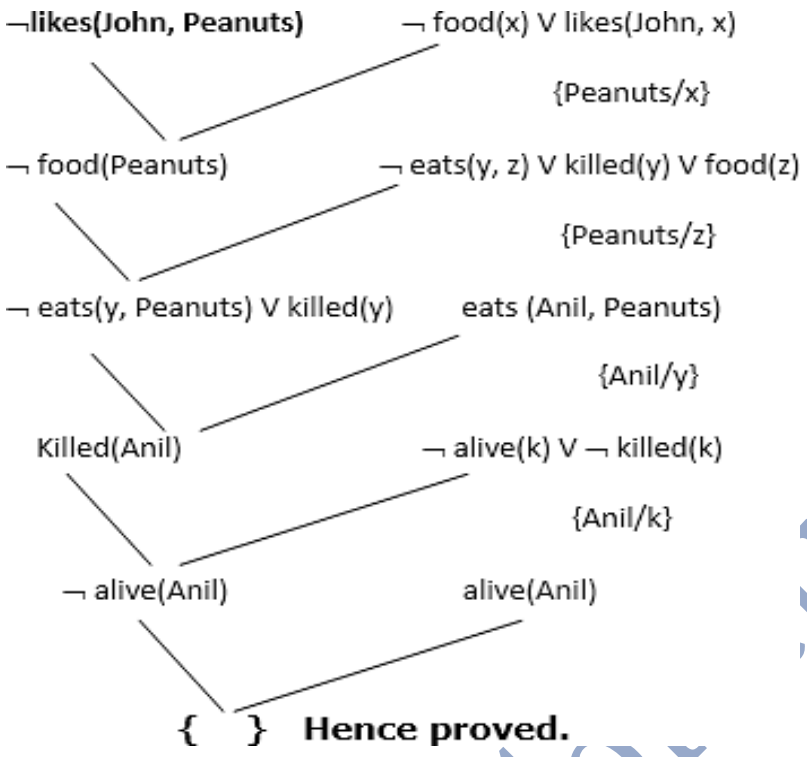
This step will not make any change in this problem.

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)

Step-4: Draw Resolution graph:

Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

Explanation of Resolution graph:

- In the first step of resolution graph, ¬likes(John, Peanuts) , and likes(John, x) get resolved(canceled) by substitution of {Peanuts/x}, and we are left with ¬ food(Peanuts)
- In the second step of the resolution graph, ¬ food(Peanuts) , and food(z) get resolved (canceled) by substitution of { Peanuts/z}, and we are left with ¬ eats(y, Peanuts) V killed(y) .
- In the third step of the resolution graph, ¬ eats(y, Peanuts) and eats (Anil, Peanuts) get resolved by substitution {Anil/y}, and we are left with Killed(Anil) .
- In the fourth step of the resolution graph, Killed(Anil) and ¬ killed(k) get resolve by substitution {Anil/k}, and we are left with ¬ alive(Anil).
- In the last step of the resolution graph ¬ alive(Anil) and alive(Anil) get resolved.

Module-3 **Lecture-32**

Learning Objective:

29. Uncertainty

29.1Causes of Uncertainty

29.2 Acting under Uncertainty

29.3 Handling Uncertainty

29. Uncertainty

- Uncertainty in Artificial Intelligence (AI) refers to the inability of models to make fully confident predictions due to incomplete, ambiguous, or noisy data. AI systems must account for uncertainty to make accurate and reliable decisions, especially in dynamic environments where information is inconsistent or evolving.
- Suppose A and B are two statements, If we implement if-then rule to these statements, we might write A→B, which means if A is true then B is true, or if A is false then B is false, if A is true then B is false, if A is false then B is true. But consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.
- So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

29.1 Causes of Uncertainty/Reasons for uncertainty

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

- Missing data, unavailable or noisy data.
- Incomplete environment details.
- Data might be present but unreliable or ambiguous.
- The representation of data may be imprecise or inconsistent.
- Data may be based on defaults, and defaults may exceptions.
- Information occurred from unreliable sources.
- Experimental Errors.
- Equipment fault.
- Temperature variation.
- Climate change etc.

29.2 Acting under Uncertainty

- The presence of uncertainty changes radically the way in which an agent makes decisions.
- Agents must still act even if world not certain. If can only act with certainty, most of the time the agent will not act.
- To make such choices, an agent must first have preferences between the different possible outcomes of the various plans, utility theory can be used to represent and reason with preferences.

- Major problem with logical-agent approaches: Agents almost never have access to the whole truth about their environments.
- In that case, an agent must reason under uncertainty
- Uncertainty also arises because of an agent's incomplete or incorrect understanding of its environment.

29.3 Handling Uncertainty

In Artificial Intelligence and Machine Learning, systems often operate in environments that are unpredictable, incomplete, or noisy. To perform effectively, they must be capable of handling uncertainty.

This involves reasoning, making decisions, and taking actions even when some information is missing or ambiguous.

Several methods have been developed to address different types of uncertainty:

1. Fuzzy logic
2. Probabilistic reasoning (Probability theory)
3. Markov models
4. Dempster-shafer theory etc.

Learning Objective:

30. Probability

30.1 Basic Probability Notations

30. Probability

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

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

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

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

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

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

Example: Think about a dice. When a dice is rolled there are six possible outcomes: 1, 2, 3, 4, 5 and 6. To find the probability of the event of rolling a 4, find the number of possible ways of rolling a 4 and divide it by the total number of possible outcomes.

There is one way of rolling a 4 and there are six possible outcomes, so the probability of rolling a 4 on a dice is 1/6.

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

P(¬A) + P(A) = 1 Or P(¬A) =1-P(A)

Probability of events not happening

Events that cannot happen at the same time are called **Mutually Exclusive Events**. For example, a football team can win, lose or draw but these things cannot happen at the same time - they are mutually exclusive. Since it is certain that one of these outcomes will happen, their probabilities must add up to 1.

Example

A bag contains 12 counters of different colours: 5 red, 4 white and 3 black. Find the probability of not selecting a red counter.

The probability of selecting a red counter is 5/12, so the probability of not selecting a red counter is 1−5/12 which is 12/12−5/12=7/12=0.58

Event: Each possible outcome of a variable is called an event.

Sample space: The collection of all possible events is called sample space.

Random variables: Random variables are used to represent the events and objects in the real world.

Prior probability: The prior probability of an event is probability computed before observing new information.

Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

30.1 Basic Probability Notations

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.

Propositions: Degrees of belief are always applied to propositions--assertions that such-and-such is the case.

Random Variable: The basic element of the language is the random1 variable, which can be thought of as referring to a "part" of the world whose "status" is initially unknown.

Domain: Each random variable has a domain of values that it can take on.

Atomic Events: The notion of an atomic event is useful in understanding the foundations of probability theory. An atomic event is a complete specification of the state of the world about which the agent is uncertain. It can be thought of as an assignment of particular values to all the variables of which the world is composed.

P(A) - probability that event A occurs
P(A') - probability that event A will not occur (A' is the complement of A)
P(A∪ B) - probability that A will occur or B will occur or both (Union of A and B)
P(A ∩B) - probability that A and B will occur simultaneously (Joint probability of A and B)
P(A | B) - probability of A, given that B is known to have occurred. (Conditional probability)

Conditional Probability/Posterior Probability:

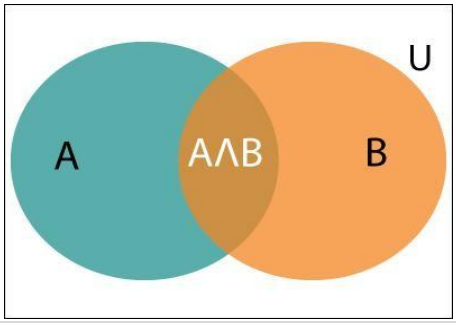
Conditional probability is a probability of occurring an event when another event has already happened.
Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:

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

Where P(A∧B)= Joint probability of A and B
P(B)= Marginal probability of B/Probability of event B.
If the probability of A is given and we need to find the probability of B, then it will be given as:

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

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of P(A∧B) by P(B).



Example:

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

Solution:

Let, A is an event that a student likes Mathematics

B is an event that a student likes English.

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

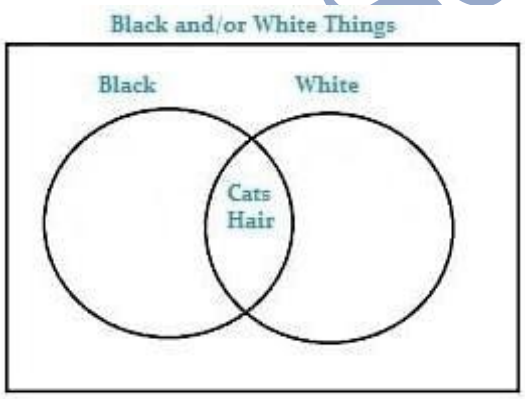
Hence, 57% are the students who like Mathematics.

Joint Probability

Joint probability is the probability of two events happening together. The two events are usually designated event A and event B. In probability terminology, it can be written as:

P(A and B) or P(A ∩ B) or P(A^B)

Joint probability can also be described as the probability of the intersection of two (or more) events. The intersection can be represented by a Venn diagram:



(A Venn diagram intersection shows the intersection of events A and B happening together.)

Example: The probability that a card is [a five and black] = p(five and black) = 2/52 = 1/26. (There are two black fives in a deck of 52 cards, the five of spades and the five of clubs).

Example:

Classification of Bank Employees				
Gender	Rank			Total
	R_1	R_2	R_3	
Male	30	80	90	200
Female	20	40	40	100
Total	50	120	130	300

Find out the probability of male person which have the rank R2.

P(M AND R2)= 80/300=0.26

Module-3 **Lecture-34**

Learning Objective:

31. Axioms of probability

31. Axioms of probability

There are three axioms of probability that make the foundation of probability theory-

Axiom 1: Probability of Event

The first one is t hat the probability of an event is always between 0 and 1. 1 indicates definite action of any of the outcomes of an event and 0 indicates no outcome of the event is possible.

Axiom 2: Probability of Sample Space

For sample space, the probability of the entire sample space is 1.

Axiom 3: Mutually Exclusive Events

And the third one is- the probability of the event containing any possible outcome of two mutually disjoint is the summation of their individual probability.

1. Probability of Event:

The first axiom of probability is that the probability of any event is between 0 and 1.

$$\text{For any event E, } 0 \leq P(E) \leq 1$$

As we know the formula of probability is that we divide the total number of outcomes in the event by the total number of outcomes in sample space.

$$P(event) = \frac{count\ of\ outcomes\ in\ Event}{count\ of\ outcomes\ in\ Sample\ Space}$$

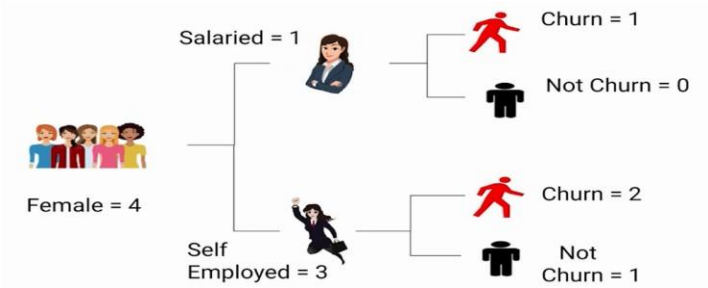
And the event is a subset of sample space, so the event cannot have more outcome than the sample space. Clearly, this value is going to be between 0 and 1 since the denominator is always greater than the numerator.

