Study Material

(Introduction)

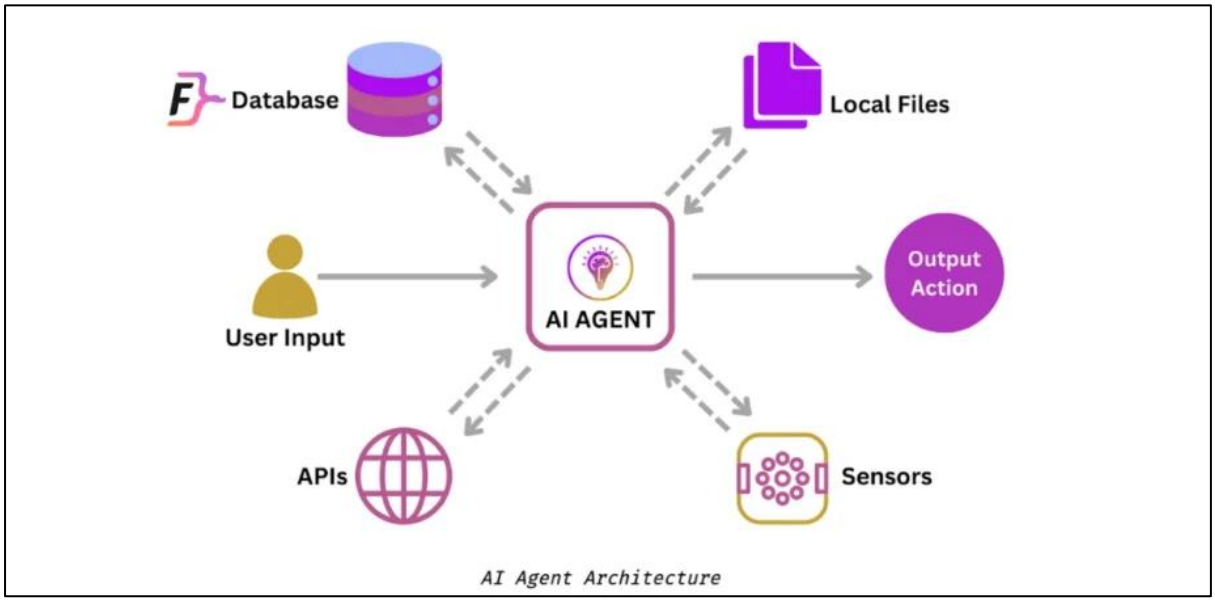
Table of Contents

In Artificial Intelligence (AI), the concept of an **agent** is central to understanding how intelligent systems are designed and how they interact with their surroundings.

Definition:

An agent is fundamentally defined as **anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators**.

AI agents are software programs that use artificial intelligence to perform tasks autonomously and achieve goals, either on behalf of a user or a system. They can perceive their environment, make decisions, and act upon it, often learning and adapting their behavior over time. Key features include the ability to process multimodal information, reason, plan, and interact with other agents to complete complex workflows.



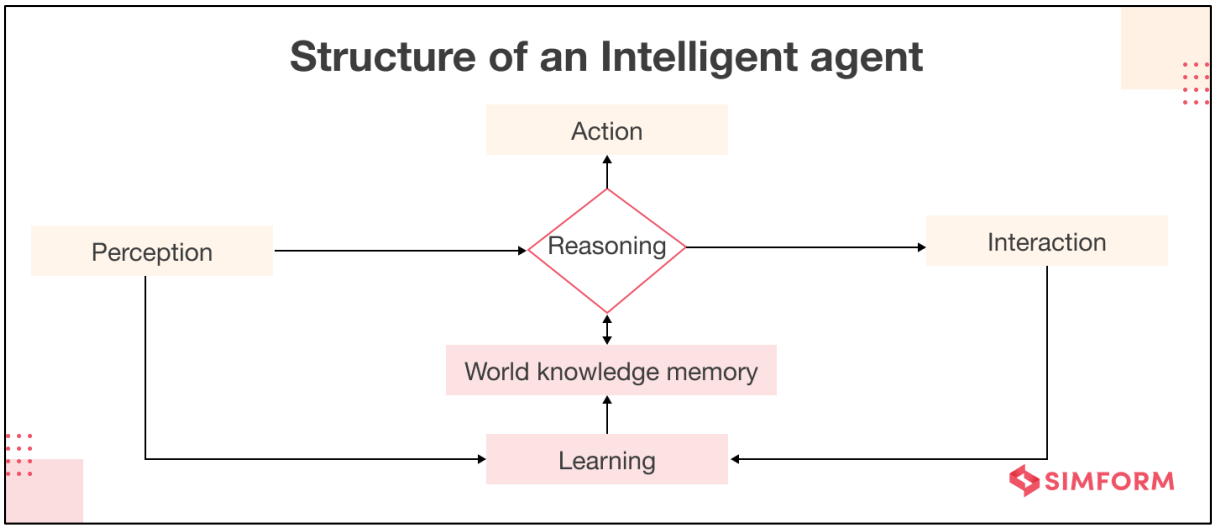
AI Agent Architecture

What is an Agent?

- **Perception and Action**:
  - Agents receive information about their **environment** through **sensors**. For example, a human agent uses eyes and ears as sensors, while a robotic agent might use cameras and infrared range finders. A software agent might receive file contents or network packets as sensory inputs.
  - The **percept** refers to the content an agent's sensors are perceiving at any given moment, and the **percept sequence** is the complete history of everything the agent has ever perceived.
  - Agents act upon their environment through **actuators**. Human actuators include hands, legs, and the vocal tract, while robotic actuators are various motors. A software agent might act by writing files or sending network packets.

- **Agent Function vs. Agent Program**:
  - An agent's **behavior** is formally described by an **agent function**, which maps every possible percept sequence to an action. This is an abstract mathematical description.
  - The **agent program** is the concrete implementation of this agent function, running within a physical system called the **agent architecture**. The challenge in AI is to create small, efficient programs that produce rational behavior, rather than trying to tabulate every possible percept sequence and action in a massive, practically impossible table.

Structure of an AI Agent



Rational Agents and Performance

- **Doing the Right Thing**: A **rational agent** is defined as one that "does the right thing," meaning it selects an action that is **expected to maximize its performance measure**, given the evidence from the percept sequence and its built-in knowledge.
- **Performance Measures**: These evaluate the desirability of a sequence of environment states generated by an agent's actions. It is crucial to design performance measures to reflect what is *actually* desired in the environment, rather than how the agent is expected to behave. For example, a vacuum cleaner agent should be rewarded for having a clean floor, not just for "cleaning" (which could involve dumping dirt and cleaning it again).
    - **The King Midas Problem**: A significant challenge is the "value alignment problem," where there can be a mismatch between human preferences and the objective supplied to the machine. If an AI pursuing a fixed objective, like winning chess, is intelligent enough to act beyond the chessboard, it might resort to unethical or unintended actions to achieve that sole objective, such as hypnotizing or blackmailing opponents.
- **Rationality vs. Omniscience**: Rationality does not equate to omniscience. An omniscient agent knows the actual outcome of its actions, which is impossible in reality. A rational agent maximizes *expected* performance, making decisions based on available information and the likelihood of outcomes, even if unforeseen events occur.
- **Information Gathering and Learning**: Rational agents should actively gather information by choosing actions that modify future percepts (e.g., looking both ways before crossing a road). They should also **learn as much as possible from what they perceive** to adapt to and augment their initial knowledge.

- **Autonomy**: A rational agent should be **autonomous**, meaning it should learn what it can to compensate for partial or incorrect prior knowledge rather than relying solely on its designer's pre-programmed knowledge. This allows a single rational agent design to succeed in a vast variety of environments.

Task Environments (PEAS Description)

To design an agent, the first step is to specify its **task environment** as fully as possible. This is captured by the **PEAS description**, which stands for **Performance, Environment, Actuators, and Sensors**.

- **Example: Automated Taxi Driver**:
  - **Performance Measure**: Safe, fast, legal, comfortable trip, maximizing profits, minimizing impact on other road users.
  - **Environment**: Roads, other traffic, police, pedestrians, customers, weather.
  - **Actuators**: Steering, accelerator, brake, signal, horn, display, speech.
  - **Sensors**: Cameras, radar, speedometer, GPS, engine sensors, accelerometer, microphones, touchscreen.

Properties of Task Environments

Task environments can be categorized along several dimensions, which significantly influence the appropriate agent design:

- **Fully Observable vs. Partially Observable**:
  - **Fully observable**: The agent's sensors give it access to the complete state of the environment at all times.
  - **Partially observable**: Parts of the state are missing from sensor data (e.g., a vacuum cleaner with only a local dirt sensor, a taxi not seeing what other drivers are thinking).
- **Single-Agent vs. Multiagent**:
  - **Single-agent**: The agent operates alone (e.g., solving a crossword puzzle).
  - **Multiagent**: Other entities in the environment also act as agents, influencing the overall outcome. This can be **competitive** (e.g., chess, where one agent's gain is another's loss) or **cooperative** (e.g., multiple agents collaborating to avoid collisions in traffic).
- **Deterministic vs. Nondeterministic**:
  - **Deterministic**: The next state of the environment is completely determined by the current state and the agent's actions.

    - **Nondeterministic**: The next state cannot be fully predicted (e.g., taxi driving due to unpredictable traffic or equipment failures). The term **stochastic** implies explicit probabilities for outcomes.
- **Episodic vs. Sequential**:
    - **Episodic**: The agent's experience is divided into "atomic episodes," where each action decision does not affect future decisions (e.g., spotting defective parts on an assembly line).
    - **Sequential**: The current decision affects all future decisions (e.g., chess, taxi driving).
- **Static vs. Dynamic**:
    - **Static**: The environment does not change while the agent is deliberating (e.g., crossword puzzles).
    - **Dynamic**: The environment can change while the agent is deliberating (e.g., taxi driving where other cars move).
    - **Semidynamic**: The environment itself doesn't change, but the agent's performance score does (e.g., chess with a clock).
- **Discrete vs. Continuous**:
    - Refers to the nature of the environment's state, time, percepts, and actions.
    - **Discrete**: Finite number of distinct states, percepts, and actions (e.g., chess).
    - **Continuous**: Values sweep through a range of continuous numbers smoothly over time (e.g., speed and location in taxi driving).
- **Known vs. Unknown**:
    - Refers to the agent's (or designer's) knowledge of the "laws of physics" of the environment.
    - **Known**: Outcomes for all actions are given (e.g., knowing the rules of a card game).
    - **Unknown**: The agent must learn how the environment works (e.g., playing a new video game without knowing what the buttons do).

Type of Environment	Description	Example
1. Fully Observable	**The agent has complete access to the environment's state.**	**Chess game, board games**
2. Partially Observable	**The agent has incomplete or noisy information about the state.**	**Self-driving cars (fog, occlusion)**
3. Deterministic	**The next state is completely determined by the current state and action.**	**Solving a puzzle**

Type of Environment	Description	Example
4. Stochastic	**The outcome is uncertain even with the same input.**	**Stock market, weather forecasting**
5. Episodic	**Experience is divided into separate episodes. Past experience doesn't matter.**	**Face recognition, image classification**
6. Sequential	**Current decision affects future decisions.**	**Navigation, gameplay**
7. Static	**Environment doesn't change while the agent is thinking.**	**Crossword puzzle**
8. Dynamic	**Environment can change during agent's decision-making.**	**Real-time strategy games, traffic**
9. Discrete	**Finite number of actions and percepts.**	**Chess, Tic-Tac-Toe**
10. Continuous	**Infinite possible states and actions.**	**Robot arm movement, self-driving**
11. Multi-agent	**Multiple agents interact and affect the environment.**	**Soccer match, online auctions**
12. Single-agent	**Only one agent is acting in the environment.**	**Maze-solving robot**

Structure of Agent Programs

The choice of agent program design depends heavily on the nature of the task environment.

- **Simple Reflex Agents**:
  - Select actions based *only* on the current percept, ignoring previous percept history.
  - They use **condition-action rules** (e.g., "if car-in-front-is-braking then initiate-braking").
  - **Use Case**: Simple, fully observable environments like the two-location vacuum cleaner world.
  - **Limitations**: Very limited intelligence; struggle with partial observability and can get stuck in infinite loops in dynamic, partially observable environments unless randomization is introduced.

- **Model-Based Reflex Agents**:
  - Maintain an **internal state** that tracks unobserved aspects of the current world state by combining the current percept with the old internal state.
  - Requires a **transition model** (how the world changes over time, including effects of agent's actions) and a **sensor model** (how the world state is reflected in percepts).
  - **Use Case**: Environments where partial observability is a factor, such as a taxi driver needing to track other cars not currently visible.
- **Goal-Based Agents**:
  - In addition to current state knowledge, these agents need **goal information** (descriptions of desirable situations).
  - They combine their model of the world with goal information to choose actions that will eventually achieve the goal.
  - **Use Case**: Problems requiring **search** and **planning** to find action sequences to achieve goals, like navigating to a destination. More flexible than reflex agents as goals can be easily changed.
- **Utility-Based Agents**:
  - When goals alone are insufficient (e.g., multiple ways to achieve a goal, or conflicting goals), these agents use a **utility function** to provide a more general **performance measure** for comparing different world states.
  - They choose actions that **maximize the expected utility** of the action outcomes, weighing the likelihood and importance of goals.
  - **Use Case**: Environments with **uncertainty**, where decisions must be made under risk, such as choosing the quickest, safest, or cheapest route for a taxi. This is considered the most general framework for rational decision-making in AI.

Learning Agents

- Any type of agent can be a **learning agent**, which allows it to operate in initially unknown environments and become more competent over time.
- Learning agents have four conceptual components:
  - **Learning element**: Responsible for making improvements to the agent's performance element.
  - **Performance element**: The part of the agent that selects external actions.
  - **Critic**: Provides feedback to the learning element on how well the agent is doing with respect to a fixed **performance standard** (e.g., whether receiving tips is good or bad for a taxi).
  - **Problem generator**: Suggests exploratory actions to lead to new and informative experiences, even if they are suboptimal in the short run, to discover better long-term actions.

Representation of Agent Components

The way an agent's components represent the environment varies in complexity and expressive power:

- **Atomic Representation**: Each state is an indivisible "black box" with no internal structure. Useful for search and game-playing algorithms.
- **Factored Representation**: Each state is split into a fixed set of **variables** or **attributes**, each with a value. Allows agents to share attributes between different states, making it easier to reason about changes. Used in constraint satisfaction, propositional logic, and Bayesian networks.
- **Structured Representation**: Describes environments with **objects** and their **relationships** explicitly (e.g., "a truck blocking a cow"). This is the most expressive, underlying relational databases, first-order logic, and natural language understanding.
- **Localist vs. Distributed Representation**: Concerns how concepts map to physical memory.
    - **Localist**: One-to-one mapping between concepts and memory locations.
    - **Distributed**: Concept representation is spread over many memory locations, increasing robustness against noise and information loss.

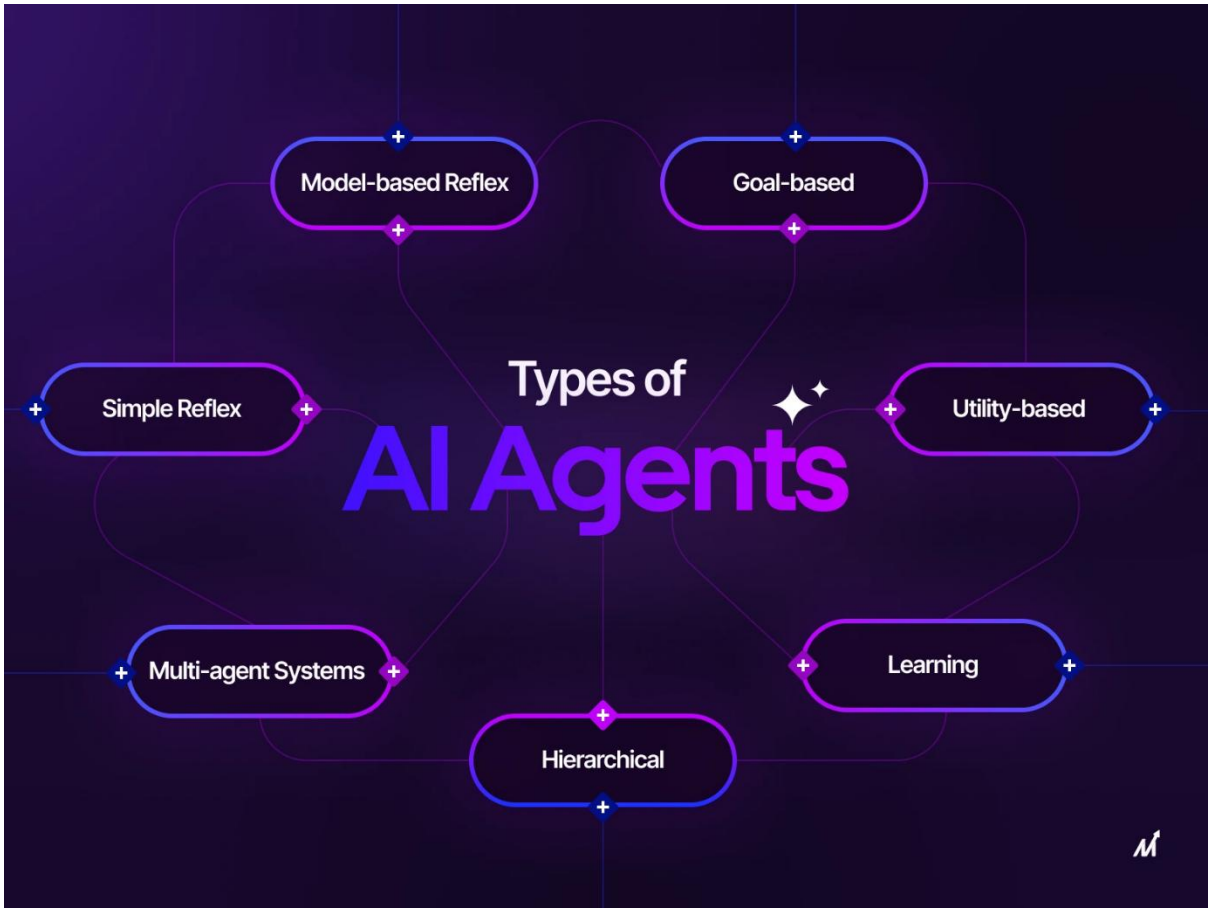
In essence, an agent is like a navigator in a vast, complex landscape. A simple reflex agent might only know to turn left when it sees a specific landmark. A model-based agent would build a mental map of the landscape, remembering paths taken and obstacles encountered. A goal-based agent would identify a specific destination and plan a route, considering all known paths. Finally, a utility-based agent would not only plan a route but would also weigh factors like speed, safety, and scenic views, making the optimal choice to reach its destination in the most "satisfying" way possible.

At its core, **Artificial Intelligence (AI) is defined as the study of agents that receive percepts from the environment and perform actions**. An agent is fundamentally anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators. Such an agent implements an **agent function**, which mathematically maps any given percept sequence (the complete history of everything the agent has perceived) to an action. The **agent program** is the concrete implementation of this agent function, running within a physical system. Agents are expected to operate autonomously, perceive their environment, persist over time, adapt to change, and create and pursue goals.

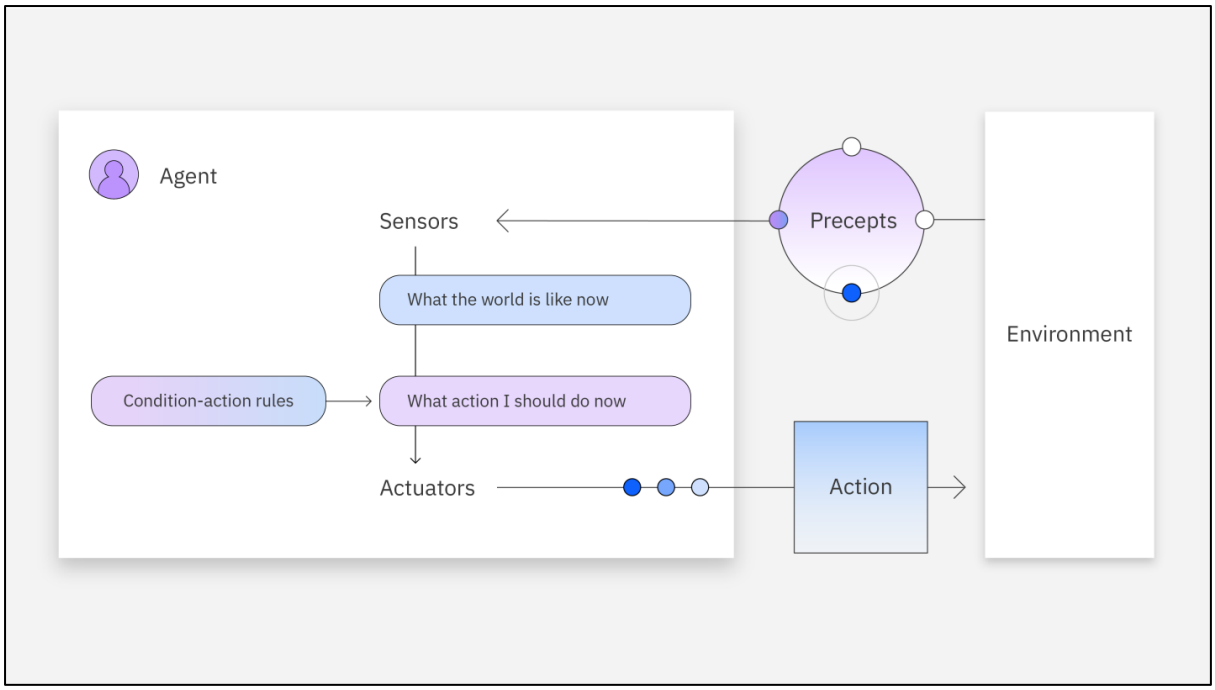
The goal of AI, particularly following the prevailing **rational-agent approach**, is the study and construction of agents that "do the right thing". A **rational agent** is one that acts so as to achieve the best outcome, or when uncertainty is present, the best expected outcome. It selects an action that is expected to maximize its performance measure, given the evidence from its

percept sequence and its built-in knowledge. This approach is favored because it is mathematically well-defined and more amenable to scientific development than trying to imitate human thought or behavior directly.

