SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(AUTONOMOUS)
MCA DEPARTMENT

LECTURE NOTES

Subject Name: Artificial Intelligence

Year / Branch: I MCA II SEM

Regulation: R24

Prepared By: Mr. N. P. Gangadhar

Assistant Professor

Syllabus Contents

I MCA – II SEMESTER			
COURSE CODE:	**24MCA124A**	**CREDITS:**	**3**
COURSE TITLE:	**ARTIFICIAL INTELLIGENCE (Professional Elective-I)**	**L-T-P:**	**3-0-0**

PREREQUISITES: *Knowledge on Mathematical logic, Problem solving techniques, Knowledge representation and Learning Techniques may be helpful.*

COURSE EDUCATIONAL OBJECTIVES:

CEO1 *To familiarize students with Artificial Intelligence techniques for building well-engineered and efficient intelligent systems.*

CEO2 *In the applied point of view, some cutting edge applications of these systems will also be discussed.*

CEO3 *To have an appreciation and understanding of both the achievements of AI and the theory underlying those achievements.*

CEO4 *To have an appreciation for the engineering issues underlying the design of AI systems.*

CEO5 *To have a basic proficiency in a traditional AI language including an ability to write simple to intermediate programs and an ability to understand code written in that language.*

CEO6 *To have a basic understanding of some of the more advanced topics of AI such as Learning.*

UNIT - I: INTRODUCTION:	***Lecture Hrs:8***

What is AI? - The History of Artificial Intelligence - The State of the Art. **Intelligent Agents**: Agents& Environments – Good Behavior: The Concept of Rationality –The Nature of Environments - Structure of Agents.

UNIT - II: SOLVING PROBLEMS BY SEARCHING:	***Lecture Hrs:12***

Problem Solving Agents – Example Problems- Searching for Solutions-Uninformed Search Strategies - Informed (Heuristic) Search Strategies - Heuristic Functions.

Beyond Classical Search: Local Search Algorithms and Optimization Problems- Local Search in Continuous Spaces- Searching with Nondeterministic Actions- Searching with Partial Observations

UNIT – III: CONSTRAINT SATISFACTION PROBLEMS	***Lecture Hrs:12***

: Defining Constraint Satisfaction Problems- Constraint Propagation: Inference in CSPs- Backtracking search for CSPs.

Logical Agents: Knowledge–Based Agent - The Wumpus World – Logic - Propositional Logic: a Very Simple Logic - Propositional Theorem Proving - Effective Propositional Model Checking - Agents Based on Propositional Logic.

UNIT – IV FIRST ORDER LOGIC:	*Lecture Hrs:12*
Syntax and Semantic 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.	
UNIT - V KNOWLEDGE REPRESENTATION:	*Lecture Hrs:12*
Ontological Engineering - Categories and Objects - Events- Reasoning Systems for Categories - Reasoning with Default Information - The Internet Shopping World. **Quantifying Uncertainty:** Acting Under Uncertainty - Basic Probability Notation - Inference Using Full Joint Distributions – Independence - Bayes' Rule and Its Use.	

TEXT BOOKS:

1. *Artificial Intelligence A Modern Approach, 2015, Third Edition , Stuart J. Russell and Peter Norvig Pearson Education.*

REFERENCE BOOKS:

1. *Artificial Intelligence, 3/e,2009, Elaine Rich, Kevin Knight and Shiva shankar B Nair Tata McGraw Hill.*
2. *Artificial Intelligence-Structures and Strategies for Complex Problem Solving, 5/e, 2009, George F. Luther, Pearson Education.*
3. *Introduction to Artificial Intelligence, Eugene Charniak and Drew McDermott, Pearson Education, 1987.*

COURSE OUTCOMES: *On successful completion of this course, students will be able to:*		POs related to COs
CO1	**Explain** the key characteristics and structure of intelligent agents	PO1,PO2,PO3
CO2	**Solve** search problems by applying a suitable search strategy.	PO1,PO2,PO3, PO4, PO8
CO3	**Design** of an intelligent agent using propositional logic and first order logic to solve reasoning problems	PO1,PO2,PO3,PO8
CO4	**Construct** a knowledge representation system using logic and ontological	PO1,PO2,PO3,PO4, PO8

	engineering to facilitate inference in the given problem domain	
CO5	**Construct** a knowledge base for uncertain knowledge inference using probability distribution and solving problems	PO1,PO2,PO3,PO8

CO-PO MAPPING(DETAILED; HIGH:3; MEDIUM:2; LOW:1)

Course / POs / COs		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8
C204: Artificial Intelligence	**C204.1**	**3**	**3**	**2**	-	-	-	-	-
	C204.2	**3**	**3**	**2**	**2**	-	-	-	2
	C204.3	**3**	**2**	**3**	**-**	**-**	-	-	3
	C204.4	**3**	**3**	**3**	**2**	**-**	-	-	3
	C204.5	**3**	**3**	**3**	**-**	**-**	**-**	-	2
	C204	**3**	**2.8**	**2.6**	**2**				2.5

UNIT - I: INTRODUCTION:

“Artificial intelligence is the science of making machines do things that would require intelligence if done by humans.” — John McCarthy

Unit Overview

This unit introduces the basic concepts of **Artificial Intelligence** and explains its historical development. It discusses **intelligent agents**, their sensors, actuators, and rational behavior, which are central to standard AI foundations.

The unit also explains different **types of environments** and the **structure of agents**, including simple reflex, model-based, goal-based, utility-based, and learning agents.

Finally, it highlights important **applications of AI** in real-world fields such as healthcare, finance, robotics, education, and transportation.

Objectives of the unit

- To understand the basic concept of Artificial Intelligence.
- To study the evolution and historical development of AI.
- To understand the concept of intelligent agents and their environments.
- To learn the different types of environments and rational behavior of agents.
- To analyze the structure and types of intelligent agents used in AI systems.

Learning Outcomes

After completing this unit, students will be able to:

1. **Define** Artificial Intelligence and explain its basic concepts and historical development.
2. **Identify** the role of intelligent agents, sensors, actuators, and effectors in AI systems.
3. **Explain** the concept of rationality and the characteristics of rational agents, as presented in standard AI texts.
4. **Classify** different types of agent environments such as fully observable/partially observable, deterministic/stochastic, and static/dynamic.
5. **Describe** the structure of agents and distinguish among simple reflex, model-based, goal-based, utility-based, and learning agents.
6. **Discuss** major real-world applications of Artificial Intelligence in different domains.

Importance of Studying this Unit

This unit is important because it provides the **foundation of Artificial Intelligence** by introducing the core ideas of **agents, environments, and rational behavior**, which standard AI texts treat as central concepts.

It helps students understand how intelligent systems **perceive, think, and act**, and how the nature of the environment affects agent design.

The unit also builds the base for advanced AI topics by explaining **performance measures, percept sequences, and different agent types** such as reflex, goal-based, utility-based, and learning agents.

Studying this unit is essential for understanding how AI is applied in real-world systems such as robotics,

software agents, and decision-making systems.

Key Concepts

- **Artificial Intelligence (AI):** The field of building systems that can perceive, reason, learn, and act intelligently.
- **Agent:** An entity that perceives its environment through sensors and acts on it through actuators.
- **Rational Agent**: An agent that selects the action expected to maximize its performance measure based on its percept sequence and knowledge.
- **Environment:** The external world in which the agent operates and takes actions.
- **Sensors and Actuators**: Sensors gather information from the environment, while actuators perform actions on the environment.
- **Performance Measure:** The standard used to evaluate how successfully an agent performs.
- **Percept / Percept Sequence:** A percept is what the agent senses at a given moment; a percept sequence is the complete history of those inputs.
- **Types of Environments:** Fully observable/partially observable, deterministic/stochastic, episodic/sequential, static/dynamic, discrete/continuous, single-agent/multi-agent, known/unknown, and accessible/inaccessible.
- **Structure of Agents:** In standard AI, Agent = Architecture + Program.
- **Types of Agents:** Simple reflex, model-based reflex, goal-based, utility-based, and learning agents.
- **Applications of AI:** AI is used in healthcare, finance, robotics, education, transportation, gaming, agriculture, and many other domains.

Introduction Part

What is Artificial Intelligence

Artificial Intelligence (AI) is composed of two words: **Artificial** and **Intelligence**.

- **Artificial** means man-made.
- **Intelligence** refers to the ability to learn, reason, and solve problems.

Artificial Intelligence is a branch of computer science that focuses on creating machines that can **think, learn, and make decisions like humans**.

AI systems demonstrate capabilities such as:

- Learning from experience
- Logical reasoning
- Problem solving
- Decision making

The History of Artificial Intelligence

Artificial Intelligence (AI) has evolved through several important stages from early theoretical ideas to modern intelligent systems. The development of AI began in the mid-20th century with research in computing, mathematics, and neuroscience that suggested machines could simulate human intelligence.

1. Maturation of Artificial Intelligence (1943–1952)

This period laid the **foundation for AI research** through early theoretical work.

1943 – Artificial Neurons

Warren McCulloch and Walter Pitts proposed the first mathematical model of **artificial neurons**. Their research demonstrated how neurons in the brain could be represented using logical circuits, forming the basis for artificial neural networks.

1949 – Hebbian Learning

Donald Hebb introduced a learning rule for neural networks that describes how the connection strength between neurons can be modified.
This concept, known as **Hebbian Learning**, explains that neurons that fire together strengthen their connections.

1950 – Turing Test

Alan Turing published the famous paper **"Computing Machinery and Intelligence."**
In this paper he proposed the **Turing Test**, which evaluates whether a machine can demonstrate intelligent behavior similar to a human.

2. Birth of Artificial Intelligence (1952–1956)

During this period AI became recognized as a **formal research field**.

1955 – Logic Theorist

Allen Newell and Herbert A. Simon developed the **Logic Theorist**, considered one of the first AI programs. The program proved **38 of 52 mathematical theorems** from *Principia Mathematica* and discovered new proofs.

1956 – Dartmouth Conference

The term **Artificial Intelligence** was first introduced by John McCarthy at the Dartmouth Summer Research Project.
This conference is widely considered the **founding event of AI as an academic discipline**.

During this period, new programming languages such as **FORTRAN, LISP, and COBOL** were developed, which supported AI research.

3. The Golden Years – Early Enthusiasm (1956–1974)

After the Dartmouth conference, AI research expanded rapidly, and researchers became optimistic about the future of intelligent machines.

1966 – ELIZA Chatbot

Joseph Weizenbaum created **ELIZA**, one of the earliest natural language processing programs. ELIZA simulated conversation with users using pattern-matching techniques and became one of the first

chatbots.

1972 – WABOT-1

Japan developed **WABOT-1**, the first intelligent humanoid robot.
It was capable of basic communication and movement, demonstrating the integration of AI with robotics.

4. First AI Winter (1974–1980)

Between 1974 and 1980, AI research experienced a slowdown known as the **AI Winter**.

During this period:

- Funding for AI projects decreased.
- Many AI systems failed to meet expectations.
- Public interest in AI research declined.

This period showed that creating human-level intelligence was far more complex than initially expected.

5. Boom of Artificial Intelligence (1980–1987)

AI research regained momentum with the development of **Expert Systems**.

1980 – Expert Systems

Expert systems were computer programs designed to mimic the decision-making ability of human experts in specific domains such as medicine and engineering.

1980 – AAAI Conference

The first national conference of the **American Association for Artificial Intelligence (AAAI)** was held at Stanford University, bringing together researchers from around the world.

6. Emergence of Intelligent Agents (1993–2011)

This period marked the rapid growth of AI applications and intelligent systems.

1997 – Deep Blue

IBM developed **Deep Blue**, a computer that defeated world chess champion **Garry Kasparov**, demonstrating the power of AI in complex strategic games.

2002 – Roomba

AI entered households through **Roomba**, an intelligent robotic vacuum cleaner capable of automatically navigating rooms.

2006 – AI in Business

Companies such as **Facebook and Twitter** began using AI technologies to analyze large volumes of data, recommend content, and improve user experience.

State of the Art in Artificial Intelligence

Artificial Intelligence has become an essential technology in modern society. It is used in many industries to solve complex problems, automate tasks, and improve efficiency. AI systems can analyze large datasets, identify

patterns, and make intelligent decisions in various fields such as healthcare, finance, agriculture, and education.

1. AI in Astronomy

Artificial Intelligence plays an important role in astronomy by helping scientists analyze huge amounts of space data.

AI algorithms are used to detect and classify celestial objects such as galaxies, stars, and planets. AI also helps researchers discover exoplanets, analyze telescope images, and predict astronomical events. These technologies allow scientists to better understand the structure and origin of the universe.

2. AI in Healthcare

Artificial Intelligence has significantly improved the healthcare industry in recent years.

AI systems help doctors perform faster and more accurate diagnoses by analyzing medical images such as X-rays, MRI scans, and CT scans. AI can also monitor patient health, predict disease progression, and assist in drug discovery. These technologies improve treatment outcomes and reduce medical errors.

3. AI in Gaming

Artificial Intelligence is widely used in the gaming industry to create intelligent and interactive gameplay.

AI systems can control non-player characters (NPCs), analyze player behavior, and generate dynamic game environments. AI also enables machines to play strategic games such as chess and Go by evaluating large numbers of possible moves.

4. AI in Finance

Artificial Intelligence has become very important in the financial sector.

Banks and financial institutions use AI for fraud detection, algorithmic trading, credit scoring, and risk management. AI systems analyze financial transactions and detect unusual patterns that may indicate fraudulent activities. AI also supports automated customer service through chatbots and virtual assistants.

5. AI in Data Security

Data security has become a major concern due to the increasing number of cyber-attacks.

AI systems help detect and prevent cyber threats by analyzing network activity and identifying suspicious patterns. AI tools can automatically detect software vulnerabilities, malware attacks, and security breaches, helping organizations protect sensitive information.

6. AI in Social Media

Social media platforms generate massive amounts of user data every day.

Artificial Intelligence helps organize and analyze this data efficiently. AI algorithms identify trending topics, hashtags, and user preferences. Platforms such as Facebook, Twitter, and Instagram use AI for content recommendation, targeted advertising, and automated moderation of harmful content.

7. AI in Travel and Transport

Artificial Intelligence is transforming the travel and transportation industry.

AI systems help optimize travel routes, manage traffic, and provide personalized travel recommendations. Travel companies use AI chatbots to assist customers with booking flights, hotels, and transportation services. AI is also used in autonomous vehicles and smart transportation systems.

8. AI in Automotive Industry

The automotive industry uses Artificial Intelligence to develop advanced driver-assistance systems and self-driving cars.

AI technologies enable vehicles to detect obstacles, recognize traffic signs, and make driving decisions automatically. Many automobile companies are developing autonomous vehicles that aim to improve road safety and reduce human errors.

9. AI in Robotics

Artificial Intelligence has a significant role in robotics.

Traditional robots are programmed to perform repetitive tasks, but AI-powered robots can learn from experience and adapt to new situations. Intelligent robots are used in manufacturing, healthcare, and service industries. Examples of humanoid robots include Sophia and Erica, which can interact with humans and perform complex tasks.

10. AI in Entertainment

Artificial Intelligence is widely used in entertainment platforms.

Streaming services such as Netflix and Amazon Prime use AI algorithms to recommend movies and television shows based on users' preferences. AI is also used in music recommendation systems, content generation, and video game development.

11. AI in Agriculture

Artificial Intelligence is transforming modern agriculture.

AI technologies such as drones, sensors, and machine learning algorithms help monitor crop health, soil conditions, and weather patterns. AI can also predict crop yields, optimize irrigation, and detect plant diseases early, helping farmers improve productivity.

12. AI in E-Commerce

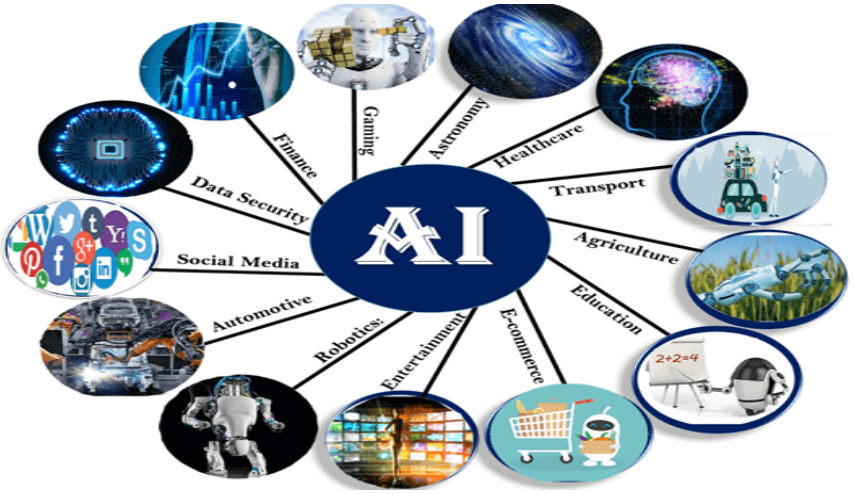
Artificial Intelligence provides a competitive advantage to online businesses.

E-commerce platforms use AI to recommend products, analyze customer behavior, and personalize shopping experiences. AI chatbots assist customers with product inquiries and support services.

13. AI in Education

Artificial Intelligence is increasingly used in the education sector.

AI systems can automate grading, track student performance, and provide personalized learning experiences. AI-based virtual tutors and chatbots can assist students anytime and anywhere, improving accessibility and learning outcomes.



What are Agents?

An **agent** is anything that perceives its environment through **sensors** and acts upon that environment through **actuators/effectors**. In AI, the basic idea is that an agent continuously follows a cycle of **perceiving, processing or thinking, and acting**. This agent model is a standard foundation in AI.

Definition of an Agent

An agent may be a human being, a robot, or a software system. The common feature is that it receives information from its environment, interprets that information, and then performs an action. Russell and Norvig describe an agent as something that perceives and acts in an environment, while modern AI references also describe AI agents as systems that can make decisions and perform tasks autonomously.

Types of Agents

1. Human Agent

A **human agent** perceives the environment through organs such as the **eyes, ears, nose, skin, and tongue**, which act as sensors. Human actions are carried out through **hands, legs, and the vocal tract**, which function as effectors or actuators. This makes a person a natural example of an intelligent agent interacting with the world.

2. Robotic Agent

A **robotic agent** uses devices such as **cameras, infrared range finders, microphones, or other sensors** to observe its surroundings. It performs actions using **motors, wheels, arms, grippers, and mechanical parts**. Robotic agents are widely used in industries, healthcare, and automation systems.

3. Software Agent

A **software agent** works in a digital environment. It may take **keystrokes, file contents, API inputs, database records, or user commands** as sensory input and produce output on a screen, send messages, or perform automated actions in software systems. In AI literature, software agents are often described as agents whose percepts and actions are encoded digitally.

Sensors, Actuators and Effectors

Before studying agents in detail, it is important to understand the related terms **sensor**, **actuator**, and **effector**.

Sensor

A **sensor** is a device or input mechanism that detects changes in the environment and provides information to the agent. Through sensors, an agent becomes aware of its surroundings. In physical systems, sensors may include cameras, microphones, thermometers, or infrared devices. In software systems, sensors may include user input, files, APIs, or digital signals.

Actuator

An **actuator** is a component that enables the agent to take action or produce movement in response to its decisions. In physical systems, actuators may include **electric motors, gears, rails, steering systems, or robotic arms**. In software agents, actuators may include APIs, commands, scripts, or output systems that execute actions.

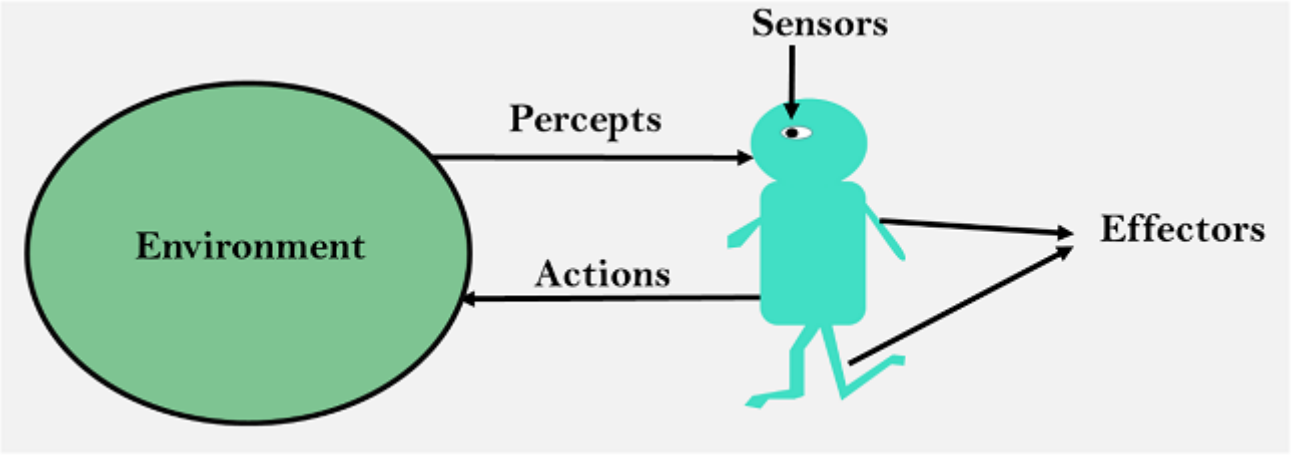
Effectors

Effectors are the parts of the agent that actually affect the environment. Examples include **legs, wheels, arms, fingers, wings, fins, and display screens**. In many AI textbooks, physical devices such as wheels or arms are described as the means by which an agent carries out actions in the environment.

