Name: Prathamesh Arvind Jadhav

Unit-1-Introduction

1.Explain Types of Robot and Technologies: Robot classification and specifications, controls and Operations.

Types of Robots and Technologies: Classification, Specifications, Controls, and Operations

Robots are **automated machines** designed to perform tasks with minimal human intervention. They are classified based on **functionality, movement, application, and technology**.

---

1. Types of Robots & Classification

A. Based on Mobility

1. **Stationary Robots**
   - Fixed in one place, commonly used in industries.
   - Example: **Robotic arms in car manufacturing**.
2. **Mobile Robots**
   - Move autonomously or semi-autonomously.
   - Example: **Self-driving cars (Tesla, Waymo), warehouse robots (Amazon Kiva)**.
3. **Humanoid Robots**
   - Resemble human form and movements.
   - Example: **Sophia (AI-powered humanoid robot), ASIMO by Honda**.
4. **Aerial Robots (Drones)**
   - Used for surveillance, delivery, and photography.
   - Example: **DJI Phantom, military UAVs**.
5. **Underwater Robots (ROVs & AUVs)**
   - Used for deep-sea exploration, pipeline inspections.
   - Example: **BOEING Echo Voyager (AUV), ROV Jason**.

B. Based on Application

1. **Industrial Robots**

- Used in **factories** for automation.
- Example: **Assembly-line robots in Tesla factories**.

2. **Medical Robots**
   - Assist in **surgery, rehabilitation, and patient care**.
   - Example: **Da Vinci Surgical System, Exoskeleton robots**.
3. **Service Robots**
   - Perform household or commercial tasks.
   - Example: **Roomba vacuum cleaner, hotel service robots**.
4. **Military & Defense Robots**
   - Used for **bomb disposal, surveillance, and combat**.
   - Example: **Boston Dynamics' BigDog, PackBot by iRobot**.
5. **Agricultural Robots**
   - Used for **harvesting, spraying pesticides, and monitoring crops**.
   - Example: **Agrobot for strawberry picking, drone farming**.

---

2. Robot Specifications

Each robot has specific technical parameters that define its functionality:

Specification	Description
Degrees of Freedom (DOF)	Number of independent movements (e.g., robotic arm has 6-7 DOF).
Payload Capacity	Maximum weight a robot can carry.
Speed	Movement speed (measured in m/s or RPM for robotic arms).
Accuracy & Precision	Ability to perform repetitive tasks with exact positioning.
Power Source	Battery-powered, hydraulic, pneumatic, or electric.
Sensors & AI	LIDAR, cameras, gyroscopes for navigation and automation.

---

3. Robot Controls & Operations

A. Control Methods

1. **Manual Control** – Human operates via joystick or remote (e.g., ROVs).

2. **Pre-Programmed Control** – Robot follows predefined instructions (e.g., industrial robots).
3. **Autonomous Control** – AI and sensors enable decision-making (e.g., self-driving cars).
4. **Teleoperation** – Controlled remotely via wireless connection (e.g., drones, surgical robots).
5. **Hybrid Control** – Combination of AI and human control (e.g., collaborative robots in factories).

B. Operation Technologies

1. **Artificial Intelligence (AI) & Machine Learning** – Enables robots to learn and adapt.
2. **Computer Vision** – Helps in object detection, face recognition (used in security robots).
3. **IoT (Internet of Things)** – Connects robots with cloud and sensors for remote monitoring.
4. **Haptic Feedback** – Allows surgeons to "feel" resistance in robotic surgery.
5. **Swarm Robotics** – Multiple robots working together (used in **drone fleets**).

2.Describe the Functions of Sensors in robotics. what are the most commonly used sensors in robotics applications.

Functions of Sensors in Robotics & Commonly Used Sensors

Sensors play a crucial role in robotics by enabling **perception, decision-making, and interaction** with the environment. They collect data and allow robots to respond intelligently to their surroundings.

---

1. Functions of Sensors in Robotics

1. Perception & Environment Awareness

- Sensors help robots understand their surroundings by detecting **obstacles, light, temperature, and motion**.
- Example: **LIDAR sensors in self-driving cars detect nearby vehicles and pedestrians.**

2. Navigation & Localization

- Used in **autonomous robots** to determine their position and movement.
- Example: **GPS sensors help drones navigate precise locations.**

3. Object Detection & Recognition

- Helps robots identify and classify objects.
- Example: **Computer vision cameras detect items in a warehouse for sorting.**

4. Motion Control & Feedback

- Measures speed, rotation, and position to control robotic arms and wheels.
- Example: **Encoders in robotic arms ensure accurate movement.**

5. Human-Robot Interaction

- Touch and pressure sensors allow robots to interact safely with humans.
- Example: **Haptic sensors in prosthetic hands provide touch sensitivity.**

---

2. Commonly Used Sensors in Robotics

1. Vision & Imaging Sensors

- **Cameras** – Used for object detection, face recognition. *(Example: Surveillance robots)*
- **LIDAR (Light Detection and Ranging)** – Measures distances using laser beams. *(Example: Self-driving cars)*

2. Proximity & Distance Sensors

- **Ultrasonic Sensors** – Detect obstacles using sound waves. *(Example: Obstacle avoidance in cleaning robots)*
- **Infrared Sensors (IR)** – Detect heat and motion. *(Example: Gesture control in smart robots)*

3. Motion & Position Sensors

- **Encoders** – Measure the rotation of wheels or arms. *(Example: Robotic arms in assembly lines)*
- **Gyroscope & Accelerometer** – Detect balance and movement. *(Example: Humanoid robots, drones)*

4. Touch & Force Sensors

- **Tactile Sensors** – Detect physical contact and pressure. *(Example: Medical robots, robotic grippers)*
- **Strain Gauges** – Measure force applied on a surface. *(Example: Prosthetic limbs)*

5. Environmental Sensors

- **Temperature Sensors** – Monitor heat levels. *(Example: Industrial robots working in extreme conditions)*
- **Gas Sensors** – Detect harmful gases. *(Example: Firefighting robots, air quality monitoring drones)*

3.Explain the Structure of an intelligent agent in robotics and describe types of agents.

Structure of an Intelligent Agent in Robotics & Types of Agents

1. What is an Intelligent Agent?

An **Intelligent Agent (IA)** in robotics is a system that **perceives its environment, processes information, and takes actions** to achieve specific goals. It uses **sensors for perception**, **actuators for movement**, and **AI/algorithms for decision-making**.

---

2. Structure of an Intelligent Agent in Robotics

An intelligent agent consists of **four main components**:

1. Sensors (Perception Unit)

- Collects data from the environment.

- Example: **Cameras, LIDAR, ultrasonic sensors.**

2. Actuators (Action Unit)

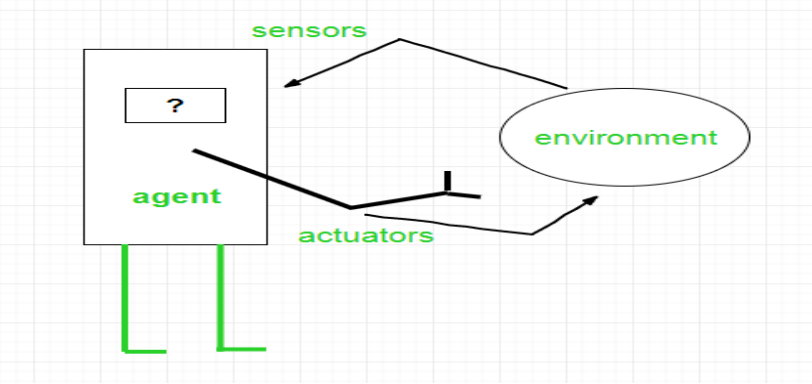
- Executes actions based on decisions made by the agent.
- Example: **Motors, robotic arms, wheels.**

3. Processor (Brain of the Agent)

- Includes **AI algorithms, decision-making models, and logic-based processing**.
- Example: **Path planning in self-driving cars using AI.**

4. Knowledge Base (Memory & Learning)

- Stores past experiences and learning models to improve future decisions.
- Example: **Machine learning in robotic assistants like Alexa.**



Types of Agents in Robotics

Intelligent agents in robotics **perceive their environment, process information, and take actions** based on their design and purpose. They are classified based on **their ability to store past data, use decision-making models, and learn from experiences**.

---

1. Simple Reflex Agents

Definition:

- These agents act **only based on current sensor inputs** without storing any past information.
- They follow **predefined rules** and do not adapt to new situations.

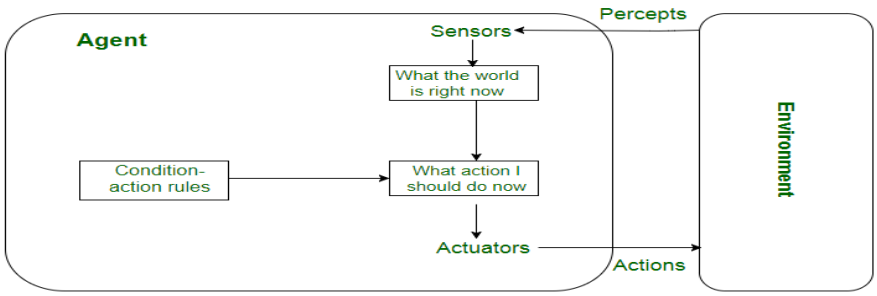
How It Works:

- Uses **condition-action rules**:
  - **IF (Obstacle Detected) → THEN (Change Direction)**
- Does not consider previous actions or the future consequences.

Example:

 Obstacle-Avoidance Robots

- Uses **infrared sensors or ultrasonic sensors** to detect obstacles.
- When it detects an obstacle, it **immediately changes direction** but does not remember the previous path.
- Used in **autonomous vacuum cleaners (e.g., Roomba), self-parking cars, warehouse robots**.



---

2. Model-Based Reflex Agents

Definition:

- Unlike simple reflex agents, **these agents maintain a partial model of the environment**.
- They store **some history** and **internal knowledge** to handle more complex situations.

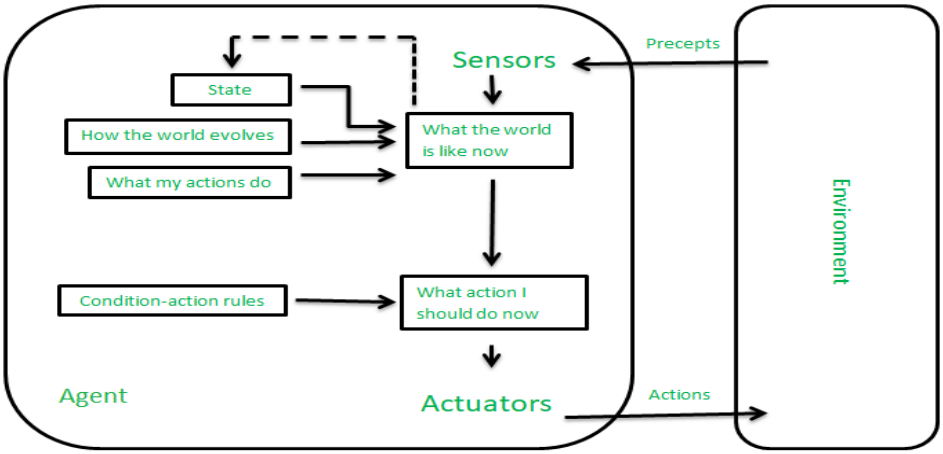
How It Works:

- Uses a **world model** to track changes.
- Updates its internal memory based on sensor inputs.
- Uses **both current and past information** to make decisions.

Example:

 Autonomous Drones Adjusting to Wind Conditions

- Uses **gyroscope, accelerometer, and GPS** to track position.
- If strong winds are detected, the drone **adjusts its movement** to stay on course.
- Used in **Amazon delivery drones, military surveillance drones**.



---

3. Goal-Based Agents

Definition:

- These agents **not only react** but also work towards a **specific goal**.
- They **evaluate multiple options** to choose the best one.

How It Works:

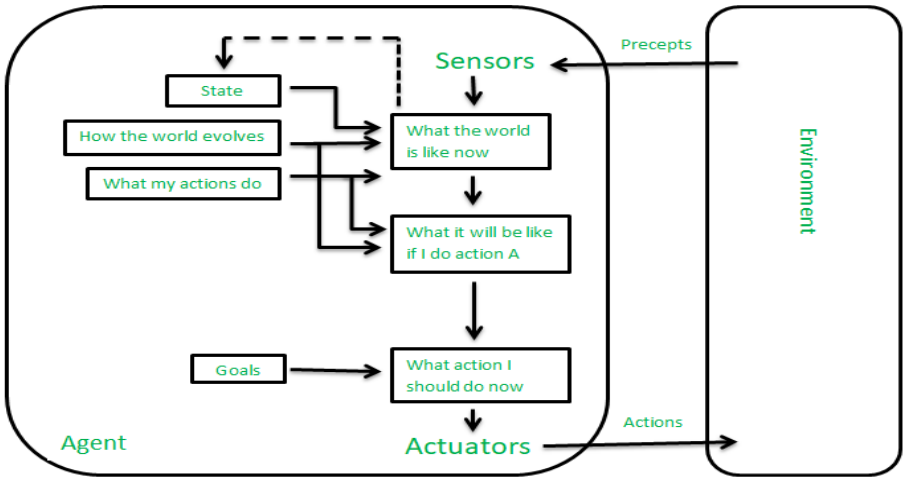
- Uses **search algorithms and AI** to determine the best path.
- Considers **current state + future goals** to make decisions.
- May use **machine learning** to improve decision-making over time.

