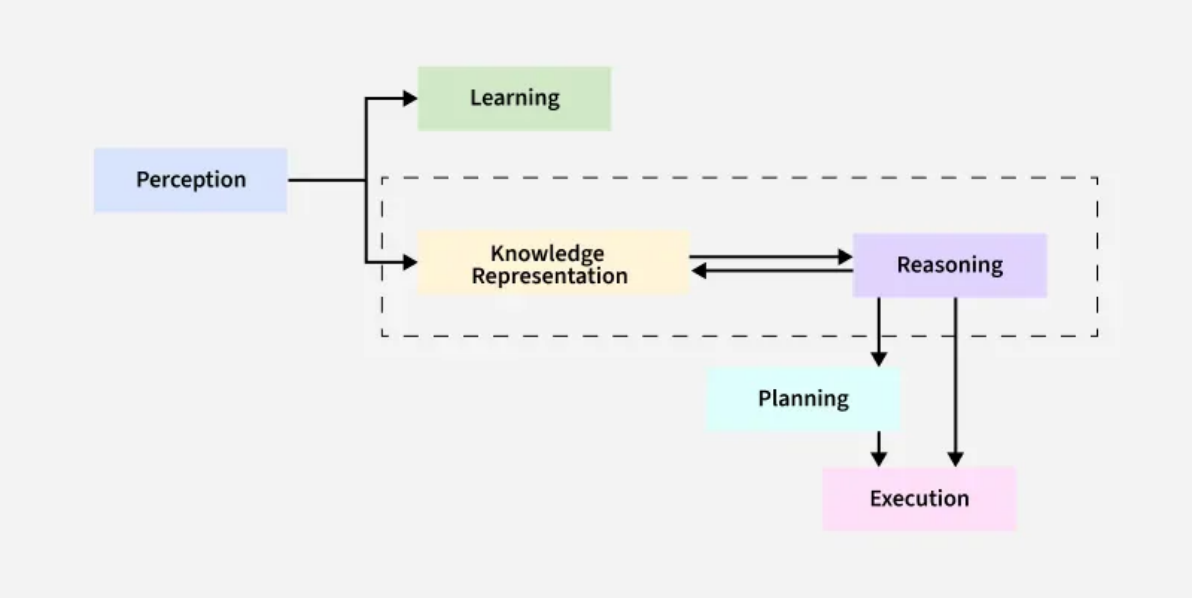
Knowledge Representation in AI

knowledge representation (KR) in AI refers to **encoding information about the world into formats that AI systems can utilize to solve complex tasks.** This process enables machines to reason, learn, and make decisions by structuring data in a way that mirrors human understanding.



Kno
wledge Representation in AI

Artificial intelligence systems operate on data. ***However, raw data alone does not lead to intelligence****.* ***AI must transform data into structured knowledge.*** *KR achieves this by defining formats and methods for organizing information. With clear representations, AI systems solve problems, make decisions, and learn from new experiences.*

The Synergy of Knowledge and Intelligence

Knowledge and intelligence in AI share a symbiotic relationship:

- **Knowledge as a Foundation**: Knowledge provides facts, rules, and data (e.g., traffic laws for self-driving cars). Without it, intelligence lacks the raw material to act.
- **Intelligence as Application**: Intelligence applies knowledge to solve problems (e.g., a robot using physics principles to navigate terrain).
- **Interdependence**: Static knowledge becomes obsolete without adaptive intelligence. Conversely, intelligence without knowledge cannot reason or learn (e.g., an AI with no medical database cannot diagnose diseases).
- **Synergy**: Effective AI systems merge robust knowledge bases (the *what*) with reasoning algorithms (the *how*). For example, ChatGPT combines vast language data (knowledge) with transformer models (intelligence) to generate coherent text.

Core Methods of Knowledge Representation

1. Logic-Based Systems

Logic-based methods use formal rules to model knowledge. These systems prioritize precision and are ideal for deterministic environments.

- **Propositional Logic**
  Represents knowledge as declarative statements (propositions) linked by logical operators like AND, OR, and NOT. For example, "If it rains (A) AND the ground is wet (B), THEN the road is slippery (C)." While simple, it struggles with complex relationships. Often follow the format "IF condition THEN conclusion." For instance, in a knowledge-based system, you might have:

IF an object is red AND round, THEN the object might be an apple.

- **First-Order Logic (FOL)**
  Extends propositional logic by introducing variables, quantifiers, and predicates. FOL can express statements like, “All humans (∀x) are mortal (Mortal(x)).” It supports nuanced reasoning but demands significant computational resources.

Legal AI tools apply logic-based rules to analyze contracts for compliance.

2. Structured Representations

These methods organize knowledge hierarchically or through networks, mimicking how humans categorize information.

- **Semantic Networks**
  Represent knowledge as nodes (concepts) and edges (relationships). For example, "Dog" links to "Animal" via an "Is-A" connection. They simplify inheritance reasoning but lack formal semantics.

- **Frames**
  Group related attributes into structured "frames." A "Vehicle" frame may include slots like wheels, engine type, and fuel. Frames excel in default reasoning but struggle with exceptions.

- **Ontologies**
  Define concepts, hierarchies, and relationships within a domain using standards like OWL (Web Ontology Language). Ontologies power semantic search engines and healthcare diagnostics by standardizing terminology.

E-commerce platforms use ontologies to classify products and enhance search accuracy.

3. Probabilistic Models

These systems handle uncertainty by assigning probabilities to outcomes.

- **Bayesian Networks**
  Use directed graphs to model causal relationships. Each node represents a variable, and edges denote conditional dependencies. For instance, a Bayesian network can predict the likelihood of equipment failure based on maintenance history and usage.

- **Markov Decision Processes (MDPs)**
  Model sequential decision-making in dynamic environments. MDPs help robotics systems navigate obstacles by evaluating potential actions and rewards.

Weather prediction systems combine historical data and sensor inputs using probabilistic models to forecast storms.

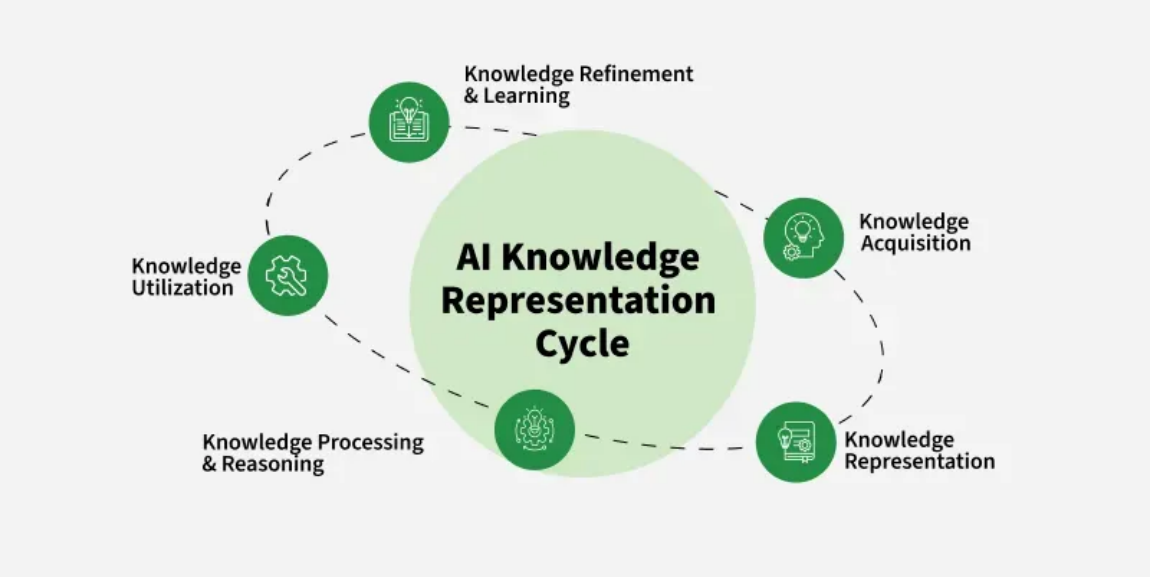
4. Distributed Representations

Modern AI leverages neural networks to encode knowledge as numerical vectors, capturing latent patterns in data.

- **Embeddings**
  Convert words, images, or entities into dense vectors. Word embeddings like Word2Vec map synonyms to nearby vectors, enabling semantic analysis.
- **Knowledge Graphs**
  Combine graph structures with embeddings to represent entities (e.g., people, places) and their relationships. Google's Knowledge Graph enhances search results by linking related concepts.

The AI Knowledge Cycle

The AI Knowledge Cycle represents the continuous process through which AI systems acquire, process, utilize, and refine knowledge.



AI Knowledge Cycle

This cycle ensures that AI remains adaptive and improves over time.

1. Knowledge Acquisition: AI gathers data from various sources, including structured databases, unstructured text, images, and real-world interactions. Techniques such as machine learning, natural language processing (NLP), and computer vision enable this acquisition.

2. Knowledge Representation : Once acquired, knowledge must be structured for efficient storage and retrieval. Represented through methods explained above:

3. Knowledge Processing & Reasoning: AI applies logical inference, probabilistic models, and deep learning to process knowledge. This step allows AI to:

- Draw conclusions (deductive and inductive reasoning)
- Solve problems using heuristic search and optimization
- Adapt through reinforcement learning and experience

4. Knowledge Utilization: AI applies knowledge to real-world tasks, including decision-making, predictions, and automation. Examples include:

- Virtual assistants understanding user queries
- AI-powered recommendation systems suggesting content
- Self-driving cars making real-time navigation decisions

5. Knowledge Refinement & Learning: AI continuously updates its knowledge base through feedback loops. Techniques like reinforcement learning, supervised fine-tuning, and active learning help improve accuracy and adaptability. This ensures AI evolves based on new data and experiences.

The AI Knowledge Cycle is iterative. AI systems refine knowledge continuously, ensuring adaptability and long-term learning. This cycle forms the backbone of intelligent systems, enabling them to grow smarter over time.

Types of Knowledge in AI

AI systems rely on different types of knowledge to function efficiently. Each type serves a specific role in reasoning, decision-making, and problem-solving. Below are the primary types of knowledge used in AI:

1. Declarative Knowledge (Descriptive Knowledge)

Declarative knowledge consists of facts and information about the world that AI systems store and retrieve when needed. It represents "what" is known rather than "how" to do something. **This type of knowledge is often stored in structured formats like databases, ontologies, and knowledge graphs**.

For example, a fact such as "Paris is the capital of France" is declarative knowledge. AI applications like search engines and virtual assistants use this type of knowledge to answer factual queries and provide relevant information.

2. Procedural Knowledge (How-To Knowledge)

Procedural knowledge **defines the steps or methods required to perform specific tasks**. It represents ***"how" to accomplish something rather than just stating a fact***.

For instance, knowing how to solve a quadratic equation or how to drive a car falls under procedural knowledge. AI systems, such as expert systems and robotics, utilize procedural knowledge to execute

tasks that require sequences of actions. This type of knowledge is often encoded in rule-based systems, decision trees, and machine learning models.

3. Meta-Knowledge (Knowledge About Knowledge)

Refers to knowledge about **how information is structured, used, and validated**. It helps AI determine the reliability, relevance, and applicability of knowledge in different scenarios.

For example, an AI system deciding whether a piece of medical advice comes from a trusted scientific source or a random blog post is using meta-knowledge. This type of knowledge is crucial in AI models for filtering misinformation, optimizing learning strategies, and improving decision-making.

4. Heuristic Knowledge (Experience-Based Knowledge)

Heuristic knowledge is derived from experience, intuition, and trial-and-error methods. It allows AI systems to make educated guesses or approximate solutions when exact answers are difficult to compute.

For example, a navigation system suggesting an alternate route based on past traffic patterns is applying heuristic knowledge. AI search algorithms, such as A search and genetic algorithms, leverage heuristics to optimize problem-solving processes, making decisions more efficient in real-world scenarios.*

5. Common-Sense Knowledge

Common-sense knowledge ***represents basic understanding about the world that humans acquire naturally but is challenging for AI to learn***. It includes facts like "**water is wet" or "if you drop something, it will fall**."

*AI systems often struggle with this type of knowledge because it requires **contextual understanding beyond explicit programming**.*

Researchers are integrating common-sense reasoning into AI using large-scale knowledge bases such as ConceptNet, which helps machines understand everyday logic and improve their interaction with humans.

6. Domain-Specific Knowledge

Domain-specific knowledge focuses on specialized fields such as medicine, finance, law, or engineering. It includes highly detailed and structured information relevant to a particular industry.

For instance, in the medical field, AI-driven diagnostic systems rely on knowledge about symptoms, diseases, and treatments. Similarly, financial AI models use economic indicators, risk assessments, and market trends. Expert systems and AI models tailored for specific industries require domain-specific knowledge to provide accurate insights and predictions.

Challenges in Knowledge Representation

While knowledge representation is fundamental to AI, it comes with several challenges:

1. **Complexity**: Representing all possible knowledge about a domain can be highly complex, requiring sophisticated methods to manage and process this information efficiently.

2. **Ambiguity and Vagueness**: Human language and concepts are often ambiguous or vague, making it difficult to create precise representations.
3. **Scalability**: As the amount of knowledge grows, AI systems must scale accordingly, which can be challenging both in terms of storage and processing power.
4. **Knowledge Acquisition**: Gathering and encoding knowledge into a machine-readable format is a significant hurdle, particularly in dynamic or specialized domains.
5. **Reasoning and Inference**: AI systems must not only store knowledge but also use it to infer new information, make decisions, and solve problems. This requires sophisticated reasoning algorithms that can operate efficiently over large knowledge bases.

Applications of Knowledge Representation in AI

Knowledge representation is applied across various domains in AI, enabling systems to perform tasks that require human-like understanding and reasoning. Some notable applications include:

1. **Expert Systems**: These systems use knowledge representation to provide advice or make decisions in specific domains, such as medical diagnosis or financial planning.
2. **Natural Language Processing (NLP)**: Knowledge representation is used to understand and generate human language, enabling applications like chatbots, translation systems, and sentiment analysis.
3. **Robotics**: Robots use knowledge representation to navigate, interact with environments, and perform tasks autonomously.
4. **Semantic Web**: The Semantic Web relies on ontologies and other knowledge representation techniques to enable machines to understand and process web content meaningfully.
5. **Cognitive Computing**: Systems like IBM's Watson use knowledge representation to process vast amounts of information, reason about it, and provide insights in fields like healthcare and research.

Conclusion

Knowledge representation is a foundational element of AI, enabling machines to understand, reason, and act on the information they process. By leveraging various representation techniques, AI systems can tackle complex tasks that require human-like intelligence. However, challenges such as complexity, ambiguity, and scalability remain critical areas of ongoing research. As AI continues to evolve, advancements in knowledge representation will play a pivotal role in the development of more intelligent and capable systems.

UNIT 2

What is Knowledge Representation in AI

Humans are great at tasks that require creativity, critical thinking, and empathy. They can learn from experience and adapt to new situations, and they possess emotional intelligence that allows them to understand and connect with other people on a deep level.

On the other hand, Artificial Intelligence or AI is excellent at tasks that require speed, accuracy, and scalability. It can quickly process vast amounts of data and perform complex calculations and analyses far beyond human capabilities.

Knowledge representation is a crucial element of Artificial Intelligence. It is believed that an intelligent system needs to have an explicit representation of its knowledge to reason and make decisions.

Knowledge representation provides a framework for representing, organizing, and manipulating knowledge that can be used to solve complex problems, make decisions, and learn from data.

For example, when you see a hot tea cup, a signal immediately comes from your brain cautioning you against picking it up. If we were to make AI more sophisticated(or humanist), we would be required to feed them with more and often complex information about our world to perform the complex task, which leads to the concept of Knowledge Representation in Artificial Intelligence.

- An intelligent agent needs **knowledge** about the real world for taking decisions and **reasoning** to act efficiently.
- Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently**.
- Knowledge-based agents are composed of two main parts:
    - **Knowledge-base and**
    - **Inference system**.