Agent Functioning Cycle

The working of an agent can be summarized in three major steps:

1. **Perceive** the environment using sensors
2. **Think or process** the received information
3. **Act** on the environment using actuators or effectors



Main Rules for an AI Agent

An AI agent is expected to interact intelligently with its environment. For an agent to behave properly, it should follow some basic principles. In AI, an agent perceives the environment through sensors, processes the received information, and acts through actuators. This perceive–think–act framework is the basis of intelligent-agent design.

Four Main Rules of an AI Agent

Rule 1: Ability to Perceive the Environment

An AI agent must have the ability to **perceive** its environment. This means the agent should collect information about the surrounding world through suitable sensors or input mechanisms. Without perception, the agent cannot

understand the current situation and therefore cannot act intelligently.

Rule 2: Observations Must Be Used for Decision Making

The observations gathered from the environment should be **analyzed and used for decision making**. The agent should not merely collect information; it must interpret that information in order to determine the most suitable action.

Rule 3: Decision Should Lead to an Action

After making a decision, the AI agent must convert that decision into an **action**. The purpose of intelligence is not only to think, but also to respond appropriately to the environment. Thus, action is the outcome of perception and decision making.

Rule 4: The Action Must Be Rational

The action selected by the agent should be a **rational action**. A rational action is the one that is expected to maximize the agent's performance measure based on the available percepts and knowledge. In simple terms, the agent should try to do the right thing under the given circumstances.

Agent Environment in Artificial Intelligence

An **environment** is everything that surrounds the agent, but it is **not a part of the agent itself**. It is the external world in which the agent exists, operates, and performs actions. The environment provides the agent with conditions to observe and respond to. In AI, the environment is important because the behavior of the agent depends on the type of environment in which it works.

The environment is the place where the agent:

- receives percepts,
- performs actions,
- and tries to achieve its goals.

Different environments may be fully observable or partially observable, deterministic or stochastic, static or dynamic, and these properties affect how the agent is designed.

Good Behaviour: The Concept of Rationality

In Artificial Intelligence, **good behaviour** means **rational behaviour**. A rational agent is one that chooses actions expected to produce the best outcome according to a defined performance measure. Russell and Norvig define a rational agent as one that, for each possible percept sequence, selects the action expected to maximize its performance measure based on the percept sequence and its built-in knowledge.

A rational agent is said to **do the right thing**. This does not mean the agent is always perfect or always successful. It means the agent selects the best possible action using:

- the information it has perceived,
- the knowledge it already possesses,
- and the set of actions available to it.

In many AI systems, especially in **reinforcement learning**, rational action is very important. The agent receives a **positive reward** for good actions and a **negative reward** for bad actions. Over time, the agent learns to choose actions that improve its performance. This is one practical way to understand rational behavior in AI. This is an

application-oriented explanation; the textbook definition of rationality remains the performance-maximizing one.

Factors Used to Judge Rationality

The rationality of an agent is measured using its **performance measure**. Whether an agent is rational can be judged on the basis of the following points:

1. Performance Measure

The **performance measure** defines the success criterion of the agent. It tells us how well the agent is performing in its environment.

2. Agent's Prior Knowledge of the Environment

The agent may already have some built-in or learned knowledge about the environment. This knowledge helps the agent choose better actions.

3. Best Possible Actions Available

The set of actions an agent can perform also affects rationality. A rational choice can only be made from the actions actually available to the agent.

4. Sequence of Percepts

The **percept sequence** means the complete history of what the agent has perceived so far. Rational action depends not only on the current percept, but often on the entire percept history.

Example: Performance Measure of a Vacuum-Cleaner Agent

A common example used in AI is the **vacuum-cleaner agent**. The success of this agent can be evaluated using a performance measure such as:

- amount of dirt cleaned up,
- amount of time taken,
- amount of electricity consumed,
- amount of noise generated.

A rational vacuum-cleaner agent should try to clean as much dirt as possible while using less time, less electricity, and producing less noise. This example shows how rationality depends on clearly defined performance criteria.

The Nature of Environments (or) Types of Environments

In Artificial Intelligence, the **environment** is the external world in which an agent operates. The design and behavior of an intelligent agent depend greatly on the type of environment in which it functions. AI environments can be classified in several ways based on how much the agent can observe, how the world changes, and whether other agents are present. These standard environment properties are described in AI textbooks such as Russell and Norvig.

Major Types of Environments

From the point of view of an agent, an environment can be classified as:

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

A **fully observable environment** is one in which the agent's sensors can access the complete state of the environment at each point in time. In such environments, the agent has all the information needed to choose an action, so there is usually no need to maintain an internal memory of hidden aspects of the world.

A **partially observable environment** is one in which the agent cannot perceive the complete state of the environment. Some information may be hidden, missing, or noisy, so the agent often needs memory or inference to act effectively. AI texts also note that if an agent has no sensors at all, the environment is effectively **unobservable** to that agent.

2. Deterministic vs Stochastic

A **deterministic environment** is one in which the next state of the environment is completely determined by the current state and the action chosen by the agent. There is no randomness in the outcome. In a deterministic and fully observable environment, the agent does not need to worry much about uncertainty.

A **stochastic environment** is one in which the next state cannot be predicted with complete certainty because randomness or uncertainty is involved. Even if the same action is performed in the same situation, the outcome may differ.

3. Episodic vs Sequential

In an **episodic environment**, the agent's experience is divided into separate episodes. Each action depends only on the current percept, and the current decision does not affect future decisions. Such environments are generally easier to handle.

In a **sequential environment**, the current decision affects future states and future actions. Therefore, the agent must consider past actions and possible future consequences before acting. Such environments usually require memory, planning, or prediction.

4. Single-Agent vs Multi-Agent

A **single-agent environment** contains only one agent operating by itself. The performance of the agent depends mainly on its own actions

A **multi-agent environment** contains more than one agent. In such environments, the actions of one agent may affect the performance of others. The design problems in multi-agent environments are different because agents may need to cooperate, compete, or coordinate with one another.

5. Static vs Dynamic

A **static environment** is one that does not change while the agent is making a decision. Such environments are

comparatively easy to deal with because the agent does not need to keep checking for changes during deliberation.

A **dynamic environment** is one that can change while the agent is thinking or acting. In this case, the agent must continuously observe the environment and update its decisions accordingly. A standard example is **taxi driving**, which is dynamic because the world changes continuously. A **crossword puzzle** is often used as an example of a static environment.

6. Discrete vs Continuous

A **discrete environment** is one in which there are a finite number of distinct percepts, states, or actions. Problems such as a **chess game** are considered discrete because the number of legal moves is limited and clearly defined at each step.

A **continuous environment** is one in which percepts, states, or actions can vary smoothly over a range of values. A **self-driving car** operates in a continuous environment because position, speed, steering angle, and time can vary continuously.

7. Known vs Unknown

Known and **unknown** do not strictly describe the environment itself; rather, they describe the agent's knowledge about how the environment works. In a **known environment**, the agent knows the outcomes or rules associated with its actions.

In an **unknown environment**, the agent does not fully know how the environment behaves and must learn or explore in order to act effectively. It is possible for an environment to be **known but partially observable**, or **unknown but fully observable**, depending on what the agent knows.

8. Accessible vs Inaccessible

An **accessible environment** is one in which the agent can obtain complete, accurate, and up-to-date information about the state of the environment. For example, an **empty room** whose condition can be described by a measurable variable such as temperature can be treated as an accessible environment in simplified AI examples. (byjus.com)

An **inaccessible environment** is one in which the agent cannot obtain complete and accurate information about the environment's state. In such cases, the agent must act with incomplete knowledge. Your example of limited information about a remote event fits the basic idea of inaccessibility, though standard AI texts more commonly discuss accessibility in terms of whether the agent can sense all relevant state information.

Structure of Agents

In Artificial Intelligence, an **agent** is an entity that perceives its environment through sensors and acts upon that environment through actuators. The structure of an agent explains how the agent receives information, processes it, and produces an action. In AI, the standard view is that an agent program implements the agent function, and an agent itself can be understood as **architecture + program**.

Basic Structure of an Agent

The general structure of an agent consists of the following parts:

1. Sensors

Sensors are used by the agent to observe or perceive the environment. They collect information from the surroundings and send it to the agent for processing.

Examples:

- Human agent: eyes, ears, nose, skin
- Robotic agent: cameras, microphones, infrared sensors
- Software agent: keyboard input, file contents, API data

2. Agent Program

The agent program is the decision-making part of the system. It takes the percepts received through sensors, processes them, and decides what action should be taken. The agent program implements the logic of the intelligent system.

3. Architecture

Architecture refers to the physical or software platform on which the agent program runs. The program alone is not sufficient; it needs some architecture to execute its instructions.

Examples:

- A robot body for a robotic agent
- A computer system for a software agent
- A human body for a human agent

4. Actuators / Effectors

Actuators or effectors are the components through which the agent acts upon the environment. They carry out the decisions made by the agent program.

Examples:

- Human agent: hands, legs, vocal tract
- Robotic agent: wheels, arms, motors, grippers
- Software agent: display output, commands, notifications, API calls

Agent = Architecture + Program

A common AI expression is:

Agent = Architecture + Program

This means:

- the **architecture** provides the platform,
- and the **program** provides the intelligence or behavior.

For example, a robotic vacuum cleaner becomes an intelligent agent only when its program runs on its robotic hardware and uses sensor data to choose actions.

Working of the Agent Structure

The structure of an agent works in the following sequence:

1. The agent **perceives** the environment through sensors.
2. The percepts are processed by the **agent program**.

3. The program decides the best possible action.
4. The action is executed through **actuators/effectors**.
5. The environment changes, and the cycle repeats.

This continuous cycle makes the agent interactive and intelligent.

Types of Agent Structures

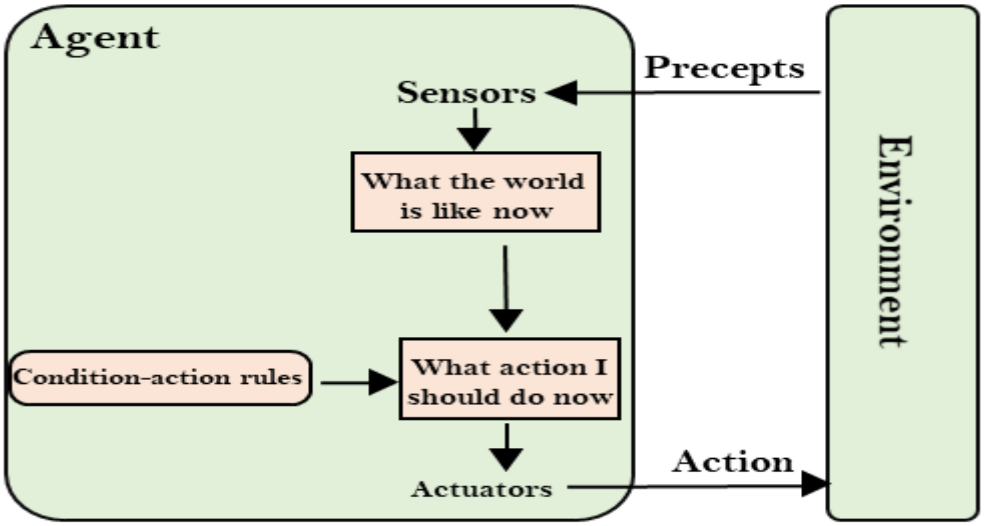
Agents can be grouped into five major classes based on their intelligence and capability. Standard AI sources describe these as increasing in generality from simple reflex behavior to learning behavior.

1. Simple Reflex Agent

The **simple reflex agent** is the simplest type of agent. It selects actions based only on the **current percept** and ignores the rest of the percept history. It works using **condition–action rules**, meaning that if a certain condition is observed, a corresponding action is performed. This type of agent is suitable mainly for **fully observable environments**.

Characteristics:

- Depends only on current input
- Easy to design and implement
- Limited intelligence
- Not suitable for partially observable environments



Example:

A vacuum cleaner agent that sucks dirt when it detects a dirty room.

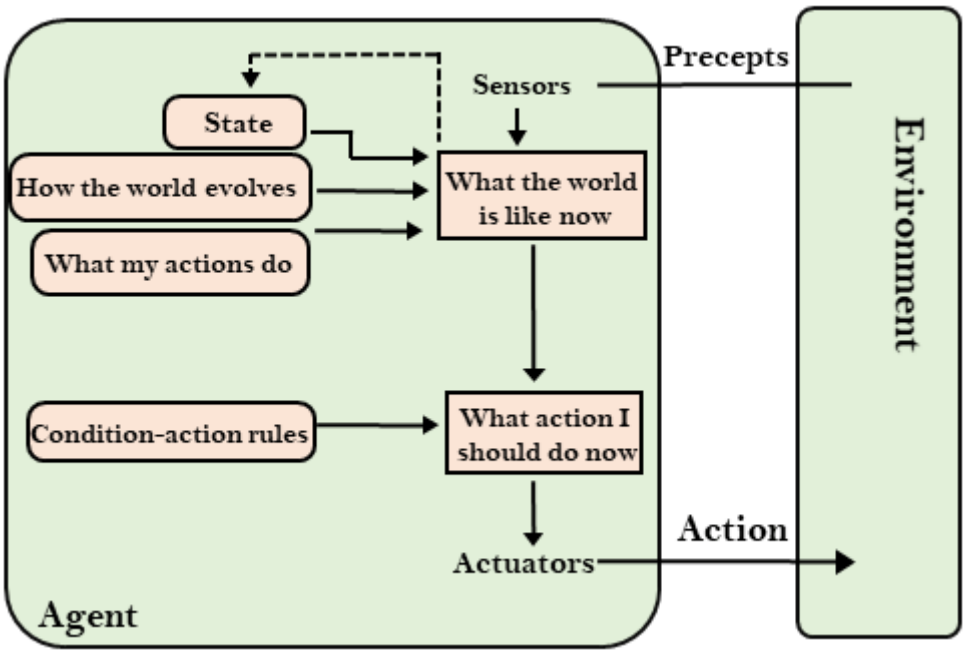
2. Model-Based Reflex Agent

A **model-based reflex agent** improves upon the simple reflex agent by maintaining an **internal state**. This internal state helps the agent keep track of the parts of the environment that cannot be directly observed at the current moment. To update this internal state, the agent needs a **model of the world**, including how the world changes and how its actions affect the world.

Characteristics:

- Maintains internal memory of the environment

- Can work in partially observable environments
- Uses a world model for better decisions

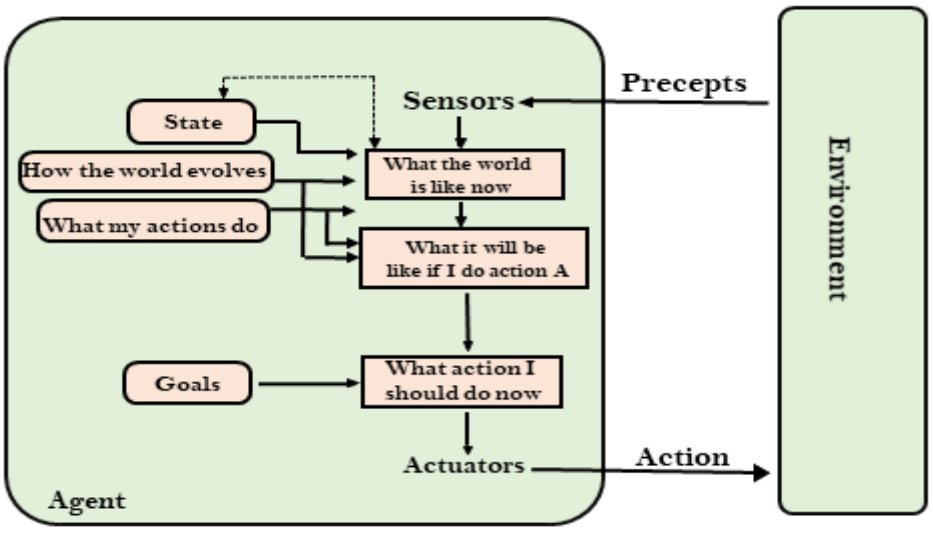


3. Goal-Based Agent

A **goal-based agent** not only knows the current state of the environment but also has information about its **goals**. It chooses actions that help it achieve a desired target state. These agents often require **searching and planning** to determine which action sequence will lead to the goal.

Characteristics:

- Uses goals to guide actions
- More flexible than reflex agents
- Requires planning and search



Example:

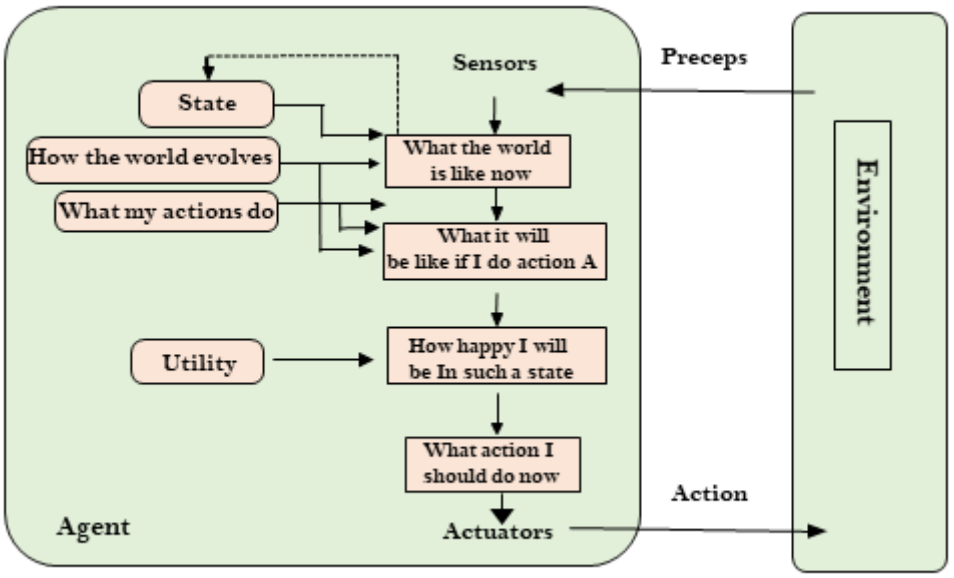
A route-finding system that selects roads to reach a destination.

4. Utility-Based Agent

A **utility-based agent** goes beyond simply achieving a goal. It tries to choose the action that provides the **highest utility**, that is, the best or most desirable outcome among several alternatives. A utility function maps each possible state to a numerical value, allowing the agent to compare options and choose the most beneficial one.

Characteristics:

- Uses a utility function to evaluate states
- Helps choose the best action among many possibilities
- Useful when there are trade-offs or uncertainty



Example:

A self-driving car choosing a route that is not only correct, but also safer and faster.

5. Learning Agent

A **learning agent** is an agent that can improve its performance by learning from past experiences. Standard AI descriptions divide a learning agent into four components: a **learning element**, a **critic**, a **performance element**, and a **problem generator**.

Components of a Learning Agent

a) **Learning Element:** Responsible for improving the agent's knowledge and behavior through experience.

b) Critic : Provides feedback to the learning element about how well the agent is performing according to a fixed standard.

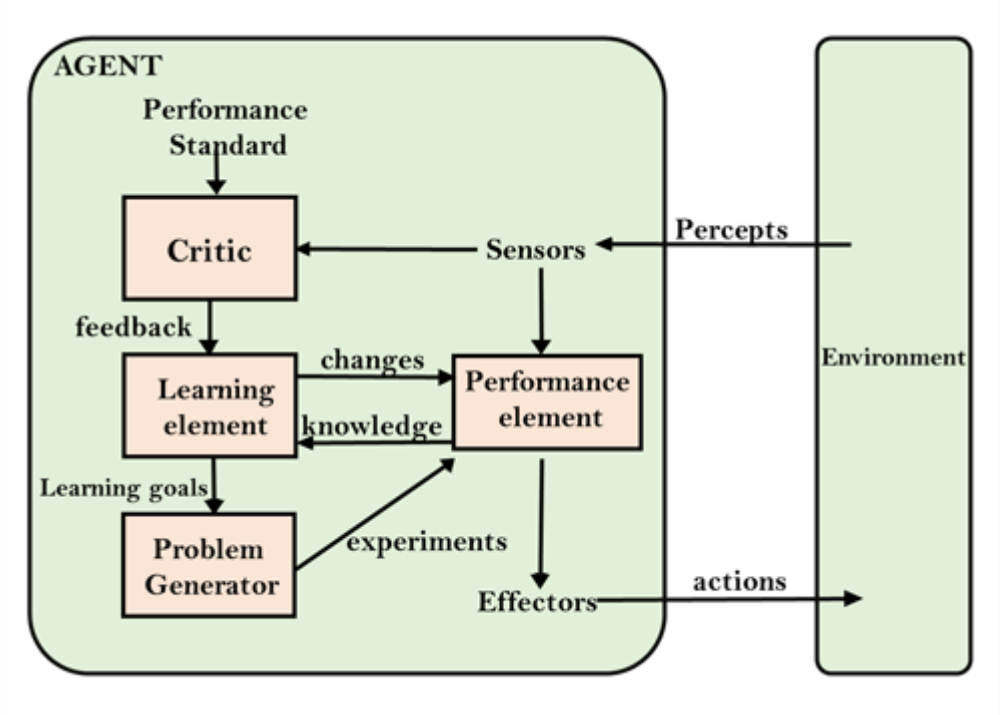
c) Performance Element : Selects external actions and interacts with the environment.

d) Problem Generator : Suggests exploratory actions that may lead to useful new experiences.