2. Probability of Sample Space:

The second axiom is that the probability for the entire sample space equals 1.

$$\text{For Sample Space, } P(S) = 1$$

Let's take an example from the dataset. Suppose we need to find out the probability of churning for the female customers by their occupation type.



In our data-set, we have 4 female customers, one of them is Salaried and three of them are self-employed. The salaried female is going to churn. Since we have only one salaried female who is going to churn, the number of salaried female customers who are not going to churn is 0. Amongst 3 self-employed female customers, two are going to churn and we can see that one self-employed female is not going to churn.

This is the complete dataset:

	gender	age	occupation	churn
0	Male	young	salaried	0
1	Male	young	self_employed	0
2	Male	old	self_employed	0
3	Male	young	self_employed	0
4	Female	young	salaried	1
5	Male	old	salaried	0
6	Female	young	self_employed	1
7	Male	young	self_employed	0
8	Male	young	salaried	1
9	Male	young	salaried	0
10	Male	young	self_employed	1
11	Female	young	self_employed	1
12	Male	young	retired	0
13	Female	young	self_employed	0
14	Male	old	self_employed	0

So the probability of the churning status of female customer by profession, in the sample space of the problem we actually have:

Salaried Churn, Salaried Not churn, Self-employed Churn, Self-employed Not churn And as we discussed their distribution earlier, in this sample space of female customer:

Salaried Churn = 1

Salaried Not churn = 0

Self-employed Churn = 2

Self-employed Not churn = 1

If you were to find out the probability that a person who is a female is salaried and is churning it will be equal to:

$$P(\text{Salaried Churn}) = \frac{1}{4} \rightarrow 0.25$$

Similarly, the probability of Salaried Not churn is:

$$P(\text{Salaried Not Churn}) = 0 \rightarrow 0$$

Then we have Self-employed Churn:

$$P(\text{Self Employed Churn}) = \frac{1}{4} \rightarrow 0.5$$

And finally Self-employed Not Churn:

$$P(\text{Self Employed Not Churn}) = \frac{1}{4} \rightarrow 0.25$$

And if we sum all of them up we get 1:

$$P(\text{Sample Space}) = 0.25 + 0 + 0.5 + .25 = 1$$

So essentially saying that this is our entire sample space and the total probability that we get here is equals to 1. This brings us to axiom 3 which is related to mutually exclusive events.

3. Mutually Exclusive Event:

$$P(A \cup B) = P(A) + P(B) \text{ for mutually exclusive events}$$

If you remember the union formula you will recall that the intersection term is not here, which means there is nothing common between A and B. Let us understand these particular type of events which is called Mutually Exclusive Events.

These Mutually exclusive events mean that such events cannot occur together or in other words, they don't have common values or we can say their intersection is zero/null. We can also represent such events as follows:

$$P(A \cap B) = 0$$

This means that the intersection is zero or they do not have any common value. For example, if the

Event A: is getting a number greater than 4 after rolling a die, the possible outcomes would be 5 and 6.

Event B: is getting a number less than 3 on rolling a die. Here the possible outcomes would be 1 and 2.

Clearly, both these events cannot have any common outcome. An interesting thing to note here is that events A and B are not complemented of each other but yet they're mutually exclusive.

Learning Objective:

32. Joint Probability Distribution

32.1 Inference using Full Joint Distributions

32. Joint Probability Distribution

- A joint probability distribution simply describes the probability that a given individual takes on two specific values for the variables.
- The word "joint" comes from the fact that we're interested in the probability of two things happening at once.
- For example,

	Baseball	Basketball	Football	Total
Male	13	15	20	48
Female	23	16	13	52
Total	36	31	33	100

- The above two-way table shows the results of a survey that asked 100 people which sport they liked best: baseball, basketball, or football. There are two variables: Sports and Gender.
- Out of the 100 total individuals there were 13 who were male and chose baseball as their favourite sport.
- Thus, we would say the joint probability that a given individual is male and chooses baseball as their favourite sport is 13/100 = **0.13** or **13%**.

Calculate the entire joint probability distribution:

P(Gender = Male, Sport = Baseball) = 13/100 = **0.13**
P(Gender = Male, Sport = Basketball) = 15/100 = **0.15**
P(Gender = Male, Sport = Football) = 20/100 = **0.20**
P(Gender = Female, Sport = Baseball) = 23/100 = **0.23**
P(Gender = Female, Sport = Basketball) = 16/100 = **0.16**
P(Gender = Female, Sport = Football) = 13/100 = **0.13**

NOTE: Notice that the sum of the probabilities is equal to **1**, or **100%**

32.1 Inference using Full Joint Distributions

Probability of all possible worlds can be described using a table called a full joint probability distribution – the elements are indexed by values of random variables.

	toothache		*¬toothache*	
	catch	*¬catch*	*catch*	*¬catch*
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

Fig. : A Full joint distribution for the Toothache, Cavity, Catch world

- **Probabilistic inference:** The computation of posterior probabilities for query propositions given observed evidence.
- The full joint probability distribution specifies the probability of each complete assignment of values to random variables. It is usually too large to create or use in its explicit form, but when it is available it can be used to answer queries simply by adding up entries for the possible worlds corresponding to the query propositions.
- **Marginalization / summing out/Marginal Probability:** The process of extract the distribution over some subset of variables or a single variable (to get the marginal probability), by summing up the probabilities for each possible value of the other variables, thereby taking them out of the equation.
- Find the marginal probability of cavity is

P(Cavity)=0.108+0.012+0.072+0.00 8 = 0.2

Computing probability of a cavity, given evidence of a toothache is as follow:

P(Cavity|Toothache) = P(Cavity^ Toothache)/P (Toothache) =0.108+0.012/0 0.108+0.012+0.016+0.064=0.6

Just to check also compute the probability that there is no cavity given toothache is as follow:

P(~Cavity|Toothache)=P(-Cavity^ Toothache)/p (toothache)=0.016+0.064/0.108+ 0.012+ 0.016 +0.064=0.4

- The two value sum is 1.0
- Notice that in these two calculations the term 1/p(toothache) remains constant, no matter which value of cavity we calculate. So it can be viewed as a normalization constant for the distribution P(Cavity| Toothache),ensuring that it adds up to 1.
- Suppose X(cavity) is a single variable. Let **E** (Toothache) be the list of evidence variables, **e** be the list of observed values for them, **Y** (Catch) be the remaining unobserved variables. The query **P**(X|e) can be evaluated as:

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y})$$

$$\begin{aligned}\mathbf{P}(Cavity \mid toothache) &= \alpha \mathbf{P}(Cavity, toothache) \\ &= \alpha [\mathbf{P}(Cavity, toothache, catch) + \mathbf{P}(Cavity, toothache, \neg catch)] \\ &= \alpha [\langle 0.108, 0.016 \rangle + \langle 0.012, 0.064 \rangle] = \alpha \langle 0.12, 0.08 \rangle = \langle 0.6, 0.4 \rangle .\end{aligned}$$

- In simply we can calculate P(Cavity| Toothache) even if we do not know the value of P(toothache).
- We temporarily forget about te factor 1/P(toothache) and add up the values for cavity and ~cavity getting 0.12 and 0.08.But they do not sum to 1.
- So we normalize them by dividing each one by 0.12+0.08[0.12/0.12+0.08=0.6 and 0.08/0.12+0.08=0.4] getting the true probabilities of 0.6 and 0.4.

Module-3 **Lecture-36**

Learning Objective:

33. Independence

34. Probabilistic Reasoning

34.1 Bayes' Rule and it's Application

33. Independence

- Independence between propositions a and b can be written as:

$$P(a \mid b) = P(a) \text{ or } P(b \mid a) = P(b) \text{ or } P(a \wedge b) = P(a)\, P(b)$$

- Independence assertions are usually based on knowledge of the domain.
- As we have seen, they can dramatically reduce the amount of information necessary to specify the full joint distribution.
- If the complete set of variables can be divided into independent subsets, then the full joint can be factored into separate joint distributions on those subsets.
- For example, the joint distribution on the outcome of n independent coin flips, P(C1,…,Cn), can be represented as the product of n single-variable distributions P(Ci).

34. Probabilistic Reasoning

- Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.
- We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.
- In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behaviour of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI:

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

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

- Bayes' rule
- Bayesian Statistics

34.1 Bayes' Theorem/Bayes' Rule and it's application:

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

- In probability theory, it relates the conditional probability and marginal probabilities of two random events.
- Bayes' theorem was named after the British mathematician Thomas Bayes. The Bayesian inference is an application of Bayes' theorem, which is fundamental to Bayesian statistics
- It is a way to calculate the value of P(B|A) with the knowledge of P(A|B).
- Bayes' theorem allows updating the proba bility prediction of an event by observing new information of the real world.

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

Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:

As from product rule we can write:

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

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

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

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

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

The above equation (a) is called as Bayes' rule or Bayes' theorem. This equation is basic of most modern AI systems for probabilistic inference.

It shows the simple relationship between joint and conditional probabilities. Here,

P(A|B) is known as posterior, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.

P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.

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

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

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

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

Applying Bayes' rule:

Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(B), and P(A). This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one.

Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

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

Example:

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

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

- The Known probability that a patient has meningitis disease is 1/30,000.
- The Known probability that a patient has a stiff neck is 2%.

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

P(a|b) = 0.8
P(b) = 1/30000
P(a)= .02

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

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

Application of Bayes' theorem in Artificial intelligence:

Following are some applications of Bayes' theorem:

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

Learning Objective:

34. Probabilistic Reasoning

34.2 Representing Knowledge in an Uncertain Domain: Bayesian Network

34.2 Representing Knowledge in an Uncertain Domain: Bayesian Network

- Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty.
- We can define a Bayesian network as: "A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."
- It is also called a Bayes' network, belief network, decision network, or Bayesian model.
- Bayesian networks are probabilistic, because these networks are built from a probability distribution, and also use probability theory for prediction and anomaly detection.
- Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction, and decision making under uncertainty.
- Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:
    - Directed Acyclic Graph
    - Table of conditional probabilities.

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

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

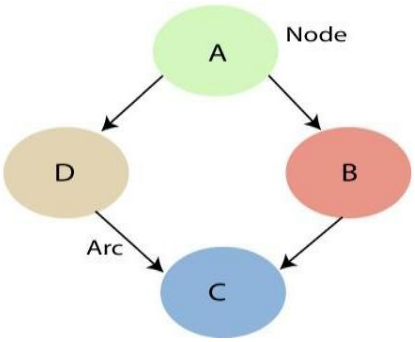


Fig.: Bayesian network graph

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

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

The Bayesian network has mainly two components:

- Causal Component
- Actual numbers

Each node in the Bayesian network has condition probability distribution P(Xi |Parent(Xi)), which determines the effect of the parent on that node.

The Bayesian network is based on Joint probability distribution and conditional probability.

Learning Objective:
35. The Semantics of Bayesian Networks

35. The Semantics of Bayesian Networks:

There are two ways in which we can understand Semantics of Bayesian networks:

1. See the network as representation of the joint probability distribution. This is useful in understanding how to construct networks.
2. See the networks as an encoding of a collection of conditional independence statements. This is useful in designing inference procedures. However, the two ways are equivalent.

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

Problem:

- Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.
- Solution:
- The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.
- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- The conditional distributions for each node are given as conditional probabilities table or CPT.
- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
- In CPT, a boolean variable with k boolean parents contains 2^K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values.