What to Represent:

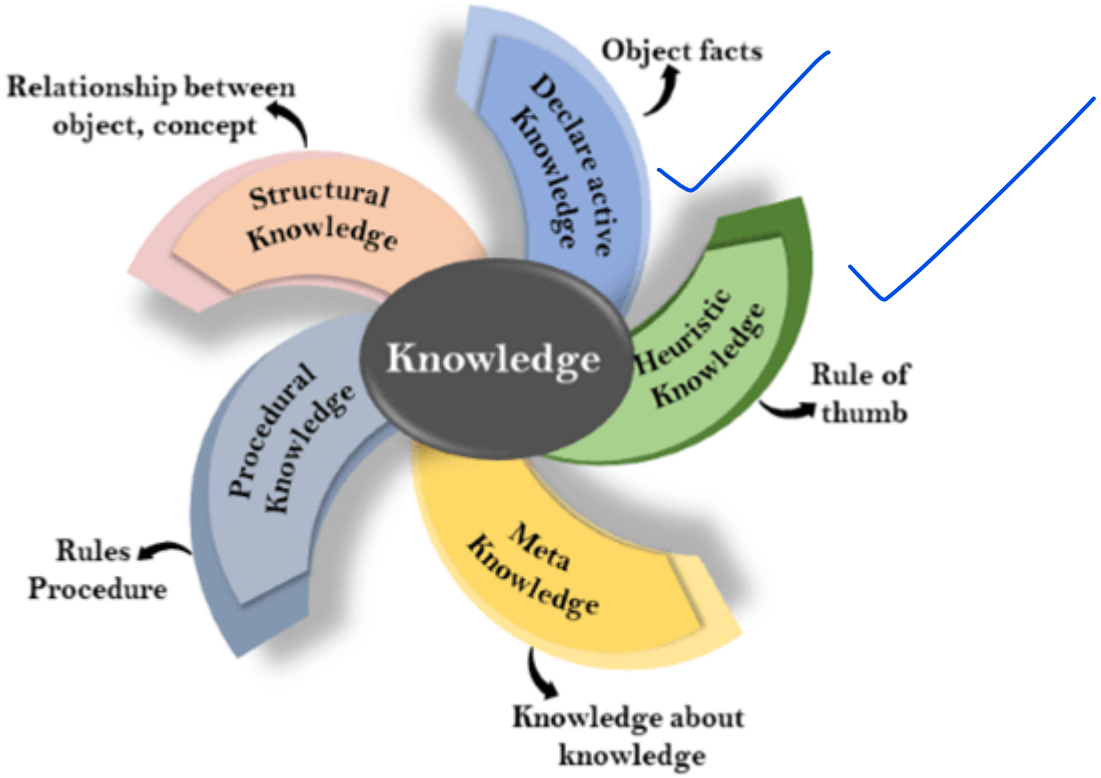
Following are the kind of knowledge which needs to be represented in AI systems:

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.

- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Types of knowledge

Following are the various types of knowledge:



1. Declarative Knowledge:

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called descriptive knowledge and expressed in declarativesentences.
- It is simpler than procedural language.

2. Procedural Knowledge

- It is also known as imperative knowledge.
- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.

4. Heuristic knowledge:

- Heuristic knowledge is representing knowledge of some experts in a filed or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

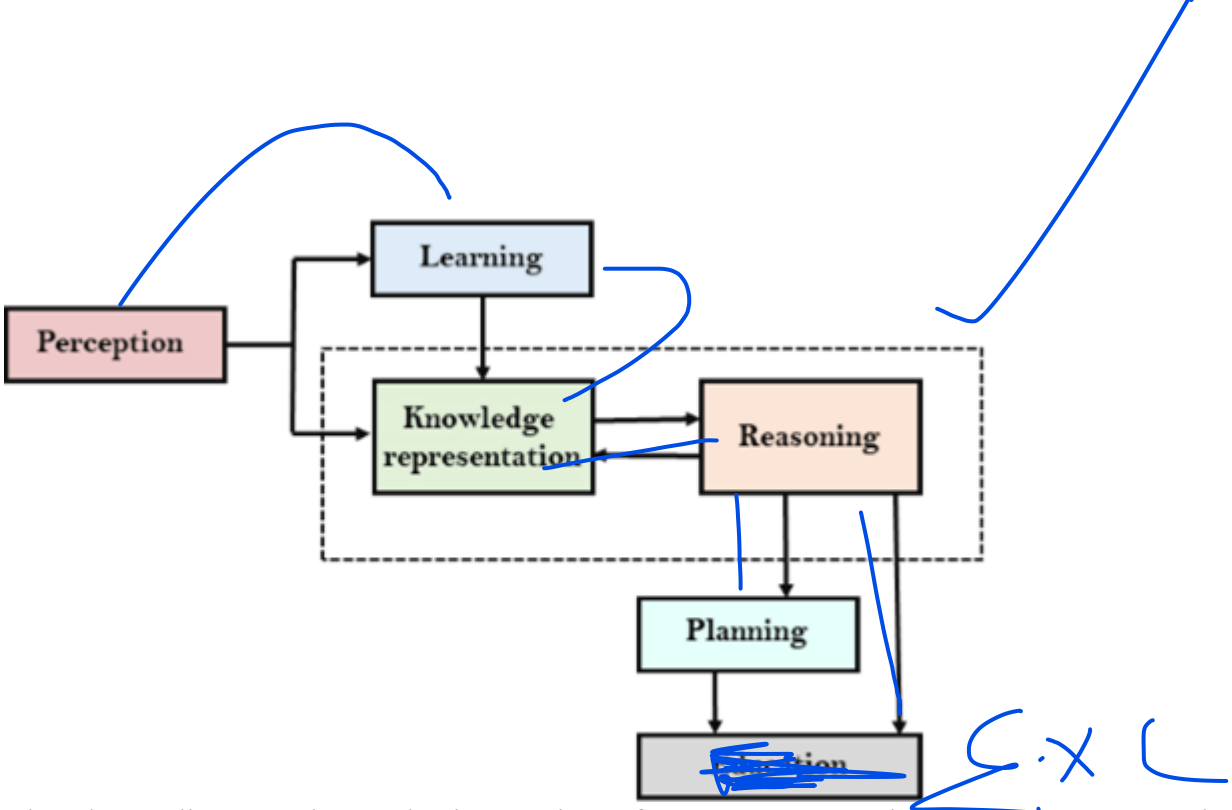
5. Structural knowledge:

- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

AI knowledge cycle:

An Artificial intelligence system has the following components for displaying intelligent behavior:

- Perception
- Learning
- Knowledge Representation and Reasoning
- Planning
- Execution



The above diagram shows the interaction of an AI system with the **real world** and the **components** involved in showing intelligence.

- The **Perception component** retrieves data or information from the environment. with the help of this component, you can retrieve data from the environment, find out the source of noises and check if the AI was damaged by anything. Also, it defines how to respond when any sense has been detected.
- Then, there is the **Learning Component** that learns from the captured data by the perception component. The goal is to build computers that can be taught instead of programming them. Learning focuses on the process of self-improvement. In order to learn new things, the system requires knowledge acquisition, inference, acquisition of heuristics, faster searches, etc.
- The main component in the cycle is **Knowledge Representation and Reasoning** which shows the human-like intelligence in the machines. Knowledge representation is all about understanding intelligence. Instead of trying to understand or build brains from the bottom up, its goal is to understand and build intelligent behavior from the top-down and focus on what an agent needs to know in order to behave intelligently. Also, it defines how automated reasoning procedures can make this knowledge available as needed.
- The **Planning and Execution** components depend on the analysis of knowledge representation and reasoning. Here, planning includes giving an initial state, finding their preconditions and effects, and a sequence of actions to achieve a state in which a particular goal holds. Now once the planning is completed, the final stage is the execution of the entire process.

Approaches to Knowledge Representation

1. Simple Relational Knowledge

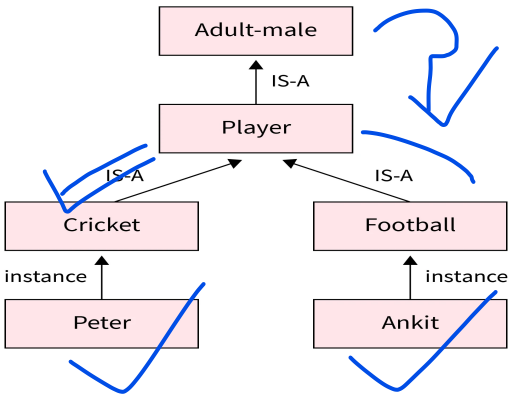
- This type of knowledge uses relational methods to store facts.
- It is one of the simplest types of knowledge representation.
- The facts are systematically set out in terms of rows and columns.
- This type of knowledge representation is used in database systems where the relationship between different entities is represented.
- There is a low opportunity for inference.

Example : The following is the simple relational knowledge representation

Player	Weigth	Age
Player1	65	23
Player2	58	18
Player3	75	24

2. Inheritable Knowledge

- Inheritable knowledge in AI refers to knowledge acquired by an AI system through learning and can be transferred or inherited by other AI systems.
- This knowledge can include models, rules, or other forms of knowledge that an AI system learns through training or experience.
- In this approach, all data must be stored in a hierarchy of classes.
- Boxed nodes are used to represent objects and their values.
- We use Arrows that point from objects to their values.
- Rather than starting from scratch, an AI system can inherit knowledge from other systems, allowing it to learn faster and avoid repeating mistakes that have already been made. Inheritable knowledge also allows for knowledge transfer across domains, allowing an AI system to apply knowledge learned in one domain to another.



3. Inferential Knowledge

- Inferential knowledge refers to the ability to draw logical conclusions or make predictions based on available data or information
- In artificial intelligence, inferential knowledge is often used in machine learning algorithms, where models are trained on large amounts of data and then used to make predictions or decisions about new data.
- For example, in image recognition, a machine learning model can be trained on a large dataset of labeled images and then used to predict the contents of new images that it has never seen before. The model can draw inferences based on the patterns it has learned from the training data.
- It represents knowledge in the form of formal logic.

Example: Statement 1: Alex is a footballer.

Statement 2: All footballers are athletes. Then it can be represented as;

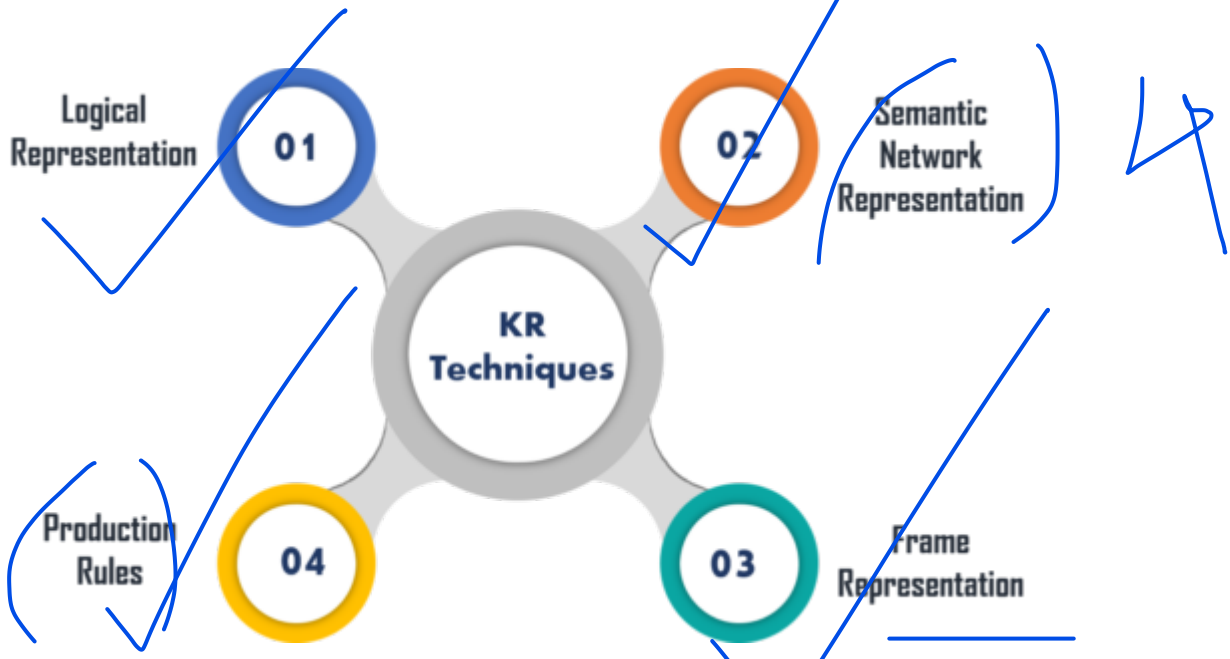
Footballer(Alex) ∀x = Footballer (x) ————-> Athelete (x)s

4. Procedural Knowledge:

- In artificial intelligence, procedural knowledge refers to the knowledge or instructions required to perform a specific task or solve a problem.
- This knowledge is often represented in algorithms or rules dictating how a machine processes data or performs tasks.
- For example, in natural language processing, procedural knowledge might involve the steps required to analyze and understand the meaning of a sentence. This could include tasks such as identifying the parts of speech in the sentence, identifying relationships between different words, and determining the overall structure and meaning of the sentence.
- One of the most important rules used is the If-then rule.
- This knowledge allows us to use various coding languages such as LISP and Prolog.
- Procedural knowledge is an important aspect of artificial intelligence, as it allows machines to perform complex tasks and make decisions based on specific instructions.

Techniques of Knowledge Representation in AI

There are four techniques of representing knowledge such as:



Logical Representation

Logical representation is a language with some **definite rules** which deal with propositions and has no ambiguity in representation. It represents a conclusion based on various conditions and lays down some important **communication rules**. Also, it consists of precisely defined syntax and semantics which supports the sound inference. Each sentence can be translated into logics using syntax and semantics.

Syntax	Semantics
• It decides how we can construct legal sentences in logic. • It determines which symbol we can use in knowledge representation. • Also, how to write those symbols.	• Semantics are the rules by which we can interpret the sentence in the logic. • It assigns a meaning to each sentence.

Logical representation can be categorised into mainly two logics:

a) Propositional Logics
b) Predicate logics

Advantages:

- Logical representation helps to perform logical reasoning.
- This representation is the basis for the programming languages.

Disadvantages:

- Logical representations have some restrictions and are challenging to work with.

- This technique may not be very natural, and inference may not be very efficient.

Propositional Logic: This type of logical representation is also known as propositional calculus or statement logic. This works in a Boolean, i.e., True or False method.

First-order Logic: This type of logical representation is also known as the First Order Predicate Calculus Logic (FOPL). This logical representation represents the objects in quantifiers and predicates and is an advanced version of propositional logic.

Propositional Logic

A **proposition** is basically a declarative sentence that has a truth value. Truth value can either be true or false, but it needs to be assigned any of the two values and not be ambiguous. The purpose of using propositional logic is to analyze a statement, individually or compositely.

Example:

1. a) It is Sunday.
2. b) The Sun rises from West (False proposition)
3. c) 3+3= 7(False proposition)
4. d) 5 is a prime number.

Syntax of propositional logic:

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

a) **Atomic Propositions**
b) **Compound propositions**

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

Example:

a) 2+2 is 4, it is an atomic proposition as it is a **true** fact.

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

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

Example:

a) "It is raining today, and street is wet."
b) "Ankit is a doctor, and his clinic is in Mumbai."

Logical Connectives:

Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

1. **Negation:** A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.
2. **Conjunction:** A sentence which has ∧ connective such as, **P ∧ Q** is called a conjunction.
   Example: Rohan is intelligent and hardworking. It can be written as,
   P= Rohan is intelligent,
   Q= Rohan is hardworking. → P∧ Q.
3. **Disjunction:** A sentence which has ∨ connective, such as **P ∨ Q**. is called disjunction, where P and Q are the propositions.
   Example: "Ritika is a doctor or Engineer",
   Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as **P ∨ Q**.
4. **Implication:** A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as
   If it is raining, then the street is wet.
   Let P= It is raining, and Q= Street is wet, so it is represented as P → Q
5. **Biconditional:** A sentence such as **P⇔ Q is a Biconditional sentence, example If I am breathing, then I am alive**
   P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

Following is the summarized table for Propositional Logic Connectives:

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

Truth Table:

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

For Negation:

P	¬ P
True	**False**
False	**True**

For Conjunction:

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

For disjunction:

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

For Implication:

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

For Biconditional:

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

Precedence of connectives:

Just like arithmetic operators, there is a precedence order for propositional connectors or logical operators. This order should be followed while evaluating a propositional problem. Following is the list of the precedence order for operators:

Precedence	Operators
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction(AND)
Fourth Precedence	Disjunction(OR)
Fifth Precedence	Implication
Six Precedence	Biconditional

Limitations of Propositional Logic

- **Limited expressivity:** Propositional logic is limited in its ability to represent complex relationships between objects or concepts. It can only express simple propositional statements with binary truth values (true/false). This makes it difficult to represent concepts such as uncertainty, ambiguity, and vagueness.
- **Inability to handle quantifiers:** Propositional logic is unable to handle quantifiers such as "all" or "some." For example, it cannot represent the statement "all humans are mortal" in a concise manner. This makes it difficult to reason about sets of objects or concepts.
- **Lack of support for negation:** Propositional logic does not provide an easy way to represent negation. This can make it difficult to represent negative statements and reason about them.