Characteristics:

- Learns from past experience
- Adapts to changes in the environment

- Can improve performance over time



Importance of the Structure of Agents

Understanding the structure of agents is important because it forms the foundation of intelligent system design. Different types of agents are suitable for different kinds of environments. Simple agents work in basic situations, while goal-based, utility-based, and learning agents are more suitable for complex and dynamic environments.

Unit Highlights

- AI enables machines to learn, reason, and make decisions.
- An agent perceives through sensors and acts through actuators.
- Rational agents choose actions that maximize performance.
- AI environments can be classified in different types.
- AI has applications in many real-world fields.

Case Study : Google AI for Diabetic Retinopathy Screening

Topic covered: AI in Healthcare, Applications of AI, Rational Decision Making

Google researchers developed AI systems to help detect diabetic retinopathy from retinal images. Google reported that deep learning could help clinicians screen more patients, especially in underserved areas, and later described partnerships to expand screening in real clinical settings.

Why this is important:

This case study shows how AI can support doctors by:

- analyzing medical images,

- identifying signs of disease early,
- improving access to screening,
- helping prioritize patients who need urgent care.

Agent perspective:

The AI system receives medical images as input, processes them using trained models, and produces a diagnostic recommendation. The rational action is the one that best supports accurate screening and timely treatment.

TEXT BOOKS:

1. *Artificial Intelligence A Modern Approach, 2015, Third Edition , Stuart J. Russell and Peter Norvig Pearson Education.*

REFERENCE BOOKS:

1. *Artificial Intelligence, 3/e,2009, Elaine Rich, Kevin Knight and Shiva shankar B Nair Tata McGraw Hill.*
2. *Artificial Intelligence-Structures and Strategies for Complex Problem Solving, 5/e, 2009, George F. Luther, Pearson Education.*
3. *Introduction to Artificial Intelligence, Eugene Charniak and Drew McDermott, Pearson Education, 1987.*

S.No	Questions	Blooms Taxonomy Level
	UNIT I - INTRODUCTION:	
	PART-A (Two Marks Questions)	
1.	Define Artificial Intelligence.	L1
2.	List out the four categories under which AI is classified	L1
3.	Identify the name of driverless car and first computer program to defeat the world champion in chess.	L1
4.	List out the characteristics of Intelligent agent.	L1
5.	How the agents will improve their performance?	L2
6.	What measure will evaluate the behavior of the Agent in an environment.	L2
7.	How to represent the environment of a problem.	L2
	PART-B (Ten Marks Questions)	
1.	What is Artificial Intelligence (AI)? Explain its scope and applications in modern technology.	L2
2.	Describe the history of Artificial Intelligence. How has the field evolved from its inception to the present day?	L3
3.	Discuss the state of the art in Artificial Intelligence. What are the recent advancements and current trends in AI research and applications?	L3
4.	Define an intelligent agent. Explain the relationship between agents and environments with suitable examples	L4
5.	What is meant by rationality in Artificial Intelligence? Discuss the concept of rational behaviour in intelligent agents.	L3
6.	Explain the nature of environments in AI. How do different types of environments affect the behaviour of agents?	L3
7.	Describe the structure of an intelligent agent. What are the key components and how do they work together to achieve intelligent behaviour?	L4

UNIT - II:SOLVING PROBLEMS BY SEARCHING:

“The formulation of goals and the formulation of the problem are the first steps in problem solving.” — Stuart Russell & Peter Norvig

Unit Overview

This unit introduces problem-solving agents and explains how AI systems search for solutions to reach goal states. It covers uninformed and informed search strategies, heuristic functions, and search beyond classical assumptions. It also discusses local search, optimization problems, continuous spaces, nondeterministic actions, and partial observability. Overall, this unit builds the foundation for understanding how intelligent agents solve problems efficiently.

Learning Outcomes

After completing this unit, students will be able to:

- define a problem-solving agent and explain its working
- identify the components of a well-defined problem
- explain example problems used in AI search
- compare uninformed and informed search strategies
- describe the role of heuristic functions in guiding search
- explain local search algorithms and optimization problems
- discuss searching with nondeterministic actions and partial observations

Importance of Studying this Unit

This unit is important because search is one of the core techniques in Artificial Intelligence. It explains how an agent can move from an initial state to a goal state by exploring possible actions. It also provides the base for advanced topics such as planning, robotics, game playing, navigation, and decision-making under uncertainty.

Key Concepts

Problem-solving agent, goal formulation, problem formulation, initial state, successor function, goal test, path cost, search tree, uninformed search, informed search, heuristic function, local search, optimization, continuous space, nondeterministic actions, partial observations, belief state.

Problem-Solving Agents

A problem-solving agent is a goal-based agent that decides what to do by finding a sequence of actions that leads from the current state to a goal state. It first formulates a goal, then formulates the problem, searches for a solution, and finally executes the solution.

Working of a Problem-Solving Agent

A problem-solving agent generally works in four stages:

- goal formulation
- problem formulation
- search for a solution
- execution of the selected action sequence

Components of a Well-Defined Problem

Initial State

The initial state is the starting point from which the agent begins solving the problem.

Successor Function

The successor function defines the possible actions available in a given state and the resulting next states.

Goal Test

The goal test checks whether the current state is the desired goal state.

Path Cost

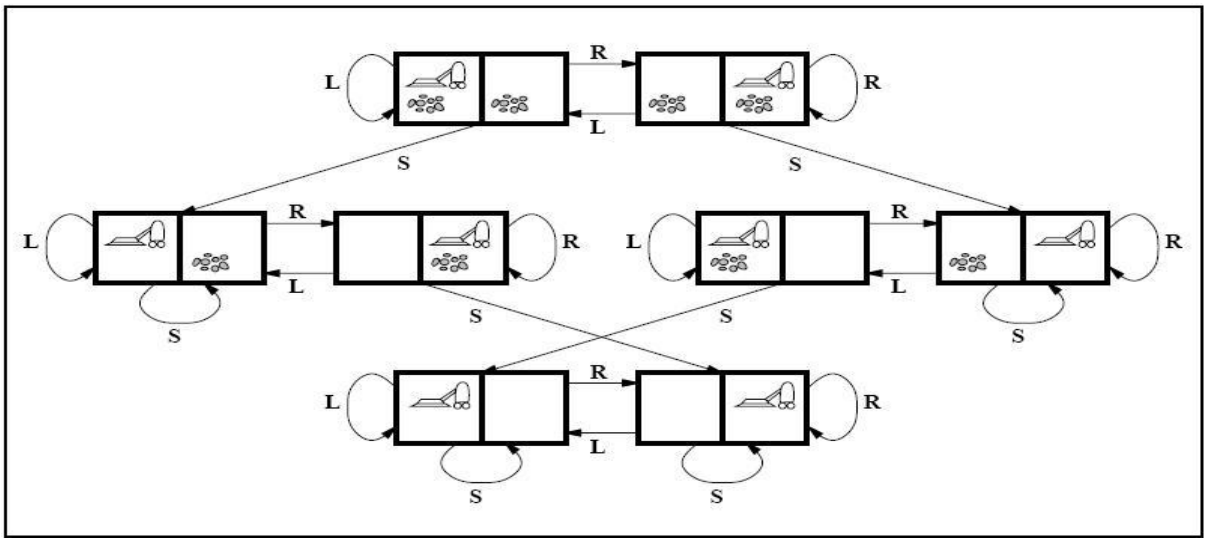
The path cost assigns a numerical value to each path. The best solution is generally the one with the lowest total cost.

Example Problems

vacuum world

This can be formulated as a problem as follows:

- **States:** The state is determined by both the agent location and the dirt locations. The agent is in one of two locations, each of which might or might not contain dirt.
- Thus, there are $2 \times 2^2 = 8$ possible world states.
- A larger environment with n locations has $n*2^n$ states
- **Initial state**: Any state can be designated as the initial state.
- **Actions:** In this simple environment, each state has just three actions: **Left, Right, and Suck.**
- Larger environments might also include Up and Down
- **Transition model:** The actions have their expected effects, except that moving Left in the leftmost square, moving Right in the rightmost square, and Sucking in a clean square have no effect.
- **Goal test:** This checks whether all the squares are clean.
- **Path cost:** Each step costs 1, so the path cost is the number of steps in the path



Arcs denote actions { L, R, S}

states? dirt and agent location

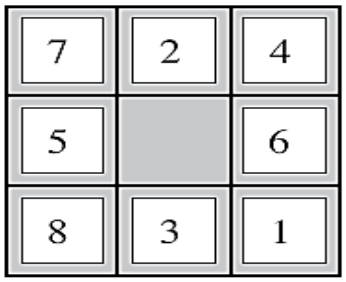
actions? Left, Right, Suck

goal test? no dirt at all locations

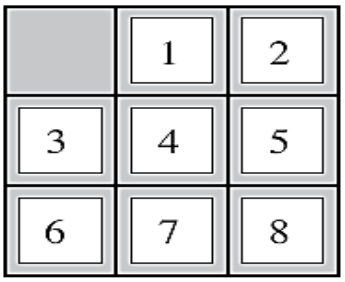
path cost? 1 per action

8-puzzle

- consists of a 3×3 board with eight numbered tiles and a blank space.
- A tile adjacent to the blank space can slide into the space.



Start State



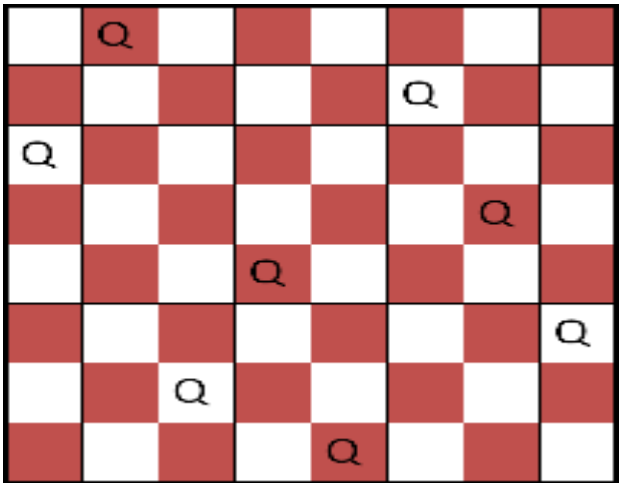
Goal State

- **States:** A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
- **Initial state:** Any state can be designated as the initial state.
- **Actions:** The simplest formulation defines the actions as movements of the blank space Left, Right, Up, or Down.
- **Transition model:** Given a state and action, this returns the resulting state; for example, if we apply Left to the start state in Figure, the resulting state has the 5 and the blank switched.
- **Goal test:** This checks whether the state matches the goal configuration shown in Figure
- **Path cost:** Each step costs 1, so the path cost is the number of steps in the path.

8-queens problem

- The goal of the 8-queens problem is to place eight queens on a chessboard such that no queen attacks any other.
- A queen attacks any piece in the same row, col or diagonal
- Efficient algorithms exist for whole n-queens family

- Still, an interesting test problem for search algorithms



- There are two main kinds of formulation

Incremental formulation

- Starting with an empty state
- For 8-queen problem, each action adds a queen to the state

Complete state formulation

- Starts with all 8 queens on the board
- Moves them around
- In both cases, no path cost because only the final state counts!

The first incremental formulation one might try is the following:

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

Initial state: No queens on the board.

Actions: Add a queen to any empty square.

Transition model: Returns the board with a queen added to the specified square.

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

We have 1.8*1014 possible sequences to investigate.

A better formulation would prohibit placing a queen in any square that is already attacked:

States:

Arrangements of n queens (n= 1 to 8), one per column in the leftmost n columns, with no queen attacking another are states

Actions:

Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.

2057 sequences to investigate

Searching for Solutions

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

- **Search Space:** Search space represents a set of possible solutions, which a system may have.

- **Start State:** It is a state from where agent begins **the search**.
- **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.

Properties of Search Algorithms:

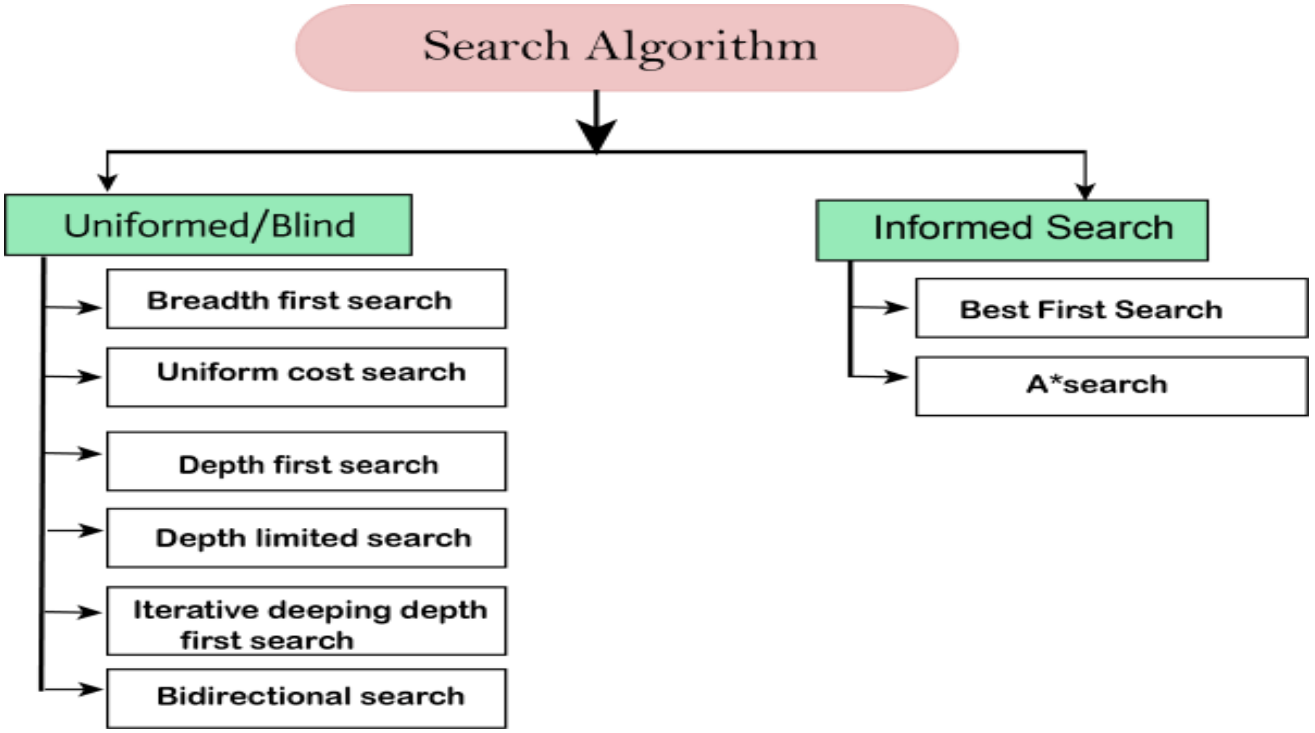
Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:

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

Types of search algorithms

There are two types of search strategies:

- Uninformed Search Strategy (Blind search)
- Informed Search Strategy (Heuristic search)



Uninformed Search Strategy (Blind search)

- The uninformed search does not contain any domain knowledge such as closeness, the location of the goal.
- It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes.

- Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search.
- Various uniformed search strategies are
  - Breadth-first search
  - Uniform cost search
  - Depth-first search
  - Depth Limited Search
  - Iterative deepening depth-first search
  - Bidirectional Search

Breadth-first search

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

Advantages:

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

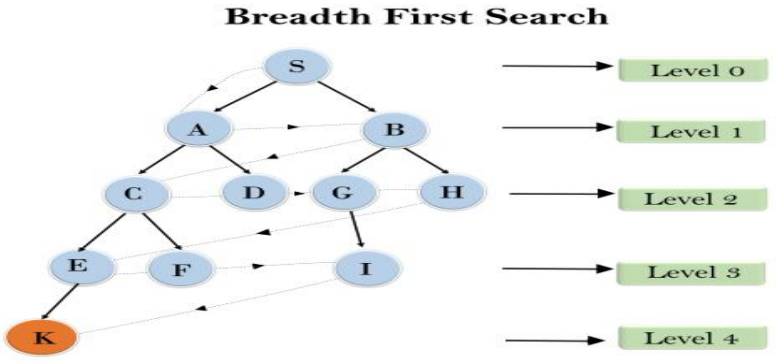
Disadvantages:

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

Example:

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

1. S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K



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

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

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

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

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

Depth-first Search:

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

Advantage:

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

Disadvantage:

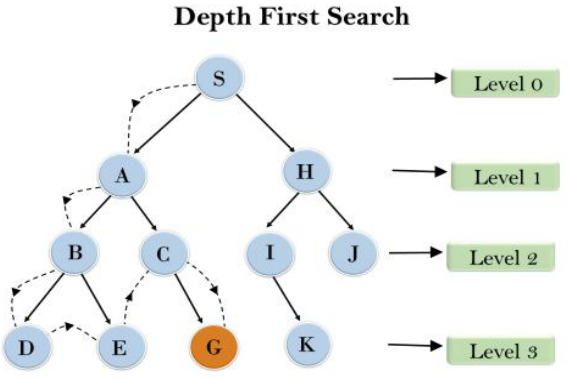
- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

Example:

In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

Root node--->Left node ----> right node.

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



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

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

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

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

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

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

Depth-Limited Search Algorithm:

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

- Standard failure value: It indicates that problem does not have any solution.
- Cutoff failure value: It defines no solution for the problem within a given depth limit.

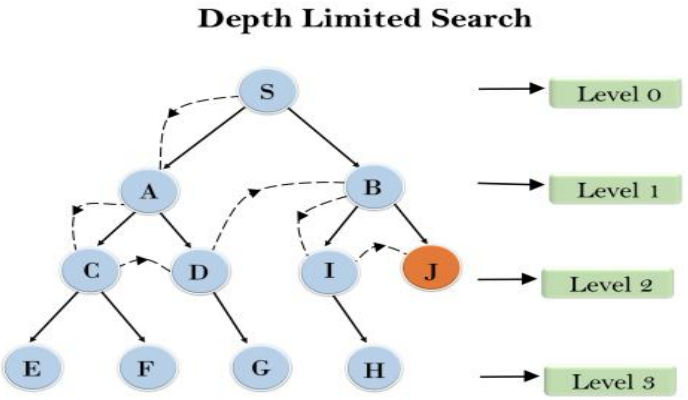
Advantages:

Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

Example:



Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is **O(b^{ℓ})**.

Space Complexity: Space complexity of DLS algorithm is O**(b×ℓ)**.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if ℓ>d.

Uniform-cost Search Algorithm:

- Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph.
- This algorithm comes into play when a different cost is available for each edge.
- The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost.

- Uniform-cost search expands nodes according to their path costs form the root node.
- It can be used to solve any graph/tree where the optimal cost is in demand.
- A uniform-cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost.
- Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

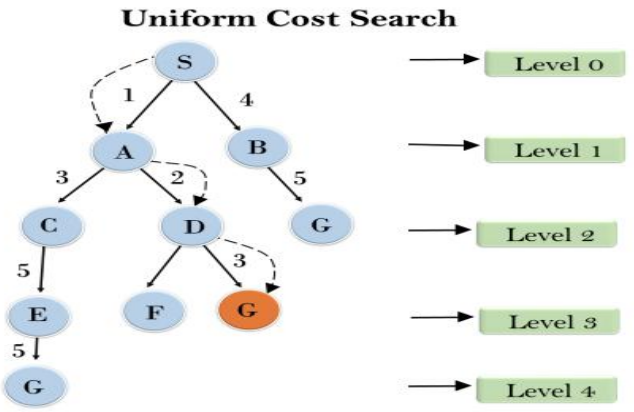
Advantages:

- Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

Example:



Completeness: Uniform-cost search is complete, such as if there is a solution, UCS will find it.
Time Complexity: Let C* **is Cost of the optimal solution**, and **ε** is each step to get closer to the goal node. Then the number of steps is = C*/ε+1. Here we have taken +1, as we start from state 0 and end to C*/ε.

Hence, the worst-case time complexity of Uniform-cost search is$\mathbf{O(b^{1 + \lfloor C^*/\varepsilon \rfloor})}$/.

Space Complexity: The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is $\mathbf{O(b^{1 + \lfloor C^*/\varepsilon \rfloor})}$.
Optimal: Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

Iterative deepening depth-first Search:

- The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.
- This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.
- This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.
- The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Advantages:

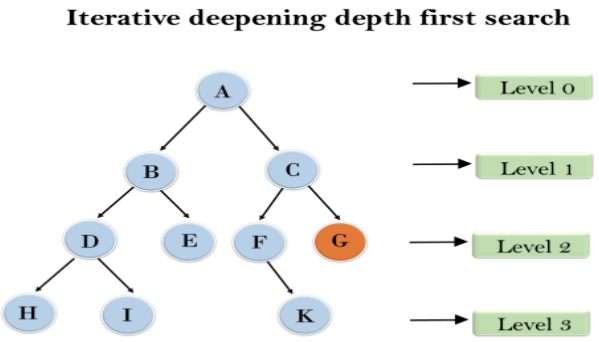
- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

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

Example:

Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:



1'stIteration----->A
2'ndIteration---->A,B,C
3'rdIteration------>A,B,D,E,C,F,G
4'thIteration------>A,B,D,H,I,E,C,F,K,G
In the fourth iteration, the algorithm will find the goal node.

Completeness: This algorithm is complete is if the branching factor is finite.

Time Complexity: Let's suppose b is the branching factor and depth is d then the worst-case time complexity is **$O(b^d)$**.

Space Complexity: The space complexity of IDDFS will be **O(bd)**.