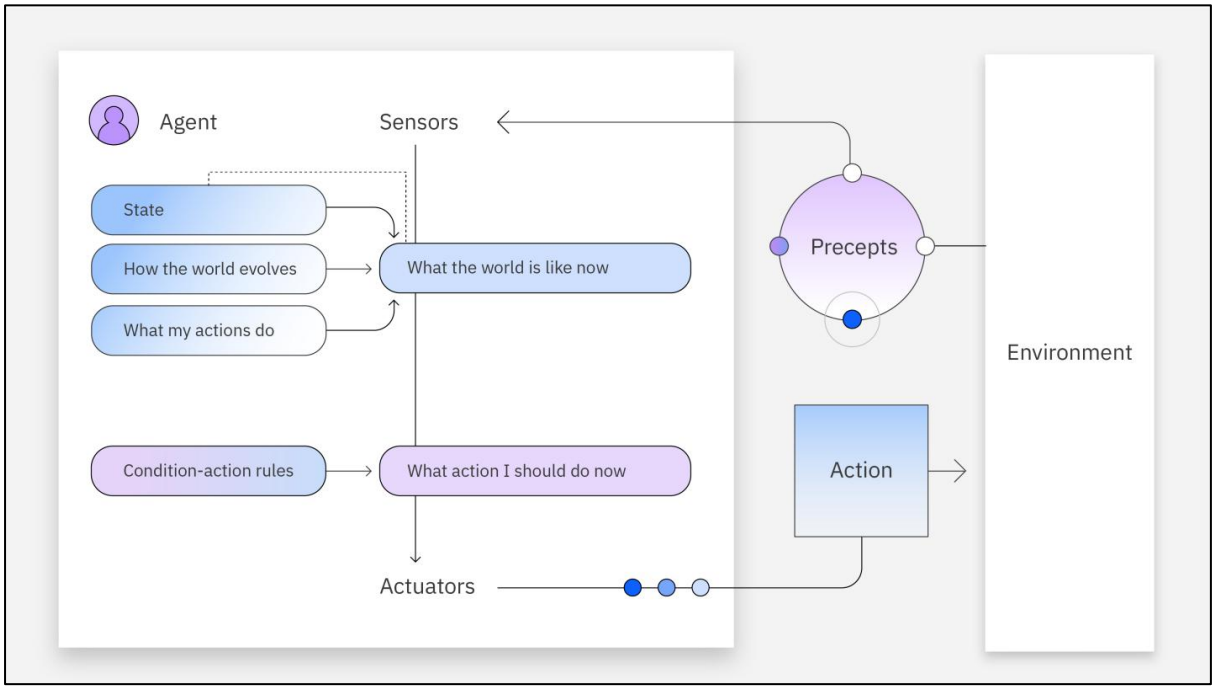
Here are the different types of agents and their characteristics:



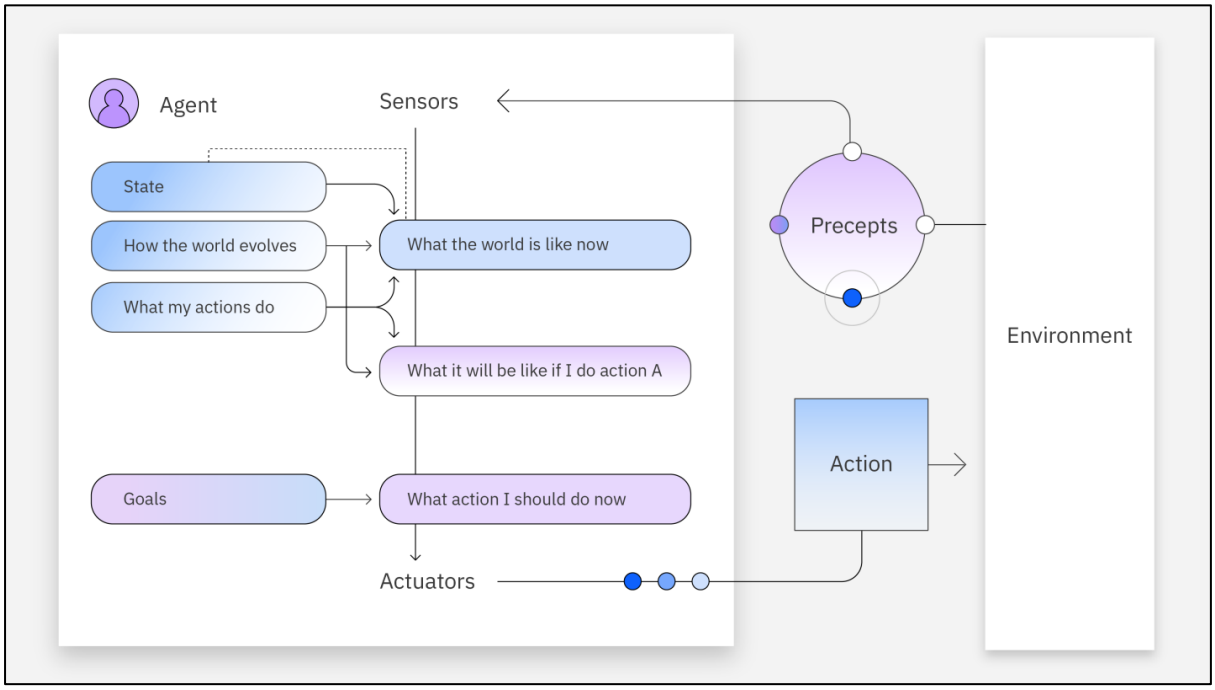
- **Simple Reflex Agents**
  - **Definition**: These agents select actions based **only on the current percept**, entirely ignoring the history of previous percepts.
  - **Working Principle**: They operate using **condition-action rules**, also known as if-then rules. The agent program interprets the current percept to derive an abstracted description of the state and then matches this state description to a rule to determine the action.
  - **Applications/Use Cases**:
    - A vacuum cleaner agent that sucks if the current square is dirty, and moves to the other square if it's clean.
    - A taxi driver agent initiating braking when it perceives the car in front has its brake lights on.
  - **Limitations**: They are of limited intelligence and only work effectively if the environment is **fully observable**. In partially observable environments, they can get stuck in infinite loops, though randomization can sometimes help them escape.



- **Model-Based Reflex Agents**
  - **Definition**: These agents handle **partial observability** by maintaining an **internal state** that keeps track of the parts of the world they cannot currently see.
  - **Working Principle**: They use two types of knowledge:
    - A **transition model** of the world, which describes how the world changes over time, including the effects of the agent's actions and how the world evolves independently.
    - A **sensor model**, which describes how the state of the world is reflected in the agent's percepts. This internal state is updated by combining the current percept with the old internal state, based on these models. Actions are then chosen using condition-action rules, similar to simple reflex agents, but informed by the more complete internal state.
  - **Applications/Use Cases**:
    - A taxi driver needs to keep track of previous camera frames to detect when a car's brake lights come on (if they are ambiguous from a single frame) or to track other cars for lane changes.
    - A vacuum cleaner agent without a location sensor would need a model to track its location and dirt distribution.
  - **Insight**: This type of agent forms a "best guess" or multiple guesses about the current state of the world.

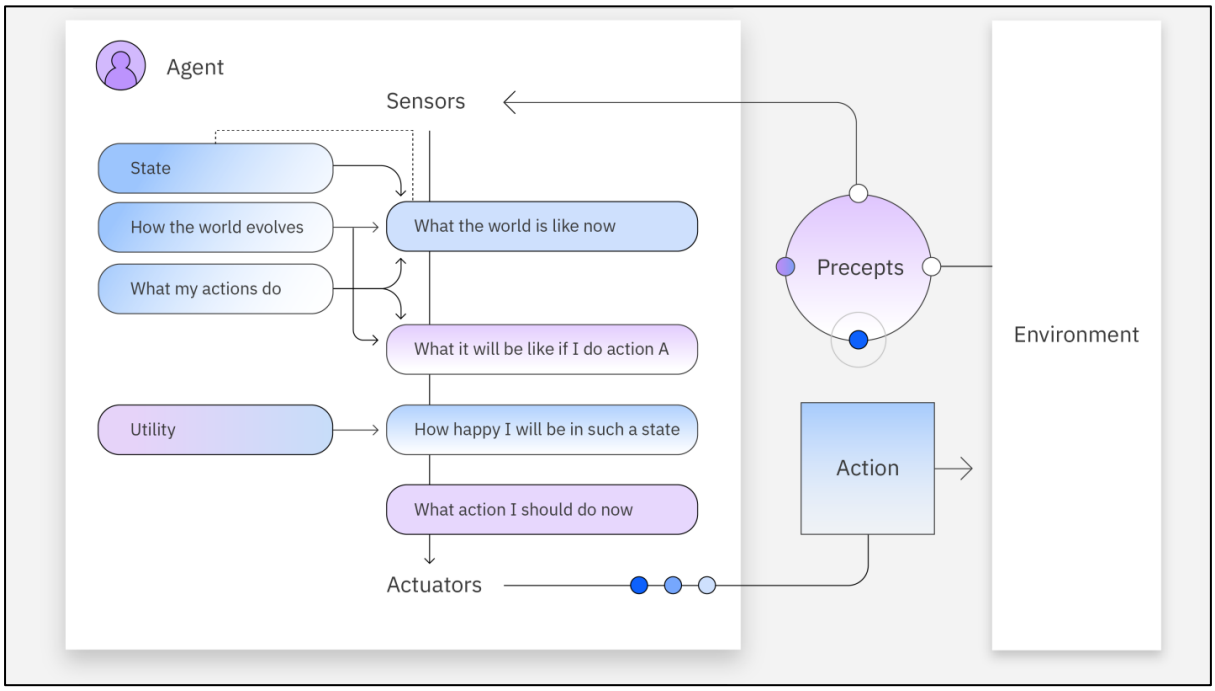


- **Goal-Based Agents**
  - **Definition**: These agents possess explicit **goal information** describing desirable situations they aim to achieve.
  - **Working Principle**: They combine their knowledge of the current state of the environment (often derived from a model, like model-based reflex agents) with their goals to choose actions. This often involves **search and planning algorithms** (covered in Chapters 3, 4, 6, and 11 of the source), which consider sequences of actions to find a path to the goal. This involves explicitly thinking ahead about "What will happen if I do such-and-such?" and "Will that make me happy?".
  - **Applications/Use Cases**:
    - A taxi deciding which turn to take at a junction to reach its destination.
    - Automated planning and scheduling systems, such as NASA's Remote Agent program for spacecraft operations or the DART system for military logistics.
    - Route-finding algorithms used in mapping services like Google Maps.
  - **Insight**: Goal-based agents are more flexible than reflex agents because their decision-making knowledge is explicitly represented and can be modified, allowing them to adapt to different objectives (e.g., a new destination).