Predicate Logic in AI (Artificial Intelligence)

What is Predicate Logic in AI? Predicate logic in artificial intelligence, also known as first-order logic or first order predicate logic in AI, is a formal system used in logic and mathematics to represent and reason about complex relationships and structures. It plays a crucial role in knowledge representation, which is a field within artificial

intelligence and philosophy concerned with representing knowledge in a way that machines or humans can use for reasoning and problem-solving.

∧	*and* [conjunction]
∨	*or* [disjunction]
⇒	*implies* [implication]
¬	*not* [negation]
∀	*For all*
∃	*There exists*

Basic Components of Predicate Logic

1. **Predicates:** Predicates are statements or propositions that can be either true or false depending on the values of their arguments. They represent properties, relations, or characteristics of objects. For example, "IsHungry(x)" can be a predicate, where "x" is a variable representing an object, and the predicate evaluates to true if that object is hungry.

2. **Variables:** Variables are symbols that can take on different values. In predicate logic, variables are used to represent objects or entities in the domain of discourse. For example, "x" in "IsHungry(x)" can represent any object in the domain, such as a person, animal, or thing.

3. **Constants:** Constants are specific values that do not change. They represent particular objects in the domain. For instance, in a knowledge base about people, "Alice" and "Bob" might be constants representing specific individuals.

4. **Quantifiers:** Quantifiers are used to specify the scope of variables in logical expressions. There are two main quantifiers in predicate logic:

   - Existential Quantifier (∃): Denoted as ∃, it indicates that there exists at least one object for which the statement within the quantifier is true. For example, "∃x IsHungry(x)" asserts that there is at least one object that is hungry.
   - Universal Quantifier (∀): Denoted as ∀, it indicates that the statement within the quantifier is true for all objects in the domain. For example, "∀x IsHuman(x) → IsMortal(x)" asserts that all humans are mortal.

Rules of inference in artificial intelligence

Rules of inference are a set of logical principles and deductive rules that draw conclusions from existing information or assertions.

Types of inference rules

Following are the types of inference rules:

1. Modus ponens
2. Modus tollens
3. Hypothetical syllogism
4. Disjunctive syllogism
5. Addition
6. Simplification
7. Resolution

Before we begin, let's suppose the following statements for all the types mentioned below:

P: It is raining.

Q: The streets are wet.

R: Roads are slippery.

Modus ponens

If P implies Q and P is true, then Q is true.

Notation

$((P \rightarrow Q) \wedge P) \Rightarrow Q$

Example

If it is raining, then the streets are wet (P -> Q), and it is raining (P); therefore, the streets are wet (Q).

Modus tollens

If P implies Q and Q is false, then P is false.

Notation

$((P \rightarrow Q) \wedge \sim Q) \Rightarrow \sim P$

Example

If it is raining, then the streets are wet (P -> Q), and streets are not wet (~Q); therefore, it is not raining (~P).

Hypothetical syllogism

If P implies Q and Q implies R, then P implies R.

Notation

$((P \rightarrow Q) \wedge (Q \rightarrow R)) \Rightarrow (P \rightarrow R)((P \rightarrow Q) \wedge (Q \rightarrow R)) \Rightarrow (P \rightarrow R)$

Example

If it is raining, then the streets are wet (P -> Q), and if the streets are wet, then roads are slippery (Q -> R); therefore, if it is raining, then roads are slippery (P -> R).

Disjunctive syllogism

If P or Q is true, and P is false, then Q is true.

Notation

$((P \vee Q) \wedge \sim P) \Rightarrow Q((P \vee Q) \wedge \sim P) \Rightarrow Q$

Example

It is raining or streets are wet (P∨Q), and it is not raining (~P); therefore, streets are wet (Q).

Addition

If P is true, then P or Q is true.

Notation

$P \Rightarrow (P \vee Q) P \Rightarrow (P \vee Q)$

Example

It is raining (P), therefore it is raining or streets are wet (P∨Q).

Simplification

If P and Q is true, then P is true.

Notation

$(P \wedge Q) \Rightarrow P(P \wedge Q) \Rightarrow P$

Example

It is raining and streets are wet (P∧Q); therefore it is raining (P).

Resolution

If both P or Q and not P or R is true, then Q or R is true.

Notation

$((P \vee Q) \wedge (\sim P \vee R)) \Rightarrow (Q \vee R)((P \vee Q) \wedge (\sim P \vee R)) \Rightarrow (Q \vee R)$

Example

It is raining or streets are wet (P∨Q) and it is not raining or roads are slippery (~P∨R); therefore streets are wet or roads are slippery (Q∨R).

Applications

Inference is a fundamental process that significantly impacts various applications, including natural language processing, expert systems, robotics, and computer vision.

Natural language processing (NLP): Based on context and prior knowledge, the inference understands the meaning of sentences.

Computer vision: The inference recognizes object as an image, based on patterns and features.

Robotics: Plans and actions are executed by the inference based on the environment.

Frames in AI: Knowledge Representation and Inheritance

In Artificial Intelligence (AI), **frames** represent a pivotal concept that helps machines understand and interpret complex real-world scenarios. Originating from cognitive science and knowledge representation, frames are utilized to structure information in a way that allows AI systems to reason, infer, and make decisions.

What Are Frames in AI?

Frames are data structures used in AI to represent stereotypical situations or scenarios. They encapsulate information about objects, events, and their interrelationships within a particular context. Each frame consists of a set of attributes and values, forming a template for understanding specific situations.

For instance, a "restaurant" frame might include attributes such as "menu," "waitstaff," and "tables," each with its own set of details.

Concept of Frames

The frame concept was introduced by **Minsky** in 1974 and is foundational in the field of knowledge representation. Frames are designed to provide a structured way to capture the essential aspects of a situation, facilitating easier retrieval and manipulation of information. They are akin to schemas or blueprints that organize knowledge into manageable chunks.

Key Components of Frames

Frames are essential for structuring knowledge in AI, and understanding their key components helps in effectively utilizing them.

Here are the main components of frames, along with examples to illustrate their use:

1. Slots

Slots are attributes or properties of a frame. They represent the different aspects or characteristics of the frame's concept.

Example: For a "Person" frame, slots might include:

- **Name:** The individual's name
- **Age:** The individual's age
- **Occupation:** The individual's profession
- **Address:** The individual's home address

2. Facets

Facets provide additional details or constraints for slots, defining acceptable values or specifying how slots should be used.

Example: For the "Age" slot in the "Person" frame:

- **Type:** Integer
- **Range:** 0 to 120
- **Default Value:** 30

3. Default Values

Default values are predefined values assigned to slots if no specific value is provided. They offer a baseline that can be overridden with more specific information.

Example: In a "Car" frame:

- **Make:** Default value could be "Unknown"
- **Model:** Default value could be "Unknown"
- **Year:** Default value could be the current year

4. Procedures

Procedures are methods or functions associated with frames that define how the information within the frame should be processed or utilized.

Example: In an "Account" frame:

- **Procedure:** CalculateInterest - A method to compute interest based on the account balance.

Example of a Complete Frame

Let's construct a complete frame for a "Book" in a library management system:

- **Frame Name**: Book
    - **Slots**:
        - **Title**: "To Kill a Mockingbird"
        - **Author**: "Harper Lee"
        - **Publication Year**: 1960
        - **ISBN**: "978-0-06-112008-4"
        - **Genre**: "Fiction"
    - **Facets**:
        - **Publication Year**:
            - **Type**: Integer
            - **Range**: 1450 to current year (reasonable range for publication years)

        - **ISBN**:
            - **Format**: 13-digit number
    - **Default Values**:
        - **Genre**: "Unknown" (if not specified)
    - **Procedures**:
        - **CheckAvailability**: A method to check if the book is currently available in the library.
        - **UpdateRecord**: A method to update the book's record when it is borrowed or returned.

```
+-----------------------------------------------+
|                Book Frame                     |
+-----------------------------------------------+
| Slots:                                        |
| Title: "To Kill a Mockingbird"                |
| Author: "Harper Lee"                          |
| Publication Year: 1960                        |
| ISBN: "978-0-06-112008-4"                     |
| Genre: "Fiction"                              |
+-----------------------------------------------+
| Facets:                                       |
| Publication Year:                             |
|   - Type: Integer                             |
|   - Range: 1450 to current year               |
| ISBN:                                         |
|   - Format: 13-digit number                   |
+-----------------------------------------------+
| Default Values:                               |
| Genre: "Unknown" (if not specified)           |
+-----------------------------------------------+
| Procedures:                                   |
| CheckAvailability: Method to check if the book |
|   is currently available in the library.      |
| UpdateRecord: Method to update the book's     |
|   record when it is borrowed or returned.     |
+-----------------------------------------------+
```

This frame encapsulates all necessary information about a book and provides mechanisms to interact with that information.

Introduction to Frame Inheritance

Frame inheritance is a method used in knowledge representation systems to manage and organize information efficiently. It allows one frame (child) to inherit attributes and properties from another frame (parent), creating a hierarchical structure. This method facilitates the reuse and extension of existing knowledge.

Key Concepts of Frame Inheritance

1. **Parent Frame**: The frame from which attributes and properties are inherited. It defines general attributes that are common to all its child frames.
2. **Child Frame**: The frame that inherits attributes and properties from the parent frame. It can add new attributes or override existing ones to represent more specific information.
3. **Inheritance Hierarchy**: A tree-like structure where frames are organized hierarchically. Each child frame can inherit from multiple parent frames, forming a network of relationships.
4. **Overriding**: When a child frame modifies or replaces an attribute inherited from the parent frame with a more specific value or definition.
5. **Extension**: Adding new attributes or properties to a child frame that are not present in the parent frame.

How Frame Inheritance Works?

1. **Define Parent Frame**: Create a general frame with common attributes. For example, a "Vehicle" frame might include attributes like "Make," "Model," and "Year."
2. **Create Child Frame**: Define a more specific frame that inherits from the parent frame. For example, a "Car" frame might inherit attributes from the "Vehicle" frame and add specific attributes like "Number of Doors."
3. **Use Inherited Attributes**: The child frame automatically includes all attributes from the parent frame, providing a structured way to build on existing knowledge.
4. **Override or Extend**: Modify or add attributes in the child frame as needed to refine the representation. For example, the "Car" frame might override the "Year" attribute to specify a range of acceptable values.

Example of Frame Inheritance

Let's consider an example with a hierarchy of frames in a library system:

- **Parent Frame**: "LibraryItem"
    - **Attributes**:
        - **Title**
        - **Author**
        - **Publication Year**
- **Child Frame 1**: "Book" (inherits from "LibraryItem")

    - **Inherited Attributes**: Title, Author, Publication Year
    - **Extended Attributes**:
        - **ISBN**
        - **Genre**
- **Child Frame 2**: "Magazine" (inherits from "LibraryItem")
    - **Inherited Attributes**: Title, Author, Publication Year
    - **Extended Attributes**:
        - **Issue Number**
        - **Publisher**

In this example:

- The "Book" frame inherits the common attributes from the "LibraryItem" frame and adds specific attributes related to books.
- The "Magazine" frame also inherits from "LibraryItem" but adds attributes specific to magazines.

Applications of Frames in AI

1. **Natural Language Processing (NLP)**: In NLP, frames are used to understand the context of words and sentences. For example, a "booking" frame might be used to interpret requests for reservations, extracting relevant information such as date, time, and number of people.
2. **Expert Systems**: Expert systems use frames to represent knowledge about specific domains. For instance, a medical diagnosis system might employ frames to represent various diseases, symptoms, and treatment options.
3. **Robotics**: Frames help robots make sense of their environment by providing structured information about objects and their properties. This allows robots to perform tasks such as object recognition and manipulation.
4. **Cognitive Modeling**: Frames are used in cognitive modeling to simulate human thought processes. By representing knowledge in frames, researchers can create models that mimic human reasoning and decision-making.

Advantages of Using Frames

- **Organized Knowledge**: Frames help in structuring information in a way that mirrors real-world scenarios, making it easier for AI systems to understand and process.
- **Flexibility**: Frames can be easily modified or extended to incorporate new information or adapt to changing contexts.
- **Reusability**: Once defined, frames can be reused across different applications or scenarios, promoting consistency and efficiency.

Challenges and Limitations

- **Complexity**: As the number of frames and their interrelationships increase, managing and maintaining the frames can become complex.
- **Context Sensitivity**: Frames may struggle to adapt to highly dynamic or ambiguous situations where predefined structures may not fit.
- **Scalability**: For large-scale systems, the sheer volume of frames and their interactions can pose challenges in terms of performance and resource management.

Difference between Frames and Ontologies

Aspect	Frames	Ontologies
Definition	Data structures representing specific situations	Formal representations of knowledge domains
Structure	Slots, facets, default values, procedures	Classes, subclasses, properties, instances
Flexibility	Adaptable to specific contexts and scenarios	Formal and standardized, designed for consistency across domains
Usage	NLP, expert systems, cognitive modeling	Semantic web, knowledge management, data integration
Context	Context-specific, can vary in structure	Domain-wide, provides a shared understanding
Formalism	Less formal, more flexible	Highly formal, uses specific languages (e.g., OWL)

Frames and ontologies are both valuable tools for knowledge representation in AI but serve different purposes. Frames are useful for representing specific, context-dependent scenarios and are often used in applications requiring flexibility and adaptation. Ontologies, on the other hand, provide a formal, standardized way to represent knowledge across entire domains, facilitating interoperability and consistency. Understanding these differences helps in choosing the appropriate tool for a given task or application.

Conclusion

Frames are a fundamental tool in AI for representing and managing knowledge about the world. By providing a structured approach to encapsulate information, frames enhance the ability of AI systems to reason, infer, and make decisions. Despite their challenges, frames remain a crucial component in various AI applications, from natural language processing to robotics. As AI continues to evolve, the role of frames in facilitating intelligent systems will likely become even more significant.

First-Order Logic in Artificial Intelligence

First-order logic (FOL), also known as predicate logic or first-order predicate calculus, is a powerful framework used in various fields such as mathematics, philosophy, linguistics, and computer science. In artificial intelligence (AI), FOL plays a crucial role in knowledge representation, automated reasoning, and natural language processing.

This article delves into the fundamentals of first-order logic, its components, and its applications in AI, providing a comprehensive overview of its significance and functionality.

Fundamentals of First-Order Logic

First-order logic extends propositional logic by incorporating quantifiers and predicates, allowing for more expressive statements about the world. The key components of FOL include constants, variables, predicates, functions, quantifiers, and logical connectives.

1. **Constants**: Constants represent specific objects within the domain of discourse. For example, in a given domain, Alice, 2, and NewYork could be constants.
2. **Variables**: Variables stand for unspecified objects in the domain. Commonly used symbols for variables include x, y, and z.
3. **Predicates**: Predicates are functions that return true or false, representing properties of objects or relationships between them. For example, Likes(Alice, Bob) indicates that Alice likes Bob, and GreaterThan(x, 2) means that x is greater than 2.
4. **Functions**: Functions map objects to other objects. For instance, MotherOf(x) might denote the mother of x.
5. **Quantifiers**: Quantifiers specify the scope of variables. The two main quantifiers are:
    - **Universal Quantifier (∀)**: Indicates that a predicate applies to all elements in the domain.
        - For example, ∀x (Person(x) → Mortal(x)) means "All persons are mortal."
    - **Existential Quantifier (∃)**: Indicates that there is at least one element in the domain for which the predicate holds.
        - For example, ∃x (Person(x) ∧ Likes(x, IceCream)) means "There exists a person who likes ice cream."
6. **Logical Connectives**: Logical connectives include conjunction (∧), disjunction (∨), implication (→), biconditional (↔), and negation (¬). These connectives are used to form complex logical statements.

Syntax and Semantics of First-Order Logic

The syntax of first-order logic specifies the rules for constructing valid expressions, including terms and formulas. Terms are expressions that refer to objects and include constants, variables, and functions.

Formulas are logical statements formed by combining predicates, terms, quantifiers, and logical connectives.

The semantics of FOL define the meaning of the expressions. An interpretation assigns a domain of discourse and provides meaning to the constants, functions, and predicates. A formula is considered true under an interpretation if it accurately describes the relationships and properties of the objects in the domain.

For instance, in the domain of natural numbers, let GreaterThan(x, y) be a predicate that holds if x is greater than y. If x is assigned the value 5 and y is assigned the value 3, then GreaterThan(5, 3) would be true under this interpretation.