List of all events occurring in this network:

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

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

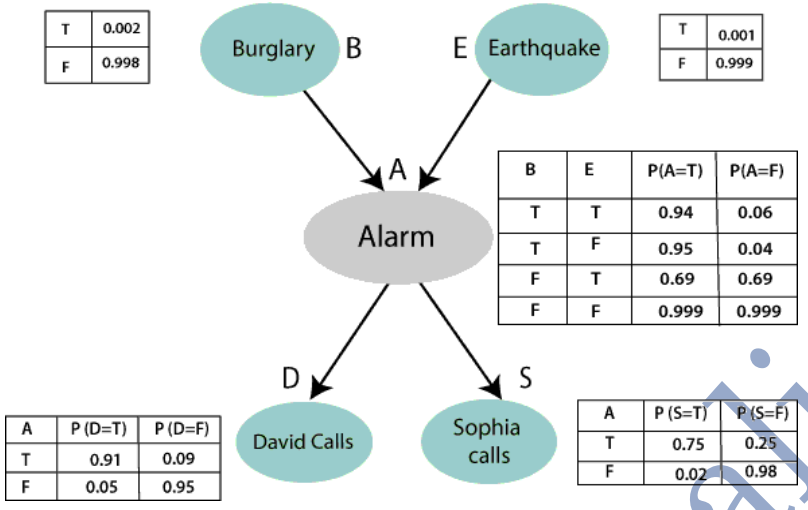
P[D, S, A, B, E]= P[D | S, A, B, E]. P[S, A, B, E]

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

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

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

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



T	0.002
F	0.998

T	0.001
F	0.999

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

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

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

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

P(B= True) = 0.002, which is the probability of burglary.

P(B= False)= 0.998, which is the probability of no burglary.

P(E= True)= 0.001, which is the probability of a minor earthquake

P(E= False)= 0.999, Which is the probability that an earthquake not occurred. We can provide the conditional probabilities as per the below tables:

Conditional probability table for Alarm A:

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

B	E	P(A= True)	P(A= False)
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

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

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

Conditional probability table for Sophia Calls:

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

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

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

P(S, D, A, ¬B, ¬E) = P (S|A) *P (D|A)*P (A|¬B ^ ¬E) *P (¬B) *P (¬E).

= 0.75* 0.91* 0.001* 0.998*0.999
= 0.00068045.

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

Learning Objective:
36. Efficient Representation of Conditional Distribution

36. Efficient Representation of Conditional Distribution

Conditional Distribution

If X and Y are two jointly distributed random variables, then the conditional distribution of Y given X is the probability distribution of Y when X is known to be a certain value.
For example, the following two-way table shows the results of a survey that asked 100 people which sport they liked best: baseball, basketball, or football.

	Baseball	Basketball	Football	Total
Male	13	15	20	48
Female	23	16	13	52
Total	36	31	33	100

If we want to know the probability that a person prefers a certain sport given that they are male, then this is an example of a conditional distribution.
The value of one random variable is known (the person is male), but the value of the other random variable is unknown (we don't know their favourite sport).
To find the conditional distribution of sports preference among males, we would simply look at the values in the row for
Male in the table:

	Baseball	Basketball	Football	Total
Male	13	15	20	48
Female	23	16	13	52
Total	36	31	33	100

The conditional distribution would be calculated as:

- Males who prefer baseball: 13/48 = .2708
- Males who prefer basketball: 15/48 = .3125
- Males who prefer football: 20/48 = .4167

Notice that the sum of the probabilities adds up to 1: 13/48 + 15/48 + 20/48 = 48/48 = 1.

We can use this conditional distribution to answer questions like: Given that an individual is male, what is the probability that baseball is their favourite sport?
From the conditional distribution we calculated earlier, we can see that the probability is .2708.

In technical terms, when we calculate a conditional distribution we say that we're interested in a particular subpopulation of the overall population. The subpopulation in the previous example was males:

	Baseball	Basketball	Football	Total
Male	13	15	20	48
Female	23	16	13	52
Total	36	31	33	100

Subpopulation

And when we want to calculate a probability related to this subpopulation, we say that we're interested in a particular **character of interest**. The character of interest in the previous example was baseball:

Character of Interest

	Baseball	Basketball	Football	Total
Male	13	15	20	48
Female	23	16	13	52
Total	36	31	33	100

Subpopulation

To find the probability that the character of interest occurs in the subpopulation, we simply divide the value of the character of interest (e.g. 13) by the total values in the subpopulation (e.g. 48) to get 13/48 = **.2708**.

Learning Objective:

37. Inference in Bayesian Networks

37.1 Exact Inference in Bayesian Networks

37. Inference in Bayesian Networks

- In practice, exact inference is not used widely, and most probabilistic inference algorithms are approximate.
- There are two types of inference techniques: exact inference and approximate inference.
- Exact inference algorithms calculate the exact value of probability P(X|Y) Algorithms in this class include the elimination algorithm, the message-passing algorithm (sum-product, belief propagation), and the junction tree algorithms.

Approaches to inference:

1. Exact methods
   - Enumeration
   - Variable elimination
   - Belief propagation in poly trees etc.
2. Approximate methods
   - Stochastic simulation / sampling methods
   - Markov chain Monte Carlo
   - Genetic algorithms
   - Neural networks
   - Simulated annealing etc.

37.1 Exact Inference in Bayesian Networks

Sum out variables from the joint without actually constructing its explicit representation.

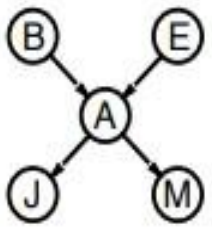
Simple query on the burglary network:

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

Rewrite full joint entries using product of CPT entries:

$\Pr(B \mid j, m) = \alpha \sum_e \sum_a \Pr(B) P(e) \Pr(a \mid B, e) P(j \mid a) P(m \mid a)$

$= \alpha \Pr(B) \sum_e P(e) \sum_a \Pr(a \mid B, e) P(j \mid a) P(m \mid a)$



Recursive depth-first enumeration: O(n) space, $O(d^n)$ time.

Enumeration Algorithm

```
function ENUMERATION-ASK(X, e, bn) returns a distribution over X
  inputs: X, the query variable
          e, observed values for variables E
          bn, a Bayesian network with variables {X} ∪ E ∪ Y
  Q(X) ← a distribution over X, initially empty
  for each value x_i of X do
      extend e with value x_i for X
      Q(x_i) ← ENUMERATE-ALL(VARS[bn], e)
  return NORMALIZE(Q(X))

function ENUMERATE-ALL(vars, e) returns a real number
  if EMPTY?(vars) then return 1.0
  Y ← FIRST(vars)
  if Y has value y in e
      then return P(y | Pa(Y)) × ENUMERATE-ALL(REST(vars), e)
      else return Σ_y P(y | Pa(Y)) × ENUMERATE-ALL(REST(vars), e_y)
          where e_y is e extended with Y = y
```

Learning Objective:
37. Inference in Bayesian Networks
37.2 Approximate Inference in Bayesian Networks

37.2 Approximate Inference in Bayesian Networks

Instead of creating a sample and then rejecting it, it is possible to mix sampling with inference to reason about the probability that a sample would be rejected. In **importance sampling** methods, each sample has a weight, and the sample average is computed using the weighted average of samples. **Likelihood weighting** is a form of importance sampling where the variables are sampled in the order defined by a belief network, and evidence is used to update the weights. The weights reflect the probability that a sample would not be rejected.

```
function Likelihood-Weighting(X, e, bn, N) returns an estimate of P(X|e)
  local variables: W, a vector of weighted counts over X, initially zero

  for j = 1 to N do
      x, w ← Weighted-Sample(bn)
      W[x] ← W[x] + w where x is the value of X in x
  return Normalize(W[X])

function Weighted-Sample(bn, e) returns an event and a weight

  x ← an event with n elements; w ← 1
  for i = 1 to n do
      if X_i has a value x_i in e
          then w ← w × P(X_i = x_i | parents(X_i))
          else x_i ← a random sample from Pr(X_i | parents(X_i))
  return x, w
```

Example

Suppose we want to use likelihood weighting to compute

P (Tampering | smoke ∧ ¬ report) .

The following table gives a few samples.

In this table, s is the sample; **e** is **¬ smoke ∧ report** . The weight is P (e | s) , which is equal to **P (smoke | Fire) * P (¬ report | Leaving)** , where the value for **Fire** and **Leaving** are from the sample.

Tampering	*Fire*	*Alarm*	*Smoke*	*Leaving*	*Report*	***weight***
false	*true*	*false*	*true*	*true*	*false*	0.9 * 0.25 = 0.225
true	*true*	*true*	*true*	*false*	*false*	0.9 * 0.99 = 0.891
false	*false*	*false*	*true*	*true*	*false*	0.01 * 0.25 = 0.0025
false	*true*	*false*	*true*	*false*	*false*	0.9 * 0.99 = 0.891

P (tampering | ¬ smoke ∧ report) is estimated from the weighted proportion of the samples that have **Tampering** true

Module-4 Lecture-42

Learning Objective:
38. Learning
39. Statistical Learning
39.1 Leaning with complete data
39.2 Learning with hidden variables

38. Learning

- Learning in AI is also called machine learning.
- Machine learning is an application of artificial intelligence that provides the systems to automatically learn and improves from experience without being explicitly programmed.
- An agent is learning if it improves its performance on future tasks after making observations about the world. Learning is the improvement of performance with experience over time.
- Learning element is the portion of a learning AI system that decides how to modify the performance element and implements those modifications.
- We all learn new knowledge through different methods, depending on the type of material to be learned, the amount of relevant knowledge we already possess, and the environment in which the learning takes place.

39. Statistical Learning

Statistical Learning is Artificial Intelligence is a set of tools for machine learning that uses statistics and functional analysis. In simple words, Statistical learning understands from training data and predicting on unseen data. Statistical learning is used to build predictive models based on the data. Statistical learning can be used to build applications for computer vision, text analytics, voice recognition, etc. These tools broadly come under two classes: supervised learning & unsupervised learning.

