

ARTIFICIAL INTELLIGENCE AND AUTOMATION





Let's get the Ground rules Straight

- 1. Presence is Prerequisite: Attendance is Mandatory.**
- 2. Deadlines are Commitments: Respect the Schedule.**
- 3. Assignments are Training, Not Optional.**
- 4. Your Work Must Be Your Own: Uphold Academic Integrity.**
- 5. Assessments are Your Proof of Mastery.**
- 6. This is a Partnership: My Commitment to You.**



Welcome to the world of AI and Automation!

- In this course, you'll explore how machines can think, reason, learn, and make decisions.
- Discover how automation is transforming industries by replacing repetitive tasks with smart systems.
- From puzzles to chatbots, learn to build intelligent and automated solutions.





Course Objectives

- Understand the **fundamentals of AI** and its potential for decision making.
- Introduce the concept of **artificial intelligence**, methods, techniques and applications.
- Gain **practical experience** through case studies and hands-on projects.



Course Outcomes

- Apply the foundational concepts of AI to perform logical reasoning.
- Analyse and implement classical and heuristic search algorithms to solve complex AI problems.
- Apply probabilistic reasoning techniques to represent and infer knowledge under uncertainty in AI systems.
- Analyse decision-making models to optimize AI-driven strategic and sequential decision-making under uncertainty.
- Design and implement AI-driven automation systems and workflows using appropriate tools to streamline tasks and enhance operational efficiency across diverse domains.

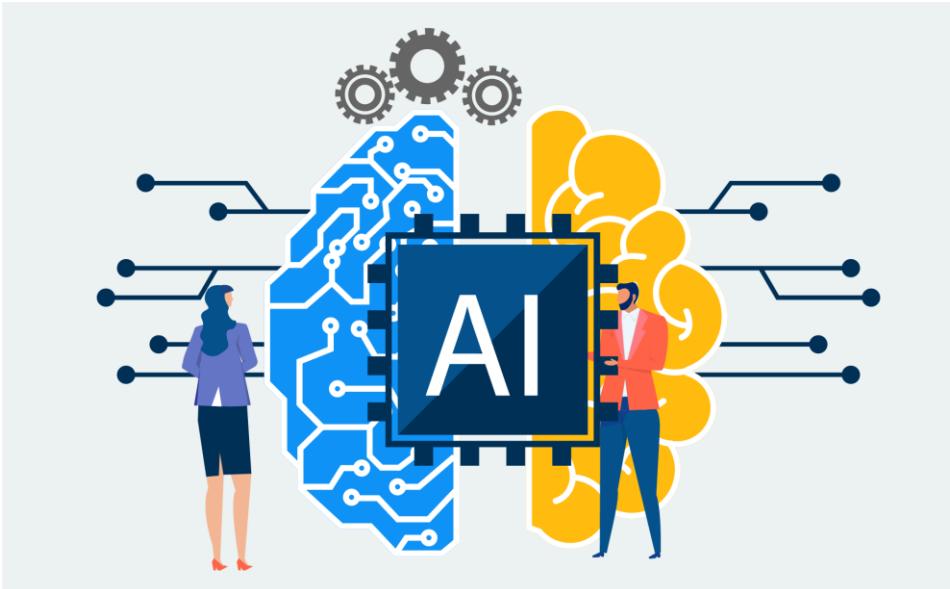


What You Will Learn (in simple terms)

- Understand what makes machines 'intelligent' – from logic to learning.
- Solve problems using AI – games, planning, and decision-making.
- Handle uncertainty with Bayesian networks and HMMs.
- Make smart decisions using probabilities and game strategies.
- Explore automation and Robotic Process Automation (RPA) tools like UiPath.



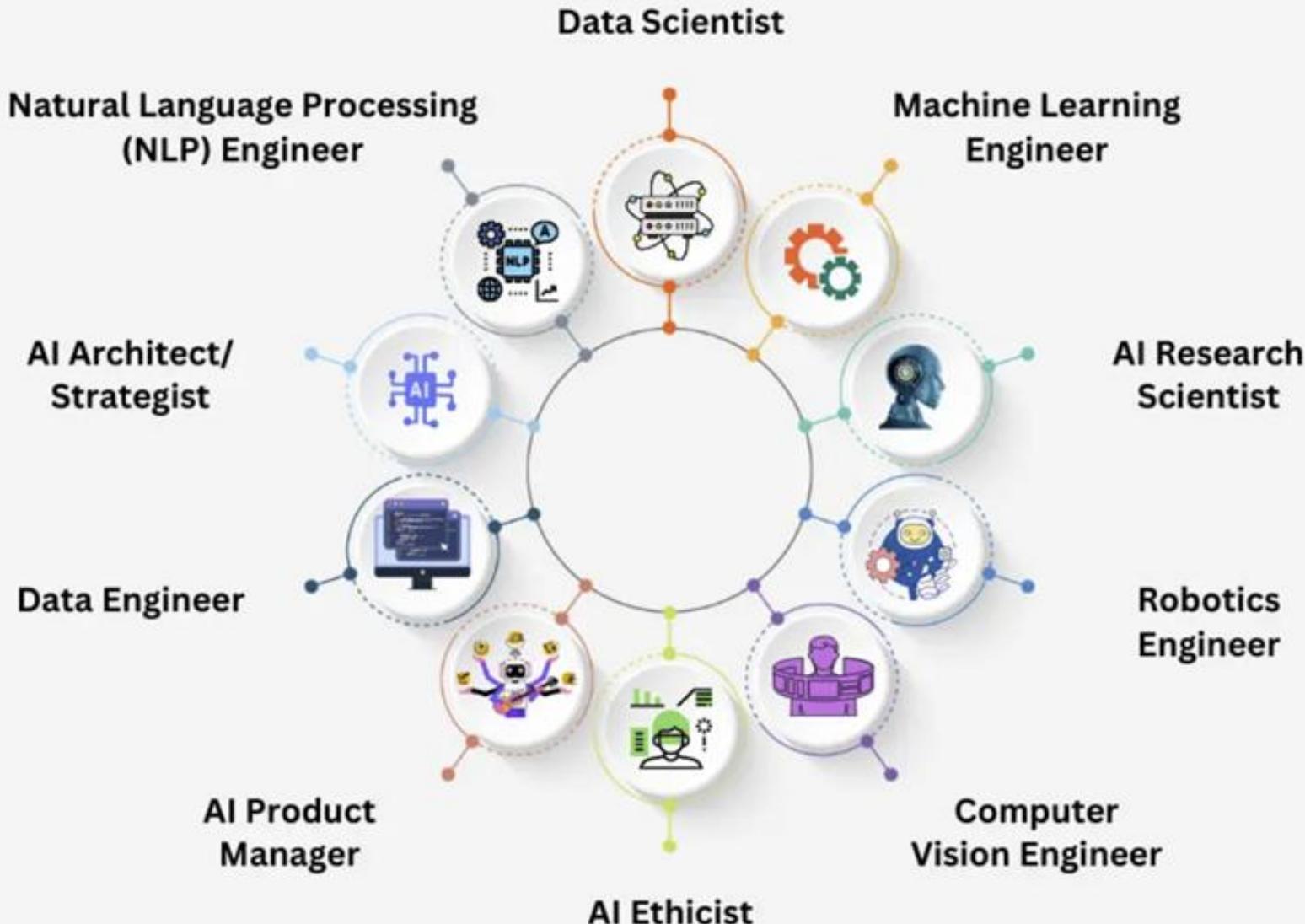
Why It Matters?



- AI powers self-driving cars, voice assistants, and recommendation systems.
- Automation enhances productivity across industries – engineering, healthcare, finance, and more.
- Gain essential skills for future-ready careers in AI and Automation.



Top 10 Career Opportunities in Artificial Intelligence





Syllabus

Unit I – Introduction to AI

Unit II – Problem Solving

Unit III – Representing and Reasoning with Uncertain Knowledge

Unit IV – Decision Making

Unit V – Artificial Intelligence for Automation



Hands-On Practice

- Create intelligent agents and simulate logic-based behavior.
- Build search algorithms and game-based AI systems (Tic-Tac-Toe, Sudoku).
- Apply probabilistic reasoning in real-world problems.
- Automate tasks using UiPath and other RPA tools.





Unit I – Introduction to AI

Fundamentals of AI - Definitions, Key concepts, Intelligent agents, Agents and Environment. Propositional Logic – Agents based on Propositional Logic – First order logic – Syntax and semantics – Knowledge Engineering in First Order Logic– Inference – Unification - Forward and backward chaining - Resolution.



Unit II – Problem Solving

State space search; production systems, search space control; depth first search, breadth-first search. Heuristic Based Search: Hill climbing, best-first search, A*Algorithm and AO* algorithm, Min-max algorithms, game playing – Alpha beta pruning branch and bound, Problem Reduction, Constraint Satisfaction.



Unit III – Representing and Reasoning with Uncertain Knowledge

Handling uncertainty in AI, Probability theory and its connection to logic, Concepts of independence and conditional probability, Structure of Bayesian Networks, Bayesian rule and its applications, Markov Models and Hidden Markov Models (HMMs), Probabilistic graphical models and Inference algorithms.



Unit IV – Decision Making

Importance of decision making in AI, Utility, preferences and Expected utility in decision-making under uncertainty, Decision Theory Basics, Markov Decision Processes (MDPs), Game theory and strategic decision-making in AI.



Unit V – Artificial Intelligence for Automation

Understanding Automation, Applications of AI-driven Automation, Opportunities and challenges in AI automation. Automation in production systems- Automation principles and strategies-Basic elements of an automated system.

Introduction to Robotic Process Automation- Benefits of RPA, Components of RPA- RPA Platforms-About Ui Path.



Get Ready To Explore...

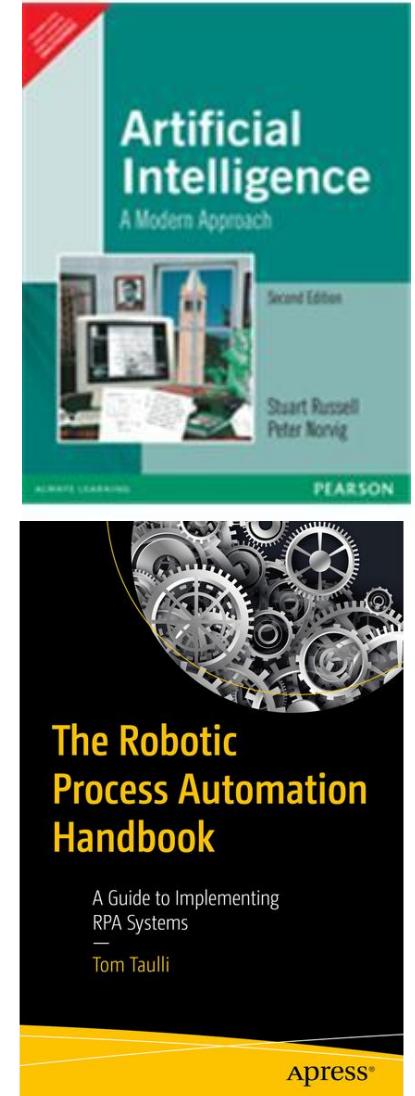
- Intelligent Agents – Simulate basic AI behaviour.
- Problem Solving – Develop puzzle and game solvers.
- Learning with Uncertainty – Model and predict outcomes.
- Decision Making – Program smart choices.
- AI for Automation – Build bots using RPA platforms.





Textbooks

- Stuart Russell, Peter Norvig, “Artificial Intelligence – A Modern Approach”, 4th Edition, Pearson Education / Prentice Hall of India, 2022.
- Tom Taulli, “The Robotic Process Automation Handbook: A Guide to Implementing RPA Systems”, A press Publications, 2020.





Reference books

1. George F. Luger, “Artificial Intelligence-Structures and Strategies for Complex Problem Solving”, Sixth Edition, Pearson Education, 2011.
2. Rich E., Knight K. and Nair B. S., Artificial Intelligence, Tata McGraw Hills, Third Edition, 2009.
3. Alok Mani Tripathi, “Learning Robotic Process Automation: Create Software robots and automate business processes with the leading RPA tool – UiPath”, Packt Publishing, 2018.



Online Resources (Recommended)

- <https://www.coursera.org/learn/ai-for-everyone?msocid=31c77fa2444f66f50f8f6bae459d6774>
- <https://www.coursera.org/specializations/roboticprocessautomation>



Why Study Artificial Intelligence?

AI is considered a subject most worthy of study for three main reasons:

- **It's a Fundamental Quest:** It addresses our age-old desire to understand our own intelligence by attempting to replicate it.
- **It's a Frontier of Science:** Unlike older fields, AI is young and full of unsolved problems, offering opportunities for groundbreaking discoveries.
- **It's a Universal Field:** Its principles can be applied to virtually any intellectual task, from creative arts to medical diagnosis and autonomous driving.



What Exactly is AI? Four Approaches

- The text explains that AI can be defined in four different ways, based on whether the goal is to replicate **human-like** behavior or achieve perfect **rationality**, and whether the focus is on the internal **thought process** or the external **action**.
- Here are the four categories with real-time examples:
 1. **Acting Humanly: The Turing Test Approach**
 2. **Thinking Humanly: The Cognitive Modeling Approach**
 3. **Thinking Rationally: The "Laws of Thought" Approach**
 4. **Acting Rationally: The Rational Agent Approach**



What Exactly is AI? Four Approaches

1. Acting Humanly: The Turing Test Approach

- **The Goal:** To create a machine that can perform tasks in a way that is indistinguishable from a human. The focus is on the *output or behavior*, not the internal reasoning.
- **Real-Time Example: Advanced Chatbots and Generative AI**
 - When you interact with a sophisticated customer service bot or use **ChatGPT** for a complex conversation, the goal is for its responses to be so natural and helpful that you might not be able to tell it's a machine.
 - This is a direct application of the Turing Test principle. Similarly, AI image generators like **Midjourney** create art that can be mistaken for human work.



What Exactly is AI? Four Approaches

1. Acting Humanly: The Turing Test Approach

The Standard Turing Test

- **Proposer and Goal:** Proposed by Alan Turing in 1950 as a practical, operational definition of intelligence.
- **The Test:** A human interrogator asks written questions to both a human and a computer.
- **Passing Condition:** The computer passes if the interrogator cannot reliably tell which one is the machine.
- **Required AI Capabilities:** To pass, a computer would need to master four key areas:
 - **Natural Language Processing:** To understand and generate human language (e.g., English).
 - **Knowledge Representation:** To store and retrieve what it learns or is told.
 - **Automated Reasoning:** To use stored knowledge to answer questions and draw new conclusions.
 - **Machine Learning:** To adapt to new situations by detecting and extrapolating patterns.



What Exactly is AI? Four Approaches

1. Acting Humanly: The Turing Test Approach

The Total Turing Test

- **Concept:** An expanded version of the test that includes physical interaction.
- **Additions:** It adds a video feed and the ability to pass physical objects to the machine.
- **Additional AI Capabilities:** To pass this more comprehensive test, a computer would also need:
 - **Computer Vision:** To perceive and identify objects.
 - **Robotics:** To manipulate objects and move around in the physical world.

Modern View and Critique of the Test

- **Relevance:** The six disciplines identified by the Standard and Total Turing Tests (NLP, knowledge representation, reasoning, learning, vision, and robotics) still compose most of the AI field.
- **Researchers' Focus:** Most AI researchers do not actively try to pass the Turing Test.
- **The Core Argument:** It is considered more important to understand the **underlying principles of intelligence** rather than simply imitating a human.
- **The Flight Analogy:** The quest for flight succeeded not when inventors imitated birds, but when they studied the principles of **aerodynamics**. Similarly, the goal of AI is not to build machines that can "fool" humans, but to understand and engineer the components of intelligence itself.



What Exactly is AI? Four Approaches

2. Thinking Humanly: The Cognitive Modeling Approach

- **The Goal:** To build a system that thinks like a human, meaning it models the internal reasoning steps, including our potential errors and biases. The process is more important than the result.
- **Real-Time Example: Neuromorphic Computing Research**
 - Scientists are building AI models that mimic the structure and function of the human brain's neural networks.
 - For instance, an AI designed to model human vision might be programmed to be susceptible to the same **optical illusions** that fool people. This helps researchers in both AI and psychology understand how the brain itself processes information.



What Exactly is AI? Four Approaches

2. Thinking Humanly: The Cognitive Modeling Approach

- The Cognitive Modeling Approach
- **The Goal:** To build a program that thinks like a human. The focus is on modeling the internal reasoning process, not just the external behavior.
- **The Prerequisite:** To achieve this, we must first have a way to understand how humans actually think.

Methods for Understanding Human Thought: The text identifies three primary methods:

- **Introspection:** Trying to observe and record our own thoughts as they happen.
- **Psychological Experiments:** Observing a person's behavior and actions while performing a task.
- **Brain Imaging:** Observing the physical brain in action using technologies like fMRI.
- **The Validation Process:**
 - Develop a precise theory about how the human mind works.
 - Express that theory as a computer program.
 - Compare the program's outputs and, more importantly, its **internal reasoning steps** to the behavior and thought processes of actual humans. A match provides evidence that the program's mechanisms may be similar to those in the human mind.



What Exactly is AI? Four Approaches

2. Thinking Humanly: The Cognitive Modeling Approach

- **Key Example: The General Problem Solver (GPS):**
 - Its developers, Newell and Simon, were not satisfied just because their program could solve problems.
 - They were primarily concerned with **comparing the trace of the program's reasoning steps** to the verbal reports of humans solving the same problems.
- **The Resulting Field: Cognitive Science:**
 - This approach led to the interdisciplinary field of **cognitive science**, which brings together AI's computational models and psychology's experimental techniques to create precise, testable theories of the mind.
- **Modern Distinction and Progress:**
 - In early AI, researchers often confused good performance with being a good model of human thought.
 - Modern science now **separates these two claims**. This distinction has allowed both AI (focused on performance) and cognitive science (focused on modeling) to advance more rapidly.
 - Despite this separation, the fields continue to inform each other, particularly in areas like computer vision.



What Exactly is AI? Four Approaches

3. Thinking Rationally: The "Laws of Thought" Approach

- **The Goal:** To create a system that operates on the principles of formal logic. It's about making provably correct inferences and "right thinking" based on a set of facts and rules.
- **Real-Time Example: Formal Verification of Microchips**
 - Before manufacturing a new processor like those from Intel or Apple, companies use AI-based theorem provers.
 - These systems use formal logic to analyze the chip's design and **prove mathematically** that it is free from critical logical errors. This is a high-stakes application where perfect, irrefutable reasoning is essential.



What Exactly is AI? Four Approaches

The "Laws of Thought" Approach

- **The Goal:** To create intelligent systems based on the principles of formal logic, or the "laws of thought." The focus is on achieving provably correct, irrefutable reasoning.
- **Historical Origins:**
 - This approach traces back to the Greek philosopher **Aristotle**, who first tried to codify "right thinking."
 - His **syllogisms** (e.g., "Socrates is a man; all men are mortal; therefore, Socrates is mortal") are early examples of argument structures that guarantee correct conclusions from correct premises.
- **Development into a Field:**
 - The study of these reasoning patterns initiated the field of **logic**.
 - By the 19th century, logicians had developed a precise formal notation to represent knowledge and relationships about the world.



The "Laws of Thought" Approach

- **The AI Connection: The "Logicist Tradition"**
 - By 1965, programs existed that could, **in principle**, solve any solvable problem that could be described in logical notation.
 - The "logician tradition" in AI hopes to build upon such programs to create intelligence.
- **Major Obstacles to this Approach:** The text highlights two significant challenges:
 - **The Knowledge Representation Problem:** It is very difficult to take informal, real-world knowledge and translate it into the strict, formal terms required by logic, especially when that knowledge is less than 100% certain.
 - **The Practicality Problem:** There is a vast difference between solving a problem "in principle" and solving it **in practice**. Even moderately complex problems can overwhelm any computer's computational resources, making them unsolvable in a realistic timeframe.



What Exactly is AI? Four Approaches

4. Acting Rationally: The Rational Agent Approach

- **The Goal:** To design an "agent" (a system that perceives and acts) that acts to achieve the **best possible outcome** in any given situation. It doesn't have to think like a human, just make the optimal decision to achieve its goal. This is the most dominant approach in modern AI.
- **Real-Time Examples: Self-Driving Cars and Recommendation Engines**
 - A **Tesla on Autopilot** doesn't drive "like a human"; it uses sensors to perceive its environment (other cars, road lines) and makes calculated decisions (brake, accelerate, turn) to achieve its goal of getting from A to B safely and efficiently.
 - **Netflix's recommendation algorithm** isn't trying to "think like you." It is a rational agent whose goal is to maximize your viewing time. It analyzes your past behavior and acts by recommending the movie or show with the highest probability of achieving that goal.



What Exactly is AI? Four Approaches

4. Acting Rationally: The Rational Agent Approach

The Rational Agent Approach

- **Definition of an Agent:** An agent is anything that acts. More specifically, a computer agent is expected to operate autonomously, perceive its environment, adapt to change, and pursue goals.
- **The Goal of Rationality:** A **rational agent** is one that acts to achieve the **best possible outcome**. In situations with uncertainty, it acts to achieve the **best expected outcome**.
- **Broader than Just Logic:** This approach includes correct logical inference ("laws of thought") as one way to be rational, but it is not the only way. Rationality also covers situations where:
 - There is no single provably correct action, yet something must be done.
 - A reflex action (e.g., pulling a hand from a hot stove) is more effective and therefore more "rational" than slow, logical deliberation.

What Exactly is AI? Four Approaches

4. Acting Rationally: The Rational Agent Approach

The Rational Agent Approach

- **It is more general:** It is not limited to correct inference and can include other mechanisms (like reflexes) for achieving optimal outcomes.
- **It is more suitable for science:** The standard of rationality is mathematically well-defined and general, making it easier to analyze and engineer. This is in contrast to human behavior, which is complex and adapted to a specific environment.
- **The Focus of Modern AI:** Because of these advantages, this book (and much of modern AI) concentrates on the general principles of designing and building rational agents.
- **An Important Caveat: Perfect vs. Limited Rationality:**
 - Achieving **perfect rationality** (always making the absolute best decision) is not feasible in complex environments due to immense computational demands.
 - However, perfect rationality is used as an ideal starting point for analysis.
 - The concept of **limited rationality**—acting appropriately when there isn't enough time for perfect computation—is a crucial topic for advanced AI.



Foundations of Artificial Intelligence

Philosophy (Pre-20th Century to 1940s)

- **1. Can formal rules be used to draw valid conclusions?**
- **Aristotle:** Was the first to formalize "right thinking" with his system of **syllogisms**, showing that reasoning could be a mechanical process.
- **Ramon Lull & Thomas Hobbes:** Conceived of reasoning as a mechanical artifact and a numerical computation ("we add and subtract in our silent thoughts").
- **Early Calculators:** The work of **da Vinci, Schickard, Pascal**, and especially **Leibniz** (who wanted a machine to operate on concepts, not just numbers) demonstrated that mechanical devices could perform tasks previously associated with human thought.

2. How does the mind arise from a physical brain?

- **René Descartes:** Introduced the idea of **mind-body dualism**, the belief that the mind is a non-physical entity separate from the physical body and not subject to physical laws. This view leaves room for free will.
- **Materialism:** Posed as the alternative to dualism, this view holds that the mind is simply the result of the brain operating according to the laws of physics. Free will is an illusion or the perception of choices. This materialist view is a necessary foundation for the AI endeavor.



Foundations of Artificial Intelligence

Philosophy (Pre-20th Century to 1940s)

3. Where does knowledge come from?

- **Empiricism (Locke, Bacon):** Argued that knowledge comes from experience through the senses ("Nothing is in the understanding, which was not first in the senses").
- **Induction (Hume):** Proposed that we form general rules by being exposed to repeated associations between events. This is a foundational principle of modern machine learning.
- **Logical Positivism (Vienna Circle, Carnap):** This doctrine combined rationalism and empiricism, holding that all knowledge can be described by logical theories ultimately connected to sensory observations.
- **A Computational Mind (Carnap):** Carnap's work proposed an explicit **computational procedure** for extracting knowledge from experience, which the text calls "probably the first theory of mind as a computational process."

4. How does knowledge lead to action?

- **Aristotle:** Argued that action is justified by a logical connection between goals and the knowledge of an action's outcome (e.g., "I need a cloak" leads to the action "I must make a cloak").
- **An Early Algorithm:** Aristotle outlined a method for achieving goals by reasoning backward from the end goal to the means required to achieve it. The text notes this is effectively a **regression planning system**, an algorithm later implemented in Newell and Simon's GPS program.
- **Rational Decision Making:** When goals are uncertain or have multiple paths, thinkers like **Antoine Arnauld** and **John Stuart Mill** (with his theory of *Utilitarianism*) promoted the idea of using quantitative and rational criteria to decide on the best course of action.



Philosophy (Pre-20th Century to 1940s)

3. Where does knowledge come from?

- **Empiricism (Locke, Bacon):** Argued that knowledge comes from experience through the senses ("Nothing is in the understanding, which was not first in the senses").
- **Induction (Hume):** Proposed that we form general rules by being exposed to repeated associations between events. This is a foundational principle of modern machine learning.
- **Logical Positivism (Vienna Circle, Carnap):** This doctrine combined rationalism and empiricism, holding that all knowledge can be described by logical theories ultimately connected to sensory observations.
- **A Computational Mind (Carnap):** Carnap's work proposed an explicit **computational procedure** for extracting knowledge from experience, which the text calls "probably the first theory of mind as a computational process."

4. How does knowledge lead to action?

- **Aristotle:** Argued that action is justified by a logical connection between goals and the knowledge of an action's outcome (e.g., "I need a cloak" leads to the action "I must make a cloak").
- **An Early Algorithm:** Aristotle outlined a method for achieving goals by reasoning backward from the end goal to the means required to achieve it. The text notes this is effectively a **regression planning system**, an algorithm later implemented in Newell and Simon's GPS program.
- **Rational Decision Making:** When goals are uncertain or have multiple paths, thinkers like **Antoine Arnauld** and **John Stuart Mill** (with his theory of *Utilitarianism*) promoted the idea of using quantitative and rational criteria to decide on the best course of action.



Mathematics

- Mathematics provided AI with the formal tools to develop its core ideas in three essential areas: logic, computation, and probability.

1. What are the formal rules to draw valid conclusions? (Logic)

- George Boole:** Developed propositional (Boolean) logic, formalizing logical statements.
- Gottlob Frege:** Extended Boolean logic to create **first-order logic**, which includes objects and relations and is still widely used today.
- Alfred Tarski:** Developed a theory of reference, which shows how to connect logical statements to objects in the real world.



Mathematics

2. What can be computed? (Computation)

- **Algorithms:** The concept of a formal set of steps for computation, with early work dating back to Euclid and al-Khowarazmi.
- **Limits of Proof (Gödel):** The **incompleteness theorem** showed that in any sufficiently powerful formal system (like arithmetic), there are true statements that cannot be proven within that system. This established fundamental limits on what logic and computation can achieve.
- **Computability (Turing):** Alan Turing defined which functions are **computable** by proposing the **Turing machine**. The Church-Turing thesis states that this machine can compute any computable function. Turing also proved that some functions are **not computable** (e.g., the halting problem).
- **Tractability (Practicality):** A problem is considered **intractable** if the time to solve it grows exponentially with its size. This is a crucial practical limit, as many real-world problems are enormous. AI must focus on finding tractable subproblems.
- **NP-Completeness (Cook & Karp):** This theory provides a formal way to identify large classes of problems that are likely to be intractable, helping researchers recognize which problems are computationally very hard to solve.



Mathematics

3. How do we reason with uncertain information? (Probability)

- **Early Foundations (Cardano, Pascal, Fermat):** These thinkers developed the initial ideas of probability, framing it in the context of gambling outcomes.
- **Statistical Methods:** Later mathematicians like Bernoulli and Laplace advanced the theory, creating new statistical methods for dealing with uncertainty.
- **Bayes' Rule (Thomas Bayes):** Proposed a crucial rule for **updating the probability of a belief in light of new evidence**. The text notes this rule is the foundation for most modern AI approaches to uncertain reasoning.



Economics

- Economics contributed to AI by providing formal frameworks for decision-making, especially when dealing with preferences, other agents, and future outcomes.
- 1. How should we make decisions so as to maximize payoff?
- **Core Idea (Adam Smith):** Economics treats individuals as agents who make choices to maximize their own well-being or "preferred outcomes."
- **Formalizing Preference (Utility):** The concept of preferred outcomes was formalized mathematically into the theory of **utility** by thinkers like Walras, Ramsey, von Neumann, and Morgenstern.
- **The Framework for Decision-Making: Decision theory,** which combines utility theory with probability theory, provides a complete formal framework for making rational decisions under uncertainty.



Economics

- Economics contributed to AI by providing formal frameworks for decision-making, especially when dealing with preferences, other agents, and future outcomes.
- 1. How should we make decisions so as to maximize payoff?
- **Core Idea (Adam Smith):** Economics treats individuals as agents who make choices to maximize their own well-being or "preferred outcomes."
- **Formalizing Preference (Utility):** The concept of preferred outcomes was formalized mathematically into the theory of **utility** by thinkers like Walras, Ramsey, von Neumann, and Morgenstern.
- **The Framework for Decision-Making: Decision theory,** which combines utility theory with probability theory, provides a complete formal framework for making rational decisions under uncertainty.



Foundations of Artificial Intelligence

- **2. How should we do this when others may not go along?**
- **The Problem:** In "small" economies or multi-agent situations, the actions of one agent can directly affect the utility of others.
- **The Framework (Game Theory):** Developed by von Neumann and Morgenstern, **game theory** provides the tools to analyze situations where outcomes depend on the choices of multiple rational agents.
- **Key Insight:** Game theory showed that for some games, the most rational strategy is to adopt a policy that appears randomized.



Foundations of Artificial Intelligence

3. How should we do this when the payoff may be far in the future?

- **The Field (Operations Research):** This question was primarily addressed not by economists, but by the field of **operations research (OR)**, which emerged during WWII to solve complex management and logistical problems.
- **The Framework (Markov Decision Processes):** Richard Bellman formalized a class of sequential decision problems where actions have delayed payoffs. These are known as **Markov Decision Processes (MDPs)**, which are now a core topic in AI.
- Key Connections and Divergences with AI
- **Satisficing (Herbert Simon):** For many years, AI and economics developed separately. AI pioneer Herbert Simon (who won a Nobel Prize in Economics) argued that humans use **satisficing**—making decisions that are "good enough" rather than perfectly optimal—because calculating the absolute best outcome is often too complex.
- **Modern Convergence:** Since the 1990s, there has been a strong resurgence of interest in using decision-theoretic ideas from economics within AI agent systems.



Foundations of Artificial Intelligence

3. How should we do this when the payoff may be far in the future?

- **The Field (Operations Research):** This question was primarily addressed not by economists, but by the field of **operations research (OR)**, which emerged during WWII to solve complex management and logistical problems.
- **The Framework (Markov Decision Processes):** Richard Bellman formalized a class of sequential decision problems where actions have delayed payoffs. These are known as **Markov Decision Processes (MDPs)**, which are now a core topic in AI.
- Key Connections and Divergences with AI
- **Satisficing (Herbert Simon):** For many years, AI and economics developed separately. AI pioneer Herbert Simon (who won a Nobel Prize in Economics) argued that humans use **satisficing**—making decisions that are "good enough" rather than perfectly optimal—because calculating the absolute best outcome is often too complex.
- **Modern Convergence:** Since the 1990s, there has been a strong resurgence of interest in using decision-theoretic ideas from economics within AI agent systems.



Neuroscience

- Neuroscience, the study of the brain, provides AI with a blueprint of how a biological system performs intelligent computation.

Early History and Key Discoveries

- **Ancient Recognition:** The brain has long been understood to be the seat of thought, based on evidence from head injuries and Aristotle's observation of its large size in humans.
- **Localization of Function (Paul Broca, 1861):** Proved that specific cognitive functions (like speech) are tied to specific, localized areas of the brain.
- **The Neuron (Golgi & Cajal, late 1800s):** Camillo Golgi's staining technique first allowed for the observation of individual **neurons**, which Santiago Ramon y Cajal then used to map the brain's neural structures.
- **Mathematical Models (Nicolas Rashevsky, 1930s):** Was the first to apply mathematical models to the nervous system, beginning the move toward a computational understanding of the brain.



Foundations of Artificial Intelligence

- Neuroscience

Modern Tools for Measuring Brain Activity

- **EEG (Electroencephalograph):** Invented in 1929 by Hans Berger, this was the first technology to measure activity in an intact brain.
- **fMRI (Functional Magnetic Resonance Imaging):** A modern tool that provides unprecedentedly detailed images of brain activity, allowing scientists to correlate brain function with cognitive processes.
- **Single-cell Recording:** Allows for the mapping of input-output relationships of individual neurons.
- The Central Conclusion and Current Limits
- **Brains Cause Minds:** The text concludes that a collection of simple cells can give rise to thought, action, and consciousness. The only alternative is mysticism, which is rejected.
- **Significant Gaps in Knowledge:** Despite advances, we are still far from a full understanding. Key mysteries include how the brain remaps itself after damage and how individual memories are stored.



- **Neuroscience**

Brain vs. Digital Computer Comparison

- **Speed:** Computers have a much faster **cycle time** (a million times faster than a neuron's).
- **Parallelism and Storage:** The brain compensates with vastly more computational units (**10^{11} neurons**) and interconnections (**10^{14} synapses**), enabling massive parallelism.
- **Raw Capacity:** While a personal computer lags far behind, a modern supercomputer's raw capacity is approaching that of the human brain.
- **The Singularity:** This comparison leads some futurists to predict a **Singularity**, where computers achieve superhuman intelligence. However, the text cautions that these raw comparisons are **not especially informative** on their own.



- Neuroscience

	Supercomputer	Personal Computer	Human Brain
Computational units	10^4 CPUs, 10^{12} transistors	4 CPUs, 10^9 transistors	10^{11} neurons
Storage units	10^{14} bits RAM 10^{15} bits disk	10^{11} bits RAM 10^{13} bits disk	10^{11} neurons 10^{14} synapses
Cycle time	10^{-9} sec	10^{-9} sec	10^{-3} sec
Operations/sec	10^{15}	10^{10}	10^{17}
Memory updates/sec	10^{14}	10^{10}	10^{14}



- **Psychology**

- Psychology, the study of how humans and animals think and act, provided AI with both behavioral observations and, more importantly, the view of the brain as an information-processing device.

- **1. Early Scientific Psychology**

- **Founders:** Hermann von Helmholtz and Wilhelm Wundt established psychology as a science in the late 1800s.

- **Method:** They used carefully controlled experiments, but relied on **introspection** (subjects reporting their own thoughts), which was subjective and difficult to verify.

- **2. Behaviorism**

- **Core Idea:** Led by John Watson, this movement was a reaction against introspection. It rejected any theory involving "mental processes" as unscientific.

- **Methodology:** Behaviorists focused only on objective, measurable data: the **stimulus** given to an animal and its resulting **response** (action).

- **Legacy:** While successful in understanding simple animal learning, behaviorism was inadequate for explaining complex human thought.



Foundations of Artificial Intelligence

- **Psychology**
- **3. Cognitive Psychology**
- **Core Idea:** This movement, which ultimately replaced behaviorism, views the brain as an **information-processing device**. This is the key link to AI.
- **Key Figure (Kenneth Craik):** In 1943, Craik re-legitimized the study of "mental" concepts like beliefs and goals. He specified the three key steps of a knowledge-based agent:
 - The stimulus is translated into an **internal representation**.
 - This representation is **manipulated** by cognitive processes to create new representations.
 - The new representations are translated back into **action**.
- **Craik's "Small-Scale Model":** He argued that an organism with a mental model of reality in its head could test alternatives, plan for the future, and react more competently—a foundational idea for AI agents.



Foundations of Artificial Intelligence

- Psychology

4. The Birth of Cognitive Science

- **Pivotal Event:** A 1956 workshop at MIT where researchers presented influential papers showing how **computer models could be used to explain psychology**:
 - George Miller on memory.
 - Noam Chomsky on language.
 - Newell and Simon on logical thinking.
- **Modern View:** The outcome of this fusion is the now common view that a cognitive theory should be like a **computer program**—a detailed description of an information-processing mechanism.



Computer Engineering

- For AI to exist, it requires an "artifact" to embody its intelligence. Computer engineering provided this artifact.

1. The Core Requirement

- For AI to succeed, it needs two things: **intelligence** and an **artifact**.
- The **computer** has been the artifact of choice for building AI.

2. The Birth of the Modern Electronic Computer

- The first modern computers were invented independently and almost simultaneously during World War II.
- **Key Early Machines:**
 - **Heath Robinson (1940):** The first operational computer, built by Alan Turing's team to decipher German codes.
 - **Colossus (1943):** A powerful, general-purpose successor to the Heath Robinson, based on vacuum tubes.
 - **Z-3 (1941):** The first **programmable** computer, invented by Konrad Zuse in Germany.
 - **ABC (1940-1942):** The first **electronic** computer, built by John Atanasoff and Clifford Berry.
 - **ENIAC:** The most influential early computer, which set the stage for modern computing.



Computer Engineering

3. Pre-Electronic Pioneers

- **Jacquard's Loom (1805):** The first programmable machine, using punched cards to store instructions for weaving patterns.
- **Charles Babbage (mid-19th century):** Designed two key machines:
 - **Difference Engine:** A machine for computing mathematical tables.
 - **Analytical Engine:** A far more ambitious design that was the first artifact capable of **universal computation**. It included addressable memory, stored programs, and conditional jumps.
- **Ada Lovelace:** A colleague of Babbage and the world's first programmer, she wrote programs for the Analytical Engine and speculated it could one day compose music or play chess.



Computer Engineering

4. Hardware Evolution

- For decades, computer performance doubled roughly every 18 months.
- Around 2005, this trend shifted from increasing clock speeds to increasing the **number of CPU cores**.
- Future increases in computing power are expected to come from **massive parallelism**, a design that has a "curious convergence with the properties of the brain."

5. The Software Contribution: A Two-Way Relationship

- **AI's Debt to Computer Science:** AI has relied on operating systems, programming languages, and tools from mainstream computer science.
- **Computer Science's Debt to AI:** AI has repaid this debt by pioneering many ideas that are now standard in computer science, including:
 - Time-sharing and interactive interpreters.
 - Personal computers with windows and mice.
 - Linked lists and automatic storage management (garbage collection).
 - Key concepts in symbolic, functional, and object-oriented programming.



Foundations of Artificial Intelligence

Control Theory and Cybernetics

- This field addresses the question of how artifacts can operate and regulate themselves.

1. The Core Question

- How can artifacts operate under their own control?

2. Early Historical Examples

- **The First Self-Controlling Machine:** The water clock (c. 250 B.C.) by Ktesibios of Alexandria, which used a regulator to maintain a constant flow. This was the first example of an artifact that could modify its behavior in response to the environment.
- **Other Self-Regulating Systems:** James Watt's steam engine governor and Cornelis Drebbel's thermostat are other classic examples of **feedback control systems**.



Foundations of Artificial Intelligence

Control Theory and Cybernetics

3. The Modern Founders and Core Ideas

- **Norbert Wiener:** The central figure who founded the field of **control theory**.
- **The Core Idea (Wiener, Rosenblueth, Bigelow):** They viewed purposeful behavior as a regulatory mechanism trying to minimize **error**—the difference between the current state and the goal state. This challenged the purely reactive view of behaviorism.
- **Cybernetics:** A broader field and movement, popularized by Wiener's bestselling book, that explored these new mathematical and computational models of cognition and control.
- **Parallel British Efforts (W. Ross Ashby):** Ashby and others in the "Ratio Club" developed similar ideas. His book *Design for a Brain* argued that intelligence could be created using **homeostatic** devices that use feedback loops to achieve stable, adaptive behavior.



Control Theory and Cybernetics

4. The Distinction Between Control Theory and AI

- **Shared Goal:** Modern control theory, like AI, aims to design systems that behave optimally by maximizing an **objective function** over time.
- **Key Difference (The Reason They Are Separate Fields):** The difference lies in the mathematical tools used and the types of problems addressed.
 - **Control Theory:** Uses **calculus and matrix algebra**. This is best suited for systems described by a fixed set of **continuous variables** (e.g., temperature, velocity).
 - **AI:** Was founded to escape these limitations. It uses tools like **logical inference and computation** to address complex, symbolic problems that are outside the scope of traditional control theory, such as **language, vision, and planning**.



• Linguistics

- Linguistics provided AI with the theories and formalisms necessary to understand the relationship between language and thought, leading to the field of Natural Language Processing.

1. The Core Question

- How does language relate to thought?

2. The Turning Point: Behaviorism vs. Chomsky

- **The Behaviorist View (B.F. Skinner):** Skinner's 1957 book *Verbal Behavior* presented a comprehensive theory of language based on behaviorist principles (i.e., learning through stimulus and response).
- **Chomsky's Critique:** Linguist **Noam Chomsky** wrote a highly influential review that devastated the behaviorist approach to language.
- **The Central Argument:** Chomsky pointed out that behaviorism could not explain the **creativity** in language—the ability for a person to understand and produce entirely new sentences they have never heard before.



Linguistics

3. The New Approach: Formal, Computable Language

- **Chomsky's Theory:** His own theory, presented in *Syntactic Structures*, used formal models of syntax that could explain this creativity.
- **The AI Connection:** Crucially, Chomsky's theory was formal enough that it could, in principle, be **programmed** on a computer.

4. The Birth of a Hybrid Field

- Modern linguistics and AI were "born" around the same time and grew up together.
- Their intersection created the hybrid field now known as **Computational Linguistics** or **Natural Language Processing (NLP)**.