Applications of First-Order Logic in AI

First-order logic is instrumental in various AI applications, enabling systems to reason about complex knowledge structures and make informed decisions. Some of the key applications of FOL in AI include:

1. **Knowledge Representation**: FOL provides a robust framework for representing complex relationships and properties of objects. For example, in a medical diagnosis system, predicates can represent symptoms, diseases, and their relationships, allowing the system to reason about potential diagnoses.
2. **Automated Theorem Proving**: Automated theorem proving involves using algorithms to prove mathematical theorems. FOL provides the foundational structure for many theorem proving systems, enabling them to verify software and hardware correctness, prove mathematical theorems, and more.
3. **Natural Language Processing (NLP)**: In NLP, FOL is used to parse and understand natural language by representing the meaning of sentences in a formal, logical manner. This allows AI systems to perform tasks such as question answering, machine translation, and text summarization.
4. **Expert Systems**: Expert systems encode expert knowledge using FOL and reason about it to make decisions. For example, a legal expert system might use FOL to represent legal rules and infer outcomes based on given facts, aiding in legal decision-making processes.
5. **Semantic Web**: The Semantic Web uses FOL to describe relationships between web resources, enabling more intelligent information retrieval and processing. This enhances the ability of AI systems to understand and interact with web content.

Example: Using First-Order Logic for Reasoning

To illustrate the use of FOL in reasoning, consider the following knowledge base:

1. ∀x (Cat(x) → Mammal(x)) (All cats are mammals)
2. ∀x (Mammal(x) → Animal(x)) (All mammals are animals)
3. Cat(Tom) (Tom is a cat)

From these statements, we can infer:

1. Mammal(Tom) (Since Tom is a cat, and all cats are mammals)
2. Animal(Tom) (Since Tom is a mammal, and all mammals are animals)

These inferences demonstrate how FOL can be used to derive new knowledge from existing facts. The ability to reason logically about relationships and properties is a key strength of FOL in AI.

Advanced Concepts in First-Order Logic

1. **Unification**: Unification is the process of finding a substitution that makes two logical expressions identical. It is a fundamental operation in automated reasoning and logic programming, enabling the matching of predicates with variables.
2. **Resolution**: Resolution is a rule of inference used for automated theorem proving. It involves refuting a set of clauses by deriving a contradiction, proving that the original statement is true. Resolution is a powerful method for proving logical theorems in FOL.
3. **Model Checking**: Model checking is a technique used to verify the correctness of systems with respect to a given specification. FOL is used to express the properties and behaviors of the system, enabling the verification of complex systems such as software, hardware, and protocols.
4. **Logic Programming**: Logic programming languages, such as Prolog, use FOL to express programs. These languages allow for declarative programming, where the programmer specifies what needs to be done rather than how to do it. Logic programming is used in various AI applications, including natural language processing, expert systems, and knowledge representation.

Challenges and Limitations

Despite its power and expressiveness, FOL has some limitations and challenges:

1. **Computational Complexity**: Reasoning in FOL can be computationally expensive, particularly for large knowledge bases. Finding efficient algorithms for reasoning in FOL is an ongoing research area in AI.
2. **Expressiveness vs. Decidability**: While FOL is more expressive than propositional logic, it is also undecidable, meaning there is no general algorithm that can determine the truth of all FOL statements. This trade-off between expressiveness and decidability is a key consideration in AI.
3. **Handling Uncertainty**: FOL is not well-suited for handling uncertainty and probabilistic reasoning, which are common in real-world AI applications. Extensions to FOL, such as probabilistic logic and fuzzy logic, have been developed to address these limitations.

Conclusion

First-order logic is a fundamental tool in artificial intelligence, providing a powerful framework for representing and reasoning about complex knowledge structures. Its ability to express relationships, properties, and quantifications makes it indispensable in various AI applications, from knowledge representation to automated reasoning and natural language processing. Despite its challenges and limitations, FOL remains a cornerstone of AI, enabling the development of intelligent systems capable of understanding and interacting with the world in a logical and structured manner.

By leveraging first-order logic, AI systems can achieve higher levels of intelligence and capability, advancing the field and opening new possibilities for future applications.

Example 1: Representing Knowledge in FOL

Question:

Convert the following English statements into First-Order Logic:

1. "All humans are mortal."
2. "Socrates is a human."
3. "Therefore, Socrates is mortal."

Solution:

1. **∀x (Human(x) → Mortal(x))** (For all x, if x is a human, then x is mortal.)
2. **Human(Socrates)** (Socrates is a human.)
3. **Mortal(Socrates)** (By applying modus ponens, Socrates is mortal.)

This follows the standard logic of **syllogism**, where we derive a conclusion based on two premises.

Example 2: Family Relationships

Question:

Express the following relationships using FOL:

- "Every parent loves their child."
- "John is the father of Mary."
- "Who does John love?"

Solution:

1. **∀x ∀y (Parent(x, y) → Loves(x, y))**
   (For all x and y, if x is a parent of y, then x loves y.)
2. **Father(John, Mary) → Parent(John, Mary)**
   (If John is Mary's father, then John is her parent.)
3. **Parent(John, Mary)** (From (2))
4. **Loves(John, Mary)** (From (1) and (3), John loves Mary.)

Thus, **John loves Mary** based on the given knowledge.

Example 3: AI-Based Expert System

Question:

Consider an AI expert system for medical diagnosis. Given:

- "If a person has a fever and a cough, they may have the flu."
- "Alice has a fever and a cough."
- "Does Alice have the flu?"

Solution:

1. **∀x (HasFever(x) ∧ HasCough(x) → HasFlu(x))**
   (If x has a fever and a cough, then x has the flu.)
2. **HasFever(Alice) ∧ HasCough(Alice)** (Given fact)
3. **HasFlu(Alice)** (Applying modus ponens)

Thus, **Alice has the flu** based on the given conditions.

Example 4: Animal Classification

Question:

Given the following statements:

- "All birds can fly, except penguins."
- "Tweety is a bird."
- "Penguins cannot fly."
- "Is Tweety able to fly?"

Solution:

1. **∀x (Bird(x) ∧ ¬Penguin(x) → CanFly(x))**
   (For all x, if x is a bird and not a penguin, then x can fly.)
2. **Bird(Tweety)** (Tweety is a bird.)
3. **¬Penguin(Tweety)** (We assume Tweety is not a penguin.)
4. **CanFly(Tweety)** (From (1), (2), and (3), Tweety can fly.)

Thus, **Tweety can fly**.

Example 5: University System

Question:

Convert the following into FOL:

- "Every student takes at least one course."
- "John is a student."
- "Which course does John take?"

Solution:

1. **∀x (Student(x) → ∃y (Course(y) ∧ Takes(x, y)))**
   (For all x, if x is a student, then there exists at least one y such that y is a course and x takes y.)
2. **Student(John)** (Given fact)
3. **∃y (Course(y) ∧ Takes(John, y))** (From (1) and (2), John must take at least one course.)

Thus, **John is taking at least one course**, but we don't have enough information to specify which one.

Propositional logic in Artificial intelligence

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.

Example:

1. a) It is Sunday.
2. b) The Sun rises from West (False proposition)
3. c) 3+3= 7(False proposition)
4. d) 5 is a prime number.

Following are some basic facts about propositional logic:

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and **logical connectives**.
- These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- A proposition formula which is always false is called **Contradiction**.
- A proposition formula which has both true and false values is called
- Statements which are questions, commands, or opinions are not propositions such as "**Where is Rohini**", "**How are you**", "**What is your name**", are not propositions.

Syntax of propositional logic:

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

a. **Atomic Propositions**

b. **Compound propositions**

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

Example:

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

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

Example:

1. a) "It is raining today, and street is wet."
2. b) "Ankit is a doctor, and his clinic is in Mumbai."

Logical Connectives:

Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

1. **Negation:** A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.
2. **Conjunction:** A sentence which has ∧ connective such as, **P ∧ Q** is called a conjunction.
   Example: Rohan is intelligent and hardworking. It can be written as, **P= Rohan is intelligent**,
   Q= Rohan is hardworking. → P∧ Q.
3. **Disjunction:** A sentence which has ∨ connective, such as **P ∨ Q**. is called disjunction, where P and Q are the propositions.

Example: "Ritika is a doctor or Engineer", Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as **P ∨ Q**.

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

   If it is raining, then the street is wet.

   Let P= It is raining, and Q= Street is wet, so it is represented as P → Q

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

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

Following is the summarized table for Propositional Logic Connectives:

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

Truth Table:

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

For Negation:

P	¬ P
True	**False**
False	**True**

For Conjunction:

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

For disjunction:

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

For Implication:

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

For Biconditional:

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

Truth table with three propositions:

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

P	Q	R	¬R	Pv Q	PvQ→¬R
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

Precedence of connectives:

Just like arithmetic operators, there is a precedence order for propositional connectors or logical operators. This order should be followed while evaluating a propositional problem. Following is the list of the precedence order for operators:

Precedence	Operators
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction(AND)
Fourth Precedence	Disjunction(OR)
Fifth Precedence	Implication
Six Precedence	Biconditional

> Note: For better understanding use parenthesis to make sure of the correct interpretations. Such as ¬Rv Q, It can be interpreted as (¬R) v Q.

Logical equivalence:

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

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

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

Properties of Operators:

- **Commutativity:**
  - P∧ Q= Q ∧ P, or
  - P ∨ Q = Q ∨ P.
- **Associativity:**
  - (P ∧ Q) ∧ R= P ∧ (Q ∧ R),
  - (P ∨ Q) ∨ R= P ∨ (Q ∨ R)
- **Identity element:**
  - P ∧ True = P,
  - P ∨ True= True.
- **Distributive:**
  - P∧ (Q ∨ R) = (P ∧ Q) ∨ (P ∧ R).
  - P ∨ (Q ∧ R) = (P ∨ Q) ∧ (P ∨ R).
- **DE Morgan's Law:**
  - ¬ (P ∧ Q) = (¬P) ∨ (¬Q)
  - ¬ (P ∨ Q) = (¬ P) ∧ (¬Q).
- **Double-negation elimination:**
  - ¬ (¬P) = P.

Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic. Example:

a. **All the girls are intelligent.**

b. **Some apples are sweet.**

- Propositional logic has limited expressive power.
- In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

First-Order Logic in Artificial intelligence

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

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

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

First-Order logic:

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

Syntax of First-Order logic:

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

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

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

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as **Predicate (term1, term2, ..., term n)**.

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

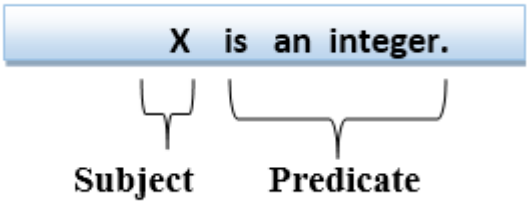
Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- **Subject:** Subject is the main part of the statement.
- **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

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



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
    - **Universal Quantifier, (for all, everyone, everything)**
    - **Existential quantifier, (for some, at least one).**

Universal Quantifier:

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

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

Note: In universal quantifier we use implication "→".

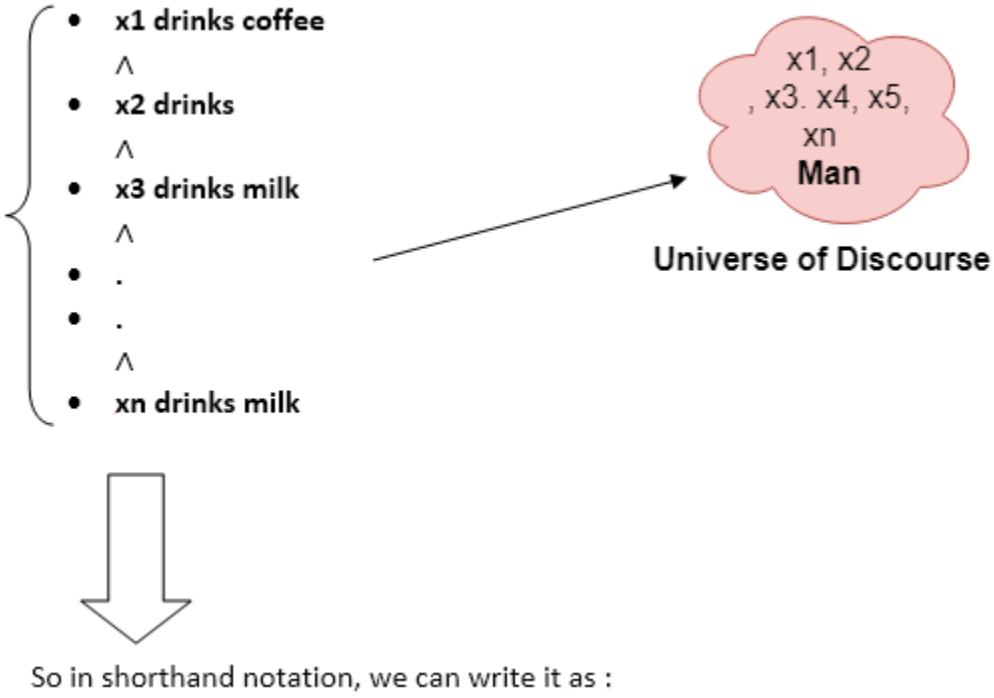
If x is a variable, then ∀x is read as:

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

Example:

All man drink coffee.

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



∀x man(x) → drink (x, coffee).

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

Existential Quantifier:

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

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

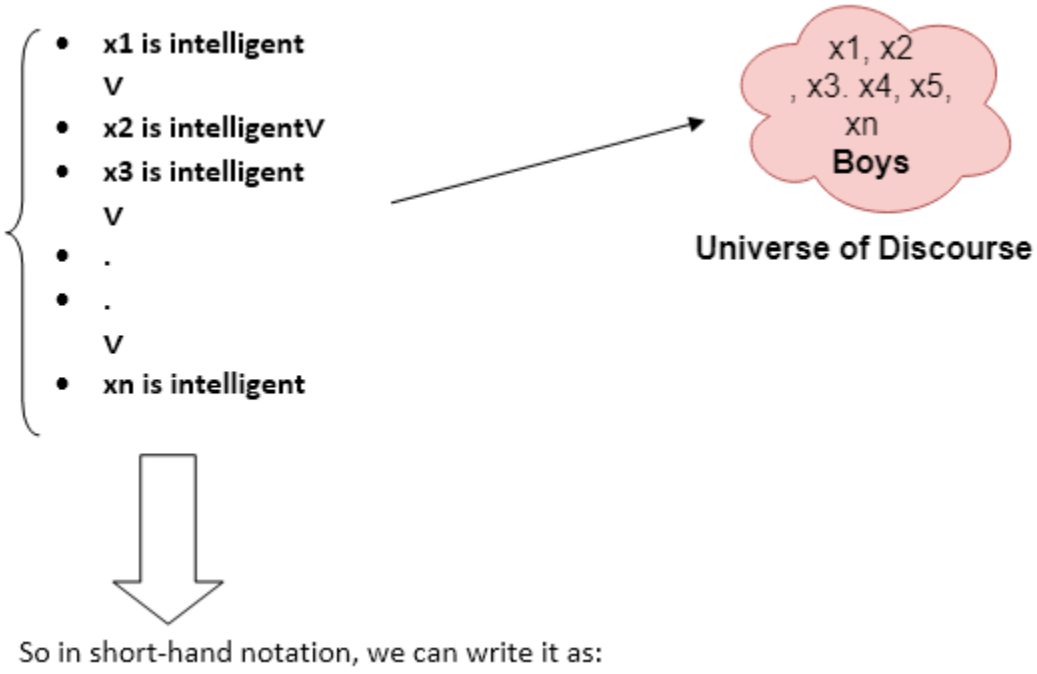
Note: In Existential quantifier we always use AND or Conjunction symbol (∧).

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

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

Example:

Some boys are intelligent.



∃x: boys(x) ∧ intelligent(x)

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

Points to remember:

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

Properties of Quantifiers:

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

Some Examples of FOL using quantifier:

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

∀x bird(x) →fly(x).

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

∀x man(x) → respects (x, parent).

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

∃x boys(x) → play(x, cricket).

4. Not all students like both Mathematics and Science.
In this question, the predicate is "**like(x, y)," where x= student, and y= subject**.
Since there are not all students, so we will use **∀ with negation, so** following representation for this:

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

5. Only one student failed in Mathematics.
In this question, the predicate is "**failed(x, y)," where x= student, and y= subject**.
Since there is only one student who failed in Mathematics, so we will use following representation for this:

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

Free and Bound Variables:

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

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

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

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

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

Representing INSTANCE and ISA Relationships

Specific attributes **instance** and **isa** play important role particularly in a useful form of reasoning called property inheritance.

The predicates instance and isa explicitly captured the relationships they are used to express, namely class membership and class inclusion.

Below figure shows the first five sentences of the last section represented in logic in three different ways.

The first part of the figure contains the representations we have already discussed. In these representations, class membership is represented with unary predicates (such as Roman), each of which corresponds to a class.

Asserting that P(x) is true is equivalent to asserting that x is an instance (or element) of P.

The second part of the figure contains representations that use the **instance** predicate explicitly.