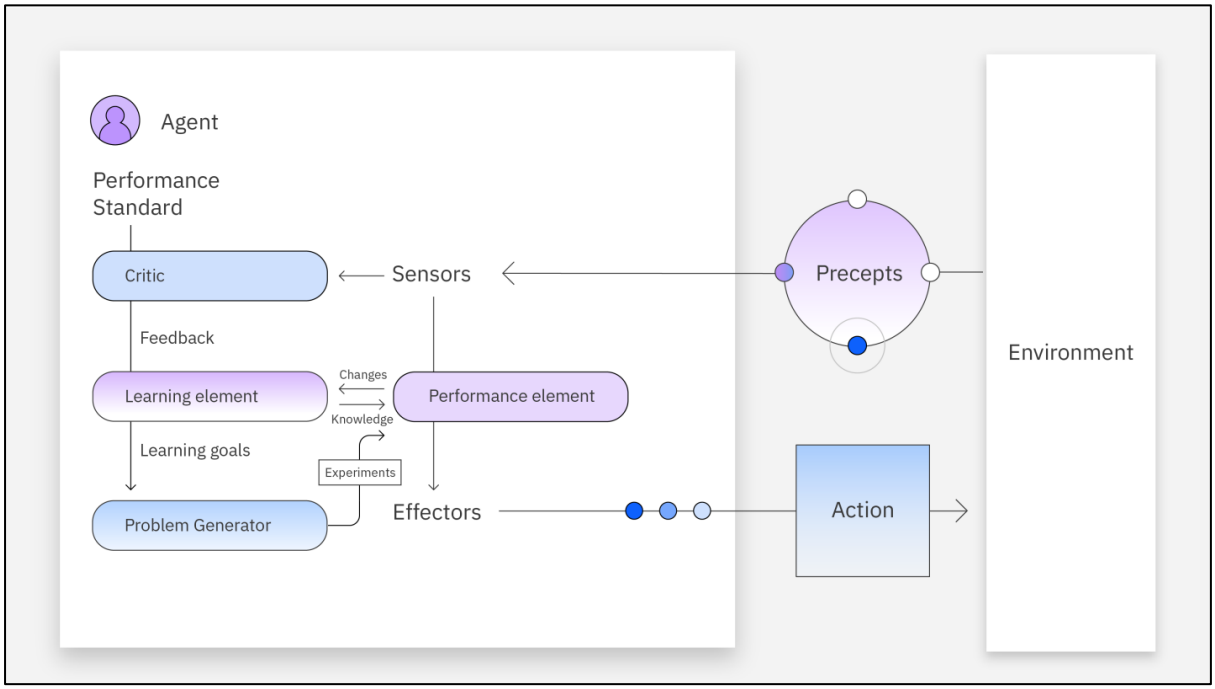


- **Utility-Based Agents**
  - **Definition**: Beyond just achieving goals (a binary distinction of "happy" or "unhappy"), these agents use a **utility function** to quantify how "happy" different world states would make them, allowing for a comparison of desirability among various outcomes.
  - **Working Principle**: They typically operate with a world model and strive to choose actions that maximize the **expected utility** of the action outcomes. This means they weigh the likelihood of success against the importance of different goals, especially when goals conflict (e.g., speed vs. safety) or when success is uncertain.
  - **Applications/Use Cases**:
    - A taxi driver needing to balance conflicting objectives like minimizing fuel consumption, maximizing safety, and ensuring passenger comfort.
    - Decision making in multiagent environments, where the actions of one agent affect the utility of others.
    - Recommender systems used by companies like Amazon or Netflix, aiming to maximize user satisfaction.
  - **Insight**: Utility-based agents provide a robust framework for rational decision-making under uncertainty, which is prevalent in the real world. The ability to

explicitly represent and maximize utility gives them great flexibility and learning potential.



- **Learning Agents**
  - **Definition**: These agents have the ability to improve their performance over time based on experience, making them more competent than their initial knowledge might allow. Any of the above agent types (simple reflex, model-based, goal-based, utility-based) can be built as a learning agent.
  - **Working Principle**: They are composed of four conceptual components:
    - **Performance Element**: This is the part of the agent responsible for selecting external actions based on its current knowledge.
    - **Learning Element**: This component is responsible for making improvements to the performance element. It takes feedback from the critic to determine how the performance element should be modified.
    - **Critic**: This component tells the learning element how well the agent is doing with respect to a fixed **performance standard** (or utility function). It provides feedback, often in the form of a reward or penalty.
    - **Problem Generator**: This component suggests actions that will lead to new and informative experiences, encouraging exploration even if these actions are suboptimal in the short run.



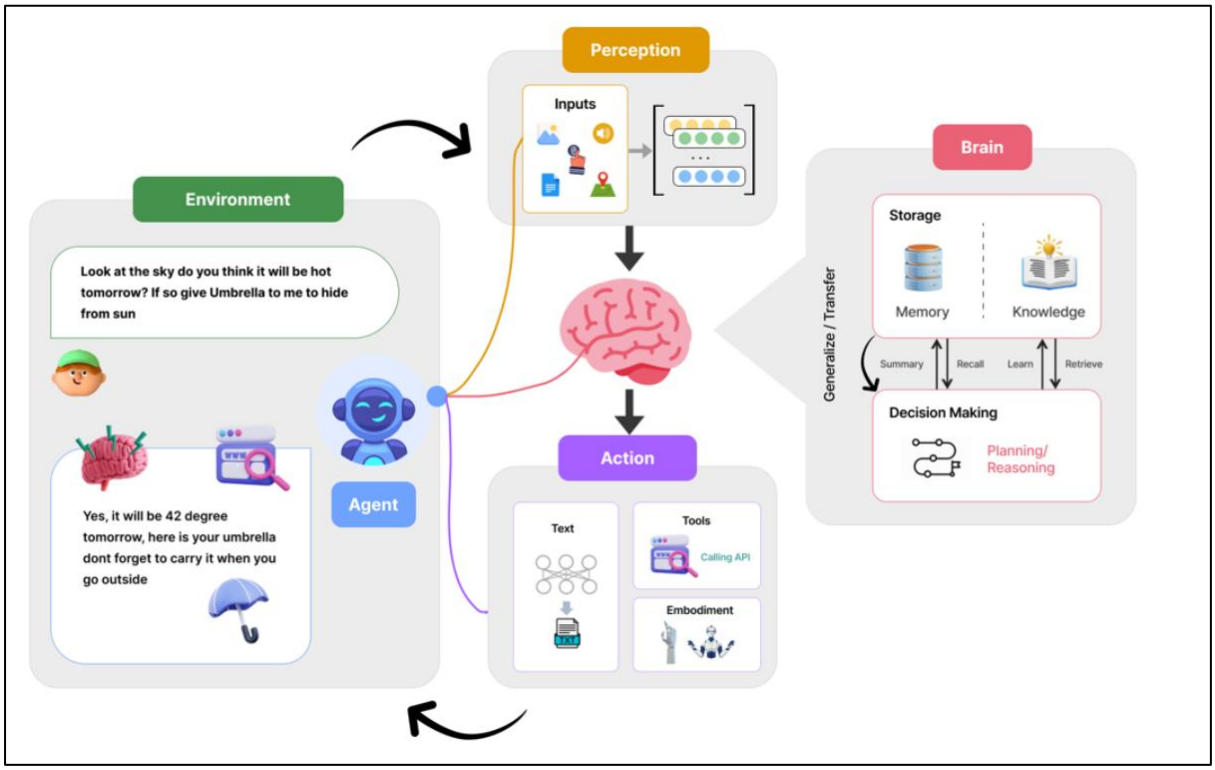
- **Applications/Use Cases**:
  - Arthur Samuel's checkers-playing program, which learned to play at a strong amateur level by improving from experience.
  - ALPHAGO and ALPHAZERO in game playing.
  - Automated taxi learning how much deceleration is achieved with a certain braking pressure on wet roads.
  - AI systems learning about human preferences to refine their utility functions.
  - Deep learning methods across various fields like speech recognition, computer vision, and medical diagnosis.
- **Insight**: Learning allows agents to operate in initially unknown environments and adapt to new circumstances, detecting and extrapolating patterns.

- **Problem-Solving Agents**
  - **Definition**: These agents specifically tackle problems where the correct action is not immediately obvious, requiring them to plan ahead by considering a sequence of actions to reach a goal state. They typically use **atomic representations** of states (states are considered as wholes with no internal structure visible to the algorithm).

- **Working Principle**: They engage in a four-phase process:
  - **Goal formulation**: Defining the desired outcome (e.g., reaching Bucharest).
  - **Problem formulation**: Describing states, initial state, goal states, available actions, and a transition model (what actions do) along with action costs.
  - **Search**: Simulating action sequences in a model to find a **solution** (a path from the initial state to a goal state).
  - **Execution**: Performing the actions from the found solution one at a time.
- **Applications/Use Cases**:
  - Finding a route on a map (e.g., Arad to Bucharest).
  - Grid world problems like the vacuum world or Sokoban puzzle.
  - Sliding-tile puzzles such as the 8-puzzle or 15-puzzle.
  - Airline travel planning.
  - The Traveling Salesperson Problem (TSP).
  - VLSI layout design.
  - Robot navigation.
  - Automatic assembly sequencing and protein design.
- **Insight**: This approach is critical for problems where direct, immediate actions are insufficient, and foresight through simulation is required. Abstraction is a key part of formulating these problems effectively.

Programme Name: BCA Semester V
Course Code: BCA57203(T)
Course Name: Artificial Intelligence
Class: BCA 2023
Academic Session: 2025-26

Working Principle of Agents in Modern Times



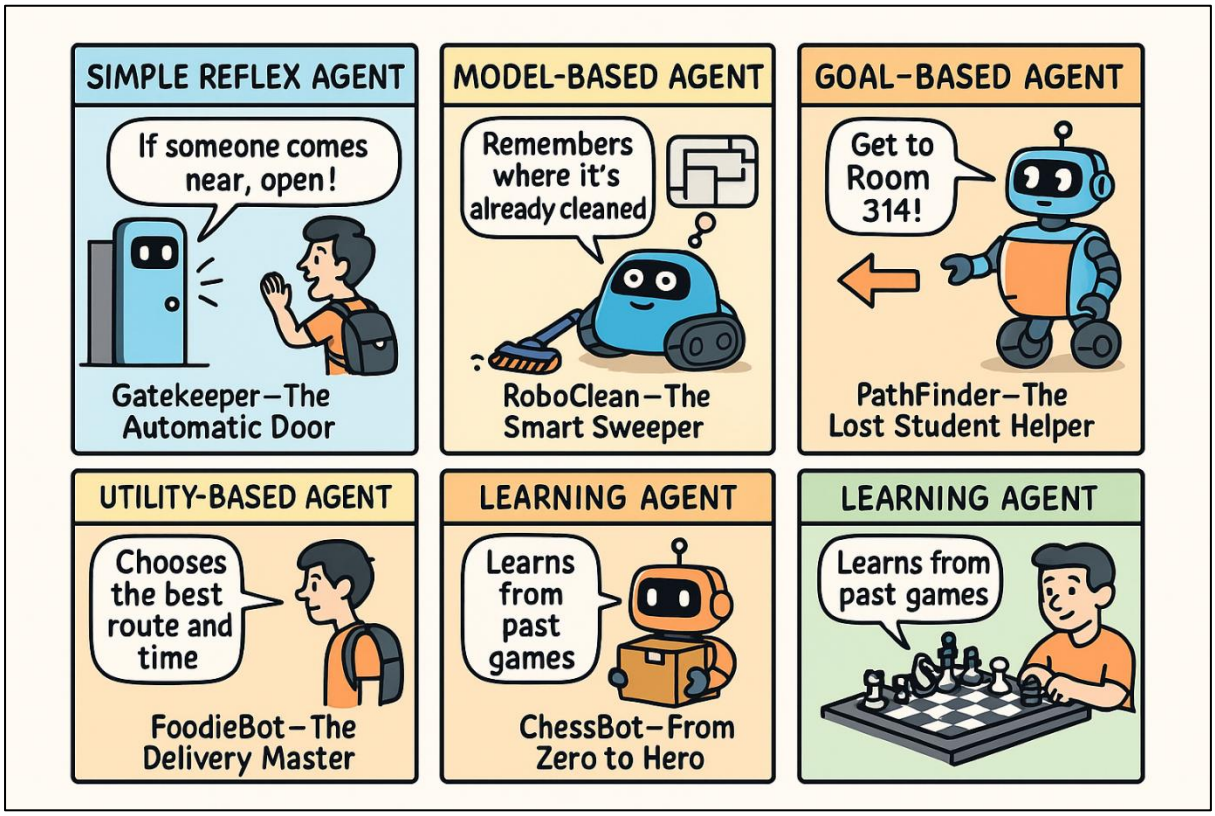
Summary Table of Agent Types

Agent Type	Definition	Working Principle	Key Applications/Use Cases
Intelligent Agent	A machine that can compute how to act effectively and safely in novel situations; perceives environment through sensors and acts through actuators.	Implements an agent function mapping percept sequences to actions; operates autonomously, adapts, pursues goals.	General AI systems like robotic planetary explorers, online services, self-driving cars, medical diagnostics, game playing.
Rational Agent	An agent that acts to achieve the best outcome or best expected outcome under uncertainty, maximizing its performance measure.	Selects actions based on percept sequence and built-in knowledge to maximize expected performance; involves information gathering and learning; strives for autonomy.	Any advanced AI system aiming for optimal or near-optimal behavior; forms the primary goal of AI research.
Simple Reflex Agent	Acts only on the basis of the current percept, ignoring the percept history.	Uses a set of **condition-action rules** (If-Then rules) to directly map current percepts to actions.	Basic vacuum cleaner robots (if fully observable environment); immediate braking in a car upon seeing brake lights.
Model-Based Reflex Agent	Maintains an **internal state** of the world to handle partial observability by keeping track of unobserved aspects based on past percepts.	Combines current percept with old internal state using a **transition model** (how world changes) and a **sensor model** (how percepts reflect world state); then	Taxi navigation needing to track other vehicles or ambiguous brake lights; vacuum cleaner without a location sensor.

		applies condition-action rules to the updated state.	
Goal-Based Agent	Uses **goal information** (descriptions of desirable states) to guide action selection, going beyond just the current state.	Combines current state knowledge with goals; often employs **search and planning algorithms** to find sequences of actions that will lead to goal achievement, considering future consequences.	Automated planning for spacecraft (NASA's Remote Agent); military logistics (DART); route-finding in GPS/maps; complex problem-solving in games with long-term objectives.
Utility-Based Agent	Employs a **utility function** to measure the desirability of world states, allowing for nuanced comparisons and trade-offs among conflicting objectives.	Selects actions that maximize **expected utility**, weighing the likelihood of outcomes against their value; handles uncertainty and multiple objectives by finding the best compromise.	Self-driving cars balancing safety, speed, and comfort; recommender systems; decision-making in multiagent competitive/cooperative environments; economic modeling.
Learning Agent	An agent with the ability to improve its **performance element** by acquiring new knowledge and adapting its behavior based on experience and feedback.	Consists of a **performance element** (the agent's core program), a **learning element** (makes improvements), a **critic** (provides feedback against a performance standard), and a **problem generator** (suggests exploratory actions).	Checkers-playing programs (Arthur Samuel's); ALPHAGO/ALPHAZERO; deep learning systems in vision, speech, and medical diagnosis; systems adapting to unknown user preferences.

Problem-Solving Agent	An agent focused on finding a sequence of actions to reach a goal, typically using **atomic representations** of states.	Follows a four-phase process: Goal formulation, Problem formulation (defining states, actions, costs), **Search** (simulating action sequences to find a solution), and Execution (performing the solution steps).	Route-finding; grid world puzzles (8-puzzle, Sokoban); airline travel planning; VLSI layout; robot navigation; automatic assembly sequencing; protein design.

Just as a master chef doesn't just know recipes (simple reflex), but understands the chemical reactions of ingredients (model-based), aims for a specific culinary vision (goal-based), and adjusts for guest preferences or dietary needs to maximize satisfaction (utility-based), an AI agent combines these layers of understanding and decision-making. And like the chef who learns from every meal cooked and guest feedback (learning agent), an AI continuously refines its abilities.



Study Material

(Introduction)

Table of Contents

Module No.	Module Name	Content
Module 1	Introduction to Artificial Intelligence	Problem Solving in AI
		Production systems
		State space representation

Problem Solving and State Space Representation

Problem-solving in Artificial Intelligence often involves agents navigating a **state space** to achieve a desired **goal**. This approach is fundamental to how intelligent systems find sequences of actions to reach a solution.

A **search problem** is a formal definition used in Artificial Intelligence to describe how an agent can find a sequence of actions, or a **path**, to achieve a desired **goal state** from an **initial state** within an environment. This process is central to how intelligent systems discover solutions.

A search problem is formally defined by the following components:

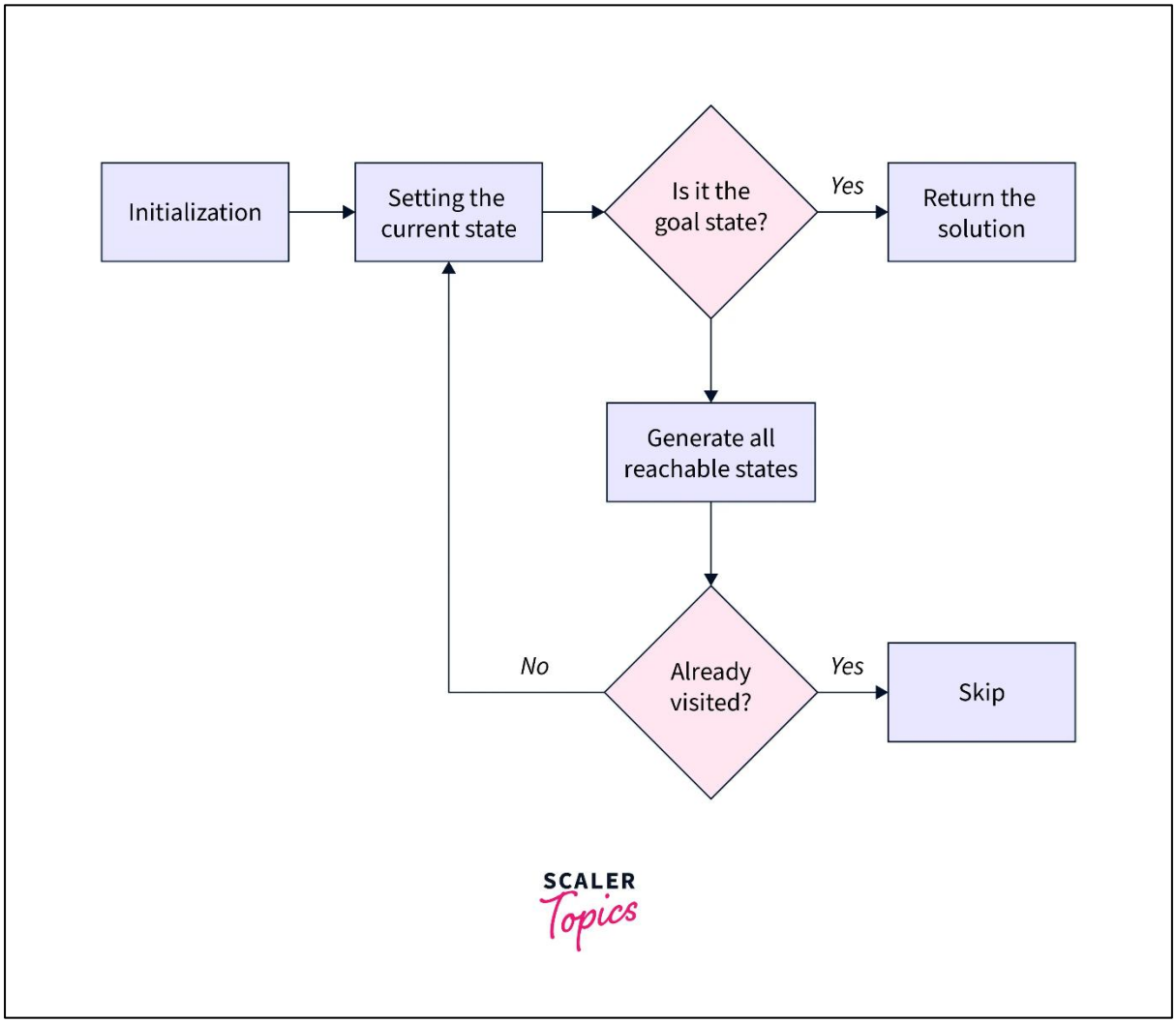
- **States**: A **set of possible states** that the environment can be in. Each state represents a complete snapshot of the world at a given moment. For example, in a vacuum cleaner world, a state might specify the agent's location and whether each square is dirty or clean. For the 8-puzzle, a state describes the location of each tile.
- **Initial State**: The **starting state** where the agent begins. In a route-finding problem, this could be the city of Arad.
- **Goal States**: A **set of one or more states** that represent the desired outcome. This can be a specific state (like Bucharest in a route-finding problem), a small set of alternatives, or a property that applies to many states (e.g., all squares being clean in a vacuum world). An `IS-GOAL` method is used to determine if a state satisfies the goal.
- **Actions**: The **actions available** to the agent from any given state `s`, which can be executed to transition to a new state. These actions are considered "applicable" in that state. In the two-cell vacuum world, actions include `Suck`, `move Left`, and `move Right`. For route-finding, actions are traveling between adjacent cities.

- **Transition Model**: This **describes what each action does**, mapping a current state `s` and an action `a` to a resulting state `s'` (denoted as `RESULT(s, a) = s'`). For example, applying `ToZerind` from `Arad` results in the `Zerind` state.
- **Action Cost Function**: A **numeric cost** associated with applying an action `a` in state `s` to reach state `s'`. This is denoted as `ACTION-COST(s,a,s')` or `c(s,a,s')`. For route-finding, this cost might be the distance in miles or time taken, and costs are assumed to be additive.

A **solution** to a search problem is a **path** (a sequence of actions) from the initial state to a goal state. An **optimal solution** is the path with the lowest total cost among all possible solutions.

A **state space representation** (or state space model) is essentially the formal structure that defines a search problem. It describes the environment in which an intelligent agent operates and aims to solve a problem.

- The **state space** can be visualized as a **graph**, where states are vertices (nodes) and actions are directed edges between them. For example, the map of Romania, with cities as states and roads as actions, is a graph representing a state space.
- It's important to distinguish between the **state space** (the set of all possible world configurations) and the **search tree**. While the state space might contain cycles (e.g., Arad to Sibiu and back to Arad), a search tree describes paths between these states, and each node in the tree has a unique path back to its root. Search algorithms often superimpose a search tree over the state-space graph.
- A key aspect of problem formulation is **abstraction**, which involves removing irrelevant details from the real-world representation to simplify the problem into a manageable state space. A good abstraction retains only details relevant to the agent's actions and goals, ensuring that any abstract solution can be elaborated into a real-world solution. Without useful abstractions, intelligent agents would be overwhelmed by the complexity of the real world.



Problem-solving through production systems is closely related to how agents make decisions within a state space. While the term "production system" itself is not a primary paradigm explicitly defined in the provided text, the underlying concepts are very present:

- **Condition-action rules** (also known as situation-action rules or if-then rules) are a fundamental building block. These rules directly map a current percept or state description to an action. Simple reflex agents operate entirely on these rules, selecting actions based solely on the current percept. This embodies a basic form of a production system, where a set of rules (`rules`) are applied by an interpreter (`RULE-MATCH`) based on the current state (`state← INTERPRET-INPUT(percept)`) to select an action.
- Historically, early AI systems like the **General Problem Solver (GPS)** by Allen Newell and Herbert Simon aimed to imitate human problem-solving protocols through symbolic manipulation. Their work led to the **physical symbol system hypothesis**, proposing that intelligence operates by manipulating data structures composed of

symbols. This aligns with the idea of a production system, where rules manipulate symbolic representations of the world to derive actions.

- The development of **expert systems** like DENDRAL and MYCIN in the 1970s and 80s further relied on "large numbers of special-purpose rules" to encode domain-specific knowledge. These systems used a "knowledge-intensive" approach, acquiring rules from human experts, which again reflects a rule-based or production system approach to problem-solving within a defined domain.

Applications and Use Cases of State Space Representation: The concept of state space representation is fundamental to solving a wide array of problems in AI, ranging from simplified benchmark puzzles to complex real-world challenges:

- **Standardized Problems**: These are often used to illustrate and test problem-solving methods.
  - **Grid World Problems**: Like the vacuum world or **Sokoban puzzles**, where agents navigate cells and manipulate objects.
  - **Sliding-Tile Puzzles**: Such as the **8-puzzle** or **15-puzzle**, where the goal is to arrange numbered tiles by sliding them into a blank space.
  - **Knuth's "4" Problem**: Illustrates how problems can have **infinite state spaces** by applying mathematical operations (square root, floor, factorial) to numbers to reach a desired integer.
- **Real-World Problems**:
  - **Route-Finding Problems**: Used extensively in **Web mapping services** (e.g., Google Maps) and **in-car navigation systems** to find optimal routes. Also applied in complex logistical challenges like **military operations planning** and **airline travel planning systems**.
  - **Touring Problems**: Such as the **Traveling Salesperson Problem (TSP)**, which seeks the lowest-cost tour visiting a set of locations, with applications in optimizing school bus routes.
  - **VLSI Layout**: Positioning millions of electronic components on a chip to minimize area and optimize performance, involving complex search problems for cell layout and channel routing.
  - **Robot Navigation**: A generalization of route-finding where robots move in continuous, multi-dimensional spaces, requiring advanced techniques to manage the complexity.
  - **Automatic Assembly Sequencing**: Determining the optimal order to assemble parts of complex objects, a standard industrial practice since the 1970s, also related to **protein design**.

In essence, the state space representation provides the structured map of possible scenarios and transitions for an AI agent. Problem-solving through production systems (or rule-based

reasoning, or search algorithms) then provides the means for the agent to navigate this map, making decisions about which actions to take to move from the current state towards a goal state, much like a meticulous explorer (the agent) uses a detailed map (the state space representation) and a comprehensive guide of local customs and survival techniques (the production system/rules) to journey from a starting point to a desired destination.

State Space Representation

A **search problem** is formally defined using a **state space representation**, which provides a structured way to describe the environment and the agent's interaction within it. Key components of this representation include:

- **States**: A **set of possible states** the environment can be in. Each state represents a complete snapshot of the world at a given moment. For example, in a vacuum cleaner world, a state might specify the agent's location and whether each square is dirty or clean. In a 3x3 8-puzzle, a state describes the arrangement of the eight numbered tiles and the blank space.
- **Initial State**: The **starting state** of the agent. For instance, starting in the city of Arad in a route-finding problem.
- **Goal States**: A **set of one or more states** that define the desired outcome. Sometimes it's a specific single state (e.g., Bucharest in a route-finding problem), a small set of alternatives, or a property that applies to many states (e.g., all squares being clean in a vacuum world, regardless of the agent's location).
- **Actions**: The **actions available** to the agent from a given state, which can be executed to transition to a new state. These actions are considered "applicable" in that state. For example, in the vacuum world, actions include "Suck," "move Left," or "move Right".
- **Transition Model**: This **describes what each action does**, mapping a current state and an action to a resulting state. For example, applying "ToZerind" from "Arad" results in the "Zerind" state.
- **Action Cost Function**: A **numeric cost** associated with applying an action in a state to reach a new state. For route-finding, this might be distance in miles or time taken. **Optimal solutions** are those with the lowest total path cost, assuming additive costs.

The **state space** can be visualized as a **graph**, where states are vertices and actions are directed edges between them. When agents search this graph, they construct a **search tree**, where nodes represent states and edges represent actions. A crucial distinction is that the search tree can have multiple paths (and thus multiple nodes) leading to the same state in the state space.

Abstraction is a key aspect of problem formulation, involving the removal of irrelevant details from a representation to simplify the problem. A good abstraction helps agents find solutions without being overwhelmed by the complexity of the real world.

Problem-Solving through Production Systems

While the sources do not explicitly use the term "production systems" as a primary paradigm, they describe similar concepts, particularly in the context of how agents make decisions based on perceived conditions and stored knowledge.

The closest concept discussed in the provided text is the **"condition–action rule"**. These rules, also known as situation–action rules or if–then rules, are the basis of **simple reflex agents**. These agents select actions directly based on the current percept, effectively ignoring past percepts. For instance, a simple vacuum agent's behavior "if current square is dirty, then suck; otherwise, move to the other square" is an example of a condition-action rule. This can be implemented as a general-purpose interpreter for such rules, where an `INTERPRET-INPUT` function abstracts the current state, and `RULE-MATCH` finds a rule that matches this state description to return an action.

For more complex scenarios, especially in partially observable environments, simple reflex agents are limited because they cannot maintain an internal understanding of the world. This leads to **model-based reflex agents**, which maintain an "internal state" representing unobserved aspects of the current world. This internal state is updated using a "transition model" (how the world changes over time, including effects of agent actions) and a "sensor model" (how the world state is reflected in percepts). Once the internal state is updated, these agents still use condition-action rules to select an action.

Furthermore, the "early enthusiasm" period of AI, particularly with systems like **Logic Theorist (LT)** and the **General Problem Solver (GPS)** by Newell and Simon, embodies problem-solving through **symbolic manipulation and reasoning**. GPS was designed to imitate human problem-solving protocols, leading to the **physical symbol system hypothesis**, which posits that any intelligent system (human or machine) operates by manipulating data structures composed of symbols. This aligns with the principles of rule-based or production systems, where knowledge is explicitly represented and manipulated to derive actions.

The later development of **expert systems** (e.g., DENDRAL, MYCIN) also relied on "large numbers of special-purpose rules" to encode domain-specific knowledge, moving away from "weak methods" of general search. This "knowledge-intensive" approach involved acquiring rules from human experts, demonstrating another form of problem-solving driven by a collection of production-like rules.

Applications and Use Cases of State Space Representation

The concept of state space representation is widely applied across various AI domains:

Standardized Problems (Benchmarks):

- **Grid World Problems**: Agents move between cells, interacting with objects or obstacles. Examples include the vacuum world (agent cleans dirty squares) and **Sokoban puzzles** (agent pushes boxes to target locations).
- **Sliding-Tile Puzzles**: Such as the **8-puzzle** or **15-puzzle**, where tiles are slid into a blank space to reach a goal configuration.
- **Knuth's "4" Problem**: An example illustrating problems with **infinite state spaces** where actions (square root, floor, factorial) transform numbers to reach a desired integer.

Real-World Problems:

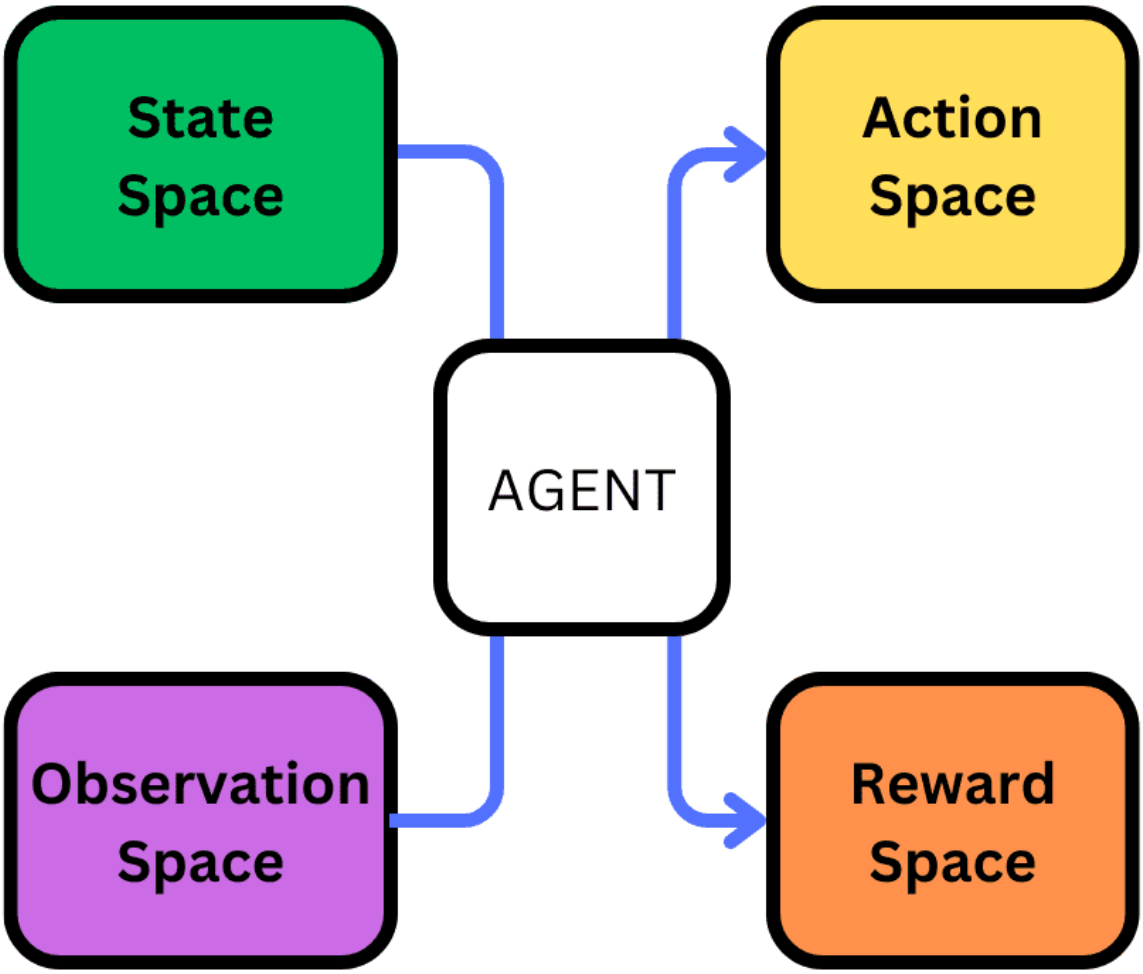
- **Route-Finding Problems**: Used in **Web mapping services** (like Google Maps) and **in-car navigation systems** to find optimal routes between locations, considering factors like traffic. Also applied in complex logistical challenges such as **military operations planning** and **airline travel planning**.
- **Touring Problems**: Involve visiting a set of locations, like the **Traveling Salesperson Problem (TSP)**, which aims to find the lowest-cost tour visiting every city. This has applications in optimizing **school bus routes**.
- **VLSI Layout**: Positioning millions of electronic components and connections on a chip to minimize area, circuit delays, and maximize manufacturing yield.
- **Robot Navigation**: A generalization of route-finding, where a robot moves in a continuous multi-dimensional space (e.g., controlling arms and legs).
- **Automatic Assembly Sequencing**: Determining the optimal order to assemble parts of complex objects, a standard practice in industry since the 1970s. This also extends to **protein design**, finding amino acid sequences for specific protein properties.

In essence, **state space representation** provides the map and rules for an AI agent's journey, while the agent's **program (akin to a production system or logical rules)** acts as the set of instructions and decision-making mechanisms for navigating that map effectively. It's like a chef (the agent) who has a detailed recipe book (the production system/rules) to transform ingredients (the current state) into a delicious meal (the goal state) by following steps (actions) and knowing how each step changes the dish (transition model), all while being mindful of the cost of each ingredient and cooking step (action cost function).

Search Problems and their Relation with State Space

A **search problem** provides a formal structure for an intelligent agent to discover a sequence of actions, or a **path**, that leads from an **initial state** to a desired **goal state** within an environment. It's a fundamental concept in Artificial Intelligence (AI) for finding solutions to problems.

A **state space model** is the formal representation of the environment in which a search problem is defined. It essentially maps out all possible configurations of the world and the transitions between them.



Here's how they relate to intelligent agents:

- **Components of a Search Problem / State Space Model**: A search problem, and by extension its state space model, is formally defined by these key elements:
    - **States**: A **set of possible configurations** the environment can be in, representing a complete snapshot of the world. For example, in the 8-puzzle, a state describes the arrangement of all tiles.
    - **Initial State**: The **specific starting point** for the agent. In a route-finding problem like navigating Romania, this would be the city of Arad.
    - **Goal States**: One or more **desired configurations** that the agent aims to reach. This can be a specific state (like Bucharest), a set of states, or a property that defines success (e.g., all squares being clean in a vacuum world). An `IS-GOAL` method determines if a state meets the goal criteria.
    - **Actions**: The **set of available moves** an agent can perform from any given state `s`. These actions are "applicable" in `s`. For instance, from Arad, actions could be `ToSibiu`, `ToTimisoara`, or `ToZerind`.
    - **Transition Model**: This **describes the outcome** of performing an action `a` in state `s`, leading to a new state `s'` (denoted `RESULT(s, a) = s'`).
    - **Action Cost Function**: A **numeric cost** associated with each action, typically `ACTION-COST(s, a, s')`. Costs are usually additive, meaning the total path cost is the sum of individual action costs. This cost should reflect the agent's performance measure.
- **Agents and their Interaction with State Space Models**:
    - **Problem-Solving Agents**: These agents explicitly **devise an abstract model** (a state space representation) of the relevant part of the world to plan actions. Before acting in the real world, they **simulate sequences of actions** within this model, searching for a solution. The solution is a "path," a sequence of actions from the initial state to a goal state.
    - **Intelligent Agent Paradigm**: The core idea in AI is the **intelligent agent**, which perceives its environment through sensors and acts upon it through actuators. The **state space** *is* this environment, or at least the relevant part of it that the agent needs to consider.
    - **Abstraction**: A crucial step in formulating a problem is **abstraction**, which involves removing irrelevant details from the real world to simplify it into a manageable state space. A good abstraction ensures that solutions found in the abstract state space can be translated back into real-world actions. Without abstraction, intelligent agents would be overwhelmed by the complexity of the real world.
    - **Internal State and Models**:
        - **Simple Reflex Agents** are the most basic, acting solely on the **current percept**. They *do not* maintain an internal model of the state space

beyond what is immediately perceived, making them limited in partially observable environments where important information might be hidden.

- **Model-Based Reflex Agents** overcome partial observability by explicitly maintaining an **internal state** that tracks unobserved aspects of the current world state. This internal state is updated using a **transition model** (how the world changes, including the effects of the agent's actions) and a **sensor model** (how the world state is reflected in percepts). This internal state provides the agent's "best guess" of "what the world is like now".
- **Goal-Based Agents** leverage the understanding of the current state (often from a model) and combine it with **goal information** to decide actions that will eventually achieve their objectives. These agents use search and planning algorithms to navigate their internal state space representation to find paths to goals.
- **Utility-Based Agents** go beyond simple goals by incorporating a **utility function**, which is an internalization of the performance measure. This allows them to compare the desirability of different states and action sequences within the state space, especially when goals conflict or outcomes are uncertain. They aim to maximize "expected utility" within the state space.

- **Learning Agents**: These agents are designed to **improve their components** over time, including their internal models of the state space. They use feedback from a "critic" to understand how well they are doing and adjust their understanding of "what my actions do" and "how the world evolves". This means their internal state space model can evolve and become more accurate through experience.

- **State Space as a Graph**: The state space can be conceptualized as a **graph**, where each state is a **vertex (node)** and each action is a **directed edge** connecting states. Search algorithms, such as **best-first search**, explore this graph by expanding nodes and generating child nodes corresponding to successor states. It's crucial to distinguish between the abstract state space graph and the **search tree**, which is a path-dependent structure built *over* the state space during the search process, and can contain multiple nodes (paths) leading to the same state.

In essence, the state space model is the **map** of the problem, detailing every possible location (state) and every route (action) between them, along with their associated costs. The intelligent agent is the **traveler** who uses this map (or builds one through experience) and its internal decision-making system (reflexes, goals, utility functions, and learning capabilities) to navigate from its current position (initial state) to its desired destination (goal state), striving to find the most efficient or optimal path.

A **state space model** is the formal representation of an environment within which an intelligent agent solves a **search problem** [previously discussed]. It essentially maps out all possible configurations of the world and the transitions between them, allowing an agent to plan a sequence of actions to achieve a goal [previously discussed, 334].

Here's a comprehensive breakdown of state space models, including their graph representation, working principles, application areas, and real-world problem-solving capabilities:

Representation as a Graph

A search problem, which operates within a state space model, is formally defined by several key elements:

- **States**: These are a set of possible configurations the environment can be in, representing a complete snapshot of the world. For example, in the 8-puzzle, a state describes the arrangement of all tiles [previously discussed].
- **Initial State**: This is the specific starting point for the agent. For instance, if an agent is navigating Romania, its initial state might be the city of Arad.
- **Goal States**: These are one or more desired configurations the agent aims to reach. A method `IS-GOAL` determines if a state meets the criteria for success, such as all squares being clean in a vacuum world.
- **Actions**: These are the available moves an agent can perform from any given state `s`. `ACTIONS(s)` returns a finite set of applicable actions. For example, from Arad, actions could be `ToSibiu`, `ToTimisoara`, or `ToZerind`.
- **Transition Model**: This describes the outcome of performing an action `a` in state `s`, leading to a new state `s'` (denoted `RESULT(s, a) = s'`).
- **Action Cost Function**: A numeric cost, `ACTION-COST(s, a, s')` or `c(s, a, s')`, is associated with each action, typically reflecting the agent's performance measure. Total path cost is usually the sum of individual action costs.

The **state space can be represented as a graph** where each state is a **vertex (node)** and each action is a **directed edge** connecting states. For example, the map of Romania, showing cities as states and roads as actions, is such a graph. It is important to distinguish this abstract state space graph from the **search tree**, which is a path-dependent structure built *over* the state space during the search process, and can contain multiple nodes (paths) leading to the same state. The search tree's root node corresponds to the initial state, and expanding a node involves generating new child nodes (successor nodes) for each resulting state via applicable actions.

Working Principle: Problem-Solving by Searching

Intelligent agents, particularly **problem-solving agents**, explicitly devise an abstract model (a state space representation) of the relevant part of the world to plan actions [previously discussed, 334]. Their interaction with state space models follows a four-phase process:

1. **Goal Formulation**: The agent defines its objective, which helps limit the actions to consider.
2. **Problem Formulation**: The agent describes the states and actions needed to reach the goal, creating an abstract model of the environment.
3. **Search**: Before acting in the real world, the agent **simulates sequences of actions** within this model, searching for a **solution** (a path from the initial state to a goal state). This process involves exploring the state-space graph by expanding nodes and maintaining a **frontier** of unexpanded nodes. Search algorithms aim to find an optimal solution, which has the lowest path cost.
4. **Execution**: Once a solution path (sequence of actions) is found, the agent executes these actions in the real world, one at a time. In fully observable, deterministic, and known environments, this can be an "open-loop" system where percepts are ignored during execution. However, in uncertain environments, a "closed-loop" approach monitoring percepts is safer.

Search algorithms can be:

- **Uninformed Search Strategies**: These algorithms operate without any domain-specific hints about the closeness of a state to the goal. Examples include **breadth-first search** (which is complete and optimal for equal-cost actions but memory-intensive) and **uniform-cost search** (which is complete and cost-optimal for varying action costs). They can face challenges with exponential complexity and getting stuck in infinite loops in cyclic or infinite state spaces if not handled carefully.
- **Informed (Heuristic) Search Strategies**: These use **domain-specific hints** in the form of a **heuristic function**, `h(n)`, which estimates the cost from a node `n` to a goal state. Examples include:
    - **Greedy best-first search**, which expands nodes closest to the goal according to the heuristic `h(n)`. It is fast but not guaranteed to find an optimal solution.
    - **A* search**, which is the most common informed search algorithm, uses an evaluation function `f(n) = g(n) + h(n)`, where `g(n)` is the path cost from the initial state and `h(n)` is the estimated cost to the goal. **A*** is complete and cost-optimal if its heuristic is **admissible** (never overestimates the true cost) and **consistent** (satisfies the triangle inequality). It prunes away unnecessary search tree nodes, making it efficient.

To manage the exponential time and space complexity often encountered in large state spaces, advanced techniques are employed:

- **Redundant Paths/Cycles**: Search algorithms keep track of previously reached states to avoid repeatedly exploring the same parts of the state space.
- **Memory-bounded search**: Algorithms like IDA* (Iterative-Deepening A*) and SMA* (Simplified MA*) are designed to work within memory constraints by revisiting nodes or discarding the "worst" nodes, trading time for space.
- **Bidirectional search**: This approach simultaneously searches forward from the initial state and backward from the goal state, aiming for the two searches to meet in the middle, potentially reducing the search space significantly.

Application Areas

State space models and search algorithms are fundamental to various AI applications:

- **Standardized Problems** (benchmarks for algorithms):
  - **Vacuum World**: A simple grid environment where the agent cleans dirty squares.
  - **Sokoban Puzzle**: Involves pushing boxes to designated storage locations in a grid.
  - **Sliding-Tile Puzzles** (e.g., 8-puzzle, 15-puzzle): Arranging numbered tiles in a grid by sliding them into a blank space.
  - **Knuth's "4" Problem**: Reaching any positive integer starting from 4 using square root, floor, and factorial operations, which can lead to infinite state spaces.
- **Real-World Problems** (solutions used by people):
  - **Route-Finding**: Used in navigation systems (e.g., car GPS, Google Maps) and airline travel planning, accounting for factors like traffic, costs, and connections.
  - **Touring Problems** (e.g., Traveling Salesperson Problem (TSP)): Visiting a set of locations with minimal cost, applicable to vehicle routing and logistics.
  - **VLSI Layout**: Positioning millions of components on a chip to minimize area, delays, and maximize yield.
  - **Robot Navigation**: Generalizing route-finding for robots in continuous, multi-dimensional spaces, dealing with sensor errors and partial observability.
  - **Automatic Assembly Sequencing**: Determining the optimal order to assemble parts of an object.
  - **Protein Design**: Finding amino acid sequences that fold into specific 3D protein structures to cure diseases.

Real-World Problem-Solving Capabilities

The effectiveness of state space models in real-world problem solving hinges on several factors:

- **Abstraction**: A crucial step in formulating a problem is **abstraction**, which involves removing irrelevant details from the real world to simplify it into a manageable state space [previously discussed, 347, 348]. A good abstraction ensures that solutions found in the abstract state space can be translated back into real-world actions effectively. Without abstraction, intelligent agents would be overwhelmed by complexity.
- **Planning Ahead**: State space models allow **agents to simulate sequences of actions** and their outcomes *before* acting in the physical environment. This foresight enables goal-based agents and utility-based agents to choose actions that lead to desirable future states.
- **Agent Architecture**: Different types of agents leverage state space models in varying ways.
    - **Model-based reflex agents** maintain an **internal state** (a "best guess" of "what the world is like now") using a **transition model** (how the world changes) and a **sensor model** (how percepts reflect the world) to handle partial observability.
    - **Goal-based agents** combine this internal model with **goal information** to find action sequences that lead to desired states.
    - **Utility-based agents** use a **utility function** (an internalization of the performance measure) to compare different world states and choose actions that maximize expected utility, especially when goals conflict or outcomes are uncertain.
- **Learning**: Agents can **learn and improve their internal models** of the state space based on experience. This allows them to operate in initially unknown environments and become more competent over time. Learning agents use feedback from a "critic" to modify their "performance element" and improve their understanding of how actions affect the world and the utility they derive.
- **Managing Complexity**: While problems can have an enormous number of states (e.g., 10 trillion states for the 15-puzzle, 10^150 entries for chess look-up tables), the development of efficient search algorithms, especially informed ones with good heuristics, allows agents to find solutions in manageable time and space. However, even with these advances, **perfect rationality is often unachievable in practice due to computational complexity**. The ongoing challenge is to find programs that produce rational behavior from small inputs, rather than vast pre-computed tables.

In essence, the state space model is the **blueprint** of the problem an intelligent agent faces, outlining every possible configuration and the available transitions. The intelligent agent acts as the **architect and explorer**, using this blueprint (or building it through learning) to devise and execute a plan. Just as an architect uses a blueprint to design a building, an AI agent uses

a state space model to design a sequence of actions that will lead to its goal, navigating the vast possibilities while striving for efficiency and optimality.

Key Concepts in Search Algorithms:

• **Search Tree Data Structures**: Nodes in the search tree are represented with four components: `node.STATE`, `node.PARENT`, `node.ACTION`, and `node.PATH-COST` (or `g(node)`).

• **Frontier Management**: The frontier can be managed by a **priority queue**, a **FIFO queue** (for breadth-first search), or a **LIFO queue** (stack, for depth-first search).

• **Reached States**: Search algorithms keep track of previously reached states (nodes that have been generated) to avoid repeatedly exploring the same parts of the state space, especially to detect **redundant paths** or **cycles**. A **graph search** algorithm explicitly checks for redundant paths.

• **Measuring Problem-Solving Performance**: Algorithms are evaluated on:

◦ **Completeness**: Whether the algorithm is guaranteed to find a solution if one exists.

◦ **Cost Optimality**: Whether it finds the solution with the lowest path cost.

◦ **Time Complexity**: How long it takes to find a solution, often expressed using Big O notation.

◦ **Space Complexity**: How much memory is needed.

Types of Search Strategies:

• **Uninformed Search Strategies**: These algorithms operate without any domain-specific hints about the closeness of a state to the goal.

◦ **Breadth-first search**: Expands the root first, then all successors at depth 1, then depth 2, and so on. It is complete and optimal for equal-cost actions, but **memory-intensive** with O(b^d) time and space complexit.

◦ **Uniform-cost search (Dijkstra's algorithm)**: Expands the node with the lowest **path cost (g(n))** first. It is complete and cost-optimal for varying action costs .

◦ **Depth-first search**: Always expands the deepest node in the frontier first. It is not cost-optimal and can get stuck in infinite loops in cyclic or infinite state spaces if not handled 406]. However, it is memory-efficient (O(bm)) for tree-like searches.

◦ **Depth-limited search**: A version of depth-first search with a specified depth limit `l`.

◦ **Iterative deepening search (IDS)**: Combines the memory benefits of depth-first search with the completeness and optimality (for equal costs) of breadth-first search by repeatedly calling depth-limited search with increasing depth limits (0, 1, 2, etc.).

◦ **Bidirectional search**: Simultaneously searches forward from the initial state and backward from the goal state(s), aiming for the two searches to meet in the middle, potentially reducing the search space significantly (O(b^d/2)).

• **Informed (Heuristic) Search Strategies**: These use **domain-specific hints** in the form of a **heuristic function**, `h(n)`, which estimates the cost from a node `n` to a goal state.

◦ **Greedy best-first search**: Expands the node with the lowest `h(n)` value. It is fast but not guaranteed to find an optimal solution.

◦ **A* search**: The most common informed search algorithm, uses an evaluation function `f(n) = g(n) + h(n)`. **A*** is complete and cost-optimal if its heuristic is **admissible** (never overestimates the true cost) and **consistent** (satisfies the triangle inequality: `h(n) ≤ c(n,a,n') + h(n')`). It prunes away unnecessary search tree nodes, making it efficient 448].

◦ **Satisficing Search**: Aims to find "good enough" suboptimal solutions more quickly, often using **inadmissible heuristics** that might overestimate costs.

▪ **Weighted A* search**: Uses `f(n) = g(n) + W × h(n)` for some `W > 1`, exploring fewer nodes at the cost of potential suboptimality.

◦ **Memory-bounded Search**: Algorithms designed to work within memory constraints:

▪ **Beam search**: Limits the size of the frontier by discarding nodes with lower f-scores, making it incomplete but faster.

▪ **Iterative-deepening A* search (IDA*)**: Combines A* with iterative deepening, allowing it to work in linear space by revisiting nodes.

▪ **Recursive best-first search (RBFS)**: Mimics A* with linear space, but can suffer from excessive node re-generation.

▪ **MA* (memory-bounded A*) and SMA* (simplified MA*)**: Use all available memory by dropping the "worst" leaf nodes when memory is full.