Supervised Learning:

- Supervised learning is a type of machine learning method in which we provide sample labelled data to the machine learning system in order to train it, and on that basis, it predicts the output.
- The system creates a model using labelled data to understand the datasets and learn about each data,
- once the training and processing are done then we test the model by providing a sample data to check
- whether it is predicting the exact output or not.
- The goal of supervised learning is to map input data with the output data. The supervised learning is
- based on supervision.
- The example of supervised learning is spam filtering.
- Supervised Learning is the one, where you can consider the learning is guided by a teacher. We have a dataset which acts as a teacher and its role is to train the model or the machine. Once the model gets trained it can start making a prediction or decision when new data is given to it.
- Supervised learning classified into two categories of algorithms:

1. Regression
2. Classification

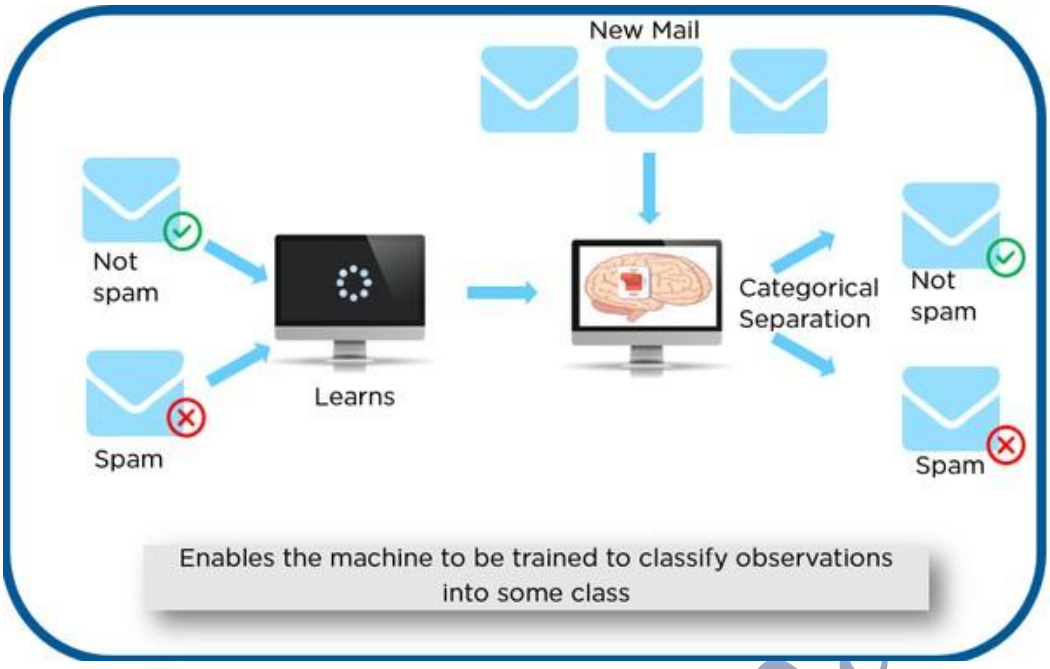


Fig:- : Example of Supervised learning

Unsupervised Learning:

- Unsupervised learning is a learning method in which a machine learns without any supervision.
- The training is provided to the machine with the set of data that has not been labelled, classified, or categorized, and the algorithm needs to act on that data without any supervision.
- The goal of unsupervised learning is to restructure the input data into new features or a group of objects with similar patterns.
- In unsupervised learning, we don't have a predetermined result. The machine tries to find useful insights from the huge amount of data.
- The model learns through observation and finds structures in the data. Once the model is given a dataset, it automatically finds patterns and relationships in the dataset by creating clusters in it. What it cannot do is add labels to the cluster; like it cannot say this a group of apples or mangoes, but it will separate all the apples from mangoes.
- Suppose we presented images of apples, bananas and mangoes to the model, so what it does, based on some patterns and relationships it creates clusters and divides the dataset into those clusters. Now if a new data is fed to the model, it adds it to one of the created clusters.
- Unsupervised learning classified into two categories of algorithms:
  - Clustering
  - Association

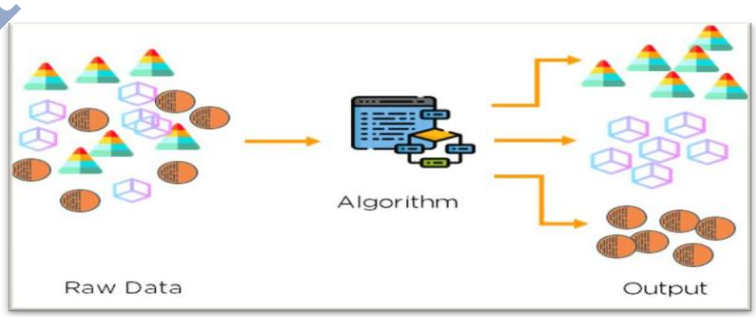


Fig:- Example of Unsupervised Learning

Module-4 **Lecture-43**

Learning Objective:

39. Statistical Learning

39.1 Leaning with complete data

39.2 Learning with hidden variables

39.1 Leaning with complete data

- Statistical learning methods are based on simple task parameter learning with complete data. Parameter learning involves finding the numerical parameters for a probability model with a fix structure. E.g: In Bayesian network conditional probabilities are obtained for a given scenario. Data are complete when each point contains values for every variable in a specific learning model.
- There are different method are present which are working in complete data. Those are:
    1. Maximum-likelihood parameter learning
    2. Naïve Bayes models
    3. Continuous Model
    4. Bayesian Parameter Learning

Parameter learning:

- Data are complete when each data point contains values for every variable in the probability model being learned.
- Statistical learning methods begin with the simplest task: parameter learning with complete data.
- Parameter learning is an important aspect of learning in Bayesian networks.
- Although the maximum likelihood algorithm is often effective, it suffers from over fitting when there is insufficient data.
- Over fitting can occur when the hypothesis space is too expressive, so that it contains many hypotheses that fit the data set.
- To address this, prior distributions of model parameters are often imposed.
- When training a Bayesian network, the parameters of the network are optimized to fit the data.
- A **Parameter learning task** involves **finding the numerical parameters** for a probability model whose structure is fixed.
- Complete data greatly simplify the problem of **learning the parameters of a complex model**.

Maximum-likelihood parameter learning: Discrete models

- Suppose we buy a bag of lime and cherry candy from a new manufacturer whose lime–cherry proportions are completely unknown—that is, the fraction could be anywhere between 0 and 1.
- In that case, we have a continuum of hypotheses.
- The parameter in this case, which we call θ, is the proportion of cherry candies, and the hypothesis is h_θ.
- The proportion of limes is just $1 - \theta$.
- If we assume that all proportions are equally likely a priori, then a maximum likelihood approach is reasonable.
- If we model the situation with a Bayesian network, we need just one random variable, Flavor (the flavor of a randomly chosen candy from the bag).

➢ It has values cherry and lime, where the probability of cherry is θ (see Fig below)

➢ Now suppose we unwrap N candies, of which **—c"** are cherries and l = N − c are limeAccording to Equation, the likelihood of this particular data set is:

$$P(\mathbf{d}|h_\theta) = \prod_{j=1}^{N} P(d_j|h_\theta) = \theta^c \cdot (1-\theta)^\ell \,.$$

- The maximum-likelihood hypothesis is given by the value of θ that maximizes this expression.
- The same value is obtained by maximizing the log likelihood.

$$L(\mathbf{d}|h_\theta) = \log P(\mathbf{d}|h_\theta) = \sum_{j=1}^{N} \log P(d_j|h_\theta) = c\log\theta + \ell\log(1-\theta) \,.$$

(By taking logarithms, we reduce the product to a sum over the data, which is usually easier to maximize.)

- To find the maximum-likelihood value of θ, we differentiate‖ L‖ with respect to θ and set the resulting expression to zero:

$$\frac{dL(\mathbf{d}|h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \quad \Rightarrow \quad \theta = \frac{c}{c+\ell} = \frac{c}{N} \,.$$

- In English, then, the maximum-likelihood hypothesis h_{ML} asserts that the actual proportion of cherries in the bag is equal to the observed proportion in the candies unwrapped so far.
- Standard method for maximum-likelihood parameter learning:
  1. Write down an expression for the likelihood of the data as a function of the parameter(s).
  2. Write down the derivative of the log likelihood with respect to each parameter.
  3. Find the parameter values such that the derivatives are zero.

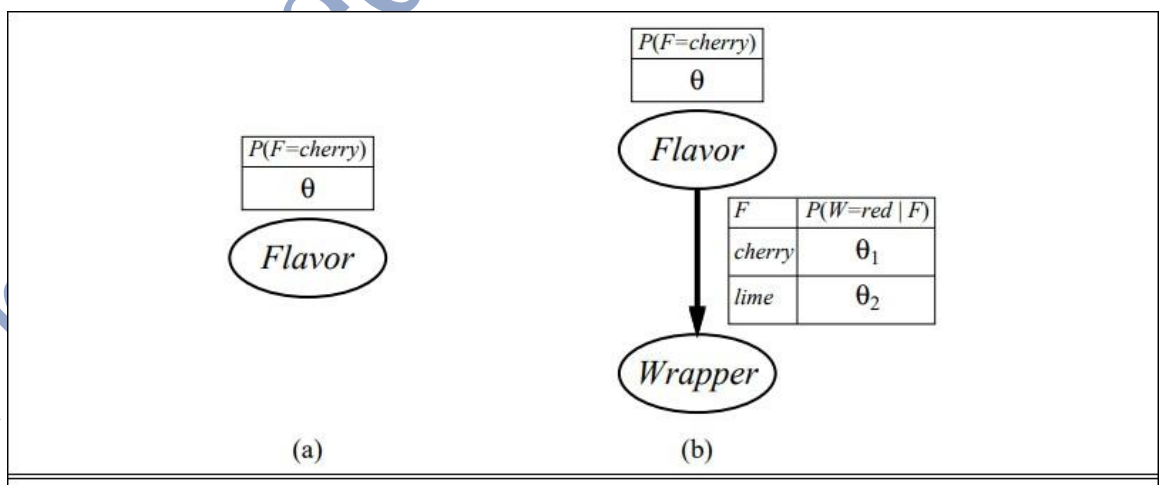


Fig :- (a) Bayesian network model for the case of candies with an unknown proportion of cherries and limes.
(b) Model for the case where the wrapper color depends (probabilistically) on the candy flavor.

Naïve Bayes models

- Probably the most common Bayesian network model used in machine learning is the naive Bayes model.
- In this model, the ―class‖ variable C (which is to be predicted) is the root and the ―attribute‖ variables X_i are the leaves.
- The model is ―naive‖ because it assumes that the attributes are conditionally independent of each other, given the class.
- The model in Fig.(b) is a naive Bayes model with just one attribute.
- Assuming Boolean variables, the parameters are

$$\theta = P(C = true), \theta_{i1} = P(X_i = true | C = true), \theta_{i2} = P(X_i = true | C = false).$$

- The maximum-likelihood parameter values are found in exactly the same way as for Fig. 7.3(b). Once the model has been trained in this way, it can be used to classify new examples for which the class variable C is unobserved.
- With observed attribute values x1, . . . , xn, the probability of each class is given by

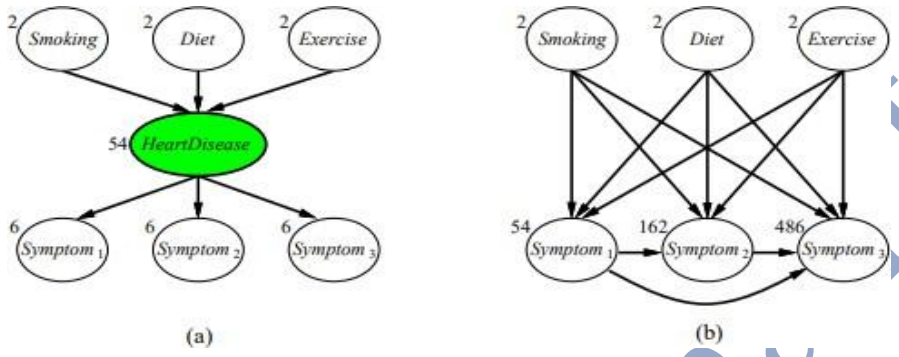
$$\mathbf{P}(C|x_1, \ldots, x_n) = \alpha \, \mathbf{P}(C) \prod_i \mathbf{P}(x_i|C) \, .$$

- A deterministic prediction can be obtained by choosing the most likely class.
- The method learns fairly well but not as well as decision-tree learning; this is presumably because the true hypothesis— which is a decision tree—is not representable exactly using a naive Bayes model. Naive Bayes learning scales well to very large problems: with n Boolean attributes, there are just 2n + 1 parameters, and no search is required to find h_{ML}, the maximum-likelihood naive Bayes hypothesis. Finally, naive Bayes learning has no difficulty with noisy data and can give probabilistic predictions when appropriate.