1.	Man(Marcus).
2.	Pompeian(Marcus).
3.	∀x: Pompeian(x) → Roman(x).
4.	ruler(Caesar).
5.	∀x: Roman(x) → loyalto(x, Caesar) ∨ hate(x, Caesar).
1.	instance(Marcus, man).
2.	instance(Marcus, Pompeian).
3.	∀x: instance(x, Pompeian) → instance(x, Roman).
4.	instance(Caesar, ruler).
5.	∀x: instance(x, Roman). → loyalto(x, Caesar) ∨ hate(x, Caesar).
1.	instance(Marcus, man).
2.	instance(Marcus, Pompeian).
3.	isa(Pompeian, Roman)
4.	instance(Caesar, ruler).
5.	∀x: instance(x, Roman). → loyalto(x, Caesar) ∨ hate(x, Caesar).
6.	∀x: ∀y: ∀z: instance(x, y) ∧ isa(y, z)→ instance(x, z).

Figure: Three ways of representing class membership

The predicate **instance** is a binary one, whose first argument is an object and whose second argument is a class to which the object belongs.

But these representations do not use an explicit **isa** predicate.

Instead, subclass relationships, such as that between Pompeians and Romans, are described as shown in sentence 3.

The implication rule states that if an object is an instance of the subclass Pompeian then it is an instance of the superclass Roman.

Note that this rule is equivalent to the standard set-theoretic definition of the subclass-superclass relationship.

The third part contains representations that use both the **instance** and **isa** predicates explicitly.

The use of the **isa** predicate simplifies the representation of sentence 3, but it requires that one additional axiom (shown here as number 6) be provided.

Resolution Method in AI

Resolution Method in AI

Resolution method is an inference rule which is used in both Propositional as well as First-order Predicate Logic in different ways. This method is basically used for proving the satisfiability of a sentence. In resolution method, we use **Proof by Refutation** technique to prove the given statement.

The key idea for the resolution method is to use the knowledge base and negated goal to obtain null clause(which indicates contradiction). Resolution method is also called **Proof by Refutation**. Since the knowledge base itself is consistent, the contradiction must be introduced by a negated goal. As a result, we have to conclude that the original goal is true.

Resolution Method in Propositional Logic

In propositional logic, resolution method is the only inference rule which gives a new clause when two or more clauses are coupled together.

Using propositional resolution, it becomes easy to make a theorem prover sound and complete for all.

The process followed to convert the propositional logic into resolution method contains the below steps:

- Convert the given axiom into clausal form, i.e., disjunction form.
- Apply and proof the given goal using negation rule.
- Use those literals which are needed to prove.

But, before solving problems using Resolution method, let's understand two normal forms

Conjunctive Normal Form(CNF)

In propositional logic, the resolution method is applied only to those clauses which are disjunction of literals. **There are followin steps used to convert into CNF:**

1) Eliminate bi-conditional implication by replacing $A \Leftrightarrow B$ with $(A \rightarrow B) \wedge (B \rightarrow A)$

2) Eliminate implication by replacing $A \rightarrow B$ with $\neg A \vee B$.

3) In CNF, negation(¬) appears only in literals, therefore we move it inwards as:

- **$\neg (\neg A) \equiv A$** (double-negation elimination
- **$\neg (A \wedge B) \equiv (\neg A \vee \neg B)$** (De Morgan)
- **$\neg (A \vee B) \equiv (\neg A \wedge \neg B)$** (De Morgan)

4) Finally, using distributive law on the sentences, and form the CNF as:

$(A_1 \vee B_1) \wedge (A_2 \vee B_2) \wedge \ldots \wedge (A_n \vee B_n)$.

Note: CNF can also be described as AND of ORS

Disjunctive Normal Form (DNF)

This is a reverse approach of CNF. The process is similar to CNF with the following difference:

$(A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \ldots \vee (A_n \wedge B_n)$. In DNF, it is OR of ANDS, a sum of products, or a cluster concept, whereas, in CNF, it is ANDs of Ors.

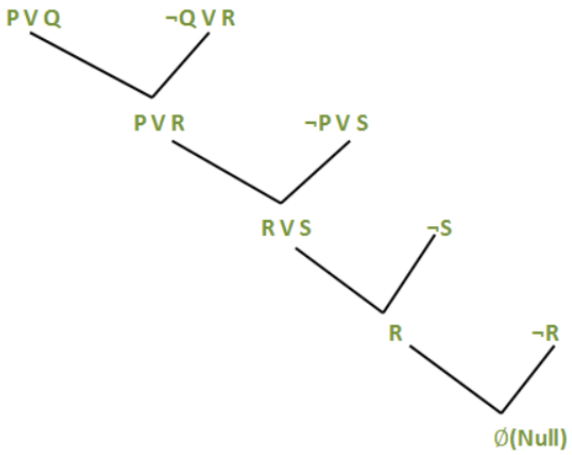
Applying resolution method:

In (2), Q → R will be converted as (¬Q V R)

In (3), P → S will be converted as (¬P V S)

Negation of Goal (¬R): It will not rain.

Finally, apply the rule as shown below:



After applying Proof by Refutation (Contradiction) on the goal, the problem is solved, and it has terminated with a **Null clause (Ø).** Hence, the goal is achieved. Thus, It is not raining.

Note: We can have many examples of Proposition logic which can be proved with the help of Propositional resolution method.

Resolution Method in FOPl/Predicate Logic

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

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

- Convert the given axiom into CNF, i.e., a conjunction of clauses. Each clause should be dis-junction of literals.
- Apply negation on the goal given.
- Use literals which are required and prove it.
- Unlike propositional logic, FOPL literals are complementary if one unifies with the negation of other literal.

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

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

{Bird(F(x)) V ¬Kills(G(x),x)}

The rule applied on the following example is called **Binary Resolution** as it has solved exactly two literals. But, binary resolution is not complete. An alternative approach is to extend the **factoring** i.e., to remove redundant literals to the first order case. Thus, the combination of binary resolution and factoring is complete.

Conjunctive Normal Form

There are following steps used to convert into CNF:

- Eliminate the implications as:

∀x: A(x) → B(x) with {¬ x: ¬A(∀x) V B(x)}

- Move negation (¬) inwards as:

¬∀x: A becomes ∃x: ¬A and,

Example OF Propositional Resolution

Consider the following Knowledge Base:

1. The humidity is high or the sky is cloudy.
2. If the sky is cloudy, then it will rain.
3. If the humidity is high, then it is hot.
4. It is not hot.

Goal: It will rain.

Use propositional logic and apply resolution method to prove that the goal is derivable from the given knowledge base.

Solution: Let's construct propositions of the given sentences one by one:

1. Let, P: Humidity is high.

Q: Sky is cloudy.

It will be represented as **P V Q.**

2) Q: Sky is cloudy. ...**from(1)**

Let, R: It will rain.

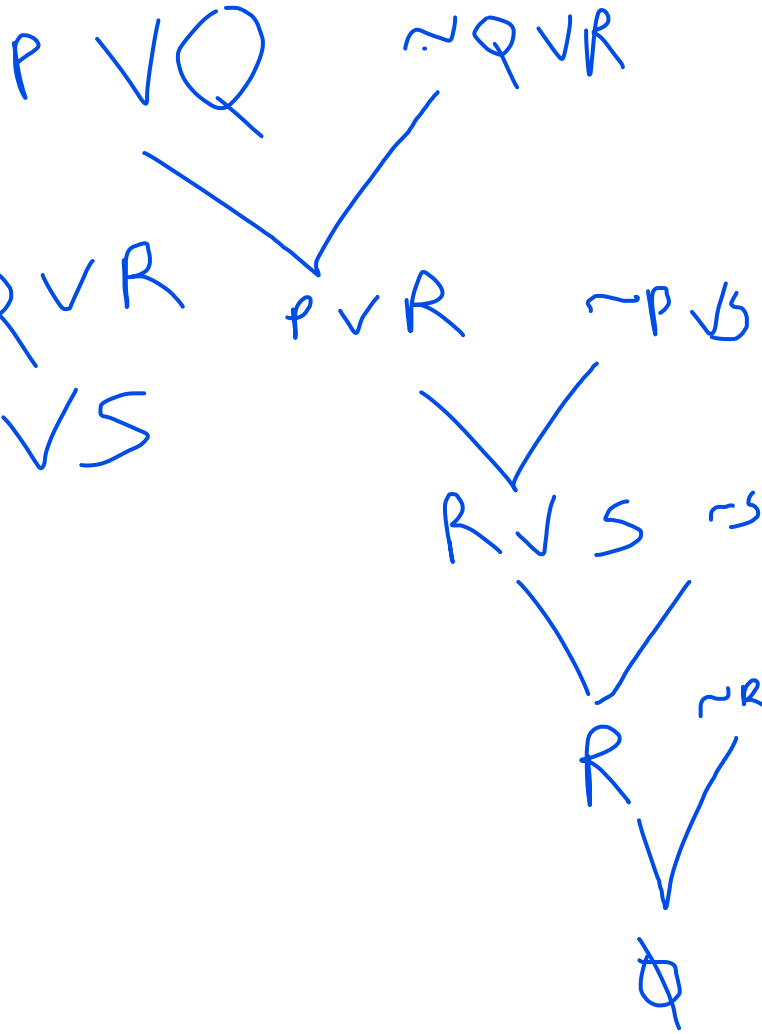
It will be represented as b**Q → R.**

3) P: Humidity is high. ...**from(1)**

Let, S: It is hot.

It will be represented as **P → S.**

4) ¬S: It is not hot.



¬∃x: A becomes ∀x: ¬A

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

- **Standardize variables:** If two sentences use same variable, it is required to change the name of one variable. This step is taken so as to remove the confusion when the quantifiers will be dropped.

For example: { ∀x: A(x) V ∀x: B(x)}

- **Skolemize:** It is the process of removing existential quantifier through elimination.
- **Drop universal quantifiers:** If we are on this step, it means all remaining variables must be universally quantified. Drop the quantifier.
- **Distribute V over Λ:** Here, the nested conjunction and disjunction are flattened.

Example of FOPL resolution

Consider the following knowledge base:

1. Gita likes all kinds of food.
2. Mango and chapati are food.
3. Gita eats almond and is still alive.
4. Anything eaten by anyone and is still alive is food.

Goal: Gita likes almond.

Solution: Convert the given sentences into FOPL as:

Let, x be the light sleeper.

1. ∀x: food(x) → likes(Gita,x)
2. food(Mango),food(chapati)
3. ∀x∀y: eats(x,y) Λ ¬ killed(x → food(y)
4. eats(Gita, almonds) Λ alive(Gita)
5. ∀x: ¬killed(x) → alive(x)
6. ∀x: alive(x) → ¬killed(x)

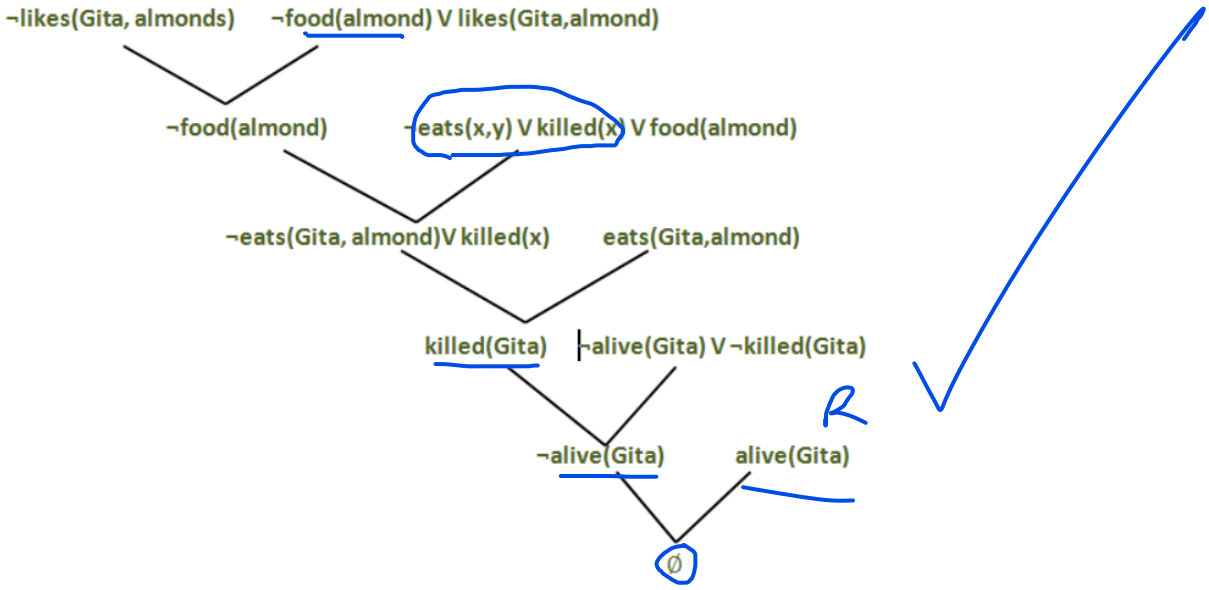
Goal: likes(Gita, almond)

Negated goal: ¬likes(Gita, almond)

Now, rewrite in CNF form:

1. ¬food(x) V likes(Gita, x)
2. food(Mango),food(chapati)
3. ¬eats(x,y) V killed(x) V food(y)
4. eats(Gita, almonds), alive(Gita)
5. killed(x) V alive(x)
6. ¬alive(x) V ¬killed(x)

Finally, construct the resolution graph:



Hence, we have achieved the given goal with the help of Proof by Contradiction. Thus, it is proved that Gita likes almond.

Unification and Resolution in FOPL(first order predicate logic)

What is Unification?

- Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
- It takes two literals as input and makes them identical using substitution.
- Let Ψ_1 and Ψ_2 be two atomic sentences and σ be a unifier such that, $\boldsymbol{\Psi_1\sigma = \Psi_2\sigma}$, then it can be expressed as **UNIFY(Ψ_1, Ψ_2)**.
- **Example: Find the MGU for Unify{King(x), King(John)}**

Let Ψ_1 = King(x), Ψ_2 = King(John),

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

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

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

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

P(x, y)..... (i)
P(a, f(z))..... (ii)

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

Conditions for Unification:

Following are some basic conditions for unification:

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
- Number of Arguments in both expressions must be identical.
- Unification will fail if there are two similar variables present in the same expression.

Unification Algorithm:

Algorithm: Unify(Ψ_1, Ψ_2)

```
Step. 1: If Ψ1 or Ψ2 is a variable or constant, then:
      a) If Ψ1 or Ψ2 are identical, then return NIL.
      b) Else if Ψ1is a variable,
            a. then if Ψ1 occurs in Ψ2, then return FAILURE
            b. Else return { (Ψ2/ Ψ1)}.
      c) Else if Ψ2 is a variable,
            a. If Ψ2 occurs in Ψ1 then return FAILURE,
            b. Else return {( Ψ1/ Ψ2)}.
      d) Else return FAILURE.
Step.2: If the initial Predicate symbol in Ψ1 and Ψ2 are not same, then
return FAILURE.
Step. 3: IF Ψ1 and Ψ2 have a different number of arguments, then return
FAILURE.
Step. 4: Set Substitution set(SUBST) to NIL.
Step. 5: For i=1 to the number of elements in Ψ1.
      a) Call Unify function with the ith element of Ψ1 and ith element of
Ψ2, and put the result into S.
      b) If S = failure then returns Failure
      c) If S ≠ NIL then do,
            a. Apply S to the remainder of both L1 and L2.
            b. SUBST= APPEND(S, SUBST).
Step.6: Return SUBST.
```

Implementation of the Algorithm

Step.1: Initialize the substitution set to be empty.

Step.2: Recursively unify atomic sentences:

a. Check for Identical expression match.

b. If one expression is a variable v_i, and the other is a term t_i which does not contain variable v_i, then:

    a. Substitute t_i / v_i in the existing substitutions

    b. Add t_i /v_i to the substitution setlist.

    c. If both the expressions are functions, then function name must be similar, and the number of arguments must be the same in both the expression.

For each pair of the following atomic sentences find the most general unifier (If exist).

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

Sol: S_0 => Here, Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(X, X)
SUBST θ= {f(a) / X}
S1 => Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(f(a), f(a))
SUBST θ= {f(a) / g(y)}, **Unification failed**.

Unification is not possible for these expressions.