5. A Deeper Realization

- It was soon discovered that understanding language is far more complex than just analyzing sentence structure.
- True understanding requires knowledge of the **subject matter and context**.
- This realization created a strong link between language processing and **knowledge representation** (the study of how to put knowledge into a form that a computer can reason with).



History of Artificial Intelligence

The Gestation of AI (1943–1955)

- **The First AI Work (McCulloch & Pitts, 1943):** Proposed a model of artificial neurons, showing that a network of them could compute any computable function and implement logical operators.
- **Learning Rule (Donald Hebb, 1949):** Described a simple rule for updating the connection strengths between neurons, now known as **Hebbian learning**.
- **First Neural Network Computer (1950):** The SNARC, built by Marvin Minsky and Dean Edmonds, simulated a network of 40 neurons using 3000 vacuum tubes.
- **Turing's Vision (1950):** Alan Turing's influential paper "Computing Machinery and Intelligence" introduced the Turing Test, machine learning, genetic algorithms, and the idea of creating a "child" program to learn, rather than programming an "adult" mind.



The Birth of AI (1956)

- **The Dartmouth Workshop:** A two-month summer workshop organized by John McCarthy, which brought together the founding figures of the field.
- **Coining the Name:** The workshop proposal coined the term "**artificial intelligence**" and articulated its central conjecture: "that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it."
- **The First Reasoning Program:** Allen Newell and Herbert Simon "stole the show" with their **Logic Theorist**, a program that could prove mathematical theorems, solving the "venerable mind–body problem."
- **A New Field is Born:** The workshop established AI as a separate field, distinct from others because it embraced duplicating human faculties (like creativity and language) and focused on building autonomous machines for complex environments.



History of Artificial Intelligence

Early Enthusiasm, Great Expectations (1952–1969)

- **The "Look, Ma, no hands!" Era:** A period of astonishing successes on limited problems, leading to great optimism.
- **General Problem Solver (GPS):** Newell and Simon's follow-up program, which was the first to embody the "thinking humanly" approach by imitating human problem-solving protocols. This led to their **physical symbol system hypothesis** (intelligence arises from manipulating symbols).
- **Key Contributions (John McCarthy, 1958):** In one year at MIT, he invented the **Lisp** programming language, pioneered **time-sharing**, and described the **Advice Taker**—a blueprint for the first complete AI system based on knowledge representation and reasoning.
- **Microworlds (Minsky's students):** AI research focused on solving narrowly defined problems that appeared to require intelligence, such as calculus, IQ tests, and the famous **blocks world**.
- **Neural Networks Flourish:** Early work on **perceptrons** (Rosenblatt) showed success, and the perceptron convergence theorem proved they could learn anything they could represent.



History of Artificial Intelligence

A Dose of Reality (1966–1973)

- **The First "AI Winter":** The initial optimism crashed against three major obstacles when systems were applied to broader, more difficult problems.
 - **Failure of Syntactic Manipulation:** Early programs (like machine translation) knew nothing of their subject matter and failed because true understanding requires background knowledge.
 - **Intractability:** Most programs used simple search, which failed to scale due to the "**combinatorial explosion**" of possibilities in real-world problems.
 - **Fundamental Limitations:** Minsky and Papert's book *Perceptrons* proved that simple neural networks had severe limitations, which led to the cancellation of nearly all research funding in the area.
- **Funding Cuts:** Government funding was cut in both the US (for machine translation) and the UK (based on the Lighthill report).



History of Artificial Intelligence

Knowledge-Based Systems: The Key to Power? (1969–1979)

- **The New Paradigm:** A shift away from general-purpose "weak methods" to using powerful, **domain-specific knowledge**.
- **DENDRAL (1969):** The first successful **knowledge-intensive system**. It used a large number of "cookbook" rules from expert chemists to infer molecular structure, demonstrating the power of domain knowledge.
- **Expert Systems:** The methodology of building systems with expert knowledge was applied to other fields. **MYCIN**, a system for medical diagnosis, used ~450 rules and introduced **certainty factors** to reason with uncertainty.
- **Knowledge Representation:** Understanding natural language was also shown to depend on general world knowledge, leading to the development of representation schemes like **frames**.



History of Artificial Intelligence

AI Becomes an Industry (1980–present)

- **Commercial Success:** **Expert systems** boomed. R1, a program that configured computer systems for DEC, was saving the company \$40 million a year by 1986.
- **Global Investment:** Japan's "Fifth Generation" project (1981) spurred massive investment in AI from the US and Britain.
- **The "AI Winter" (Part 2):** The boom was followed by a bust in the late 80s as hundreds of companies failed to deliver on extravagant promises.

The Return of Neural Networks (1986–present)

- **The Catalyst:** The reinvention of the **back-propagation** learning algorithm.
- **The Resurgence:** This caused great excitement and a massive resurgence in research into **connectionist** models (neural networks), which were seen as competitors to symbolic AI. The modern view is that they are complementary.



History of Artificial Intelligence

AI Adopts the Scientific Method (1987–present)

- **Maturation of the Field:** A revolution in methodology occurred.
 - Claims began to be based on **rigorous theorems** and **hard experimental evidence**.
 - AI began to re-embrace and build upon related fields like statistics and control theory.
 - Shared repositories of test data allowed for the replication of experiments.
- **Example (Speech Recognition):** The field shifted from ad-hoc methods to rigorous mathematical models like **Hidden Markov Models (HMMs)**, which are trained on large data sets and steadily improve.
- **Rise of Probability:** **Bayesian networks** and decision theory became the dominant approach for handling uncertainty, leading to **normative expert systems** that act rationally.



History of Artificial Intelligence

The Emergence of Intelligent Agents (1995–present)

- **Putting it all Together:** Researchers began to tackle the "whole agent" problem again, integrating subfields like vision, robotics, and planning.
- **The Internet as an Environment:** AI became common in web applications, leading to the popular use of the "-bot" suffix for search engines, recommender systems, etc.
- **Discontent and A New Push:** Some AI founders expressed discontent with the field's focus on narrow applications, calling for a return to its original goals. This led to movements for **Human-Level AI (HLAI)** and **Artificial General Intelligence (AGI)**, along with concerns about creating **Friendly AI**.

The Availability of Very Large Data Sets (2001–present)

- **The New Paradigm:** For many problems, the **amount of data is more important than the choice of algorithm**.
- **Solving the Knowledge Bottleneck:** The problem of how to give a system knowledge can often be solved by learning from massive data sets (trillions of words, billions of images) instead of hand-coding rules.
- **The Result:** A mediocre algorithm with vast data can outperform the best algorithm with little data. This has led to a surge of successful applications and a new "AI Spring," with AI now "deeply embedded in the infrastructure of every industry."



State of the Art

- **Robotic Vehicles:** Drive autonomously in challenging environments, as demonstrated by vehicles winning the DARPA Grand Challenge in the desert and navigating simulated urban traffic.
- **Speech Recognition:** Manage entire conversations to perform tasks, such as the automated flight booking and dialog system used by United Airlines.
- **Autonomous Planning and Scheduling:** Autonomously plan, schedule, and control the complex operations of spacecraft and Mars rovers millions of miles from Earth, as shown by NASA's Remote Agent program.
- **Game Playing:** Defeat the best human players in complex strategic games, famously demonstrated when IBM's Deep Blue beat world chess champion Garry Kasparov.
- **Spam Fighting:** Use learning algorithms to classify billions of messages as spam daily, adapting to new tactics more effectively than static rules.
- **Logistics Planning:** Handle large-scale logistics, such as the DART system used by the U.S. military to coordinate transportation for tens of thousands of units, creating plans in hours instead of weeks.
- **Robotics:** Perform physical tasks in the real world, from consumer products like the Roomba vacuum cleaner to rugged robots like the PackBot used for bomb disposal.
- **Machine Translation:** Automatically translate languages by building statistical models from vast amounts of text data, even without the programmers having linguistic expertise in those languages.



Intelligent Agents: AGENTS AND ENVIRONMENTS

- **1. What is an Agent?**

- **Definition:** An **agent** is anything that **perceives** its **environment** using **sensors** and **acts** upon that environment using **actuators**.

Examples:

- **Human Agent:**

- **Sensors:** Eyes, ears, skin (to perceive)
- **Actuators:** Hands, legs, mouth (to act)

- **Robot:**

- **Sensors:** Cameras, proximity sensors
- **Actuators:** Motors, robotic arms

- **Software Agent (e.g., Chatbot):**

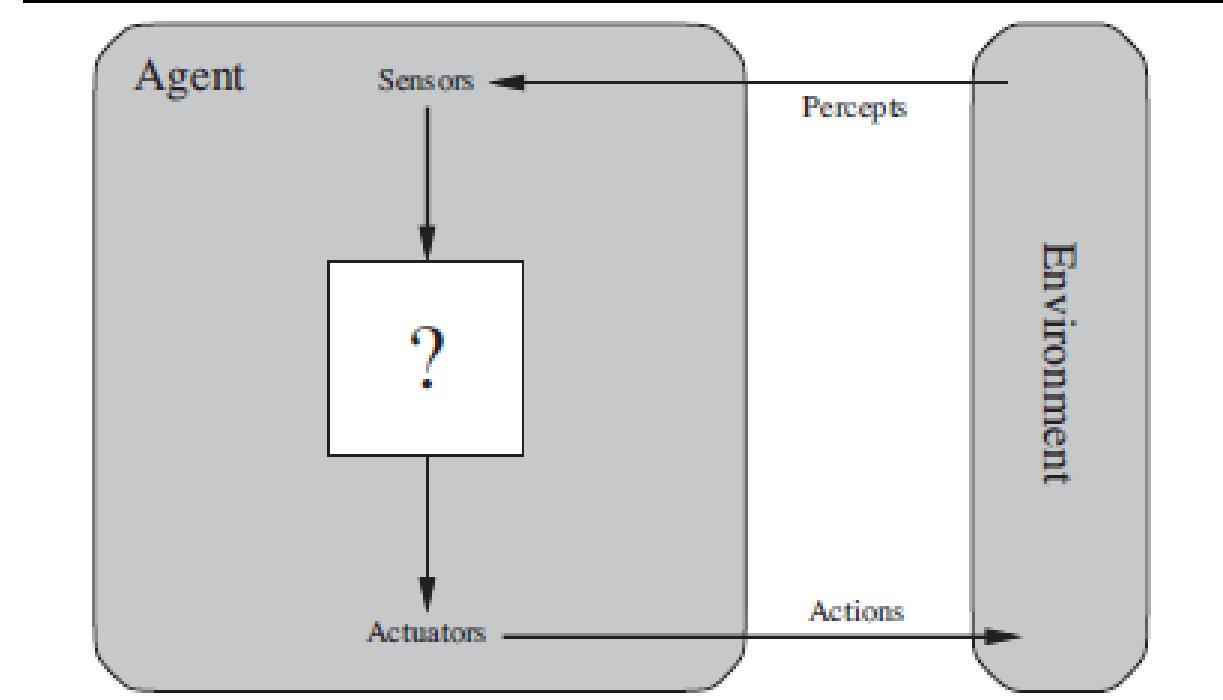
- **Sensors:** Keystrokes, file input, web data
- **Actuators:** Screen output, sending messages or data



Intelligent Agents: AGENTS AND ENVIRONMENTS

2. How Does an Agent "Think"?

- An agent's behavior is determined by its history of perceptions.
- A **percept** is what the agent perceives at a single moment.
- A **percept sequence** is the complete history of everything the agent has ever perceived.
- The agent's decision-making process is described by its **Agent Function**, which is an abstract map that says what action to take for any given history of perceptions.





Intelligent Agents: AGENTS AND ENVIRONMENTS

3. The Function vs. The Program (A Crucial Distinction)

- It's important to separate the abstract idea from the concrete code.
- **Agent Function:** The abstract behavioral description (e.g., a giant theoretical table mapping every possible percept history to an action). It's the **what** the agent does.
- **Agent Program:** The actual code that runs on the physical system and implements the agent function. It's the **how** it does it.

•



Intelligent Agents: AGENTS AND ENVIRONMENTS

3. The Function vs. The Program (A Crucial Distinction)

- It's important to separate the abstract idea from the concrete code.
- **Agent Function:** The abstract behavioral description (e.g., a giant theoretical table mapping every possible percept history to an action). It's the **what** the agent does.
- **Agent Program:** The actual code that runs on the physical system and implements the agent function. It's the **how** it does it.

•



Intelligent Agents: AGENTS AND ENVIRONMENTS

3. The Function vs. The Program (A Crucial Distinction)

Example: A Smart Speaker (like Amazon Alexa or Google Assistant)

- **Agent:** The smart speaker itself.
- **Environment:** The room it's in, including the user, ambient noise, and its connection to the internet.
- **Sensors:** Microphones (to hear commands), maybe a camera on some models.
- **Actuators:** The speaker (to talk back), lights (to show status), and its ability to control other smart devices (lights, thermostats).
- **Percept:** Hearing the words, "Alexa, what's the weather?"
- **Percept Sequence:** The full history might be: [Silence], [User enters room], ["Alexa, what's the weather?"].
- **Agent Function:** The internal logic that maps the sound of that specific question to the action of looking up the forecast and speaking it aloud. The agent's program is the actual software running on its processors that makes this happen.



Intelligent Agents: AGENTS AND ENVIRONMENTS

3. The Function vs. The Program (A Crucial Distinction)

Example: A Smart Speaker (like Amazon Alexa or Google Assistant)

Percept Sequence → Action Table for a Smart Speaker



Agent Percept Sequence	Action
[Silence]	Do nothing
[Silence], [User enters room]	Light up to show it's listening or on standby
[Silence], [User enters room], ["Alexa, what's the weather?"]	Fetch weather from internet and speak it out
[Silence], [User enters room], ["Alexa, turn on the lights"]	Send signal to smart lights and say "Okay"
[User enters room], ["Alexa, play music"]	Connect to music service and start playback
[Background noise], ["Alexa, set a timer for 5 minutes"]	Start timer and confirm with voice feedback
[User says nothing], [Device hears unusual loud noise (e.g., crash)]	Optionally alert the user or flash lights (in advanced implementations)



Intelligent Agents: AGENTS AND ENVIRONMENTS

3. The Function vs. The Program (A Crucial Distinction)

Example: A Smart Speaker (like Amazon Alexa or Google Assistant)

Conceptual Mapping:

Agent Function:

A mathematical/logical model that maps every possible percept sequence to a corresponding action.

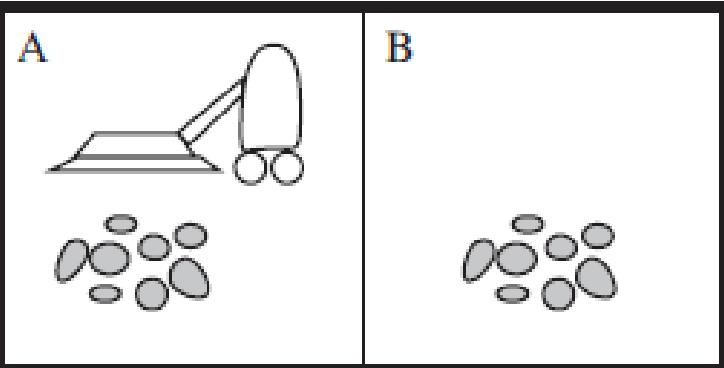
Agent Program:

The real software code that receives voice input, processes natural language, connects to services, and performs actions like speaking or controlling devices.



When you say "**Alexa, play music**", the **agent function** maps this command to an action like streaming music. The **agent program**—coded in Python, C++, or another language—actually interprets your speech, connects to Spotify, and plays the song.

Intelligent Agents: AGENTS AND ENVIRONMENTS



A vacuum-cleaner world with just two locations.

Environment: Two locations — A and B.

Percepts: Agent senses:
• Its **current location** (A or B).

Dirt status (Dirty or Clean).

Actions: The agent can:

• Suck (clean the current square),

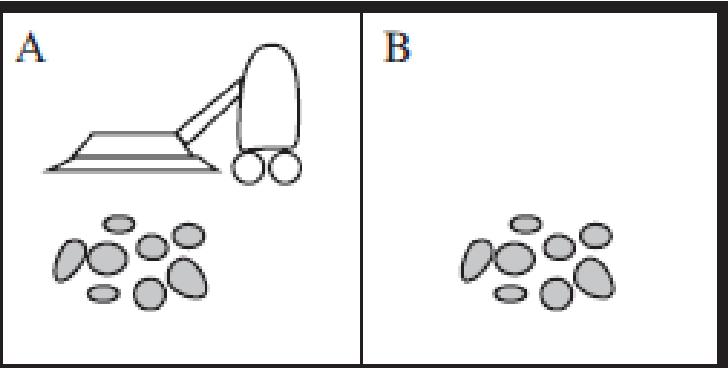
• Left (move to the left square),

• Right (move to the right square),

• NoOp (do nothing).



Intelligent Agents: AGENTS AND ENVIRONMENTS



A vacuum-cleaner world with just two locations.

- **Environment:** Two locations — A and B.

- **Percepts:** Agent senses:
• Its **current location** (A or B).

- **Dirt status** (Dirty or Clean).

- **Actions:** The agent can:

- Suck (clean the current square),

- Left (move to the left square),

- Right (move to the right square),

- NoOp (do nothing).

Simple Agent Rule:

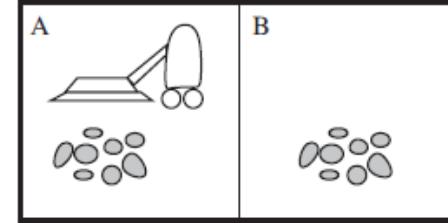
If the current square is dirty, then **Suck**; otherwise, move to the other square.



Intelligent Agents: AGENTS AND ENVIRONMENTS

Percept Sequence → Action Table

Percept Sequence	Action
[(A, Dirty)]	Suck
[(A, Clean)]	Right
[(B, Dirty)]	Suck
[(B, Clean)]	Left
[(A, Clean), (B, Dirty)]	Suck
[(A, Clean), (B, Clean)]	Left
[(A, Dirty), (A, Clean)]	Right
[(B, Dirty), (B, Clean)]	Left
[(A, Clean), (B, Clean), (A, Clean)]	Right
[(A, Clean), (B, Clean), (A, Clean), (B, Clean)]	NoOp



A vacuum-cleaner world with just two locations.

- The agent acts solely based on its **current percept** (not remembering the past).
- It **sucks if dirty**, otherwise **moves to the other location**.
- If both locations have been cleaned (as inferred from the percept sequence), the agent can do NoOp.

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY



What Is a Rational Agent?

- An agent that does the "right thing."
- The "right thing" is the action that maximizes its performance.
- Focuses on achieving the best **outcome** based on consequences.

How Do We Measure Performance?

- With a **Performance Measure**: An objective standard that evaluates a sequence of environment states.
- **Crucial Point**: Success is measured by the state of the **environment**, not the agent's internal opinion.
 - This prevents an agent from simply "deluding itself" that it's doing well.

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY



The Designer's Trap: Be Careful What You Ask For

A rational agent will do exactly what it's told to do to maximize its score.

A poorly designed performance measure will be exploited, leading to unintended and undesirable behavior.

Slogan: "**What you ask for is what you get.**"

Example: Vacuum Cleaner Agent

Let's consider two performance measures for the vacuum cleaner agent:

Performance Measure 1 (Bad Design):

- **"Amount of dirt cleaned over 8 hours"**
→ This could **incentivize cheating**:

- Agent cleans dirt
- Agent dumps it again
- Agent re-cleans it

Infinite loop to boost performance artificially

Performance Measure 2 (Better Design):

- **"1 point per clean square at every time step"**

- Encourages **sustained cleanliness**, not just action
- Penalizes inefficiency (like excessive movement, electricity, noise)

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

How Do We Measure Performance?

- With a **Performance Measure**: An objective standard that evaluates a sequence of environment states.
- **Crucial Point**: Success is measured by the state of the **environment**, not the agent's internal opinion.
 - This prevents an agent from simply "deluding itself" that it's doing well.

The Golden Rule of Performance Measures

- Design the measure based on the **OUTCOME** you want in the environment...
- ...not the **PROCESS** you think the agent should follow.

Good Performance Measures Focus on Environment States, Not Agent Behavior

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Reflex Agent vs. Rational Agent





GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Reflex Agent vs. Rational Agent

Feature	Reflex Agent	Rational Agent
Decision Basis	Current percept only	Percept history , knowledge , and expected outcomes
Memory	None (stateless)	May use memory , models, and internal state
Goal Awareness	Not goal-oriented	Always goal-oriented (maximizes performance measure)
Adaptability	Not adaptable to new situations	Adapts based on new knowledge and experience
Examples	Basic vacuum agent that sucks if it sees dirt	Smart vacuum that plans cleaning based on room map + battery
Agent Type	Simple reflex or reflex with condition-action rules	Goal-based or utility-based agent

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Rationality:

A **rational agent** is one that:

For **every percept sequence**, selects an action that is **expected to maximize** its **performance measure**, given:

- 1.What it has **perceived so far** (percept history),
- 2.What it **knows about the environment** (built-in knowledge),
- 3.What it **can do** (available actions),
- 4.And how **success is defined** (performance measure).

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Components That Affect Rationality

Component	Meaning
Performance Measure	What defines success (e.g., how many squares are clean)
Prior Knowledge	What the agent knows in advance (e.g., layout, rules)
Available Actions	What it can do (e.g., Left, Right, Suck)
Percept Sequence	What it has sensed so far (e.g., "I'm in A, and it's clean")

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Vacuum Cleaner Agent — Is It Rational?



simple reflex agent:

If square is dirty → Suck

Else → move to the other square (Left or Right)

Under These Conditions:

- ✓ **Performance measure:** +1 per clean square per time step, for 1000 steps
- ✓ **Geography** is known (just A and B)
- ✗ **Dirt status and start location** are unknown
- ✓ **Sensors:** Know current location and whether it is dirty
- ✓ **Actions:** Suck, Left, Right

Verdict: This agent *is rational* in this context

Because:

- It **cleans when needed**
- It **moves to find dirt if not present**
- It has no memory, but **its policy is optimal** for this static, fully observable world



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Vacuum Cleaner Agent — Is It Rational?

✗ When Is It NOT Rational?

Let's tweak the environment:

⚠ Case 1: Penalty for movement

Movement (Left/Right) costs -1 point.

- The reflex agent **keeps oscillating** even after everything is clean.
- Better: An agent that **stops moving** once it **knows** everything is clean.
- Reflex agent = ✗ irrational here.

Case 2: Environment gets dirty again

Dirt **reappears** over time.

- Reflex agent won't know this unless it **revisits each square**.
- Better: An agent that **periodically checks** or follows a patrol pattern.
- Reflex agent = ✗ irrational here.

Case 3: Unknown environment (more than A & B)

Geography is not known initially.

- Reflex agent only acts within A and B.
- A **rational agent should explore**.
- Reflex agent = ✗ irrational again.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Vacuum Cleaner Agent — Is It Rational?

Challenge (Design Task)

Design agents for the three cases above:

1. With movement penalty:

- Agent maintains an internal state:
 - Marks square as clean once cleaned.
 - Stops moving once both squares are known to be clean.
- Only acts if dirt is present.
- **Saves points by staying still.**

2. If dirt reappears:

- Agent implements **periodic checking**:
 - E.g., Alternate between A and B every 10 time steps.
 - Keeps track of last dirt time per square.
- Keeps performance up even if dirt returns.

3. Unknown environment:

- Agent starts exploring:
 - Moves in a direction until hitting a wall.
 - Builds internal map.
 - Cleans as it goes.
- Rationality here includes **exploration + cleaning**.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy

1. Rationality ≠ Omniscience

Rationality: Do the best you can *with what you currently know.*

Omniscience: Know everything, including the future (which is impossible).

Example: Crossing the street

You look both ways — no cars. You start crossing.

 A random airplane part falls on you.

Were you **irrational**? No!

You made the **best decision based on available information.**

 **Moral:** Rational ≠ Perfect. Expecting perfection = demanding a *crystal ball.*

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy

2. Information Gathering is Rational

Rational agents should **take actions to improve future decisions.**

Example: Crossing a busy road

If the agent doesn't **look** for cars first, it's acting blindly. Even if no car is visible, **not looking is irrational** because of the **risk**.

So: "**Looking before crossing**" is part of rational behavior.

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy

3. Exploration Example (Vacuum Agent)

In a **partially known house**, the vacuum agent must:

- Explore new rooms
- Find dirt
- Build a map of its world

This is **exploration as information gathering** — an essential part of being rational.





GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy

4. Learning

Even if an agent starts with some built-in knowledge, it should **improve over time.**

Funny Bug Example: Dung Beetle

- Lays eggs
- Fetches dung ball
- If ball is taken away → Still "plugs" the hole
- Doesn't learn that the ball is **gone**

Another: Sphex Wasp

- Drags a caterpillar to its burrow
- If you move it slightly... she **starts over** each time!
- She never learns the plan is failing.

These insects **lack learning**. They rely only on built-in behaviors. **Not autonomous.**



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy

4. Learning

Even if an agent starts with some built-in knowledge, it should **improve over time.**

Funny Bug Example: Dung Beetle

- Lays eggs
- Fetches dung ball
- If ball is taken away → Still "plugs" the hole
- Doesn't learn that the ball is **gone**

Another: Sphex Wasp

- Drags a caterpillar to its burrow
- If you move it slightly... she **starts over** each time!
- She never learns the plan is failing.

These insects **lack learning**. They rely only on built-in behaviors. **Not autonomous.**

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy



5. Autonomy

The more an agent relies on its **own percepts** rather than **programmed instructions**, the more **autonomous** it is.

Vacuum Cleaner Example:

- Reflex agent: Cleans based on fixed rules
- Autonomous agent: Learns patterns of dirt reappearance (e.g., kids spill snacks at 4 PM)

Over time, a **learning agent** becomes **less dependent on its designer** and more **self-sufficient**.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy

Robot vs. Pet

Concept	Reflex Agent (Non-autonomous)	Rational, Learning Agent (Autonomous)
Behavior Basis	Hard-coded by designer	Learns from environment + percepts
Flexibility	Fails in new situations	Adapts, improves over time
Omniscience	Assumes nothing about future	Plans using expected outcomes
Autonomy	Low	High (grows independent)



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

When designing an intelligent agent, the first and most essential step is to **define the task environment** using the **PEAS** model:

- **P**: Performance Measure — Criteria for success
- **E**: Environment — The surroundings in which the agent operates
- **A**: Actuators — The tools or components used by the agent to act
- **S**: Sensors — Devices used by the agent to perceive the environment

This model helps clearly specify what the agent is supposed to do, in what setting, and how it interacts with its surroundings.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

Example 1: Simple Vacuum Cleaner Agent

PEAS Element	Description
Performance	Floor cleaned efficiently, minimal electricity use
Environment	Rooms with floors, walls, and dirt patches
Actuators	Motors to move and vacuum dust
Sensors	Dirt detector, wall bumper sensor

Simplified World: Only two locations (A and B), clean or dirty.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

Example 2: Automated Taxi Driver Agent

A more complex, real-world agent, with a highly dynamic and unpredictable environment.

PEAS Element	Description
Performance	Safe, quick, legal, comfortable rides; low fuel use; high profit
Environment	Urban and rural roads, traffic, pedestrians, weather, passengers
Actuators	Steering wheel, accelerator, brakes, gear, horn, indicators, display/speaker
Sensors	Cameras, GPS, speedometer, sonar, accelerometer, microphone, engine sensors

Key Considerations:

- Must navigate different geographies (e.g., snowy Alaska vs. sunny California)
- Might drive on the right (India/US) or left (UK/Japan)
- Needs to **balance** multiple goals: safety vs. speed, comfort vs. profit



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

Real vs. Artificial Environments

- Some agents operate in **physical environments** (like robots).
- Others exist in **virtual environments** (like software agents).
- What matters most is the **complexity** of the agent's interaction with the environment, not whether it's real or digital.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

Example 1: Conveyor Belt Inspection Robot

PEAS Element	Description
Performance	Accurate defect detection, high throughput
Environment	Controlled factory belt with predictable object flow
Actuators	Robot arms to accept/reject parts
Sensors	Camera, light sensors

Simplified Conditions: Same lighting, object types, and position → Easier to design agent.

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

Example 4: Web-based Software Agent (Softbot)

A **softbot** is a software agent that operates in a virtual, highly complex environment (e.g., Internet).

PEAS Element	Description
Performance	Provide relevant news, maintain uptime, generate ad revenue
Environment	Internet: dynamic sources, users, advertisers
Actuators	Display content, adjust layout, serve ads
Sensors	User input (clicks, preferences), server status, web scrapers

- Example: If a news website goes down, it must find alternate sources.
- Must learn and adapt based on user interest and advertiser behavior.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

- **PEAS** helps in clearly defining agent design tasks.
- Agents range from simple to highly complex:
 - Simple: Vacuum cleaner, part inspection robots
 - Complex: Autonomous cars, web-based softbots
- Designing intelligent agents depends heavily on **how well the task environment is understood and specified.**



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Properties of Task Environments (Dimensions to Classify Them)

Each task environment can be described along **several key dimensions**.

These characteristics influence the **design of the agent** and the **techniques used** to build it.

1. Fully Observable vs. Partially Observable

- **Fully Observable:** The agent's sensors capture the **entire relevant state** of the environment at all times.
 - Example: Chess – the entire board is visible.

- **Partially Observable:** Agent has **limited** or **noisy perception** of the environment.

- Example: Taxi driving – can't see around corners or know other drivers' intentions.

Example: Poker – can't see opponents' cards.

2. Single Agent vs. Multiagent

- **Single Agent:** Only **one intelligent agent** acts in the environment.

-

Example: Crossword puzzle solver.

- **Multiagent:** **Multiple intelligent agents** interact.

-

Cooperative: All agents aim to achieve a shared goal (e.g., self-driving cars avoiding collisions).

-

Competitive: Opposing goals (e.g., chess or poker).

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Properties of Task Environments .

3. Deterministic vs. Stochastic

•**Deterministic:** Next state is **completely determined** by the current state and action.
Example: Chess (ignoring opponent's unpredictability).

•**Stochastic:** Involves **randomness or uncertainty** in outcomes.
Example: Taxi driving – unpredictable traffic or weather.

•**Uncertain:** Either **partially observable** or **stochastic**.

•**Nondeterministic:** Actions have **possible outcomes** but without probabilities.

4. Episodic vs. Sequential

•**Episodic:** Each action is **independent** of previous ones.
Example: Image classification – each image is treated separately.

•**Sequential:** Current actions **affect future states**.

Example: Chess or medical diagnosis – early decisions have long-term impact.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Properties of Task Environments .

5. Static vs. Dynamic

• **Static:** Environment **does not change** while the agent is thinking.

 Example: Crossword puzzle.

• **Dynamic:** Environment **changes over time**, even while the agent is deciding.

Example: Driving – traffic and surroundings constantly shift.

• **Semidynamic:** Environment remains unchanged, but **performance score changes** over time.

Example: Chess with a clock.

6. Discrete vs. Continuous

• **Discrete:** A **finite number** of distinct percepts, actions, or states.

Example: Chess – fixed number of board states and moves.

• **Continuous:** States and actions can take on **any value within a range**.

Example: Driving – position, speed, steering angles vary continuously.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Properties of Task Environments .

7. Known vs. Unknown

• **Known:** Agent (or designer) **knows the rules** of how the environment works.

Example: Chess – rules are well-defined.

• **Unknown:** Agent must **learn** how the environment behaves.

Example: New video game – agent explores how the controls work.

- ◆ Known vs. Unknown refers to **knowledge of environment mechanics**, not observability.

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Properties of Task Environments .

Examples of Task Environments

Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess (with clock)	Fully	Multi	Deterministic	Sequential	Semidynamic	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semidynamic	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

THE STRUCTURE OF AGENTS



Agent = Architecture + Program

- Agent Function:** Maps percepts → actions
- Agent Program:** Implements this function
- Architecture:** Executes the program using physical hardware
- Designing an intelligent agent means choosing the **right program** for the **right architecture** to interact effectively with the environment.



THE STRUCTURE OF AGENTS

Agent = Architecture + Program

Agent Program

- The **agent program** is the core software that implements the **agent function**.
- It defines **how the agent maps percepts to actions**.
- It's the "brain" of the agent.
- Examples:
 - A vacuum cleaner program that tells the robot to turn left if it detects a wall.
 - A chess-playing program that chooses the best move after analyzing the board.

Agent Architecture

- The **architecture** is the **computing platform** that the agent program runs on.
- It includes:
 - **Sensors** (to perceive the environment)
 - **Actuators** (to perform actions)
 - **Computational hardware** (e.g., CPU, memory)
- The architecture might be:
 - A **simple PC** for a software agent
 - A **robot** with legs, wheels, cameras, microphones, etc.

THE STRUCTURE OF AGENTS



Agent = Architecture + Program

Architecture + Program

- The architecture makes **sensor data** available to the program.
- The agent program **processes percepts**, makes decisions, and outputs **actions**.
- These actions are executed by the **actuators**.

Loop:

1. Sensor detects input from environment
2. Architecture passes percepts to agent program
3. Agent program decides what to do
4. Architecture sends the action to actuators
5. Environment changes → cycle repeats

THE STRUCTURE OF AGENTS

Agent = Architecture + Program

Agent Programs

An **agent program** is the **software** part of an agent that:

Takes current percepts (sensory inputs) from the environment

Returns an action to be performed by the agent

💡 Think of it like this:

If an agent is a robot chef, the **agent program** is its cooking logic (e.g., "if the onion is chopped, then sauté it").

Agent Program vs. Agent Function

•**Agent Function:** Maps the **entire history of percepts** → action

•**Agent Program:** Uses **only the current percept** (because that's all it gets at the moment)

If the agent needs to remember past events, it must build memory **inside the program**.

THE STRUCTURE OF AGENTS

Agent = Architecture + Program

Example: Table-Driven Agent



```
function TABLE-DRIVEN-AGENT(percept) returns an action
  persistent: percepts, a sequence, initially empty
              table, a table of actions, indexed by percept sequences, initially fully specified
  append percept to the end of percepts
  action  $\leftarrow$  LOOKUP(percepts, table)
  return action
```

THE STRUCTURE OF AGENTS

Agent = Architecture + Program

Example: Table-Driven Agent



Why Table-Driven Agents Are Impractical

Let's say:

- You receive 1 percept per second
- Your agent runs for 1 hour → 3600 percepts
- Even with 10 types of percepts → Table needs 10^{3600} entries

That's **more entries than atoms in the universe!**

Problems:

- Can't store such a huge table (too much memory)
- Can't fill it manually or learn all combinations
- No guidance on what the entries should be

But Table-Driven Agent Does Work

The approach technically **implements the correct agent function** — it gives the right action — but it's not **efficient or intelligent**.

So the goal of AI is:

To design **small, clever programs** that act rationally — **without** needing huge tables.

Like replacing **books of square roots** with a tiny calculator algorithm (like Newton's Method)!



THE STRUCTURE OF AGENTS

Types of Agent Programs

AI uses **4 basic types** of agents to generate intelligent behavior more efficiently.

① Simple Reflex Agents

- React only to **current percept**
- No memory of past
- Use **if-then rules**

Example: Reflex Vacuum Agent

IF status = Dirty THEN Suck
ELSE IF location = A THEN go Right
ELSE IF location = B THEN go Left

③ Goal-Based Agents

- Decide actions based on **desired outcomes** (goals)
- Can **plan** and evaluate different possibilities

Example: Google Maps

It knows your goal is to reach “Coimbatore Railway Station.”
It explores multiple routes and picks the fastest one using its knowledge of traffic and roads.

② Model-Based Reflex Agents

- Keep **track of the world** internally (have a model)
- Update the model based on percepts
- Use rules and memory

Example: A Smart Thermostat

If it knows the current temperature and remembers what time it last changed settings, it can avoid turning the AC on/off too frequently.

④ Utility-Based Agents

- Choose actions that **maximize utility** (i.e., satisfaction or benefit)
- Can handle **trade-offs** (e.g., cost vs. comfort)

Example: Self-Driving Taxi

It may choose a slightly longer but smoother route to keep the passenger happy, even if it earns a little less — because it values long-term **customer satisfaction**.

THE STRUCTURE OF AGENTS

Types of Agent Programs

AI uses **4 basic types** of agents to generate intelligent behavior more efficiently.

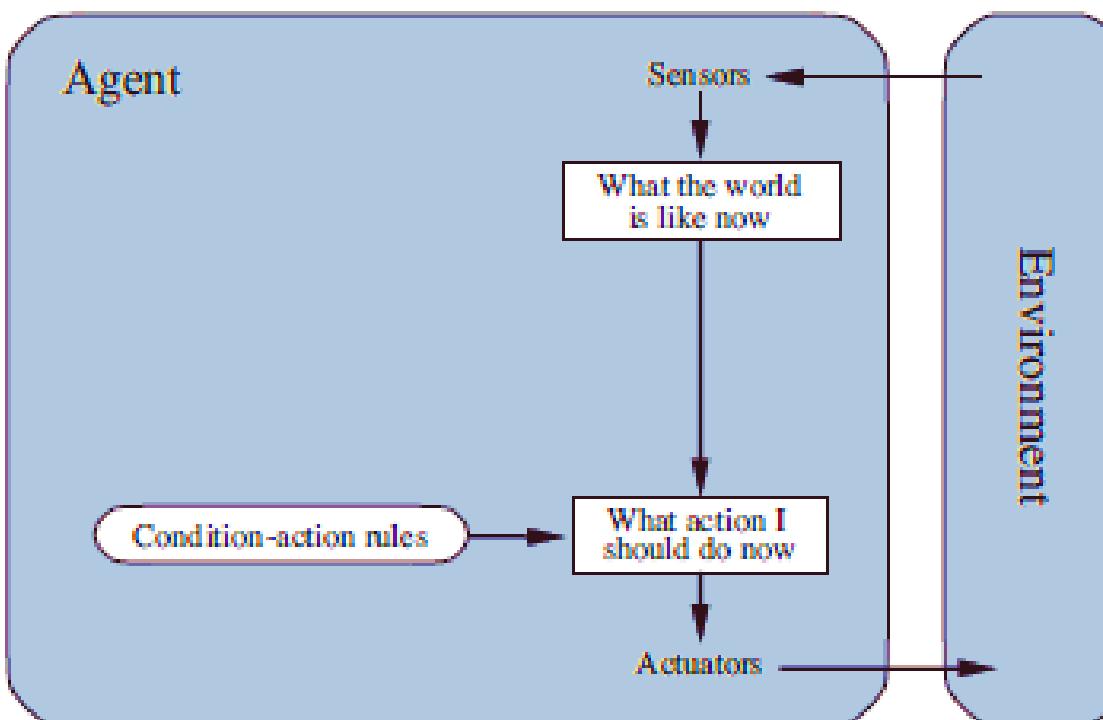
Instead of building **giant tables**, we build **intelligent programs** that combine:

- Perception
- Memory (model)
- Goals
- Utility

These programs can **learn, adapt**, and **scale** to complex environments — unlike table-driven ones.

THE STRUCTURE OF AGENTS

1. Simple reflex agents



A **simple reflex agent** chooses its action **based only on the current percept**—that is, what it sees or senses **right now**—without considering:

- The history of previous perceptions
- Any internal memory or world model

"See something → Do something" (Like an instinctive reflex)



THE STRUCTURE OF AGENTS

1. Simple reflex agents

Example 1: Reflex Vacuum Cleaner

- **Percept:** [Location, Clean/Dirty]

- **Action Rule:**

- If current location is **Dirty**, then → **Suck**
- Else if location is **A**, then → **Right**
- Else if location is **B**, then → **Left**

This agent only looks at **what's under it right now** and acts.

It **doesn't remember** if it cleaned the other square or not.

Example 2: Car Braking Reflex

Imagine you're driving:

- You see **brake lights** on the car ahead (percept)

- You **immediately brake** (action)

That's a reflex. You didn't stop to check if the car actually stopped, or remember past traffic—just reacted.

This is a **condition-action rule**:

if car-in-front-is-braking then initiate-braking

These are also called:

- **If-then rules**
- **Situation-action rules**
- **Productions**



THE STRUCTURE OF AGENTS

1. Simple reflex agents

```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
    persistent: state, the agent's current conception of the world state
                model, a description of how the next state depends on current state and action
                rules, a set of condition-action rules
                action, the most recent action, initially none

    state  $\leftarrow$  UPDATE-STATE(state, action, percept, model)
    rule  $\leftarrow$  RULE-MATCH(state, rules)
    action  $\leftarrow$  rule.ACTION
    return action
```

1. Interpret the current percept into a state description
2. Match it with a **rule** from the rule set
3. Perform the **action** from that rule

Example: Blinking Reflex

When something flies toward your eye:

- You don't think or check history.
- You **automatically blink**.

That's a **hard-wired simple reflex** — no memory, just stimulus \rightarrow response.



THE STRUCTURE OF AGENTS

1. Simple reflex agents

Limitations of Simple Reflex Agents

Although they're simple and fast, they can be **pretty dumb** in complex or uncertain environments.

Problem 1: Partial Observability

Imagine a vacuum agent with **no location sensor**, just a **dirt sensor**:

- It only knows: [Clean] or [Dirty]
- In [Clean], should it move left or right?
- If it chooses wrong (say, always goes left), and it started on the leftmost square—it's stuck in an infinite loop!

Solution? Randomization!

If the agent randomly chooses between Left and Right when it sees [Clean], then:

- It will eventually reach the other square
- If the square is Dirty → Clean it!

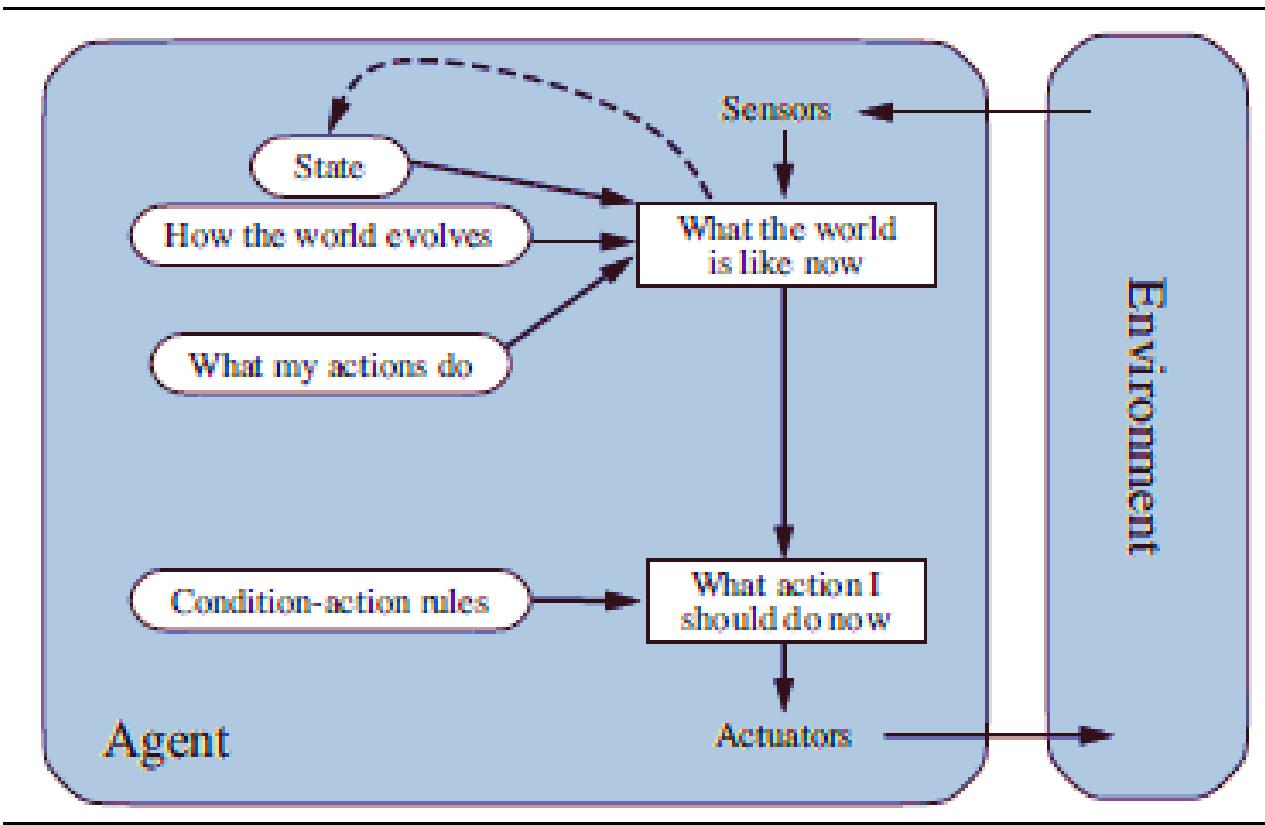
This **randomized agent** actually performs **better** than a purely deterministic one in such a case.

Is Random Behavior Always Good?

- In **multiagent games** or **unpredictable environments**, randomness can be strategic.
- But in **single-agent environments**, randomization is often a temporary fix—not the best long-term strategy.

THE STRUCTURE OF AGENTS

2. Model-Based Reflex Agents



Use an Internal State (Memory)

To handle partial observability, a **model-based reflex agent** maintains an **internal state** — a kind of memory that keeps track of important information it can't see right now but might have seen before. This internal state helps the agent to:

- Infer what the world is like now, even if not everything is visible.
- Make decisions based on *past perceptions* and *actions taken*.

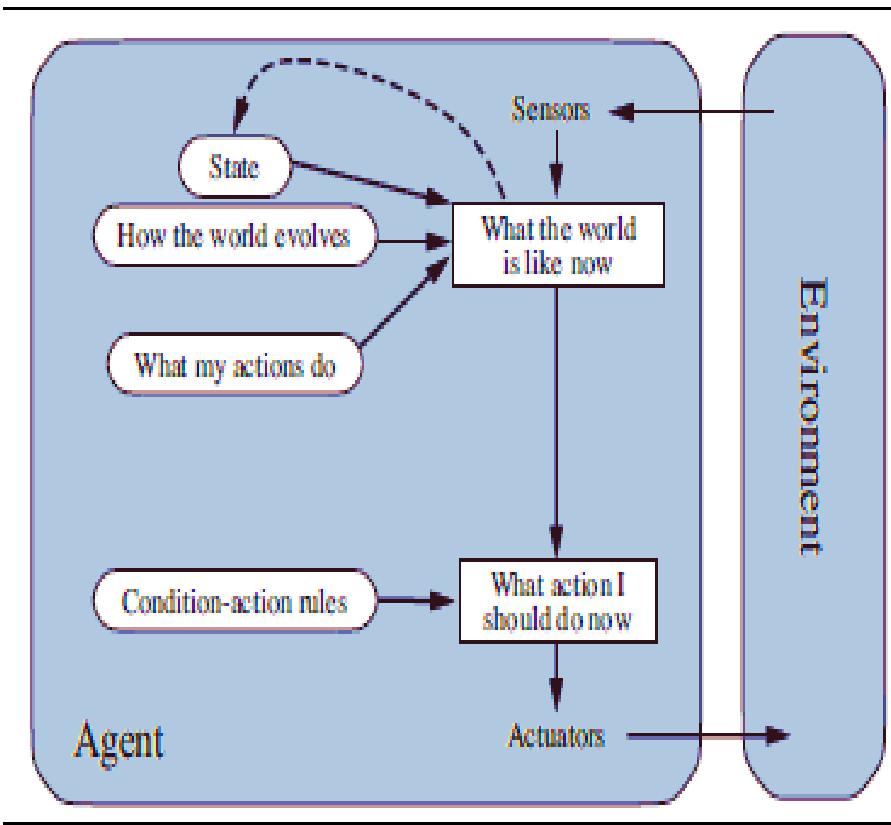
For example:

- A self-driving car might store the **previous camera frame** to tell whether **brake lights** of the car in front have turned on.
- To change lanes safely, the car must remember where other vehicles were last seen, in case they're currently out of view.
- To even start driving, it must remember if it has the **car keys**.



THE STRUCTURE OF AGENTS

2. Model-Based Reflex Agents



How Internal State Works

Updating this internal state needs two types of knowledge:

1. How the world evolves:

1. Example: A car that was behind you a second ago is probably closer now.

2. How the agent's actions change the world:

1. Example: Turning the steering wheel to the right makes the car go right.

This combined knowledge is called the **model of the world**. Hence, we call this a **model-based agent**.

THE STRUCTURE OF AGENTS

2. Model-Based Reflex Agents



```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
            model, a description of how the next state depends on current state and action
            rules, a set of condition-action rules
            action, the most recent action, initially none

  state  $\leftarrow$  UPDATE-STATE(state, action, percept, model)
  rule  $\leftarrow$  RULE-MATCH(state, rules)
  action  $\leftarrow$  rule.ACTION
  return action
```

Uncertainty

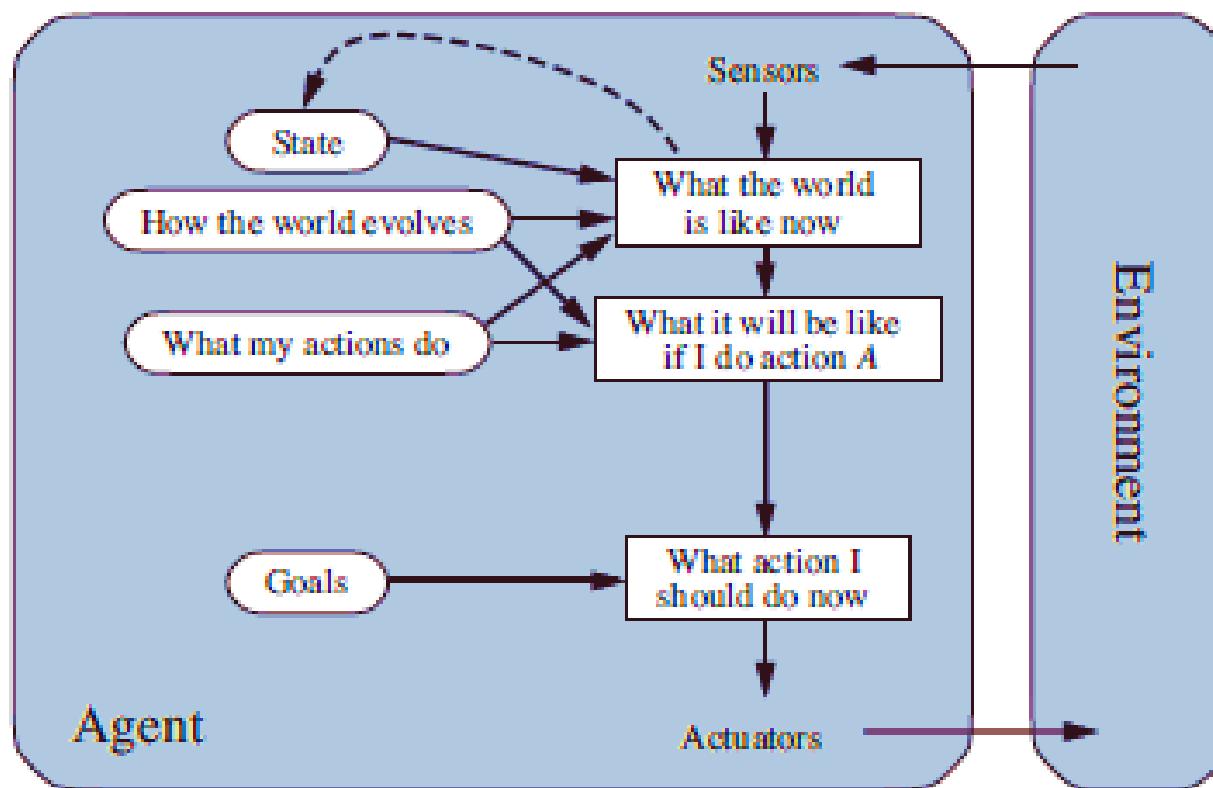
In practice, the agent **can't always know exactly** what's happening (e.g., a blocked road behind a truck). So the internal state is often a **best guess**, based on incomplete information.

Example: Self-Driving Taxi

- The taxi might be on the same road at the same time but heading to different destinations.
- The destination is not part of the environment — it's stored in the **internal state**.
- So even if the external situation is the same, the **internal state** (destination) influences the agent's actions (e.g., whether to stop for fuel).

THE STRUCTURE OF AGENTS

3. Goal-Based Agents



Goal-based agents go beyond simply reacting to the current environment—they select actions based on desired outcomes (goals).

Knowing just the current state isn't always enough to decide the next action.

Example: A taxi at a junction can't decide whether to go left, right, or straight without knowing the passenger's destination.



THE STRUCTURE OF AGENTS

3. Goal-Based Agents

- A **goal-based agent** combines:
- **Current state** of the environment.
- **Goal information** (desired outcome).
- **Model** of the environment (how the world evolves based on actions).
- The agent selects actions that will lead to goal achievement.

Advantages Over Reflex Agents:

Reflex Agent	Goal-Based Agent
Reacts to current percepts using condition-action rules.	Considers future consequences of actions.
Less flexible – hardcoded rules.	More flexible – goals and models can be updated.
Cannot adapt easily to new destinations or conditions.	Can change goals and adapt behaviors dynamically.



THE STRUCTURE OF AGENTS

3. Goal-Based Agents

Goal-Based Reasoning:

- Involves **search** and **planning**:
 - “What will happen if I take this action?”
 - “Will this lead me closer to my goal?”

Example Scenario:

- Reflex Agent: Sees brake lights → Brakes (hardcoded rule).
- Goal-Based Agent: Sees brake lights → Reasons “Car in front is slowing down” → Brakes to avoid collision.

Flexibility in Behavior:

- Goal can change (e.g., new destination).
- Knowledge updates (e.g., road is slippery when it rains).
- Behavior adjusts automatically—no need to rewrite rules.

THE STRUCTURE OF AGENTS



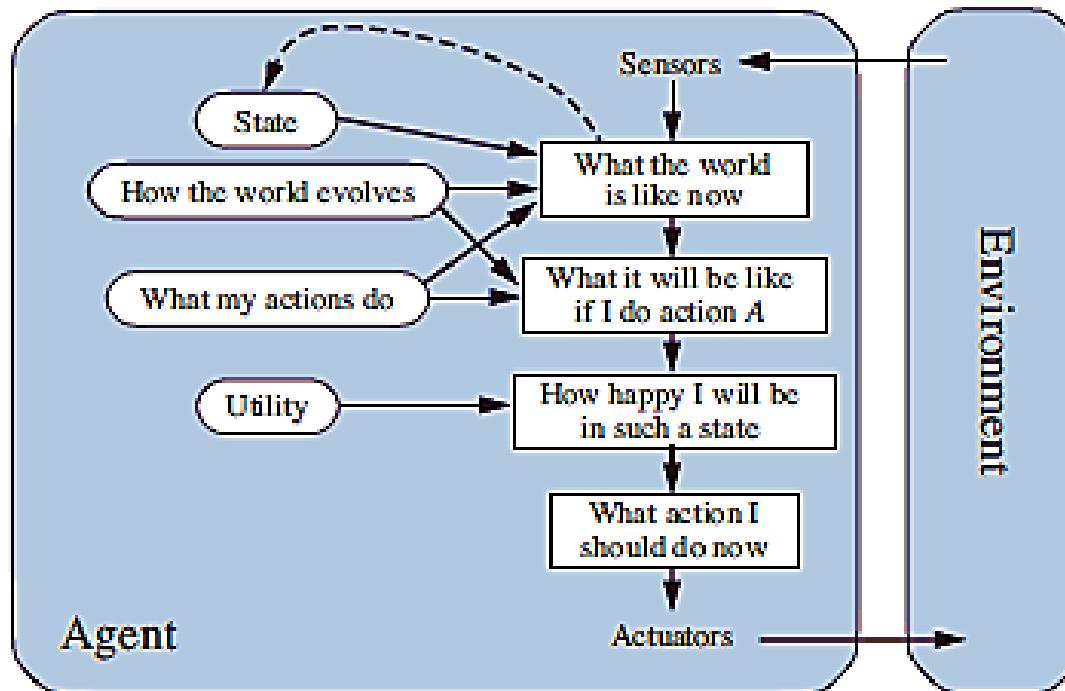
Why Goals Alone Are Not Enough

In AI, setting a **goal** gives the agent something to aim for — like getting a taxi to its destination. But achieving the goal is not the only concern:

- There are **many ways** to reach the same goal.
- Some ways may be **faster, safer, cheaper, or more reliable** than others. So, just knowing whether a goal is achieved (a binary happy/unhappy outcome) is too simplistic.

THE STRUCTURE OF AGENTS

4. Utility-based agents



To make better decisions, agents need to **evaluate** and **compare** how good each possible outcome is — **not just whether a goal is reached**.

- The term "**utility**" is used instead of "happiness" to sound more scientific and measurable.

- Utility represents how **desirable** a world state is for the agent.

Think of **utility as a numerical score** that measures how well the agent is doing — not just pass/fail.



THE STRUCTURE OF AGENTS

4. Utility-based agents

◆ Utility Function vs. Performance Measure

- The **external performance measure** is how the environment (or a human observer) evaluates the agent's behavior.
- The **utility function** is the **agent's internal version** of this — a function it uses to choose between actions.
If the utility function **matches** the performance measure, then choosing actions that **maximize utility** will make the agent appear **rational**.



THE STRUCTURE OF AGENTS

4. Utility-based agents

Why Utility-Based Agents Are Powerful

Utility-based agents are more **flexible** and **intelligent** than goal-based ones. Here's why:

1. Trade-offs between conflicting goals:

1. Example: Safety vs. Speed in a taxi.
2. Utility allows the agent to balance such trade-offs wisely.

2. Uncertain outcomes:

1. When the agent can't be sure it will achieve a goal (due to uncertainty in the environment), **expected utility** helps:
 1. The agent chooses the action that gives the **highest average utility** based on all possible outcomes.



THE STRUCTURE OF AGENTS

4. Utility-based agents

◆ **Expected Utility**

- In uncertain environments, agents must think probabilistically.
- **Expected Utility** = Sum of (Probability of outcome × Utility of outcome).
- The agent selects the action with the **maximum expected utility**.

This concept is foundational to decision-making under uncertainty — seen in fields like economics, AI planning, and reinforcement learning.

THE STRUCTURE OF AGENTS



Is It That Simple to Build Intelligent Agents?

While **maximizing utility** seems like a clean and rational strategy, **in practice it's very challenging**:

- The agent must:
 - **Model the environment**
 - **Track its state**
 - **Make predictions**
 - **Reason and learn** from experience
 - Choosing the best action involves solving **computationally hard problems**, especially in **complex, dynamic, and partially observable** environments.
- So, even though the theory says “maximize expected utility,” **building agents that can actually do this is hard**, and solving these challenges is a big part of AI research.

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY



What Is a Rational Agent?

- An agent that does the "right thing."
- The "right thing" is the action that maximizes its performance.
- Focuses on achieving the best **outcome** based on consequences.

How Do We Measure Performance?

- With a **Performance Measure**: An objective standard that evaluates a sequence of environment states.
- **Crucial Point**: Success is measured by the state of the **environment**, not the agent's internal opinion.
 - This prevents an agent from simply "deluding itself" that it's doing well.

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY



The Designer's Trap: Be Careful What You Ask For

A rational agent will do exactly what it's told to do to maximize its score.

A poorly designed performance measure will be exploited, leading to unintended and undesirable behavior.

Slogan: "**What you ask for is what you get.**"

Example: Vacuum Cleaner Agent

Let's consider two performance measures for the vacuum cleaner agent:

Performance Measure 1 (Bad Design):

- **"Amount of dirt cleaned over 8 hours"**
→ This could **incentivize cheating**:

- Agent cleans dirt
- Agent dumps it again
- Agent re-cleans it

Infinite loop to boost performance artificially

Performance Measure 2 (Better Design):

- **"1 point per clean square at every time step"**

- Encourages **sustained cleanliness**, not just action
- Penalizes inefficiency (like excessive movement, electricity, noise)

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

How Do We Measure Performance?

- With a **Performance Measure**: An objective standard that evaluates a sequence of environment states.
- **Crucial Point**: Success is measured by the state of the **environment**, not the agent's internal opinion.
 - This prevents an agent from simply "deluding itself" that it's doing well.

The Golden Rule of Performance Measures

- Design the measure based on the **OUTCOME** you want in the environment...
- ...not the **PROCESS** you think the agent should follow.

Good Performance Measures Focus on Environment States, Not Agent Behavior

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Reflex Agent vs. Rational Agent





GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Reflex Agent vs. Rational Agent

Feature	Reflex Agent	Rational Agent
Decision Basis	Current percept only	Percept history , knowledge , and expected outcomes
Memory	None (stateless)	May use memory , models, and internal state
Goal Awareness	Not goal-oriented	Always goal-oriented (maximizes performance measure)
Adaptability	Not adaptable to new situations	Adapts based on new knowledge and experience
Examples	Basic vacuum agent that sucks if it sees dirt	Smart vacuum that plans cleaning based on room map + battery
Agent Type	Simple reflex or reflex with condition-action rules	Goal-based or utility-based agent

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Rationality:

A **rational agent** is one that:

For **every percept sequence**, selects an action that is **expected to maximize** its **performance measure**, given:

- 1.What it has **perceived so far** (percept history),
- 2.What it **knows about the environment** (built-in knowledge),
- 3.What it **can do** (available actions),
- 4.And how **success is defined** (performance measure).

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Components That Affect Rationality

Component	Meaning
Performance Measure	What defines success (e.g., how many squares are clean)
Prior Knowledge	What the agent knows in advance (e.g., layout, rules)
Available Actions	What it can do (e.g., Left, Right, Suck)
Percept Sequence	What it has sensed so far (e.g., "I'm in A, and it's clean")

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Vacuum Cleaner Agent — Is It Rational?



simple reflex agent:

If square is dirty → Suck

Else → move to the other square (Left or Right)

Under These Conditions:

- ✓ **Performance measure:** +1 per clean square per time step, for 1000 steps
- ✓ **Geography** is known (just A and B)
- ✗ **Dirt status and start location** are unknown
- ✓ **Sensors:** Know current location and whether it is dirty
- ✓ **Actions:** Suck, Left, Right

Verdict: This agent *is rational* in this context

Because:

- It **cleans when needed**
- It **moves to find dirt if not present**
- It has no memory, but **its policy is optimal** for this static, fully observable world



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Vacuum Cleaner Agent — Is It Rational?

✗ When Is It NOT Rational?

Let's tweak the environment:

⚠ Case 1: Penalty for movement

Movement (Left/Right) costs -1 point.

- The reflex agent **keeps oscillating** even after everything is clean.
- Better: An agent that **stops moving** once it **knows** everything is clean.
- Reflex agent = ✗ irrational here.

Case 2: Environment gets dirty again

Dirt **reappears** over time.

- Reflex agent won't know this unless it **revisits each square**.
- Better: An agent that **periodically checks** or follows a patrol pattern.
- Reflex agent = ✗ irrational here.

Case 3: Unknown environment (more than A & B)

Geography is not known initially.

- Reflex agent only acts within A and B.
- A **rational agent should explore**.
- Reflex agent = ✗ irrational again.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Vacuum Cleaner Agent — Is It Rational?

Challenge (Design Task)

Design agents for the three cases above:

1. With movement penalty:

- Agent maintains an internal state:
 - Marks square as clean once cleaned.
 - Stops moving once both squares are known to be clean.
- Only acts if dirt is present.
- **Saves points by staying still.**

2. If dirt reappears:

- Agent implements **periodic checking**:
 - E.g., Alternate between A and B every 10 time steps.
 - Keeps track of last dirt time per square.
- Keeps performance up even if dirt returns.

3. Unknown environment:

- Agent starts exploring:
 - Moves in a direction until hitting a wall.
 - Builds internal map.
 - Cleans as it goes.
- Rationality here includes **exploration + cleaning**.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy

1. Rationality ≠ Omniscience

Rationality: Do the best you can *with what you currently know.*

Omniscience: Know everything, including the future (which is impossible).

Example: Crossing the street

You look both ways — no cars. You start crossing.

 A random airplane part falls on you.

Were you **irrational**? No!

You made the **best decision based on available information.**

 **Moral:** Rational ≠ Perfect. Expecting perfection = demanding a *crystal ball.*

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy

2. Information Gathering is Rational

Rational agents should **take actions to improve future decisions.**

Example: Crossing a busy road

If the agent doesn't **look** for cars first, it's acting blindly. Even if no car is visible, **not looking is irrational** because of the **risk**.

So: "**Looking before crossing**" is part of rational behavior.

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy

3. Exploration Example (Vacuum Agent)

In a **partially known house**, the vacuum agent must:

- Explore new rooms
- Find dirt
- Build a map of its world

This is **exploration as information gathering** — an essential part of being rational.





GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy

4. Learning

Even if an agent starts with some built-in knowledge, it should **improve over time.**

Funny Bug Example: Dung Beetle

- Lays eggs
- Fetches dung ball
- If ball is taken away → Still "plugs" the hole
- Doesn't learn that the ball is **gone**

Another: Sphex Wasp

- Drags a caterpillar to its burrow
- If you move it slightly... she **starts over** each time!
- She never learns the plan is failing.

These insects **lack learning**. They rely only on built-in behaviors. **Not autonomous.**



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy

4. Learning

Even if an agent starts with some built-in knowledge, it should **improve over time.**

Funny Bug Example: Dung Beetle

- Lays eggs
- Fetches dung ball
- If ball is taken away → Still "plugs" the hole
- Doesn't learn that the ball is **gone**

Another: Sphex Wasp

- Drags a caterpillar to its burrow
- If you move it slightly... she **starts over** each time!
- She never learns the plan is failing.

These insects **lack learning**. They rely only on built-in behaviors. **Not autonomous.**



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy

5. Autonomy

The more an agent relies on its **own percepts** rather than **programmed instructions**, the more **autonomous** it is.

Vacuum Cleaner Example:

- Reflex agent: Cleans based on fixed rules
- Autonomous agent: Learns patterns of dirt reappearance (e.g., kids spill snacks at 4 PM)

Over time, a **learning agent** becomes **less dependent on its designer** and more **self-sufficient**.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Omniscience, Learning, and Autonomy

Robot vs. Pet

Concept	Reflex Agent (Non-autonomous)	Rational, Learning Agent (Autonomous)
Behavior Basis	Hard-coded by designer	Learns from environment + percepts
Flexibility	Fails in new situations	Adapts, improves over time
Omniscience	Assumes nothing about future	Plans using expected outcomes
Autonomy	Low	High (grows independent)



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

When designing an intelligent agent, the first and most essential step is to **define the task environment** using the **PEAS** model:

- **P**: Performance Measure — Criteria for success
- **E**: Environment — The surroundings in which the agent operates
- **A**: Actuators — The tools or components used by the agent to act
- **S**: Sensors — Devices used by the agent to perceive the environment

This model helps clearly specify what the agent is supposed to do, in what setting, and how it interacts with its surroundings.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

Example 1: Simple Vacuum Cleaner Agent

PEAS Element	Description
Performance	Floor cleaned efficiently, minimal electricity use
Environment	Rooms with floors, walls, and dirt patches
Actuators	Motors to move and vacuum dust
Sensors	Dirt detector, wall bumper sensor

Simplified World: Only two locations (A and B), clean or dirty.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

Example 2: Automated Taxi Driver Agent

A more complex, real-world agent, with a highly dynamic and unpredictable environment.

PEAS Element	Description
Performance	Safe, quick, legal, comfortable rides; low fuel use; high profit
Environment	Urban and rural roads, traffic, pedestrians, weather, passengers
Actuators	Steering wheel, accelerator, brakes, gear, horn, indicators, display/speaker
Sensors	Cameras, GPS, speedometer, sonar, accelerometer, microphone, engine sensors

Key Considerations:

- Must navigate different geographies (e.g., snowy Alaska vs. sunny California)
- Might drive on the right (India/US) or left (UK/Japan)
- Needs to **balance** multiple goals: safety vs. speed, comfort vs. profit



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

Real vs. Artificial Environments

- Some agents operate in **physical environments** (like robots).
- Others exist in **virtual environments** (like software agents).
- What matters most is the **complexity** of the agent's interaction with the environment, not whether it's real or digital.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

Example 1: Conveyor Belt Inspection Robot

PEAS Element	Description
Performance	Accurate defect detection, high throughput
Environment	Controlled factory belt with predictable object flow
Actuators	Robot arms to accept/reject parts
Sensors	Camera, light sensors

Simplified Conditions: Same lighting, object types, and position → Easier to design agent.

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

Example 4: Web-based Software Agent (Softbot)

A **softbot** is a software agent that operates in a virtual, highly complex environment (e.g., Internet).

PEAS Element	Description
Performance	Provide relevant news, maintain uptime, generate ad revenue
Environment	Internet: dynamic sources, users, advertisers
Actuators	Display content, adjust layout, serve ads
Sensors	User input (clicks, preferences), server status, web scrapers

- Example: If a news website goes down, it must find alternate sources.
- Must learn and adapt based on user interest and advertiser behavior.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Specifying the task environment

PEAS Framework – Task Environment Design

- **PEAS** helps in clearly defining agent design tasks.
- Agents range from simple to highly complex:
 - Simple: Vacuum cleaner, part inspection robots
 - Complex: Autonomous cars, web-based softbots
- Designing intelligent agents depends heavily on **how well the task environment is understood and specified.**



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Properties of Task Environments (Dimensions to Classify Them)

Each task environment can be described along **several key dimensions**.

These characteristics influence the **design of the agent** and the **techniques used** to build it.

1. Fully Observable vs. Partially Observable

- **Fully Observable:** The agent's sensors capture the **entire relevant state** of the environment at all times.
 - Example: Chess – the entire board is visible.

- **Partially Observable:** Agent has **limited** or **noisy perception** of the environment.

- Example: Taxi driving – can't see around corners or know other drivers' intentions.

Example: Poker – can't see opponents' cards.

2. Single Agent vs. Multiagent

- **Single Agent:** Only **one intelligent agent** acts in the environment.

-

Example: Crossword puzzle solver.

- **Multiagent:** **Multiple intelligent agents** interact.

-

Cooperative: All agents aim to achieve a shared goal (e.g., self-driving cars avoiding collisions).

-

Competitive: Opposing goals (e.g., chess or poker).

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Properties of Task Environments .

3. Deterministic vs. Stochastic

•**Deterministic:** Next state is **completely determined** by the current state and action.
Example: Chess (ignoring opponent's unpredictability).

•**Stochastic:** Involves **randomness or uncertainty** in outcomes.
Example: Taxi driving – unpredictable traffic or weather.

•**Uncertain:** Either **partially observable** or **stochastic**.

•**Nondeterministic:** Actions have **possible outcomes** but without probabilities.

4. Episodic vs. Sequential

•**Episodic:** Each action is **independent** of previous ones.
Example: Image classification – each image is treated separately.

•**Sequential:** Current actions **affect future states**.

Example: Chess or medical diagnosis – early decisions have long-term impact.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Properties of Task Environments .

5. Static vs. Dynamic

• **Static:** Environment **does not change** while the agent is thinking.

 Example: Crossword puzzle.

• **Dynamic:** Environment **changes over time**, even while the agent is deciding.

Example: Driving – traffic and surroundings constantly shift.

• **Semidynamic:** Environment remains unchanged, but **performance score changes** over time.

Example: Chess with a clock.

6. Discrete vs. Continuous

• **Discrete:** A **finite number** of distinct percepts, actions, or states.

Example: Chess – fixed number of board states and moves.

• **Continuous:** States and actions can take on **any value within a range**.

Example: Driving – position, speed, steering angles vary continuously.



GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Properties of Task Environments .

7. Known vs. Unknown

• **Known:** Agent (or designer) **knows the rules** of how the environment works.

Example: Chess – rules are well-defined.

• **Unknown:** Agent must **learn** how the environment behaves.

Example: New video game – agent explores how the controls work.

- ◆ Known vs. Unknown refers to **knowledge of environment mechanics**, not observability.

GOOD BEHAVIOR: THE CONCEPT OF RATIONALITY

Properties of Task Environments .

Examples of Task Environments

Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess (with clock)	Fully	Multi	Deterministic	Sequential	Semidynamic	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semidynamic	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete



THE STRUCTURE OF AGENTS

Agent = Architecture + Program

- Agent Function:** Maps percepts → actions
- Agent Program:** Implements this function
- Architecture:** Executes the program using physical hardware
- Designing an intelligent agent means choosing the **right program** for the **right architecture** to interact effectively with the environment.

THE STRUCTURE OF AGENTS



Agent = Architecture + Program

Agent Program

- The **agent program** is the core software that implements the **agent function**.
- It defines **how the agent maps percepts to actions**.
- It's the "brain" of the agent.
- Examples:
 - A vacuum cleaner program that tells the robot to turn left if it detects a wall.
 - A chess-playing program that chooses the best move after analyzing the board.

Agent Architecture

- The **architecture** is the **computing platform** that the agent program runs on.
- It includes:
 - **Sensors** (to perceive the environment)
 - **Actuators** (to perform actions)
 - **Computational hardware** (e.g., CPU, memory)
- The architecture might be:
 - A **simple PC** for a software agent
 - A **robot** with legs, wheels, cameras, microphones, etc.



THE STRUCTURE OF AGENTS

Agent = Architecture + Program

Architecture + Program

- The architecture makes **sensor data** available to the program.
- The agent program **processes percepts**, makes decisions, and outputs **actions**.
- These actions are executed by the **actuators**.

Loop:

1. Sensor detects input from environment
2. Architecture passes percepts to agent program
3. Agent program decides what to do
4. Architecture sends the action to actuators
5. Environment changes → cycle repeats

THE STRUCTURE OF AGENTS

Agent = Architecture + Program

Agent Programs

An **agent program** is the **software** part of an agent that:

Takes current percepts (sensory inputs) from the environment

Returns an action to be performed by the agent

💡 Think of it like this:

If an agent is a robot chef, the **agent program** is its cooking logic (e.g., "if the onion is chopped, then sauté it").

Agent Program vs. Agent Function

•**Agent Function:** Maps the **entire history of percepts** → action

•**Agent Program:** Uses **only the current percept** (because that's all it gets at the moment)

If the agent needs to remember past events, it must build memory **inside the program**.

THE STRUCTURE OF AGENTS

Agent = Architecture + Program

Example: Table-Driven Agent

```
function TABLE-DRIVEN-AGENT(percept) returns an action
    persistent: percepts, a sequence, initially empty
                table, a table of actions, indexed by percept sequences, initially fully specified
    append percept to the end of percepts
    action  $\leftarrow$  LOOKUP(percepts, table)
    return action
```

THE STRUCTURE OF AGENTS

Agent = Architecture + Program

Example: Table-Driven Agent



Why Table-Driven Agents Are Impractical

Let's say:

- You receive 1 percept per second
- Your agent runs for 1 hour → 3600 percepts
- Even with 10 types of percepts → Table needs 10^{3600} entries

That's **more entries than atoms in the universe!**

Problems:

- Can't store such a huge table (too much memory)
- Can't fill it manually or learn all combinations
- No guidance on what the entries should be

But Table-Driven Agent Does Work

The approach technically **implements the correct agent function** — it gives the right action — but it's not **efficient or intelligent**.

So the goal of AI is:

To design **small, clever programs** that act rationally — **without** needing huge tables.

Like replacing **books of square roots** with a tiny calculator algorithm (like Newton's Method)!



THE STRUCTURE OF AGENTS

Types of Agent Programs

AI uses **4 basic types** of agents to generate intelligent behavior more efficiently.

① Simple Reflex Agents

- React only to current percept
- No memory of past
- Use **if-then rules**

Example: Reflex Vacuum Agent

IF status = Dirty THEN Suck
ELSE IF location = A THEN go Right
ELSE IF location = B THEN go Left

③ Goal-Based Agents

- Decide actions based on **desired outcomes** (goals)
- Can **plan** and evaluate different possibilities

Example: Google Maps

It knows your goal is to reach “Coimbatore Railway Station.”
It explores multiple routes and picks the fastest one using its knowledge of traffic and roads.

② Model-Based Reflex Agents

- Keep **track of the world** internally (have a model)
- Update the model based on percepts
- Use rules and memory

Example: A Smart Thermostat

If it knows the current temperature and remembers what time it last changed settings, it can avoid turning the AC on/off too frequently.

④ Utility-Based Agents

- Choose actions that **maximize utility** (i.e., satisfaction or benefit)
- Can handle **trade-offs** (e.g., cost vs. comfort)

Example: Self-Driving Taxi

It may choose a slightly longer but smoother route to keep the passenger happy, even if it earns a little less — because it values long-term **customer satisfaction**.

THE STRUCTURE OF AGENTS

Types of Agent Programs

AI uses **4 basic types** of agents to generate intelligent behavior more efficiently.

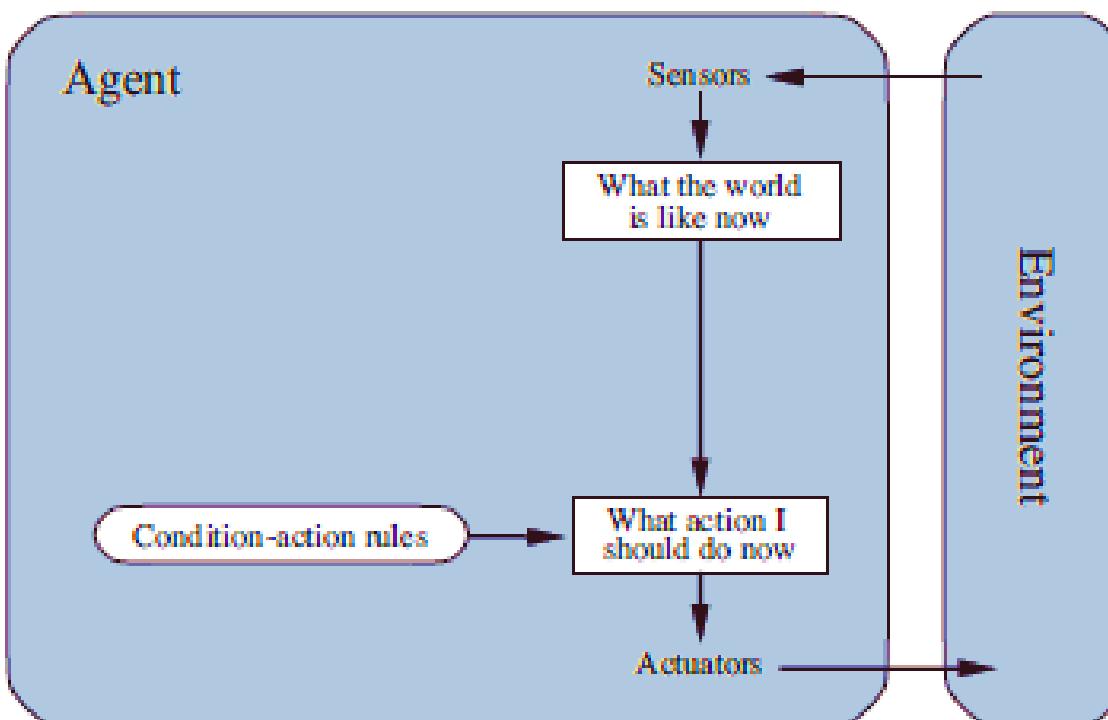
Instead of building **giant tables**, we build **intelligent programs** that combine:

- Perception
- Memory (model)
- Goals
- Utility

These programs can **learn, adapt**, and **scale** to complex environments — unlike table-driven ones.

THE STRUCTURE OF AGENTS

1. Simple reflex agents



A **simple reflex agent** chooses its action **based only on the current percept**—that is, what it sees or senses **right now**—without considering:

- The history of previous perceptions
- Any internal memory or world model

"See something → Do something" (Like an instinctive reflex)

THE STRUCTURE OF AGENTS



1. Simple reflex agents

Example 1: Reflex Vacuum Cleaner

- **Percept:** [Location, Clean/Dirty]

- **Action Rule:**

- If current location is **Dirty**, then → **Suck**
- Else if location is **A**, then → **Right**
- Else if location is **B**, then → **Left**

This agent only looks at **what's under it right now** and acts.

It **doesn't remember** if it cleaned the other square or not.

Example 2: Car Braking Reflex

Imagine you're driving:

- You see **brake lights** on the car ahead (percept)

- You **immediately brake** (action)

That's a reflex. You didn't stop to check if the car actually stopped, or remember past traffic—just reacted.

This is a **condition-action rule**:

if car-in-front-is-braking then initiate-braking

These are also called:

- **If-then rules**
- **Situation-action rules**
- **Productions**



THE STRUCTURE OF AGENTS

1. Simple reflex agents

```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
    persistent: state, the agent's current conception of the world state
                model, a description of how the next state depends on current state and action
                rules, a set of condition-action rules
                action, the most recent action, initially none

    state  $\leftarrow$  UPDATE-STATE(state, action, percept, model)
    rule  $\leftarrow$  RULE-MATCH(state, rules)
    action  $\leftarrow$  rule.ACTION
    return action
```

1. Interpret the current percept into a state description
2. Match it with a **rule** from the rule set
3. Perform the **action** from that rule

Example: Blinking Reflex

When something flies toward your eye:

- You don't think or check history.
- You **automatically blink**.

That's a **hard-wired simple reflex** — no memory, just stimulus \rightarrow response.



THE STRUCTURE OF AGENTS

1. Simple reflex agents

Limitations of Simple Reflex Agents

Although they're simple and fast, they can be **pretty dumb** in complex or uncertain environments.

Problem 1: Partial Observability

Imagine a vacuum agent with **no location sensor**, just a **dirt sensor**:

- It only knows: [Clean] or [Dirty]
- In [Clean], should it move left or right?
- If it chooses wrong (say, always goes left), and it started on the leftmost square—it's stuck in an infinite loop!

Solution? Randomization!

If the agent randomly chooses between Left and Right when it sees [Clean], then:

- It will eventually reach the other square
- If the square is Dirty → Clean it!

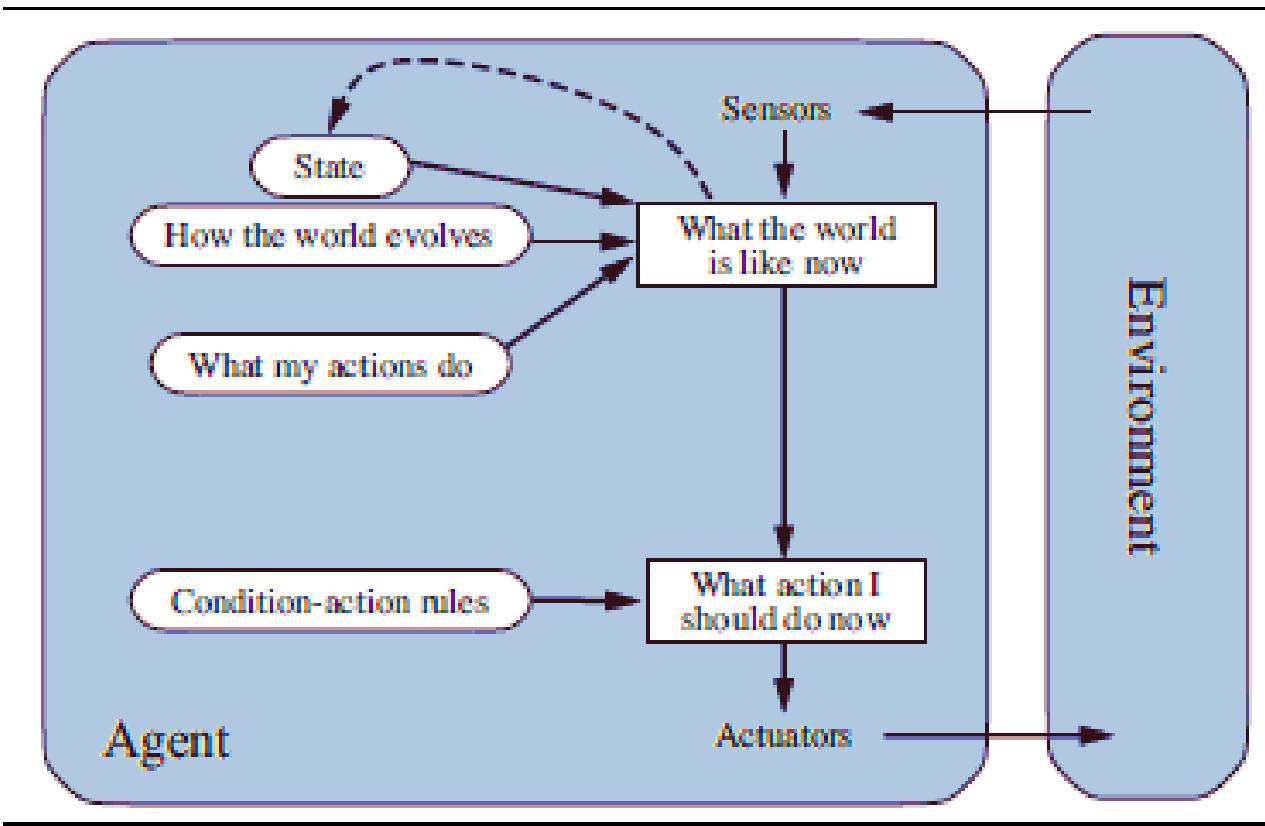
This **randomized agent** actually performs **better** than a purely deterministic one in such a case.

Is Random Behavior Always Good?

- In **multiagent games** or **unpredictable environments**, randomness can be strategic.
- But in **single-agent environments**, randomization is often a temporary fix—not the best long-term strategy.

THE STRUCTURE OF AGENTS

2. Model-Based Reflex Agents



Use an Internal State (Memory)

To handle partial observability, a **model-based reflex agent** maintains an **internal state** — a kind of memory that keeps track of important information it can't see right now but might have seen before. This internal state helps the agent to:

- Infer what the world is like now, even if not everything is visible.
- Make decisions based on *past perceptions* and *actions taken*.

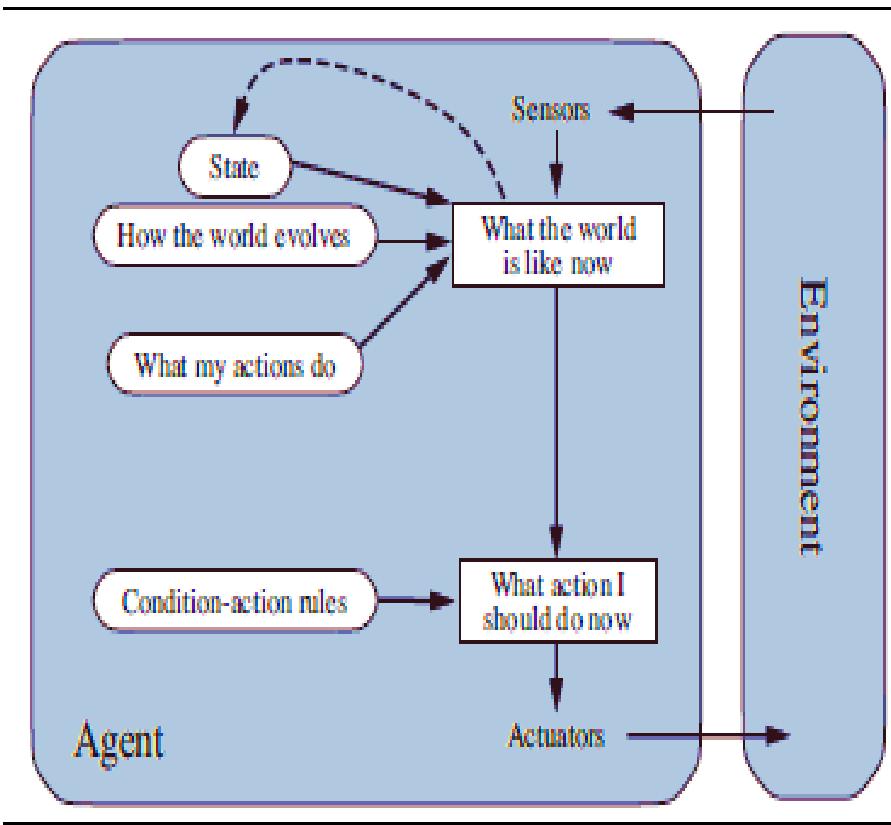
For example:

- A self-driving car might store the **previous camera frame** to tell whether **brake lights** of the car in front have turned on.
- To change lanes safely, the car must remember where other vehicles were last seen, in case they're currently out of view.
- To even start driving, it must remember if it has the **car keys**.



THE STRUCTURE OF AGENTS

2. Model-Based Reflex Agents



How Internal State Works

Updating this internal state needs two types of knowledge:

1. How the world evolves:

1. Example: A car that was behind you a second ago is probably closer now.

2. How the agent's actions change the world:

1. Example: Turning the steering wheel to the right makes the car go right.

This combined knowledge is called the **model of the world**. Hence, we call this a **model-based agent**.

THE STRUCTURE OF AGENTS



2. Model-Based Reflex Agents

```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
    persistent: state, the agent's current conception of the world state
                model, a description of how the next state depends on current state and action
                rules, a set of condition-action rules
                action, the most recent action, initially none

    state  $\leftarrow$  UPDATE-STATE(state, action, percept, model)
    rule  $\leftarrow$  RULE-MATCH(state, rules)
    action  $\leftarrow$  rule.ACTION
    return action
```

Uncertainty

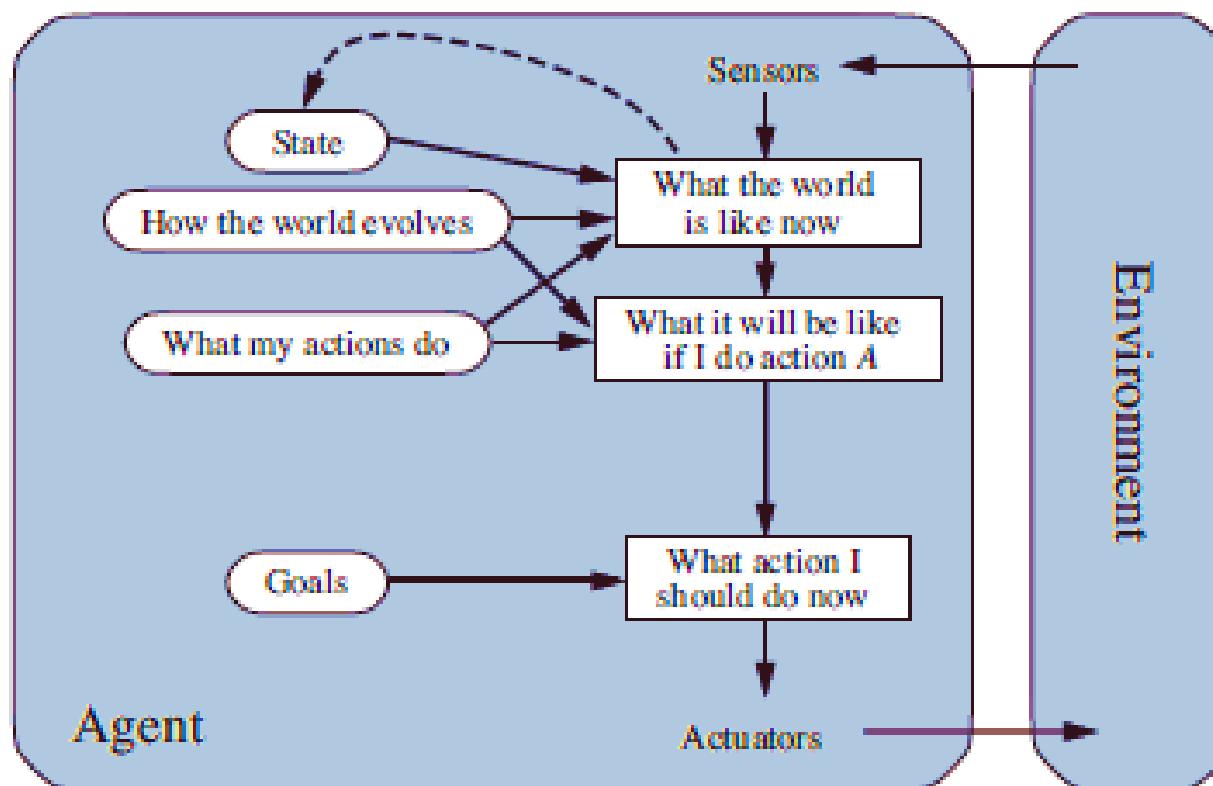
In practice, the agent **can't always know exactly** what's happening (e.g., a blocked road behind a truck). So the internal state is often a **best guess**, based on incomplete information.

Example: Self-Driving Taxi

- The taxi might be on the same road at the same time but heading to different destinations.
- The destination is not part of the environment — it's stored in the **internal state**.
- So even if the external situation is the same, the **internal state** (destination) influences the agent's actions (e.g., whether to stop for fuel).

THE STRUCTURE OF AGENTS

3. Goal-Based Agents



Goal-based agents go beyond simply reacting to the current environment—they select actions based on desired outcomes (goals).

Knowing just the current state isn't always enough to decide the next action.

Example: A taxi at a junction can't decide whether to go left, right, or straight without knowing the passenger's destination.



THE STRUCTURE OF AGENTS

3. Goal-Based Agents

- A **goal-based agent** combines:
- **Current state** of the environment.
- **Goal information** (desired outcome).
- **Model** of the environment (how the world evolves based on actions).
- The agent selects actions that will lead to goal achievement.

Advantages Over Reflex Agents:

Reflex Agent	Goal-Based Agent
Reacts to current percepts using condition-action rules.	Considers future consequences of actions.
Less flexible – hardcoded rules.	More flexible – goals and models can be updated.
Cannot adapt easily to new destinations or conditions.	Can change goals and adapt behaviors dynamically.



THE STRUCTURE OF AGENTS

3. Goal-Based Agents

Goal-Based Reasoning:

- Involves **search** and **planning**:
 - “What will happen if I take this action?”
 - “Will this lead me closer to my goal?”

Example Scenario:

- Reflex Agent: Sees brake lights → Brakes (hardcoded rule).
- Goal-Based Agent: Sees brake lights → Reasons “Car in front is slowing down” → Brakes to avoid collision.

Flexibility in Behavior:

- Goal can change (e.g., new destination).
- Knowledge updates (e.g., road is slippery when it rains).
- Behavior adjusts automatically—no need to rewrite rules.

THE STRUCTURE OF AGENTS



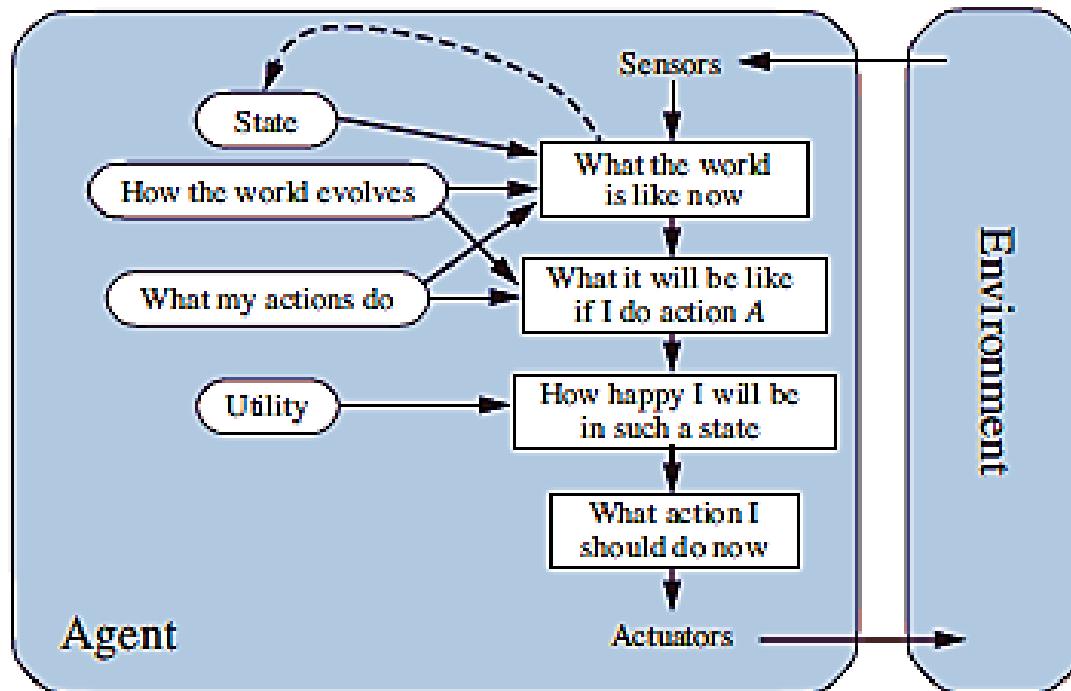
Why Goals Alone Are Not Enough

In AI, setting a **goal** gives the agent something to aim for — like getting a taxi to its destination. But achieving the goal is not the only concern:

- There are **many ways** to reach the same goal.
- Some ways may be **faster, safer, cheaper, or more reliable** than others. So, just knowing whether a goal is achieved (a binary happy/unhappy outcome) is too simplistic.

THE STRUCTURE OF AGENTS

4. Utility-based agents



To make better decisions, agents need to **evaluate** and **compare** how good each possible outcome is — **not just whether a goal is reached**.

- The term "**utility**" is used instead of "happiness" to sound more scientific and measurable.

- Utility represents how **desirable** a world state is for the agent.

Think of **utility as a numerical score** that measures how well the agent is doing — not just pass/fail.

THE STRUCTURE OF AGENTS



4. Utility-based agents

◆ Utility Function vs. Performance Measure

- The **external performance measure** is how the environment (or a human observer) evaluates the agent's behavior.
- The **utility function** is the **agent's internal version** of this — a function it uses to choose between actions.
If the utility function **matches** the performance measure, then choosing actions that **maximize utility** will make the agent appear **rational**.



THE STRUCTURE OF AGENTS

4. Utility-based agents

Why Utility-Based Agents Are Powerful

Utility-based agents are more **flexible** and **intelligent** than goal-based ones. Here's why:

1. Trade-offs between conflicting goals:

1. Example: Safety vs. Speed in a taxi.
2. Utility allows the agent to balance such trade-offs wisely.

2. Uncertain outcomes:

1. When the agent can't be sure it will achieve a goal (due to uncertainty in the environment), **expected utility** helps:
 1. The agent chooses the action that gives the **highest average utility** based on all possible outcomes.



THE STRUCTURE OF AGENTS

4. Utility-based agents

◆ **Expected Utility**

- In uncertain environments, agents must think probabilistically.
- **Expected Utility** = Sum of (Probability of outcome × Utility of outcome).
- The agent selects the action with the **maximum expected utility**.

This concept is foundational to decision-making under uncertainty — seen in fields like economics, AI planning, and reinforcement learning.

THE STRUCTURE OF AGENTS



Is It That Simple to Build Intelligent Agents?

While **maximizing utility** seems like a clean and rational strategy, **in practice it's very challenging**:

- The agent must:
 - **Model the environment**
 - **Track its state**
 - **Make predictions**
 - **Reason and learn** from experience
 - Choosing the best action involves solving **computationally hard problems**, especially in **complex, dynamic, and partially observable** environments.
- So, even though the theory says “maximize expected utility,” **building agents that can actually do this is hard**, and solving these challenges is a big part of AI research.

THE STRUCTURE OF AGENTS



Learning Agents

1. Why Do We Need Learning Agents?

• Manual programming is difficult:

- Turing (1950) pointed out that programming intelligent behavior by hand is time-consuming and inefficient.

• Learning is a better alternative:

- Instead of hard-coding everything, it's better to design agents that **learn from experience**—just like humans and animals do.
- **Example:** Teaching a self-driving car to improve its driving by experience rather than coding every traffic scenario manually.

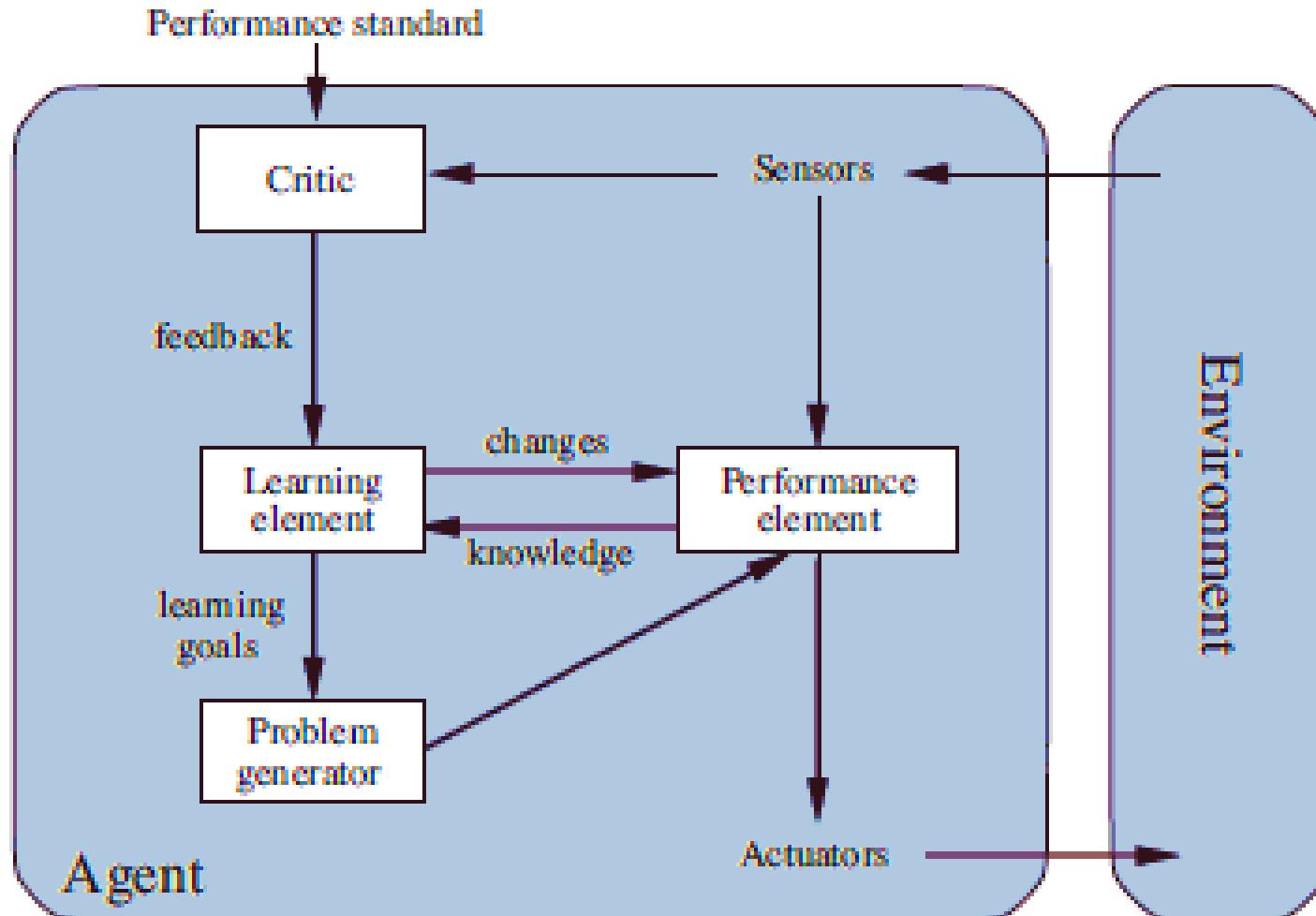
• Advantage:

- Helps agents deal with **unknown or dynamic environments**.
- Enables them to become **smarter over time** than their original design.

THE STRUCTURE OF AGENTS



Learning Agents





THE STRUCTURE OF AGENTS

Learning Agents

2. Components of a Learning Agent

A learning agent has **four core components** that work together to improve its performance:

i. Performance Element

- Function:** Responsible for selecting actions based on current percepts (inputs).

- This is the "doer" of the agent.**

- Example:** In a self-driving taxi, it decides when to stop, turn, or accelerate based on sensor data.

ii. Learning Element

- Function:** Improves the performance element using feedback.

- This is the "thinker" that helps the agent get better at its task.**

- Example:** If a sudden lane change leads to honking or accidents, the learning element updates the rule to avoid such maneuvers.

iii. Critic

- Function:** Evaluates how well the agent is performing based on a fixed standard.

- Tells the agent what is good or bad, like a teacher giving grades.**

- Example:** If passengers complain or don't leave tips, the critic tells the learning element that the ride experience was poor.

The **performance standard** is **fixed and external** – the agent should not modify it. It acts as the "goal" or "objective measure" of success.

iv. Problem Generator

- Function:** Suggests **exploratory actions** to try new things.

- Encourages the agent to experiment** so it can discover better actions in the long run.

- Example:** The taxi tries different braking patterns on wet roads to learn how to stop safely in rain. Like a scientist running experiments—*trying new things even if they might fail*.

THE STRUCTURE OF AGENTS



Learning Agents

3. How Does Learning Happen?

The learning agent improves by **observing outcomes** and **modifying its behavior**:

a) Learning how the world evolves:

- By observing how the environment changes with time or action.
- **Example:** If the agent sees that pressing the brake hard on a dry road stops the car quickly, it learns the effect of its action.

b) Learning the effects of its own actions:

- Learns what each of its actions causes in the environment.
- **Example:** If the agent honks too much, people get annoyed—so it learns to use the horn wisely.

THE STRUCTURE OF AGENTS



Learning Agents

4. Learning and Utility (Reward-Based Learning)

- Some agents aim to **maximize a utility function** (like earning more tips or reducing discomfort).
 - These agents need to learn which actions give **more reward**.
- **Example:** If rough driving results in no tips from customers, the agent learns that smoother driving increases utility.
In animals, such performance standards are hard-wired as **pain, hunger**, etc.

Feedback-Driven Improvement

The learning agent updates its knowledge, behavior, or rules based on feedback from experience to improve its actions over time.

- This applies to all types of intelligent agents: rule-based, model-based, goal-based, or utility-based.

Learning = Adjusting agent components based on feedback to improve future decisions.

- It's about becoming smarter **by experience**, just like humans.
- The agent learns "**What works best**," even in unfamiliar environments, by trial, error, and correction.

THE STRUCTURE OF AGENTS

Learning Agents

Example : Self-Driving Taxi

Component	Role	Example
Performance Element	Makes decisions like lane changes, speed, turns	Taxi chooses to make a sharp left turn
Learning Element	Learns from experience and updates rules	Learns that sharp turns upset passengers
Critic	Evaluates performance based on external feedback	Observes angry passengers or no tip → labels it a bad choice
Problem Generator	Suggests actions to explore new possibilities	Suggests testing braking on different road types

THE STRUCTURE OF AGENTS



Learning Agents

How the Components of Agent Programs Work

What Do Agent Components Do?

Agent programs have components that answer questions like:

- “What is the world like now?”
- “What action should I take now?”
- “What effect will my action have?”

To answer these questions effectively, agents must **represent the world** they interact with.



THE STRUCTURE OF AGENTS

Learning Agents

How the Components of Agent Programs Work

Types of Representations of the Environment (Increasing in Complexity)

1. Atomic Representation (Simple, Black-Box States)

- **Definition:** Each state is treated as a **single, indivisible entity** (like a name or ID). No internal structure is considered.

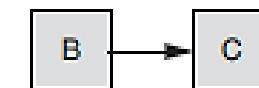
- **View:** The agent only knows **what state it is in**, not the details.

- **Example:** "I'm in Chennai," "I'm in Mumbai" — city names are treated as whole states.

Application Examples:

- Route-finding (Chennai → Delhi)
- Game playing (each board position is a separate state)
- Hidden Markov Models and Markov Decision Processes (MDPs)

Limitation: No ability to capture *why* or *how* one state differs from another.



(a) Atomic



THE STRUCTURE OF AGENTS

Learning Agents

Types of Representations of the Environment (Increasing in Complexity)

2. Factored Representation (State = Set of Variables)

• **Definition:** Each state is described as a set of **attributes or variables**, each having specific values.

• **View:** The agent knows **the structure** of the state and the values of parts of it.

• **Example:** A self-driving car's state might include:

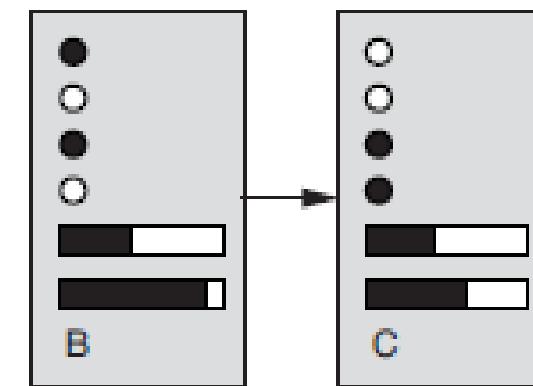
- City = Chennai
- FuelLevel = Low
- GPS = (13.08, 80.27)
- OilLight = On
- Radio = 93.5FM

Application Examples:

- Constraint satisfaction problems
- Propositional logic & planning
- Bayesian networks
- Most supervised learning algorithms

Advantage:

- Can **compare and share information** across states.
- Can **represent uncertainty** (e.g., unknown gas level).
- Easier to update and reason about changes.



(b) Factored



THE STRUCTURE OF AGENTS

Learning Agents

Types of Representations of the Environment (Increasing in Complexity)

3. Structured Representation (Objects + Relationships)

- **Definition:** The state consists of **objects**, their **properties**, and their **relationships** to each other.
- **View:** The agent sees the world as made of **things that interact**.
- **Example:** “A truck is reversing into a dairy farm driveway, but a cow is blocking it.”
 - Objects: Truck, Cow, Driveway
 - Relationships: isReversingInto, isBlocking

Application Examples:

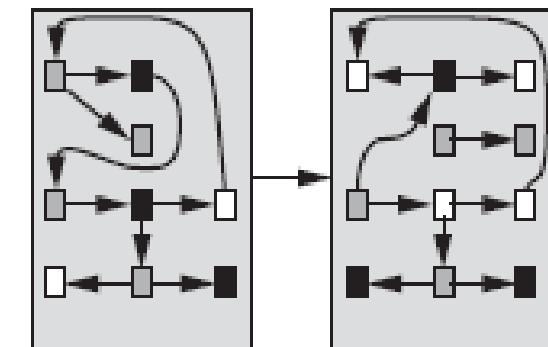
- First-order logic & relational databases
- Knowledge representation and reasoning
- Natural language understanding
- Ontology-based systems

Advantage:

- Captures **complex, realistic scenarios**
- Can represent **interacting entities**, just like in human language

Limitation:

- More **expressive** → more **computationally expensive**
- More **difficult to learn and reason**



(b) Structured



THE STRUCTURE OF AGENTS

Learning Agents

Types of Representations of the Environment (Increasing in Complexity)

Type	Structure	Example	Pros	Used In
Atomic	Single identifier	"In Chennai"	Simple, fast	Search, MDPs, HMMs
Factored	Variable set	City=Chennai, Fuel=Low	Can share info, handle uncertainty	Logic, planning, ML
Structured	Objects + relations	Cow blocks truck reversing	Very expressive, realistic	Logic, NLP, knowledge systems

THE STRUCTURE OF AGENTS



Learning Agents

Types of Representations of the Environment (Increasing in Complexity)

Expressiveness vs Simplicity

- **Atomic → Factored → Structured:** More expressive and realistic, but more complex to reason with.
- Agents in the real world may need to **combine all three:**
 - Use atomic for fast decisions.
 - Use factored for general reasoning.
 - Use structured for deep understanding (e.g., language, knowledge).



LOGICAL AGENTS

What are Knowledge-Based Agents?

- These are **intelligent agents** that:
 - Represent knowledge about the world.
 - Use **reasoning** to **infer new knowledge**.
 - Use that knowledge to **decide actions**.

In short: They know things, they think, and they act **based on what they know and infer**.

Why Is This Important?

- **Humans don't act on reflex alone**—we:
 - Use logic.
 - Combine knowledge.
 - Deduce consequences before acting.

Example: If you see dark clouds and know it usually rains afterward, you take an umbrella. That's reasoning.



LOGICAL AGENTS

Problem with Basic Agents (e.g., Problem-Solving Agents)

- In earlier AI models:
 - **Knowledge was hard-coded**, like in an 8-puzzle agent.
 - Agents knew **what actions do**, but couldn't reason or generalize.
 - No ability to infer:
 - That two tiles can't occupy the same space.
 - Or that certain configurations are unreachable.

The environment was treated as **atomic black-box states**, limiting reasoning and flexibility.

Limitation in Atomic Representations

- In **partially observable environments**, listing all possible states isn't practical.
- No understanding of internal structure or rules of the world.

For example, if a robot doesn't see everything, it can't guess intelligently what's behind a wall.

• Improved: Variable-Based (Factored) Representation

- Using **variables** and **values** is a step forward:
 - Makes reasoning more efficient.
 - Allows **domain-independent** algorithms.
 - Example: RoomClean = True, DoorOpen = False

But still limited unless you combine this with logic and inference.



LOGICAL AGENTS

Logic & Reasoning

- Logic allows the agent to:
 - **Represent complex relationships**
 - **Infer new facts**
 - **Deduce actions** even in uncertain or changing environments

Example: If the agent knows “If there’s a breeze, there’s a pit nearby” and it senses a breeze, it can infer danger.

Benefits of Knowledge-Based Agents

Capability	Description	Example
Goal Flexibility	Can accept new tasks/goals	E.g., told to clean a new type of room
Quick Learning	Can be told new facts	“Room 3 has a pit” → adapts instantly
Adaptation	Updates knowledge when environment changes	Learns a blocked door is now open

LOGICAL AGENTS

Knowledge-Based Agents (KBA)

1. Central Component: Knowledge Base (KB)

- The **knowledge base (KB)** is the **heart of a knowledge-based agent**.
- It is a **collection of sentences** that describe facts, rules, or assumptions about the world.
 - These sentences are written in a **knowledge representation language** (like logic).
 - Some sentences are called **axioms** – they are accepted as true without proof.

Example: A sentence like “If it is raining, the road is wet” is part of the KB.

2. Operations: TELL and ASK

- **TELL:** Adds a sentence (knowledge) to the KB.
 - *“It is raining at 9 AM” → added to KB.*
- **ASK:** Queries the KB to infer or retrieve information.
 - *“Is the road wet at 9 AM?” → KB uses reasoning to answer.*

These may use **inference**, i.e., deriving new facts from existing knowledge (without making things up!).



LOGICAL AGENTS

Knowledge-Based Agents (KBA)

3. How the Knowledge-Based Agent Works (3 Steps)

Every time the agent receives a percept (input), it follows this process:

Step 1: Perceive and TELL

- Converts the percept into a sentence using MAKE-PERCEPT-SENTENCE and adds it to KB via TELL.

Step 2: ASK what action to take

- Queries the KB using MAKE-ACTION-QUERY to decide on the best action.
- KB uses reasoning to deduce the best course of action.

Step 3: Act and TELL

- Executes the action and records it in the KB using MAKE-ACTION-SENTENCE.

These steps allow the agent to update and reason with knowledge **over time**, maintaining internal state and improving behavior.

LOGICAL AGENTS

Knowledge-Based Agents (KBA)

function KB-AGENT(*percept*) **returns** an *action*

persistent: *KB*, a knowledge base

t, a counter, initially 0, indicating time

TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))

action \leftarrow ASK(*KB*, MAKE-ACTION-QUERY(*t*))

TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))

t \leftarrow *t* + 1

return *action*



LOGICAL AGENTS

Knowledge-Based Agents (KBA)

4. Knowledge vs. Implementation Level

- The **Knowledge Level** describes *what the agent knows and what it wants* (its goal).
 - Example: The agent knows the Golden Gate Bridge connects San Francisco and Marin County.
 - Based on this, it decides to use the bridge to reach the goal.
- The **Implementation Level** refers to *how it works behind the scenes* (code, data structures, hardware).
 - Doesn't affect the reasoning process.

Whether the taxi's map is stored as an array or neural net doesn't matter — if it *knows* the bridge connects the locations, it acts accordingly.



LOGICAL AGENTS

Knowledge-Based Agents (KBA)

5. Design Approaches: Declarative vs. Procedural

Approach	Description	Example
Declarative	Agent is told facts and rules in a logical form (sentences).	"Cows can't climb stairs." "Left is west of right."
Procedural	Agent is coded with explicit behavior rules (like writing a function).	if (trafficLight == "red") then stop();

In modern AI, agents often use **both**:

- Learn or store **knowledge declaratively**
- Execute behavior **procedurally** for efficiency



LOGICAL AGENTS

Knowledge-Based Agents (KBA)

6. Learning Capability

- KB agents can also be designed to **learn automatically** from experience:
 - Observing patterns in percepts.
 - Generalizing them into knowledge.

A learning KBA can become **fully autonomous** – no need to TELL it everything manually.

Step	Action
1. Perceive	Use MAKE-PERCEPT-SENTENCE, then TELL the KB
2. Reason	Use MAKE-ACTION-QUERY, then ASK KB for the best action
3. Execute	Perform the action, and use MAKE-ACTION-SENTENCE to TELL KB



LOGICAL AGENTS

Knowledge-Based Agents (KBA)

Key Benefits of Knowledge-Based Agents

- **Flexible:** Can accept new goals or tasks.
- **Reusable:** Can be used in different domains by changing the KB.
- **Transparent:** Easier to explain and debug (unlike black-box systems).
- **Upgradeable:** Can learn new knowledge or be TELLED manually.

A **knowledge-based agent** is not just a program—it's a system that **reasons** about the world using **what it knows**, and adapts its actions accordingly. It acts **intelligently** because it can **combine old knowledge with new information** and **make logical decisions**.

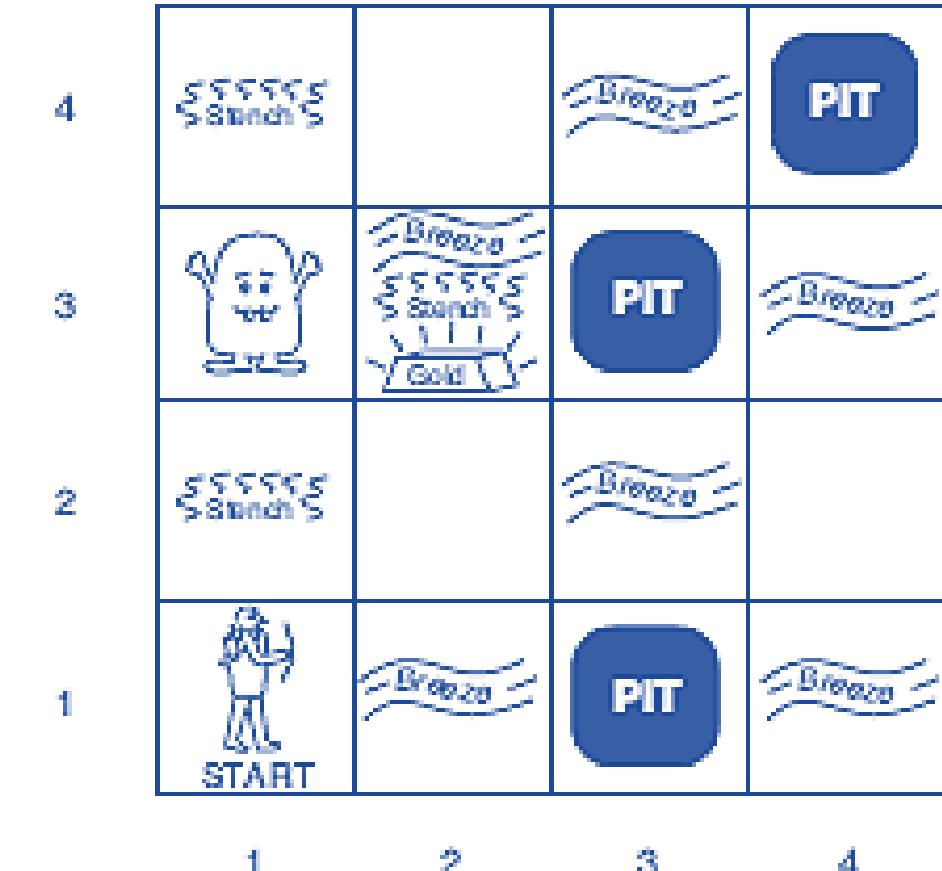


LOGICAL AGENTS

Knowledge-Based Agents (KBA)

THE WUMPUS WORLD

- It's a **grid-like cave** (typically 4×4) where each square represents a room.
- Some rooms contain **bottomless pits** — agents die if they enter.
- One room contains the **Wumpus** — a monster that **eats** agents if they enter its square.
- One room contains a **heap of gold** .
- The agent's job is to **find the gold and climb out alive**.



LOGICAL AGENTS

Knowledge-Based Agents (KBA)



THE WUMPUS WORLD

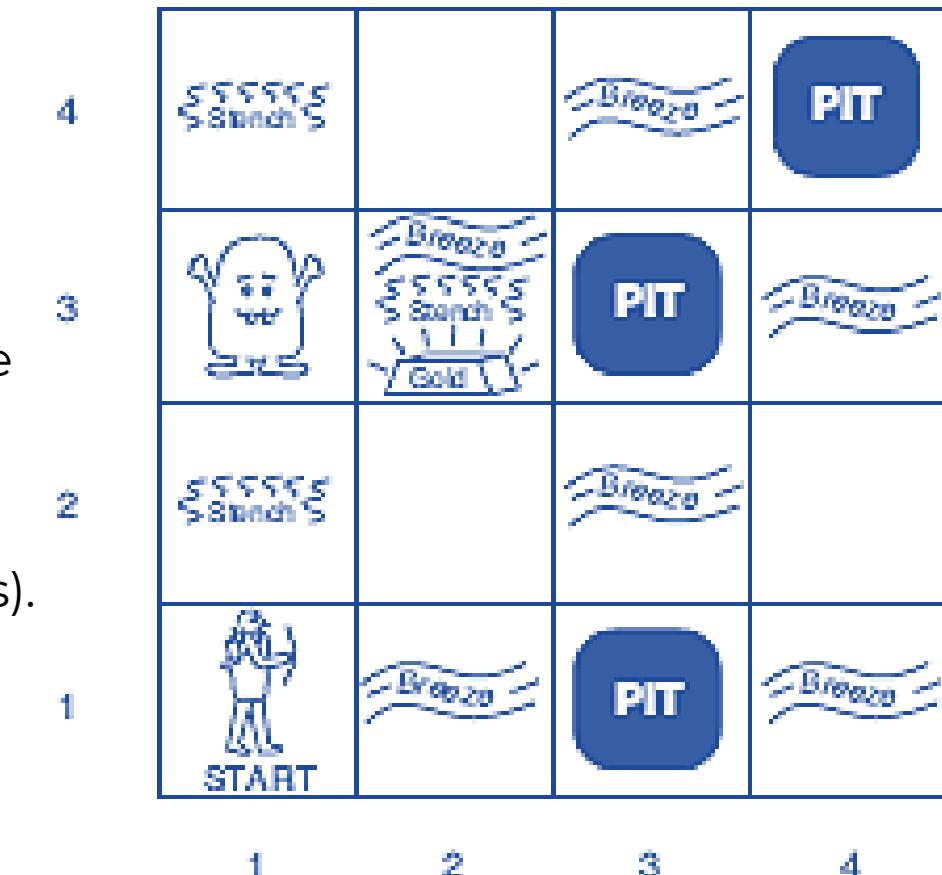
2. The Agent's Capabilities

Actuators (What the agent can do):

- Forward: Move ahead one square.
- TurnLeft / TurnRight: Rotate 90°.
- Grab: Pick up gold if in the same square.
- Shoot: Fire an arrow in a straight line (only once).
- Climb: Exit the cave (only from starting square [1,1]).

Sensors (What the agent can perceive):

- **Stench**: If near the wumpus (adjacent squares).
- **Breeze**: If near a pit.
- **Glitter**: If gold is in the current square.
- **Bump**: If agent hits a wall.
- **Scream**: If the wumpus is killed.



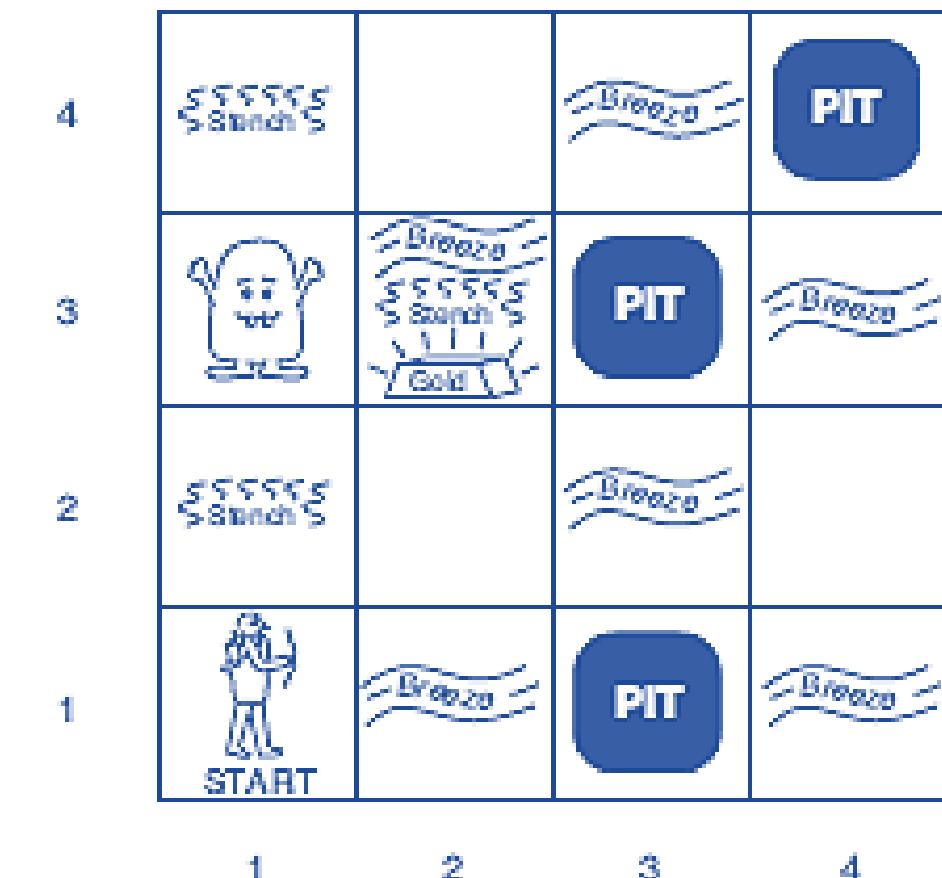
LOGICAL AGENTS

Knowledge-Based Agents (KBA)

THE WUMPUS WORLD

3. PEAS Description of the Task Environment

Component	Description
Performance	+1000 for grabbing gold and climbing out, -1000 for death, -10 for shooting, -1 per move
Environment	4×4 grid, random placement of pits (20% chance), wumpus, and gold
Actuators	Move, turn, grab, shoot, climb
Sensors	Perceive breeze, stench, glitter, bump, scream



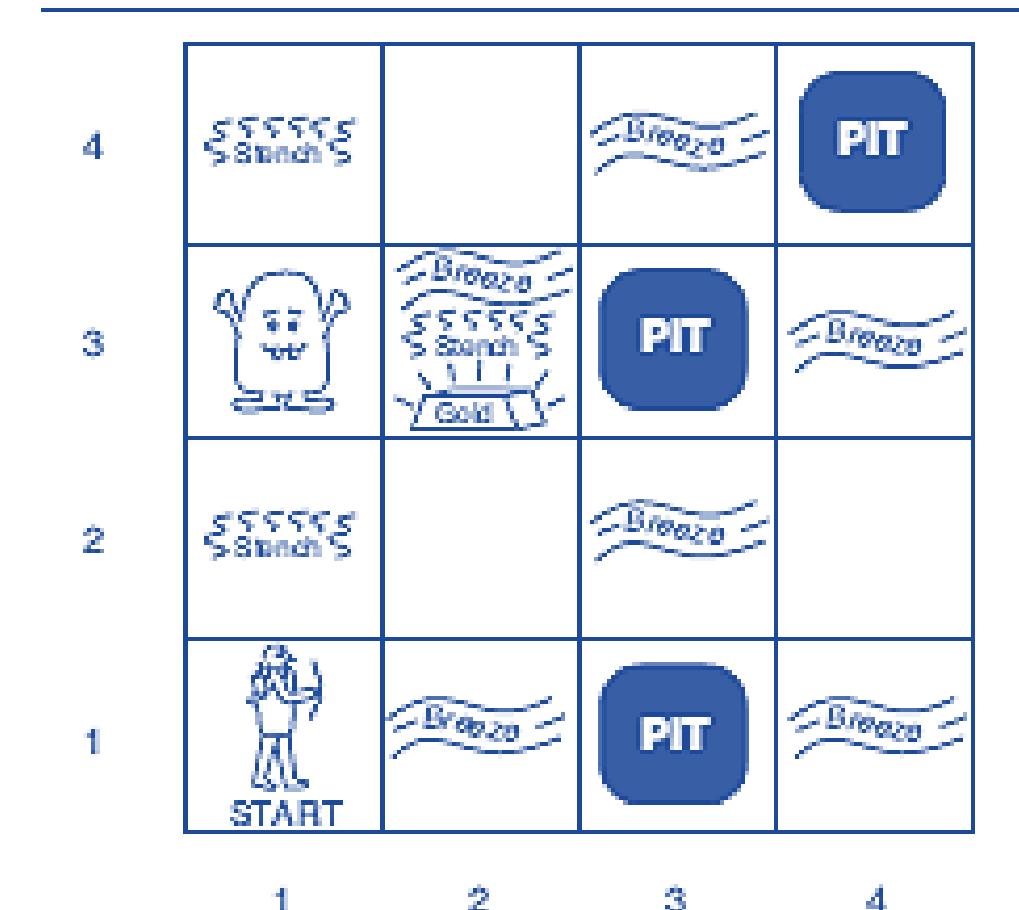
LOGICAL AGENTS

Knowledge-Based Agents (KBA)

THE WUMPUS WORLD

4. Characteristics of the Wumpus World

Feature	Value
Observable?	Partially observable (only get local percepts)
Deterministic?	Yes (actions have predictable effects)
Static?	Yes (environment doesn't change on its own)
Discrete?	Yes (finite number of states and actions)
Single-agent?	Yes (no competing entities except environment hazards)



LOGICAL AGENTS

Knowledge-Based Agents (KBA)

THE WUMPUS WORLD

5. Logical Reasoning in the Wumpus World

The agent uses a **knowledge base (KB)** to track:

- What is safe (OK),
- Where the Wumpus or pits might be,
- Where the gold is,
- Which squares it has visited.

Logical Inference Steps

1. Initial Percept at [1,1]:

- [None, None, None, None, None]
- \Rightarrow No breeze or stench \rightarrow [1,2] and [2,1] must be **safe (OK)**.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1	A	2,1	3,1
OK	OK		4,1

(a)

The first step taken by the agent in the wumpus world. (a) The initial situation, after percept [None, None, None, None, None].



LOGICAL AGENTS

Knowledge-Based Agents (KBA)

THE WUMPUS WORLD

5. Logical Reasoning in the Wumpus World

Logical Inference Steps

2. Move to [2,1]:

- Percept: Breeze
- \Rightarrow A pit is in [2,2] or [3,1] (not [1,1], it's known to be safe).

 = Agent
 B = Breeze
 G = Glitter, Gold
 OK = Safe square
 P = Pit
 S = Stench
 V = Visited
 W = Wumpus

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 P?	3,2	4,2
OK			
1,1 V OK	2,1 A B OK	3,1 P?	4,1

(b)

(b) After one move, with percept [None, Breeze, None, None, None].

LOGICAL AGENTS



Knowledge-Based Agents (KBA)

THE WUMPUS WORLD

5. Logical Reasoning in the Wumpus World

3. Backtrack to [1,1], then go to [1,2]:

- Percept: Stench
- \Rightarrow Wumpus must be adjacent \rightarrow Not in [1,1] or [2,2]
- \Rightarrow Must be in [1,3]

4. No breeze in [1,2]:

- \Rightarrow [2,2] is **not a pit**
- Since the pit is either [2,2] or [3,1] \rightarrow must be [3,1]

This reasoning uses **negative information** (lack of a percept) and **cross-checks** between earlier observations — a hallmark of logical intelligence.

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

(a) After the third move,
with percept [Stench, None, None, None, None].



LOGICAL AGENTS

Knowledge-Based Agents (KBA)

THE WUMPUS WORLD

5. Logical Reasoning in the Wumpus World

5. Move to [2,2], then [2,3]:

- Percept: Glitter
- ⇒ Gold found! Use Grab, then retrace steps to [1,1], then Climb.

A	= Agent
B	= Breeze
G	= Glitter, Gold
OK	= Safe square
P	= Pit
S	= Stench
V	= Visited
W	= Wumpus

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

(b) After the fifth move, with percept [Stench, Breeze,Glitter, None, None].



LOGICAL AGENTS

Knowledge-Based Agents (KBA)

THE WUMPUS WORLD

6. Why This Environment Is Ideal for Knowledge-Based Agents

- **Logical reasoning** is essential to survive — you can't afford random exploration.
- **Percepts are partial and noisy**, so agents must **infer hidden dangers**.
- Actions are **costly and limited**, so **planning is critical**.
- Knowledge gained must be **updated over time** (belief revision).

7. Challenges for the Agent

- Starts with **no knowledge of the environment layout**.
- Must **deduce** where hazards are **without dying**.
- Sometimes must take **calculated risks**.
 - ~21% of randomly generated caves are **unfair** (gold unreachable or fatally surrounded).
- Must **remember** and **reason over time**, combining past and present perceptions.



LOGICAL AGENTS

Knowledge-Based Agents (KBA)

LOGIC

1. Syntax vs. Semantics

► Syntax

Defines **how a sentence should be written** (the grammar or structure of a language). It tells us what combinations of symbols are **valid**.

• Example: $x + y = 4$ is **well-formed**.

Example: $x4y+=$ is **not well-formed** (violates syntax rules).

► Semantics

Deals with the **meaning** or **truth** of a sentence in a given **world or situation**.

• Example:

In a world where $x = 2$ and $y = 2$, the sentence $x + y = 4$ is **true**.

But if $x = 1$ and $y = 1$, the sentence is **false**.

LOGICAL AGENTS

Knowledge-Based Agents (KBA)

LOGIC

2. Model and Possible World

• A **model** represents a **mathematical version** of a possible world — it assigns values to variables and tells us whether a sentence is **true or false** in that setup.

• *Example:*

Model m1: $x = 2, y = 2 \Rightarrow x + y = 4$ is **true**

Model m2: $x = 1, y = 1 \Rightarrow x + y = 4$ is **false**

We write $m \models \alpha$ if model m satisfies sentence α .

3. Entailment (\models)

Entailment means that if sentence α is true, then sentence β must also be true in **every model**.

We write:

$\alpha \models \beta$ if in **every model** where α is true, β is also true.

• *Example:*

$x = 0 \models xy = 0$ — because when $x = 0$, no matter what y is, $xy = 0$.

• **Set notation:**

$\alpha \models \beta \Leftrightarrow M(\alpha) \subseteq M(\beta)$ — the set of models of α is a **subset** of models of β .

LOGICAL AGENTS

Knowledge-Based Agents (KBA)

LOGIC



4. Inference (\vdash)

- **Inference** is the **mechanical/logical process** of deriving new knowledge from known facts (the knowledge base, KB).
- If an inference procedure i derives sentence α from KB, we write:
$$\text{KB} \vdash_i \alpha$$

5. Soundness and Completeness

Soundness

- An inference method is **sound** if **only true conclusions** are derived.
- That is, if $\text{KB} \vdash_i \alpha$, then $\text{KB} \vDash \alpha$
- **Analogy:** You never find a needle that **isn't really there**.

Completeness

- An inference method is **complete** if it can **derive all true conclusions**.
- That is, if $\text{KB} \vDash \alpha$, then $\text{KB} \vdash_i \alpha$
- **Analogy:** You are guaranteed to find every needle that **actually exists** in the haystack.

LOGICAL AGENTS

Knowledge-Based Agents (KBA)

LOGIC

6. Model Checking (Inference Method)

- **Model checking** checks **all possible models** to see whether the conclusion α is true in all models where KB is true.
- Works well if the number of models is **finite**.



LOGICAL AGENTS

Knowledge-Based Agents (KBA)

LOGIC

Example: Wumpus World Reasoning

Let's apply these ideas to the **Wumpus World**:

Situation:

- The agent starts at [1,1] and **feels no breeze**.
- It moves to [2,1] and **feels a breeze**.

From this, it forms the **knowledge base (KB)**:

- Breeze at [2,1]
- No breeze at [1,1]

Questions:

1. Can the agent **conclude** there is **no pit** in [1,2]? → **Yes**
2. Can the agent **conclude** there is **no pit** in [2,2]? → **No**

Why?

There are 3 squares near [2,1] that could cause a breeze: [1,2], [2,2], and [3,1] → $2^3 = 8$ models possible (each may or may not have a pit).

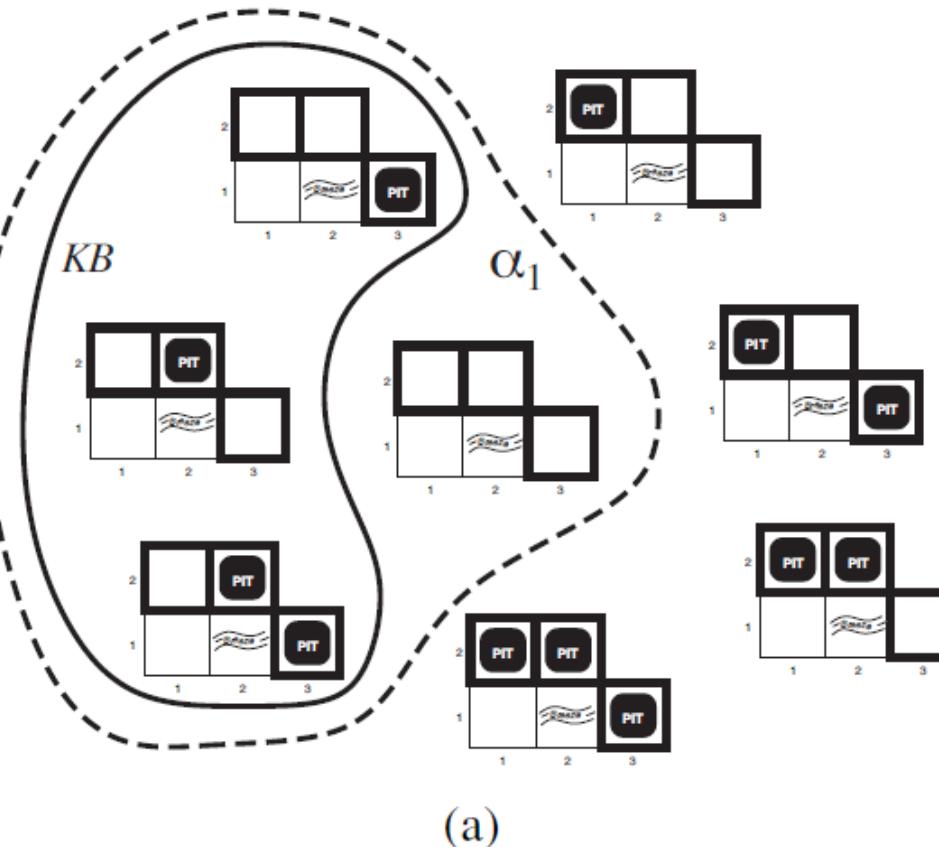
- In **all models where KB is true**, [1,2] **never** has a pit → so $\text{KB} \models \alpha_1$ (safe)
- In **some models**, [2,2] has a pit, and in others it doesn't → so $\text{KB} \not\models \alpha_2$ (uncertain)

LOGICAL AGENTS

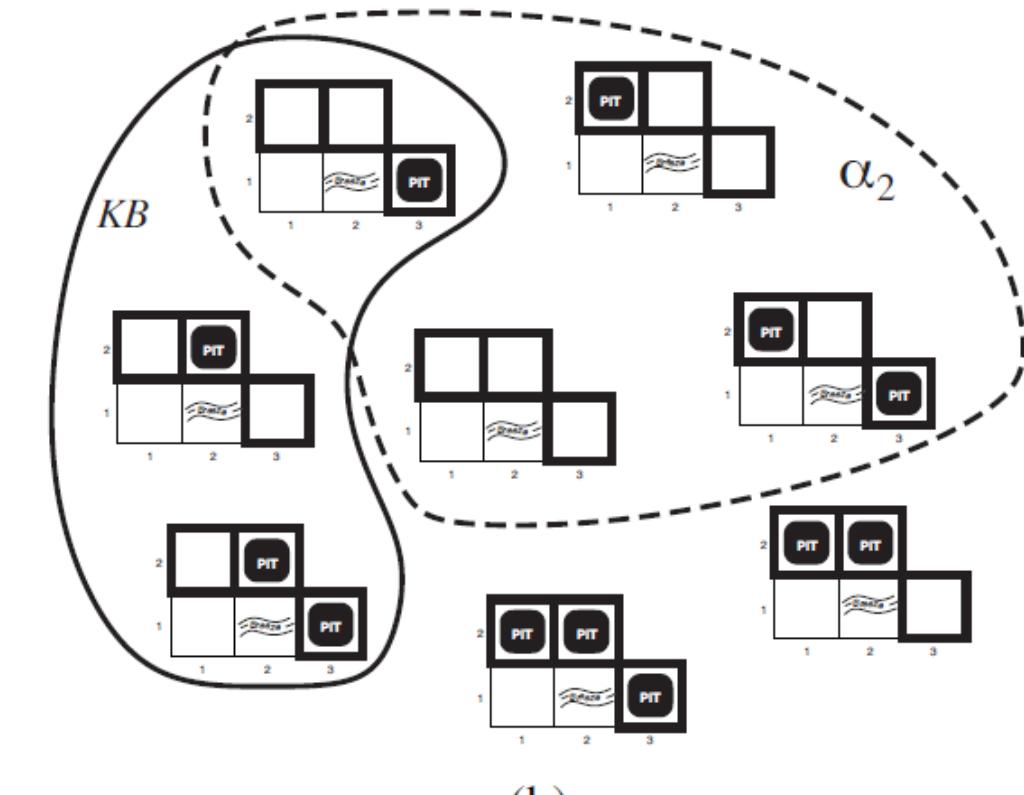
Knowledge-Based Agents (KBA)

LOGIC

Possible models for the presence of pits in squares [1,2], [2,2], and [3,1]. The KB corresponding to the observations of nothing in [1,1] and a breeze in [2,1] is shown by the solid line. (a) Dotted line shows models of α_1 (no pit in [1,2]). (b) Dotted line shows models of α_2 (no pit in [2,2]).



(a)



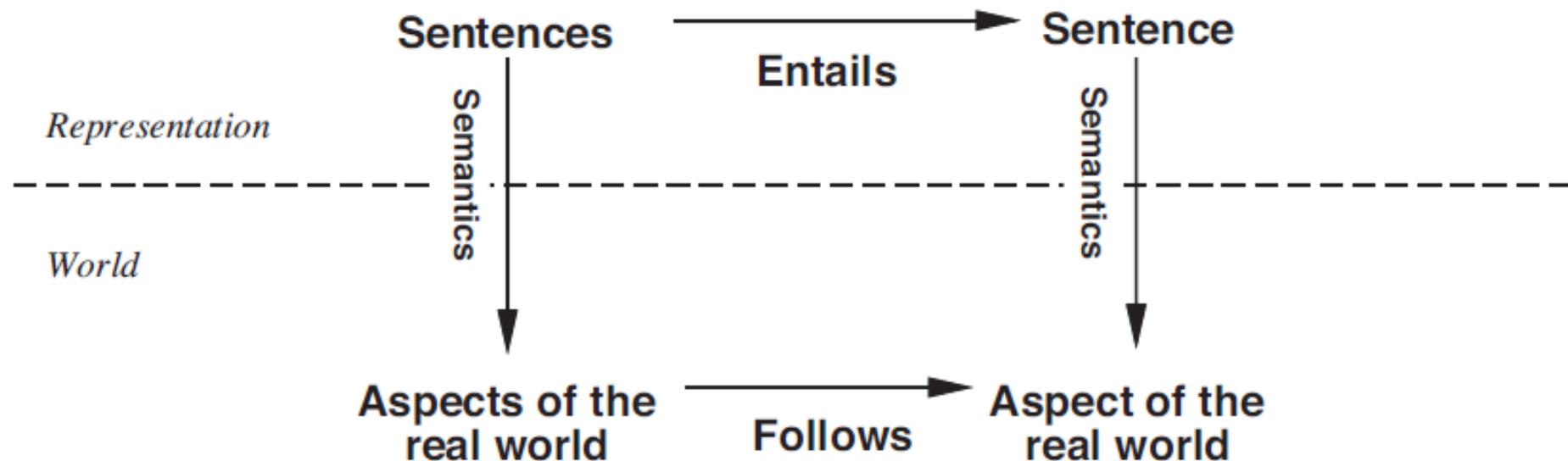
(b)

LOGICAL AGENTS

Knowledge-Based Agents (KBA)

LOGIC

Sentences are physical configurations of the agent, and reasoning is a process of constructing new physical configurations from old ones. Logical reasoning should ensure that the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent.



LOGICAL AGENTS

Knowledge-Based Agents (KBA)

LOGIC

8. Grounding (Link to the Real World)

- A **logical sentence** in the agent's KB is just a symbol (syntax).
- How does it get its **meaning**?
 - Through **sensors** and **experience**.

Example:

- A sensor detects a smell → agent creates a sentence “Smell at [2,1]”
- This sentence becomes part of the KB and is **true** in the real world.

But what about general rules like:

“If there’s a breeze, there might be a pit nearby”?

- These are **learned** from **experience**, not directly from a sensor.
- Learning might not be 100% accurate (e.g., “Wumpuses don’t cause breezes on leap years” might be a missing exception!).



LOGICAL AGENTS

Knowledge-Based Agents (KBA)

LOGIC

Concept	Meaning	Example
Syntax	Structure or format of sentences	$x + y = 4$ is well-formed
Semantics	Meaning or truth of sentences in a world	$x + y = 4$ is true if $x=2, y=2$
Model	A specific assignment of values	$m: x=2, y=2$
Entailment (\models)	Sentence β is true in every model where α is true	$x=0 \models xy=0$
Inference (\vdash)	Deriving β from KB using an algorithm	$KB \vdash i \alpha$
Soundness	Only deriving what is actually true	Never wrong
Completeness	Able to derive everything that is true	Never misses
Model Checking	Brute-force method of verifying entailment	Works if finite models
Grounding	How syntax connects to real-world truth	Smell sensor → "Smell" sentence

LOGICAL AGENTS

Knowledge-Based Agents (KBA)

PROPOSITIONAL LOGIC

Propositional logic is a simple, formal system used in **logical reasoning**, especially useful in rule-based AI systems like the **Wumpus World agent**.

1. Syntax of Propositional Logic

Syntax defines the **structure or format** of valid sentences.

► Atomic Sentences (Propositions)

- The **basic building blocks** of propositional logic.
- Each **atomic sentence** is a **proposition symbol** that can be **true or false**.
- Symbols usually start with **capital letters**: P, Q, R, etc.
- Can also use subscripts for clarity:
 - W1,3 = “There is a **Wumpus** in cell [1,3]”
 - B2,1 = “There is a **breeze** in cell [2,1]”

Think of each symbol like a **light switch** — it's either ON (true) or OFF (false).

- Special symbols:
 - True — always true.
 - False — always false.



LOGICAL AGENTS

PROPOSITIONAL LOGIC

2. Complex Sentences

These are built using **logical connectives**.

Common Logical Connectives:

Symbol	Name	Example	Meaning
\neg	Negation	$\neg W1,3$	"Not Wumpus in [1,3]"
\wedge	Conjunction	$W1,3 \wedge P3,1$	Both Wumpus in [1,3] AND Pit in [3,1]
\vee	Disjunction	$(W1,3 \wedge P3,1) \vee W2,2$	Either both W1,3 and P3,1 OR Wumpus in [2,2]
\Rightarrow	Implication	$(B2,1 \wedge \neg W2,2) \Rightarrow W1,3$	If breeze and no Wumpus in [2,2], then W1,3
\Leftrightarrow	Biconditional	$W1,3 \Leftrightarrow \neg W2,2$	Wumpus in [1,3] if and only if no Wumpus in [2,2]



LOGICAL AGENTS

PROPOSITIONAL LOGIC

2. Complex Sentences

Terminology:

• **Literals**: atomic or negated atomic sentences

- $W_1, W_3 \rightarrow \text{positive literal}$

- $\neg W_1, W_3 \rightarrow \text{negative literal}$

• **Conjuncts**: the parts of a conjunction

• **Disjuncts**: the parts of a disjunction

• **Premise & Conclusion**: in implications, the “if” part is the **premise**, and the “then” part is the **conclusion**



LOGICAL AGENTS

PROPOSITIONAL LOGIC

A **BNF (Backus–Naur Form)** grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

3. Sentence Construction (Grammar)

The grammar (in **BNF format**) tells us how to **form sentences** in propositional logic.

Sentence → AtomicSentence | ComplexSentence

AtomicSentence → True | False | P | Q | R | ...

ComplexSentence → (Sentence)
| [Sentence]
| \neg Sentence
| Sentence \wedge Sentence
| Sentence \vee Sentence
| Sentence \Rightarrow Sentence
| Sentence \Leftrightarrow Sentence



LOGICAL AGENTS

PROPOSITIONAL LOGIC

4. Operator Precedence (Order of Evaluation)

To avoid ambiguity when multiple operators are used, we follow **precedence rules**:

1. \neg (Not)
2. \wedge (And)
3. \vee (Or)
4. \Rightarrow (Implies)
5. \Leftrightarrow (If and only if)

Example:

$\neg A \wedge B$ means $(\neg A) \wedge B$, **not** $\neg(A \wedge B)$

If unsure, always **use parentheses** to clarify grouping.



LOGICAL AGENTS

PROPOSITIONAL LOGIC

5. Application in the Wumpus World

Let's say the agent is in [2,1] and feels a breeze. Based on the Wumpus World rules:

- Rule: If there's a **breeze in a cell**, then **at least one adjacent cell has a pit**.
- This can be written in propositional logic as:

$$B_{2,1} \Rightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

Which means:

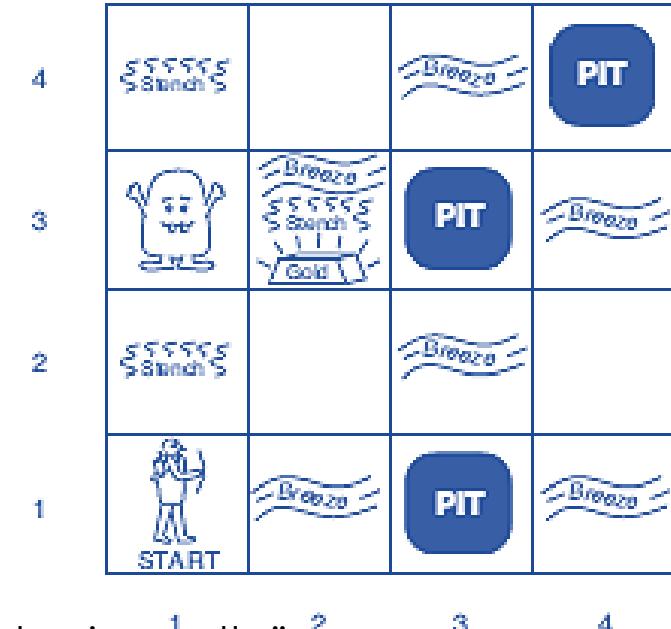
"If there is a breeze at [2,1], then there is a pit in one of the neighboring cells."

Similarly, if the agent **doesn't feel** a breeze in [1,1], then:

$$\neg B_{1,1} \Rightarrow (\neg P_{1,2} \wedge \neg P_{2,1})$$

"If there is no breeze at [1,1], then there are no pits in adjacent cells [1,2] and [2,1]."

These logical expressions help the agent deduce safe or dangerous squares.





LOGICAL AGENTS

PROPOSITIONAL LOGIC

Semantics of Propositional Logic

- **Semantics** defines **what sentences mean**—specifically, **how to determine if a sentence is true or false** in a given situation.
- This situation is called a **model**.

Model (m): A Complete Assignment of Truth Values

A **model** is a complete specification of the **truth values** for all proposition symbols used in a given context.

Example:

Let's say your knowledge base includes three propositions: P1,2, P2,2, and P3,1. Then a possible model, m_1 , could be:

$$m_1 = \{P1,2 = \text{false}, P2,2 = \text{false}, P3,1 = \text{true}\}$$

• This means:

- There is **no pit** in [1,2],
- No pit in [2,2],
- There **is a pit** in [3,1].

With 3 proposition symbols, there are $2^3 = 8$ possible models.



LOGICAL AGENTS

PROPOSITIONAL LOGIC

Evaluating Truth Values in a Model

All sentences are made of:

- **Atomic sentences** (like P_{1,2})
- And **connectives** (like \wedge , \neg , \Rightarrow , etc.)

We need rules for evaluating both.

1. Atomic Sentences:

- True is always **true** in every model.
- False is always **false** in every model.
- Other symbols like P_{1,2} get their value **directly from the model**.

Example:

If $m_1 = \{P_{1,2} = \text{false}\}$, then P_{1,2} is **false** in m_1 .

2. Complex Sentences:

Use **recursive rules** to evaluate based on truth values of subparts:

Sentence	Rule (in model m)
$\neg P$	true iff P is false
$P \wedge Q$	true iff P is true AND Q is true
$P \vee Q$	true iff P is true , Q is true , or both
$P \Rightarrow Q$	false only when P is true and Q is false
$P \Leftrightarrow Q$	true iff P and Q have same truth values

LOGICAL AGENTS

PROPOSITIONAL LOGIC



P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

Example:

Evaluate $\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1})$ in model $m_1 = \{P_{1,2} = \text{false}, P_{2,2} = \text{false}, P_{3,1} = \text{true}\}$

Step-by-step:

1. $\neg P_{1,2} \rightarrow \neg \text{false} \rightarrow \text{true}$
2. $P_{2,2} \vee P_{3,1} \rightarrow \text{false} \vee \text{true} \rightarrow \text{true}$
3. $\text{true} \wedge \text{true} \rightarrow \text{true}$

Final result: **true**



LOGICAL AGENTS

PROPOSITIONAL LOGIC

Common Confusions

1. Implication (\Rightarrow) feels weird in English

- In logic:

$P \Rightarrow Q$ is **true unless** P is **true** and Q is **false**

- Example:

"If 5 is even \Rightarrow Sam is smart"

This is **true**—because 5 is **not even**, so the implication **holds vacuously**.

Why?

Think of it like a **promise**:

"If I come to class (P), I'll bring candy (Q)"

- If I don't come, I didn't break the promise.



LOGICAL AGENTS

PROPOSITIONAL LOGIC

2. Inclusive OR (\vee) vs Exclusive OR (\oplus)

- In propositional logic:

$P \vee Q$ is **true** if **either or both** are true
(i.e., inclusive OR)

- Exclusive OR:

True **only** if **exactly one** is true

Not standard in propositional logic but symbolized as \oplus , \vee , etc.

Biconditional (\Leftrightarrow) = “if and only if”

- $P \Leftrightarrow Q$ is true if:

- both are true, or
- both are false

Example in Wumpus World:

A square is breezy **if and only if** a neighboring square has a pit.

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$



LOGICAL AGENTS

PROPOSITIONAL LOGIC

Concept	Meaning
Model (m)	Assigns true/false to every proposition symbol
Semantics	Rules for determining sentence truth under a model
$\neg P$	True if P is false
$P \wedge Q$	True if both are true
$P \vee Q$	True if either/both are true
$P \Rightarrow Q$	False only if P is true and Q is false
$P \Leftrightarrow Q$	True if P and Q are both true or both false
Truth Tables	Used to compute sentence values systematically
Exclusive OR	Not standard in logic; true only if one is true, not both



LOGICAL AGENTS

PROPOSITIONAL LOGIC

Knowledge Base in Propositional Logic?

A **knowledge base (KB)** is just a set of propositional logic sentences that describe facts and rules about the world—in this case, the **wumpus world**.

Each sentence in the KB has a **truth value**—either **true** or **false**—depending on a particular **model** (an assignment of truth values to proposition symbols).

Symbols Used in the Wumpus World KB

The symbols represent what's happening at each location $[x,y]$:
 $[x, y]$:

- $P_{x,y}$ → there's a **pit** at $[x,y]$
- $W_{x,y}$ → there's a **wumpus** at $[x,y]$
- $B_{x,y}$ → the agent **perceives a breeze** at $[x,y]$
- $S_{x,y}$ → the agent **perceives a stench** at $[x,y]$

Example:

- $P_{1,2}$ is true \Rightarrow there's a pit at square $[1,2]$
- $\neg P_{1,1}$ \Rightarrow there's **no** pit at square $[1,1]$

LOGICAL AGENTS

PROPOSITIONAL LOGIC



Knowledge Base Sentences (Immutable Facts)

Each sentence is labeled (R1 to R5) to refer to them easily.

1.R1: $\neg P1,1$

There is **no pit** at [1,1][1,1][1,1].

2.R2: $B1,1 \Leftrightarrow (P1,2 \vee P2,1)$

Square [1,1][1,1][1,1] is **breezy** if and only if there's a pit in one of its neighboring squares.

3.R3: $B2,1 \Leftrightarrow (P1,1 \vee P2,2 \vee P3,1)$

Same logic for square [2,1][2,1][2,1].

4.R4: $\neg B1,1$

The agent **does not feel a breeze** at [1,1][1,1][1,1].

5.R5: $B2,1$

The agent **does feel a breeze** at [2,1][2,1][2,1].



LOGICAL AGENTS

PROPOSITIONAL LOGIC

What Does Entailment Mean Here?

We want to check if the KB **logically entails** a new sentence. That is:

Does $\text{KB} \models \alpha$ hold?

Meaning: In **every model** where KB is true, is α also true?

Example:

- Does the KB entail $\neg P1,2$ (no pit at [1,2][1,2][1,2])?

Truth Table Enumeration

To answer this, we **check all possible models** (truth assignments).

Each proposition symbol (e.g., $P1,1$, $P1,2$, etc.) can be **true or false**, so with 7 symbols there are $2^7 = 128$ possible models.

From the KB, we find that **only 3 models** make all the sentences (R1 to R5) true.

Then we look:

- In all these 3 models, is $\neg P1,2$ true?
 - YES \rightarrow So $\text{KB} \models \neg P1,2$
 - What about $P2,2$?
NO \rightarrow It's true in some models, false in others \rightarrow not entailed.

LOGICAL AGENTS

PROPOSITIONAL LOGIC



function TT-ENTAILS?(*KB*, α) **returns** true or false

inputs: *KB*, the knowledge base, a sentence in propositional logic
 α , the query, a sentence in propositional logic

symbols \leftarrow a list of the proposition symbols in *KB* and α
return TT-CHECK-ALL(*KB*, α , *symbols*, { })

function TT-CHECK-ALL(*KB*, α , *symbols*, *model*) **returns** true or false

if EMPTY?(*symbols*) **then**
if PL-TRUE?(*KB*, *model*) **then return** PL-TRUE?(α , *model*)
else return true // when *KB* is false, always return true
else do
 $P \leftarrow \text{FIRST}(\text{symbols})$
 $rest \leftarrow \text{REST}(\text{symbols})$
return (TT-CHECK-ALL(*KB*, α , *rest*, *model* \cup { $P = \text{true}$ })
and
TT-CHECK-ALL(*KB*, α , *rest*, *model* \cup { $P = \text{false}$ }))

This is a **model-checking algorithm**:

It returns true if α is **true in all models** where *KB* is true.

Here's how it works:

1. Enumerate **all possible models** by assigning each symbol true or false.

2. For each model:

- If *KB* is false \rightarrow skip (we only care about models where *KB* is true).
- If *KB* is true \rightarrow check if α is true.

3. If α is true in **all such models**, then **KB entails α** .



LOGICAL AGENTS

PROPOSITIONAL LOGIC

Standard logical equivalences. The symbols α , β , and γ stand for arbitrary sentences of propositional logic.

- $(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$ commutativity of \wedge
- $(\alpha \vee \beta) \equiv (\beta \vee \alpha)$ commutativity of \vee
- $((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$ associativity of \wedge
- $((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$ associativity of \vee
- $\neg(\neg\alpha) \equiv \alpha$ double-negation elimination
- $(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$ contraposition
- $(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$ implication elimination
- $(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$ biconditional elimination
- $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ De Morgan
- $\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ De Morgan
- $(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$ distributivity of \wedge over \vee
- $(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$ distributivity of \vee over \wedge



LOGICAL AGENTS

FIRST-ORDER LOGIC

1. Motivation for First-Order Logic

- **Propositional logic**, though useful, is **limited** in expressing knowledge about complex environments.
- **First-order logic (FOL)** extends propositional logic by incorporating **objects**, **relations**, and **functions**, making it far more **expressive**.

Why a Richer Language Is Needed

- **Programming languages** use data structures and procedural code, which are domain-specific and non-declarative.
- **Propositional logic** is **declarative, compositional**, and supports **partial information**, but it fails to **generalize** across objects (e.g., repeating rules for every square in the Wumpus World).
- **FOL** allows the **generalization** of rules (e.g., "All squares adjacent to pits are breezy").



LOGICAL AGENTS

FIRST-ORDER LOGIC

3. Natural Languages and Representation

• **Natural languages** (e.g., English, Spanish) are highly expressive but:

- Suffer from **ambiguity**
- Depend heavily on **context**
- Are **not suitable** for precise reasoning without extra context

Example Issues

- “Look!” requires contextual knowledge.
- Words like “spring” or “bridge” are ambiguous.
- Experiments show we remember **concepts**, not exact wordings (Wanner, 1974).

LOGICAL AGENTS

FIRST-ORDER LOGIC

4. Language Influences Thought

- The **Sapir–Whorf hypothesis** suggests language shapes thought.
- Examples:
 - **Guugu Yimithirr** uses absolute directions (north/south) instead of “left/right”.
 - Gendered nouns influence perception (Boroditsky, 2003).
 - Word choice (e.g., "contacted" vs. "smashed") changes memory of events (Loftus and Palmer, 1974).





LOGICAL AGENTS

FIRST-ORDER LOGIC

5. Bridging Formal and Natural Languages

- FOL combines:
 - **Declarative nature** of propositional logic
 - **Expressive power** of natural language

- Components of FOL:

- **Objects**: people, places, numbers (e.g., "wumpus", "three")
- **Relations**: "is breezy", "bigger than"
- **Functions**: "father of", "sum of"

Examples:

- "One plus two equals three":

- Objects: one, two, three
- Function: plus
- Relation: equals

- "Squares neighboring the wumpus are smelly":

- Objects: squares, wumpus
- Property: smelly
- Relation: neighboring



LOGICAL AGENTS

FIRST-ORDER LOGIC

Ontological vs. Epistemological Commitments

Logic	Ontological Commitment	Epistemological Commitment
Propositional Logic	Facts	True / False / Unknown
First-Order Logic	Facts, Objects, Relations	True / False / Unknown
Temporal Logic	Facts, Objects, Relations, Times	True / False / Unknown
Probability Theory	Facts	Degree of belief $\in [0, 1]$
Fuzzy Logic	Facts with degree of truth $\in [0, 1]$	Known interval value

- **Ontological commitment:** What entities the logic assumes exist in the world.
- **Epistemological commitment:** What beliefs or knowledge states the agent can have about those entities.



LOGICAL AGENTS

FIRST-ORDER LOGIC

7. Representational Efficiency

- FOL allows **more compact, general**, and **flexible** knowledge representation.
- Different representations of the same fact may affect **efficiency** of inference or learning.
- Learning systems** often prefer **succinct representations** (simpler theories).

8. Implications for AI

- FOL forms the **foundation** for representing structured knowledge.
- A student of AI must become **comfortable with logical notations**, not for their syntax but for their **semantic power and reasoning capability**.



LOGICAL AGENTS

FIRST-ORDER LOGIC

1. Motivation for First-Order Logic

- **Propositional logic**, though useful, is **limited** in expressing knowledge about complex environments.
- **First-order logic (FOL)** extends propositional logic by incorporating **objects**, **relations**, and **functions**, making it far more **expressive**.

Why a Richer Language Is Needed

- **Programming languages** use data structures and procedural code, which are domain-specific and non-declarative.
- **Propositional logic** is **declarative**, **compositional**, and supports **partial information**, but it fails to **generalize** across objects (e.g., repeating rules for every square in the Wumpus World).
- **FOL** allows the **generalization** of rules (e.g., "All squares adjacent to pits are breezy").



LOGICAL AGENTS

FIRST-ORDER LOGIC

3. Natural Languages and Representation

• **Natural languages** (e.g., English, Spanish) are highly expressive but:

- Suffer from **ambiguity**
- Depend heavily on **context**
- Are **not suitable** for precise reasoning without extra context

Example Issues

- “Look!” requires contextual knowledge.
- Words like “spring” or “bridge” are ambiguous.
- Experiments show we remember **concepts**, not exact wordings (Wanner, 1974).

LOGICAL AGENTS

FIRST-ORDER LOGIC

4. Language Influences Thought

- The **Sapir–Whorf hypothesis** suggests language shapes thought.
- Examples:
 - **Guugu Yimithirr** uses absolute directions (north/south) instead of “left/right”.
 - Gendered nouns influence perception (Boroditsky, 2003).
 - Word choice (e.g., "contacted" vs. "smashed") changes memory of events (Loftus and Palmer, 1974).





LOGICAL AGENTS

FIRST-ORDER LOGIC

5. Bridging Formal and Natural Languages

- FOL combines:
 - **Declarative nature** of propositional logic
 - **Expressive power** of natural language

- Components of FOL:

- **Objects**: people, places, numbers (e.g., "wumpus", "three")
- **Relations**: "is breezy", "bigger than"
- **Functions**: "father of", "sum of"

Examples:

- "One plus two equals three":

- Objects: one, two, three
- Function: plus
- Relation: equals

- "Squares neighboring the wumpus are smelly":

- Objects: squares, wumpus
- Property: smelly
- Relation: neighboring



LOGICAL AGENTS

FIRST-ORDER LOGIC

Ontological vs. Epistemological Commitments

Logic	Ontological Commitment	Epistemological Commitment
Propositional Logic	Facts	True / False / Unknown
First-Order Logic	Facts, Objects, Relations	True / False / Unknown
Temporal Logic	Facts, Objects, Relations, Times	True / False / Unknown
Probability Theory	Facts	Degree of belief $\in [0, 1]$
Fuzzy Logic	Facts with degree of truth $\in [0, 1]$	Known interval value

- **Ontological commitment:** What entities the logic assumes exist in the world.
- **Epistemological commitment:** What beliefs or knowledge states the agent can have about those entities.