Example:

Name: Prathamesh Arvind Jadhav

□ Self-Driving Cars Planning the Shortest Route

- Uses **Google Maps, GPS, LIDAR, and AI** to analyze traffic conditions.
- Chooses the **fastest route** to reach the destination.
- Adjusts based on **real-time traffic updates**.
- Example: **Tesla Autopilot, Waymo self-driving taxis**.



---

4. Utility-Based Agents

Definition:

- These agents make decisions based on a **performance measure (utility function)**.
- Instead of just achieving a goal, they find the **most optimal way** to reach it.

How It Works:

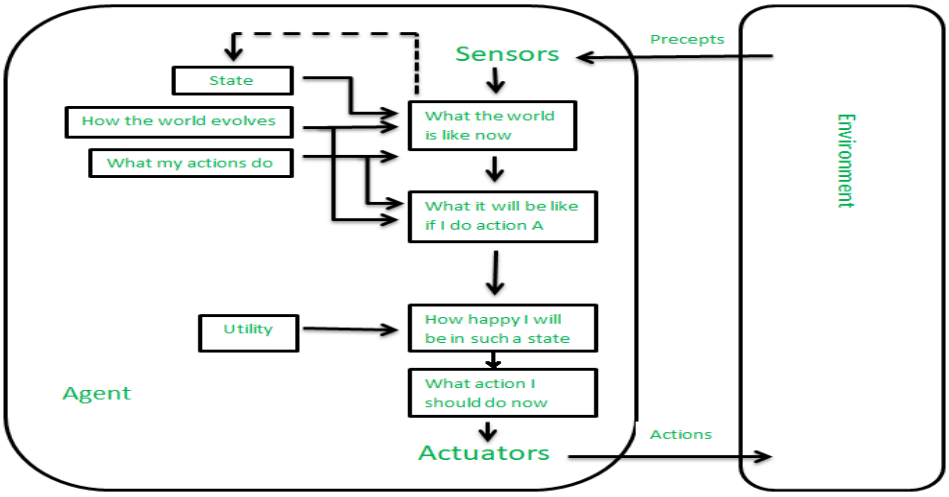
- Uses **mathematical models** to evaluate multiple outcomes.
- Tries to **maximize benefits and minimize risks**.
- Often used in **financial markets, gaming, and business analytics**.

Example:

□ AI Trading Bots Maximizing Stock Profits

- Analyzes **stock market trends, historical data, and real-time news**.

- Decides **when to buy/sell stocks** for maximum profit.
- Example: **High-frequency trading (HFT) systems, cryptocurrency trading bots**.



---

5. Learning Agents

Definition:

- These agents **learn from past experiences** and improve their performance over time.
- Uses **machine learning, deep learning, and neural networks**.

How It Works:

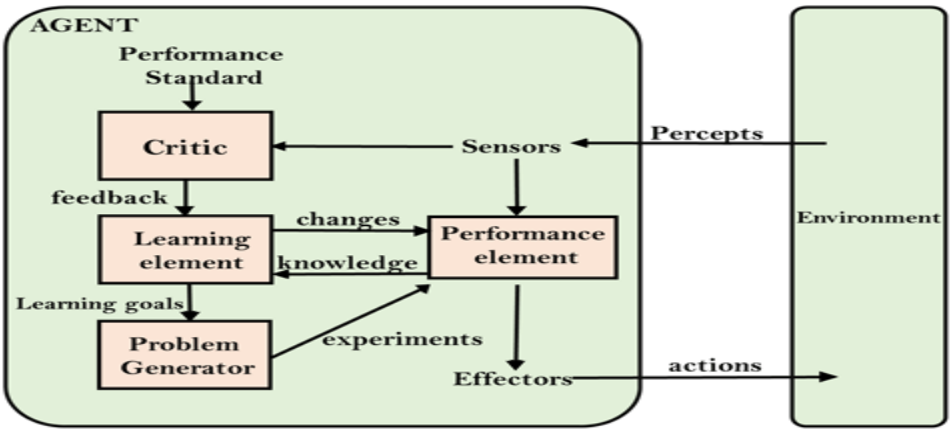
- Uses **feedback mechanisms** (reinforcement learning).
- Updates **its knowledge base** after every decision.
- Can adapt to **new, unseen scenarios**.

Example:

 AI-Powered Humanoid Robots (Sophia, Tesla Optimus)

- Uses **AI + NLP (Natural Language Processing)** to talk and understand humans.
- Learns from **conversations, emotions, and interactions**.

- Example: **Sophia (AI Robot by Hanson Robotics), Tesla Optimus (AI-powered humanoid assistant)**.



4.Compare and contrast the various classification of robots based on their specification and functionalities.

Comparison of Robot Classifications Based on Specification & Functionality

Robots are classified based on **specifications, functionalities, and operational environments**. The major classifications include **industrial, service, mobile, humanoid, and autonomous robots**. Below is a detailed comparison of these categories.

---

1. Classification of Robots

Type of Robot	Specifications	Functionality	Examples
1. Industrial Robots	High precision, fixed structure, multiple DOF (Degrees of Freedom)	Used for **assembly, welding, material handling** in industries	Robotic Arms in car manufacturing (e.g., FANUC, ABB Robots)
2. Service	Semi-autonomous,	Assists in **healthcare,**	Automated vacuum

Name: Prathamesh Arvind Jadhav

Type of Robot	Specifications	Functionality	Examples
Robots	works in non-industrial settings	**cleaning, security**	cleaners (Roomba), Medical robots (Da Vinci)
3. Mobile Robots	Self-moving, uses wheels/tracks or legs	Navigates dynamically, used in **warehouse, delivery, space missions**	AGVs (Amazon warehouse robots), Mars Rover
4. Humanoid Robots	Resembles human shape, AI-powered, uses NLP	Interacts with humans, used in **education, entertainment, customer service**	Sophia, ASIMO, Tesla Optimus
5. Autonomous Robots	AI-based, decision-making capability, sensors-driven	Performs tasks **independently**, used in **drones, self-driving cars**	Tesla Autopilot, DJI Drones, Boston Dynamics' robots

---

2. Key Differences Between Robot Classifications

Feature	Industrial Robots	Service Robots	Mobile Robots	Humanoid Robots	Autonomous Robots
Mobility	Stationary	Limited	High	Medium	Very High
AI Integration	Low	Medium	High	Very High	Very High
Interaction	None	Human Assistance	Sensors-Based	Human-Like	Fully AI-Controlled

Feature	Industrial Robots	Service Robots	Mobile Robots	Humanoid Robots	Autonomous Robots
Autonomy	Programmed	Semi-Autonomous	Fully Autonomous	AI-Based	Fully AI-Controlled
Usage	Manufacturing	Healthcare, Security	Warehouses, Space	Education, Service	Self-driving Vehicles, Smart Assistants

---

3. Comparison Based on Functionality

1. Industrial Robots vs. Service Robots

- **Industrial Robots**: Focus on **precision, speed, and repetitive tasks** in controlled environments.
- **Service Robots**: More flexible, work in dynamic environments with human interaction.

2. Mobile Robots vs. Autonomous Robots

- **Mobile Robots**: Require external commands to function.
- **Autonomous Robots**: Use **AI and sensors** to make independent decisions.

3. Humanoid Robots vs. Other Robots

- **Humanoid Robots**: Have **human-like appearance and behavior** for social interaction.
- **Other Robots**: Designed for specific tasks without human-like behavior.

Name: Prathamesh Arvind Jadhav

5.Describe the function of sensors in robotics and how they contributes to work cell design.

Functions of Sensors in Robotics and Their Contribution to Work Cell Design

1. Introduction to Sensors in Robotics

Sensors are essential components in **robotics** that enable robots to **perceive, analyze, and interact with their environment**. They collect data from the surroundings and provide **feedback** to improve precision, safety, and automation in robotic operations.

---

2. Functions of Sensors in Robotics

Function	Description	Example Sensors
1. Perception	Helps robots recognize objects, colors, and motion	Camera, LiDAR, RGB sensors
2. Object Detection & Avoidance	Detects obstacles to navigate safely	Ultrasonic, Infrared, Proximity sensors
3. Positioning & Localization	Determines robot's exact position in space	GPS, Gyroscope, IMU (Inertial Measurement Unit)
4. Force & Pressure Sensing	Measures applied force during operations	Tactile sensors, Strain gauges
5. Temperature Monitoring	Prevents overheating and ensures safety	Thermal sensors, Thermocouples
6. Feedback Control	Adjusts movements based on sensor inputs	Encoders, Hall effect sensors

---

3. Contribution of Sensors to Work Cell Design

Name: Prathamesh Arvind Jadhav

A **work cell** in robotics refers to a **dedicated area where robots perform automated tasks** (e.g., **manufacturing, packaging, assembly lines**). Sensors play a critical role in optimizing work cell design by ensuring **efficiency, accuracy, and safety**.

Contribution	Description	Examples
1. Enhanced Safety	Detects human presence to prevent accidents	Proximity sensors, Motion detectors
2. Precision in Task Execution	Ensures accuracy in assembly & production	Vision sensors, Laser scanners
3. Automated Quality Control	Identifies defects or inconsistencies	Camera-based inspection, Pressure sensors
4. Efficient Workflow Management	Helps in **scheduling & material movement**	RFID sensors, Barcode scanners
5. Energy Optimization	Monitors power consumption & reduces waste	Smart temperature & load sensors

---

4. Real-World Applications of Sensors in Work Cell Design

- **Automobile Manufacturing** → **Vision sensors** inspect defects in car assembly.
- **Food Processing** → **Thermal sensors** monitor temperature in food packaging.
- **Warehouses** → **LiDAR & Ultrasonic sensors** guide autonomous robots for inventory management.
- **Healthcare Robots** → **Tactile sensors** assist robotic surgery for precision.

Name: Prathamesh Arvind Jadhav

Unit-2-Problem Solving

1.Explain Solving problems by searching: Informed search and exploration and uninformed Search with example.

Solving Problems by Searching

Searching is a fundamental technique in Artificial Intelligence (AI) used to find solutions in a problem space. The search algorithms are categorized into **Uninformed Search** (Blind Search) and **Informed Search** (Heuristic Search).

---

1. Uninformed Search (Blind Search)

Uninformed search algorithms do not have prior knowledge about the goal state and explore the search space blindly. They rely only on the given problem definition (initial state, actions, and goal test).

Types of Uninformed Search Algorithms:

1. **Breadth-First Search (BFS)** – Explores all nodes at the current level before moving to the next level.
2. **Depth-First Search (DFS)** – Explores as deep as possible along one branch before backtracking.
3. **Uniform Cost Search (UCS)** – Expands the node with the lowest path cost.
4. **Depth-Limited Search (DLS)** – A variation of DFS with a depth limit.
5. **Iterative Deepening DFS (IDDFS)** – Repeatedly applies DFS with increasing depth limits.

Example:

Consider a **maze-solving problem** where a robot must find the shortest path from start to the goal.

- **BFS** would explore all possible paths level by level, ensuring the shortest path is found.
- **DFS** might take a longer route by going deep into one path first.

---

2. Informed Search (Heuristic Search)

Informed search uses additional information (heuristics) to guide the search process toward the goal efficiently. It reduces the number of explored nodes and improves efficiency.

Types of Informed Search Algorithms:

1. **Greedy Best-First Search** – Uses a heuristic function h(n) to choose the best next move.
2. *A Search** – Uses both the heuristic function h(n) and the cost-so-far g(n), making it optimal and complete.
3. **Hill Climbing** – Moves towards the best immediate neighbor (may get stuck in local optima).
4. **Beam Search** – Expands only a limited number of best nodes at each level.

Example:

Consider the **8-puzzle problem**, where tiles must be arranged in order.

- **A*** search can efficiently find the shortest path to the goal using heuristics like the **Manhattan Distance** (sum of tile movements needed to reach the goal state).

Comparison Table:

Feature	Uninformed Search	Informed Search
Uses heuristics?	No	Yes
Efficiency	Less efficient	More efficient
Example Algorithms	BFS, DFS, UCS	A*, Greedy BFS
Best Use Case	When no heuristic is available	When heuristic information is available

Name: Prathamesh Arvind Jadhav

2.explain Constraint satisfaction problems with examples.

Constraint Satisfaction Problems (CSPs)

A **Constraint Satisfaction Problem (CSP)** is a mathematical problem defined by a set of **variables**, a **domain** of possible values for each variable, and a set of **constraints** that restrict the values the variables can take. The goal is to find an assignment of values to variables that satisfies all constraints.

1. Components of CSP:

A CSP consists of three main components:

1. **Variables (X)**: A set of variables $X = \{X_1, X_2, ..., X_n\}$.
2. **Domains (D)**: Each variable X_i has a domain D_i of possible values.
3. **Constraints (C)**: A set of constraints that specify allowable combinations of values for subsets of variables.

2. Types of CSPs:

1. **Binary CSP:** Constraints involve two variables at a time.
2. **Higher-order CSP:** Constraints involve more than two variables.
3. **Continuous CSP:** Domains are continuous instead of discrete.

Name: Prathamesh Arvind Jadhav

3. Example CSPs:

Example 1: Sudoku Puzzle

- **Variables:** Each cell in the 9×9 grid.
- **Domain:** Numbers {1, 2, 3, 4, 5, 6, 7, 8, 9}.
- **Constraints:**
  - Each row must contain unique numbers (1-9).
  - Each column must contain unique numbers (1-9).
  - Each 3×3 sub-grid must contain unique numbers (1-9).

Example 3: N-Queens Problem

- **Variables:** Positions of N queens on an $N \times N$ chessboard.
- **Domain:** Each queen can be placed in one of the N rows.
- **Constraints:** No two queens should be in the same row, column, or diagonal.

Solution Approach:

- Using **Backtracking** or **Constraint Propagation (Arc Consistency, Forward Checking)**.

4. CSP Solving Techniques:

1. **Backtracking Search:** Tries assigning values recursively and backtracks when constraints are violated.
2. **Constraint Propagation (e.g., Arc Consistency - AC-3 Algorithm):** Reduces the search space by enforcing local consistency.
3. **Local Search (e.g., Min-Conflicts Algorithm):** Starts with an initial solution and iteratively improves it.

---

5. Real-World Applications of CSP:

- **Scheduling problems** – Employee shift scheduling, university timetabling.
- **Graph coloring** – Register allocation in compilers.
- **Puzzle solving** – Sudoku, crossword puzzles.
- **Circuit design** – Checking layout constraints in VLSI design.

3.explain Adversarial search with example.

Adversarial Search

Adversarial search is a type of search used in competitive environments where multiple agents (players) have **conflicting goals**. It is commonly used in **game-playing AI**, where one player's gain is another player's loss.

---

1. Characteristics of Adversarial Search:

- **Involves two or more players** competing against each other.
- **Zero-sum games**, meaning one player's gain is another's loss.
- **Uses a game tree** to explore possible moves.
- **Decision-making involves maximizing own advantage while minimizing the opponent's advantage.**

---

2. Example of Adversarial Search – Tic-Tac-Toe

Consider a **Tic-Tac-Toe** game where two players (X and O) compete to win by placing marks on a 3×3 grid. The AI must choose the best move by exploring all possible future states.

- **Game tree:** Each node represents a possible game state.
- **Maximizing player (X):** Tries to **maximize** the chance of winning.
- **Minimizing player (O):** Tries to **minimize** the maximizing player's chances.

Solution Approach:

The **Minimax Algorithm** is used to determine the best move.

---

3. Minimax Algorithm (Basic Adversarial Search Algorithm)

The **Minimax Algorithm** is a recursive strategy to determine the best move by assuming that the opponent plays optimally.

Steps:

1. **Generate the game tree** up to a certain depth.
2. **Assign scores** to terminal states (+1 for a win, -1 for a loss, 0 for a draw).
3. **Backpropagate the values**:
    - Maximizing player (AI) selects the maximum value.
    - Minimizing player (opponent) selects the minimum value.
4. **AI picks the best move** by choosing the branch with the highest minimax value.

Example in Tic-Tac-Toe:

If the AI (X) has two choices:

1. Moving to a state where it wins (+1).
2. Moving to a state where the opponent (O) can win (-1).
   It will choose **option 1** because it maximizes its chance of winning.

---

4. Alpha-Beta Pruning (Optimization of Minimax)

- **Alpha-Beta Pruning** improves Minimax by **eliminating** unnecessary branches of the game tree that won't be selected.
- This **reduces computation time** without affecting the final decision.

Example in Chess:

Instead of evaluating every possible move, Alpha-Beta Pruning **ignores branches** that will never be chosen by the opponent, making the search faster.

---

5. Real-World Applications of Adversarial Search

 Chess AI – Used in engines like Stockfish and AlphaZero.
 Checkers, Go, Tic-Tac-Toe – AI plays optimally using Minimax.
 Strategic decision-making – Used in cybersecurity and competitive business strategies.

4.explain knowledge and reasoning: knowledge representation, first order logic.

Knowledge and Reasoning

Knowledge representation and reasoning (KR&R) is a fundamental concept in Artificial Intelligence (AI) that deals with how knowledge is stored and used to make intelligent decisions. It enables AI systems to understand and reason about the world.

---

1. Knowledge Representation (KR)

Knowledge Representation refers to the way knowledge is structured and stored in an AI system so that it can be used for reasoning and decision-making.

Types of Knowledge Representation:

1. **Logical Representation:** Uses formal logic to represent facts and rules (e.g., Propositional Logic, First-Order Logic).
2. **Semantic Networks:** Represents knowledge in a network of interconnected nodes and edges.
3. **Frames:** Uses structured templates to store knowledge about objects and events.
4. **Production Rules:** Represents knowledge as IF-THEN rules used in expert systems.
5. **Ontologies:** Represents knowledge hierarchically with defined relationships between concepts.

---

2. First-Order Logic (FOL) (Predicate Logic)

Name: Prathamesh Arvind Jadhav

First-Order Logic (FOL) is an extension of **Propositional Logic** that allows the use of **quantifiers, predicates, and variables** to represent more complex statements. It provides a powerful way to describe relationships between objects.

Components of First-Order Logic:

1. **Constants:** Specific objects in the domain (e.g., **John, Apple, Car**).
2. **Variables:** General placeholders for objects (e.g., **x, y, z**).
3. **Predicates:** Describe properties or relationships between objects (e.g., **Loves(John, Mary)** means "John loves Mary").
4. **Functions:** Return a unique value based on input (e.g., **Father(John) = Robert**).
5. **Quantifiers:**
   - **Universal Quantifier (∀)** – Represents "for all" (e.g., ∀x Loves(x, IceCream) → "Everyone loves ice cream").
   - **Existential Quantifier (∃)** – Represents "there exists" (e.g., ∃x Loves(John, x) → "There exists someone whom John loves").

Example of First-Order Logic:

1. **Statement in English:** "All humans are mortal."
2. **FOL Representation:**
   ∀x (Human(x) → Mortal(x))
   (For all x, if x is a human, then x is mortal.)

3. Reasoning in First-Order Logic

Reasoning allows AI to derive new knowledge from existing facts. Common reasoning techniques include:

1. **Forward Chaining:** Starts with known facts and applies rules to derive conclusions.
2. **Backward Chaining:** Starts with a goal and works backward to find supporting facts.
3. **Resolution:** A proof technique used in automated theorem proving.

---

4. Applications of Knowledge Representation and First-Order Logic

 Expert Systems – Medical diagnosis, legal reasoning (e.g., MYCIN, DENDRAL).
 Natural Language Processing (NLP) – Understanding and translating human language.
 Robotics and AI Agents – Intelligent decision-making in dynamic environments.
 Semantic Web – Structured knowledge representation for web search engines.

5.Apply a* algorithm on given 8 puzzle problem use misplaced tiles method to calculate herusitic value

1. Problem Definition

- Initial State:

$$\begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & \_ & 5 \end{bmatrix}$$

- Goal State:

$$\begin{bmatrix} 1 & 2 & 3 \\ 8 & \_ & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

Name: Prathamesh Arvind Jadhav

*2. A Algorithm with Misplaced Tiles Heuristic**

A* uses:

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

where:

- $g(n)$ = cost from the start node to the current node.
- $h(n)$ = heuristic value (number of misplaced tiles).

Step 2: Calculate the Heuristic Value (h) for Initial State

The **Misplaced Tiles heuristic** counts how many tiles are in the wrong position compared to the goal state.

Tile	Initial Position	Goal Position	Misplaced?
1	(1,0)	(2,0)	Yes
2	(0,0)	(0,1)	Yes
3	(0,2)	(0,2)	No
4	(1,2)	(1,2)	No
5	(2,2)	(2,2)	No
6	(1,1)	(2,1)	Yes
7	(2,0)	(2,0)	No
8	(0,1)	(1,0)	Yes

Total misplaced tiles = 4
Thus, **h(initial) = 4**
Since we start from the initial state, **g(initial) = 0**
So, **f(initial) = g + h = 0 + 4 = 4**

Step 3: Generate Successors

The **blank space (_) is at (2,1)**. It can move in three possible directions:

1. **Up to (1,1)** → Swap **6** and **_**
2. **Left to (2,0)** → Swap **7** and **_**
3. **Right to (2,2)** → Swap **5** and **_**

Step 4: Calculate f(n) for Each Successor

Move Blank Up (Swap with 6)

Move Blank Up (Swap with 6)

New State:

```
2  8  3
1  _  4
7  6  5
```

- Misplaced Tiles: 3 (2, 8, 6)
- g = 1, h = 3, so f = 1 + 3 = 4

Move Blank Left (Swap with 7)

New State:

```
2  8  3
1  6  4
_  7  5
```

- Misplaced Tiles: **5**
- **g = 1, h = 5**, so **f = 1 + 5 = 6**

Move Blank Right (Swap with 5)

New State:

```
2  8  3
1  6  4
7  5  _
```

- Misplaced Tiles: **5**
- **g = 1, h = 5**, so **f = 1 + 5 = 6**

Step 5: Expand the Best Node

The state with the **lowest f-value** is chosen first.

- **Move Blank Up** has **f = 4** (lowest), so it is expanded.

Step 6: Repeat Until the Goal State is Reached

Name: Prathamesh Arvind Jadhav

1. **Initial State:** f=4
2. **Move Blank Up:** f=4
3. **Move Blank Left:** f=4
4. **Move Blank Up:** f=3
5. **Move Blank Right:** Goal State f=2

Final Path to Solution

1. Initial State:

```
2  8  3
1  6  4
7  _  5
```

2. Move Blank Up (Swap with 6)

```
2  8  3
1  _  4
7  6  5
```

3. **Move Blank Left (Swap with 1)**

```
2 8 3
_ 1 4
7 6 5
```

4. **Move Blank Up (Swap with 2)**

```nginx
_ 8 3
2 1 4
7 6 5
```

5. **Move Blank Right (Swap with 8) → Goal State Reached**

```
1 2 3
8 _ 4
7 6 5
```

6.Solve given cryptoarithmatic problem using CSP SEND+MORE = MONEY and TWO + TWO = Four

Solving Crypt arithmetic Problems using Constraint Satisfaction (CSP)

Crypt arithmetic puzzles involve assigning unique digits to letters such that a given arithmetic equation holds true. The **Constraint Satisfaction Problem (CSP)** approach ensures that:

1. Each letter represents a unique digit (0-9).
2. No two letters have the same digit.
3. The equation must be mathematically valid.

Name: Prathamesh Arvind Jadhav

Problem 1: SEND + MORE = MONEY

Solution: Assigning unique digits satisfying constraints:

Letter	Digit
S	9
E	5
N	6
D	7
M	1
O	0
R	8
Y	2

Verification:

$$9567 + 1085 = 10652$$ ✅

Problem 2: TWO + TWO = FOUR

Solution: Assigning unique digits satisfying constraints:

Letter	Digit
T	7
W	6
O	4
F	1
U	3
R	8

Verification:

$$764 + 764 = 1528$$ ✅

Name: Prathamesh Arvind Jadhav

7.Take one example and solve alph beta purning algorithm.

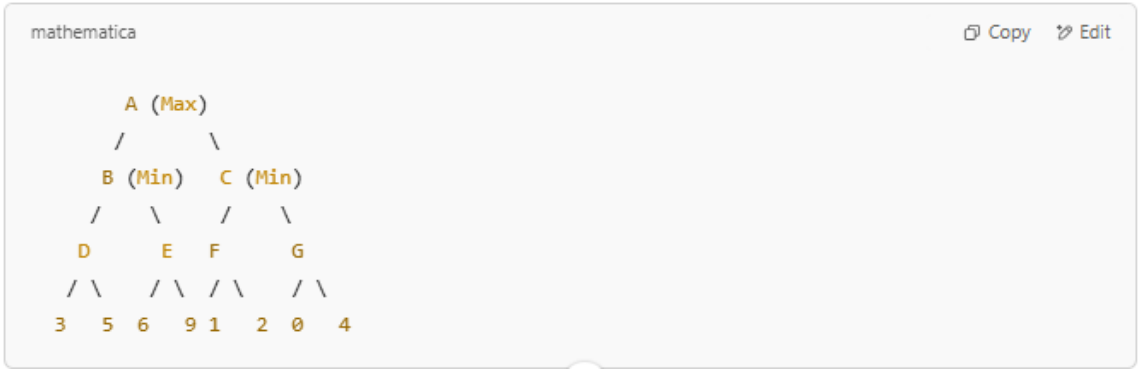
Alpha-Beta Pruning Algorithm – Example & Solution

Problem Statement:

Consider the following **Minimax tree** where we apply **Alpha-Beta Pruning** to optimize the decision-making process.

Game Tree Example

mathematica Copy Edit



Step-by-Step Execution of Alpha-Beta Pruning

We apply the **alpha (α) and beta (β) values** to prune unnecessary branches.

1. Initialize Values

- **Max Node (A):** $\alpha = -\infty, \beta = +\infty$
- **Min Nodes (B, C):** $\alpha = -\infty, \beta = +\infty$

---

2. Evaluate Left Subtree (B)

1. **Evaluate D:**
   - Left child = **3** → **D's min value = 3**
   - Right child = **5** → **D's final value = 3** (since Min node takes the smaller value)
   - **Update B:** $\beta = 3$
2. **Evaluate E:**
   - Left child = **6**, Right child = **9** → **E's min value = 6**
   - **B chooses the minimum of (D, E) → min(3,6) = 3**
   - **A (Max Node) updates α = 3**

---

3. Evaluate Right Subtree (C)

1. **Evaluate F:**
   - Left child = **1**, Right child = **2** → **F's min value = 1**
   - **C updates β = 1**
2. **Pruning at G:**
   - **Since C's β = 1 and A's α = 3, pruning occurs!**
   - No need to evaluate **G**, as we already know C ≤ 1, which won't affect A's decision.

---

4. Determine Final Value

- **B = 3, C = 1**
- **A (Max node) selects max(3,1) = 3**
- **Final decision value = 3**

Final Pruned Game Tree with Alpha-Beta Pruning

After applying **Alpha-Beta Pruning**, the **pruned tree** looks like this:

mathematica Copy Edit

```
        A (Max) = 3
       /           \
     B (Min) = 3     C (Min) = 1 (Pruned G)
    /     \      /
   D=3    E=6   F=1
  / \     / \   / \
 3   5   6   9 1   2
```

Name: Prathamesh Arvind Jadhav

Unit-3

1.Explain Planning with forward and backward State space search in AI.

Planning with Forward and Backward State Space Search in AI

State-space search is a fundamental approach in Artificial Intelligence (AI) planning, where a problem is represented as a set of states and transitions between them. Two main approaches to state-space search are **Forward Search (Progression Planning)** and **Backward Search (Regression Planning)**.

---

1. Forward State Space Search (Progression Planning)

In **Forward Search**, the search begins from the **initial state** and applies actions to generate new states until the **goal state** is reached.

Steps:

1. Start from the initial state.
2. Apply possible actions to generate successor states.
3. Check if the goal state is reached.
4. If not, continue expanding the states until the goal is found.

Advantages:

- Simple and intuitive as it directly explores from the initial state.
- Suitable for problems with a small number of possible actions.

Disadvantages:

- Can be inefficient due to large state spaces.
- May explore unnecessary paths leading to high computational costs.

Example:

Consider a **robot navigating a grid**. The initial state is the starting position, and actions like "Move Up," "Move Down," etc., generate new states. The search continues until the robot reaches the goal position.

---

2. Backward State Space Search (Regression Planning)

In **Backward Search**, the search starts from the **goal state** and works backward to find a path to the **initial state**.

Steps:

1. Start from the goal state.
2. Identify actions that could have resulted in the goal state.
3. Move backward by determining predecessor states.
4. Repeat until the initial state is reached.

Advantages:

- Focuses only on relevant states, reducing unnecessary exploration.
- More efficient in problems where there are fewer goal conditions than initial conditions.

Disadvantages:

- Requires reasoning about the preconditions of actions.
- Difficult if the goal is not clearly defined or has multiple constraints.

Example:

In a **block-stacking problem**, the goal state is a particular block arrangement. The search works backward by determining what actions could have resulted in that arrangement, eventually leading back to the initial state.

Comparison of Forward and Backward Search

Feature	Forward Search (Progression)	Backward Search (Regression)
Direction	From initial state to goal	From goal to initial state

Feature	Forward Search (Progression)	Backward Search (Regression)
Exploration	Explores all possible actions	Focuses on actions relevant to achieving the goal
Efficiency	Can be inefficient if many actions are available	More efficient when goal conditions are limited
Best for	Problems with well-defined initial states	Problems with specific goal states

2.Explain Partial order planning, Planning graphs, in AI.

Partial Order Planning & Planning Graphs in AI

Planning in AI involves generating a sequence of actions to achieve a goal from an initial state. Two important concepts in AI planning are **Partial Order Planning (POP)** and **Planning Graphs**.

---

1. Partial Order Planning (POP)

Partial Order Planning is a **non-linear** approach to planning that **does not fix the order of actions unless necessary**. Unlike linear (total-order) planners that generate actions sequentially, POP allows flexibility in action ordering, making it **more efficient** for complex tasks.

Key Concepts:

- **Causal Links:** Relationships where an action produces an effect needed by another action.
- **Open Preconditions:** Conditions that must be satisfied before an action executes.
- **Threats:** Actions that may interfere with causal links.
- **Plan Refinement:** Resolving threats and completing open preconditions to build a valid plan.

Steps in Partial Order Planning:

1. **Start with an empty plan** (only initial and goal states).
2. **Identify sub-goals** that need to be satisfied.
3. **Add actions** that achieve these sub-goals.
4. **Order actions only when necessary** to avoid conflicts.
5. **Resolve conflicts (threats) and ensure correctness.**
6. **Finalize the plan** when all sub-goals are satisfied.

Advantages of Partial Order Planning:

- More efficient for complex problems due to flexibility in action ordering.
- Avoids unnecessary commitment to a specific sequence.
- Suitable for multi-agent and concurrent planning.

Example of POP:

Imagine a **robot making breakfast** with actions:

1. **Pour coffee**
2. **Toast bread**
3. **Spread butter on toast**
4. **Serve breakfast**

Instead of deciding a strict order, the planner ensures:

- Toasting happens **before** spreading butter.
- Pouring coffee and toasting bread can **happen in parallel**. Thus, the planner only **orders necessary actions**, allowing flexibility.

---

2. Planning Graphs

Planning graphs are **graph-based structures** used to efficiently estimate the feasibility and cost of reaching a goal. They are used in **GraphPlan**, a well-known AI planning algorithm.

Structure of a Planning Graph:

A planning graph consists of alternating **state levels** and **action levels**:

- **State Level:** Represents all possible states (facts) at that level.
- **Action Level:** Represents all possible actions applicable in the previous state level.

Steps in Planning Graph Construction:

1. **Start with an initial state (S0).**
2. **Expand action levels:** Add all actions applicable in the current state.
3. **Expand state levels:** Add effects of actions to generate the next state.
4. **Continue expansion until the goal state appears** (or no new states can be generated).

Key Features of Planning Graphs:

- **Mutual Exclusion (Mutex):** Identifies conflicting actions that cannot occur together.
- **Compact Representation:** More efficient than traditional search methods.
- **Guides Heuristic Search:** Helps in estimating the shortest path to the goal.

Advantages of Planning Graphs:

- Faster than traditional search-based planning.
- Efficient for heuristic-based planners like **GraphPlan**.
- Identifies infeasible goals quickly using mutex constraints.

Example of a Planning Graph:

Consider a **robot moving objects**:

1. **Initial State:** Box at location A.
2. **Action Level 1:** Move box from A to B or Move box from A to C.
3. **State Level 2:** Box at B or Box at C.
4. **Action Level 2:** Further movements until reaching the goal.

By expanding the graph, we can efficiently determine the shortest sequence of actions.

Comparison: Partial Order Planning vs. Planning Graphs

Feature	Partial Order Planning (POP)	Planning Graphs
Nature	Flexible, non-linear planning	Graph-based structured planning
Execution Order	Orders actions only when needed	Expands all possible actions systematically
Efficiency	Avoids unnecessary commitments	Fast and compact representation
Used in	Complex, multi-agent, or parallel tasks	Heuristic and optimal planning problems
Example Use Case	Making breakfast (some tasks in parallel)	Robot movement planning

3.explain Planning with propositional logic in AI.

Planning with Propositional Logic in AI

Introduction

Planning with **propositional logic** is a formal approach where an AI agent formulates a planning problem using **logical statements (propositions)** and then applies **logical inference** to determine a valid sequence of actions to achieve the goal. This method is based on **classical planning** and is often used in **Automated Planning and Theorem Proving**.

---

1. Representing Planning with Propositional Logic

To model a planning problem using propositional logic, we need to define:

a) States as Propositional Variables

A **state** represents the current situation of the world using **propositions**.
For example, in a **robot vacuum cleaner** scenario:

- Clean(Room1): Room 1 is clean.
- At(Robot, Room1): The robot is in Room 1.

b) Actions as Logical Rules

Actions are represented as **preconditions, effects, and frame axioms**:

- **Preconditions**: Conditions that must be true before an action is applied.
- **Effects**: Changes in the state after executing the action.
- **Frame Axioms**: Conditions that remain unchanged after an action.

For example, an action **Move(Room1, Room2)** for the robot vacuum:

- **Precondition**: At(Robot, Room1) ∧ DoorOpen(Room1, Room2)
- **Effect**: ¬At(Robot, Room1) ∧ At(Robot, Room2)

c) Goal as a Logical Formula

The **goal** is defined as a logical expression that must be satisfied.
For example, the goal **"Both rooms should be clean"** can be written as:
Clean(Room1) ∧ Clean(Room2)

---

2. Planning as Logical Inference

Planning using propositional logic involves **finding a sequence of actions** that lead from the **initial state** to the **goal state** using **logical inference techniques**.

a) Situation Calculus

A formalism that represents **states, actions, and results** using first-order logic.

- **Do(A, S)**: Represents applying action A in state S.
- **Result(A, S)**: Represents the new state after action A.

Example:
At(Robot, Room1, S0) → Move(Room1, Room2, S1) → At(Robot, Room2, S2)

b) SAT-based Planning (Satisfiability Planning)

- The planning problem is converted into a **Boolean satisfiability (SAT) problem**.
- A SAT solver finds a **set of actions** that lead to the goal.
- The solution is extracted from the SAT solver's output.

Example: **SAT encodes**:

1. **Initial state:** At(Robot, Room1) ∧ ¬Clean(Room2)
2. **Goal state:** Clean(Room1) ∧ Clean(Room2)
3. **Action constraints:** Using **Boolean variables** like Move(Room1, Room2, t)

---

3. Advantages & Disadvantages of Propositional Logic Planning

Feature	Pros	Cons
Logical Representation	Provides a clear, formal structure	Can become complex for large problems
Automated Reasoning	Can use SAT solvers & theorem proving	Requires efficient encoding
Optimality	Ensures logically sound plans	May require large search spaces
Scalability	Good for small to medium domains	Hard to scale for real-world problems

---

4. Example: Planning for a Robot in a 2-Room Environment

Problem Definition

- **Initial State:** At(Robot, Room1) ∧ ¬Clean(Room2)
- **Actions:**
  - Move(Room1, Room2): Moves the robot to Room 2.
  - Clean(Room): Cleans the current room.
- **Goal State:** Clean(Room1) ∧ Clean(Room2)

Encoding in Propositional Logic

1. **Initial state:**
   - At(Robot, Room1, S0)

- ¬Clean(Room2, S0)

2. **Actions & Preconditions:**
   - Move(Room1, Room2, S) → At(Robot, Room1, S)
   - Clean(Room2, S) → At(Robot, Room2, S)
3. **Goal formula:**
   - Clean(Room1, Sn) ∧ Clean(Room2, Sn)

A **SAT solver** can determine a sequence of actions such as:

1. Move(Room1, Room2)
2. Clean(Room2)

This sequence **satisfies** the goal state.

4.Explain Planning and acting in real world.

Planning and Acting in the Real World (AI Perspective)

Introduction

In Artificial Intelligence (AI), **planning** refers to the process of generating a sequence of actions to achieve a goal, while **acting** is the execution of those actions in a real-world environment. Unlike theoretical planning, real-world scenarios introduce **uncertainty, dynamic changes, and incomplete information**, making planning more complex.

---

1. Challenges in Real-World Planning and Acting

Planning in a **real-world environment** differs from classical AI planning due to various challenges:

Challenge	Explanation
Uncertainty	The environment is unpredictable (e.g., sensor errors, unexpected events).
Dynamic Changes	The world changes while the plan is being executed.

Challenge	Explanation
Incomplete Information	The agent may not have full knowledge of the environment.
Real-Time Constraints	Plans must be executed within time limits.
Multiple Agents	The presence of humans or other AI systems affects planning.

Example:

A **self-driving car** plans a route but encounters a **sudden roadblock**. It must **replan in real-time** while considering **traffic rules, pedestrian safety, and sensor data**.

---

2. Approaches to Real-World Planning and Acting

To handle these challenges, AI uses various strategies:

A) Reactive Planning

- The agent responds **immediately** to changes in the environment without a fixed plan.
- Based on **if-then rules** or simple **reflex actions**.
- **Example:** A **robot vacuum** moves based on obstacle detection.

B) Conditional Planning

- The plan includes **alternative actions** based on possible situations.
- Works well in environments with **predictable uncertainties**.
- **Example:** A **Mars rover** plans multiple routes based on terrain conditions.

C) Hierarchical Task Networks (HTN)

- Breaks down complex tasks into **subtasks** for easier execution.

- Used in robotics and real-time strategy games.
- **Example:** A **robotic arm** assembling a product in steps.

D) Replanning & Execution Monitoring

- The agent **monitors execution** and **replans** when necessary.
- Helps in handling **unexpected changes**.
- **Example:** A **self-driving car** changes lanes if traffic conditions change.

E) Reinforcement Learning (RL) for Adaptive Planning

- The agent **learns** the best actions through **trial and error**.
- Useful for **complex, dynamic** environments.
- **Example:** A **robot dog** learning to walk on uneven surfaces.

---

3. Real-World Applications of Planning and Acting

A) Robotics

- **Autonomous drones** plan delivery routes while avoiding obstacles.
- **Factory robots** adapt to changing production demands.

B) Autonomous Vehicles

- **Self-driving cars** plan routes while responding to traffic signals and pedestrians.
- **Smart traffic systems** adjust signals based on congestion data.

C) Healthcare

- **AI-assisted surgery** plans robotic movements while adjusting for real-time patient conditions.
- **Medical diagnosis systems** suggest treatments based on patient history.

D) Smart Assistants & Home Automation

- **Alexa/Google Assistant** plans tasks like setting reminders or controlling smart devices.
- **Smart thermostats** adjust heating/cooling based on occupancy.

Name: Prathamesh Arvind Jadhav