39.2 Learning with hidden variables

- Many real-world problems have hidden variables, which are not observable in the data that are available for learning.
- For example, medical records often include the observed symptoms, the diagnosis, and the treatment applied, but they seldom contain a direct observation of the disease itself.
- The hidden variables can dramatically reduce the number of parameters required to specify a Bayesian network. The below Fig. which shows a small, fictitious diagnostic model for heart disease.
- There are three observable predisposing factors and three observable symptoms (which are too depressing to name). Assume that each variable has three possible values (e.g., none, moderate, and severe).
- Removing the hidden variable from the network in (a) yields the network in (b); the total number of parameters increases from 78 to 708.Thus, latent variables can dramatically reduce the number of parameters required to specify a Bayesian network. This, in turn, can dramatically reduce the amount of data needed to learn the parameters.
- Hidden variables are important, but they do complicate the learning problem. In below Fig.(a), for example, it is not obvious how to learn the conditional distribution for Heart Disease, given its parents, because we do not know the value of Heart Disease in each case; the same problem arises in learning the distributions for the symptoms.

- This section describes an algorithm called expectation–maximization, or EM, that solves this problem in a very general way.
- We will show three examples and then provide a general description. The algorithm seems like magic at first, but once the intuition has been developed, one can find applications for EM in a huge range of learning problems.



(Fig. : (a) A simple diagnostic network for heart disease, which is assumed to be a hidden variable. Each variable has three possible values and is labeled with the number of independent parameters in its conditional distribution; the total number is 78. (b) The equivalent network with Heart Disease removed. Note that the symptom variables are no longer conditionally independent given their parents. This network requires 708 parameters.)

The **EM Algorithm :**It is a very general algorithm used to learn probabilistic models in which variables are hidden; that is, some of the variables are not observed. Models with hidden variables are sometimes called latent variable models.

Module-4 Lecture-44

Learning Objective:
40. Rote Learning
41. Learning by taking advice

40. Rote Learning

- Rote learning is the basic learning activity. Rote learning is a memorization technique based on repetition. It is also called memorization because the knowledge, without any modification, is simply copied into the knowledge base. As computed values are stored, this technique can save a significant amount of time.
- Rote learning techniques can also be used in complex learning systems provided sophisticated techniques are employed to use the stored values faster and there is a generalization to keep the number of stored information down to a manageable level. Example: Checkers-playing program.
- The idea is that one will be able to quickly recall the meaning of the material the more one repeats it.

  Ex:- 5!=5*4*3*2*1=120
  6!=5!*6=120*6=720
- For example we may use this type of learning when we memorize multiplication tables. In this method we store the previous computed values, for which we do not have to recompute them later.
- Also we can say rote learning is one type of existing or base learning. For example, in our childhood, we have the knowledge that "sun rises in the east". So in our later stage of learning we can easily memorize things. Hence in this context, a system may simply memorize previous solutions and recall them when confronted with the same problem. Generally access of stored value must be faster than it would be to recompute.
- The idea is that one will be able to quickly recall the meaning of the material the more one repeats it.
- Some of the alternatives to rote learning include meaningful learning, associative learning, and active learning. Uses this technique to learn the board positions it evaluates in its look-ahead search.
- Rote learning is most basic learning activities when the computer stores the date it is performing rudimentary form of learning.
- Hence the act of storage allows the program to perform better in the future. Also, in the case of data caching, we store computed value then we do not recompute again when the computation is more expensive than this strategy can save the significant amount of time.
- Hence caching has been used in AI program to produce some surprising performance improvements. Such caching is known as Rote learning.

Rote learning includes the capabilities:

- **Organized storage of information:** In order to improve the performance and speed up to use the stored value than it would be to recompute it. Then there must be a special technique that accesses the stored value quickly.
- **Generalization:** - Here the number of distinct object that stores is very large. So that to keep the number of stored object manageable level some kind of generalization technique is necessary.

41.Learning by taking advice

- This is a simple form of learning. Suppose a programmer writes a set of instructions to instruct the computer what to do, the programmer is a teacher and the computer is a student. Once learned (i.e. programmed), the system will be in a position to do new things.
- The advice may come from many sources: human experts, internet to name a few. This type of learning

requires more inference than rote learning. The knowledge must be transformed into an operational form before stored in the knowledge base. Moreover the reliability of the source of knowledge should be considered.

- The system should ensure that the new knowledge is conflicting with the existing knowledge. FOO (First Operational Operationalised), for example, is a learning system which is used to learn the game of Hearts. It converts the advice which is in the form of principles, problems, and methods into effective executable (LISP) procedures (or knowledge). Now this knowledge is ready to use.
- Computer program might make use of the advice by adjusting its static evaluation function to include a factor depending on the other control. If we have designed a data structure for playing any game then first, we rule out all the advice before playing the game. Hence human user first translates the advice then plays the game.

Module-4 Lecture-45

Learning Objective:
42. Learning In Problem Solving

42. Learning In Problem Solving

- Humans have a tendency to learn by solving various real world problems.
- The forms or representation, or the exact entity, problem solving principle is based on reinforcement learning.
- Therefore, repeating a certain action results in a desirable outcome while the action is avoided if it results into undesirable outcomes.
- As the outcomes have to be evaluated, this type of learning also involves the definition of a utility function. This function shows how much is a particular outcome worth?
- In reinforcement learning, the system knows the desirable outcomes but does not know which actions result into desirable outcomes.
- In such a problem or domain, the effects of performing the actions are usually compounded with side-effects. Thus, it becomes impossible to specify the actions to be performed in accordance with the given parameters.
- Q-Learning is the most widely used reinforcement learning algorithm.
- Learning in problem takes various techniques to improve the performance. Like that problem solver, solve the problem by taking advice from someone else or teacher.

Learning By Parameter Adjustment: The most important question in the design of a learning program based on parameter adjustment. When the value of parameter increased and when the value of parameter decreased. The second question is how much should the value be changed. Hence the answer to the first question is that value of parameter that predicted the final outcome accurately should be increased while the value of parameter of poor predictors should be decreased. In designing the program, we have to know a priority how much weight should be attached to each feature being used. The solution of this we estimate the weight of problem through solving.

Learning By Chunking: Chunking is the process similar to macro-operators. The idea of chunking comes from psychological literature on memory and problem solving. Its computation basis is in production systems. So that solving the problem we have to define the number of productions in the memory depending upon the problem we called chunk from memory and solve the problem.

Learning Objective:
43. Learning from Examples: Induction Learning

43. Learning from Examples: Induction Learning

- This involves the Process of learning by example.
- Here the system tries to induce a general rule from a set of observed instances.The learning method extracts rules and patterns out of massive datasets.
- The learning a process belongs to supervised learning,does classification and construct class definitions, called induction.
- Inductive learning also called Concept Learning is a way in which AI systems try to use a generalized rule to carry out observations.
- The data is obtained as a result of machine learning or from domain experts (humans) where it is used to drive algorithms often called the Inductive Learning Algorithms (ALIs) that are used to generate a set of classification rules.
- Generally inductive learning is frequently used by humans. This form of learning is more powerful than the others.These classification rules that are generated are in the "If this then that" form.
- These rules determine the state of an entity at each iteration step in Learning and how the Learning can be effectively changed by adding more rules to the existing rule set.
- When the output and examples of the function are fed into the A.I. system, inductive Learning attempts to learn the function for new data.
- There are two methods for obtaining knowledge in the real world: first, from domain experts, and second, from machine learning.
- Domain experts are not very useful or reliable for large amounts of data. As a result, we are adopting a machine learning approach for this project.
- The other method, machine learning, replicates the logic of 'experts' in algorithms, but this work may be very complex, time-consuming, and expensive.
- As a result, an option is the inductive algorithms, which generate a strategy for performing a task without requiring instruction at each step.
- If we are given input samples (x),given to a function f and the output sample is (f(x)) .Then we can give different set of inputs(raw inputs) to the same function f and verify the output f(x).
- By using the outputs we generate (learn) the rules.

Example:

Mango->f(Mango)->Sweet(e1)
Banana->f(Banana)->Sweet(e2)

Fruits->f(fruits)->Sweet(General)

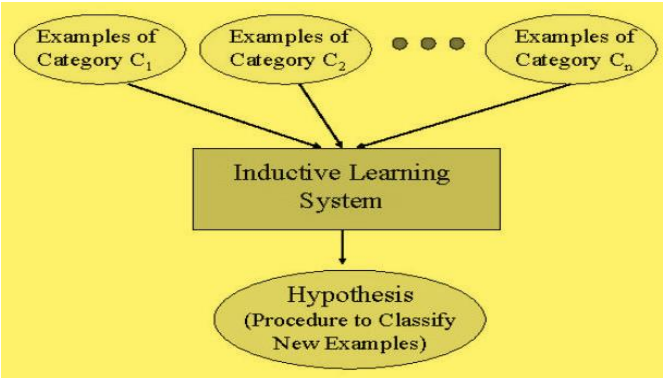


Fig.: Inductive Learning

Learning Objective:
44. Explanation Based Learning

44. Explanation Based Learning

- Explanation Based Learning or Explanation based generalization (EBG) is an algorithm for explanation based learning.
- It has two steps: first, explain the method and secondly, generalize the 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.
- In Explanation Based Learning (EBL), agent learns by examining particular situations and relating them to gained knowledge base. Also agent makes use of this gained knowledge for solving similar type of problems.
- EBL architecture takes two inputs from the environment: Specific goal and partial solution. Problem solver processes these inputs and gives justification to generalizer.
- Generalizer takes general concepts as input from the knowledge base and compared the explanation of the problem solver with it to come up with solution to the given problem.

Learning by Generalizing Explanations:

Given that

- 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.

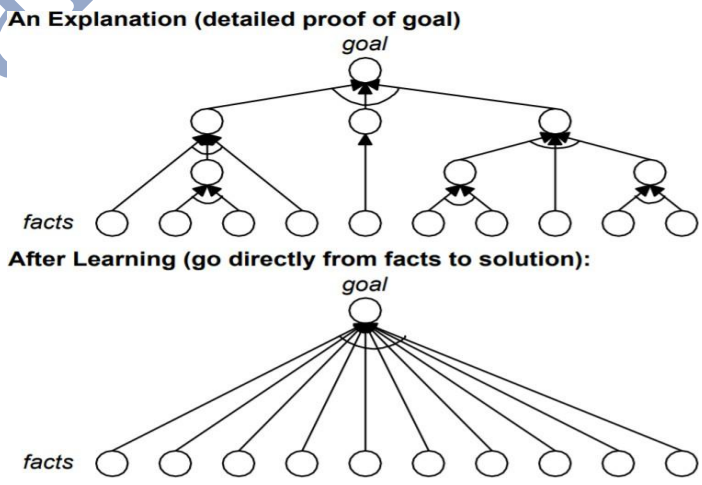


Fig.: Standard Approach to EBL

Unification-Based Generalization

- An explanation is an inter-connected collection of "pieces" of knowledge (inference rules, rewrite rules, etc.)
- These "rules" are connected using unification, as in Prolog .
- The generalization task is to compute the most general unifier that allows the "knowledge pieces" to be connected together as generally as possible

Sample EBL Problem

Initial Domain Theory

knows(?x,?y) AND nice-person(?y) -> likes(?x,?y)

animate(?z) -> knows(?z,?z)

human(?u) -> animate(?u)

friendly(?v) -> nice-person(?v)

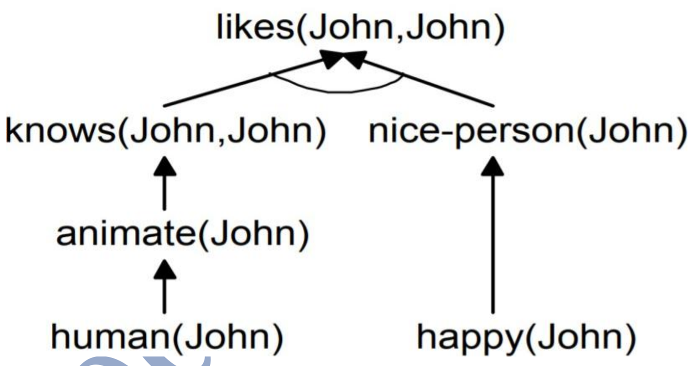
happy(?w) -> nice-person(?w)

Specific Example

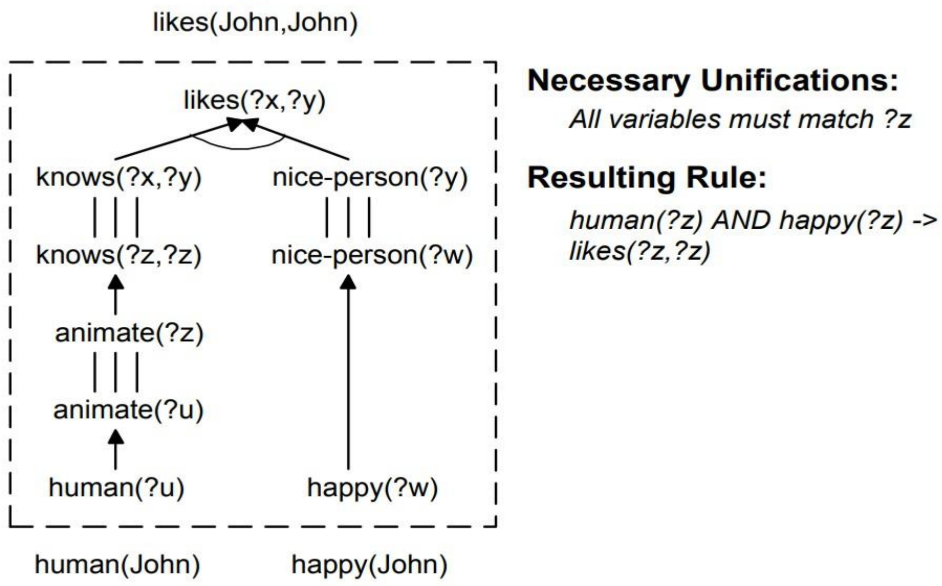
Given human(John) AND happy(John) AND male(John),

show that likes(John,John)

Explanation to Solve Problem



Explanation Structure



EBL Architecture:

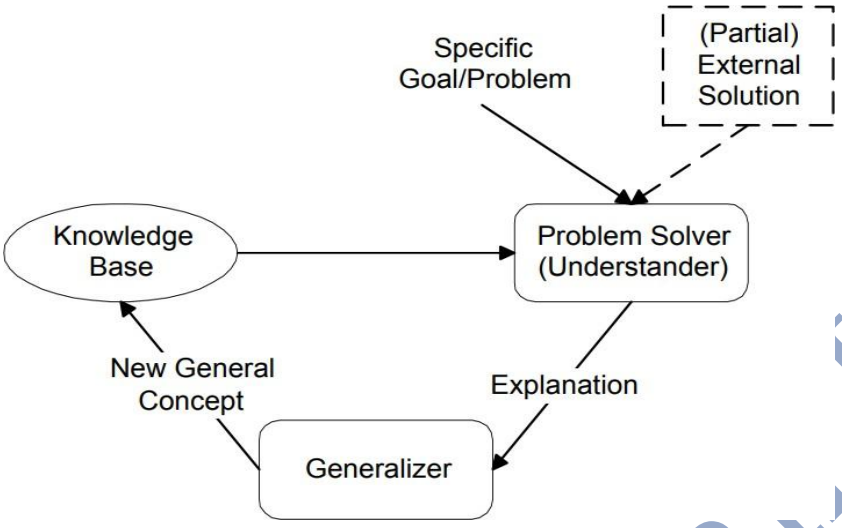


Fig.: EBL Architecture

Problem Solver:

It accepts 3 types of external inputs.

1. Goal concept is a problem statement in a complex form and agent needs to learn it.
2. Training examples are facts which explain an instance of the goal concept.
3. Inference rules represent the facts and protocols which show what learner already knows.

Generalizer: Output of the problem solver is given as input to the generalizer which compares the explanation of problem solver with knowledge base and gives output to the operationally pruner.

Operationally pruner: It takes two inputs one from generalized and one from operationally standard. Operationally standard gives description of the final concept; also it specifies the form in which learned concept should be expressed.

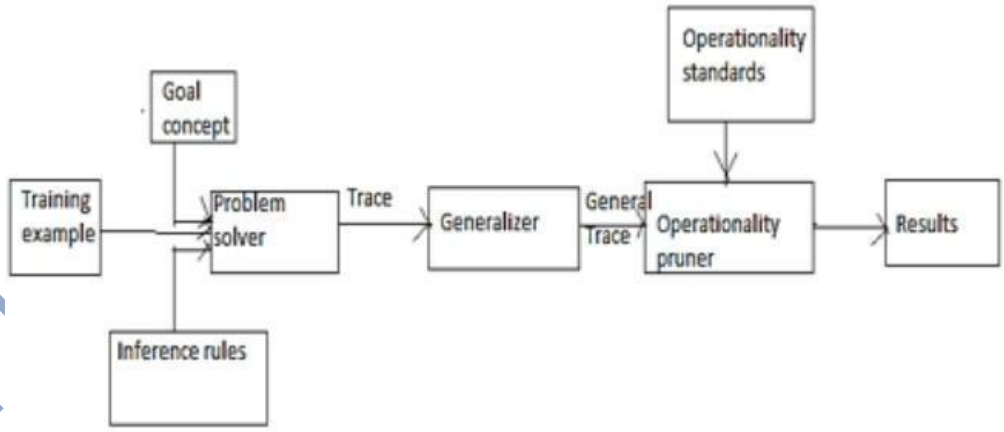


Fig.: Working of Problem Solver, Generalizer and Operationally pruner

Module-4 **Lecture-48**

Learning Objective:
45. Discovery and Analogical Learning

45. Discovery and Analogical Learning

Discovery

- Discovery is a restricted form of learning in which one entity acquires knowledge without the help of a teacher.
- Discovery learning takes place in problem solving situations where learners interact with their environment by exploring and manipulating objects, wrestling with questions and controversies, or performing experiments, while drawing on their own experience and prior knowledge.

Analogical Learning/Learning by Analogy:

- It is the process of learning a new concept or solution through the use of similar known concepts or solutions.
- We use this type of learning when solving problems on an exam where previously learned examples serve as a guide or when making frequent use of analogical learning. This form of learning requires still more inferring than either of the previous forms.
- It is a powerful inference tool.
- It generally involves abstracting details from a particular set of problems and resolving structural similarities between previously distinct problems.
- Analogical reasoning refers to this process of recognition and then applying the solution from the known problem to new problem.
- It involves developing a set of mappings between features of two instances.

Analogical Reasoning Steps

1. **Retrieve:-** Retrieve cases from memory that are relevant to solving it.
2. **Reuse:-** Map the solution from previous case to the target problem. This involves adapting the solution to fit new solution.
3. **Revise:-** Test the new solution to real world and, if necessary, revise.
4. **Retain:-** After the solution has been successfully adapted to target problem, store the resulting experience as the new case in memory.

Transformational Analogy:

- Suppose we are asked to prove a theorem in plane geometry.
- We might look for a previous theorem that is very similar and copy its proof, making substitutions when necessary.
- The idea is to transform a solution to a previous problem in to solution for the current problem.

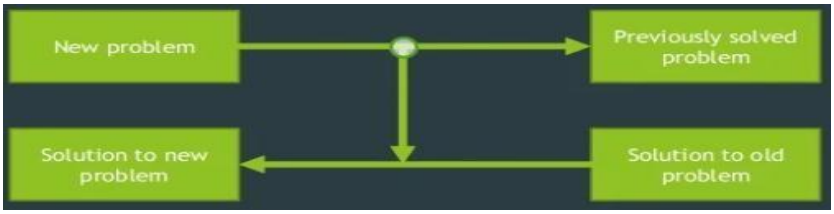


Fig.: Transformational Analogy

Derivational Analogy

- It only looks at the final solution.
- Often the twists and turns involved in solving an old problem are relevant to solving a new problem.
- The detailed history of problem solving episode is called derivation.
- Analogical reasoning that takes these histories into account is called derivational analogy.

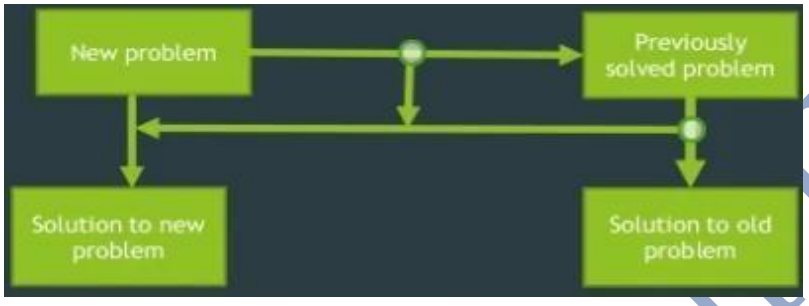


Fig.: Derivational Analogy

Learning Objective:
46. Formal Learning Theory

46. Formal Learning Theory

Formal learning theory is the mathematical embodiment of a normative epistemology. It deals with the question of how an agent should use observations about her environment to arrive at correct and informative conclusions.

- Given positive and negative examples.
- Produce algorithm that will classify future examples correctly with probability 1/h

Complexity of learning :

(i) The error tolerance (h).
(ii) The number of binary features present in the examples (t).
(iii) The size of the rule necessary to make the discrimination (f).

- If the number of training examples required is polynomial in h,t, and f→ then the concept is learnable.
- Few training examples are needed→ learnable we restrict the learner to the positive examples only.
  For example from the list of positive and negative examples of elephants shown in the figure below we want to induce the description "gray mammal,large"

gray?	mammal?	large?	vegetarian?	wild?		
+	+	+	+	+	+	*(Elephant)*
+	+	+	−	+	+	*(Elephant)*
+	+	−	+	+	−	*(Mouse)*
−	+	+	+	+	−	*(Giraffe)*
+	−	+	−	+	−	*(Dinosaur)*
+	+	+	+	−	+	*(Elephant)*

Fig.: Six positive and negative examples of the concept Elephant

Learning Objective:
47. Neural Net Learning

47. Neural Net Learning

Biological Neural Network:

- Biological neural network describes about the working principle of human brain.
- Brain is a most powerful computing machine from others. The inner working of human brain is built on concept of neurons and the networks of the neurons known as biological neural network.
- The brain contains more than 86 billion neurons. The neurons are connected and communicate with other neurons through axons.
- Dendrites are used for taking input from external environment or sensory organs. The electrical signal created by the input and these are quickly pass through the neural network and send to the other neuron through synapse to handle the issue.

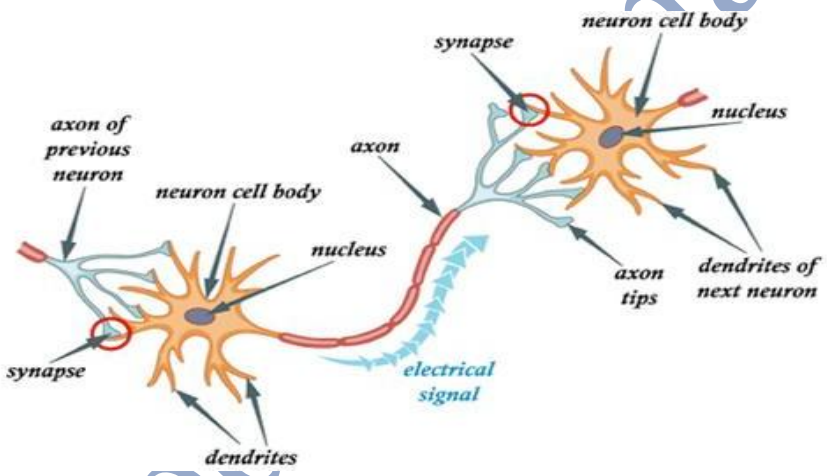


Fig.: Structure of a biological neuron

Artificial Neural Network (ANN):

- Neural network or ANN is based on biological neural network.
- Here multiple nodes are act as neurons. The neurons or nodes are interconnected and communicate with each other by links.
- ANN contains three layers. First layer is known as **input layer**, second layer is known as **hidden layer** and third layer is known as **output layer**.
- Each layer contains one or more neurons.
- The nodes of the input layer can take input data and perform operations on it and send the results of these operations to other neurons of the hidden layer. The hidden layer sends data to the output layer.
- The output of each node is known as its **activation or node value**.
- For increase the problem solving capabilities, we can increase the number of hidden layers and number of neurons in any given layer, and number of paths between neurons.

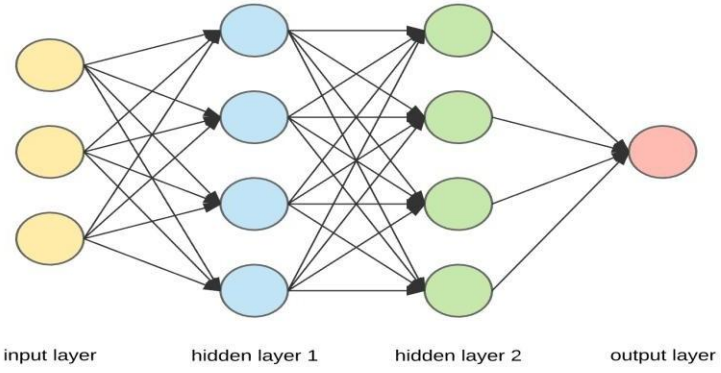


Fig.: Structure of Artificial Neural Network (ANN)

- In the network or model, each link assigned with some weight.
- Weight is nothing an integer number that controls the signal between the two neurons. If the network generates a "good or desired" output, there is no need to adjust the weights.
- However, if the network generates a "poor or undesired" output or an error, then the system update the weights in order to improve subsequent results.

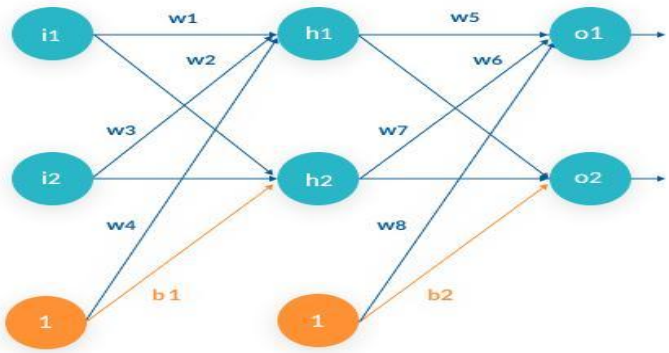


Fig.: Structure of Artificial Neural Network (ANN) with weights

Working of ANN

- At First, information is feed into the input layer which then transfers it to the hidden layers, and interconnection between these two layers assign weights to each input randomly at the initial point and then bias is added to each input neuron and after this, the weighted sum which is a combination of weights and bias is passed through the activation function.
- Activation Function has the responsibility of which node to fire for feature extraction and finally output is calculated.
- This whole process is known as Foreword Propagation.
- After getting the output model to compare it with the original output and the error is known and finally, weights are updated in backward propagation/back propagation to reduce the error and this process continues for a certain number of epochs (iteration). Finally, model weights get updated and prediction is done.
- Bias- It is an additional parameter in the Neural Network which is used to adjust the output along with the weighted sum of the inputs to the neuron.

Types of ANN

There are two important types of Artificial Neural Networks –

- FeedForward Neural Network
- FeedBack Neural Network

Module-4 Lecture-51

Learning Objective:
48. Genetic Learning

48. Genetic Learning

- Genetic Algorithms are algorithms that are based on the evolutionary idea of natural selection and genetics. GAs are adaptive heuristic search algorithms i.e. the algorithms follow an iterative pattern that changes with time. It is a type of reinforcement learning where the feedback is necessary without telling the correct path to follow. The feedback can either be positive or negative.
- A genetic algorithm (GA) is a heuristic search algorithm used to solve search and optimization problems. This algorithm is a subset of evolutionary algorithms, which are used in computation. Genetic algorithms employ the concept of genetics and natural selection to provide solutions to problems.
- These algorithms have better intelligence than random search algorithms because they use historical data to take the search to the best performing region within the solution space.
- GAs are also based on the behaviour of chromosomes and their genetic structure. Every chromosome plays the role of providing a possible solution. The fitness function helps in providing the characteristics of all individuals within the population. The greater the function, the better the solution.

Working of Genetic Algorithm:

Initialization: The genetic algorithm starts by generating an initial population. This initial population consists of all the probable solutions to the given problem. The most popular technique for initialization is the use of random binary strings.

Fitness assignment: The fitness function helps in establishing the fitness of all individuals in the population. It assigns a fitness score to every individual, which further determines the probability of being chosen for reproduction. The higher the fitness score, the higher the chances of being chosen for reproduction.

Selection: In this phase, individuals are selected for the reproduction of offspring. The selected individuals are then arranged in pairs of two to enhance reproduction. These individuals pass on their genes to the next generation.

The main objective of this phase is to establish the region with high chances of generating the best solution to the problem (better than the previous generation). The genetic algorithm uses the fitness proportionate selection technique to ensure that useful solutions are used for recombination.

Reproduction: This phase involves the creation of a child population. The algorithm employs variation operators that are applied to the parent population. The two main operators in this phase include crossover and mutation.

Crossover: This operator swaps the genetic information of two parents to reproduce an offspring. It is performed on parent pairs that are selected randomly to generate a child population of equal size as the parent population.

Mutation: This operator adds new genetic information to the new child population. This is achieved by flipping some bits in the chromosome. Mutation solves the problem of local minimum and enhances diversification. The following image shows how mutation is done.

Before Mutation

A5	1	1	1	0	0	0

After Mutation

A5	1	1	0	1	1	0

Replacement: Generational replacement takes place in this phase, which is a replacement of the old population with the new child population. The new population consists of higher fitness scores than the old population, which is an indication that an improved solution has been generated.

Termination: After replacement has been done, a stopping criterion is used to provide the basis for termination. The algorithm will terminate after the threshold fitness solution has been attained. It will identify this solution as the best solution in the population.

Application of Genetic Algorithms

Genetic algorithms have many applications, some of them are –

- Recurrent Neural Network
- Mutation testing
- Code breaking
- Filtering and signal processing
- Learning fuzzy rule base etc

Limitations of Genetic Algorithms:

1. **Computational Cost:** GAs often require significant computational resources due to the evaluation of large populations over multiple generations, especially for complex problems.
2. **Premature Convergence:** There is a risk of the algorithm converging to local optima, particularly if diversity within the population is not maintained.
3. **Dependence on Fitness Function:** The performance of GAs heavily relies on the quality and design of the fitness function. Poorly defined fitness functions can lead to suboptimal solutions or slow convergence.

Advantages of Genetic Algorithms

Genetic Algorithms (GAs) offer several unique advantages, making them highly effective for solving complex optimization problems:

1. **Global Optimization:** GAs are capable of finding global optima in complex, nonlinear, and high-dimensional search spaces, avoiding the pitfalls of local optima that plague traditional methods.
2. **Adaptability:** They can be applied to a wide range of problems, including combinatorial optimization, continuous optimization, and machine learning tasks, showcasing their versatility across domains.
3. **No Gradient Requirement:** Unlike gradient-based optimization methods, GAs do not rely on differentiable functions. This makes them suitable for problems with non-differentiable or discontinuous fitness landscapes, where traditional approaches fail.

Module-4 Lecture-52

Learning Objective:

49. Expert System

49.1 Characteristics of Expert System

49.2 Advantages of Expert System

49.3 Disadvantages of Expert System

49. Expert System

- An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert. It performs this by extracting knowledge from its knowledge base using the reasoning and inference rules according to the user queries.
- The data added in the knowledge base is added by humans that are expert in a particular domain and this software is used to acquire some information.
- These systems are designed for a specific domain, such as medicine, science, etc.
- The expert system can advise users as well as provide explanations to them about how they reached a particular conclusion or advice.
- The expert system is a part of AI, and the first ES was developed in the year 1970, which was the first successful approach of artificial intelligence.One of the common examples of an ES is a suggestion of spelling errors while typing in the Google search box.

Examples of Expert Systems:

MYCIN: It was based on backward chaining and could identify various bacteria that could cause acute infections. It could also recommend drugs based on the patient's weight.
DENDRAL: This Expert system used-fore chemical analysis to Predict molecular structure.
PXDES: This Expert system used to predict the degree and type of lung cancer.
CaDet: Expert system that could identify cancer at early stages.

49.1 Characteristics of Expert System

An expert system is usually designed to have the following general characteristics.

1. **High level Performance:** The system offers the highest level of expertise. It provides efficiency, accuracy and imaginative problem solving.
2. **Good Reliability:** The expert system must be as reliable as a human expert.
3. **Adequate Response time:** The system should be designed in such a way that it is able to perform within a small amount of time, comparable to or better than the time taken by a human expert to reach at a decision point.
4. **Understandable:** The system should be understandable i.e. be able to explain the steps of reasoning while executing. The expert system should have an explanation capability similar to the reasoning ability of human experts.
5. **Use symbolic representations:** Expert system use symbolic representations for knowledge (rules, networks or frames) and perform their inference through symbolic computations that closely resemble manipulations of natural language.
6. **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.

49.2 Advantages of Expert System:

(i) It improves the decision quality.
(ii) Reduce the number of human errors.
(iii) Offers consistent answer for the repetitive problem.
(iv) Helps you to get fast and accurate answers.
(v) Capable of explaining how they reached a solution.
(vi) Hold huge amounts of information.
(vii) Artificial Intelligence Expert Systems can steadily work without getting emotional, tensed or tired.

49.3 Disadvantages of Expert System:

(i) Errors in the knowledge base can lead to wrong decision.
(ii) The maintenance cost of an expert system is too expensive.
(iii) It is developed for a specific domain.
(iv) It needs to be up dated manually. It does not learn itself.
(v) Not able to recognize when there is no answer.
(vi) There is no flexibility and ability to adapt to changing environments.

Module-4 **Lecture-53**

Learning Objective:
50. Architecture of Expert System
51. Expert System Shell
52. Applications of Expert System

50. Architecture of Expert System

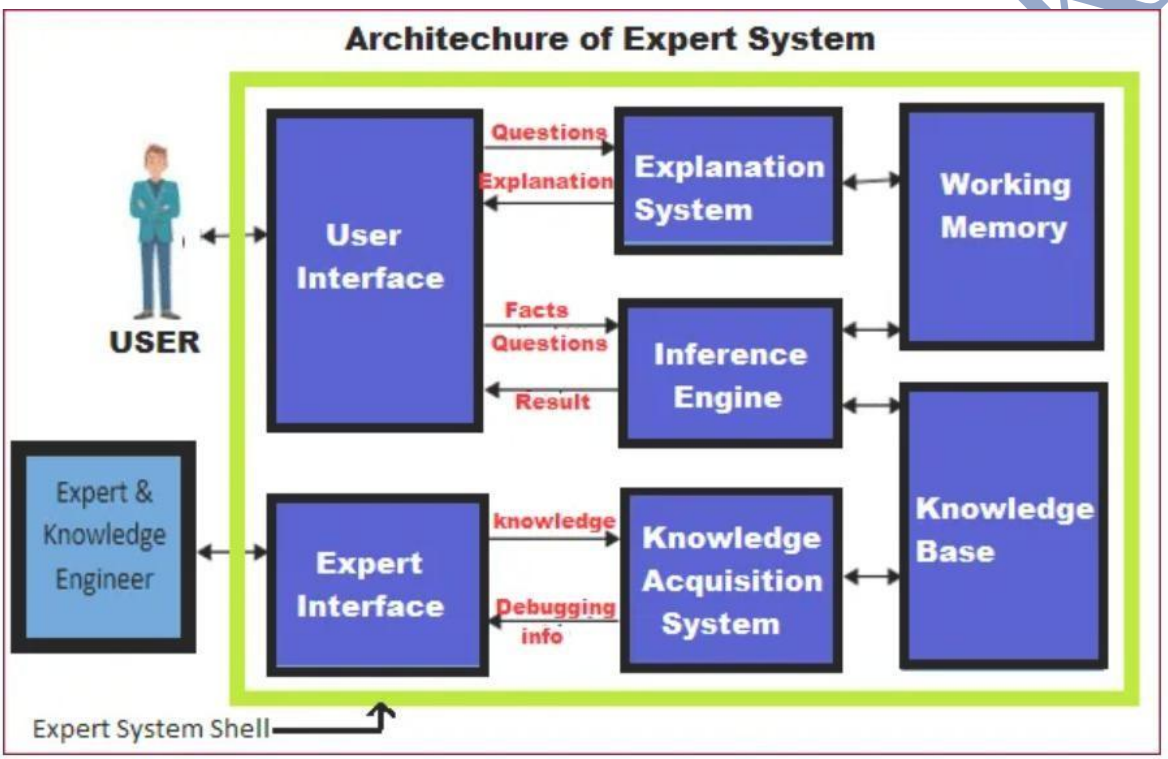


Fig. :Architecture of Expert System

The Expert System in AI consists of the following given three major components:

1. Knowledge Base
2. Inference engine
3. User Interface

1. Knowledge Base:

- The knowledge base contains the knowledge necessary for understanding, formulating and for solving problems.
- It is a warehouse of the domain specific knowledge captured from the human expert via the knowledge acquisition module.
- Thus we can say that the success of the Expert System Software mainly depends on the highly accurate and precise knowledge.
- The knowledge base of an ES is a store of both, factual and heuristic knowledge.

Factual Knowledge:- It is the information widely accepted by the Knowledge Engineers and scholars, typically found in textbooks or journals in the task domain.

Heuristic Knowledge:- It is about practice, accurate judgement, one's ability of evaluation, real life experiences and guessing.

Knowledge Acquisition

The term knowledge acquisition means how to get required domain knowledge by the expert system. The entire process starts by extracting knowledge from a human expert, converting the acquired knowledge into rules and injecting the developed rules into the knowledge base.

2. Inference engine:

The inference engine is the brain of the expert system. Inference engine contains rules to solve a specific problem. It refers the knowledge from the Knowledge Base. It selects facts and rules to apply when trying to answer the user's query. It provides reasoning about the information in the knowledge base. It also helps in deducting the problem to find the solution. This component is also helpful for formulating conclusions.

3. User Interface:

The user interface is the most crucial part of the Expert System Software. This component takes the user's query in a readable form and passes it to the inference engine. After that, it displays the results to the user. In other words, it's an interface that helps the user communicate with the expert system.

Other components are:

Working Memory

- Contains facts about a problem that are discovered during consultation with expert system.
- System matches this information with knowledge contained in the knowledge base to infer new facts. The inferred facts are added to the working memory.
- If forward chaining is used: It helps to describe the current running problem and record intermediate output.
- Records Intermediate Hypothesis & Decisions: 1. Plan, 2. Agenda, 3. Solution

Explanation System

- It helps to trace responsibility and justify the behaviour of expert system by firing questions and answers, such as Why, How, What, Where, When, Who. This module helps in providing the user with an explanation of the achieved conclusion.

Participants in the development of Expert System

There are three primary participants in the building of Expert System:

1. **Expert:** The success of an ES much depends on the knowledge provided by human experts. These experts are those persons who are specialized in that specific domain.
2. **Knowledge Engineer:** Knowledge engineer is the person who gathers the knowledge from the domain experts and then encodes that knowledge in a form that can be used by the expert system.

3. **End-User:** This is a particular person or a group of people who may not be experts, and working on the expert system needs the solution or advice for his queries, which are complex.

51. Expert System Shell

An **expert system shell** is a software development environment containing the basic components (Explanation facility, Reasoning capacity, Inference engine, user interface etc.) for building **expert systems**. It does not contain knowledge base. In other words, we can say that it is a readymade expert system without knowledge base. For every domain specific **system**, a knowledge engineer prepares knowledge base with the help of domain **experts** in a particular area. For example, if the knowledge engineer feeds, expert level knowledge of 'diagnosis of papaya plant 'then the tool will behave as an expert system for diagnosis of papaya plant. Thus an expert system shell provides a quick way of developing expert system.

Example of Expert System Shell:

(i) CLIPS (C Language Integrated Production System)
(ii) OPS5 and Eclipse
(iii) Java Expert System Shell (JESS) that provides fully developed Java API for creating an expert system.
(iv) Vidwan, a shell developed at the National Centre for Software Technology, Mumbai in 1993. It enables knowledge encoding in the form of IF-THEN rules.

52. Applications of Expert System

The following shows where ES can be applied.

i. Information management
ii. Hospitals and medical facilities
iii. Employee performance evaluation
iv. Virus detection
v. Useful for repair and maintenance projects
vi. Process monitoring and control
vii. Supervise the operation of the plant and controller
viii. Stock market trading
ix. Airline scheduling
x. Automobile design etc.

Module-4 Lecture-54

Learning Objective:

53. Knowledge Acquisition
53.1 Knowledge Acquisition Techniques

53. Knowledge Acquisition

- Knowledge Acquisition is the process of obtaining, gaining, extracting, receiving, and acquiring knowledge from the human experts, machines, specialist and high-qualified persons for an expert system, which must be carefully organized into rules or some other form of knowledge representation.
- Knowledge acquisition ropes the generation of knowledge-based systems through the growth of ethics, procedures, methodologies and tools.

53.1 Knowledge Acquisition Techniques

The following list introduces a few types of techniques used for acquiring, analysing and modelling knowledge:

A. Protocol Generation Techniques:

The aim of these techniques is to produce a protocol, i.e. a record of behavior, whether in audio, video or electronic media. Audio recording is the usual method, which is then transcribed to produce a transcript. It is include various types of interviews (unstructured, semi-structured and structured), reporting techniques (such as self-report and shadowing) and observational techniques.

Interviews: The interview is the most commonly used knowledge elicitation technique and takes many forms, from the completely unstructured interview to the formally planned, structured interview. It is a KA technique in which the knowledge engineer asks questions of the expert or end user .

Observation: Simply observing and making notes as the expert performs their daily activities can be useful, although a time-consuming process. Videotaping their task performance can be useful especially if combined with retrospective reporting techniques.

Commentary: These techniques generate protocols by having the expert provide a running commentary on a typical task used in the domain. In on-line PA, the expert is being recorded solving a problem, and concurrently a commentary is made. The nature of this commentary specifies two sub-types of the on-line method.

The basic technique here is the self-report: The expert performing the task may be describing what they are doing as problem solving proceeds.

Shadowing: A variant on this is to have another expert provide a running commentary on what the expert performing the task is doing. This is called shadowing.

Off-line PA: This allows the expert(s) to comment retrospectively on the problem solving session - usually by being shown an audio-visual record of it. An advantage of this is that the video can be paused or run at slow speed to allow time for full explanation. Variants of these reporting techniques involve a second expert commenting on another expert's performance or there could be group discussion of the protocol by a number of experts including its .

B. Laddering Techniques

Laddering techniques involve the construction, reviewing modification and validation of hierarchical knowledge, often in the form of ladders (i.e. tree diagrams).Here the expert and knowledge engineer both refer to a ladder presented on paper or a computer screen, and add, delete, rename or re-classify nodes as appropriate.

Laddering means setting elements in a ladder according to a common criterion in order to visualize them (easier for the expert) and confirm model completion (and, in rule systems generate the knowledge in the form of rules)

Concept Ladder: It shows classes of concepts and their sub-types. All relationships in the ladder, there is a relationship, also is more commonly known as a taxonomy and is vital to representing knowledge in almost all domains.

Composition Ladder: It shows the way a knowledge object is composed of its constituent parts. All relationships in the ladder are the part or part-of relationship. Also is a useful way of understanding complex entities such as machines, organisations and documents.

Decision Ladder: It shows the alternative courses of action for a particular decision. It also shows the pros and cons for each course of action, and possibly the assumptions for each pro and con. It is a useful way of representing detailed process knowledge.

Attribute Ladder: It shows attributes and values. All the adjectival values relevant to an attribute are shown as sub-nodes, but numerical values are not usually shown it is a useful way of representing knowledge of all the properties that can be associated with concepts in a domain.

Process Ladder: It shows process (tasks, activities) and the sub-processes (sub-tasks, sub-activities) of which they are composed. All relationships are the part of relationship; it is a useful way of representing process knowledge.

C. Matrix-based Techniques

It involves the construction of grids indicating such things as problems encountered against possible solutions. Important types include the use of frames for representing the properties of concepts and the repertory grid technique used to elicit, rate, analyze and categorize the properties of concepts .

Frames: Frames are a way of representing knowledge in which each concept in a domain is described by a group of attributes and values using a matrix representation. The left-hand column represents the attributes associated with the concept and the right-hand column represents the appropriate values. When the concept is a class, typical (default) values are entered in the right-hand column. The use of frames can also be adopted, although this would typically be used for validating previously acquired knowledge rather than for eliciting knowledge from scratch.

Timeline: A timeline is a type of tabular representation that shows time along the horizontal axis and such things as processes, tasks or project phases along the vertical axis. It is very useful for representing time-based process or role knowledge. It can also be used to acquire time-based knowledge. It is a simple representation that is often used in the early stages of knowledge elicitation to capture the basic of processes from the expert.

Matrix: A matrix is a type of tabular representation that comprises a 2-dimensional grid with filled-in grid cells. Ticks, crosses or comments in the matrix cells indicate which row object is applicable to which column

object. Two kinds of matrix are attributed matrix and relationship matrix.

Forms: A more recent form of knowledge model is the use of hypertext and web pages, in which relationships between concepts, or other types of knowledge, are represented by hyperlinks. This affords the use of structured text by making use of templates, i.e. generic headings. Different templates can be created for different knowledge types.