Optimal: IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

Bidirectional Search Algorithm:

- Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node.
- Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex.
- The search stops when these two graphs intersect each other.
- Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Advantages:

- Bidirectional search is fast.
- Bidirectional search requires less memory

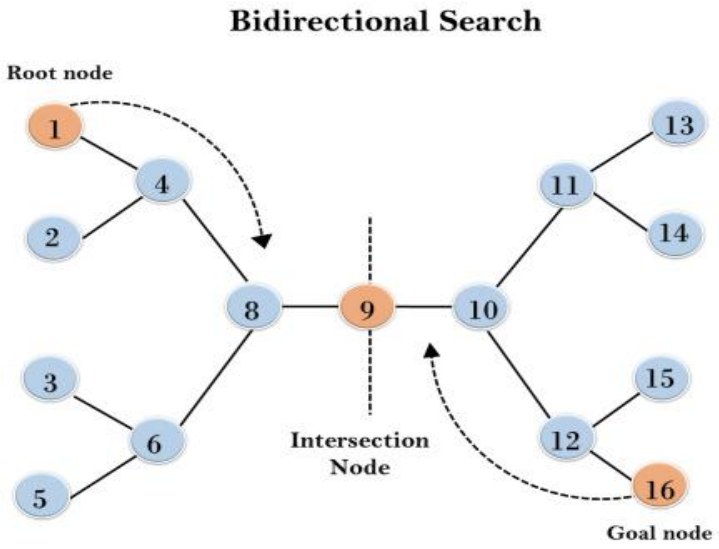
Disadvantages:

- Implementation of the bidirectional search tree is difficult.
- In bidirectional search, one should know the goal state in advance.

Example:

- In the below search tree, bidirectional search algorithm is applied.
- This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.

- The algorithm terminates at node 9 where two searches meet.



Completeness: Bidirectional Search is complete if we use BFS in both searches.

Time Complexity: Time complexity of bidirectional search using BFS is **$O(b^d)$**.

Space Complexity: Space complexity of bidirectional search is **$O(b^d)$**.

Optimal: Bidirectional search is Optimal.

Informed Search Algorithms

- Informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.
- The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called **Heuristic search**.
- Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n).
- It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.
- On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues unit a goal state is found.

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

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

Best-first Search Algorithm (Greedy Search):

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

f(n)= h(n).

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

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

Best first search algorithm:

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

Advantages:

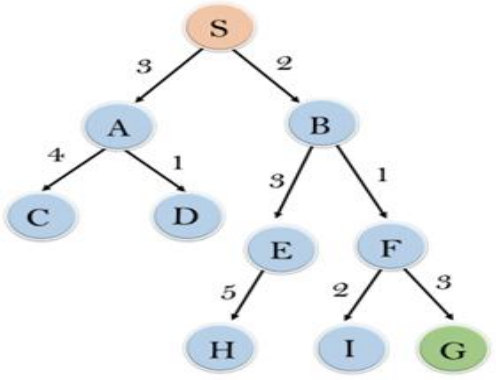
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

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

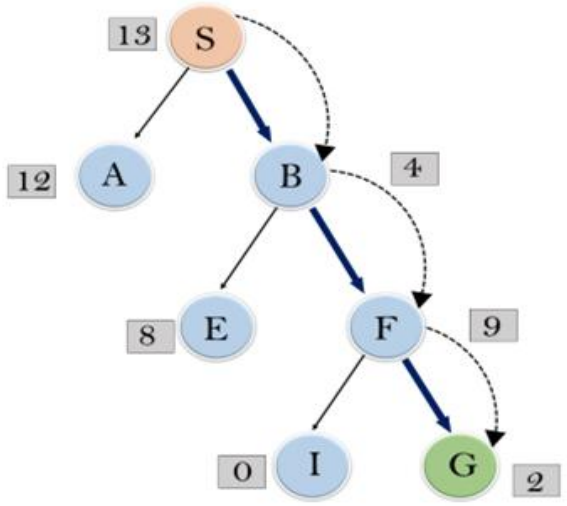
Example:

- Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.



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

- In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.



- **Expand the nodes of S and put in the CLOSED list**
- **Initialization:** Open [A, B], Closed [S]
- **Iteration 1:** Open [A], Closed [S, B]
- **Iteration2:** Open[E,F,A],Closed[S,B]
  : Open [E, A], Closed [S, B, F]
- **Iteration3:** Open[I,G,E,A],Closed[S,B,F]
  : Open [I, E, A], Closed [S, B, F, G]
- Hence the final solution path will be: **S----> B----->F----> G**

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

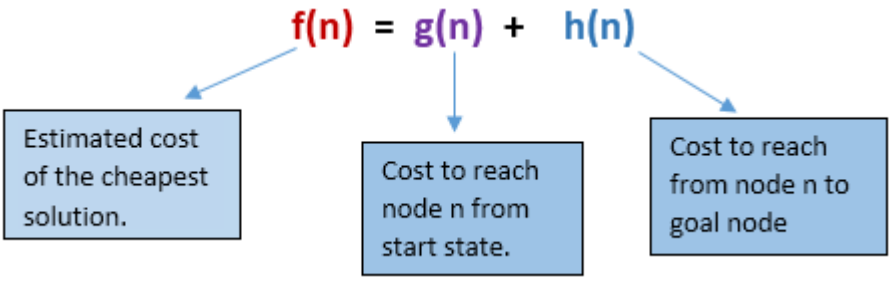
Space Complexity: The worst-case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.

Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

Optimal: Greedy best first search algorithm is not optimal.

A* Search Algorithm:

- A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n).
- It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function.
- This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).
- In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

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

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

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

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

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

Advantages:

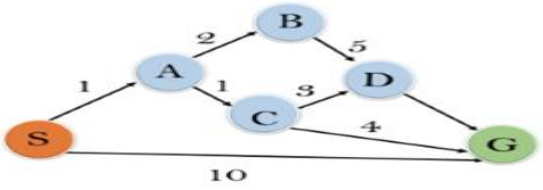
- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

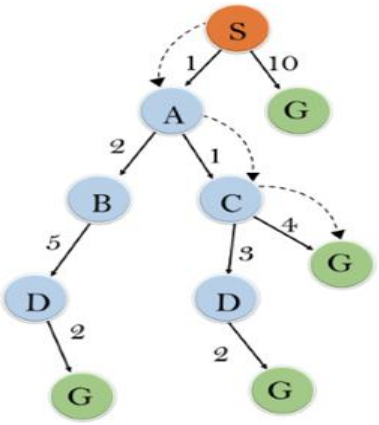
- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Example:

In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.
Here we will use OPEN and CLOSED list.



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



Solution:

Initialization: {(S, 5)}

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

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

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

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

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n)<="" li="">

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

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

Heuristic Functions

Heuristics function: Heuristic is a function which is used in Informed Search, and it finds the most promising path.

It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal.

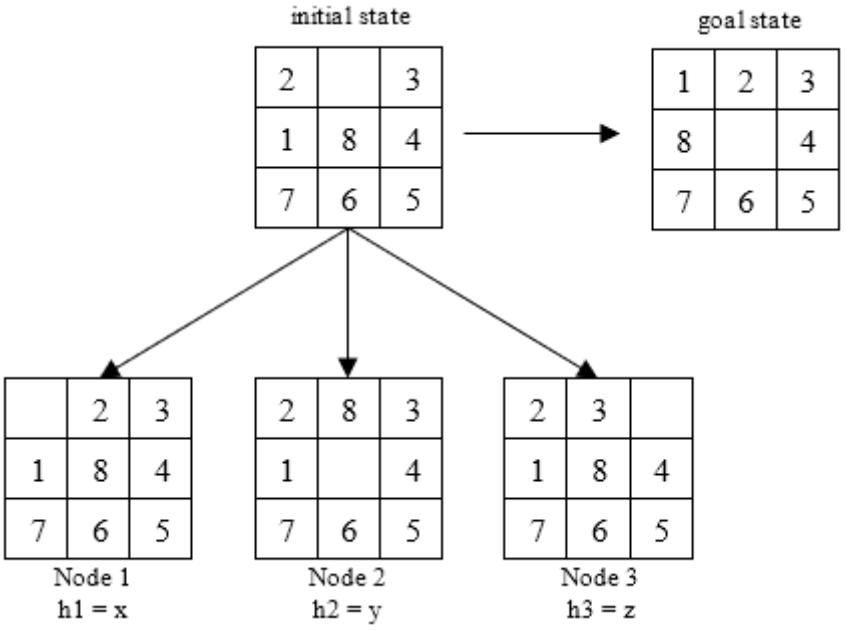
The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time.

Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

Admissibility of the heuristic function is given as:

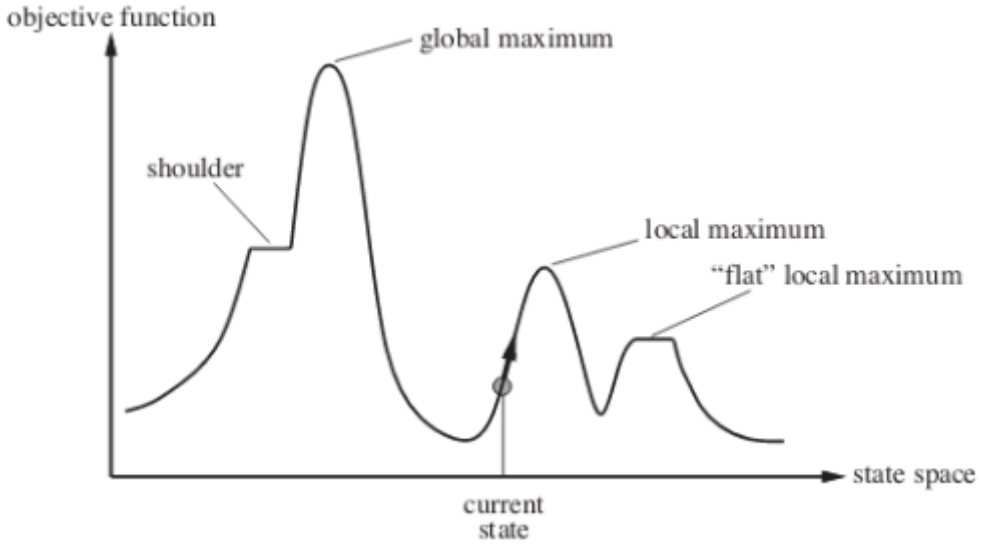
1. h(n) <= h*(n)

Here h(n) is heuristic cost, and h*(n) is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.
we look at heuristics for the 8-puzzle, in order to understand the nature of heuristics in general.
The 8-puzzle was one of the earliest heuristic search problems.
The instance of the puzzle is to slide the tiles horizontally or vertically into the empty space until the configuration matches the goal configuration
The heuristic function should never over estimate the number of steps to the goal
Two commonly used candidates:
h1=the number of misplaced tiles
h2=the sum of the Manhattan distances of the tiles from their goal positions (i.e., no. of squares from desired location of each tile)



Local search algorithms and optimization problems

- The search algorithms that we have seen so far, explores search space systematically. i.e. when a goal state is found, the path to the goal state constitutes the solution
- In many problems, however the path to the goal state is irrelevant.
- Example :8queen problem
- If the path to goal state doesn't matter then, we can use local search algorithms
- To understand local search, we consider the state-space landscape
- A landscape has both "location" (defined by the state) and "elevation" (defined by the value of the heuristic cost function or objective function).
- If elevation corresponds to cost, then the aim is to find the lowest valley—a global minimum
- if elevation corresponds to an objective function, then the aim is to find the highest peak—a global maximum.
- Local search algorithms explore this landscape.
- A complete local search algorithm always finds a goal if one exists;
- an optimal algorithm always finds a global minimum/maximum



Local search algorithms

- Hill-climbing search
- Simulated annealing
- Local beam search
- Genetic algorithms

Hill-climbing search

- It is a local search algorithm which continuously moves in the direction of increasing value to find best solution to the problem (uphill)
- It terminates when it reaches a peak value where no neighbour has a higher value.
- The algorithm does not maintain a search tree, so the data structure for the current node need only to record the state and the value of the objective function.
- Hill climbing does not look ahead beyond the immediate neighbors of the current state
- Hill climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next
- This resembles trying to find the top of Mount Everest in a thick fog while suffering from amnesia

Simulated annealing Algorithm:

- A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum.

- If algorithm applies a random walk, moving to a successor chosen uniformly at random from the set of successors—is complete but extremely inefficient
- Simulated Annealing is an algorithm which yields both efficiency and completeness
- Annealing is a process of hardening a metal or glass to a high temperature then cooling gradually, so this allows the metal to reach a low-energy crystalline state
- The same process is used in simulated annealing in which the algorithm picks a random move, instead of picking the best move. If the random move improves the state, then it follows the same path. Otherwise, the algorithm follows the path which has a probability of less than 1 or it moves downhill and chooses another path

Local beam search

- In local beam search, it keeps track of k-states rather than just 1.
- It begins with k- randomly generated states. At each step all the successors of all K states are generated.
- If anyone is a goal state the algorithm terminated. Otherwise, it selects k-best successors from the list and repeats the same process.
- **Stochastic beam search:** Instead of choosing k-best successor, stochastic beam search selects k-successors randomly.
- A local beam search with k-states is Similar to running k-random restarts in parallel. This is called a local beam search.

Genetic Algorithms:

- it is a variant of stochastic beam search.
- successor are generated by combining two parent states rather than by modifying a single state.
- Like stochastic beam search genetic algorithm begins with k randomly generated states that is known as population
- Ex: consider 8-queens problem each state is represented as a 8 digit string representing position of the queen in a 8x8 board

Case Study : Puzzle Solving

The 8-puzzle is a classic search problem. Different algorithms can be compared by how quickly they find a solution and whether the solution is optimal. Heuristics such as Manhattan distance can greatly improve efficiency.

Unit Highlights

- Problem-solving agents solve tasks by searching for action sequences that reach goals.
- A well-defined problem includes **initial state, successor function, goal test, and path cost**.
- Search strategies are broadly classified into **uninformed** and **informed** methods.
- Heuristic functions guide search and improve efficiency.
- **A*** is one of the most important informed search algorithms.
- Local search is useful for optimization problems and large state spaces.
- Search becomes more realistic and complex in **continuous, nondeterministic, and partially observable** environments.

TEXT BOOKS:

1. *Artificial Intelligence A Modern Approach, 2015, Third Edition , Stuart J. Russell and Peter Norvig Pearson Education.*

REFERENCE BOOKS:

2. *Artificial Intelligence, 3/e,2009, Elaine Rich, Kevin Knight and Shiva shankar B Nair Tata McGraw Hill.*
3. *Artificial Intelligence-Structures and Strategies for Complex Problem Solving, 5/e, 2009, George F. Luther, Pearson Education.*
4. *Introduction to Artificial Intelligence, Eugene Charniak and Drew McDermott, Pearson Education, 1987.*

S.No	Questions	Blooms Taxonomy Level
	UNIT II - SOLVING PROBLEMS BY SEARCHING	
	PART-A (Two Marks Questions)	
1.	List the five components of a problem.	L1
2.	Define search.	L1
3.	Compare the goal formulation and problem formulation.	L2
4.	How to represent the environment of a problem.	L2
5.	List out some of the uninformed search techniques.	L1
6.	List out some of the informed search techniques.	L1
7.	Compare the access method in uninformed search and informed search.	L2
8.	List the criterion used to compare the uninformed search strategies.	L1
9.	Define heuristic function.	L1
10.	What kind of problems can be solvable using local search algorithms.	L3
11.	Compare global minimum and global maximum.	L2
12.	Define Genetic algorithm.	L1
	PART-B (Ten Marks Questions)	
1.	a. Explain about Searching for solutions b. Explain about Heuristic functions.	L2 L2
2.	Describe about Well-defined problems and solutions.	L2
3.	Illustrate the Uninformed Search Strategies with examples.	L3
4.	Illustrate the Informed Search Strategies with examples.	L3
5.	Explain about Local Search Algorithms and Optimization Problems	L2
6.	Explain about local search in continuous spaces.	L3
7.	Discuss about Vacuum world problem	L3
8.	Compare and contrast Breadth-First Search (BFS), Depth-First Search (DFS), Discuss their completeness, optimality, time and space complexity	L5
9.	Describe how the Greedy Best-First Search and A* Search algorithms use heuristic functions	L3

UNIT – III: CONSTRAINT SATISFACTION PROBLEMS

“It is better to solve one problem five different ways, than to solve five problems one way.”

— George Pólya

Unit Overview

This unit explains how AI can solve structured problems using **constraints** and how intelligent behavior can be achieved through **logic-based reasoning**. It covers the definition of CSPs, inference through constraint propagation, and backtracking search. It also introduces knowledge-based agents, the Wumpus World, propositional logic, theorem proving, model checking, and agents based on propositional logic.

Learning Outcomes

After completing this unit, students will be able to:

1. Define a **Constraint Satisfaction Problem** and identify its components.
2. Explain **constraint propagation** and inference in CSPs.
3. Describe **backtracking search** for CSPs.
4. Explain the concept of a **knowledge-based agent**.
5. Describe the **Wumpus World** as an example of logical reasoning.
6. Explain **propositional logic**, theorem proving, and model checking.
7. Discuss how agents can be built using **propositional logic**.

Importance of Studying this Unit

This unit is important because many real-world AI problems can be represented either as **constraint satisfaction tasks** or as **knowledge-based reasoning tasks**. CSP methods are used in scheduling, planning, timetabling, and resource allocation, while logical agents are useful for reasoning, diagnosis, and decision-making in uncertain environments. These topics form a strong conceptual base for advanced AI methods.

Key Concepts

Constraint satisfaction problem, variable, domain, constraint, constraint propagation, arc consistency, backtracking, knowledge base, inference, logical agent, Wumpus World, propositional logic, theorem proving, model checking, entailment, satisfiability.

Constraint Satisfaction Problems (CSP)

- Constraint Satisfaction Problems (CSP) play a crucial role in artificial intelligence (AI) as they help solve 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.

Components of Constraint Satisfaction Problems

Variables: The things that need to be determined are variables. Variables in a CSP are the objects that must have values assigned to them in order to satisfy a particular set of constraints. Boolean, integer, and categorical variables are just a few examples of the various types of variables, for instance, could stand in for the many puzzle cells that need to be filled with numbers in a sudoku puzzle.

Domains: The range of potential values that a variable can have is represented by domains. Depending on the issue, a domain may be finite or limitless. For instance, in Sudoku, the set of numbers from 1 to 9 can serve as the domain of a variable representing a problem cell.

Constraints: The guidelines that control how variables relate to one another are known as constraints. Constraints in a CSP define the ranges of possible values for variables. Unary constraints, binary constraints, and higher-order constraints are only a few examples of the various sorts of constraints. For instance, in a sudoku problem, the restrictions might be that each row, column, and 3×3 box can only have one instance of each number from 1 to 9.

Types of Constraint Satisfaction Problems

CSPs can be classified into different types based on their constraints and problem characteristics:

1. **Binary CSPs:** In these problems, each constraint involves only two variables. For example, in a scheduling problem, the constraint could specify that task A must be completed before task B.
2. **Non-Binary CSPs:** These problems have constraints that involve more than two variables. For instance, in a seating arrangement problem, a constraint could state that three people cannot sit next to each other.
3. **Hard and Soft Constraints:** Hard constraints must be strictly satisfied, while soft constraints can be violated, but at a certain cost. This distinction is often used in real-world applications where not all constraints are equally important.

Representation of Constraint Satisfaction Problems (CSP)

In **Constraint Satisfaction Problems (CSP)**, the solution process involves the interaction of variables, domains, and constraints. Below is a structured representation of how CSP is formulated:

1. **Finite Set of Variables** (V1,V2,…,Vn)($V1,V2,…,Vn$)**:**The problem consists of a set of variables, each of which needs to be assigned a value that satisfies the given constraints.
2. **Non-Empty Domain for Each Variable** (D1,D2,…,Dn)($D1,D2,…,Dn$)**:**Each variable has a domain—a set of possible values that it can take. For example, in a Sudoku puzzle, the domain could be the numbers 1 to 9 for each cell.
3. **Finite Set of Constraints** (C1,C2,…,Cm)($C1,C2,…,Cm$)**:**Constraints restrict the possible values that variables can take. Each constraint defines a rule or relationship between variables.
4. **Constraint Representation:**

   Each constraint Ci is represented as a pair **<scope, relation>**, where:

   - **Scope:** The set of variables involved in the constraint.
   - **Relation:** A list of valid combinations of variable values that satisfy the constraint.
5. **Example:**

   Let's say you have two variables V1$V1$ and V2$V2$. A possible constraint could be V1≠V2$V1 \neq V2$, which means the values assigned to these variables must not be equal.

   - **Detailed Explanation:**

- **Scope:** The variables V1$V1$ and V2$V2$.
- **Relation:** A list of valid value combinations where V1$V1$ is not equal to V2$V2$.

Types of Domains in CSP

There are following two types of domains which are used by the variables :

Discrete Domain: It is an infinite domain which can have one state for multiple variables. For example, a start state can be allocated infinite times for each variable.

Finite Domain: It is a finite domain which can have continuous states describing one domain for one specific variable. It is also called a continuous domain.

Constraint Types in CSP

With respect to the variables, basically there are following types of constraints:

Unary Constraints: It is the simplest type of constraints that restricts the value of a single variable.

Binary Constraints: It is the constraint type which relates two variables. A value x2 will contain a value which lies between x1 and x3.

Global Constraints: It is the constraint type which involves an arbitrary number of variables.

Constraint Propagation: Inference in CSPs

Constraint propagation is a fundamental concept in constraint satisfaction problems (CSPs). A CSP involves variables that must be assigned values from a given domain while satisfying a set of constraints. Constraint propagation aims to simplify these problems by reducing the domains of variables, thereby making the search for solutions more efficient.

How Constraint Propagation Works

Constraint propagation works by iteratively narrowing down the domains of variables based on the constraints. This process continues until no more values can be eliminated from any domain. The primary goal is to reduce the search space and make it easier to find a solution.

Steps in Constraint Propagation

1. **Initialization**: Start with the initial domains of all variables.
2. **Propagation**: Apply constraints to reduce the domains of variables.
3. **Iteration**: Repeat the propagation step until a stable state is reached, where no further reduction is possible.

Example

Consider a simple CSP with two variables, X and Y, each with domains {1, 2, 3}, and a constraint $X \neq Y$. Constraint propagation will iteratively reduce the domains as follows:

- If X is assigned 1, then Y cannot be 1, so Y's domain becomes {2, 3}.
- If Y is then assigned 2, X cannot be 2, so X's domain is reduced to {1, 3}.
- This process continues until a stable state is reached.

Algorithms for Constraint Propagation

- Several algorithms are used for constraint propagation, each with its strengths and weaknesses. Some common algorithms include:

Arc Consistency

- Arc consistency ensures that for every value of one variable, there is a consistent value in another variable connected by a constraint. This algorithm is often used as a preprocessing step to simplify CSPs before applying more complex algorithms.

Path Consistency

- Path consistency extends arc consistency by considering triples of variables. It ensures that for every pair of variables, there is a consistent value in the third variable. This further reduces the domains and simplifies the problem.

k-Consistency

- k-Consistency generalizes the concept of arc and path consistency to k variables. It ensures that for every subset of k-1 variables, there is a consistent value in the kth variable. Higher levels of consistency provide more pruning but are computationally more expensive.

Applications of Constraint Propagation

Constraint propagation is widely used in various AI applications. Some notable areas include:

Scheduling

- In scheduling problems, tasks must be assigned to time slots without conflicts. Constraint propagation helps by reducing the possible time slots for each task based on constraints like availability and dependencies.

Planning

- AI planning involves creating a sequence of actions to achieve a goal. Constraint propagation simplifies the planning process by reducing the possible actions at each step, ensuring that the resulting plan satisfies all constraints.

Resource Allocation

- In resource allocation problems, resources must be assigned to tasks in a way that meets all constraints, such as capacity limits and priority rules. Constraint propagation helps by narrowing down the possible assignments, making the search for an optimal allocation more efficient.

Backtracking search for CSPs

- Backtracking search, a form of depth-first search, is commonly used for solving CSPs. Inference can be interwoven with search.
- Commutativity: CSPs are all commutative. A problem is commutative if the order of application of any given set of actions has no effect on the outcome.
- Backtracking search: A depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.

- Backtracking algorithm repeatedly chooses an unassigned variable, and then tries all values in the domain of that variable in turn, trying to find a solution. If an inconsistency is detected, then BACKTRACK returns failure, causing the previous call to try another value.
- There is no need to supply BACKTRACKING-SEARCH with a domain-specific initial state, action function, transition model, or goal test.
- BACKTRACKING-SARCH keeps only a single representation of a state and alters that representation rather than creating a new ones.

Steps in Backtracking search

Initialization: Start with an empty assignment.

Selection: Choose an unassigned variable.

Assignment: Assign a value to the chosen variable.

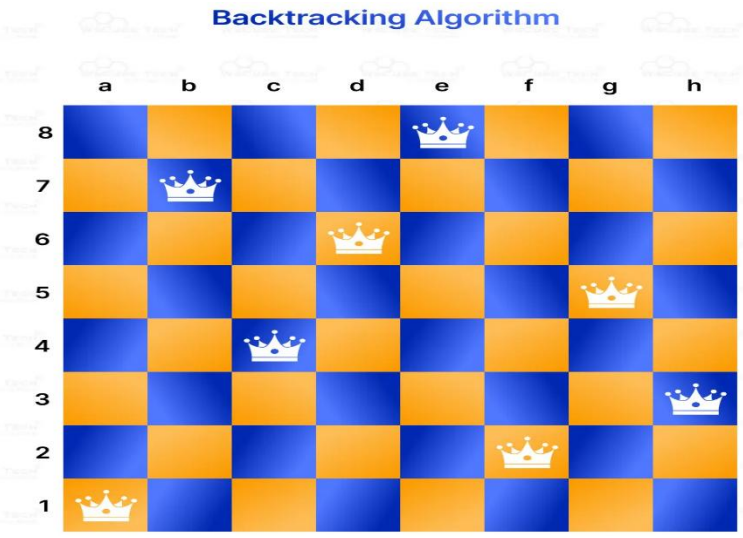
Consistency Check: Check if the current assignment is consistent with the constraints.

Recursion: If the assignment is consistent, recursively try to assign values to the remaining variables.

Backtrack: If the assignment is not consistent, or if further assignments do not lead to a solution, undo the last assignment (backtrack) and try the next possible value.

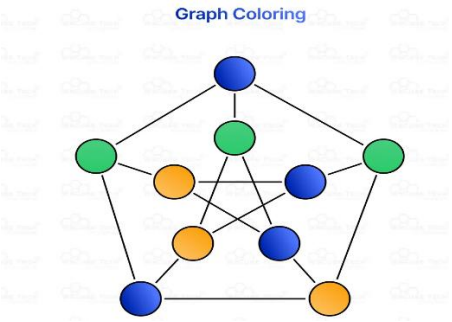
Examples of Backtracking Algorithms

1. **N-Queens Problem**
   - The N queen problem using backtracking algorithm involves placing N queens on an N×N chessboard such that no two queens threaten each other.
   - This means no two queens can share the same row, column, or diagonal.
   - The backtracking algorithm places queens one by one in different rows, checking for conflicts, and backtracking when a conflict is found until all queens are safely placed.



2. **Graph Coloring**
   - In the Graph Coloring problem, the goal is to color the vertices of a graph using the minimum number of colors so that no two adjacent vertices share the same colour.

- The backtracking algorithm assigns colors to vertices one by one, ensuring that each color assignment is valid. If a conflict arises, it backtracks and tries a different color.



Knowledge–Based Agent

An intelligent agent needs knowledge about the real world for taking decisions and reasoning to act efficiently.

Knowledge-based agents are those agents who have the capability of maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently.

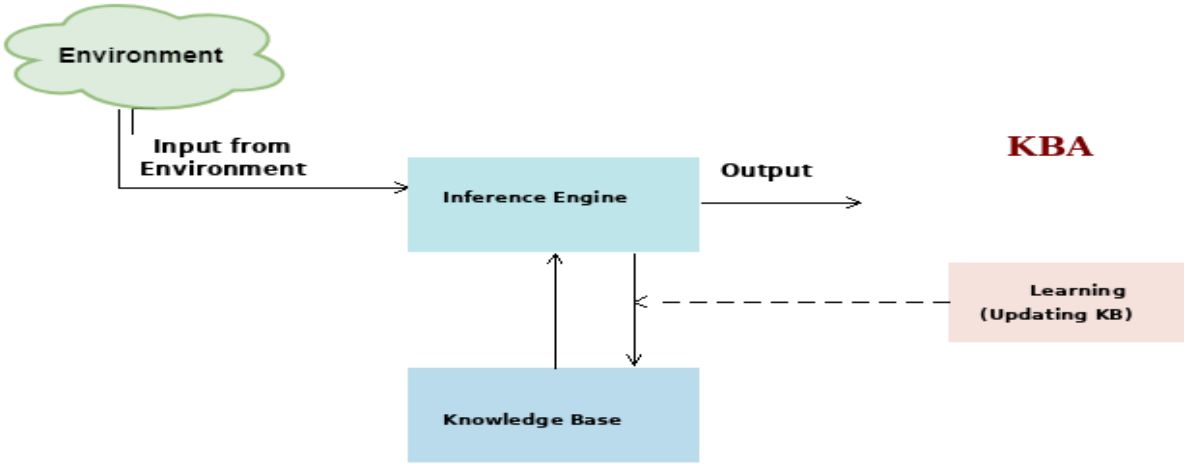
Knowledge-based agents are composed of two main parts:

- **Knowledge-base and**
- **Inference system**.

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

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

The architecture of knowledge-based agent:



The above diagram is representing a generalized architecture for a knowledge-based agent.
The knowledge-based agent (KBA) take input from the environment by perceiving the environment.
The input is taken by the inference engine of the agent and which also communicate with KB to decide as per the knowledge store in KB.
The learning element of KBA regularly updates the KB by learning new knowledge.
Knowledge base: Knowledge-base is a central component of a knowledge-based agent, it is also known as KB. It is a collection of sentences (here 'sentence' is a technical term and it is not identical to sentence in English). These sentences are expressed in a language which is called a knowledge representation language.
The Knowledge-base of KBA stores fact about the world.

Why use a knowledge base?
Knowledge-base is required for updating knowledge for an agent to learn with experiences and take action as per the knowledge

Inference system

Inference means deriving new sentences from old. Inference system allows us to add a new sentence to the knowledge base. A sentence is a proposition about the world. Inference system applies logical rules to the KB to deduce new information.
Inference system generates new facts so that an agent can update the KB. An inference system works mainly in two rules which are given as:

- **Forward chaining**
- **Backward chaining**

Operations Performed by KBA
Following are three operations which are performed by KBA in order to show the intelligent behavior:

1. **TELL:** This operation tells the knowledge base what it perceives from the environment.
2. **ASK:** This operation asks the knowledge base what action it should perform.
3. **Perform:** It performs the selected action.

Various levels of knowledge-based agent:
A knowledge-based agent can be viewed at different levels which are given below:

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.

Logical level:

At this level, we understand that how the knowledge representation of knowledge is stored. At this level, sentences are encoded into different logics. At the logical level, an encoding of knowledge into logical sentences occurs. At the logical level we can expect to the automated taxi agent to reach to the destination B.

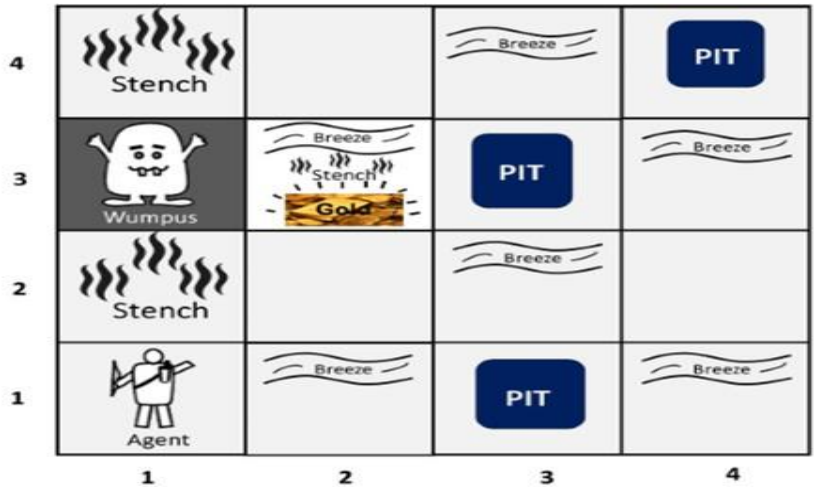
Implementation level:

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.

The Wumpus World in Artificial intelligence

- The Wumpus world is a simple world example to illustrate the worth of a knowledge-based agent and to represent knowledge representation. It was inspired by a video game Hunt the Wumpus by Gregory Yob in 1973.
- The Wumpus world is a cave which has 4/4 rooms connected with passageways. So there are total 16 rooms which are connected with each other.
- We have a knowledge-based agent who will go forward in this world. The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room.
- The Wumpus can be shot by the agent, but the agent has a single arrow. In the Wumpus world, there are some Pits rooms which are bottomless, and if agent falls in Pits, then he will be stuck there forever.
- The exciting thing with this cave is that in one room there is a possibility of finding a heap of gold.
- So the agent goal is to find the gold and climb out the cave without fallen into Pits or eaten by Wumpus.
- The agent will get a reward if he comes out with gold, and he will get a penalty if eaten by Wumpus or falls in the pit.

Following is a sample diagram for representing the Wumpus world. It is showing some rooms with Pits, one room with Wumpus and one agent at (1, 1) square location of the world.



There are also some components which can help the agent to navigate the cave. These components are given as follows:

- The rooms adjacent to the Wumpus room are smelly, so that it would have some stench.
- The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.
- There will be glitter in the room if and only if the room has gold.
- The Wumpus can be killed by the agent if the agent is facing to it, and Wumpus will emit a horrible scream which can be heard anywhere in the cave.

PEAS description of Wumpus world:

To explain the Wumpus world we have given PEAS description as below:

Performance measure:

+1000 reward points if the agent comes out of the cave with the gold.

-1000 points penalty for being eaten by the Wumpus or falling into the pit.

-1 for each action, and -10 for using an arrow.

The game ends if either agent dies or came out of the cave.

Environment:

A 4*4 grid of rooms.

The agent initially in room square [1, 1], facing toward the right.

Location of Wumpus and gold are chosen randomly except the first square [1,1].

Each square of the cave can be a pit with probability 0.2 except the first square.

Actuators:

- Left turn,
- Right turn
- Move forward
- Grab
- Release
- Shoot.

Sensors:

- The agent will perceive the **stench** if he is in the room adjacent to the Wumpus. (Not diagonally).
- The agent will perceive **breeze** if he is in the room directly adjacent to the Pit.
- The agent will perceive the **glitter** in the room where the gold is present.
- The agent will perceive the **bump** if he walks into a wall.
- When the Wumpus is shot, it emits a horrible **scream** which can be perceived anywhere in the cave.
- These percepts can be represented as five element list, in which we will have different indicators for each sensor.
- Example if agent perceives stench, breeze, but no glitter, no bump, and no scream then it can be represented as:**[Stench, Breeze, None, None, None]**.

The Wumpus world Properties:

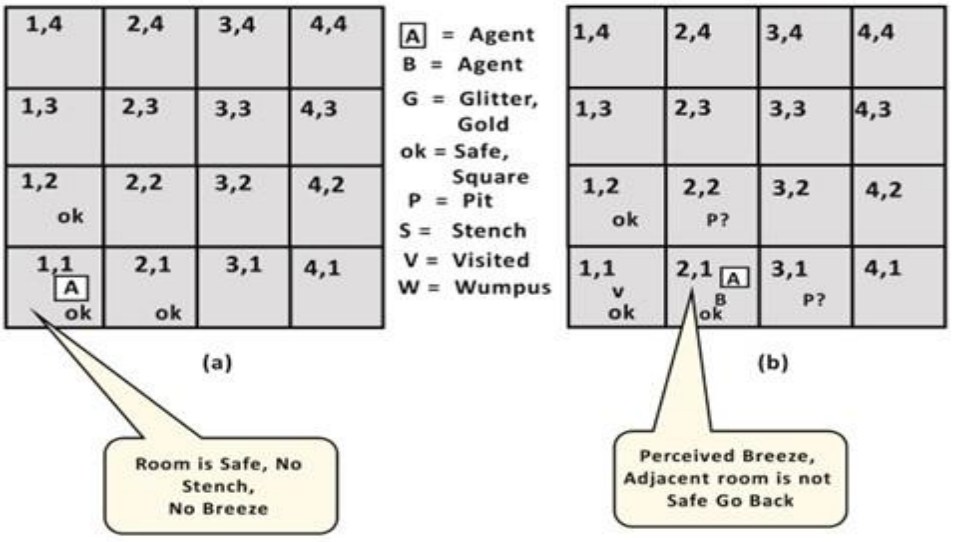
- **Partially observable:** The Wumpus world is partially observable because the agent can only perceive the close environment such as an adjacent room.
- **Deterministic:** It is deterministic, as the result and outcome of the world are already known.
- **Sequential:** The order is important, so it is sequential.
- **Static:** It is static as Wumpus and Pits are not moving.
- **Discrete:** The environment is discrete.
- **One agent:** The environment is a single agent as we have one agent only and Wumpus is not considered as an agent.

Exploring the Wumpus world:

Now we will explore the Wumpus world and will determine how the agent will find its goal by applying logical reasoning.

Agent's First step:

Initially, the agent is in the first room or on the square [1,1], and we already know that this room is safe for the agent, so to represent on the below diagram (a) that room is safe we will add symbol OK. Symbol A is used to represent agent, symbol B for the breeze, G for Glitter or gold, V for the visited room, P for pits, W for Wumpus. At Room [1,1] agent does not feel any breeze or any Stench which means the adjacent squares are also OK.



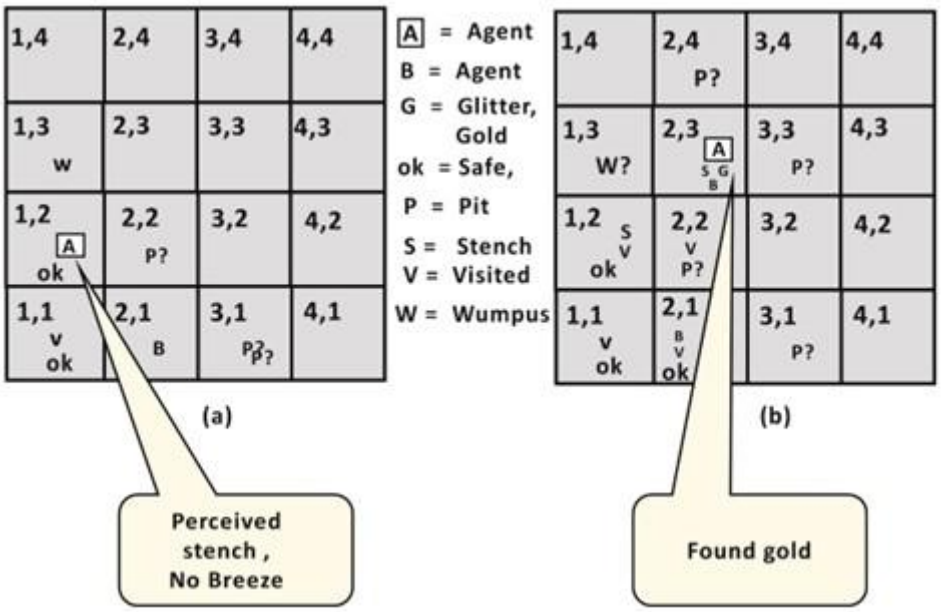
Agent's second Step:

Now agent needs to move forward, so it will either move to [1, 2], or [2,1]. Let's suppose agent moves to the room [2, 1], at this room agent perceives some breeze which means Pit is around this room. The pit can be in [3, 1], or [2,2], so we will add symbol P? to say that, is this Pit room?

Now agent will stop and think and will not make any harmful move. The agent will go back to the [1, 1] room. The room [1,1], and [2,1] are visited by the agent, so we will use symbol V to represent the visited squares.

Agent's third step:

At the third step, now agent will move to the room [1,2] which is OK. In the room [1,2] agent perceives a stench which means there must be a Wumpus nearby. But Wumpus cannot be in the room [1,1] as by rules of the game, and also not in [2,2] (Agent had not detected any stench when he was at [2,1]). Therefore agent infers that Wumpus is in the room [1,3], and in current state, there is no breeze which means in [2,2] there is no Pit and no Wumpus. So it is safe, and we will mark it OK, and the agent moves further in [2,2].



Agent's fourth step:

At room [2,2], here no stench and no breezes present so let's suppose agent decides to move to [2,3]. At room [2,3] agent perceives glitter, so it should grab the gold and climb out of the cave.

Propositional logic :a very simple logic

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

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

Example:

a) It is Sunday.

b) The Sun rises from West (False proposition)

c) 3+3= 7(False proposition)

d) 5 is a prime number.

Following are some basic facts about propositional logic:

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and logical connectives.

- These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called tautology, and it is also called a valid sentence.
- A proposition formula which is always false is called Contradiction.
- A proposition formula which has both true and false values is called
- Statements which are questions, commands, or opinions are not propositions such as "Where is Rohini", "How are you", "What is your name", are not propositions.

Syntax of propositional logic:

There are two types of Propositions:

- Atomic Propositions
- Compound propositions

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

Example:

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

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

Example:

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

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

Logical Connectives:

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

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

Negation: A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.

Conjunction: A sentence which has ∧ connective such as, P ∧ Q is called a conjunction.

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

P=Rohan is intelligent,

Q= Rohan is hardworking. P∧ Q.

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 Doctor, so we can write it as P ∨ Q.

Implication/Conditional: 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, and Q= Street is wet, so it is represented as P → Q

Biconditional: A sentence such as P⇔ Q is a Biconditional sentence, example If I am breathing, then I am alive

P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

Following is the summarized table for Propositional Logic Connectives:

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

Truth Table:

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

For Negation:

P	¬ P
True	**False**
False	**True**

For Conjunction:

P	Q	P∧ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

For disjunction:

P	Q	P ∨ Q.
True	True	**True**
False	True	**True**
True	False	**True**
False	False	**False**

For Implication:

P	Q	P→ Q
True	True	**True**
True	False	**False**
False	True	**True**
False	False	**True**

For Biconditional:

P	Q	P⇔ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**True**

Truth table with three propositions:

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

P	Q	R	¬R	P∨ Q	P∨Q→¬R
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

Logical equivalence:

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

Let's take two propositions A and B, so for logical equivalence, we can write it as A⇔B. In below truth table we can see that column for ¬A∨ B and A→B, are identical hence A is Equivalent to B

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

Properties of Operators:

Commutativity:

- P∧ Q= Q ∧ P, or
- P ∨ Q = Q ∨ P.

Associativity:

- (P ∧ Q) ∧ R= P ∧ (Q ∧ R),
- (P ∨ Q) ∨ R= P ∨ (Q ∨ R)

Identity element:

- P ∧ True = P,
- P ∨ True= True.

Distributive:

- P∧ (Q ∨ R) = (P ∧ Q) ∨ (P ∧ R).
- P ∨ (Q ∧ R) = (P ∨ Q) ∧ (P ∨ R).

DE Morgan's Law:

- ¬ (P ∧ Q) = (¬P) ∨ (¬Q)
- ¬ (P ∨ Q) = (¬ P) ∧ (¬Q).

Double-negation elimination:

- ¬ (¬P) = P.

Propositional theorem proving:

Inference:

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

Inference rules:

- Inference rules are the templates for generating valid arguments. Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal.
- In inference rules, the implication among all the connectives plays an important role. Following are some terminologies related to inference rules:
- Implication: It is one of the logical connectives which can be represented as P →Q. It is a Boolean expression.
- Converse: The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as Q → P.
- Contrapositive: The negation of converse is termed as contrapositive, and it can be represented as ¬ Q → ¬ P.
- Inverse: The negation of implication is called inverse. It can be represented as ¬ P → ¬ Q.

From the above term some of the compound statements are equivalent to each other, which we can prove using truth table:

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

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

Types of Inference rules:

Modus Ponens:

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

Notation for Modus ponens: $\frac{P \rightarrow Q, \quad P}{\therefore Q}$

Example:

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

Statement-2: "I am sleepy" ==> P

Conclusion: "I go to bed."==> Q.

Hence, we can say that, if P→ Q is true and P is true then Q will be true.

Proof by Truth table:

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

Modus Tollens:

The Modus Tollens rule state that if P→ Q is true and ¬ Q is true, then ¬ P will also true. It can be represented as:

Notation for Modus Tollens: $\frac{P \rightarrow Q, \quad \sim Q}{\sim P}$

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

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

Statement-3: Which infers that "I am not sleepy" => ~P

P	Q	~P	~Q	$P \rightarrow Q$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	0
1	1	0	0	1

Hypothetical Syllogism:

The Hypothetical Syllogism rule state that if P→R is true whenever P→Q is true, and Q→R is true. It can be represented as the following notation:

Example:

Statement-1: If you have my home key then you can unlock my home. P→Q

Statement-2: If you can unlock my home then you can take my money. Q→R

Conclusion: If you have my home key then you can take my money. P→R

Proof by truth table:

P	Q	R	$P \to Q$	$Q \to R$	$P \to R$
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	1

Disjunctive Syllogism:

The Disjunctive syllogism rule state that if P∨Q is true, and ¬P is true, then Q will be true. It can be represented as:

Notation of Disjunctive syllogism: $\frac{P \vee Q,\ \neg P}{Q}$

Example:

Statement-1: Today is Sunday or Monday. ==>P∨Q

Statement-2: Today is not Sunday.==> ¬P

Conclusion: Today is Monday. ==> Q

Proof by truth-table:

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

Addition:

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

Notation of Addition: $\frac{P}{P \vee Q}$

Example:

Statement: I have a vanilla ice-cream. ==> P

Statement-2: I have Chocolate ice-cream.

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

Proof by Truth-Table:

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

Simplification:

The simplification rule state that if P∧ Q is true, then Q or P will also be true. It can be represented as:

Notation of Simplification rule: $\frac{P \wedge Q}{Q}$ Or $\frac{P \wedge Q}{P}$

Proof by Truth-Table:

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

Resolution:

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

Notation of Resolution $\frac{P \vee Q,\ \neg P \wedge R}{Q \vee R}$

Proof by Truth-Table:

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

Unit Highlights

- A **CSP** consists of variables, domains, and constraints.
- **Constraint propagation** reduces domains and improves efficiency.
- **Backtracking search** is the standard search method for CSPs.
- A **knowledge-based agent** stores and reasons with facts in a knowledge base.
- The **Wumpus World** is a classic example of logical reasoning in AI.
- **Propositional logic** is a simple formal language for true/false facts.
- **Theorem proving** and **model checking** are important reasoning methods.
- Agents based on **propositional logic** can infer actions from stored knowledge.

Case Study: Wumpus World as a Logical Agent Problem

The Wumpus World is a good case study for logical agents. The agent begins with limited knowledge, perceives breeze or stench in nearby squares, and uses propositional logic rules to infer which squares are safe. It then

moves carefully, updates its knowledge base, and reasons about where the gold may be. This case study shows how perception, knowledge representation, and logical inference work together in an intelligent agent.

TEXT BOOKS:

1. *Artificial Intelligence A Modern Approach, 2015, Third Edition , Stuart J. Russell and Peter Norvig Pearson Education.*

REFERENCE BOOKS:

5. *Artificial Intelligence, 3/e,2009, Elaine Rich, Kevin Knight and Shiva shankar B Nair Tata McGraw Hill.*
6. *Artificial Intelligence-Structures and Strategies for Complex Problem Solving, 5/e, 2009, George F. Luther, Pearson Education.*
7. *Introduction to Artificial Intelligence, Eugene Charniak and Drew McDermott, Pearson Education, 1987.*

S.No	Questions	Blooms Taxonomy Level
UNIT III - CONSTRAINT SATISFACTION PROBLEMS		
PART-A (Two Marks Questions)		
1.	What are the three main components of a CSP?	L1
2.	Give an example of a real-world CSP.	L1
3.	What is a binary constraint in CSPs?	L1
4.	What is constraint propagation?	L1
5.	What is arc consistency in CSPs?	L1
6.	What is node consistency?	L1
7.	What is path consistency?	L1
8.	What is the purpose of inference in CSPs?	L1
9.	What is backtracking search?	L1
10.	What is forward checking in backtracking search?	L1
11.	What is Knowledge base.	L1
12.	What are the components of knowledge based agent.	L1
13.	List the logical connectives.	L1
14.	What are the types of Quantifiers in first order logic	L1
15	What is propositional logic?	L1
16	Give an example of a propositional logic statement.	L1
17	What is the difference between a tautology and a contradiction?	L1
18	What are the basic logical connectives in propositional logic?	L1
19	What is a truth table?	L1
20	What is propositional theorem proving?	L1
PART-B (Ten Marks Questions)		
1.	Explain the Constraint Satisfaction Problems with suitable example	L3
2.	What is backtracking search in CSPs? Explain the basic algorithm and discuss its strengths and weaknesses.	L4
3.	How does a knowledge-based agent use inference to make decisions? Provide examples to illustrate your answer.	L3
4.	Explain the role of percepts such as Breeze, Stench, and Glitter in the Wumpus World. How do these help an agent make logical inferences?	L3
5.	What is propositional logic? Explain its syntax, semantics, and use in AI with examples.	L2
6.	Explain the significance of logical connectives (AND, OR, NOT, IMPLICATION, BICONDITIONAL) in propositional logic. Provide truth tables for each.	L2
7.	What is propositional theorem proving? Explain its importance in AI and knowledge representation.	L4
8.	Explain the process of propositional model checking. How does it verify the truth of logical statements?	L2
9.	How do logical agents use propositional logic for decision-making? Explain with examples.	L3
10.	Explain about Inference in Constraint Satisfaction problems	L3

UNIT – IV FIRST ORDER LOGIC:

“Logic and probability theory are two of the main tools in the formal study of reasoning.”

— Stanford Encyclopedia of Philosophy

Unit Overview

This unit explains the **syntax and semantics of first-order logic**, how first-order logic is used for knowledge representation, and how knowledge engineering is carried out using FOL. It also covers important inference methods in first-order logic, including the difference between propositional and first-order inference, unification and lifting, forward chaining, backward chaining, and resolution. These topics are central to building intelligent systems that can reason with structured knowledge.

Learning Outcomes

After completing this unit, students will be able to:

1. Define the **syntax and semantics of first-order logic**.
2. Explain how **first-order logic** is used to represent knowledge.
3. Describe the process of **knowledge engineering in first-order logic**.
4. Compare **propositional inference** and **first-order inference**.
5. Explain **unification and lifting** in first-order reasoning.
6. Describe **forward chaining**, **backward chaining**, and **resolution** in first-order logic.

Importance of Studying this Unit

This unit is important because first-order logic provides a much richer language for representing real-world knowledge than propositional logic. It allows agents to reason about classes of objects, relationships among entities, and universally or existentially quantified facts. AI textbooks treat first-order inference as a core method for answering logically stated questions, especially when knowledge must be represented in a structured and general way.

Key Concepts

First-order logic, syntax, semantics, constant, variable, function, predicate, quantifier, interpretation, inference, unification, lifting, forward chaining, backward chaining, resolution, entailment.

First-Order Logic in Artificial intelligence

In the topic of Propositional logic, we have seen that how to represent statements using propositional logic. But unfortunately, in propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

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

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

First-Order logic:

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.

- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
- **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus,
- **Relations: It can be unary relation such as:** red, round, is adjacent, **or n- any relation such as:** the sister of, brother of, has color, comes between
- **Function:** Father of, best friend, third inning of, end of, ...
- As a natural language, first-order logic also has two main parts:
    - **Syntax**
    - **Semantics**

Syntax of First-Order logic:

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

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

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)**.

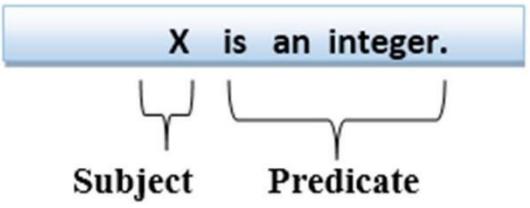
Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky).

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:
- **Universal Quantifier, (for all, everyone, everything)**
- **Existential quantifier, (for some, at least one).**

Universal Quantifier:

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

The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

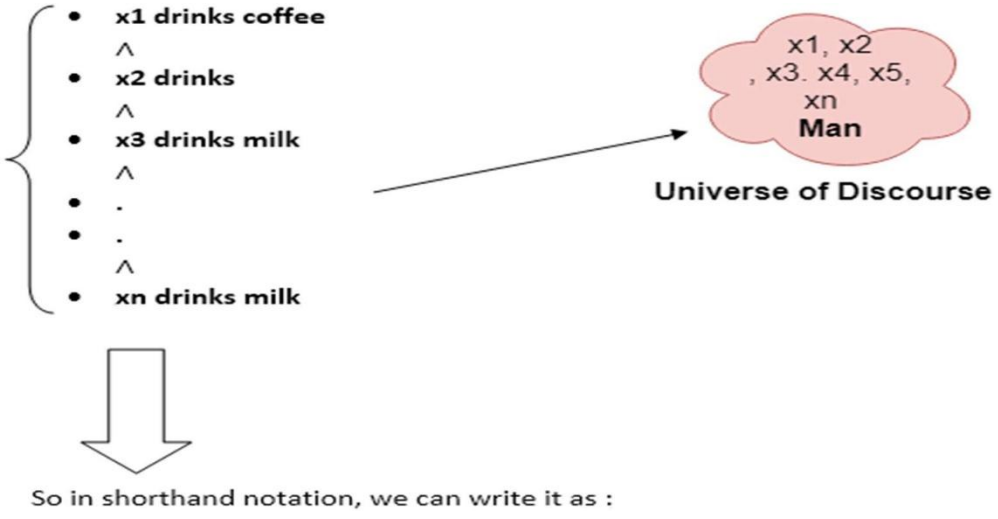
If x is a variable, then ∀x is read as:

- **For all x**
- **For each x**
- **For every x.**

Example:

All man drink coffee.

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



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

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

Existential Quantifier:

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

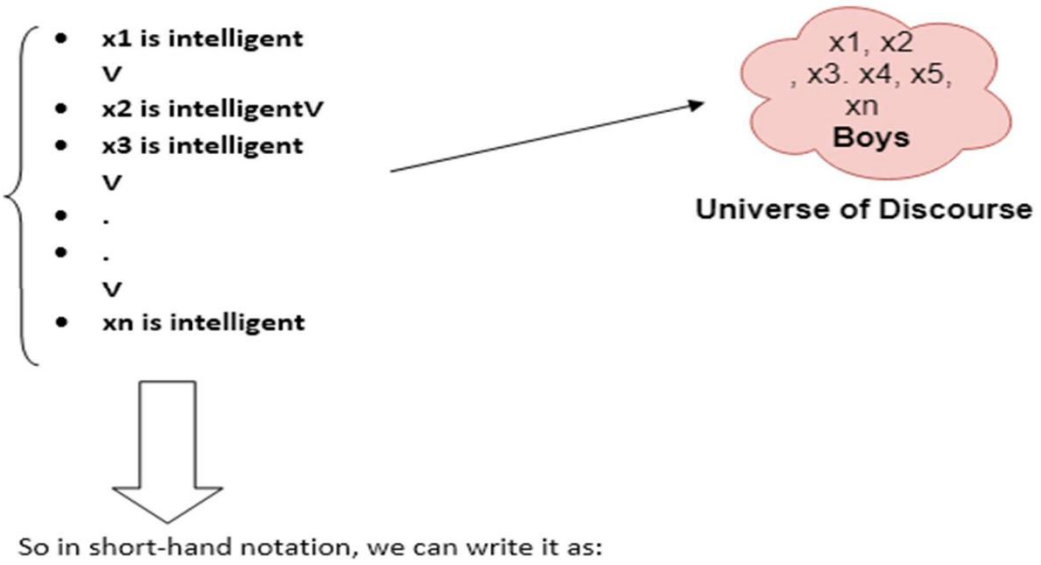
It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

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

- There exists a 'x.'
- For some 'x.'
- For at least one 'x.'

Example:

Some boys are intelligent.



∃x: boys(x) ∧ intelligent(x)

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

Points to remember:

- The main connective for universal quantifier ∀ 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.

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.

Knowledge Engineering in First-order logic

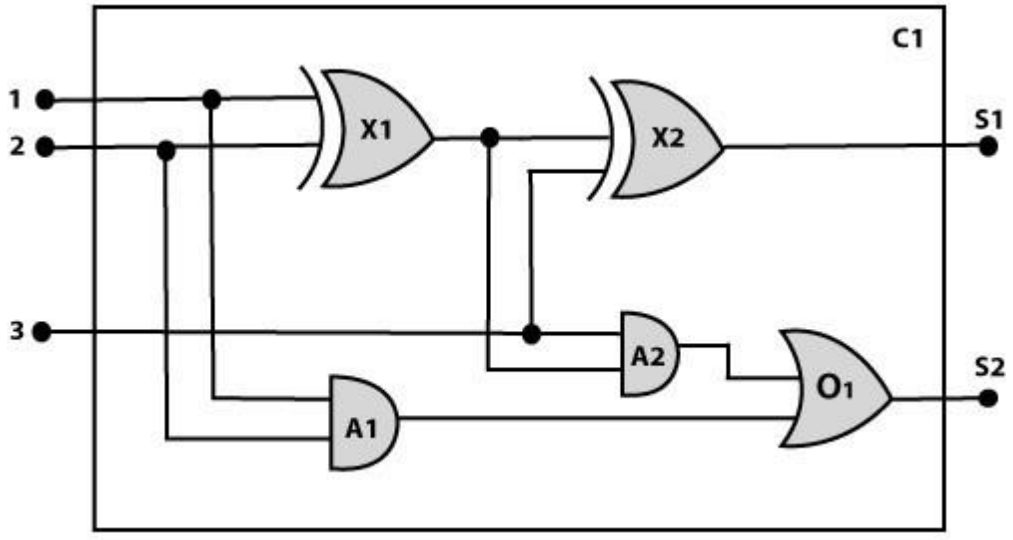
What is knowledge-engineering?

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.

In this topic, we will understand the Knowledge engineering process in an electronic circuit domain, which is already familiar. This approach is mainly suitable for creating special- purpose knowledge base.

The knowledge-engineering process:

Following are some main steps of the knowledge-engineering process. Using these steps, we will develop a knowledge base which will allow us to reason about digital circuit (One- bit full adder) which is given below



1. **Identify the task:**

The first step of the process is to identify the task, and for the digital circuit, there are various reasoning tasks.

At the first level or highest level, we will examine the functionality of the circuit:

o **Does the circuit add properly?**

- **What will be the output of gate A2, if all the inputs are high?**

At the second level, we will examine the circuit structure details such as:

- **Which gate is connected to the first input terminal?**
- **Does the circuit have feedback loops?**

2.Assemble the relevant knowledge:

In the second step, we will assemble the relevant knowledge which is required for digital circuits. So for digital circuits, we have the following required knowledge:

- Logic circuits are made up of wires and gates.
- Signal flows through wires to the input terminal of the gate, and each gate produces the corresponding output which flows further.
- In this logic circuit, there are four types of gates used: **AND, OR, XOR, and NOT**.
- All these gates have one output terminal and two input terminals (except NOT gate, it has one input terminal).

3.Decide on vocabulary:

The next step of the process is to select functions, predicate, and constants to represent the circuits, terminals, signals, and gates.

Firstly we will distinguish the gates from each other and from other objects. Each gate is represented as an object which is named by a constant, such as, **Gate(X1)**. The functionality of each gate is determined by its type, which is taken as constants such as **AND, OR, XOR, or NOT**. Circuits will be identified by a predicate: **Circuit (C1)**.

For the terminal, we will use predicate: **Terminal(x)**.

For gate input, we will use the function **In(1, X1)** for denoting the first input terminal of the gate, and for output terminal we will use **Out (1, X1)**.

The function **Arity(c, i, j)** is used to denote that circuit c has i input, j output.

The connectivity between gates can be represented by predicate **Connect(Out(1, X1), In(1, X1))**.

We use a unary predicate **On (t)**, which is true if the signal at a terminal is on.

4.Encode general knowledge about the domain:

To encode the general knowledge about the logic circuit, we need some following rules:

- If two terminals are connected then they have the same input signal, it can be represented as:

1.∀ t1, t2 Terminal (t1) ∧ Terminal (t2) ∧ Connect (t1, t2) → Signal (t1) = Signal (2).

- Signal at every terminal will have either value 0 or 1, it will be represented as:

2. ∀ t Terminal (t) →Signal (t) = 1 ∨Signal (t) = 0.

- Connect predicates are commutative:

3. ∀ t1, t2 Connect(t1, t2) → Connect (t2, t1).

- Representation of types of gates:

4. ∀ g Gate(g) ∧ r = Type(g) → r = OR ∨r = AND ∨r = XOR ∨r = NOT.

- Output of AND gate will be zero if and only if any of its input is zero

5. ∀ g Gate(g) ∧ Type(g) = AND →Signal (Out(1, g))= 0 ⇔ ∃n Signal (In(n, g))= 0.

- Output of OR gate is 1 if and only if any of its input

6. ∀ g Gate(g) ∧ Type(g) = OR → Signal (Out(1, g))= 1 ⇔ ∃n Signal (In(n, g))= 1

- Output of XOR gate is 1 if and only if its inputs are different:

7. ∀ g Gate(g) ∧ Type(g) = XOR → Signal (Out(1, g)) = 1 ⇔ Signal (In(1, g)) ≠ Signal (In(2, g)).

- Output of NOT gate is invert of its input:

8. ∀ g Gate(g) ∧ Type(g) = NOT → Signal (In(1, g)) ≠ Signal (Out(1, g)).

- All the gates in the above circuit have two inputs and one output (except NOT gate).

Encode a description of the problem instance:

Now we encode problem of circuit C1, firstly we categorize the circuit and its gate components. This step is easy if ontology about the problem is already thought.

This step involves the writing simple atomics sentences of instances of concepts, which is known as ontology.

For the given circuit C1, we can encode the problem instance in atomic sentences as below:

Since in the circuit there are two XOR, two AND, and one OR gate so atomic sentences for these gates will be:

1. For XOR gate: Type(x1)= XOR, Type(X2) = XOR
2. For AND gate: Type(A1) = AND, Type(A2)= AND
3. For OR gate: Type (O1) = OR. And then represent the connections between all the gates.

Pose queries to the inference procedure and get answers:

In this step, we will find all the possible set of values of all the terminal for the adder circuit. The first query will be:

What should be the combination of input which would generate the first output of circuit C1, as 0 and a second output to be 1?

1. ∃ i1, i2, i3 Signal (In(1, C1))=i1 ∧ Signal (In(2, C1))=i2 ∧ Signal (In(3, C1))= i3
2. ∧ Signal (Out(1, C1)) =0 ∧ Signal (Out(2, C1))=1

Debug the knowledge base:

Now we will debug the knowledge base, and this is the last step of the complete process. In this step, we will try to debug the issues of knowledge base.

Inference in First-Order Logic

Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences. Before understanding the FOL inference rule, let's understand some basic terminologies used in FOL.

Substitution:

Substitution is a fundamental operation performed on terms and formulas. It occurs in all

inference systems in first-order logic. The substitution is complex in the presence of quantifiers in FOL. If we write F[a/x], so it refers to substitute a constant "a" in place of variable "x".

Equality:

First-Order logic does not only use predicate and terms for making atomic sentences but also uses another way, which is equality in FOL. For this, we can use equality symbols which specify that the two terms refer to the same object.

Example: Brother (John) = Smith.

As in the above example, the object referred by the Brother (John) is similar to the object referred by Smith. The equality symbol can also be used with negation to represent that two terms are not the same objects.

Example: ¬(x=y) which is equivalent to x ≠y.

FOL inference rules for quantifier:

As propositional logic we also have inference rules in first-order logic, so following are some basic inference

rules in FOL:

- **Universal Generalization**
- **Universal Instantiation**
- **Existential Instantiation**
- **Existential introduction**

Universal Generalization:

- Universal generalization is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as ∀ x P(x).
- It can be represented as: $\frac{P(c)}{\forall x\, P(x)}$.
- This rule can be used if we want to show that every element has a similar property.
- In this rule, x must not appear as a free variable.

- **Example:** Let's represent, P(c): "**A byte contains 8 bits**", so for ∀ **x P(x)** "**All bytes contain 8 bits**.", it will also be true.