LOGICAL AGENTS

FIRST-ORDER LOGIC

7. Representational Efficiency

- FOL allows **more compact, general**, and **flexible** knowledge representation.
- Different representations of the same fact may affect **efficiency** of inference or learning.
- Learning systems** often prefer **succinct representations** (simpler theories).

8. Implications for AI

- FOL forms the **foundation** for representing structured knowledge.
- A student of AI must become **comfortable with logical notations**, not for their syntax but for their **semantic power and reasoning capability**.



LOGICAL AGENTS

FIRST-ORDER LOGIC

Model in First-Order Logic?

A **model** is like a “possible world” that helps us understand the truth of logical sentences.

Elements in a Model:

- **Domain:** The set of all objects (e.g., people, things).

Example: {Richard, John, Richard's left leg, John's left leg, Crown}

- **Relations (Predicates):** How objects are related.

Example: Brother(Richard, John) means Richard is John's brother.

- **Functions:** Mapping of one object to another.

Example: LeftLeg(Richard) refers to Richard's left leg.

- **Interpretation:** Assigns real-world meaning to symbols.

Richard → Richard the Lionheart

King → {John}

OnHead(crown, John) is true.

- Domain **must be non-empty.**

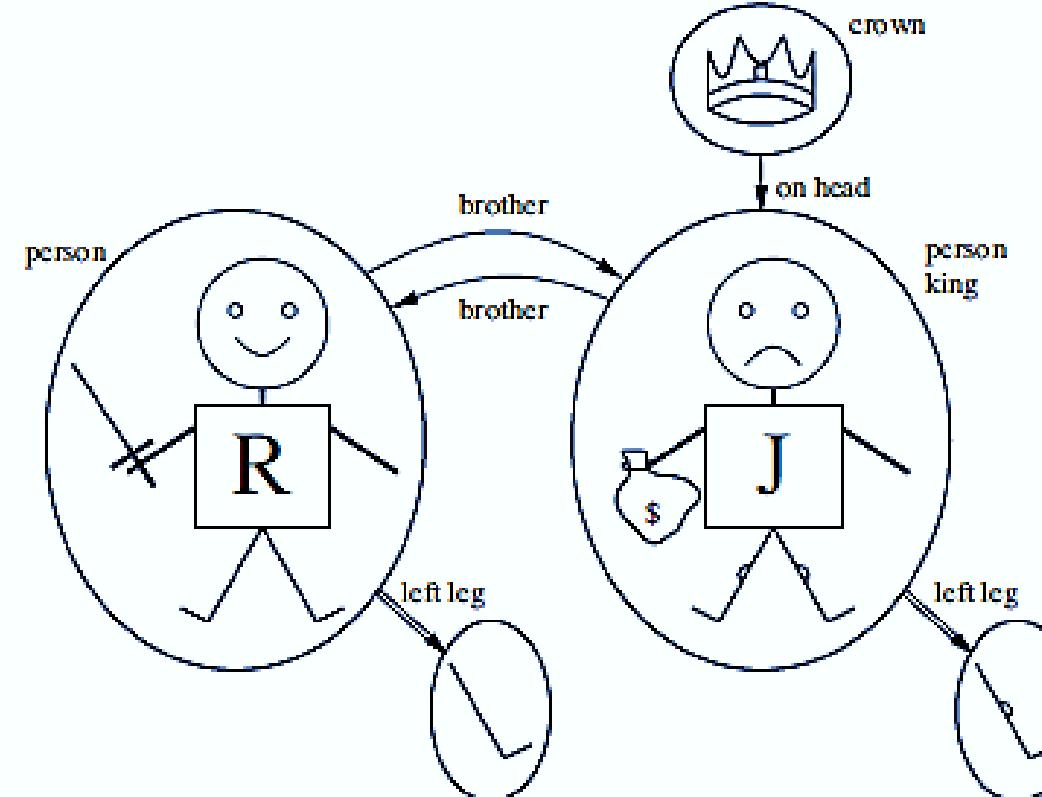
- A function must return a value for every object (total function).

Even a crown must have a "left leg"—we handle this by imagining a default dummy object.

LOGICAL AGENTS

FIRST-ORDER LOGIC

Model in First-Order Logic?



- Domain **must be non-empty**.
- A function must return a value for every object (total function). Even a crown must have a "left leg"—we handle this by imagining a default dummy object.



LOGICAL AGENTS

FIRST-ORDER LOGIC

2. Syntax: The Structure of First-Order Logic

There are three main types of **symbols**:

Symbol Type	Example	What It Refers To
Constant Symbol	Richard	A specific object
Predicate Symbol	Brother(x,y)	A relationship between objects
Function Symbol	LeftLeg(x)	A function returning an object



LOGICAL AGENTS

FIRST-ORDER LOGIC

2. Syntax: The Structure of First-Order Logic

There are three main types of **symbols**:

Symbol Type	Example	What It Refers To
Constant Symbol	Richard	A specific object
Predicate Symbol	Brother(x,y)	A relationship between objects
Function Symbol	LeftLeg(x)	A function returning an object

Each **predicate** and **function** has an **arity**—number of arguments.

LOGICAL AGENTS

FIRST-ORDER LOGIC

3. Terms

A **term** is an expression that refers to an object.

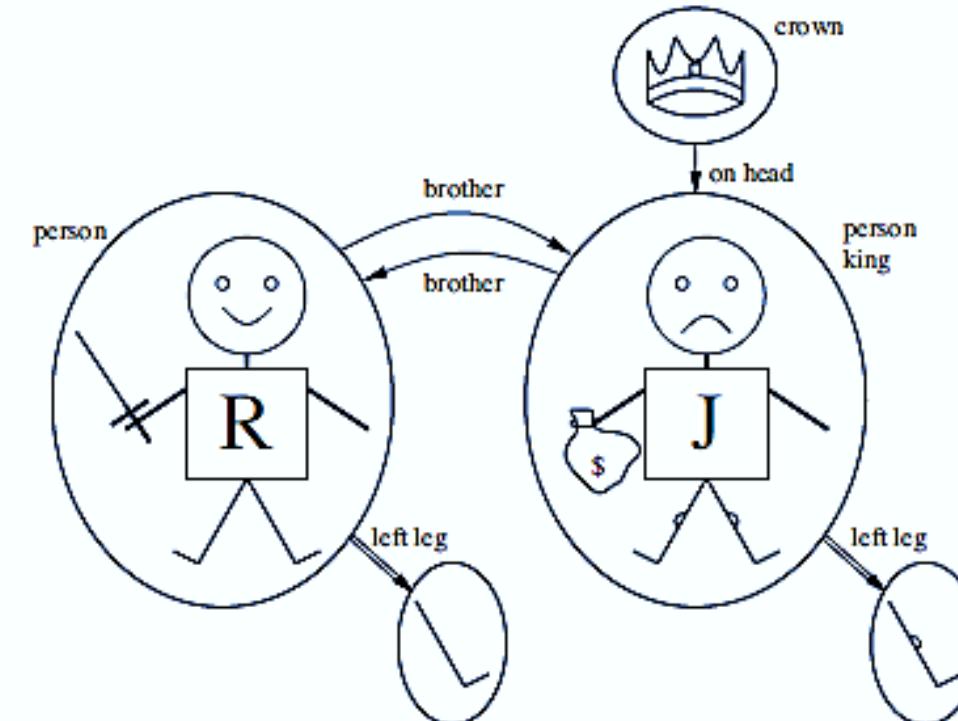
- ◆ **Types of Terms:**

- **Constant:** John

- **Variable:** x, y

- **Function:** LeftLeg(John) →
John's left leg

Think of $\text{LeftLeg}(\text{John})$ like a reference to an object, **not** a function call like in programming.





LOGICAL AGENTS

FIRST-ORDER LOGIC

4. Atomic Sentences (Atoms)

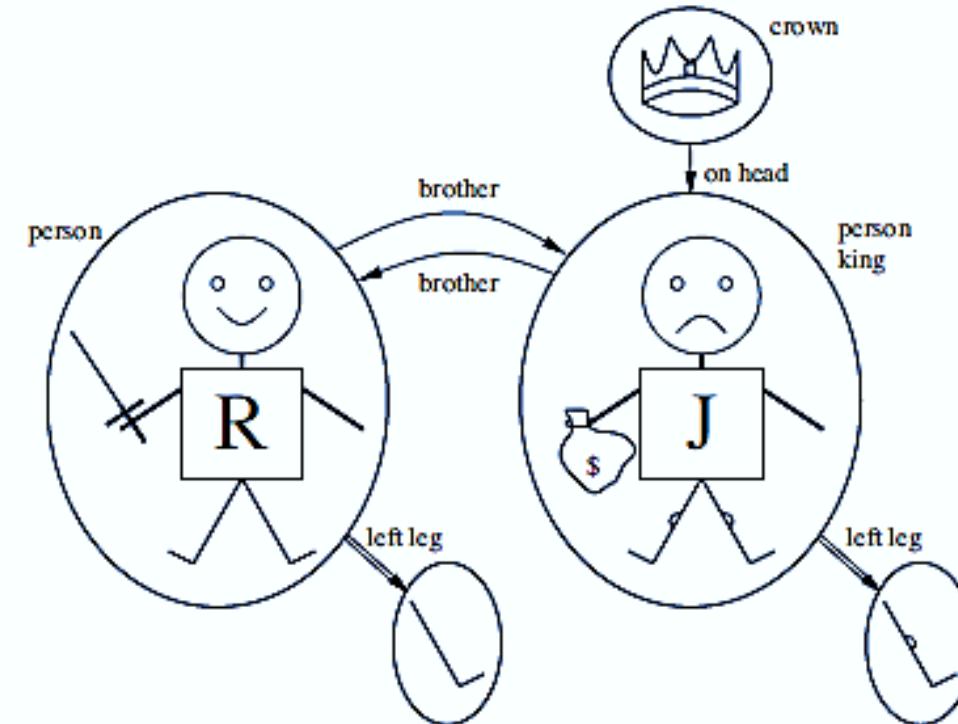
These are the **simplest logical statements.**

- ◆ **Forms:**

- Predicate(Constant)
- Predicate(Constant1, Constant2)
- Term1 = Term2 (equality)

Examples:

- Brother(Richard, John) – true if Richard is John's brother.
- Crown(LeftLeg(Richard)) – likely false.
- LeftLeg(Richard) = RichardLeftLeg – true if they refer to the same object.





LOGICAL AGENTS

FIRST-ORDER LOGIC

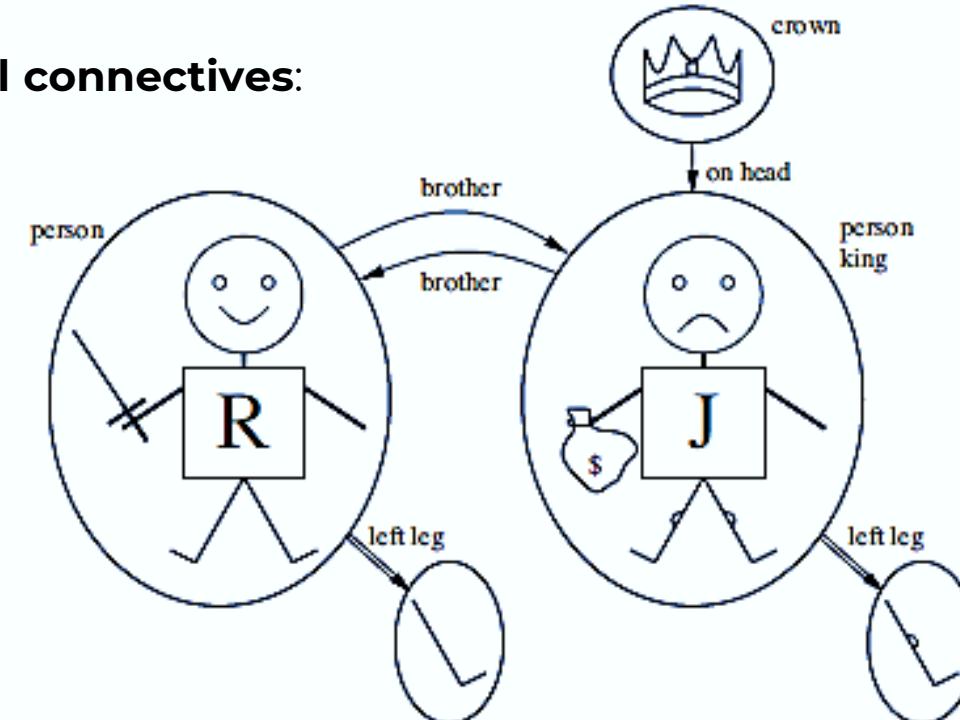
5. Complex Sentences

Built from atomic sentences using **logical connectives**:

Symbol	Meaning
\neg	Not
\wedge	And
\vee	Or
\Rightarrow	Implies
\Leftrightarrow	If and only if

Examples:

- $\neg \text{Brother}(\text{LeftLeg}(\text{Richard}), \text{John})$
→ Left leg is not John's brother (true).
- $\text{Brother}(\text{Richard}, \text{John}) \wedge \text{Brother}(\text{John}, \text{Richard})$
→ Brotherhood is mutual.
- $\neg \text{King}(\text{Richard}) \Rightarrow \text{King}(\text{John})$
→ If Richard isn't a king, then John must be.



LOGICAL AGENTS

FIRST-ORDER LOGIC

6. Quantifiers

Allow us to speak about **all** or **some** objects in the domain.

Universal Quantifier $\forall x$ — “For all x”

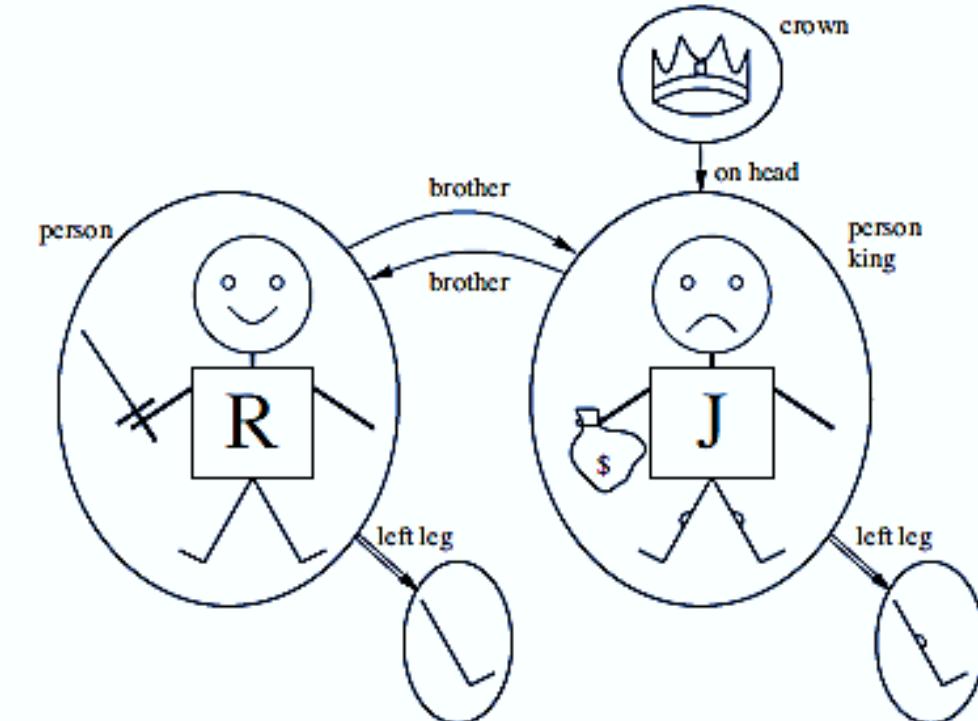
Means the statement is true for every object.

Example:

$\forall x \text{ King}(x) \Rightarrow \text{Person}(x)$

→ If anyone is a king, they must be a person.

If x is not a king (e.g., a crown), the implication is **still true**.



LOGICAL AGENTS

FIRST-ORDER LOGIC



6. Quantifiers

Allow us to speak about **all** or **some** objects in the domain.

Universal Quantifier $\forall x$ — “For all x”

Means the statement is true for every object.

Example:

$$\forall x \text{ King}(x) \Rightarrow \text{Person}(x)$$

→ If anyone is a king, they must be a person.

If x is not a king (e.g., a crown), the implication is **still true**.

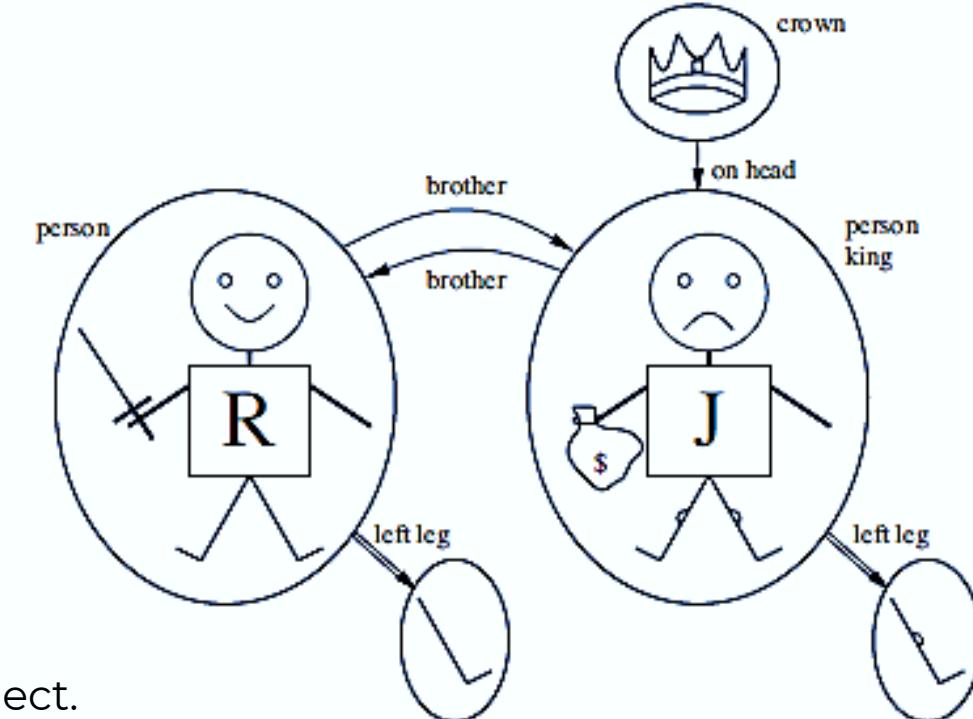
Existential Quantifier $\exists x$ — “There exists an x”

Means the statement is true for **at least one** object.

Example:

$$\exists x \text{ Crown}(x) \wedge \text{OnHead}(x, \text{John})$$

→ There exists some object that is a crown and is on John's head.



LOGICAL AGENTS

FIRST-ORDER LOGIC



7. Nested Quantifiers

The **order matters**.

Examples:

- $\forall x \exists y \text{ Loves}(x, y) \rightarrow$ Everyone loves someone.
- $\exists y \forall x \text{ Loves}(x, y) \rightarrow$ There is someone loved by everyone.

Tip: Use parentheses to understand nesting:

$$\forall x (\exists y \text{ Loves}(x, y)) \neq \exists y (\forall x \text{ Loves}(x, y))$$

8. Equality in First-Order Logic

Equality (=) is used to **assert that two terms refer to the same object**.

Examples:

- $\text{Father}(\text{John}) = \text{Henry} \rightarrow$ John's father is Henry.
- $\exists x, y \text{ Brother}(x, \text{Richard}) \wedge \text{Brother}(y, \text{Richard}) \wedge \neg(x = y)$
 \rightarrow Richard has **two different brothers**.



LOGICAL AGENTS

FIRST-ORDER LOGIC

9. Limitations of Standard Semantics

A statement like

Brother(John, Richard) \wedge Brother(Geoffrey, Richard)

doesn't mean John and Geoffrey are the **only** brothers.

To state exactly two:

Brother(John, Richard) \wedge Brother(Geoffrey, Richard) \wedge John \neq Geoffrey

$\wedge \forall x \text{ Brother}(x, \text{Richard}) \Rightarrow (x = \text{John} \vee x = \text{Geoffrey})$

This is precise, but not very **natural** or **intuitive**.

LOGICAL AGENTS

FIRST-ORDER LOGIC

10. Alternative: Database Semantics

Used in databases and logic programming.

Assumptions:

1. **Unique names:** Each constant refers to a distinct object.
2. **Closed world:** What isn't known to be true is assumed false.
3. **Domain closure:** Only the named constants exist.

Under this:

Brother(John, Richard) \wedge Brother(Geoffrey, Richard)

does mean that **only John and Geoffrey** are Richard's brothers.

More natural, but assumes complete knowledge.



LOGICAL AGENTS

FIRST-ORDER LOGIC

11. De Morgan's Laws for Quantifiers

Link \forall and \exists using negation:

Statement	Equivalent
$\forall x \neg P(x)$	$\neg \exists x P(x)$
$\neg \forall x P(x)$	$\exists x \neg P(x)$
$\exists x \neg P(x)$	$\neg \forall x P(x)$
$\forall x P(x)$	$\neg \exists x \neg P(x)$

Example:

- Everyone dislikes parsnips: $\forall x \neg \text{Likes}(x, \text{Parsnips})$
- Equivalent: $\neg \exists x \text{ Likes}(x, \text{Parsnips})$

LOGICAL AGENTS

FIRST-ORDER LOGIC



Concept	FOL Approach
World Representation	Use models with objects, functions, relations
Object Identification	Terms (constants, functions)
Stating Facts	Atomic and complex sentences
Generalization	Quantifiers \forall, \exists
Precise Conditions	Use equality and nested quantifiers
Simpler Expression	Consider database semantics



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Domain in FOL

In **knowledge representation**, a **domain** refers to a specific part of the world we are interested in — the **subject or environment** we want to describe logically.

Examples of domains:

- Family relationships
- Numbers
- Sets and lists
- The Wumpus world
- Electronic circuits
- The entire universe



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

TELL/ASK Interface for First-Order Knowledge Bases

FOL systems use a **TELL/ASK interface** to interact with the **Knowledge Base (KB)**:

TELL

- Used to **add facts or rules** to the knowledge base.
- These are called **assertions**.

Examples of assertions:

TELL(KB, King(John)) // John is a king

TELL(KB, Person(Richard)) // Richard is a person

TELL(KB, $\forall x \text{ King}(x) \Rightarrow \text{Person}(x)$) // All kings are persons



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

ASK

- Used to **ask questions (queries or goals)** based on what is known.
- The system checks if the query is **logically entailed** by the KB.

Example queries:

ASK(KB, King(John)) // Returns TRUE (as John is asserted to be a king)

ASK(KB, Person(John)) // Returns TRUE (by inference: John is a King ⇒ Person)



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Quantified Queries in ASK

We can ask more **generalized questions** using **quantifiers**, like:

ASK(KB, $\exists x \text{ Person}(x)$) // Is there someone who is a person?

- This returns TRUE, since at least John and Richard are known to be persons.
- But it doesn't tell us **who** these people are — just that they **exist**.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Getting Variable Bindings: ASKVARS

To know **which specific values** make a query true, we use:

ASKVARS(KB, Person(x))

This returns **substitutions** or **binding lists**, such as:

{x/John}, {x/Richard}

These show that both **John** and **Richard** are persons.

This process is more powerful when the knowledge base consists of **Horn clauses**, allowing variables to be **concretely bound**.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Limitations of FOL with Disjunctions

If the knowledge base has a sentence like:

TELL(KB, King(John) \vee King(Richard))

Then,

ASK(KB, $\exists x$ King(x))

• Still returns **TRUE**, but:

- You **can't bind** x to a specific value (John or Richard), because **disjunction** does not identify which one is true.
- So **no specific substitution** can be returned — only the truth of the existence.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Operation	Purpose	Example	Output
TELL	Add facts/rules to KB	TELL(KB, King(John))	Adds assertion
ASK	Check if a query is logically entailed	ASK(KB, Person(John))	TRUE or FALSE
ASKVARS	Retrieve variable bindings	ASKVARS(KB, Person(x))	{x/John}, {x/Richard}
Disjunction	Doesn't support specific bindings	TELL(KB, King(John) v King(Richard))	Truth known, but no binding



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Kinship Domain in First-Order Logic

First-Order Logic (FOL) is widely used in AI to **represent knowledge** and **reason** about it. A classic example to illustrate this is **family relationships** — the **kinship domain**.

Kinship Domain:

The **kinship domain** includes:

- **Facts** (e.g., “Elizabeth is the mother of Charles”)
- **Rules** (e.g., “A grandmother is the mother of a parent”)

We use **FOL** to represent these formally using:

- **Objects**: People like Elizabeth, Charles, William.
- **Predicates**: Describe properties or relations (e.g., $\text{Male}(x)$, $\text{Parent}(x, y)$).
- **Functions**: Map individuals to individuals (e.g., $\text{Mother}(c)$ returns c's mother).
- **Constants**: Specific people like Elizabeth, Charles.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Core Predicates and Functions

Type	Name/Example	Meaning
Unary	Male(x)	x is a male
Unary	Female(x)	x is a female
Binary	Parent(x, y)	x is a parent of y
Binary	Sibling(x, y)	x and y are siblings
Binary	Spouse(x, y)	x is married to y
Binary	Husband(h, w)	h is husband of w
Binary	Child(x, y)	x is a child of y
Function	Mother(c)	returns the mother of c
Function	Father(c)	returns the father of c



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Sample Axioms (Definitions in FOL)

These logical formulas define relationships **in terms of other predicates**.

Example 1: Definition of Mother

“A person’s mother is their female parent.”

$$\forall m, c: \text{Mother}(c) = m \Leftrightarrow \text{Female}(m) \wedge \text{Parent}(m, c)$$

Example 2: Definition of Husband

“A husband is a male spouse.”

$$\forall h, w: \text{Husband}(h, w) \Leftrightarrow \text{Male}(h) \wedge \text{Spouse}(h, w)$$



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Sample Axioms (Definitions in FOL)

These logical formulas define relationships **in terms of other predicates**.

Example 3: Male and Female are disjoint

“No one is both male and female.”

$$\forall x: \text{Male}(x) \Leftrightarrow \neg \text{Female}(x)$$

Example 4: Parent and Child are inverses

$$\forall p, c: \text{Parent}(p, c) \Leftrightarrow \text{Child}(c, p)$$



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Sample Axioms (Definitions in FOL)

These logical formulas define relationships **in terms of other predicates**.

Example 5: Grandparent Definition

“A grandparent is a parent of a parent.”

$$\forall g, c: \text{Grandparent}(g, c) \Leftrightarrow \exists p: \text{Parent}(g, p) \wedge \text{Parent}(p, c)$$

Example 6: Sibling Definition

“Two people are siblings if they share at least one parent and are not the same person.”

$$\forall x, y: \text{Sibling}(x, y) \Leftrightarrow x \neq y \wedge \exists p: \text{Parent}(p, x) \wedge \text{Parent}(p, y)$$



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Axioms vs Theorems

- **Axioms** are **basic truths** provided in the knowledge base.
 - **Theorems** are **logical consequences** of axioms. They are **derived** using inference.

Example:

The **symmetry of siblinghood** is not given directly as an axiom:

$\forall x, y: \text{Sibling}(x, y) \Leftrightarrow \text{Sibling}(y, x)$

But it can be **proven from the definition of Sibling**, so it's a **theorem**.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Importance of Axioms

1. Base Knowledge: All reasoning in AI depends on axioms as the foundation.

2. Definitions: Many axioms define new predicates/functions in terms of others.

3. Just Facts: Some axioms are plain facts like Male(Jim) or Spouse(Jim, Laura).



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Why Some Answers May Fail

If your knowledge base has:

Spouse(Jim, Laura)
Jim ≠ George

You **cannot automatically infer** that:

¬Spouse(George, Laura)

Why?

Because we haven't added the axiom:

"A person can have only one spouse."

You must explicitly **add such constraints** to allow valid reasoning.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Bottom-Up Construction of Domain

- **Start with basic predicates** (e.g., Parent, Spouse, Female)
- **Define more complex ones** (e.g., Grandparent, Sibling)
- This mimics how software is built: basic libraries → subroutines → programs

Not Everything Needs Full Definition

For instance, Person(x) may be used in statements like:

$$\forall x: \text{Person}(x) \Rightarrow \dots$$

But we **don't need a full definition** of Person — just enough to use it in reasoning.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Why FOL for Kinship is Useful in AI

- Represents complex **human relationships**
- Enables **logical inference** and **query answering**
- Can be **modularly extended** with new relationships
- Encourages **explicit modeling** of world knowledge



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Natural Numbers and Peano Axioms

Purpose:

To formally define natural numbers (0, 1, 2, ...) using **logical axioms**.

Building Blocks:

- **Predicate:** $\text{NatNum}(n)$ – true if n is a natural number.
- **Constant:** 0 – base element of natural numbers.
- **Function:** $S(n)$ – successor of n , i.e., $n + 1$.

Peano Axioms (basic axioms to define natural numbers):

1. $\text{NatNum}(0)$

→ 0 is a natural number.

2. $\forall n \text{ NatNum}(n) \Rightarrow \text{NatNum}(S(n))$

→ If n is a natural number, then its successor is also a natural number.

Together, these recursively define the sequence:

0, $S(0)$, $S(S(0))$, ... (i.e., 0, 1, 2, 3, ...).



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Successor Function Constraints:

To ensure the successor behaves correctly:

- $\forall n \ 0 \neq S(n)$
→ 0 is not the successor of any number.
- $\forall m, n \ m \neq n \Rightarrow S(m) \neq S(n)$
→ Different numbers have different successors (injective function).

Infix and Syntactic Sugar:

To make expressions readable:

- Write $m + n$ instead of $+(m, n)$
- Write $n + 1$ instead of $S(n)$

These are **syntactic sugar** – they make logic expressions easier to read, but are internally equivalent to the raw logical forms.

Defining Addition:

Addition is defined using the successor:

- 1. $+ (0, m) = m$
→ Adding 0 doesn't change the number.
 - 2. $+ (S(m), n) = S(+ (m, n))$
→ Adding $S(m)$ to n is like taking $m + n$ and adding 1.
- This is **recursive addition** – addition as repeated succession.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Sets

Purpose:

To represent and reason about **sets** and **set operations** using logic.

Vocabulary:

- **Constant:** {} – the empty set.
- **Predicate:** Set(s) – true if s is a set.

• Binary predicates:

- $x \in s$ – membership.
- $s_1 \subseteq s_2$ – subset relation.

• Functions:

- $\{x|s\}$ – add element x to set s.
- $s_1 \cup s_2$ – union.
- $s_1 \cap s_2$ – intersection.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Set Axioms:

1. $\text{Set}(s) \Leftrightarrow (s = \{\}) \vee (\exists x, s_2 \text{ Set}(s_2) \wedge s = \{x|s_2\})$

→ Only the empty set and those formed by adjoining elements are sets.

2. $\neg \exists x, s \{x|s\} = \{\}$

→ The empty set cannot be formed by adding an element.

3. $\forall x, s x \in s \Leftrightarrow s = \{x|s\}$

→ Adding an element already in the set doesn't change the set.

4. $\forall x, s x \in s \Leftrightarrow \exists y, s_2 (s = \{y|s_2\} \wedge (x = y \vee x \in s_2))$

→ Recursive definition of membership.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Set Axioms:

$$5. s_1 \subseteq s_2 \Leftrightarrow \forall x (x \in s_1 \Rightarrow x \in s_2)$$

→ Subset if all elements of one set are in the other.

$$6. s_1 = s_2 \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$$

→ Two sets are equal if they are subsets of each other.

$$7. x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$$

→ Intersection.

$$8. x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$$

→ Union.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Set Axioms:

$$5. s_1 \subseteq s_2 \Leftrightarrow \forall x (x \in s_1 \Rightarrow x \in s_2)$$

→ Subset if all elements of one set are in the other.

$$6. s_1 = s_2 \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$$

→ Two sets are equal if they are subsets of each other.

$$7. x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$$

→ Intersection.

$$8. x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$$

→ Union.

These axioms together give a complete logical structure to reason about **set theory** using FOL.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Lists

Purpose:

Lists are ordered collections (unlike sets) and allow duplicates.

Vocabulary (inspired by Lisp):

• **Constant:** Nil – the empty list.

• Functions:

- Cons(x, y) – add element x to front of list y.
- Append(l1, l2) – concatenate lists.
- First(l) – first element.
- Rest(l) – all elements except the first.

• Predicates:

- Find(x, l) – true if x is in list l.
- List?(l) – true if l is a list.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Lists

Syntactic Sugar for Lists:

- [] → empty list (Nil)
- [x] → list with one element = Cons(x, Nil)
- [x|y] → list with head x and tail y = Cons(x, y)
- [A,B,C] → Cons(A, Cons(B, Cons(C, Nil)))

Key Differences from Sets:

- Lists maintain **order**.
- Lists allow **repetition**.
- Sets are defined by **membership**, lists by **construction** (head/tail).



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

Concept	Constant	Function(s)	Predicate(s)	Notes
Natural Numbers	0	$S(n)$, $+$	$\text{NatNum}(n)$	Uses Peano axioms
Sets	$\{\}$	$\backslash\{x\}$	$s], \cap, \cup$	$\text{Set}(s), x \in s, s1 \subseteq s2$
Lists	Nil / $[]$	Cons, Append, First, Rest	List?, Find	Ordered and allow duplicates



LOGICAL AGENTS

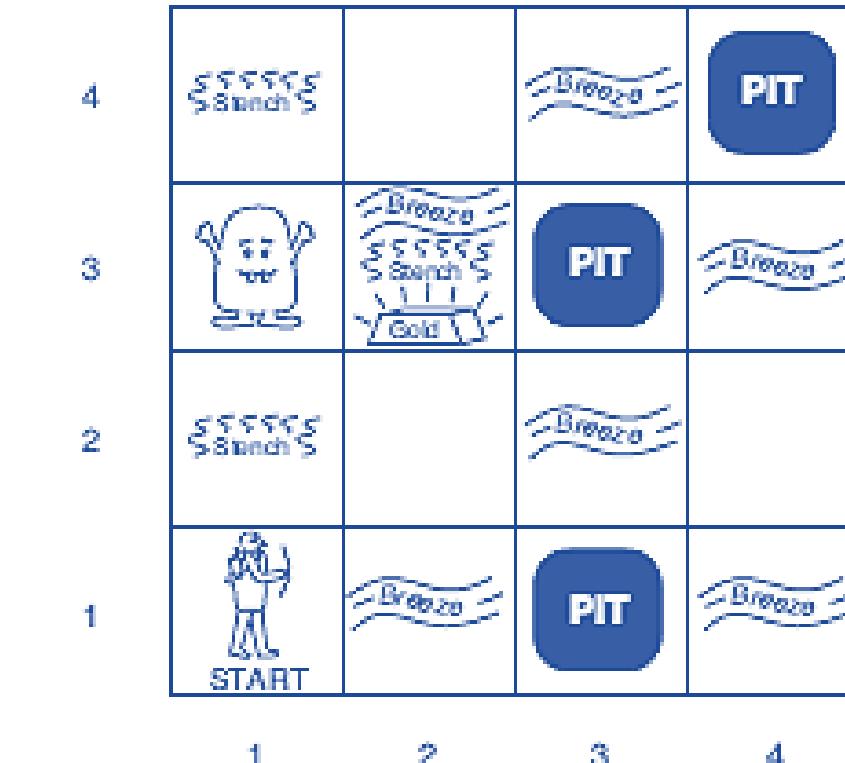
FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

The Wumpus World in First-Order Logic

The Wumpus World is a **grid-based environment** consisting of:

- **Agent**: The player/AI agent navigating the world.
- **Wumpus**: A monster that kills the agent if it enters its square.
- **Pits**: Deadly holes that also kill the agent.
- **Gold**: The objective for the agent to find.
- **Percepts**: Clues the agent gets in each square (stench, breeze, glitter, bump, scream).





LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

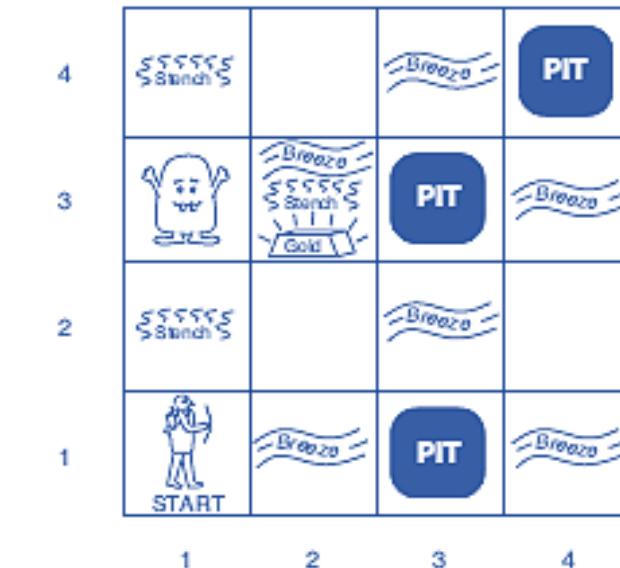
The Wumpus World in First-Order Logic

Why Use First-Order Logic (FOL)?

Propositional logic is too verbose for dynamic, time-based reasoning like in the Wumpus World.

FOL allows concise, general, and reusable representations by enabling:

- Variables (x, t, s, etc.)
- Quantifiers (\forall , \exists)
- Functions and predicates (At, Percept, Breeze)
- Logical reasoning across **space and time**



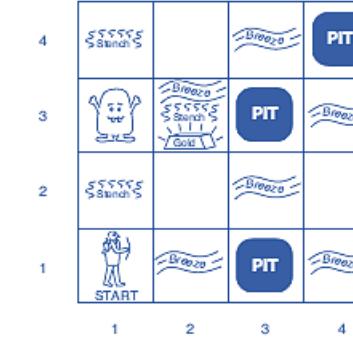


LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

The Wumpus World in First-Order Logic



2. Representing Percepts and Time

Each time the agent receives a percept (like seeing glitter or feeling a breeze), it stores that information **along with the time** it occurred.

Example:

Percept([Stench, Breeze, Glitter, None, None], 5)

- Percept is a **predicate**.
- The list contains 5 perceptual elements.
- 5 is the **time step**.

This ensures the agent doesn't confuse what it saw **now** versus what it saw **earlier**.

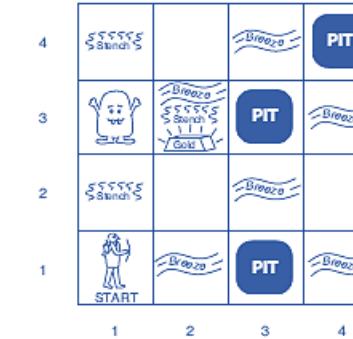


LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

The Wumpus World in First-Order Logic



3. Representing Actions

Actions the agent can perform are represented as **terms**:

Turn(Right), Turn(Left), Forward, Shoot, Grab, Climb

When the agent decides on the best action, it may issue a **query**:
plaintext

ASKVARS(\exists a BestAction(a, 5))

This **returns a binding**, like {a/Grab}, meaning at time 5, the best action is Grab.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

The Wumpus World in First-Order Logic

4. Reasoning About Percepts (Perception Axioms)

We can write general rules to **interpret percepts**:

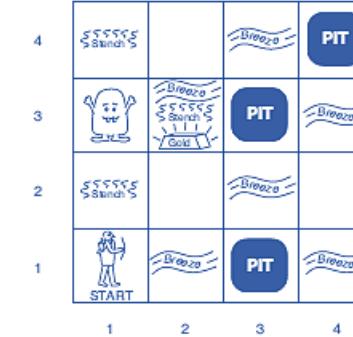
$$\forall t, s, g, m, c: \text{Percept}([s, \text{Breeze}, g, m, c], t) \Rightarrow \text{Breeze}(t)$$

If the percept contains "Breeze" at time t, then we assert $\text{Breeze}(t)$.

Similarly:

$$\forall t, s, b, m, c: \text{Percept}([s, b, \text{Glitter}, m, c], t) \Rightarrow \text{Glitter}(t)$$

This is the **first layer of reasoning**, directly translating sensor input into internal facts.



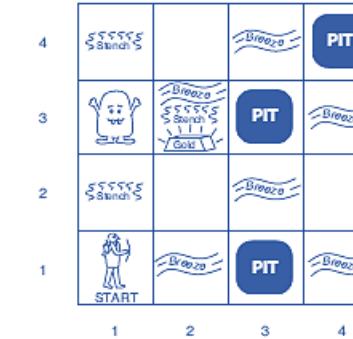


LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

The Wumpus World in First-Order Logic



5. Reflex Behavior (Simple Decision Making)

Simple rule-based behavior can be encoded with implications:

$$\forall t: \text{Glitter}(t) \Rightarrow \text{BestAction(Grab, t)}$$

So, if there's **glitter**, the best action is to **grab** at that time.

This is **reflex-level AI**—reacting to immediate inputs.



LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

The Wumpus World in First-Order Logic

6. Representing the Environment

Squares and Locations

Instead of naming each square (e.g., Square1_2), we use coordinates:
plaintext

[1, 2] → means row 1, column 2

Adjacency

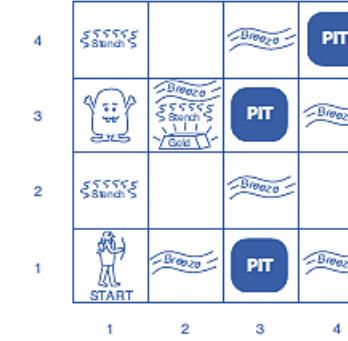
plaintext

$\forall x, y, a, b: \text{Adjacent}([x, y], [a, b]) \Leftrightarrow (x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1))$

This defines what it means for two squares to be **next to each other**.

Pits and Wumpus

- Pit([x, y]) → There is a pit at square [x, y]
- Wumpus → A constant, since there is only **one Wumpus**





LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

The Wumpus World in First-Order Logic

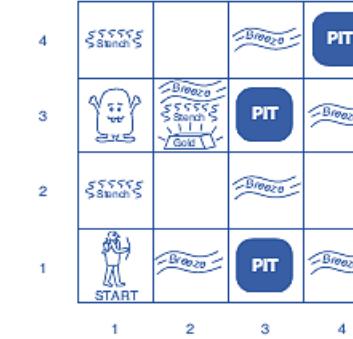
7. Agent Location and Time

To track the agent's position over time:

At(Agent, [x, y], t) → Agent is at square [x, y] at time t At(Wumpus, [2, 2], t) → Wumpus is at [2,2] at all times

Ensure uniqueness of position:

$$\forall x, s1, s2, t: \text{At}(x, s1, t) \wedge \text{At}(x, s2, t) \Rightarrow s1 = s2$$





LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

The Wumpus World in First-Order Logic

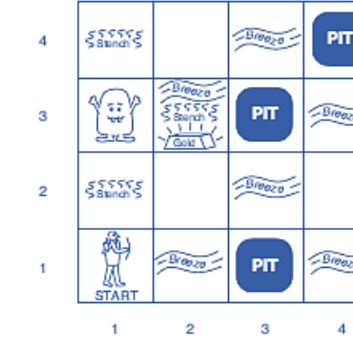
8. Inferring Facts From Location

From the agent's current location and percepts, infer square properties:

$$\forall s, t: \text{At}(\text{Agent}, s, t) \wedge \text{Breeze}(t) \Rightarrow \text{Breezy}(s)$$

So if the agent feels a breeze and it's at square s, then s is **breezy**

Notice: Breezy(s) has **no time** — pits don't move!



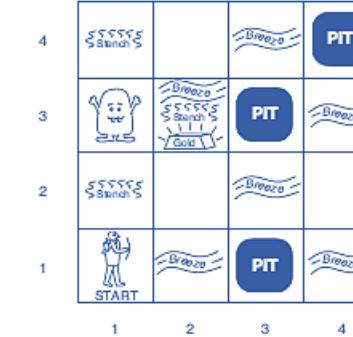


LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

The Wumpus World in First-Order Logic



9. Deduction: Where are the pits?

Use the inverse logic:

$$\forall s: \text{Breezy}(s) \Leftrightarrow \exists r: \text{Adjacent}(r, s) \wedge \text{Pit}(r)$$

If a square is breezy, it means **some adjacent square has a pit**.

This is much more efficient than propositional logic, where you'd write this rule for **each square separately**.

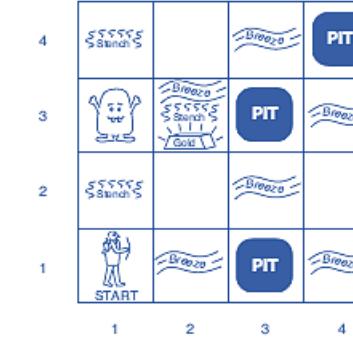


LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

The Wumpus World in First-Order Logic



10. Successor-State Axioms (Time Reasoning)

Example: Arrow possession:

$$\forall t: \text{HaveArrow}(t+1) \Leftrightarrow \text{HaveArrow}(t) \wedge \neg \text{Action}(\text{Shoot}, t)$$

You still have the arrow at $t+1$ if you had it at t and didn't shoot.

This is a **compact way** to describe how the world evolves over time.

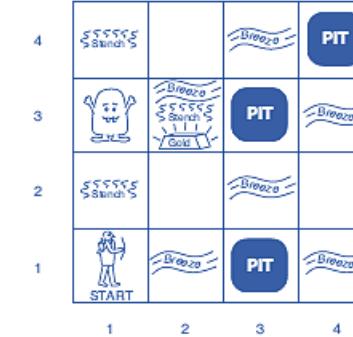


LOGICAL AGENTS

FIRST-ORDER LOGIC

USING FIRST-ORDER LOGIC

The Wumpus World in First-Order Logic



Concept	First-Order Logic Example
Percept at time	Percept([Stench, Breeze, ...], t)
Action	Action(Grab, t)
Location	At(Agent, [x, y], t)
Reflex rule	Glitter(t) \Rightarrow BestAction(Grab, t)
Breeze rule	Breezy(s) \Leftrightarrow $\exists r: \text{Adjacent}(r, s) \wedge \text{Pit}(r)$
Arrow possession	HaveArrow(t+1) \Leftrightarrow HaveArrow(t) \wedge \neg Action(Shoot, t)

LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC



Knowledge Engineering is the **process of building a knowledge base**—a structured collection of facts and rules about a specific domain—using logical representations like First-Order Logic (FOL).

A **Knowledge Engineer** plays a crucial role: they explore the domain, identify key concepts, and translate that understanding into a formal, logical structure.

Think of it as turning human expertise into a machine-readable logical format.

LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process

Even though projects may differ in complexity or subject, they usually follow **7 core steps**:

1. Identify the Task

- Define **what the knowledge base must do**:
 - What kind of questions should it answer?
 - What information will be provided for each case?
- This step is like defining the **goal and scope** of the AI system.

Example:

In a Wumpus World agent, is the goal to **find gold, survive**, or just **describe the environment**? Will the input include the agent's current location?

LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process

2. Assemble the Relevant Knowledge

- Collect domain knowledge through:
 - Expert interviews
 - Manuals
 - Observation
- This stage is called **Knowledge Acquisition**.

At this stage, knowledge is informal—it's just about understanding the **concepts and relationships**.



LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process

3. Define the Vocabulary (Ontology)

- Choose **predicates**, **functions**, and **constants** to represent the domain concepts.
- Decisions made here form the **ontology**: the vocabulary used to describe the domain logically.

Design Choices:

- Represent "Pit" as a predicate Pit(x) or as an object with properties?
- Use Facing(North) or a function Orientation(Agent)?

LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process

4. Encode General Knowledge (Axioms)

- Write logical **axioms** using the chosen vocabulary.
- These axioms define **how the domain works**, independent of specific cases.

Often reveals mistakes or missing concepts → you may need to revisit Step 3.



LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process

5. Encode Problem-Specific Facts

- Add logical facts for a **specific instance** or scenario.
- These can be:
 - Sensor inputs (in agents)
 - Direct input (in static knowledge systems)

Example:

In Wumpus World:
Breezy([2,2]), Smelly([1,3])

6. Pose Queries and Get Answers

- Use a **logical inference engine** to query the knowledge base.
 - You don't write algorithms—just let the system deduce answers from facts and axioms.
-  *Example Query:*
Is there a pit in [2,2]?

LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process

6. Pose Queries and Get Answers

- Use a **logical inference engine** to query the knowledge base.
- You don't write algorithms—just let the system deduce answers from facts and axioms.

Example Query:

Is there a pit in [2,2]?

LOGICAL AGENTS

FIRST-ORDER LOGIC



KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process

7. Debug the Knowledge Base

- Verify if the answers make sense.
- Common issues:
 - Missing axioms: reasoning chain breaks
 - Incorrect axioms: false statements
 - Weak axioms: can't derive strong conclusions

Examples of bugs:

- **Missing rule:** You can't conclude $\neg \text{Wumpus}(x)$ because a biconditional was left out.
- **Incorrect logic:**
 $\forall x \text{ NumLegs}(x, 4) \Rightarrow \text{Mammal}(x)$ is **false** (reptiles and tables also have 4 legs!).

Unlike programming bugs, logic bugs can often be spotted by **examining statements in isolation**.

LOGICAL AGENTS

FIRST-ORDER LOGIC



KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process

7. Debug the Knowledge Base

- Verify if the answers make sense.
- Common issues:
 - Missing axioms: reasoning chain breaks
 - Incorrect axioms: false statements
 - Weak axioms: can't derive strong conclusions

Examples of bugs:

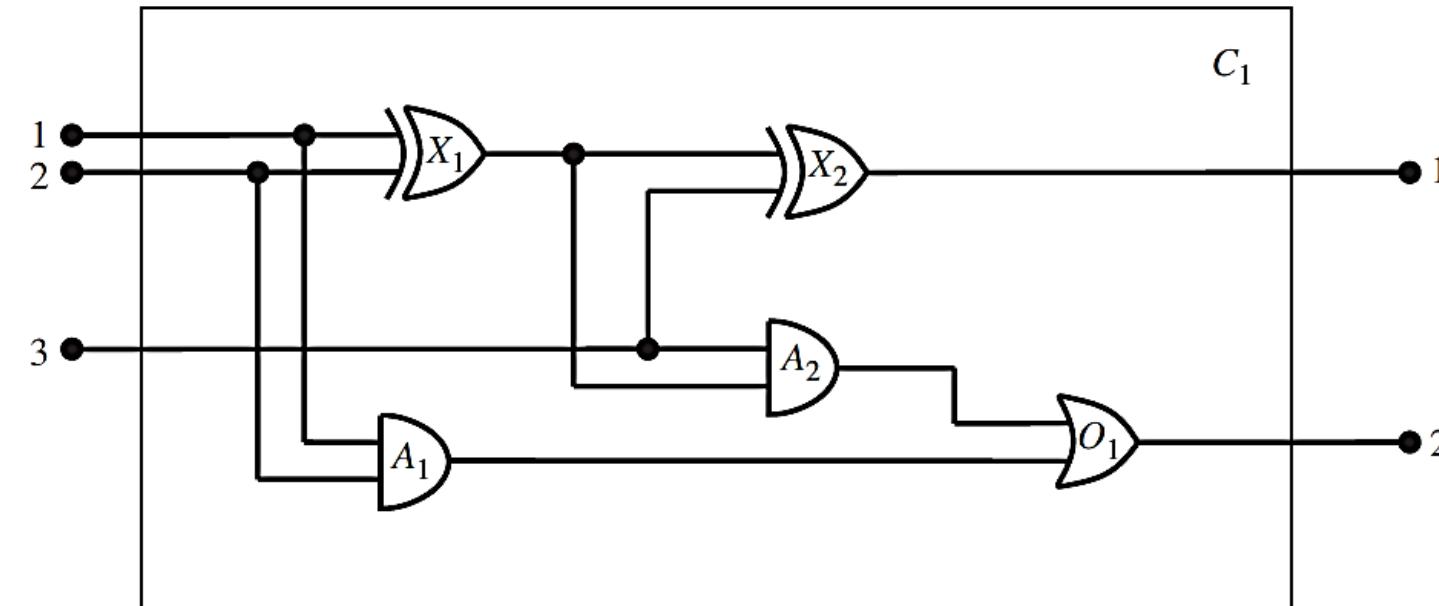
- **Missing rule:** You can't conclude $\neg \text{Wumpus}(x)$ because a biconditional was left out.
 - **Incorrect logic:**
 $\forall x \text{ NumLegs}(x, 4) \Rightarrow \text{Mammal}(x)$ is **false** (reptiles and tables also have 4 legs!).
- Unlike programming bugs, logic bugs can often be spotted by **examining statements in isolation**.

LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process The electronic circuits domain



A digital circuit C_1 , purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.



LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process The electronic circuits domain

1. Identify the Task

We want to **reason about digital circuits**, particularly:

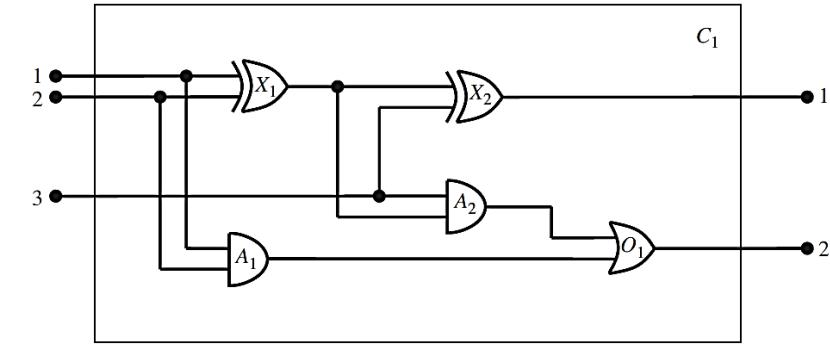
- Their **functionality**: Does the circuit compute correctly?
- Their **structure**: How are components connected?

◆ Examples:

• **Functionality query**: What is the output of gate A2 if all inputs are 1?

• **Structure query**: Which gates are connected to the first input?

Other tasks like timing delays, power usage, and cost require additional knowledge (but are not our focus here).



LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process The electronic circuits domain

2. Assemble the Relevant Knowledge

To analyze digital circuits:

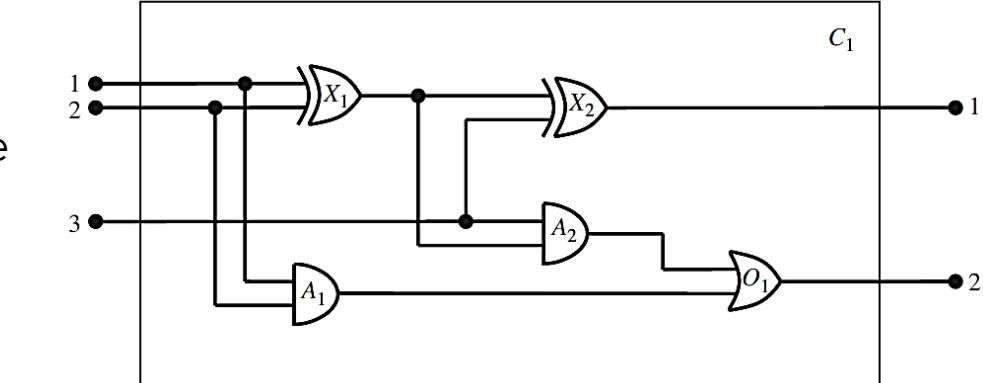
- **Components:** Gates and wires.
- **Behavior:** Gates receive input via wires and produce output.
- **Types of Gates:**
 - **AND, OR, XOR:** Two inputs.
 - **NOT:** One input.
- **Abstraction:** Ignore wire paths, shapes, or colors — only care about **connections between terminals**.

Example:

Instead of tracking the physical wire from gate X1 to X2, we only say:

Connected(Out(1, X1), In(1, X2))

If we were debugging faults, we **would** model wires and delays.





LOGICAL AGENTS

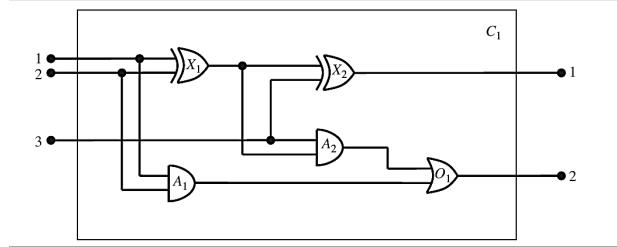
FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process The electronic circuits domain

3. Decide on a Vocabulary

We define constants, functions, and predicates to represent components and signals.



Concept	Representation	Example
Gate	Gate(X1)	X1 is a gate
Gate type	Type(X1) = XOR	X1 is an XOR gate
Circuit	Circuit(C1)	C1 is the circuit
Terminal	Terminal(t)	t is a terminal
Input terminal	In(1, X1)	First input to gate X1
Output terminal	Out(1, X1)	Output from gate X1
Connection	Connected(Out(1, X1), In(1, X2))	Output of X1 connects to input of X2
Signal value	Signal(t) = 1 or 0	Signal at terminal t is ON or OFF

LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process The electronic circuits domain

4. Encode General Knowledge (Rules/Axioms)

These rules describe how circuits behave.

- ◆ **Axioms and Examples:**

1. Connected terminals share signal

→ If Out(1, X1) is connected to In(1, X2), then they carry the same signal.

2. Signal values are binary (0 or 1)

→ No floating/undefined signals.

3. Connection is symmetric

→ Connected(A,B) \Leftrightarrow Connected(B,A)

4. Valid gate types

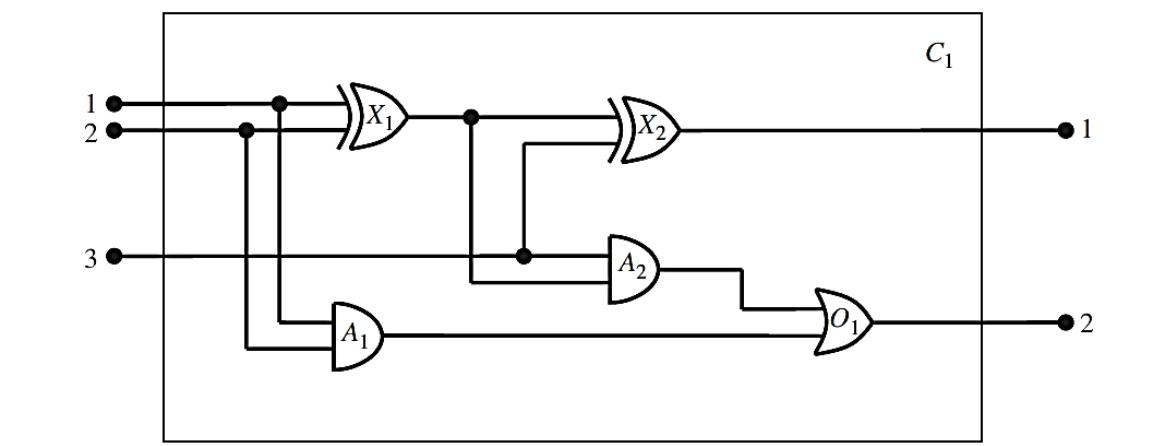
→ AND, OR, XOR, NOT

5. AND gate behavior

→ Output is 0 \Leftrightarrow any input is 0

6. OR gate behavior

→ Output is 1 \Leftrightarrow any input is 1





LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process The electronic circuits domain

4. Encode General Knowledge (Rules/Axioms)

These rules describe how circuits behave.

- ◆ **Axioms and Examples:**

7.XOR gate behavior

→ Output is $1 \Leftrightarrow$ inputs are different

8.NOT gate behavior

→ Output is inverse of input

9.Gate arity

→ XOR/AND/OR: 2 inputs, 1 output

→ NOT: 1 input, 1 output

10.Circuit arity

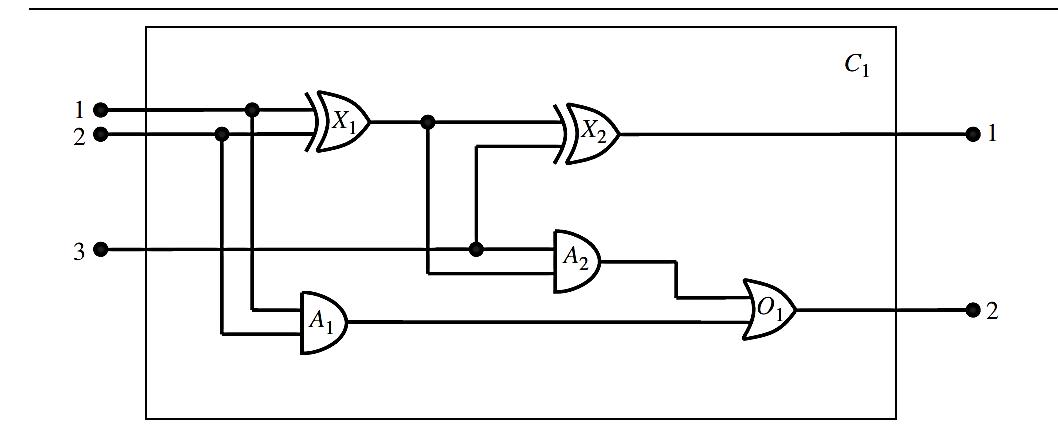
→ Number of input/output terminals must match circuit definition.

11.All entities are distinct

→ Gate \neq Terminal $\neq 0 \neq 1$, etc.

12.Gates are circuits

→ Every gate is also a circuit



LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process The electronic circuits domain

5. Encode the Specific Circuit

The digital circuit C1 (a 1-bit full adder) contains:

- **Inputs:** A, B, Carry-in
- **Outputs:** Sum, Carry-out
- **Gates:**
 - X₁, X₂ → XOR gates
 - A₁, A₂ → AND gates
 - O₁ → OR gate

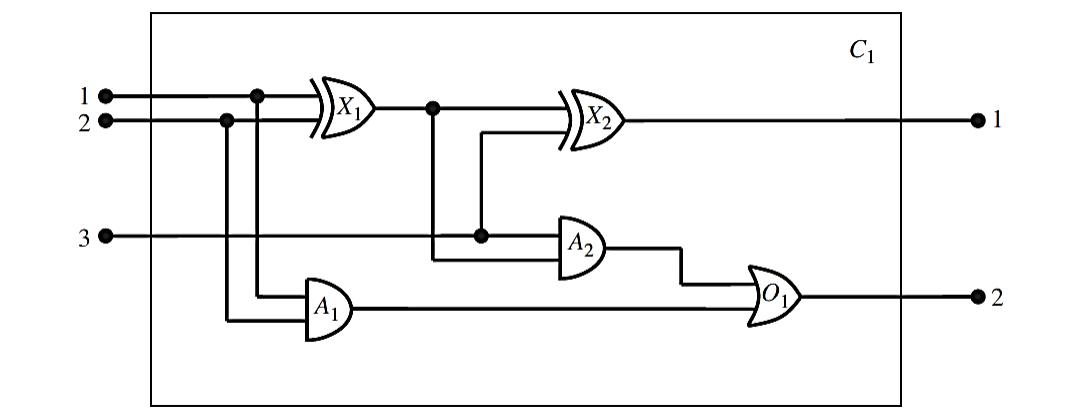
◆ Define Components:

Circuit(C1), Arity(C1, 3, 2) Gate(X1), Type(X1)=XOR Gate(A1), Type(A1)=AND ...

◆ Define Connections:

prolog

Connected(In(1,C1), In(1,X1)) Connected(Out(1,X1),
 In(2,A2)) Connected(Out(1,X2), Out(1,C1)) Connected(Out(1,O1), Out(2,C1))
 This builds the logical model of how signals move through the gates.



LOGICAL AGENTS

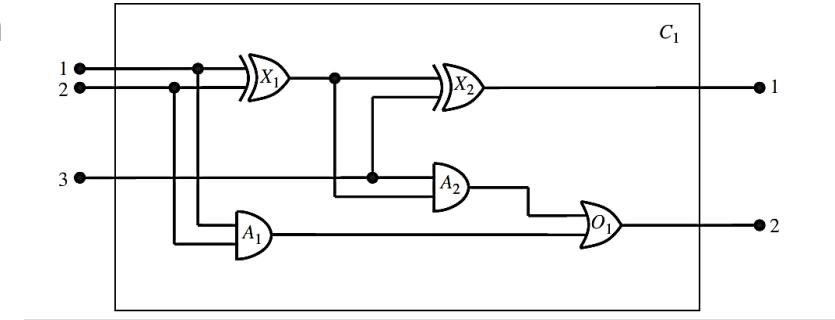
FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process The electronic circuits domain

6. Pose Queries (Reasoning Examples)

Q1: Which inputs make Sum = 0 and Carry = 1?

$$\exists i_1, i_2, i_3 \text{ Signal}(In(1,C1))=i_1 \wedge \text{Signal}(In(2,C1))=i_2 \wedge \text{Signal}(In(3,C1))=i_3 \wedge \\ \text{Signal}(Out(1,C1))=0 \wedge \text{Signal}(Out(2,C1))=1$$


Solutions (i1, i2, i3):

- (1, 1, 0)
- (1, 0, 1)
- (0, 1, 1)

These are input combinations where the **sum bit = 0**,
and the **carry bit = 1**.

LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process The electronic circuits domain

Q2: What are **all input-output combinations** (truth table)?

$$\exists i1, i2, i3, o1, o2 \text{ Signal}(In(1,C1))=i1 \wedge \text{Signal}(In(2,C1))=i2 \wedge \text{Signal}(In(3,C1))=i3 \wedge \text{Signal}(Out(1,C1))=o1 \wedge \text{Signal}(Out(2,C1))=o2$$

A (i1)	B (i2)	Cin (i3)	Sum (o1)	Cout (o2)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

LOGICAL AGENTS

FIRST-ORDER LOGIC

KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

Steps in the Knowledge Engineering Process The electronic circuits domain

7. Debug the Knowledge Base

Problem:

If we forget to say $1 \neq 0$, the system can't evaluate XOR output.

Test:

Query:

$$\exists i_1, i_2, o \text{ Signal}(In(1, X_1))=1 \wedge \text{Signal}(In(2, X_1))=0 \wedge \text{Signal}(Out(1, X_1))=o$$
Without $1 \neq 0$, XOR logic fails:
$$\text{Signal}(Out(1, X_1))=1 \Leftrightarrow 1 \neq 0$$

→ System can't infer result unless it knows that $1 \neq 0$

Fix:

Explicitly add axiom:

prolog

$$1 \neq 0$$



INFERENCE IN FIRST-ORDER LOGIC

Understanding First-Order Logic (FOL) Inference

First-Order Logic (FOL) allows reasoning about **objects**, their **properties**, and **relationships** using **quantifiers**:

- **Universal Quantifier ($\forall x$)** – “For all x...”
- **Existential Quantifier ($\exists x$)** – “There exists an x...”

FOL is more powerful than **Propositional Logic**, but also more complex when it comes to inference.



INFERENCE IN FIRST-ORDER LOGIC

INFERENCE RULES FOR QUANTIFIERS

1. Universal Instantiation (UI)

If a property is true for *all objects*, then it must be true for *any specific object*.

Formal Rule:

From

$$\forall x \alpha(x)$$

we can infer

$$\alpha(g)$$

where g is a constant (ground term).

Example:

Knowledge Base (KB):

$$\forall x (\text{King}(x) \wedge \text{Greedy}(x)) \Rightarrow \text{Evil}(x)$$

Meaning: "All greedy kings are evil."

Now, let's say we know:

$$\text{King}(\text{John}) \text{ Greedy}(\text{John})$$

Apply UI to the first sentence with x = John:

$$(\text{King}(\text{John}) \wedge \text{Greedy}(\text{John})) \Rightarrow \text{Evil}(\text{John})$$

Now we can use **Modus Ponens**:

- Premise 1: King(John)
 - Premise 2: Greedy(John)
 - Premise 3: (King(John) \wedge Greedy(John)) \Rightarrow Evil(John)
- ⇒ Conclusion: Evil(John)

Important Notes:

- UI can be applied **multiple times**, with different constants like x = Richard, x = Father(John) etc.
- It creates multiple **specific cases** from a **general rule**.



INFERENCE IN FIRST-ORDER LOGIC

INFERENCE RULES FOR QUANTIFIERS

2. Existential Instantiation (EI)

If we know **something exists**, we can give it a **temporary name** (a new constant) to work with.

Formal Rule:

From

$$\exists x \alpha(x)$$

we can infer

$$\alpha(c)$$

where c is a new constant (Skolem constant) not used anywhere else in the KB.

Example:

Given:

$$\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$$

This means: "There exists a crown on John's head."

Apply EI: introduce a new symbol C1:

$$\text{Crown}(C1) \wedge \text{OnHead}(C1, \text{John})$$

We **name** the unknown object ($\exists x$) with a **fresh constant** (C1) so we can reason about it.

Important Notes:

- The new name (e.g., C1) must be **unique** and **not conflict** with other terms in the KB.
- EI is applied **only once**, and the original \exists sentence can be **removed** from the KB.
- It is a special case of **Skolemization**.



INFERENCE IN FIRST-ORDER LOGIC

INFERENCE RULES FOR QUANTIFIERS

REDUCTION TO PROPOSITIONAL INFERENCE

Grounding First-Order Sentences

Once quantifiers are removed, all remaining statements are **ground sentences** (no variables), and we can now **treat them as propositional sentences**.

Example:

KB:

1. $\forall x (\text{King}(x) \wedge \text{Greedy}(x)) \Rightarrow \text{Evil}(x)$
2. $\text{King}(\text{John})$
3. $\text{Greedy}(\text{John})$
4. $\text{Brother}(\text{Richard}, \text{John})$

Step-by-Step Inference:

Step 1: Apply UI to sentence (1) with $x = \text{John}$ and $x = \text{Richard}$:

1a. $(\text{King}(\text{John}) \wedge \text{Greedy}(\text{John})) \Rightarrow$

$\text{Evil}(\text{John})$ 1b. $(\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard})) \Rightarrow \text{Evil}(\text{Richard})$

INFERENCE IN FIRST-ORDER LOGIC

INFERENCE RULES FOR QUANTIFIERS



Step-by-Step Inference:

Step 2: Treat atomic sentences as propositional variables:

- A = King(John)
- B = Greedy(John)
- C = Evil(John)
- D = King(Richard)
- E = Greedy(Richard)
- F = Evil(Richard)

Step 3: Use propositional logic:

- From $A \wedge B \Rightarrow C$, and A = True, B = True
 $\Rightarrow C = \text{True}$, hence: **Evil(John)**

Example:

KB:

1. $\forall x (\text{King}(x) \wedge \text{Greedy}(x)) \Rightarrow \text{Evil}(x)$
2. King(John)
3. Greedy(John)
4. Brother(Richard, John)



INFERENCE IN FIRST-ORDER LOGIC

Problem: Infinite Ground Terms

Suppose KB has a function like **Father(x)**:

• Then terms like:

Father(John)

Father(Father(John))

Father(Father(Father(John))) ...

can go on **infinitely!**

This creates **an infinite number of ground terms**, making full propositionalization **impossible** in practice.



INFERENCE IN FIRST-ORDER LOGIC

Herbrand's Theorem (1930)

If a sentence is **entailed** by a first-order KB, there exists a **finite subset** of ground instances that can prove it.

How it works:

- Generate **ground instances** from your KB:
 - Start with known constants (John, Richard)
 - Then terms of depth 1: Father(John)
 - Then depth 2: Father(Father(John))
- Try propositional reasoning at **each level**.
- If inference **succeeds**, you're done.
- If not, continue to the next level.

This gives us a **semi-decidable** method:

- If the sentence **can** be proved, we'll **eventually** find the proof.
- But if it **cannot**, we might **search forever** without knowing.



INFERENCE IN FIRST-ORDER LOGIC

Limitation: Semi-decidability

Problem:

What if the sentence is **not entailed**?

We can't be sure if:

- We just haven't gone deep enough, OR
- No proof exists

This is equivalent to the **Halting Problem**:

- If a proof exists, we'll eventually find it.
- But **no algorithm** can guarantee termination when no proof exists.

Yes-answer is possible for entailed sentences

No-answer is not guaranteed for non-entailed ones



INFERENCE IN FIRST-ORDER LOGIC

Real-World Analogy: Universal Instantiation

- “All students must submit the assignment.”
→ So, John must submit it. (Specific case from general rule)

Existential Instantiation

- “There is someone who hacked the server.”
→ Let’s call that person ‘Hacker1’ and investigate.

Propositionalization

- Turn general rules into specific facts so we can **reason like yes/no logic**.



INFERENCE IN FIRST-ORDER LOGIC

Concept	Description	Example
UI (\forall Instantiation)	Replace variable in $\forall x$ with specific constant	$\forall x P(x) \Rightarrow Q(x) \rightarrow P(\text{John}) \Rightarrow Q(\text{John})$
EI (\exists Instantiation)	Replace $\exists x$ with a new constant (Skolem constant)	$\exists x P(x) \rightarrow P(C1)$
Propositionalization	Turn FOL into propositional logic using known constants	Convert $P(x)$ to $P(\text{John})$, $P(\text{Richard})$
Herbrand's Theorem	Only finite ground instances are needed to prove entailment	Try all substitutions up to some depth
Semi-decidability	Can say YES if entailed; can't always say NO if not entailed	Infinite search if sentence isn't provable

INFERENCE IN FIRST-ORDER LOGIC

Unification and Lifting in First-Order Logic



In the early days of AI and logic-based inference, reasoning was largely done using **propositional logic**—which works only with **specific, ground facts** (facts with no variables). But this approach becomes inefficient and unnatural when dealing with **general statements** or **patterns** that apply to many entities.

Problem with Propositionalization

Take this example:

You know that **all greedy kings are evil**:

$$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$$

You also know:

King(John), Greedy(John)

Now you're asked:

Is John evil?

This is a simple conclusion for humans—but if we had to generate every possible combination of these predicates and substitute constants (e.g., King(Richard), Greedy(Richard), etc.), we'd be doing redundant work. The system would essentially recheck all irrelevant cases just to reach an obvious conclusion.



INFERENCE IN FIRST-ORDER LOGIC

Unification and Lifting in First-Order Logic

Generalized Modus Ponens (GMP)

To make inference in **First-Order Logic (FOL)** more efficient, we move to "lifted" reasoning—using **Generalized Modus Ponens**.

How GMP works:

If you have:

- 1.Facts (like King(John), Greedy(John))
- 2.A rule (like $\text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$)

Then if you can **find a substitution** (called a **unifier**) that makes the premises of the rule match your known facts, you can infer the conclusion with that same substitution.

In our example:

- Rule: $\text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
- Known: King(John), Greedy(John)
- Substitution: $\theta = \{x/\text{John}\}$
- Inferred: Evil(John)

This is GMP in action. You don't need to instantiate the rule for every person in your knowledge base—just the one that matches.

INFERENCE IN FIRST-ORDER LOGIC

Unification and Lifting in First-Order Logic

Unification: The Heart of Lifted Inference

Unification is the process of finding a substitution θ that makes two expressions identical.

What does UNIFY do?

Given two logical expressions (**e.g., Knows(John, x) and Knows(John, Jane)**), the **UNIFY algorithm** tries to make them look the same by replacing variables with terms or constants.

Examples:

1. UNIFY(**Knows(John, x)**, **Knows(John, Jane)**) $\rightarrow \{x/Jane\}$
 2. UNIFY(**Knows(John, x)**, **Knows(y, Bill)**) $\rightarrow \{x/Bill, y/John\}$
 3. UNIFY(**Knows(John, x)**, **Knows(y, Mother(y))**) $\rightarrow \{x/Mother(John), y/John\}$
 4. UNIFY(**Knows(John, x)**, **Knows(x, Elizabeth)**) $\rightarrow \text{fails}$

The last case fails because it implies $x = \text{John}$ and $x = \text{Elizabeth}$ at the same time.

Standardizing Apart

When two clauses use the **same variable name** but refer to **different things**, it causes conflicts. We solve this by **renaming variables** in one clause before unification. This is called **standardizing apart**.

Example:

- Rename Knows(x, Elizabeth) to Knows(x17, Elizabeth)
 - Then UNIFY(Knows(John, x), Knows(x17, Elizabeth)) \rightarrow {x/Elizabeth, x17/John}



INFERENCE IN FIRST-ORDER LOGIC

Unification and Lifting in First-Order Logic

Most General Unifier (MGU)

When two expressions can be unified in multiple ways, the **Most General Unifier** (MGU) is the one that places the **fewest constraints**.

For instance:

- UNIFY(Knows(John, x), Knows(y, z)) could give:
 - $\{y/John, x/z\} \rightarrow \text{most general}$
 - $\{y/John, x/John, z/John\} \rightarrow \text{less general}$

The MGU helps make the reasoning system more **flexible** and **efficient**, avoiding overly specific cases.



INFERENCE IN FIRST-ORDER LOGIC

Unification and Lifting in First-Order Logic

The UNIFY Algorithm

```
function UNIFY(x, y,  $\theta$ ) returns a substitution to make x and y identical
  inputs: x, a variable, constant, list, or compound expression
         y, a variable, constant, list, or compound expression
          $\theta$ , the substitution built up so far (optional, defaults to empty)

  if  $\theta = \text{failure}$  then return failure
  else if x = y then return  $\theta$ 
  else if VARIABLE?(x) then return UNIFY-VAR(x, y,  $\theta$ )
  else if VARIABLE?(y) then return UNIFY-VAR(y, x,  $\theta$ )
  else if COMPOUND?(x) and COMPOUND?(y) then
    return UNIFY(x.ARGS, y.ARGS, UNIFY(x.OP, y.OP,  $\theta$ ))
  else if LIST?(x) and LIST?(y) then
    return UNIFY(x.REST, y.REST, UNIFY(x.FIRST, y.FIRST,  $\theta$ ))
  else return failure
```

function UNIFY-VAR(*var*, *x*, θ) returns a substitution

```
if {var/val}  $\in \theta$  then return UNIFY(val, x,  $\theta$ )
else if {x/val}  $\in \theta$  then return UNIFY(var, val,  $\theta$ )
else if OCCUR-CHECK?(var, x) then return failure
else return add {var/x} to  $\theta$ 
```

The algorithm works recursively and checks whether:

1. The expressions are identical — if yes, done.
2. One is a variable — try to bind it.
3. Both are compound — unify their operators and arguments.
4. Otherwise — fail.

One critical step is the **occur check**:

- Prevents a variable from being unified with a structure that contains itself (e.g., *x* with *f(x)*) → would lead to circular, invalid substitutions.
- Some systems skip this check for performance, risking incorrect results.



INFERENCE IN FIRST-ORDER LOGIC

Unification and Lifting in First-Order Logic

Storage and Retrieval in Knowledge Bases

Inference systems don't just unify on demand—they need efficient **retrieval** of relevant facts. Two basic functions support this:

- STORE(s)**: Adds fact s to the knowledge base.
- FETCH(q)**: Finds all facts in the knowledge base that **unify** with q.

How to make FETCH fast:

Instead of scanning everything, facts can be **indexed**:

- By **predicate**: All Knows facts in one bucket.
- By **argument position**: e.g., index Employs(x, y) by both x and y.

Subsumption Lattice

Every fact (like Employs(IBM, Richard)) can match many queries:

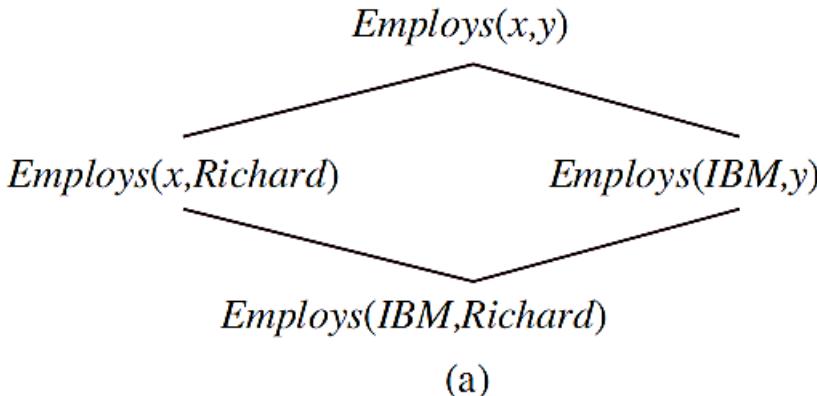
- Employs(IBM, Richard)
- Employs(x, Richard)
- Employs(IBM, y)
- Employs(x, y)

These form a **subsumption lattice**—a structure where each level generalizes the one below.

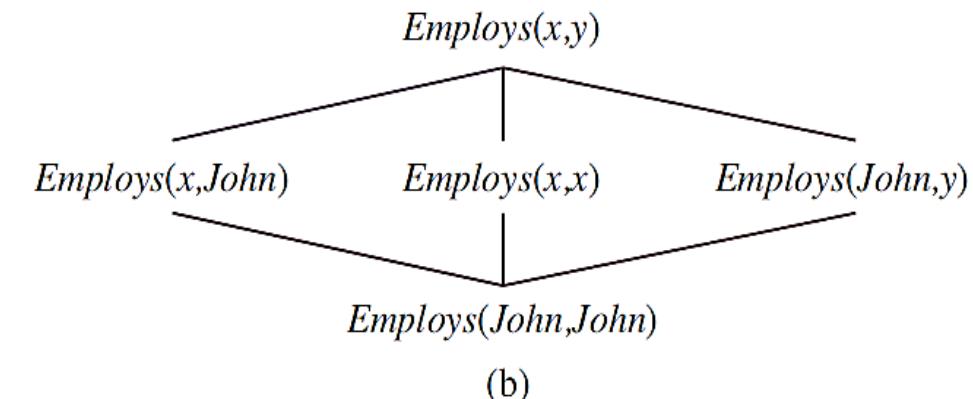


INFERENCE IN FIRST-ORDER LOGIC

Unification and Lifting in First-Order Logic



(a) The subsumption lattice whose lowest node is Employs(IBM, Richard).



(b) The subsumption lattice for the sentence Employs (John, John).



INFERENCE IN FIRST-ORDER LOGIC

Unification and Lifting in First-Order Logic

Key Advantages of Lifting and Unification

- 1. Avoids full propositional expansion** — much more efficient.
- 2. Works with variables directly**, allowing general rules.
- 3. Enables reuse of facts and rules** with minimal substitutions.
- 4. Forms the foundation of logical inference in AI** — including Prolog, logic programming, resolution-based theorem proving, and automated reasoning systems.



INFERENCE IN FIRST-ORDER LOGIC

Forward Chaining

- **Forward chaining** is an inference method that starts from **known facts** and applies **rules** to deduce **new facts**.
- It proceeds in a **data-driven** manner: as new data arrives, the system tries to draw more conclusions.
- It stops when:
 - The **goal/query** is proven, or
 - **No more new facts** can be inferred (i.e., it reaches a **fixed point**).



INFERENCE IN FIRST-ORDER LOGIC

First-Order Definite Clauses

In FOL, definite clauses are:

- **Atomic facts** (e.g., King(John))
- Or **Implications** (rules) of the form:

$$A_1 \wedge A_2 \wedge \dots \wedge A_n \Rightarrow B$$

Where:

- A_1, \dots, A_n and B are **positive literals** (no negation).
- Variables are **implicitly universally quantified** (i.e., "for all x ...").

Examples:

- King(x) \wedge Greedy(x) \Rightarrow Evil(x)
- Missile(x) \Rightarrow Weapon(x)
- American(West)



INFERENCE IN FIRST-ORDER LOGIC

The Crime Problem

Problem Statement:

"It is a crime for an American to sell weapons to hostile nations. Nono is a hostile nation. Colonel West is an American who sold missiles to Nono."

Step 1: Encode the facts as **definite clauses**

English Statement	First-Order Clause
It's a crime...	$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$
Nono owns a missile	$\text{Owns}(\text{Nono}, M1) \wedge \text{Missile}(M1)$
All missiles owned by Nono were sold by West	$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$
Missiles are weapons	$\text{Missile}(x) \Rightarrow \text{Weapon}(x)$
Enemies of America are hostile	$\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$
West is American	$\text{American}(\text{West})$
Nono is enemy of America	$\text{Enemy}(\text{Nono}, \text{America})$

INFERENCE IN FIRST-ORDER LOGIC



Step 2: Apply Forward Chaining Iteratively

Iteration 1:

From these facts:

- **Missile(M1)**
- **Owns(Nono, M1)**
- **Enemy(Nono, America)**

Apply:

1. **Missile(x) \Rightarrow Weapon(x) \rightarrow Add Weapon(M1)**
2. **Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono) \rightarrow Add Sells(West, M1, Nono)**
3. **Enemy(x, America) \Rightarrow Hostile(x) \rightarrow Add Hostile(Nono)**

Iteration 2:

Apply:

- **American(West)**
- **Weapon(M1)**
- **Sells(West, M1, Nono)**
- **Hostile(Nono)**

To:

- **American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)**

With substitution:

- **{x = West, y = M1, z = Nono}**
- Add: **Criminal(West)**

Final Output:

Goal Criminal(West) is derived through forward chaining.



INFERENCE IN FIRST-ORDER LOGIC

FOL-FC-ASK Algorithm

```
function FOL-FC-ASK(KB, query  $\alpha$ ):  
    repeat until no new facts are added:  
        for each rule  $(p_1 \wedge \dots \wedge p_n \Rightarrow q)$ :  
            for each substitution  $\theta$  where  $\{p_1, \dots, p_n\}$  unify  
            with known facts:  
                infer  $q\theta$   
                if  $q\theta$  is new, add it to KB  
                if  $q\theta$  unifies with query  $\alpha$ , return  $\theta$   
    return false
```



INFERENCE IN FIRST-ORDER LOGIC

FOL-FC-ASK Algorithm

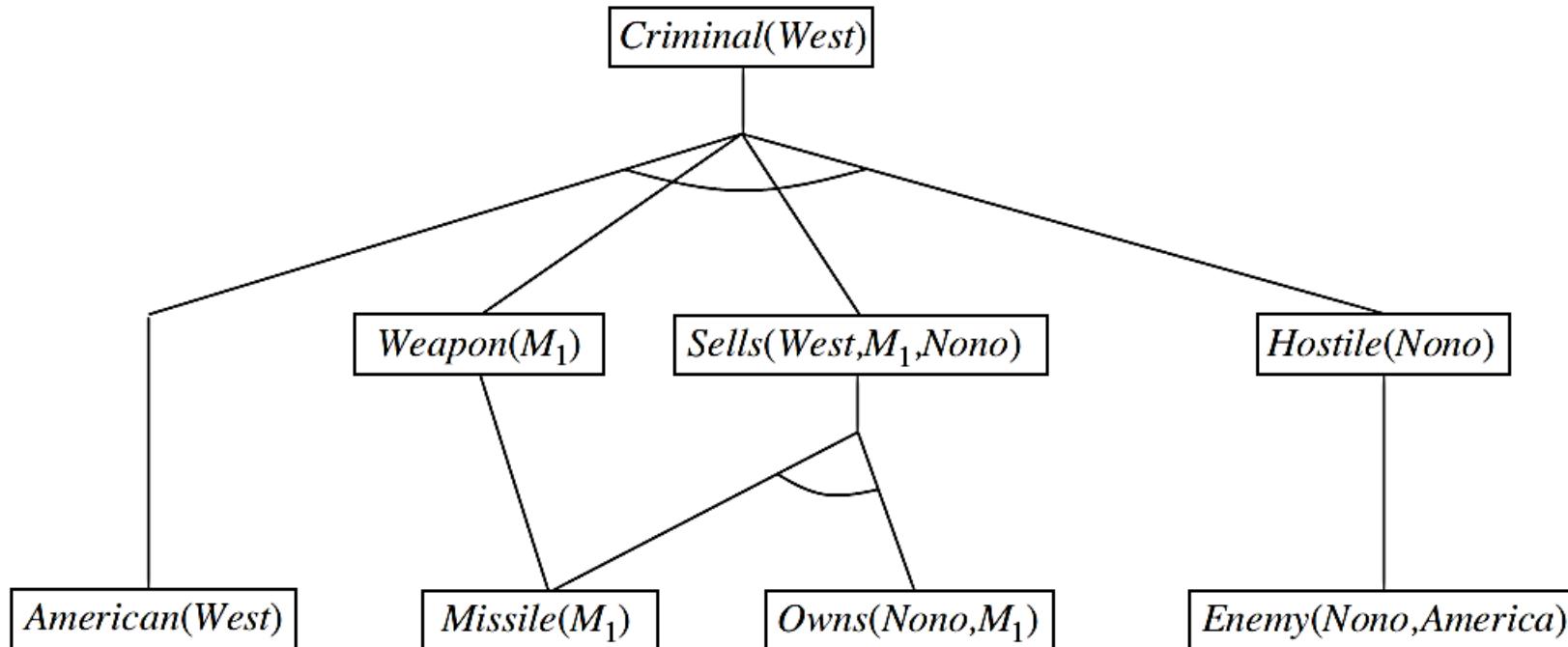
```
function FOL-FC-Ask( $KB, \alpha$ ) returns a substitution or false
  inputs:  $KB$ , the knowledge base, a set of first-order definite clauses
          $\alpha$ , the query, an atomic sentence
  local variables:  $new$ , the new sentences inferred on each iteration

  repeat until  $new$  is empty
     $new \leftarrow \{ \}$ 
    for each rule in  $KB$  do
       $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-VARIABLES}(rule)$ 
      for each  $\theta$  such that  $\text{SUBST}(\theta, p_1 \wedge \dots \wedge p_n) = \text{SUBST}(\theta, p'_1 \wedge \dots \wedge p'_n)$ 
        for some  $p'_1, \dots, p'_n$  in  $KB$ 
           $q' \leftarrow \text{SUBST}(\theta, q)$ 
          if  $q'$  does not unify with some sentence already in  $KB$  or  $new$  then
            add  $q'$  to  $new$ 
             $\phi \leftarrow \text{UNIFY}(q', \alpha)$ 
            if  $\phi$  is not fail then return  $\phi$ 
    add  $new$  to  $KB$ 
  return false
```



INFERENCE IN FIRST-ORDER LOGIC

FOL-FC-ASK Algorithm



The proof tree generated by forward chaining on the crime example. The initial facts appear at the bottom level, facts inferred on the first iteration in the middle level, and facts inferred on the second iteration at the top level.



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Backward chaining is a **goal-driven inference** method used in logic-based systems. It starts with a **goal (query)** and works **backward** through inference rules to determine if the goal can be derived from known facts.

- It is a **depth-first search** algorithm.
- It starts from the **goal** (what you want to prove) and works backward by finding rules that could lead to that goal.
- If a rule leads to the goal, it tries to prove the **premises** (conditions) of that rule.



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

2. Structure of a Rule (Definite Clause)

Each rule is of the form:

premise1 \wedge premise2 \wedge ... \wedge premisen \Rightarrow conclusion

For example:

Missile(x) \wedge Owns(Nono, x) \wedge American(West) \wedge Sells(West, x, Nono) \wedge Hostile(Nono) \Rightarrow Criminal(West)

To **prove** Criminal(West), we must **prove all** premises on the left side.



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

3. Simple Real-World Analogy

Imagine trying to prove:

"John is guilty."

To prove this, you check:

- **Was there a crime?**
- **Did John commit it?**
- **Is there evidence?**

Each check may require **more checks**, until you reach known facts (e.g., CCTV footage, fingerprint matches).



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

4. Step-by-Step Example: Proving Criminal(West)

Goal: Prove Criminal(West)

Rule:

$$\text{Missile}(y) \wedge \text{Owns}(\text{Nono}, y) \wedge \text{Sells}(\text{West}, y, \text{Nono}) \wedge \\ \text{American}(\text{West}) \wedge \text{Hostile}(\text{Nono}) \Rightarrow \text{Criminal}(\text{West})$$

To prove Criminal(West), we must prove:

1. Missile(y)
2. Owns(Nono, y)
3. Sells(West, y, Nono)
4. American(West)
5. Hostile(Nono)

5. Facts in Knowledge Base (KB):

- Missile(M1)
- Owns(Nono, M1)
- Sells(West, M1, Nono)
- American(West)
- Enemy(Nono, America)
- $\forall x (\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x))$

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING



6. How Backward Chaining Works

It uses two functions:

a. **FOL-BC-ASK(KB, query):**

- Tries to prove query from the KB.
- Starts the process with the main goal.

b. **FOL-BC-OR(KB, goal, θ):**

- Tries to prove the goal using **any matching rule**.
- For each rule with conclusion matching the goal, try to prove all the premises (lhs).
- Uses **unification** to match variables.

c. **FOL-BC-AND(KB, goals, θ):**

- Tries to prove a **conjunction** of subgoals.
- Proves one subgoal at a time, carrying forward the **substitutions** from unifications.

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

```
function FOL-BC-Ask( $KB$ ,  $query$ ) returns a generator of substitutions
  return FOL-BC-Or( $KB$ ,  $query$ , { })
```

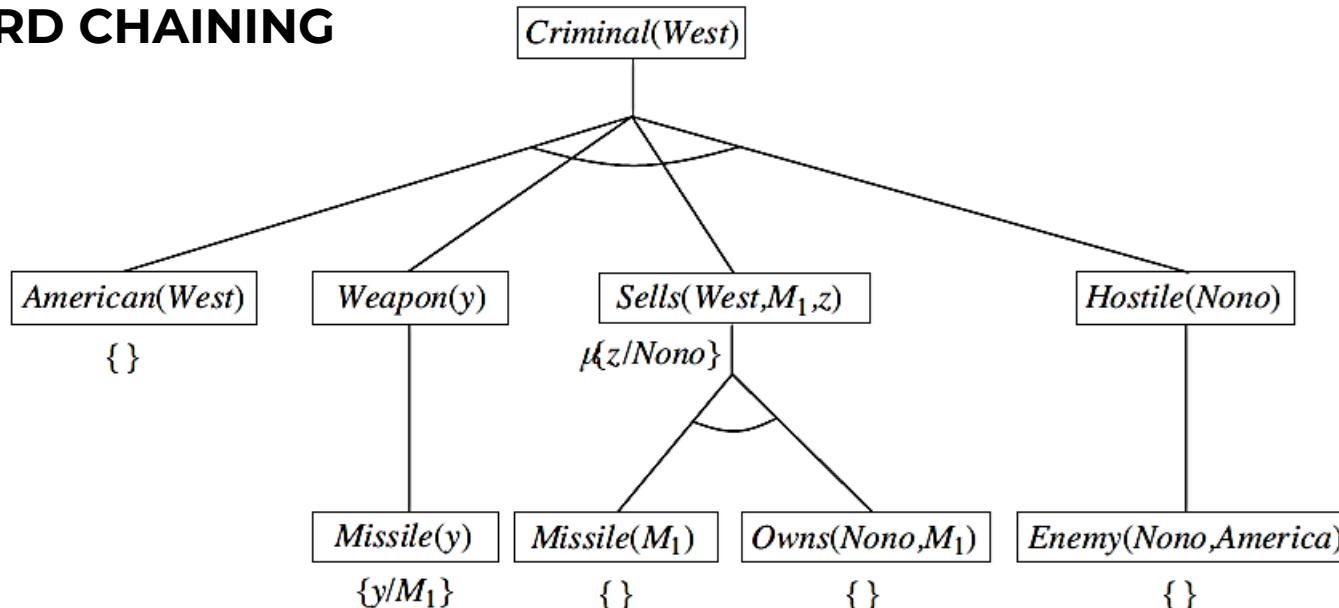
```
generator FOL-BC-Or( $KB$ ,  $goal$ ,  $\theta$ ) yields a substitution
  for each rule  $(lhs \Rightarrow rhs)$  in FETCH-RULES-FOR-GOAL( $KB$ ,  $goal$ ) do
     $(lhs, rhs) \leftarrow$  STANDARDIZE-VARIABLES( $(lhs, rhs)$ )
    for each  $\theta'$  in FOL-BC-And( $KB$ ,  $lhs$ , UNIFY( $rhs$ ,  $goal$ ,  $\theta$ )) do
      yield  $\theta'$ 
```

```
generator FOL-BC-And( $KB$ ,  $goals$ ,  $\theta$ ) yields a substitution
  if  $\theta = \text{failure}$  then return
  else if LENGTH( $goals$ ) = 0 then yield  $\theta$ 
  else do
    first, rest  $\leftarrow$  FIRST( $goals$ ), REST( $goals$ )
    for each  $\theta'$  in FOL-BC-Or( $KB$ , SUBST( $\theta$ , first),  $\theta$ ) do
      for each  $\theta''$  in FOL-BC-And( $KB$ , rest,  $\theta'$ ) do
        yield  $\theta''$ 
```



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING



```

Criminal(West)
  └─ Missile(y)      ← Matches Missile(M1) → {y/M1}
  └─ Owns(Nono, y)   ← Becomes Owns(Nono, M1)
  └─ Sells(West, y, Nono)   ← Becomes Sells(West, M1,
    Nono)
    └─ American(West)   ← Already in KB
    └─ Hostile(Nono)    ← Need to prove this
      └─ Enemy(Nono, America)   ← Already in KB
      └─ Rule: Enemy(x, America) ⇒ Hostile(x)
  
```

Thus, by **resolving subgoals one by one**, and applying **substitutions**, we eventually prove the main goal.

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

8. Advantages of Backward Chaining

- Efficient when:
 - The number of **possible facts is large.**
 - You have a **specific goal** in mind.
- Works **backwards**, so avoids deriving unnecessary facts.

9. Limitations

- **Repeated States:** May revisit same subgoal multiple times.
- **Incompleteness:** May get stuck in **infinite loops** if not controlled.
- Not suitable when **all possible outcomes** are needed (use forward chaining instead).

10. Solutions to Limitations

- Use **memoization** to store intermediate results.
- Apply **loop detection** to avoid infinite recursion.
- Use **breadth-first** or **iterative deepening** when completeness is critical.



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Logic Programming

- A paradigm where **knowledge** is expressed using **logical rules**, and **inference** is used to solve problems.

- Embodies the idea:

“Algorithm = Logic + Control” (Robert Kowalski)

- **Logic:** What we want to say (knowledge/rules)
- **Control:** How we compute answers (inference strategy)

Most Popular Logic Programming Language: Prolog

- Used in:

- **Expert systems** (medical, legal)
- **Natural language processing**
- **Symbolic computation**
- **Rapid prototyping**



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Logic Programming

Syntax of Prolog vs First-Order Logic

First-Order Logic	Prolog
$A \wedge B \Rightarrow C$	$C :- A, B.$
Uppercase = constants	Uppercase = variables
Lowercase = variables	Lowercase = constants

Prolog

criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).

Means:

If X is an American, Y is a weapon, X sells Y to Z, and Z is hostile → then X is a criminal.



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

4. Prolog is Based on Backward Chaining

- Uses **depth-first search** from **goal (query)** backwards.
- Clauses are **tried in order** as written in the program.
- Uses **unification** to match query with rules/facts.

5. Prolog Example: List Appending

```
append([], Y, Y). append([A|X], Y, [A|Z]) :- append(X, Y, Z).
```

Meaning in natural language:

1. Appending an empty list [] with a list Y gives Y.
2. Appending [A|X] to Y gives [A|Z], if appending X and Y gives Z.

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

6. Power of Relations in Prolog

• Prolog doesn't just compute functions – it expresses **relations**.

Example Query:

?- append(X, Y, [1,2]).

Asks: What X and Y make up the list [1,2]?

Prolog answers:

X = [], Y = [1,2];
X = [1], Y = [2];
X = [1,2], Y = [].

This **relational power** allows **multi-directional reasoning**, unlike regular functions.



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

7. Limitations & Differences from First-Order Logic

Feature	Explanation
Not pure FOL	Prolog uses database semantics (facts are added/removed), not strict model-theoretic logic
Arithmetic	Uses built-in evaluation ($X \text{ is } 4+3 \rightarrow X = 7$), but not equation solving
Side Effects	Predicates like assert or write can modify state or output— not logically pure
Unification (no occur check)	Omits check to avoid cyclic structures; small chance of unsound results
Incomplete	No check for infinite recursion (e.g., $p(X) :- p(X).$) can loop forever
Fast	Very fast if given right rules and order , due to depth-first control

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Sample Use Case: Proving Criminal(West)

```
missile(m1).  
owns(nono, m1).  
sells(west, m1, nono).  
american(west).  
enemy(nono, america).  
hostile(X) :- enemy(X, america).  
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z),  
hostile(Z).  
weapon(Y) :- missile(Y).
```

Query:

```
?- criminal(west).
```

Output:

```
true.
```

How it works (Backward Chaining):

1. criminal(west) ← check if american(west), etc.
2. Resolves each condition backward using known facts & rules.

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Why Use Logic Programming?

- Focus is on **what** to solve, not **how** to solve it.
- Very useful in domains where **rules and knowledge** are central (e.g., law, diagnostics).
- Easy to write and update knowledge—**modular & declarative**.

10. Kowalski's Equation: Algorithm = Logic + Control

Component	Role
Logic	Set of rules and facts (what is true?)
Control	Inference mechanism (how to apply?)
In Prolog	Logic = Program clauses, Control = Backward chaining search



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Efficient Implementation of Logic Programs

1. Execution Modes: Interpreted vs Compiled

Mode	Description
Interpreted	Runs directly using an algorithm like FOL-BC-ASK (backward chaining).
Compiled	Translates Prolog code into efficient machine-level or intermediate code to reduce overhead.

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Efficient Implementation of Logic Programs

2. Optimizations in Prolog Interpreters

a. Choice Point Stack

- Backward chaining may yield **multiple possible answers**.
- Prolog keeps track of these using a **global choice point stack**.
- Helps in **efficient backtracking** and **debugging**.

When one path fails, Prolog pops the stack and **tries the next option**.

b. Logic Variables with Implicit Substitutions

• Variables in Prolog are **bound during execution**, and their bindings **represent substitutions**.

• **Unification** either:

- Binds a variable (if it's unbound), or
- Fails (if already bound inconsistently).

On failure, variable bindings are **reversed** using a **trail**.

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING



Efficient Implementation of Logic Programs

3. Trail Stack

- Records all variable bindings made during proof.
- If a clause fails, **RESET-TRAIL** unbinds variables up to a choice point.

4. Why Interpreters Are Slower

- Every inference:
 - Re-analyzes structure.
 - Searches for matching clauses.
 - Builds recursive call stacks.
- Hence, even optimized interpreters need **thousands of machine instructions** per inference step.

5. Compiled Prolog: Speeding Things Up

Compiled Prolog:

- **Generates specialized code** for each predicate.
- **Knows in advance** which clauses to try — no need to search.
- Can **open-code unification**, avoiding runtime term inspection.

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Efficient Implementation of Logic Programs

6. The Warren Abstract Machine (WAM)

Feature	Description
What is WAM?	A virtual machine optimized for Prolog execution.
Why WAM?	Real machine architectures are not a good fit for Prolog's logic-driven style.
How it helps?	WAM serves as a bridge to efficient execution: either interpreted or compiled to machine code.

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Efficient Implementation of Logic Programs

7. Append Predicate Compilation

```
append([], Y, Y).  
append([A|X], Y, [A|Z]) :- append(X, Y, Z).
```

pseudocode

```
procedure APPEND(ax, y, az, continuation)  
    trail  $\leftarrow$  GLOBAL-TRAIL-POINTER()  
    if ax = [] and UNIFY(y, az) then CALL(continuation)  
    RESET-TRAIL(trail)  
    a, x, z  $\leftarrow$  NEW-VARIABLE(), NEW-VARIABLE(), NEW-VARIABLE()  
    if UNIFY(ax, [a | x]) and UNIFY(az, [a | z]) then APPEND(x, y, z, continuation)
```

• **trail**: Saves state to undo bindings if needed.

• **continuation**: Represents “what to do next” after success (like a saved function pointer).

• **UNIFY**: Matches terms; binds variables if needed.

• **RESET-TRAIL**: Undoes changes on backtracking.

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Efficient Implementation of Logic Programs

8. Continuations in Prolog

- Instead of just “returning” from a successful predicate, Prolog **packs the next goal** into a **continuation**.
- Allows **multiple solutions** to be explored after success.
If a goal succeeds in multiple ways, **each success calls the continuation** to move forward.

9. Parallel Execution in Logic Programming

Type	Description
OR-parallelism	Different rules can prove a goal. Each can be run in parallel .
AND-parallelism	Conjuncts in the body of a rule can be solved in parallel (harder to implement).

In AND-parallelism, all variable bindings must be **consistent** across branches, which requires **synchronization**.

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Efficient Implementation of Logic Programs

Summary: Prolog Append Performance

·Interpreted Prolog:

- Re-analyzes append clauses on every call.
- Binds and unbinds variables dynamically.

·Compiled Prolog:

- Transforms append into a dedicated procedure.
- No clause search—just direct calls.
- Uses **WAM**, **trail**, and **continuations** for speed.

11. Real-World Significance

- Compiled Prolog with WAM made Prolog **competitive with C** in speed.
- This enabled its widespread use in AI, NLP, and **expert systems**.
- Its **declarative + control model** made Prolog ideal for **small-scale AI research**.

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Efficient Implementation of Logic Programs

Traditional Approach	Prolog Advantage
Procedural logic (C, Python)	Must tell how to compute
Logic programming	Just say what is true (rules + facts)
Inefficient backtracking	Compiled Prolog reduces overhead
No built-in inference	Built-in goal resolution & unification
No relational queries	Easily solve for multiple unknowns



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Redundant Inference and Infinite Loops

The Problem: Depth-First Search Isn't Always Smart

Prolog uses **depth-first backward chaining** by default. While efficient in some cases, it has two **major flaws**:

1. **Infinite Loops** from cyclic rules or incorrect rule ordering.
2. **Redundant Inference**, repeatedly recomputing the same subgoals unnecessarily.

Example: Pathfinding in a Graph

```
path(X,Z) :- link(X,Z).  
path(X,Z) :- path(X,Y), link(Y,Z).
```

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

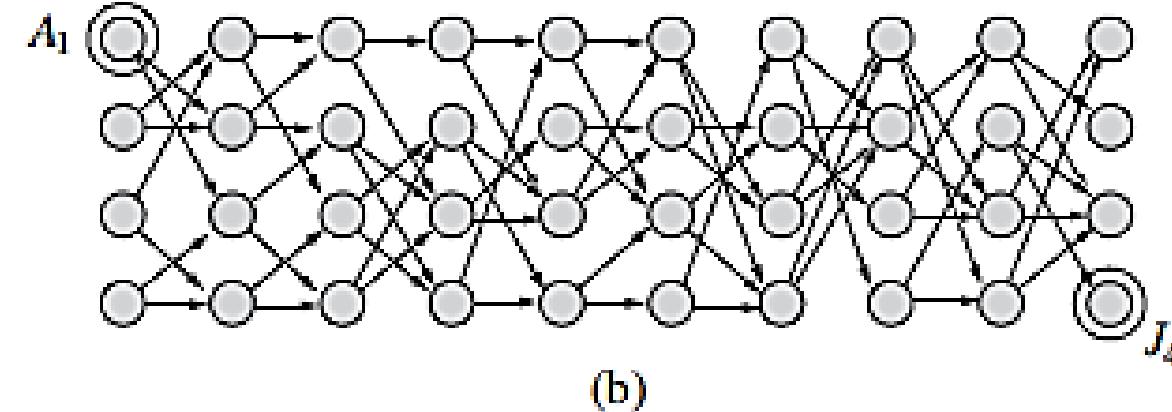
Redundant Inference and Infinite Loops



(a)

`link(a, b).
link(b, c).`

Query:
`path(a, c)`
Rule order:
Correct



(b)

If you try `path(a, c)` using:

`path(X,Z) :- link(X,Z).
path(X,Z) :- path(X,Y), link(Y,Z).`

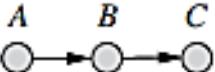
**path(a,b) via link(a,b)
path(b,c) via link(b,c)
=> path(a,c)**

Success!

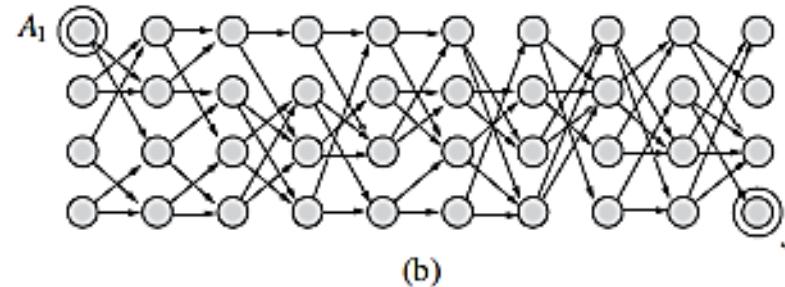
INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Redundant Inference and Infinite Loops



(a)



(b)

Now switch the clause order:

link(a, b). This loops forever.

link(b, c).

path(X,Z) :- path(X,Y), link(Y,Z).

path(X,Z) :- link(X,Z).

Query:

path(a, c)

Rule order:

ICorrect

Now, on trying to find $\text{path}(a, c)$, Prolog first tries to recursively evaluate $\text{path}(a, Y)$ before ever checking $\text{link}(a, Z)$. This creates an **infinite recursive loop** like:

path(a, c)

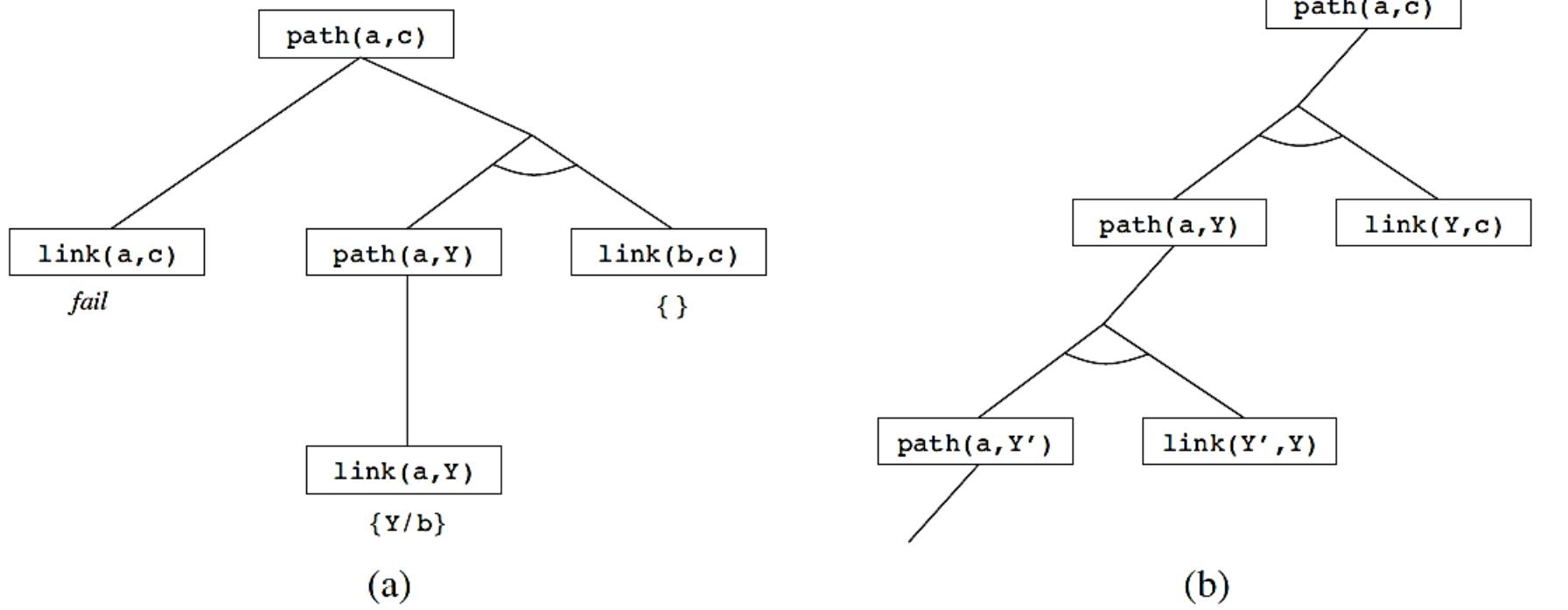
→ **path(a, Y), link(Y, c)**

→ **path(a, Y1), link(Y1, Y)**

→ **path(a, Y2), link(Y2, Y1)**

→ ...

This loops forever





INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Redundant Inference and Infinite Loops

Why Does This Happen?

- **Depth-first search** explores one path deeply before trying alternatives.
- If the **first path leads to an infinite branch**, it never tries the second clause.

That's why **Prolog is incomplete**: It **might not find a valid answer even if one exists**.

In larger graphs, Prolog might explore the **same subgoal multiple times**, wasting computation.

For example, in the graph in **Figure 9.9(b)** with nodes from A1 to J4:

- The graph has multiple layers.
- Each node connects to two random successors in the next layer.
- Prolog might try **877 inferences** to find path(A1, J4), many of which go to dead ends.

In contrast, **forward chaining** (or dynamic programming) solves this in only **62 inferences**.

This loops forever



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Redundant Inference and Infinite Loops

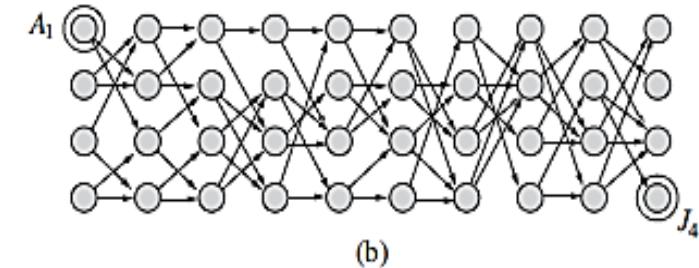
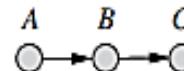
Redundant Inference

In larger graphs, Prolog might explore the **same subgoal multiple times**, wasting computation.

For example, in the graph in **Figure (b)** with nodes from A1 to J4:

- The graph has multiple layers.
- Each node connects to two random successors in the next layer.
- Prolog might try **877 inferences** to find path(A1, J4), many of which go to dead ends.

In contrast, **forward chaining** (or dynamic programming) solves this in only **62 inferences**.



Forward Chaining = Dynamic Programming

- In forward chaining, we **build up solutions** from known facts.
- This is similar to **dynamic programming**:
 - Avoid re computation.
 - Cache intermediate results.
 - Once you infer path(X,Y), it's stored and never recomputed.

This loops forever



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Redundant Inference and Infinite Loops

Memoization to the Rescue: Tabled Logic

Programming

- **Memoization:** Cache answers to subgoals.
- If the same subgoal arises again → just **reuse** the result.
- **Tabled logic programming** adds this feature to Prolog.

Benefits:

- Avoids redundant computation.
- Avoids infinite loops **caused by cycles**.
- Guarantees **completeness** for **Datalog** programs.

Practical Systems Using Tabled Logic

Some Prolog implementations support tabling:

System	Supports Tabled Logic?
XSB Prolog	Yes
SWI-Prolog	Yes (via library(tabling))
YAP Prolog	Yes



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Redundant Inference and Infinite Loops

Technique	Pros	Cons
Depth-First Backward Chaining (Prolog default)	Efficient memory, goal-directed	Can loop infinitely, redundant inference
Forward Chaining	No loops, good for dynamic programming	Less goal-directed, may infer irrelevant facts
Tabled Logic Programming	Combines best of both: complete + efficient	Slightly more complex, higher memory usage

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Database Semantics of Prolog

In Prolog, logic programs operate under two key assumptions

1. Unique Names Assumption (UNA)

Each constant or ground term refers to a **different object**.

Course(CS, 101).

Course(EE, 101).

Means CS ≠ EE, even though the number 101 appears in both.



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Database Semantics of Prolog

2. Closed World Assumption (CWA)

Only facts **explicitly stated** or **logically entailed** by the program are considered true.

Everything else is **assumed false** by default.

`Course(CS, 101).`

`Course(CS, 102).`

`Course(CS, 106).`

`Course(EE, 101).`

Under Prolog semantics:

- There are **exactly 4 courses** (by CWA).
- All constants (CS, EE, 101, 102, 106) are **distinct** (by UNA).

In contrast, in First-Order Logic (FOL):

- We can't assume these are the **only** courses unless we **explicitly** say so.
- We also can't assume constants like CS and EE are different unless we add axioms like $CS \neq EE$.

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Database Semantics of Prolog

Translating to First-Order Logic (FOL)

To make FOL behave like Prolog, we'd need **completion**:

$$\begin{aligned}\text{Course}(d, n) \Leftrightarrow \\ (\text{d=CS} \wedge \text{n}=101) \vee \\ (\text{d=CS} \wedge \text{n}=102) \vee \\ (\text{d=CS} \wedge \text{n}=106) \vee \\ (\text{d=EE} \wedge \text{n}=101)\end{aligned}$$

This restricts the meaning of $\text{Course}(d, n)$ to exactly those 4 pairs.

Completion of equality ($x = y$):

We must **manually state** that constants are distinct:

$$\begin{aligned}x = y \Leftrightarrow \\ (\text{x=CS} \wedge \text{y=CS}) \vee \\ (\text{x=EE} \wedge \text{y=EE}) \vee \\ (\text{x}=101 \wedge \text{y}=101) \vee \dots\end{aligned}$$

INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Database Semantics of Prolog

Why Use Database Semantics?

- Much more **efficient** than full FOL.
- Works well for domains that resemble **relational databases**.
- No need to explicitly state non-existence or inequality.





INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Constraint Logic Programming (CLP)

From Logic Programming to Constraints

Standard Prolog uses **backtracking** to search for solutions to goals.

However, this works **only if** the domain of each variable is **finite**.
Why? Because Prolog tries to **enumerate all possible values**.

Example: The Triangle Inequality

```
triangle(X, Y, Z) :-  
    X > 0, Y > 0, Z > 0,  
    X + Y >= Z,  
    Y + Z >= X,  
    X + Z >= Y.
```

Query 1:

```
?- triangle(3, 4, 5).
```

Success.

Query 2:

```
?- triangle(3, 4, Z).
```

Fails in **standard Prolog**, because:

$Z \geq 0$ cannot be evaluated if Z is **unbound**.



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Constraint Logic Programming (CLP)

Solution: Constraint Logic Programming

CLP allows variables to be **constrained** rather than bound to specific values.

So instead of saying:

$$Z = 5$$

We can say:

CLP **returns constraints**, not just bindings.

Example Outcome in CLP:

?- triangle(3, 4, Z).

Returns:

$$1 \leq Z \leq 7$$



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Constraint Logic Programming (CLP)

CLP = Logic Programming + Constraint Solving

CLP languages include solvers for:

Type of Constraint	Solver Used
Linear inequalities (e.g., $X + Y \leq 5$)	Linear programming
Boolean constraints	SAT solvers
Finite domain (e.g., Sudoku)	Backtracking + propagation

Flexible Search Strategies in CLP

Unlike standard Prolog (which uses depth-first, left-to-right backtracking), CLP systems can use:

- **Heuristic-based search**
- **Backjumping**
- **Constraint propagation**
- **Metarules** (e.g., try conjuncts with fewer variables first)



INFERENCE IN FIRST-ORDER LOGIC

BACKWARD CHAINING

Constraint Logic Programming (CLP)

Advanced Control: Metarules

Some CLP languages (like MRS) allow custom search control:

Try the conjunct with the fewest variables first.

Or domain-specific strategies:

Try color assignment before constraint checking in map coloring.

Feature	Standard Prolog	Constraint Logic Programming (CLP)
Variable Binding	Equality only (e.g., $X = 3$)	Arbitrary constraints (e.g., $X > 0$, $X + Y = 5$)
Infinite Domains	Not supported	Supported
Expressiveness	Limited	Much broader
Solver Strategy	Backtracking	Constraint solvers (e.g., LP, CSP)
Flexibility	Rigid control flow	Customizable with metarules



INFERENCE IN FIRST-ORDER LOGIC

Resolution in FOL

Resolution in **First-Order Logic** is an inference rule used to **derive new information** or **prove something by contradiction**. It works by:

1. **Converting all statements** into **clause form** (CNF - Conjunctive Normal Form).
2. **Unifying** variables (using substitutions) to match literals.
3. **Resolving** pairs of clauses with complementary literals.
4. **Deriving a contradiction (empty clause \square)** to prove something.

Steps in Resolution (FOL):

1. **Convert all sentences** to **CNF**.
2. **Negate** the statement to be proven.
3. **Add** the negated goal to the knowledge base.
4. **Resolve** clauses repeatedly using **unification**.
5. If you get an **empty clause \square** , you've found a **contradiction**, so the original statement is true.



INFERENCE IN FIRST-ORDER LOGIC

Prove "John is a criminal."

Given:

1. Everyone who sells an illegal weapon is a criminal.

$\forall x (\text{Illegal}(x) \wedge \text{Sells}(\text{John}, x) \rightarrow \text{Criminal}(\text{John}))$

2. Guns are illegal.

$\text{Illegal}(\text{Gun})$

3. John sells a gun.

$\text{Sells}(\text{John}, \text{Gun})$

4. Goal to prove: $\text{Criminal}(\text{John})$



INFERENCE IN FIRST-ORDER LOGIC

Prove "John is a criminal."

Step 1: Convert to CNF

(1)

$\forall x (\neg \text{Illegal}(x) \vee \neg \text{Sells}(\text{John}, x) \vee \text{Criminal}(\text{John}))$

(2)

$\text{Illegal}(\text{Gun})$

(3)

$\text{Sells}(\text{John}, \text{Gun})$

Negate goal (4):

$\neg \text{Criminal}(\text{John})$

Now knowledge base:

- Clause 1: $\neg \text{Illegal}(x) \vee \neg \text{Sells}(\text{John}, x) \vee \text{Criminal}(\text{John})$
- Clause 2: $\text{Illegal}(\text{Gun})$
- Clause 3: $\text{Sells}(\text{John}, \text{Gun})$
- Clause 4: $\neg \text{Criminal}(\text{John}) \leftarrow$ added negated goal



INFERENCE IN FIRST-ORDER LOGIC

Prove "John is a criminal."

Step 2: Apply Resolution

From (1) and (4):

Clause 1: $\neg\text{Illegal}(x) \vee \neg\text{Sells}(\text{John}, x) \vee \text{Criminal}(\text{John})$

Clause 4: $\neg\text{Criminal}(\text{John})$

→ Resolves on $\text{Criminal}(\text{John})$

→ Result: $\neg\text{Illegal}(x) \vee \neg\text{Sells}(\text{John}, x)$

Unify $x = \text{Gun}$ in result clause

→ Clause becomes: $\neg\text{Illegal}(\text{Gun}) \vee \neg\text{Sells}(\text{John}, \text{Gun})$

Now resolve with:

Clause 2: $\text{Illegal}(\text{Gun})$

→ Resolves with $\neg\text{Illegal}(\text{Gun})$

→ Result: $\neg\text{Sells}(\text{John}, \text{Gun})$

Then resolve with:

Clause 3: $\text{Sells}(\text{John}, \text{Gun})$

→ Resolves with $\neg\text{Sells}(\text{John}, \text{Gun})$

→ **Empty clause** \square reached!

Conclusion:

Since the resolution led to a **contradiction**, the **original goal is proven true**:

John is a criminal.



FOUR DECADES OF NATION BUILDING THROUGH EXCELLENCE IN EDUCATION

KUMARAGURU

ENGINEERING | LIBERAL ARTS | AGRICULTURE | MANAGEMENT
BUSINESS INCUBATION | TAMIL RESEARCH



KUMARAGURU
COLLEGE OF TECHNOLOGY
character is life



KCT
BUSINESS SCHOOL



KUMARAGURU
COLLEGE OF LIBERAL
ARTS AND SCIENCE



KUMARAGURU
INSTITUTE OF
AGRICULTURE



KUMARAGURU
SCHOOL OF BUSINESS



KRIA
KNACKING RESEARCH & INNOVATION
ALLIANCE



KUMARAGURU
SCHOOL OF
INNOVATION



KUMARAGURU
KCIRI
CENTRE FOR INDUSTRIAL
RESEARCH & INNOVATION



FORGE



SEA
SAKTHI
EXCELLENCE
ACADEMY
GATEWAY INTEGRATED EDUCATION



N.MAHALINGAM
TAMIL RESEARCH
CENTRE



KCT
TECH PARK



KARE
Kumaraguru
Action
Relief And Empowerment



Dr.N.MAHALINGAM
CHESS ACADEMY

THANK YOU

ARTIFICIAL INTELLIGENCE AND AUTOMATION | PREETHI G | AP | AIDS