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

Here, Ψ_1 = p(b, X, f(g(Z))) , and Ψ_2 = p(Z, f(Y), f(Y))
S_0 => { p(b, X, f(g(Z))); p(Z, f(Y), f(Y))}
SUBST θ={b/Z}

S_1 => { p(b, X, f(g(b))); p(b, f(Y), f(Y))}
SUBST θ={f(Y) /X}

S_2 => { p(b, f(Y), f(g(b))); p(b, f(Y), f(Y))}
SUBST θ= {g(b) /Y}

S_1 => { p(b, X, f(g(b))); p(b, f(Y), f(Y))}
SUBST θ={f(Y) /X}

S_2 => { p(b, f(Y), f(g(b))); p(b, f(Y), f(Y))}
SUBST θ= {g(b) /Y}

S_2 => { p(b, f(g(b)), f(g(b)); p(b, f(g(b)), f(g(b))} **Unified Successfully.**
And Unifier = { b/Z, f(Y) /X , g(b) /Y}.

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

Here, Ψ_1 = {p (X, X), and Ψ_2 = p (Z, f(Z))
S_0 => {p (X, X), p (Z, f(Z))}
SUBST θ= {X/Z}
S1 => {p (Z, Z), p (Z, f(Z))}
SUBST θ= {f(Z) / Z}, **Unification Failed**.

Hence, unification is not possible for these expressions.

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

Here, Ψ_1 = {prime(11) , and Ψ_2 = prime(y)}
S_0 => {prime(11) , prime(y)}
SUBST θ= {11/y}

S_1 => {prime(11) , prime(11)} , **Successfully unified.**
Unifier: {11/y}.

5. Find the MGU of Q(a, g(x, a), f(y)), Q(a, g(f(b), a), x)}

Here, Ψ_1 = Q(a, g(x, a), f(y)), and Ψ_2 = Q(a, g(f(b), a), x)
S_0 => {Q(a, g(x, a), f(y)); Q(a, g(f(b), a), x)}
SUBST θ= {f(b)/x}
S_1 => {Q(a, g(f(b), a), f(y)); Q(a, g(f(b), a), f(b))}

SUBST θ= {b/y}
S_1 => {Q(a, g(f(b), a), f(b)); Q(a, g(f(b), a), f(b))}, **Successfully Unified.**

Unifier: [a/a, f(b)/x, b/y].

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

Here, Ψ_1 = knows(Richard, x), and Ψ_2 = knows(Richard, John)
S_0 => { knows(Richard, x); knows(Richard, John)}
SUBST θ= {John/x}
S_1 => { knows(Richard, John); knows(Richard, John)}, **Successfully Unified.**
Unifier: {John/x}

Resolution in FOL

Resolution

Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by a Mathematician John Alan Robinson in the year 1965.

Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.

Clause: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.

Conjunctive Normal Form: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**

The resolution inference rule:

The resolution rule for first-order logic is simply a lifted version of the propositional rule. Resolution can resolve two clauses if they contain complementary literals, which are assumed to be standardized apart so that they share no variables.

$$\frac{l_1 \vee \ldots\ldots \vee l_k, \qquad m_1 \vee \ldots\ldots \vee m_n}{\text{SUBST}(\theta, l_1 \vee \ldots\ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots\ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots\ldots \vee m_n)}$$

Where $\mathbf{l_i}$ and $\mathbf{m_j}$ are complementary literals.

This rule is also called the **binary resolution rule** because it only resolves exactly two literals.

Example:

We can resolve two clauses which are given below:

[Animal (g(x) V Loves (f(x), x)] **and** **[¬ Loves(a, b) V ¬Kills(a, b)]**

Where two complimentary literals are: **Loves (f(x), x) and ¬ Loves (a, b)**

These literals can be unified with unifier **θ= [a/f(x), and b/x]** , and it will generate a resolvent clause:

[Animal (g(x) V ¬ Kills(f(x), x)].

Steps for Resolution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

To better understand all the above steps, we will take an example in which we will apply resolution.

Example:

a. **John likes all kind of food.**

b. **Apple and vegetable are food**

c. **Anything anyone eats and not killed is food.**

d. **Anil eats peanuts and still alive**

e. **Harry eats everything that Anil eats.**

Prove by resolution that:

f. **John likes peanuts."**

Step-1: Conversion of Facts into FOL

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

a. ∀x: food(x) → likes(John, x)

b. food(Apple) ∧ food(vegetables)

c. ∀x ∀y: eats(x, y) ∧ ¬ killed(x) → food(y)

d. eats (Anil, Peanuts) ∧ alive(Anil).

e. ∀x : eats(Anil, x) → eats(Harry, x)

f. ∀x: ¬ killed(x) → alive(x) } **added predicates.**

g. ∀x: alive(x) →¬ killed(x) }

h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

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

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

  1. ∀x ¬ food(x) V likes(John, x)
  2. food(Apple) ∧ food(vegetables)
  3. ∀x ∀y ¬ [eats(x, y) ∧ ¬ killed(x)] V food(y)
  4. eats (Anil, Peanuts) ∧ alive(Anil)
  5. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  6. ∀x¬ [¬ killed(x)] V alive(x)
  7. ∀x ¬ alive(x) V ¬ killed(x)
  8. likes(John, Peanuts).

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

  1. ∀x ¬ food(x) V likes(John, x)

2. food(Apple) Λ food(vegetables)
3. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
4. eats (Anil, Peanuts) Λ alive(Anil)
5. ∀x ¬ eats(Anil, x) V eats(Harry, x)
6. ∀x ¬killed(x)] V alive(x)
7. ∀x ¬ alive(x) V ¬ killed(x)
8. likes(John, Peanuts).

- **Rename variables or standardize variables**

1. ∀x ¬ food(x) V likes(John, x)
2. food(Apple) Λ food(vegetables)
3. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)
4. eats (Anil, Peanuts) Λ alive(Anil)
5. ∀w¬ eats(Anil, w) V eats(Harry, w)
6. ∀g ¬killed(g)] V alive(g)
7. ∀k ¬ alive(k) V ¬ killed(k)
8. likes(John, Peanuts).

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

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

  1. ¬ food(x) V likes(John, x)
  2. food(Apple)
  3. food(vegetables)
  4. ¬ eats(y, z) V killed(y) V food(z)
  5. eats (Anil, Peanuts)
  6. alive(Anil)
  7. ¬ eats(Anil, w) V eats(Harry, w)
  8. killed(g) V alive(g)
  9. ¬ alive(k) V ¬ killed(k)
  10. likes(John, Peanuts).

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

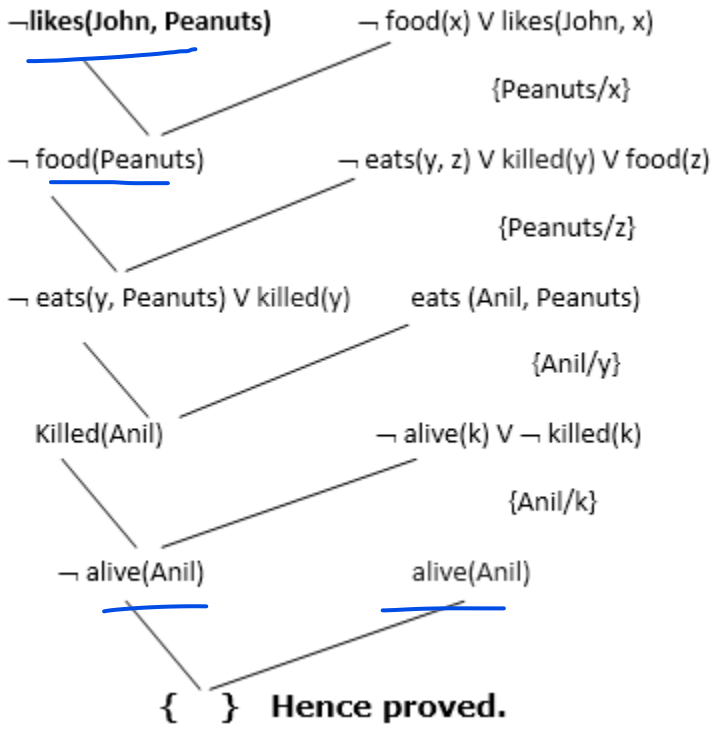
- **Distribute conjunction Λ over disjunction ¬.**
  This step will not make any change in this problem.

Step-3: Negate the statement to be proved

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

Step-4: Draw Resolution graph:

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



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

Explanation of Resolution graph:

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

Forward Chaining and backward chaining in AI

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

Inference engine:

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

1. **Forward chaining**
2. **Backward chaining**

Horn Clause and Definite clause:

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

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

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

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

It is equivalent to p ∧ q → k.

A. Forward Chaining

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.

- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

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

Example:

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

Prove that **"Robert is criminal."**

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

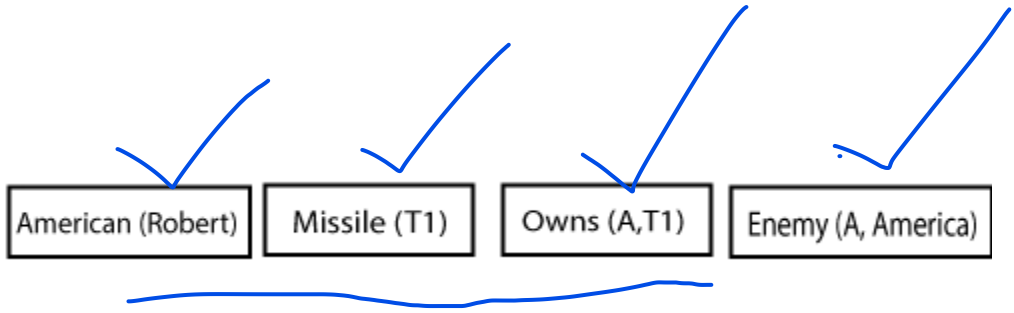
Facts Conversion into FOL:

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)
  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)
- Country A has some missiles. **?p Owns(A, p) ∧ Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.
  Owns(A, T1) .....(2)
  Missile(T1) .....(3)
- All of the missiles were sold to country A by Robert.
  ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) .....(4)
- Missiles are weapons.
  Missile(p) → Weapons (p) .....(5)
- Enemy of America is known as hostile.
  Enemy(p, America) →Hostile(p) .....(6)
- Country A is an enemy of America.
  Enemy (A, America) .....(7)
- Robert is American
  American(Robert). .....(8)

Forward chaining proof:

Step-1:

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



Step-2:

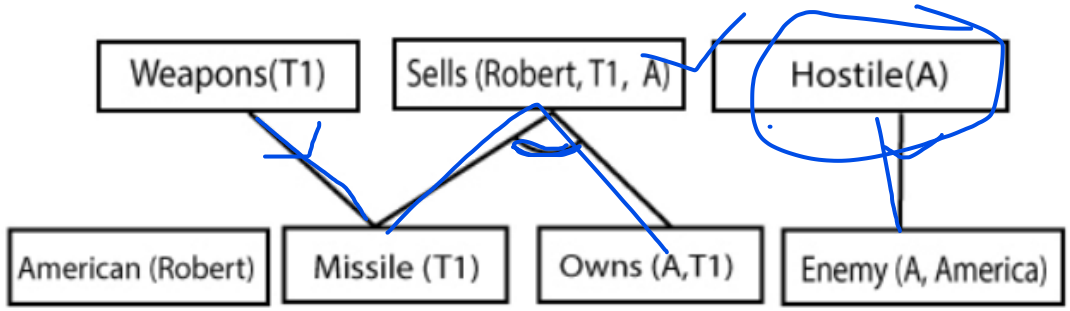
At the second step, we will see those facts which infer from available facts and with satisfied premises.

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

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

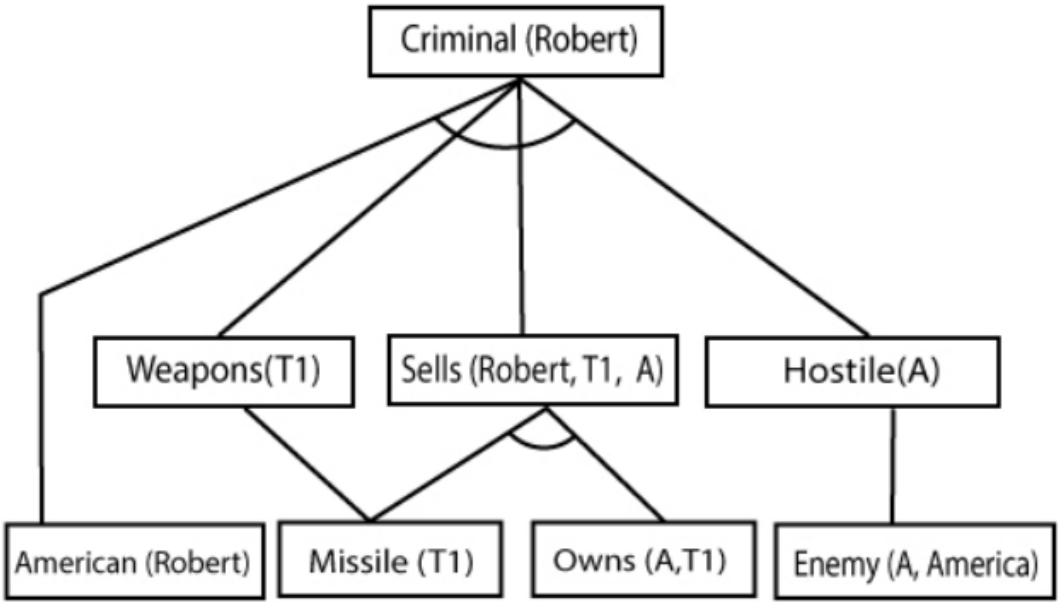
Rule-(4) satisfy with the substitution {p/T1}, **so Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

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



Step-3:

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



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

B. Backward Chaining:

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

Properties of backward chaining:

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

Example:

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

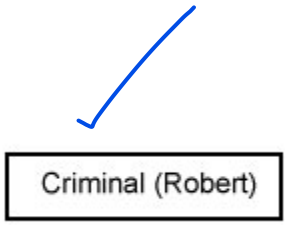
- **American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)**
  Owns(A, T1) .....(2)
- **Missile(T1)**
- **?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) .....(4)**
- **Missile(p) → Weapons (p) .....(5)**
- **Enemy(p, America) →Hostile(p) .....(6)**
- **Enemy (A, America) .....(7)**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

Backward-Chaining proof:

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

Step-1:

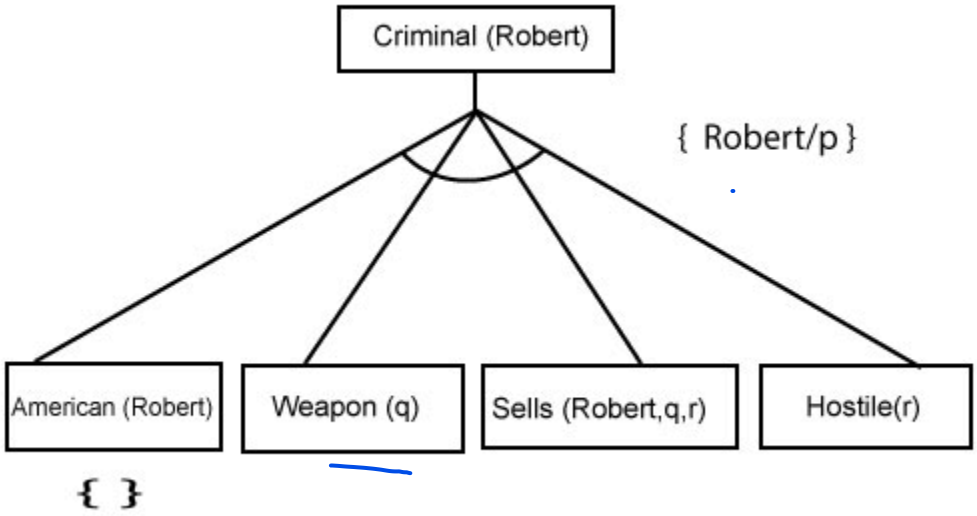
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.