Universal Instantiation:

- Universal instantiation is also called as universal elimination or UI 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)),**

Existential Instantiation:

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

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**.

Existential introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the property P.
- It can be represented as: $\frac{P(c)}{\exists x P(x)}$

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.

What is 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, **Ψ1 = Ψ2** , 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.

Forward Chaining and backward chaining in AI

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

Inference engine:

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

a. **Forward chaining**
b. **Backward chaining**

Forward chaining

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

Properties of Forward-Chaining:

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

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

Example:

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

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

Facts Conversion into FOL:

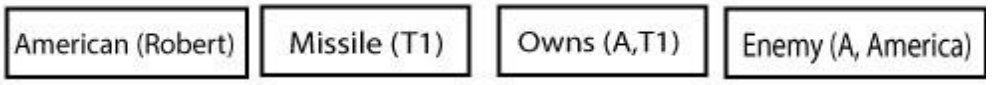
- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)
  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p)……(1)
- Country A has some missiles. **?p Owns(A, p) ∧ Missile(p)**. It can be written in two definite clauses by using Existential 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:

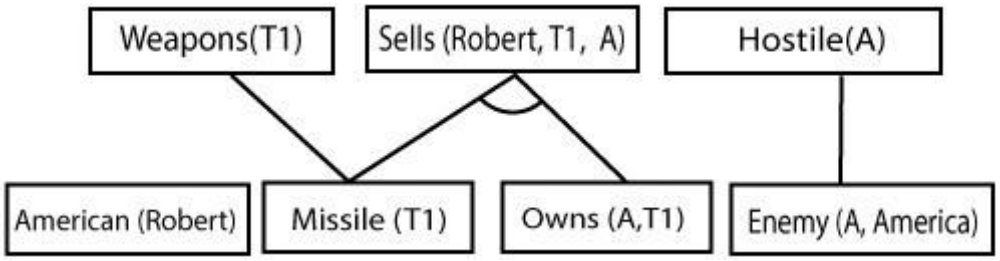
In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: **American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1)**. All these facts will be represented as below.



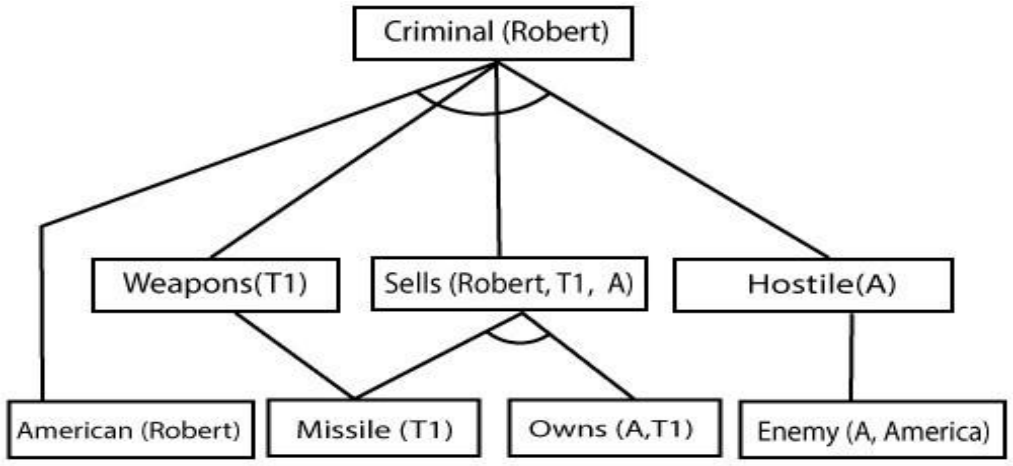
Step-2:

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.

Backward Chaining:

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

Properties of backward chaining:

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

Example:

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

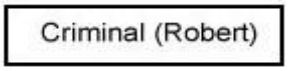
- American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1) Owns(A, T1) (2)
- Missile(T1)
- ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) (4)
- Missile(p) → Weapons (p) (5)
- Enemy(p, America) →Hostile(p) (6)
- Enemy (A, America) (7)
- American(Robert). (8)

Backward-Chaining proof:

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

Step-1:

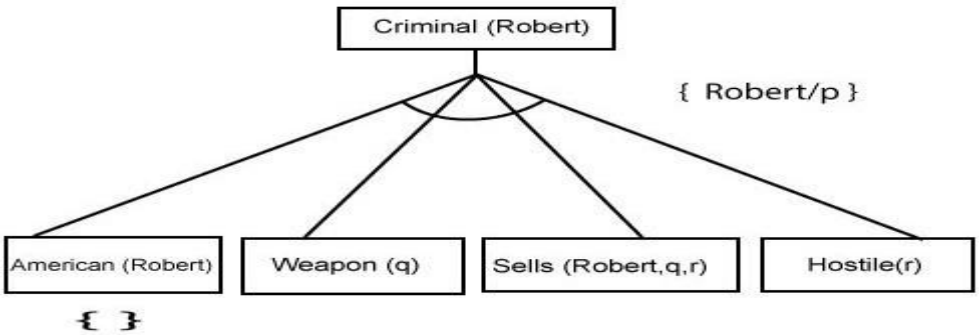
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.



Step-2:

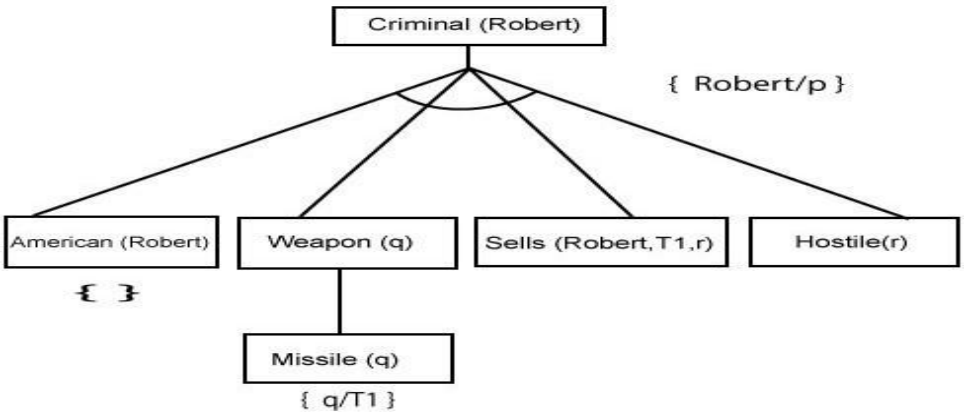
At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.

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

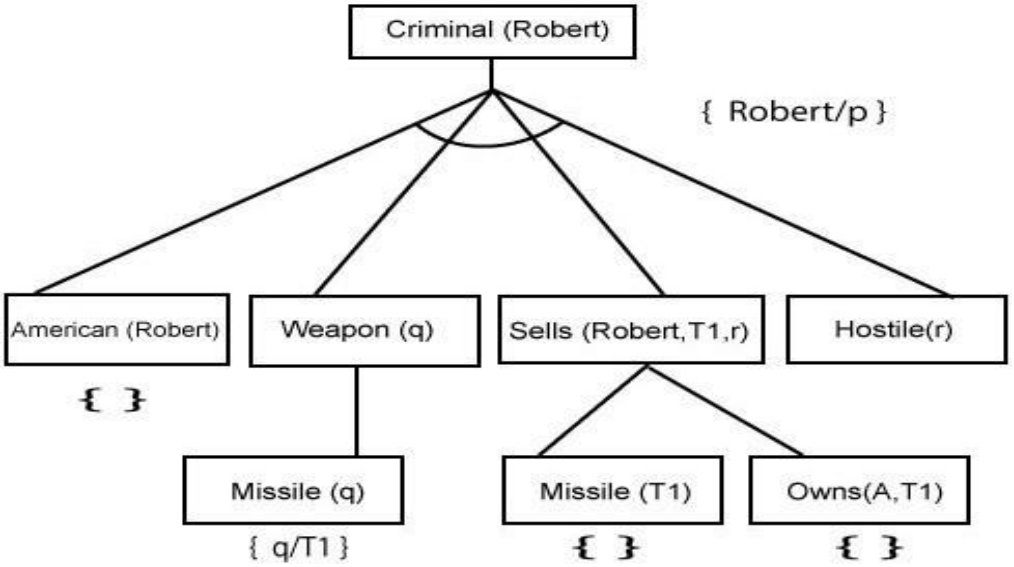


Step-3:t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5).

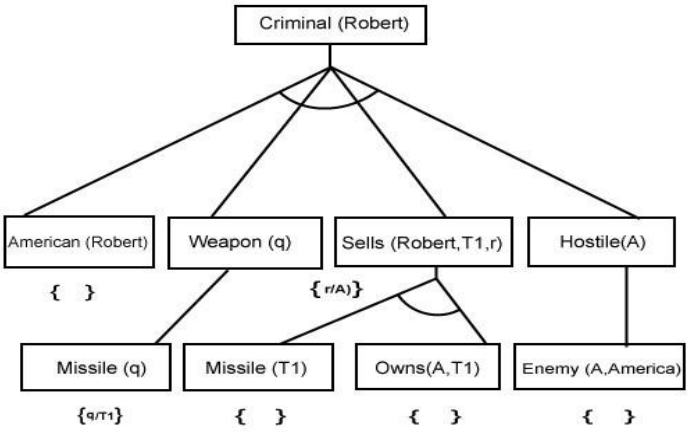
Weapon (q) is also true with the substitution of a constant T1 at q.



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- And hence all the statements are proved true using backward chaining



Unit Highlights

- First-order logic is more expressive than propositional logic.
- Syntax defines the structure of FOL sentences, while semantics defines their meaning.
- First-order logic can represent objects, properties, relations, and quantified rules.
- Knowledge engineering in FOL involves identifying tasks, encoding knowledge, and testing queries.
- First-order inference uses methods such as unification, lifting, forward chaining, backward chaining, and resolution.
- Forward chaining is data-driven, while backward chaining is goal-driven.
- Resolution is a general theorem-proving method in first-order logic.

TEXT BOOKS:

1. *Artificial Intelligence A Modern Approach, 2015, Third Edition , Stuart J. Russell and Peter Norvig Pearson Education.*

REFERENCE BOOKS:

2. *Artificial Intelligence, 3/e,2009, Elaine Rich, Kevin Knight and Shiva shankar B Nair Tata McGraw Hill.*
3. *Artificial Intelligence-Structures and Strategies for Complex Problem Solving, 5/e, 2009, George F. Luther, Pearson Education.*
4. *Introduction to Artificial Intelligence, Eugene Charniak and Drew McDermott, Pearson Education, 1987.*

S.No	Questions	Blooms Taxonomy Level
UNIT IV – FIRST ORDER LOGIC		
PART-A (Two Marks Questions)		
1.	What is the syntax of First-Order Logic (FOL)?	L1
2.	What is the semantics of FOL?	L1
3.	How does FOL differ from propositional logic?	L1
4.	What are predicates in FOL?	L2
5.	What is knowledge engineering in FOL?	L2
6.	What is the role of a knowledge base in FOL?	L2
7.	What is unification in FOL?	L3
8.	What is lifting in the context of FOL?	L2
9.	What is forward chaining?	L2
10.	What is backward chaining?	L2
11.	What is resolution in FOL?	L2
PART-B (Ten Marks Questions)		
1.	Explain the syntax and semantics of First-Order Logic (FOL). Provide examples to illustrate your explanation.	L3
2.	Discuss how First-Order Logic is utilized in Artificial Intelligence for knowledge representation and reasoning.	L4
3.	Describe the process of knowledge engineering using First-Order Logic. What are the key steps involved in building a knowledge base.	L3
4.	Compare propositional logic inference with first-order logic inference.	L4
5.	Define unification and lifting in the context of first-order logic inference. Why are they important?	L2
6.	Explain forward chaining as an inference technique in First-Order Logic.	L3
7.	Discuss backward chaining and illustrate how it differs from forward chaining in logical reasoning.	L3
8.	What is resolution in First-Order Logic? Explain the steps involved in resolution-based theorem proving.	L2

UNIT - V KNOWLEDGE REPRESENTATION:

“Machines as intelligent as humans should be able to do most of the things humans can do.” — Nils J. Nilsson

Unit Overview

This unit explains how AI systems represent the world using **categories, objects, and events**, how they reason about categories and defaults, and how these ideas are illustrated in the **Internet Shopping World** example. It also introduces probabilistic reasoning, including **basic probability notation, full joint distributions, independence, and Bayes' Rule**, which are central tools for acting under uncertainty.

Learning Outcomes

After completing this unit, students will be able to:

1. Explain the concept of **ontological engineering**.
2. Describe **categories, objects, and events** in knowledge representation.
3. Explain reasoning systems for categories and reasoning with default information.
4. Describe the **Internet Shopping World** as an ontology example.
5. Explain why intelligent agents must act under uncertainty.
6. Use **basic probability notation** correctly.
7. Explain inference using **full joint distributions**, **independence**, and **Bayes' Rule**.

Importance of Studying this Unit

This unit is important because intelligent agents must both **represent structured knowledge** about the world and **make decisions in uncertain situations**. Ontological engineering supports clear and reusable knowledge representation, while probability provides a mathematically sound way to reason when knowledge is incomplete or noisy. These ideas are foundational in modern AI systems.

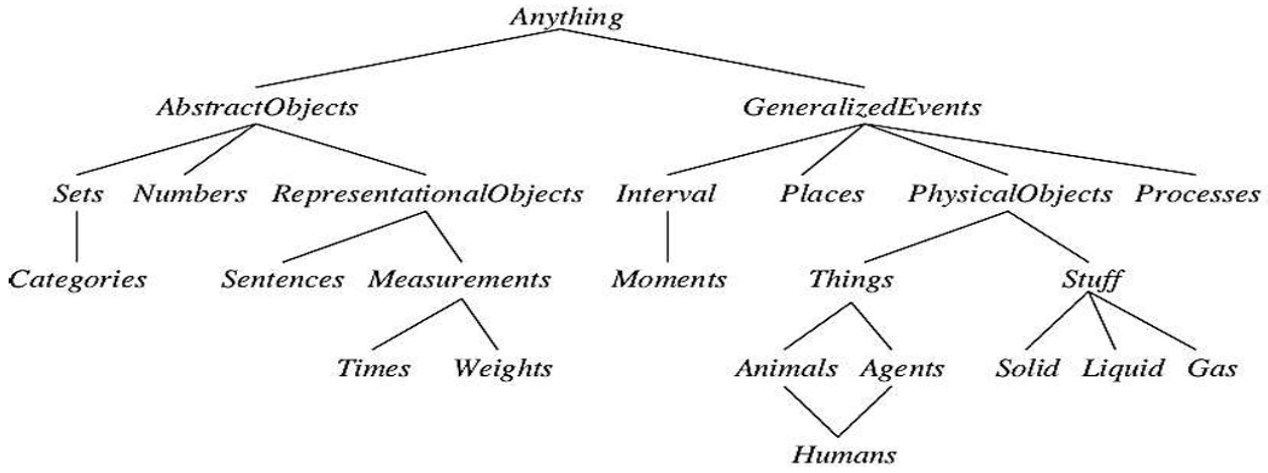
Key Concepts

Ontology, ontological engineering, category, object, event, inheritance, default reasoning, Internet Shopping World, uncertainty, probability, random variable, full joint distribution, independence, conditional probability, Bayes' Rule.

Ontological Engineering

- Knowledge is the information about a domain that is used to solve problems of a domain.
- As part of designing a program to solve problems, we must define how the knowledge is represented.
- Complex domains require more general and hexible representations of concepts like events, time, physical object and beliefs of a domain.
- Representing these abstract concepts is called Ontological Engineering
- The general frame work of concepts is called upper ontology, because of the convention of drawing graphs with the general concepts at the top and the more specific concepts below them, as in Figure

Upper Ontology



General representation of ontology of world

CATEGORIES AND OBJECTS

- The organization of objects into categories is a vital part of knowledge representation.
- Although interaction with the world takes place at the level of individual objects, much reasoning takes place at the level of categories.
- For example, a shopper would normally have the goal of buying a basketball, rather than a particular basketball such as BB9.
- Categories also serve to make predictions about objects once they are classified.
- There are two ways of representing categories in first-order logic:
    - Predicates : eg.Basketball (b),
    - Objects: eg.Basketballs
- Categories serve to organize and simplify the knowledge base through inheritance.
- If we say that all instances of the category Food are edible, and if we assert that Fruit is a subclass of Food and Apples is a subclass of Fruit, then we can infer that every apple is edible.
- We say that the individual apples inherit the property of edibility, in this case from their membership in the Food category.

Relations between categories

- Two or more categories are ***disjoint*** if they have no members in common:
  - Disjoint(s)$\Leftrightarrow(\forall\ c_1,c_2\ \ c_1 \in s \wedge c_2 \in s \wedge c_1 \neq c_2 \Rightarrow$ Intersection$(c_1,c_2) = \varnothing)$
  - Example; Disjoint({animals, vegetables})
- A set of categories *s* constitutes an ***exhaustive decomposition*** of a category *c* if all members of the set *c* are covered by categories in *s*:
  - E.D.$(s,c) \Leftrightarrow (\forall\ i\ \ i \in c \Rightarrow \exists\ c_2\ \ c_2 \in s \wedge i \in c_2)$
  - Example: ExhaustiveDecomposition ({Americans, Canadian, Mexicans},NorthAmericans)
- A ***partition*** is a disjoint exhaustive decomposition:
  - Partition(s,c) $\Leftrightarrow$ Disjoint(s) $\wedge$ E.D.(s,c)
  - Example: Partition({Males,Females},Persons).
- Is ({Americans,Canadian, Mexicans}, NorthAmericans) a partition? • No! There might be dual citizenships.
- Categories can be defined by providing necessary and sufficient conditions for membership
  - $\forall$ x Bachelor(x) $\Leftrightarrow$ Male(x) $\wedge$ Adult(x) $\wedge$ Unmarried(x)

Objects

- The real world can be seen as consisting of primitive objects(e.g., atomic particIes)and composite objects built from them.
- Stuff ex: butter
- Things ex: dog
- some properties are intrinsic: they belong to the very substance of the object, rather than to the objects as a whole extrinsic properties-weight, length, shape and so on -are not retained under subdivision.
- A category of objects that includes in its definition only intrinsic properties is then a substance, or mass noun; a class that includes any extrinsic properties in its definition is a count noun
- The category Stuff is the most general substance category, specifying no intrinsic properties.
- The category Thing is the most general discrete object category, specifying no extrinsic properties.

Ex:

- any part of a butter-object is also a butter-object:
- bE Butter A Partof (p, b) <* p EButter .

- We can now say that butter melts at around 30 degrees centigrade:
- bG Butter °<• MeltingPoint(b,Centigrade(30))

Events

- situation calculus represents actions and their effects
- situation calculus- it was designed to describe a world in which actions are discrete, instantaneous and happen one at a time.
- Event calculus-based on points of time rather than on situations
- event calculus reifies fluents and events
- The fluent At(shankar, berkeley): is an object that refers to the fact of shankar being in Berkeley, but does not by itself say anything about whether it is true.
- To assert that a fluent is actually true at some point in time we use the predicate T, as in T(At(shankar, Berkeley),t).
- Events are described as instances of event categories
- The event E1 of shankar flying from San Francisco to Washington D.C. is described as
- E1 G Flyings A Flyer (E1, Shankar) A Origin(E1, SF) A Destination(E1,DC) .
- we can define an alternative three-argument version of the category of flying events and say
- E1 G FIyings(Shankar , SF,DC) .
- The complete set of predicates for one version of the event calculus is
- T(f, t) Fluent f is true at time t
- Happens(e, i) Event e happens over the time interval i

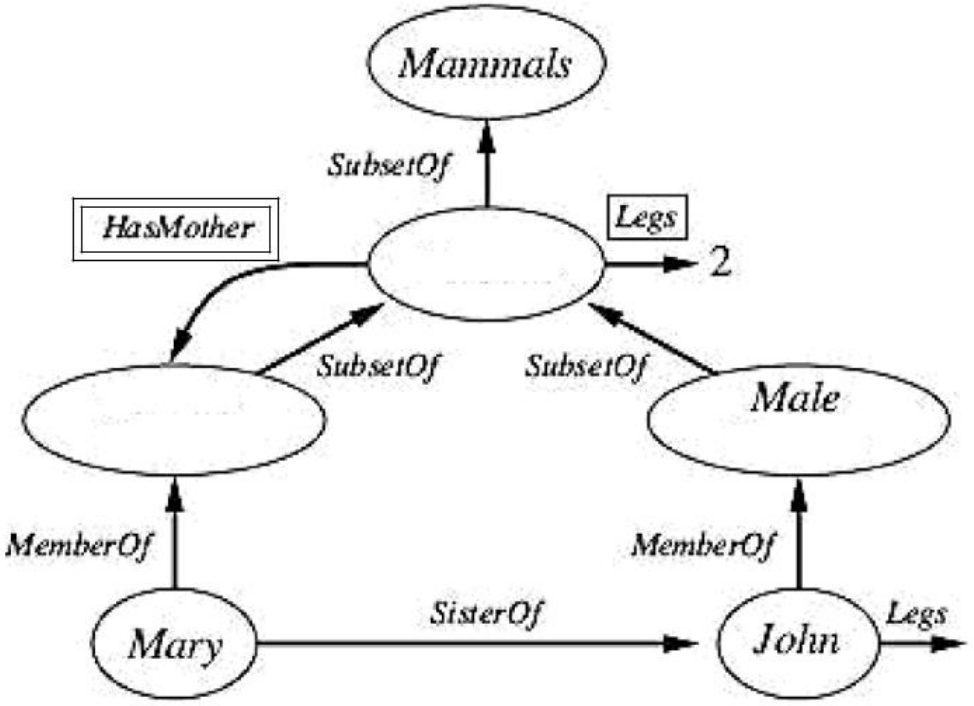
Reasoning Systems For Categories

- Categories are the primary building blocks of large-scale knowledge representation schemes.
- There are two systems specially designed for organizing and reasoning with categories.
- semantic networks provide graphical aids for visualizing a knowledge base and efficient algorithms for inferring properties of an object on the basis of its category membership;
- description logics provids a formal language for constructing and combining category definitions and efficient algorithms for deciding subset and superset relationships between categories.

Semantic Networks:

- a graphical notation of nodes and edges called existential graphs.
- There are many variants of semantic networks, but all are capable of representing individual objects, categories of objects, and relations among objects.

- A typical graphical notation displays object or category names in ovals or boxes, and connects them with labeled links
- For example, Figure has a Memberof link between Mary and FemalePersons ,corresponding to the logical assertion Mary E FemalePersons
- similarly, the Sisterof link between Mary and John corresponds to the assertion Sisterof (Mary, John).
- We can connect categories using Subsetof links, and so on.



Description logics

- Description logics are notations that are designed to make it easier to describe definitions and properties of categories.
- The principal inference tasks for description logics are subsumption (checking if one category is a subset of another by comparing their definitions) and classification (checking whether an object belongs to a category)
- Some systems also include consistency of a category definition whether the membership criteria are logically satisfiable

REASONING WITH DEFAULT INFORMATION

- Circumscription can be seen as a more powerful and precise version of the closed world assumption.
- The idea is to specify particular predicates that are assumed to be "as false as possible"—that is, false for every object except those for which they are known to be true.
- For example, suppose we want to assert the default rule that birds fly.
- We would introduce a predicate, say Abnormal1(x), and write
- Bird(x) A Abnormal1(x) n Flies(x) .
- If we say that Abnormal 1 is to be circumscribed, a circumscriptive reasoner is entitled to assume Abnormal 1(x) unless Abnormal 1(x) is known to be true.

- This allows the conclusion Flies (Tweety) to be drawn from the premise Bird (Tweety), but the conclusion no longer holds if Abnormal1(Tweety) is asserted.
- Default logic is a formalism in which default rules can be written to generate contingent, nonmonotonic conclusions. A default rule looks like this:
- Bird(x): Flies(x)/Flies(x) .
- This rule means that if Bird(x) is true, and if Flies(x) is consistent with the knowledge base, then Flies(x) may be concluded by default.
- In general, a default rule has the form P : J1, …, Jn/C
- where P is called the prerequisite, C is the conclusion, and Ji are the justifications.
- if any one of them can be proven false, then the conclusion cannot be drawn.
- Any variable that appears in Ji or C must also appear in P

The Internet Shopping world:

- In this section , we will create a shopping research agent that helps a buyer find product offers on the Internet .
- The shopping agent is given a product description by the buyer and has the task of producing a list of Web pages that offer such a product for sale and ranking for the best.

Example Online Store

Select from our fine line of products:

- Computers
- Cameras
- Books
- Videos
- Music

```
<h1>Example Online Store</h1>
<i>Select</i> from our fine line of products:
<ul>
<li> <a href="http://example.com/compu">Computers</a>
<li> <a href="http://example.com/camer">Cameras</a>
<li> <a href="http://example.com/books">Books</a>
<li> <a href="http://example.com/video">Videos</a>
<li> <a href="http://example.com/music">Music</a>
</ul>
```

The above figure shows a web page and a corresponding HTML character string.

- The Agents first task is to find relevant product offers.
- Let us consider query be a product description that the user types in (e.g.”laptop”);
- then a page is relevant offer for query,if the page is relevant and the page
- is indeed an offer. Also keep track of URL associated with the page.
- Relevant offer(page,urI,query) ••ReIevant(page,urI,query)^Offer(page).
- We can say a page is an offer if it contains the word "buy" or "price" within an HTML link or form on the page.
- If the page contains a string of the form "<a...buy..." then it is an offer, it could also be say "price" instead of "buy" or use "form" instead of "a". We can write axioms for this:
- Offer(page) •• (InTag(“a”,str,page)v InTag(“form”,str,page)) ^ (In(“buy”,str) v In(“price”,str)).
- We need to find the relevant pages.
- The strategy is to start at the home page of an online store and consider all the pages that can be reached

by the following relevant links.

- The Agent will have a knowledge of a number of stores, for example: Amazon EOnlineStores A Homepage(Amazon, “amazon.com”) . Ebay GOnlineStores A Homepage(Ebay, “ebay.com”) .
- Example Store G Online Stores A Homepage(ExampIeStore, “example.com”)
- These stores classify their goods into product categories, and provide links to
- the Major categories from their home page.
- Minor categories can be reached by following a chain of relevant links, and eventually we will reach offers .
- A page is relevant to the query if it can be reached by a chain of relevant category links from a stores home page, and following one more link to the product offer :
- Relevant(page, query) O 3 store, home store GOnlineStores A Homepage(store, home) A 3urI , url 2 ReIevantChain(home, url 2, query) A Link(url 2, url) A page = Contents(url)
- A chain of links between two URLs, start and end ,is relevant to a description d if the anchor text of each link is relevant category name for d.
- The existence of the chain itself is determined by a recursive definition , with the empty chain (start = end)as the base case:
- First we need to relate strings to the categories they name.
- This is done by using the predicate Name(s,c), which says that string s is a name for category c —for example , we might assert that Name(“Iaptops”,Laptop Computers).
- Suppose the query is “laptops”, then RelevantCategoryName(query,text) is true when one of the following holds:
- The text and query name the same category—e.g.,”laptop computers” and “laptops”.
- The text names a super category such as “computers”.
- The text names a subcategory such as “ultralight notebooks”.
- The logical definition of Relevant Category Name is as follows:
- ReIevant Category Name(query, text) <<• 3 c1, c2 Name(query, c1) A 83 Name(text, c2) A (c1 fg c2 V c2 fg c1)

Acting Under Uncertainty

- Artificial intelligence (AI) uncertainty is when there’s not enough information or ambiguity in data or decision-making. It is a fundamental concept in AI, as real-world data is often noisy and incomplete. AI systems must account for uncertainty to make informed decisions.
- AI deals with uncertainty by using models and methods that assign probabilities to different outcomes. Managing uncertainty is important for AI applications like self-driving cars and medical diagnosis, where safety and accuracy are key

Sources of Uncertainty in AI

There are several sources of uncertainty in AI that can impact the reliability and effectiveness of AI systems. Here are some common sources of uncertainty in AI:

Data Uncertainty: AI models are trained on data, and the quality and accuracy of the data can affect the performance of the model. Noisy or incomplete data can lead to uncertain predictions or decisions made by the AI system.

Model Uncertainty: AI models are complex and can have various parameters and hyperparameters that need to be tuned. The choice of model architecture, optimization algorithm, and hyperparameters can significantly impact the performance of the model, leading to uncertainty in the results.

Algorithmic Uncertainty: AI algorithms can be based on different mathematical formulations, leading to different results for the same problem. For example, different machine learning algorithms can produce different predictions for the same dataset.

Environmental Uncertainty: AI systems operate in dynamic environments, and changes in the environment can affect the performance of the system. For example, an autonomous vehicle may encounter unexpected weather conditions or road construction that can impact its ability to navigate safely.

Human Uncertainty: AI systems often interact with humans, either as users or as part of the decision-making process. Human behaviour and preferences can be difficult to predict, leading to uncertainty in the use and adoption of AI systems.

Ethical Uncertainty: AI systems often raise ethical concerns, such as privacy, bias, and transparency. These concerns can lead to uncertainty in the development and deployment of AI systems, particularly in regulated industries.

Legal Uncertainty: AI systems must comply with laws and regulations, which can be ambiguous or unclear. Legal challenges and disputes can arise from the use of AI systems, leading to uncertainty in their adoption and implementation.

Uncertainty in AI Reasoning: AI systems use reasoning techniques to make decisions or predictions. However, these reasoning techniques can be uncertain due to the complexity of the problems they address or the limitations

of the data used to train the models.

Uncertainty in AI Perception: AI systems perceive their environment through sensors and cameras, which can be subject to noise, occlusion, or other forms of interference. This can lead to uncertainty in the accuracy of the data used to train AI models or the effectiveness of AI systems in real-world applications.

Uncertainty in AI Communication: AI systems communicate with humans through natural language processing or computer vision. However, language and visual cues can be ambiguous or misunderstood, leading to uncertainty in the effective communication between humans and AI systems.

Types of Uncertainty in AI

Aleatoric Uncertainty: This type of uncertainty arises from the inherent randomness or variability in data. It is often referred to as "data uncertainty." For example, in a classification task, aleatoric uncertainty may arise from variations in sensor measurements or noisy labels.

Epistemic Uncertainty: Epistemic uncertainty is related to the lack of knowledge or information about a model. It represents uncertainty that can potentially be reduced with more data or better modelling techniques. It is also known as "model uncertainty" and arises from model limitations, such as simplifications or assumptions.

Parameter Uncertainty: This type of uncertainty is specific to probabilistic models, such as Bayesian neural networks. It reflects uncertainty about the values of model parameters and is characterized by probability distributions over those parameters.

Uncertainty in Decision-Making: Uncertainty in AI systems can affect the decision-making process. For instance, in reinforcement learning, agents often need to make decisions in environments with uncertain outcomes, leading to decision-making uncertainty.

Uncertainty in Natural Language Understanding: In natural language processing (NLP), understanding and generating human language can be inherently uncertain due to language ambiguity, polysemy (multiple meanings), and context-dependent interpretations.

Probability Notation

- Probabilistic notation refers to the symbols and conventions used to represent and manipulate probabilities and statistical concepts.
- This notation is fundamental in fields such as statistics, machine learning, and artificial intelligence

Basic Probabilistic Notations

Here are some key elements of probabilistic notation, which form the foundation for more advanced probabilistic models in AI:

Probability Notation:

Probability Notation	Description
$P(A)$	The probability of event A occurring

Probability Notation	Description
$P(A')$	The probability of event A not occurring
$P(A \cap B)$	The probability of both A and B occurring at the same time
$P(A \cup B)$	The probability of either A or B occurring
$P(A \cap B')$	The probability of A occurring but not B
$P(A' \cup B)$	The probability of either A not occurring or B occurring

Conditional Probability:

- **P(A | B)**: The probability of event A occurring given that event B has occurred. This is fundamental in AI for updating beliefs based on new evidence.
- **Bayes' Theorem**: $P(A|B)=P(B)P(B|A)\cdot P(A)P(A|B)=P(B)P(B|A)\cdot P(A)$, which provides a way to update probabilities based on new data.

Joint Probability:

- The probability of both A and B occurring, which can also be written as $P(A \cap B)$. This is essential for understanding the relationships between multiple variables.

Marginal Probability:

- The probability of event A **P(A)** occurring, regardless of other events. This is derived by summing or integrating over the joint probabilities of A with all other possible events.

Advanced Probabilistic Notations

Random Variables:

- **X**: A random variable representing a possible outcome.
- **P(X = x)**: The probability that the random variable X takes the value x.
- **P(X ≤ x)**: The probability that the random variable X takes a value less than or equal to x.

Probability Distributions:

- **Probability Mass Function (PMF)**: For discrete random variables, $P(X=x)$ denotes the PMF.
- **Probability Density Function (PDF)**: For continuous random variables, $fX(x)$ denotes the PDF.
- **Cumulative Distribution Function (CDF)**: $FX(x)=P(X\leq x)FX(x)=P(X\leq x)$ gives the cumulative probability up to x.

Expectation and Variance:

- **E[X]**: The expected value or mean of the random variable X.
- **Var(X)**: The variance of the random variable X, representing the spread of its possible values.

Covariance and Correlation:

- **Cov(X, Y):** The covariance between random variables X and Y, indicating the degree to which they change together.
- **Corr(X, Y)**: The correlation coefficient between X and Y, a normalized measure of their linear relationship.

Inference Using Full Joint Distributions

- Joint probability offers valuable insights into the likelihood of multiple events happening together. This helps us in several ways:

Co-occurrence: Joint probability helps us understand how likely it is for two or more events to happen at the same time. This is important for seeing how events are connected and the probability of them occurring together.

Risk Evaluation: In areas like finance and insurance, joint probability helps us assess the risk when multiple events overlap. For instance, it can estimate the chance of multiple financial instruments facing losses simultaneously.

Quality Check: Businesses can use joint probability to gauge the reliability and quality of their products or processes. It shows the likelihood of multiple defects or issues occurring at once, which allows for proactive quality improvement efforts.

Event Relationships: Joint probability can indicate if events are related or not. If joint probability significantly differs from the product of individual probabilities, it suggests events are connected, and the occurrence of one affects the likelihood of the other.

Decision Support: When businesses need to make choices involving multiple factors or events, joint probability provides a numerical foundation for decision-making. It helps assess how different variables together impact the desired outcome.

Resource Management: In situations with limited resources, understanding joint probability helps optimise resource allocation. For example, in supply chain management, it can estimate the chance of multiple supply chain disruptions happening at the same time, enabling better risk management strategies.

Formula for Joint Probability

For Independent Events

When events A and B are independent, meaning that the occurrence of one event does not impact the other, we use the multiplication rule:

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

Here, P(A) is the probability of occurrence of event A, P(B) is the probability of occurrence of event B, and $P(A \cap B)$ is the joint probability of events A and B.

For Dependent Events

Events are often dependent on each other, meaning that one event's occurrence influences the likelihood of the other. Here, we employ a modified formula:

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

Here, P(A) is the probability of occurrence of event A, P(B|A) is the conditional probability of occurrence of event B when event A has already occurred, and $P(A \cap B)$ is the joint probability of events A and B.

Bayesian Classification:

Bayesian classification:-

Bayesian classifiers are statistical classifiers. They can predict class membership probabilities such as the probability that a given sample belongs to a particular class.

Bayesian classification Based on Bayes theorem

Baye's theorem:

$$P(c|x) = \frac{P(x|c) \cdot P(c)}{P(x)}$$

It finds posterior probability and posterior probability.

- If find the posterior probability of a class conditional.

Algorithm:-

step 1:-

* let a 'D' be a training dataset of data tuple associated with class labels.

step 2:-

* each tuple is represented by an n-dimensional vector $x = (x_1, x_2 \ldots x_n)$ where $x_1, x_2 \ldots x_n$ are values of attributes.

Suppose that there are m no. of classes $C_1, C_2 \ldots C_m$ given a data tuple x the classifier is predict that $x \in$ the class having the highest posterior probability condition on x.

This can be written mathematically where $j = 1, 2, \ldots m$ and $i \neq j$.

i.e, we maximize $P(C_i|x)$

Bayes theorem $P(C_i|x) = \frac{P(x|C_i)\,P(C_i)}{P(x)}$

step 3:-

As $P(x)$ is constant for all classes it is enough to maximize only that is numerical function has $P(x|C_i)$.

* If the class probability is not known.

$P(C_1) = P(C_2) = \cdots = P(C_m)$

so it is enough to maximize only $P(x/C_i)$

step 4:-

$$P(x/C_i) = \prod_{k=1}^{m} P(x_k/C_i)$$

$$= P(x_1/C_i) \times P(x_2/C_i) \times \cdots \times P(x_m/C_i)$$

(i) If A_k categorical $P(x_k/C_i)$ is number of tuples of class C_i in 'D' having the value small S_k for A_k / no. of tuples of C_i in D.

(ii) A_k is continuous $g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma C_i} \times e^{\frac{-(x-\mu)^2}{2\sigma^2}}$

where $g(x, \mu, \sigma)$ is the Gaussian (normal) density function for attribute A_k while μ and σ are the mean and standard deviation, respectively given the values for attributes A_k for training samples of class C.

step 5:-

The classifier predict the class lable is C_i if and only if $P(x/C_i) P(C_i) > P(x/C_j) \cdot P(C_j)$ for $1 \le j \le m$, $j \neq i$

Problem :-

RID	Age	Income	student	credit rating	buys Computer.
1	youth	high	NO	Avg	NO
2	"	"	"	Exce.	"
3	middle	"	"	Avg	yes
4	senior	medium	"	"	"
5	senior	low	yes	"	"
6	senior	"	yes	Exce	NO
7	middle	"	yes	"	yes
8	youth	medium	yes NO	Avg	NO
9	youth	low	NO yes	"	yes
10	senior	medium	yes	"	"
11	youth	"	"	Exce	"
12	middle	"	NO	Exce	"
13	middle	high	yes	Avg	yes
14	senior	medium	NO	Exce	NO

X = (age = "<= 30", income = "medium", student = "yes", credit rating = "fair / avg.") predict the class labeled data tuple 'x' using bayesian classification algorithm.

Sol :- The posion probability of each class.

x

$P(\text{buys\_computer} = \text{"yes"}) = \frac{9}{14} = 0.6428$

$P(\text{buys\_computer} = \text{"NO"}) = \frac{5}{14} = 0.3571$

young :-

$P(\text{age} <= 30 \mid \text{buys\_computer} = \text{"yes"}) = \frac{2}{9} = 0.222$

$P(\text{age} <= 30 \mid \text{buys computer} = \text{"NO"}) = \frac{3}{5} = 0.600$

medium :-

$P(\text{income} = \text{"medium"} \mid \text{buys computer} = \text{"yes"}) = \frac{4}{9} = 0.444$

$P(\text{income} = \text{"medium"} \mid \text{buys computer} = \text{"NO"}) = \frac{2}{5} = 0.40$

student :-

$P(\text{student} = \text{"yes"} \mid \text{buys computer} = \text{"yes"}) = \frac{6}{9} = 0.666$

$P(\text{student} = \text{"yes"} \mid \text{buys computer} = \text{"NO"}) = \frac{1}{5} = 0.200$

credit - rating :-

$P(\text{credit\_rating} = \text{"avg"} \mid \text{buys computer} = \text{"yes"}) = \frac{6}{9} = 0.666$

$P(\text{"} = \text{"avg"} \mid \text{buys computer} = \text{"NO"}) = \frac{2}{5} = 0.400$

using the above probabilities :-

$P(X \mid \text{buys\_computer} = \text{"yes"}) = 0.222 \times 0.444 \times 0.666 \times 0.666$

$= 0.0437$

$P(X \mid \text{buys computer} = \text{"NO"}) = 0.600 \times 0.400 \times 0.200 \times 0.400$

$= 0.019$

Case Study: Online Shopping Recommendation Under Uncertainty

Consider an online shopping agent that must decide whether to recommend a product to a customer. The agent has uncertain information such as:

- whether the customer likes the product category,
- whether the product is in stock,
- whether the user is likely to buy.

The ontology part of the system represents categories such as Customer, Product, and Order. The probabilistic part estimates the chance of purchase using prior data and Bayes-style updating. This case study shows how **ontological engineering** and **uncertainty reasoning** can work together in a practical AI system, matching the Internet shopping theme highlighted in AIMA's ontology coverage and the standard probabilistic reasoning examples used later in the book

Unit Highlights

- **Ontological engineering** helps represent the structure of a domain.
- **Categories and objects** are central to structured knowledge representation.
- **Events** represent actions and changes in the world.
- Reasoning systems for categories support **inheritance** and classification.
- **Default reasoning** allows agents to handle typical knowledge with exceptions.
- The **Internet Shopping World** is a practical ontology example.
- **Probability** is used to represent and reason under uncertainty.
- **Full joint distributions**, **independence**, and **Bayes' Rule** are key tools for probabilistic inference.

TEXT BOOKS:

1. *Artificial Intelligence A Modern Approach, 2015, Third Edition , Stuart J. Russell and Peter Norvig Pearson Education.*

REFERENCE BOOKS:

1. *Artificial Intelligence, 3/e,2009, Elaine Rich, Kevin Knight and Shiva shankar B Nair Tata McGraw Hill.*
2. *Artificial Intelligence-Structures and Strategies for Complex Problem Solving, 5/e, 2009, George F. Luther, Pearson Education.*
3. *Introduction to Artificial Intelligence, Eugene Charniak and Drew McDermott, Pearson Education, 1987.*

S.No	Questions	Blooms Taxonomy Level
UNIT V – KNOWLEDGE REPRESENTATION:		
PART-A (Two Marks Questions)		
1.	What is ontological engineering in AI?	L1
2.	Define 'categories' and 'objects' in knowledge representation.	L1
3.	What are 'events' in the context of AI?	L1
4.	Explain reasoning systems for categories.	L1
5.	What is reasoning with default information?	L2
6.	Describe the 'Internet Shopping World' in AI.	L2
7.	What does 'acting under uncertainty' mean in AI?	L2
8.	Define basic probability notation.	L2
9.	What is a full joint probability distribution?	L2
10.	Explain the concept of independence in probability.	L2
11.	What is Bayes' Rule and its significance in AI?	L2
PART-B (Ten Marks Questions)		
1.	Explain the concept of ontological engineering in Artificial Intelligence.	L2
2.	Discuss the role of categories and objects in knowledge representation. How do they facilitate reasoning in AI systems?	L4
3.	What are events in the context of AI, and how are they represented and utilized in reasoning systems?	L3
4.	Explain reasoning systems for categories. How do they support inference in AI applications?	L4
5.	Describe reasoning with default information. How do AI systems handle exceptions to default assumptions?	L5
6.	Illustrate the concept of the 'Internet Shopping World' as a domain in AI. How does ontological engineering apply to this domain?	L3
7.	What does 'acting under uncertainty' entail in AI, and why is it significant?	L2
8.	Explain the concept of a full joint probability distribution. How is it utilized in AI inference?	L2
9.	What is Bayes' Rule, and how is it applied in AI systems for updating beliefs?	L4