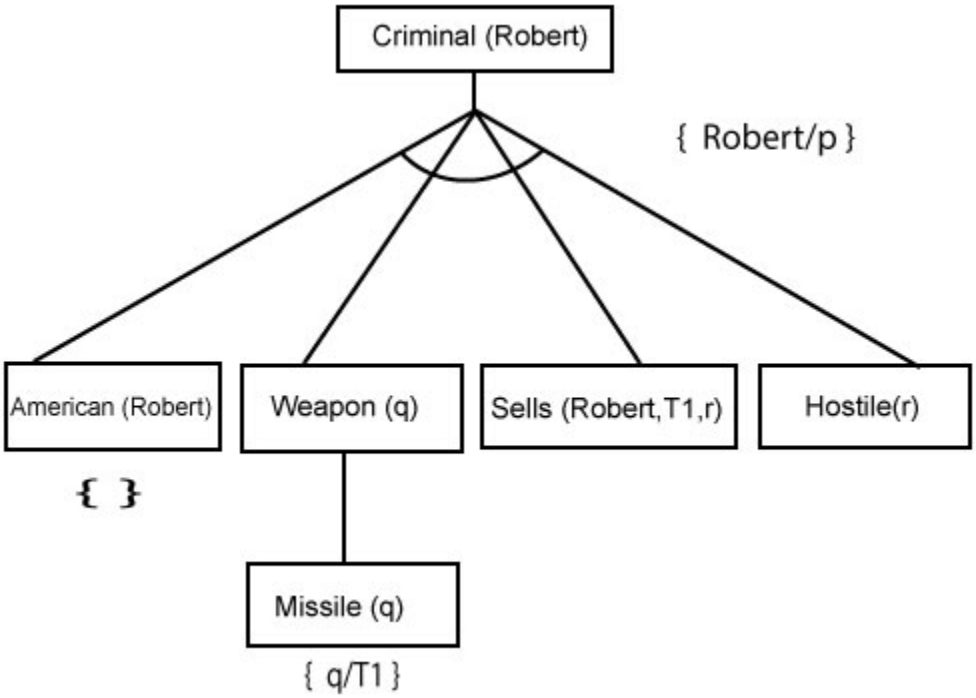
Step-2:

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

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

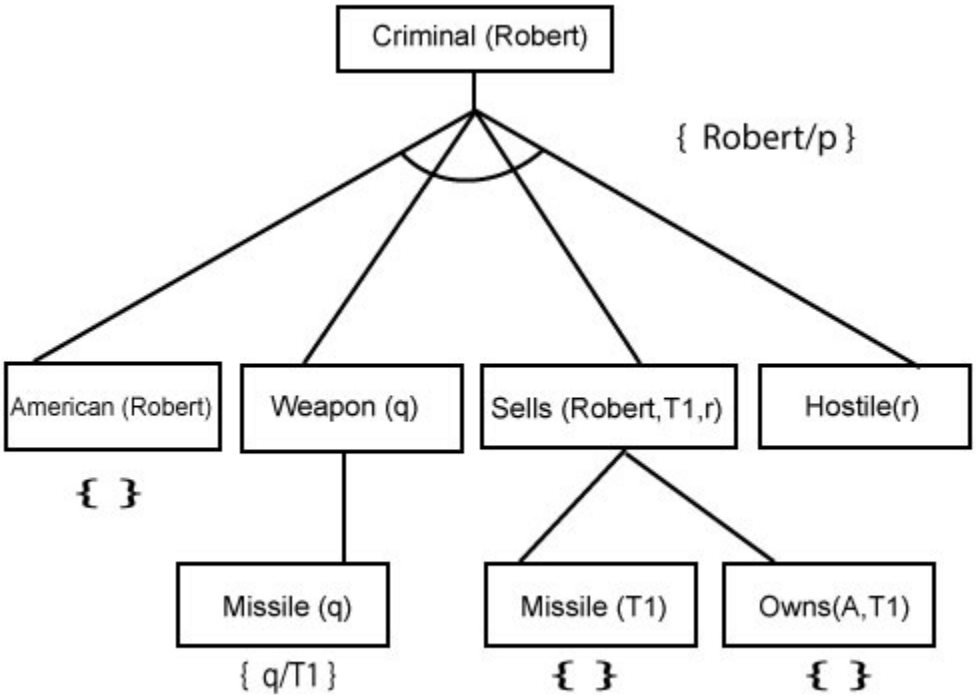


Step-3:t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



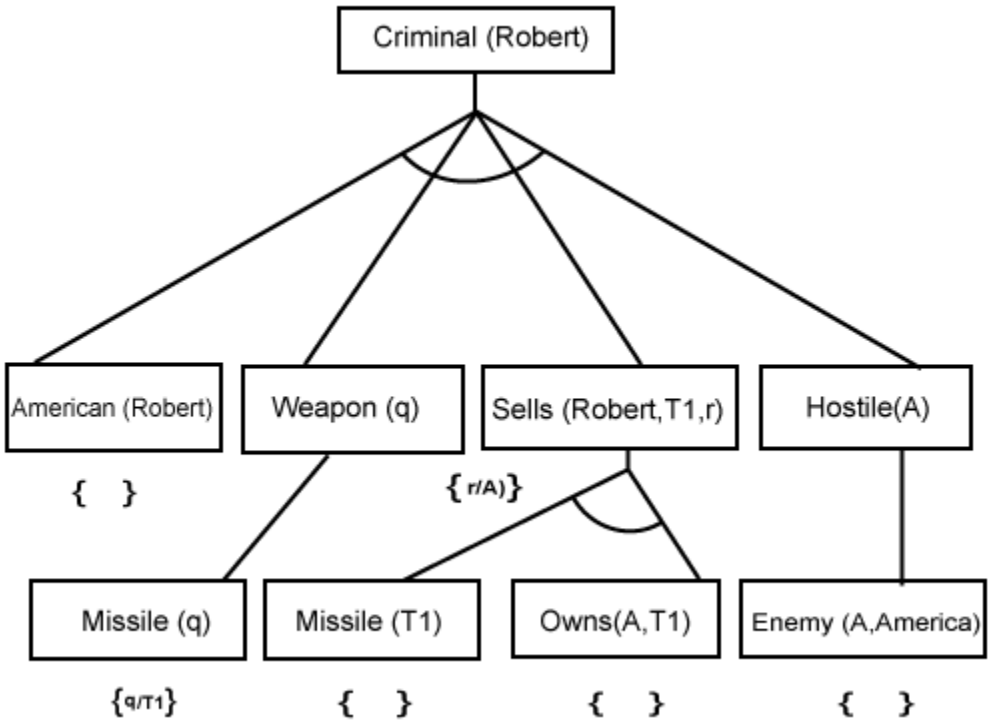
Step-4:

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



Step-5:

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



Difference between backward chaining and forward chaining

Following is the difference between the forward chaining and backward chaining:

- Forward chaining as the name suggests, start from the known facts and move forward by applying inference rules to extract more data, and it continues until it reaches to the goal, whereas backward chaining starts from the goal, move backward by using inference rules to determine the facts that satisfy the goal.
- Forward chaining is called a **data-driven** inference technique, whereas backward chaining is called a **goal-driven** inference technique.
- Forward chaining is known as the **down-up** approach, whereas backward chaining is known as a **top-down** approach.
- Forward chaining uses **breadth-first search** strategy, whereas backward chaining uses **depth-first search** strategy.
- Forward and backward chaining both applies **Modus ponens** inference rule.
- Forward chaining can be used for tasks such as **planning, design process monitoring, diagnosis, and classification**, whereas backward chaining can be used for **classification and diagnosis tasks**.

- Forward chaining can be like an exhaustive search, whereas backward chaining tries to avoid the unnecessary path of reasoning.
- In forward-chaining there can be various ASK questions from the knowledge base, whereas in backward chaining there can be fewer ASK questions.
- Forward chaining is slow as it checks for all the rules, whereas backward chaining is fast as it checks few required rules only.

S. No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
8.	It operates in the forward direction.	It operates in the backward direction.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.

Forward Chaining and backward chaining in AI

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

Inference engine:

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

a. **Forward chaining**

b. **Backward chaining**

Horn Clause and Definite clause:

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

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

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

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

It is equivalent to p ∧ q → k.

A. Forward Chaining

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start

with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Properties of Forward-Chaining:

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

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

Example:

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

Prove that **"Robert is criminal."**

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

Facts Conversion into FOL:

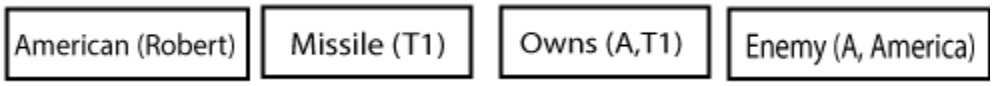
- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)
  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)
- Country A has some missiles. **?p Owns(A, p) ∧ Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.
  Owns(A,T1) .....(2)
  Missile(T1) .....(3)

- All of the missiles were sold to country A by Robert.
  ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ……(4)
- Missiles are weapons.
  Missile(p) → Weapons (p) …….(5)
- Enemy of America is known as hostile.
  Enemy(p, America) →Hostile(p) ……..(6)
- Country A is an enemy of America.
  Enemy (A, America) ……..(7)
- Robert is American
  American(Robert). ……..(8)

Forward chaining proof:

Step-1:

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



Step-2:

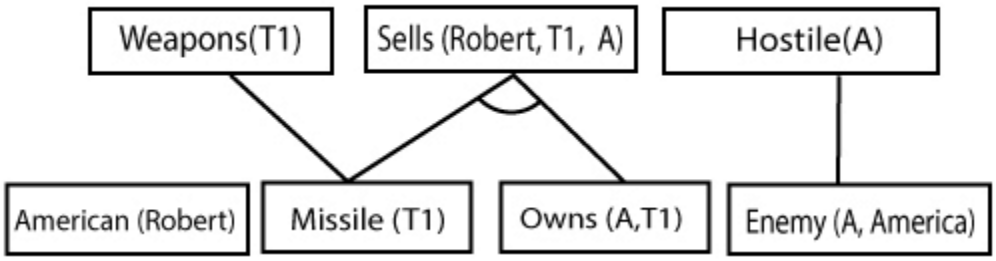
At the second step, we will see those facts which infer from available facts and with satisfied premises.

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

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

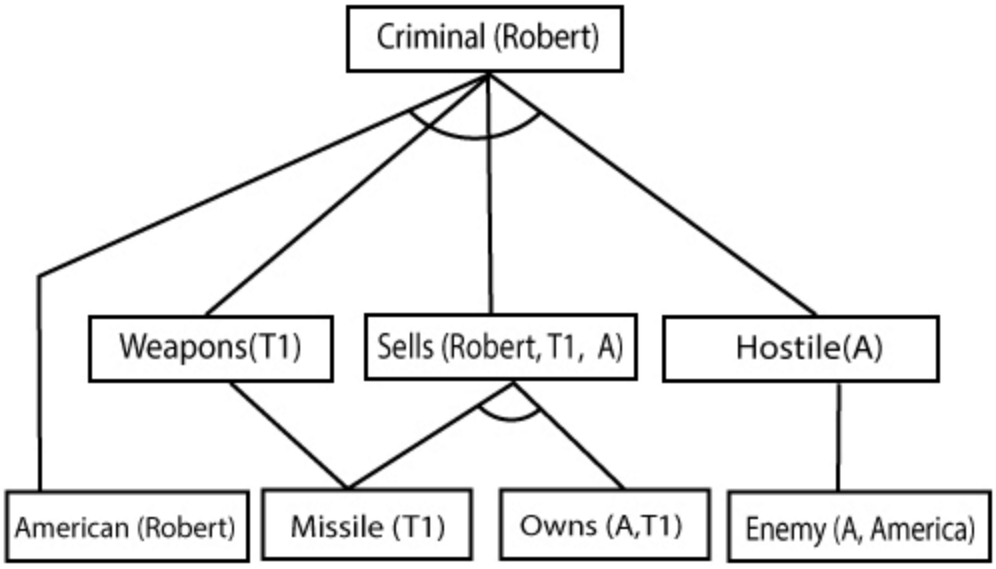
Rule-(4) satisfy with the substitution {p/T1}, **so Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

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



Step-3:

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



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

B. Backward Chaining:

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

Properties of backward chaining:

- It is known as a top-down approach.

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

Example:

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

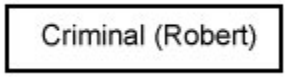
- **American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)**
  Owns(A, T1) ……..(2)
- **Missile(T1)**
- **?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ……(4)**
- **Missile(p) → Weapons (p) …….(5)**
- **Enemy(p, America) →Hostile(p) ……..(6)**
- **Enemy (A, America) ……..(7)**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

Backward-Chaining proof:

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

Step-1:

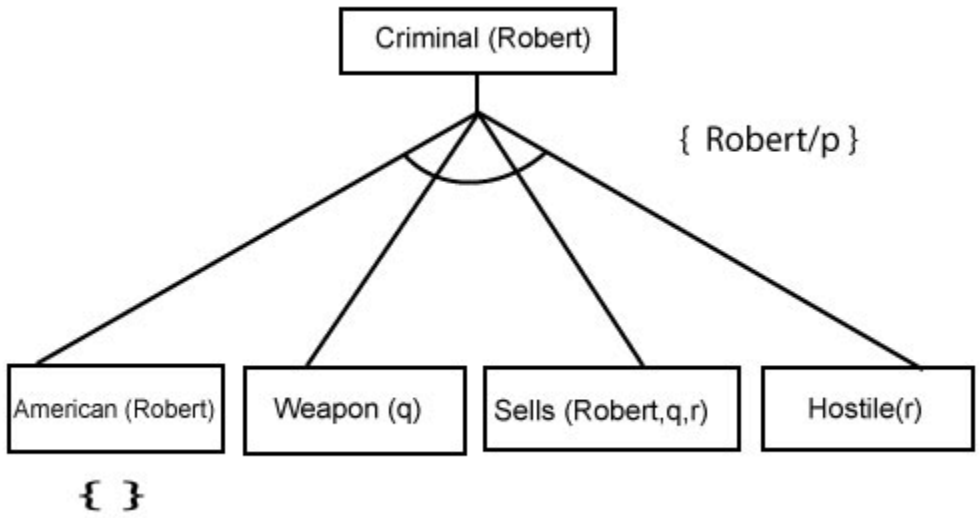
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.



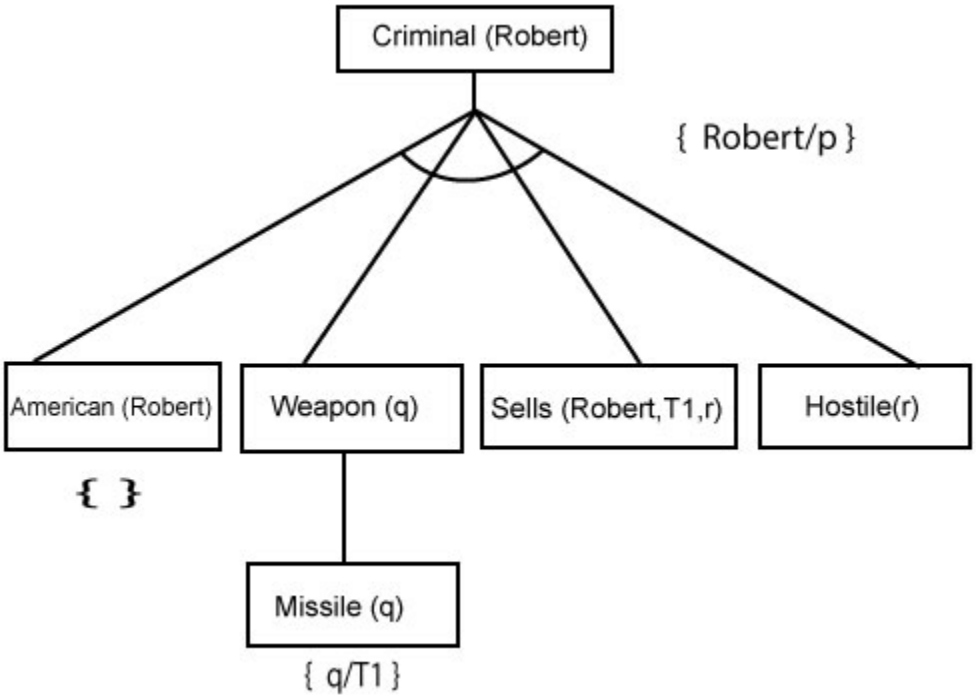
Step-2:

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

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

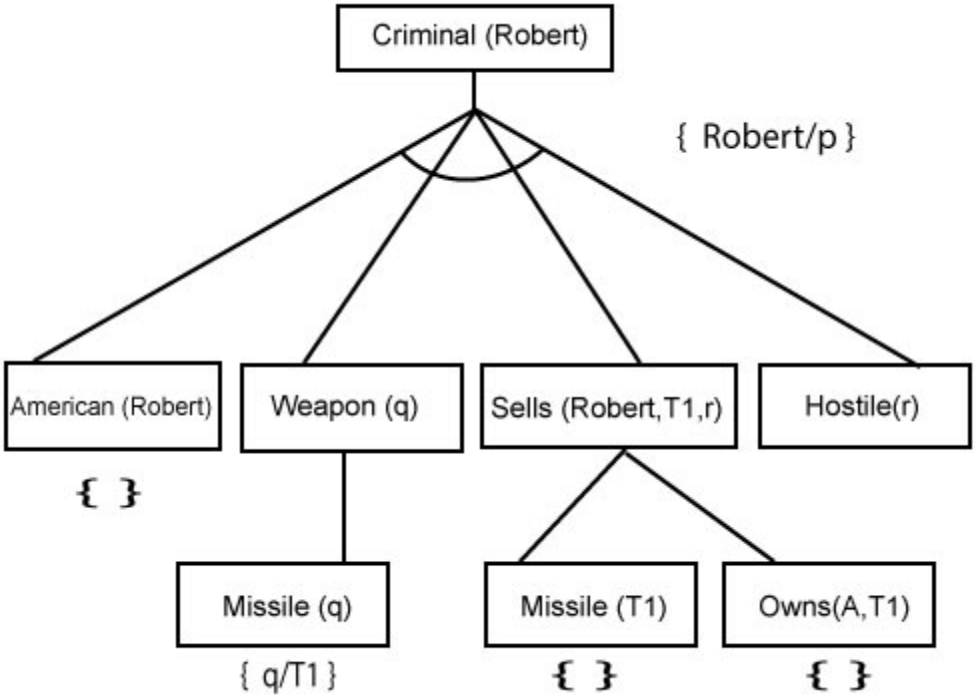


Step-3:t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



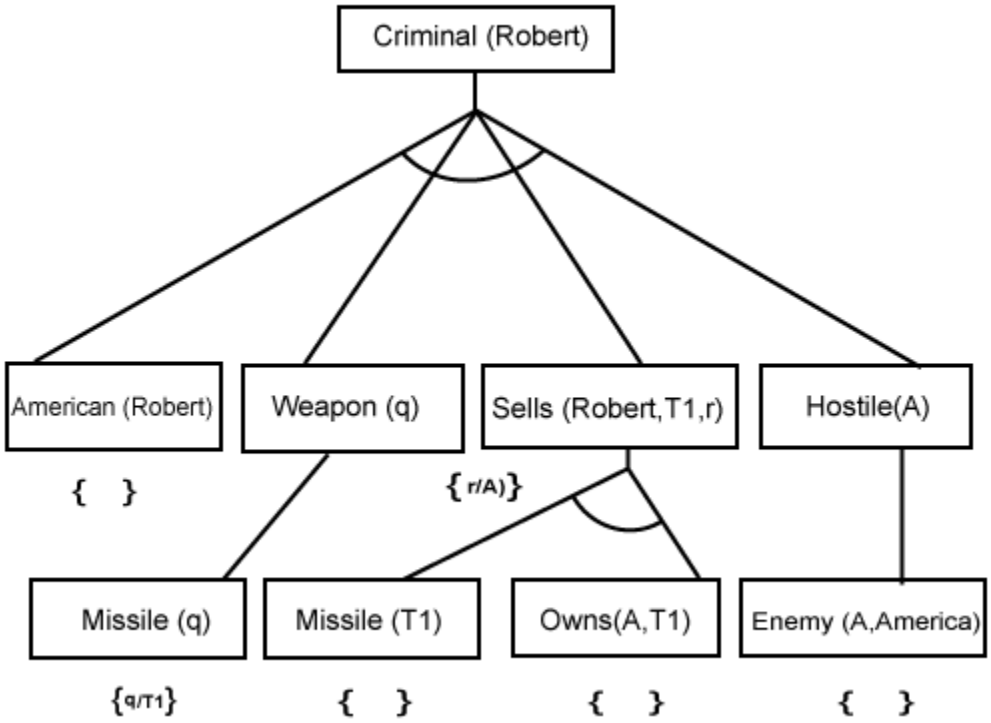
Step-4:

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



Step-5:

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



Difference between backward chaining and forward chaining

Following is the difference between the forward chaining and backward chaining:

- Forward chaining as the name suggests, start from the known facts and move forward by applying inference rules to extract more data, and it continues until it reaches to the goal, whereas backward chaining starts from the goal, move backward by using inference rules to determine the facts that satisfy the goal.
- Forward chaining is called a **data-driven** inference technique, whereas backward chaining is called a **goal-driven** inference technique.
- Forward chaining is known as the **down-up** approach, whereas backward chaining is known as a **top-down** approach.
- Forward chaining uses **breadth-first search** strategy, whereas backward chaining uses **depth-first search** strategy.
- Forward and backward chaining both applies **Modus ponens** inference rule.

- Forward chaining can be used for tasks such as **planning, design process monitoring, diagnosis, and classification**, whereas backward chaining can be used for **classification and diagnosis tasks**.
- Forward chaining can be like an exhaustive search, whereas backward chaining tries to avoid the unnecessary path of reasoning.
- In forward-chaining there can be various ASK questions from the knowledge base, whereas in backward chaining there can be fewer ASK questions.
- Forward chaining is slow as it checks for all the rules, whereas backward chaining is fast as it checks few required rules only.

. No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugg
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.

8.	It operates in the forward direction.	It operates in the backward direction.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.

Reasoning in Artificial intelligence

In previous topics, we have learned various ways of knowledge representation in artificial intelligence. Now we will learn the various ways to reason on this knowledge using different logical schemes.

Reasoning:

The reasoning is the mental process of deriving logical conclusion and making predictions from available knowledge, facts, and beliefs. Or we can say, "**Reasoning is a way to infer facts from existing data**." It is a general process of thinking rationally, to find valid conclusions.

In artificial intelligence, the reasoning is essential so that the machine can also think rationally as a human brain, and can perform like a human.

Types of Reasoning

In artificial intelligence, reasoning can be divided into the following categories:

Backward Skip 10sPlay VideoForward Skip 10s

- Deductive reasoning
- Inductive reasoning
- Abductive reasoning
- Common Sense Reasoning
- Monotonic Reasoning
- Non-monotonic Reasoning

Note: Inductive and deductive reasoning are the forms of propositional logic.

1. Deductive reasoning:

Deductive reasoning is deducing new information from logically related known information. It is the form of valid reasoning, which means the argument's conclusion must be true when the premises are true.

Deductive reasoning is a type of propositional logic in AI, and it requires various rules and facts. It is sometimes referred to as top-down reasoning, and contradictory to inductive reasoning.

In deductive reasoning, the truth of the premises guarantees the truth of the conclusion.

Deductive reasoning mostly starts from the general premises to the specific conclusion, which can be explained as below example.

Example:

Premise-1: All the human eats veggies

Premise-2: Suresh is human.

Conclusion: Suresh eats veggies.

The general process of deductive reasoning is given below:

2. Inductive Reasoning:

Inductive reasoning is a form of reasoning to arrive at a conclusion using limited sets of facts by the process of generalization. It starts with the series of specific facts or data and reaches to a general statement or conclusion.

Inductive reasoning is a type of propositional logic, which is also known as cause-effect reasoning or bottom-up reasoning.

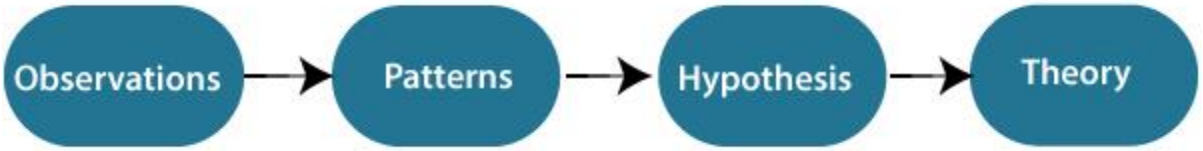
In inductive reasoning, we use historical data or various premises to generate a generic rule, for which premises support the conclusion.

In inductive reasoning, premises provide probable supports to the conclusion, so the truth of premises does not guarantee the truth of the conclusion.

Example:

Premise: All of the pigeons we have seen in the zoo are white.

Conclusion: Therefore, we can expect all the pigeons to be white.



3. Abductive reasoning:

Abductive reasoning is a form of logical reasoning which starts with single or multiple observations then seeks to find the most likely explanation or conclusion for the observation.

Abductive reasoning is an extension of deductive reasoning, but in abductive reasoning, the premises do not guarantee the conclusion.

Example:

Implication: Cricket ground is wet if it is raining

Axiom: Cricket ground is wet.

Conclusion It is raining.

4. Common Sense Reasoning

Common sense reasoning is an informal form of reasoning, which can be gained through experiences.

Common Sense reasoning simulates the human ability to make presumptions about events which occurs on every day.

It relies on good judgment rather than exact logic and operates on **heuristic knowledge** and **heuristic rules**.

Example:

1. **One person can be at one place at a time.**
2. **If I put my hand in a fire, then it will burn.**

The above two statements are the examples of common sense reasoning which a human mind can easily understand and assume.

5. Monotonic Reasoning:

In monotonic reasoning, once the conclusion is taken, then it will remain the same even if we add some other information to existing information in our knowledge base. In monotonic reasoning, adding knowledge does not decrease the set of prepositions that can be derived.

To solve monotonic problems, we can derive the valid conclusion from the available facts only, and it will not be affected by new facts.

Monotonic reasoning is not useful for the real-time systems, as in real time, facts get changed, so we cannot use monotonic reasoning.

Monotonic reasoning is used in conventional reasoning systems, and a logic-based system is monotonic.

Any theorem proving is an example of monotonic reasoning.

Example:

- **Earth revolves around the Sun.**

It is a true fact, and it cannot be changed even if we add another sentence in knowledge base like, "The moon revolves around the earth" Or "Earth is not round," etc.

Advantages of Monotonic Reasoning:

- In monotonic reasoning, each old proof will always remain valid.
- If we deduce some facts from available facts, then it will remain valid for always.

Disadvantages of Monotonic Reasoning:

- We cannot represent the real world scenarios using Monotonic reasoning.
- Hypothesis knowledge cannot be expressed with monotonic reasoning, which means facts should be true.
- Since we can only derive conclusions from the old proofs, so new knowledge from the real world cannot be added.

6. Non-monotonic Reasoning

In Non-monotonic reasoning, some conclusions may be invalidated if we add some more information to our knowledge base.

Logic will be said as non-monotonic if some conclusions can be invalidated by adding more knowledge into our knowledge base.

Non-monotonic reasoning deals with incomplete and uncertain models.

"Human perceptions for various things in daily life, "is a general example of non-monotonic reasoning.

Example: Let suppose the knowledge base contains the following knowledge:

- **Birds can fly**
- **Penguins cannot fly**
- **Pitty is a bird**

So from the above sentences, we can conclude that **Pitty can fly**.

However, if we add one another sentence into knowledge base "**Pitty is a penguin**", which concludes "**Pitty cannot fly**", so it invalidates the above conclusion.

Advantages of Non-monotonic reasoning:

- For real-world systems such as Robot navigation, we can use non-monotonic reasoning.
- In Non-monotonic reasoning, we can choose probabilistic facts or can make assumptions.

Disadvantages of Non-monotonic Reasoning:

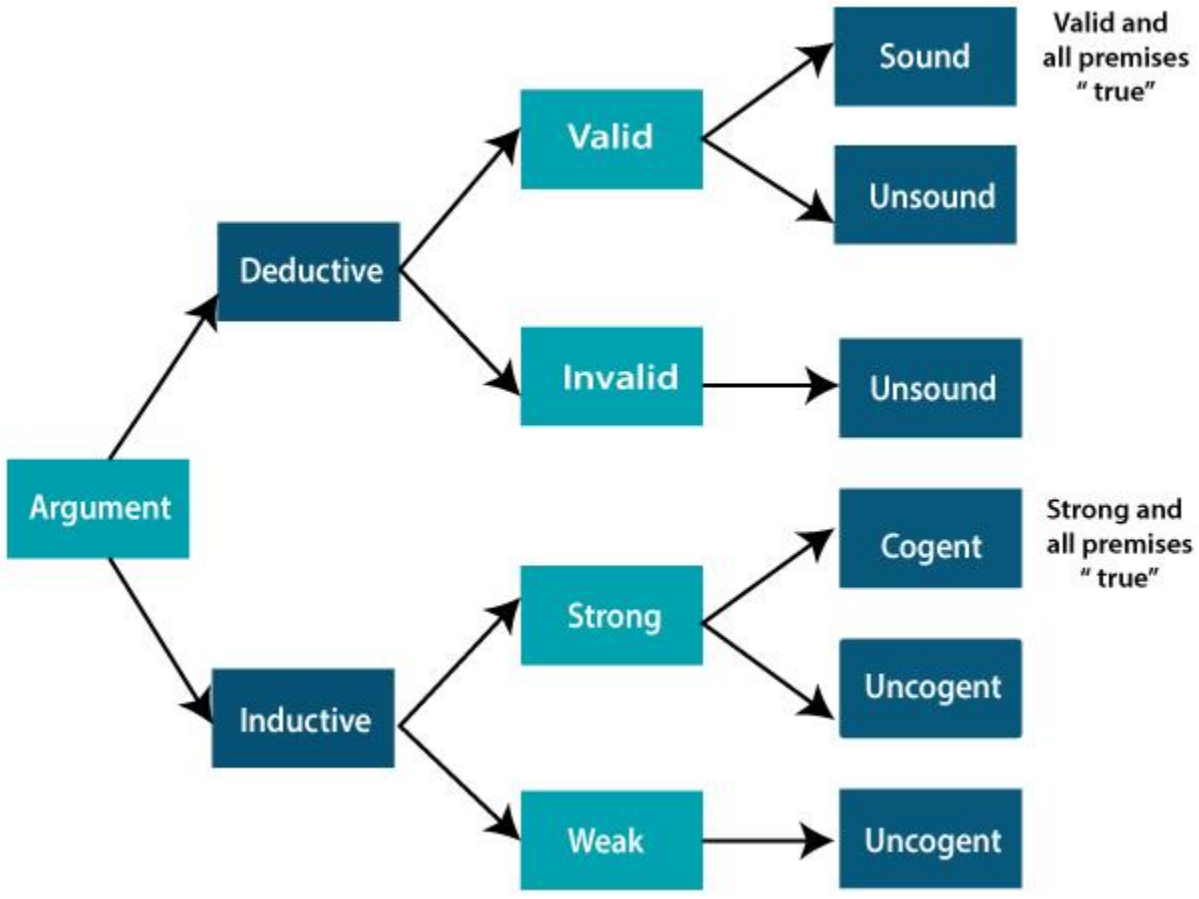
- In non-monotonic reasoning, the old facts may be invalidated by adding new sentences.
- It cannot be used for theorem **proving**.

Difference between Inductive and Deductive reasoning

Reasoning in artificial intelligence has two important forms, Inductive reasoning, and Deductive reasoning. Both reasoning forms have premises and conclusions, but both reasoning are contradictory to each other. Following is a list for comparison between inductive and deductive reasoning:

- Deductive reasoning uses available facts, information, or knowledge to deduce a valid conclusion, whereas inductive reasoning involves making a generalization from specific facts, and observations.
- Deductive reasoning uses a top-down approach, whereas inductive reasoning uses a bottom-up approach.
- Deductive reasoning moves from generalized statement to a valid conclusion, whereas Inductive reasoning moves from specific observation to a generalization.
- In deductive reasoning, the conclusions are certain, whereas, in Inductive reasoning, the conclusions are probabilistic.
- Deductive arguments can be valid or invalid, which means if premises are true, the conclusion must be true, whereas inductive argument can be strong or weak, which means conclusion may be false even if premises are true.

The differences between inductive and deductive can be explained using the below diagram on the basis of arguments:



Comparison Chart:

Basis for comparison	Deductive Reasoning	Inductive Reasoning
Definition	Deductive reasoning is the form of valid reasoning, to deduce new information or conclusion from known related facts and information.	Inductive reasoning arrives at a conclusion by the process of generalization using specific facts or data.
Approach	Deductive reasoning follows a top-down approach.	Inductive reasoning follows a bottom-up approach.
Starts from	Deductive reasoning starts from Premises.	Inductive reasoning starts from the Conclusion.
Validity	In deductive reasoning conclusion must be true if the premises are true.	In inductive reasoning, the truth of premises does not guarantee the truth of conclusions.
Usage	Use of deductive reasoning is difficult, as we need facts which must be true.	Use of inductive reasoning is fast and easy, as we need evidence instead of true facts. We often use it in our daily life.
Process	Theory→ hypothesis→ patterns→confirmation.	Observations- →patterns→hypothesis→Theory.
Argument	In deductive reasoning, arguments may be valid or invalid.	In inductive reasoning, arguments may be weak or strong.
Structure	Deductive reasoning reaches from general facts to specific facts.	Inductive reasoning reaches from specific facts to general facts.

Probabilistic reasoning in Artificial intelligence

Uncertainty:

Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write A→B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.

So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

Causes of uncertainty:

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

1. Information occurred from unreliable sources.
2. Experimental Errors
3. Equipment fault
4. Temperature variation
5. Climate change.

Probabilistic reasoning:

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.

In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI:

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

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

- **Bayes' rule**
- **Bayesian Statistics**

> Note: We will learn the above two rules in later chapters.

As probabilistic reasoning uses probability and related terms, so before understanding probabilistic reasoning, let's understand some common terms: