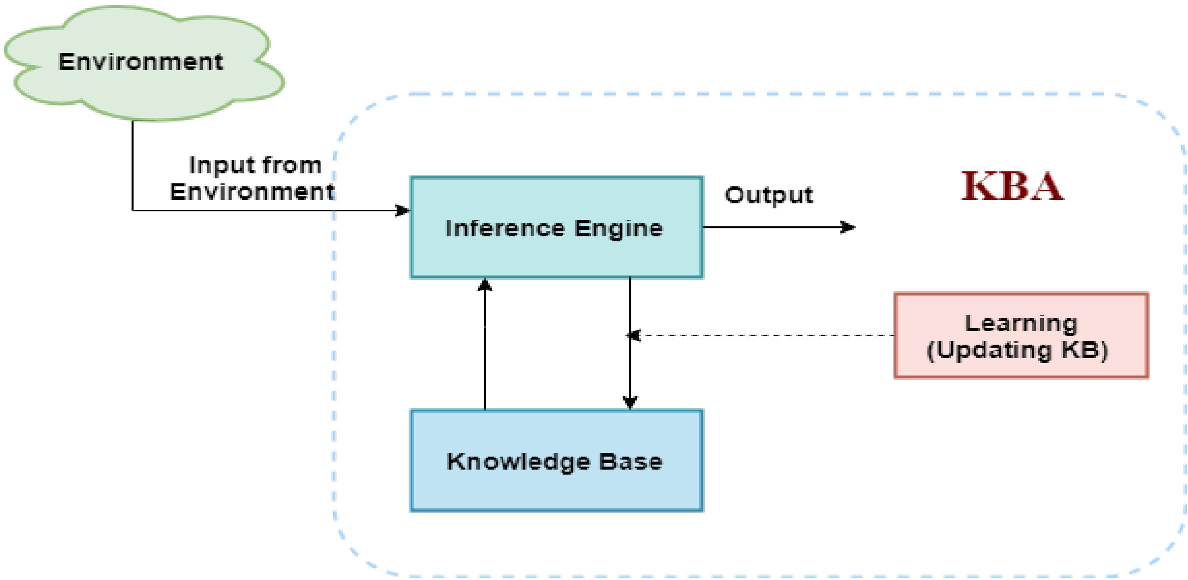
Knowledge-Based Agent in Artificial intelligence

- 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**.

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

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

The architecture of knowledge-based agent:



The above diagram is representing a generalized architecture for a knowledge-based agent. The knowledge-based agent (KBA) take input from the environment by perceiving the environment. The input is taken by the inference engine of the agent and which also communicate with KB to decide as per the knowledge store in KB. The learning element of KBA regularly updates the KB by learning new knowledge.

Knowledge base: Knowledge-base is a central component of a knowledge-based agent, it is also known as KB. It is a collection of sentences (here 'sentence' is a technical term and it is not identical to sentence in English). These sentences are expressed in a language which is called a knowledge representation language. The Knowledge-base of KBA stores fact about the world

Why use a knowledge base?

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

Inference system

Inference means deriving new sentences from old. Inference system allows us to add a new sentence to the knowledge base. A sentence is a proposition about the world. Inference system applies logical rules to the KB to deduce new information.

Inference system generates new facts so that an agent can update the KB. An inference system works mainly in two rules which are given as:

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

Operations Performed by KBA

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

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

A generic knowledge-based agent:

Following is the structure outline of a generic knowledge-based agents program:

1. function KB-AGENT(percept):
2. persistent: KB, a knowledge base
3. t, a counter, initially 0, indicating time
4. TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
5. Action = ASK(KB, MAKE-ACTION-QUERY(t))
6. TELL(KB, MAKE-ACTION-SENTENCE(action, t))
7. t = t + 1
8. **return** action

The knowledge-based agent takes percept as input and returns an action as output. The agent maintains the knowledge base, KB, and it initially has some background knowledge of the real world. It also has a counter to indicate the time for the whole process, and this counter is initialized with zero.

Each time when the function is called, it performs its three operations:

- Firstly it TELLs the KB what it perceives.
- Secondly, it asks KB what action it should take
- Third agent program TELLS the KB that which action was chosen.

The MAKE-PERCEPT-SENTENCE generates a sentence as setting that the agent perceived the given percept at the given time.

The MAKE-ACTION-QUERY generates a sentence to ask which action should be done at the current time.

MAKE-ACTION-SENTENCE generates a sentence which asserts that the chosen action was executed.

Various levels of knowledge-based agent:

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

1. Knowledge level

Knowledge level is the first level of knowledge-based agent, and in this level, we need to specify what the agent knows, and what the agent goals are. With these specifications, we can fix its behavior. For example, suppose an automated taxi agent needs to go from a station A to station B, and he knows the way from A to B, so this comes at the knowledge level.

2. Logical level:

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

3. Implementation level:

This is the physical representation of logic and knowledge. At the implementation level agent perform actions as per logical and knowledge level. At this level, an automated taxi agent actually implement his knowledge and logic so that he can reach to the destination.

Approaches to designing a knowledge-based agent:

There are mainly two approaches to build a knowledge-based agent:

1. **1. Declarative approach:** We can create a knowledge-based agent by initializing with an empty knowledge base and telling the agent all the sentences with which we want to start with. This approach is called Declarative approach.
2. **2. Procedural approach:** In the procedural approach, we directly encode desired behavior as a program code. Which means we just need to write a program that already encodes the desired behavior or agent.

However, in the real world, a successful agent can be built by combining both declarative and procedural approaches, and declarative knowledge can often be compiled into more efficient procedural code.

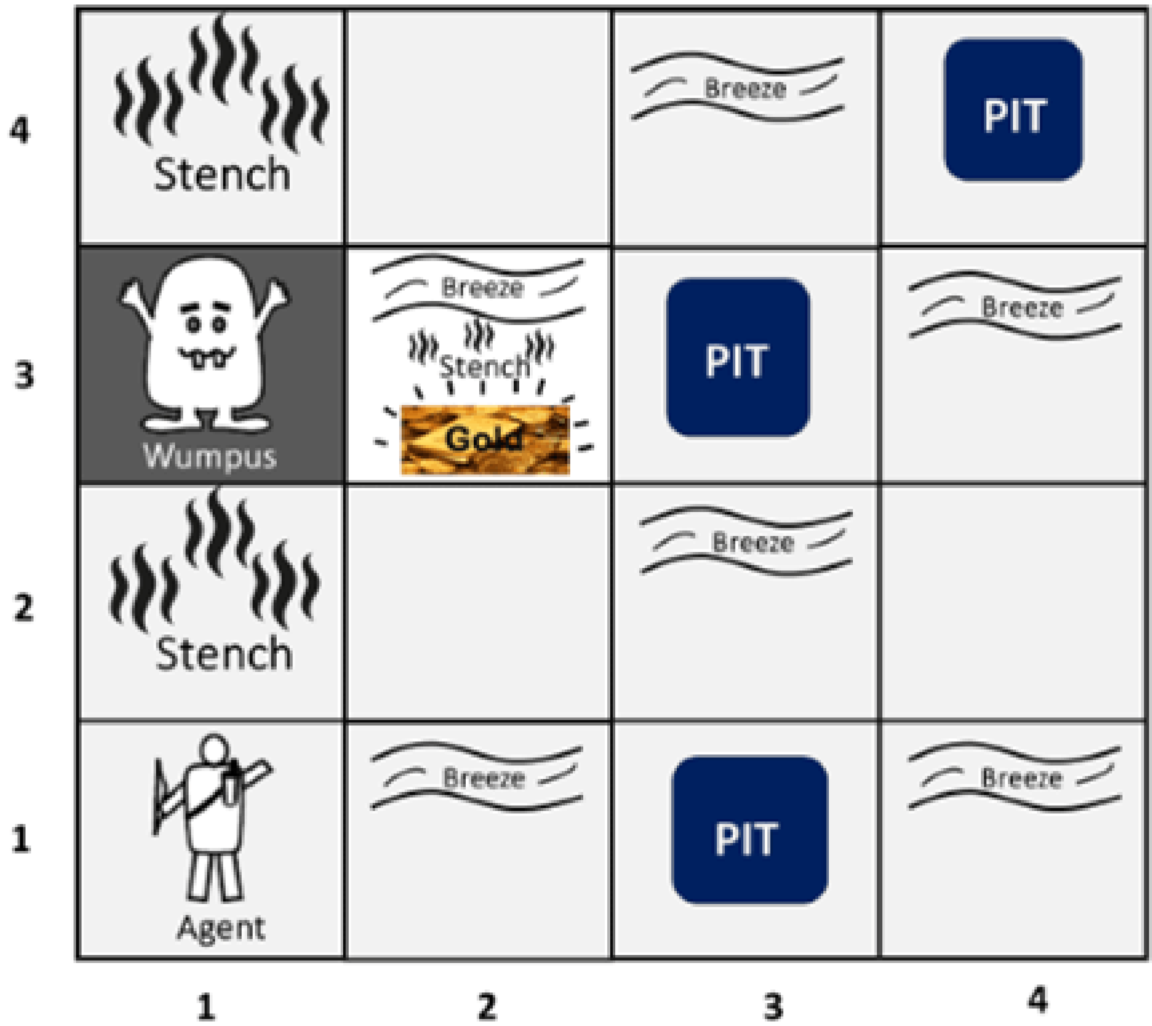
The Wumpus World in Artificial intelligence

Wumpus world:

The Wumpus world is a simple world example to illustrate the worth of a knowledge-based agent and to represent knowledge representation. It was inspired by a video game **Hunt the Wumpus** by Gregory Yob in 1973.

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

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



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

a. The rooms adjacent to the Wumpus room are smelly, so that it would have some stench.

b. The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.

c. There will be glitter in the room if and only if the room has gold.

d. The Wumpus can be killed by the agent if the agent is facing to it, and Wumpus will emit a horrible scream which can be heard anywhere in the cave.

PEAS description of Wumpus world:

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

Performance measure:

- +1000 reward points if the agent comes out of the cave with the gold.
- -1000 points penalty for being eaten by the Wumpus or falling into the pit.
- -1 for each action, and -10 for using an arrow.
- The game ends if either agent dies or came out of the cave.

Environment:

- A 4*4 grid of rooms.
- The agent initially in room square [1, 1], facing toward the right.
- Location of Wumpus and gold are chosen randomly except the first square [1,1].
- Each square of the cave can be a pit with probability 0.2 except the first square.

Actuators:

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

Sensors:

- The agent will perceive the **stench** if he is in the room adjacent to the Wumpus. (Not diagonally).

- The agent will perceive **breeze** if he is in the room directly adjacent to the Pit.
- The agent will perceive the **glitter** in the room where the gold is present.
- The agent will perceive the **bump** if he walks into a wall.
- When the Wumpus is shot, it emits a horrible **scream** which can be perceived anywhere in the cave.
- These percepts can be represented as five element list, in which we will have different indicators for each sensor.
- Example if agent perceives stench, breeze, but no glitter, no bump, and no scream then it can be represented as:
  [Stench, Breeze, None, None, None].

The Wumpus world Properties:

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

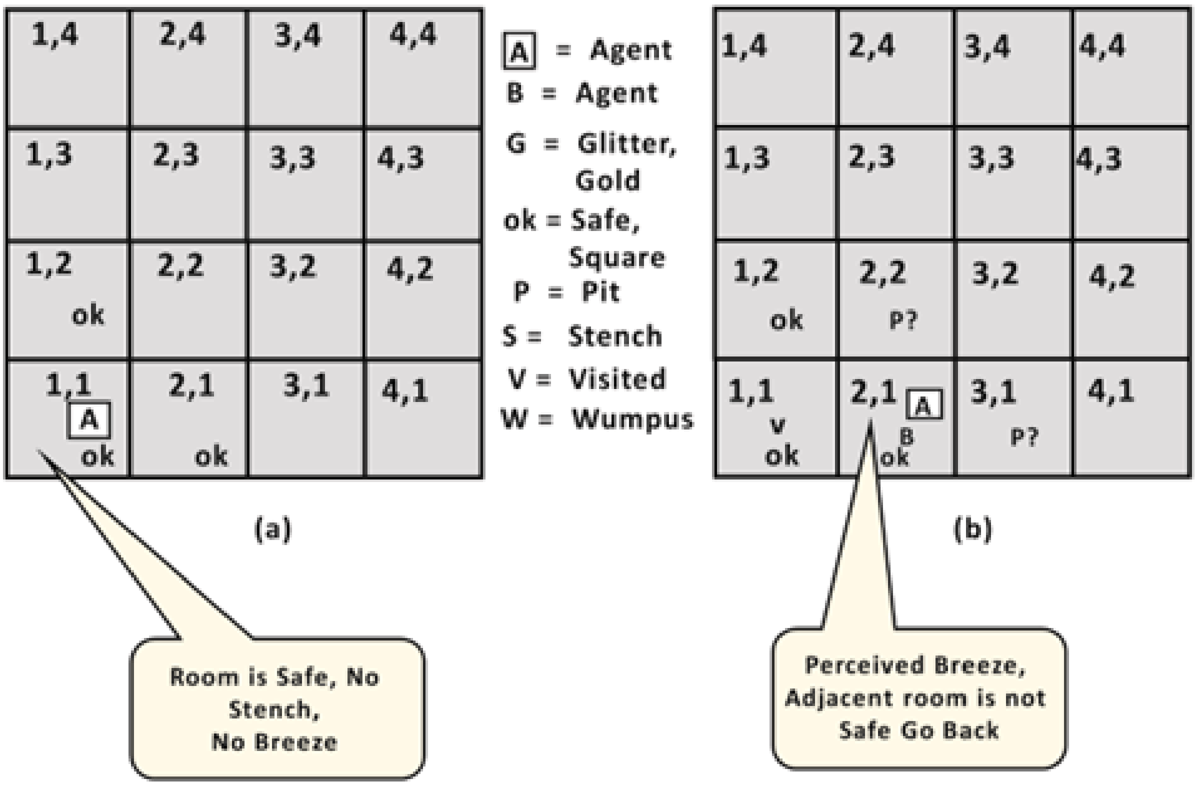
Exploring the Wumpus world:

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

Agent's First step:

Initially, the agent is in the first room or on the square [1,1], and we already know that this room is safe for the agent, so to represent on the below diagram (a) that room is safe we will add symbol OK. Symbol A is used to represent agent, symbol B for the breeze, G for Glitter or gold, V for the visited room, P for pits, W for Wumpus.

At Room [1,1] agent does not feel any breeze or any Stench which means the adjacent squares are also OK.



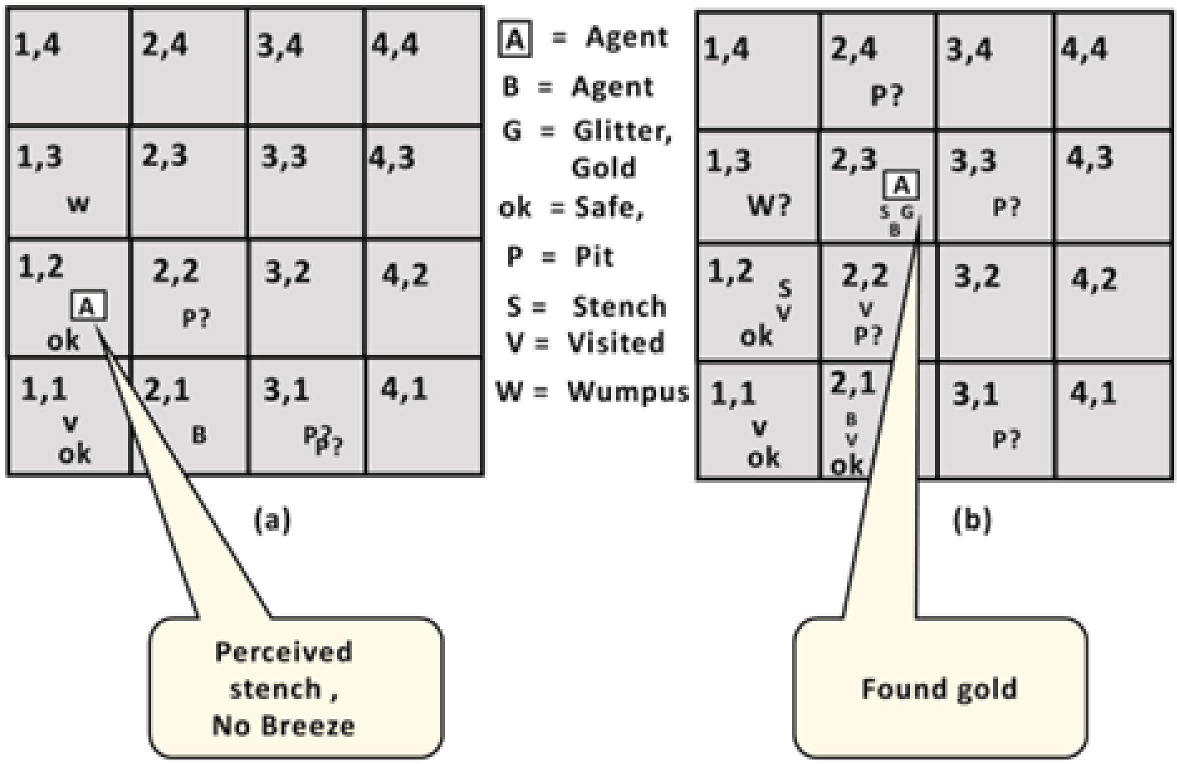
Agent's second Step:

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

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

Agent's third step:

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



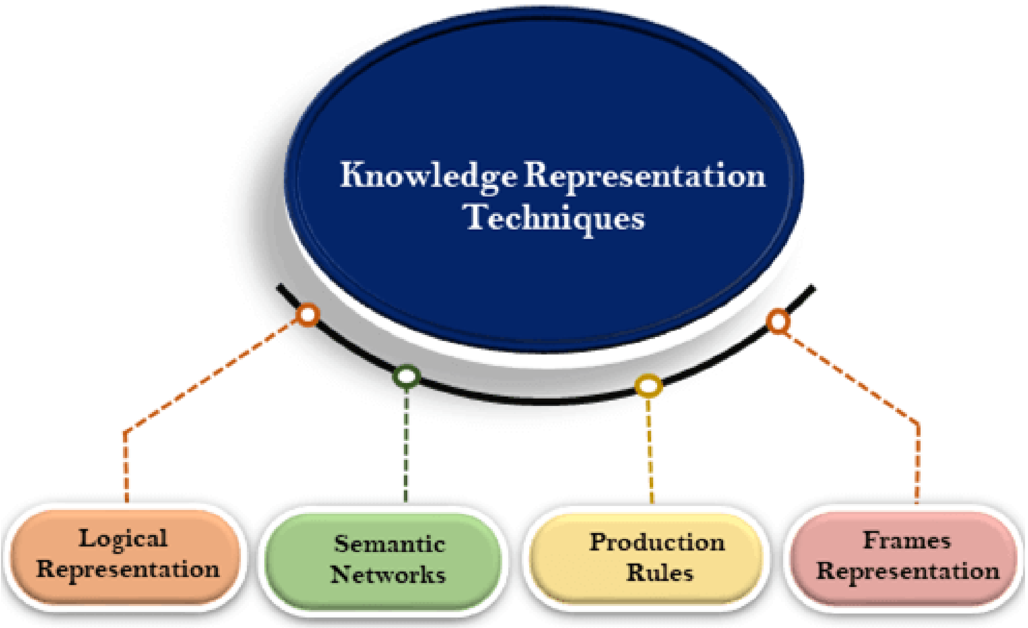
Agent's fourth step:

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

Techniques of knowledge representation

There are mainly four ways of knowledge representation which are given as follows:

1. Logical Representation
2. Semantic Network Representation
3. Frame Representation
4. Production Rules



1. Logical Representation

Logical representation is a language with some concrete rules which deals with propositions and has no ambiguity in representation. Logical representation means drawing a conclusion based on various conditions. This representation lays down some important communication rules. 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:

- Syntaxes are the rules which decide how we can construct legal sentences in the logic.
- It determines which symbol we can use in knowledge representation.
- How to write those symbols.

Semantics:

- Semantics are the rules by which we can interpret the sentence in the logic.
- Semantic also involves assigning a meaning to each sentence.

Logical representation can be categorised into mainly two logics:

a. Propositional Logics
b. Predicate logics

Advantages of logical representation:

1. Logical representation enables us to do logical reasoning.
2. Logical representation is the basis for the programming languages.

Disadvantages of logical Representation:

1. Logical representations have some restrictions and are challenging to work with.
2. Logical representation technique may not be very natural, and inference may not be so efficient.

2. Semantic Network Representation

Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects. Semantic networks can categorize the object in different forms and can also link those objects. Semantic networks are easy to understand and can be easily extended.

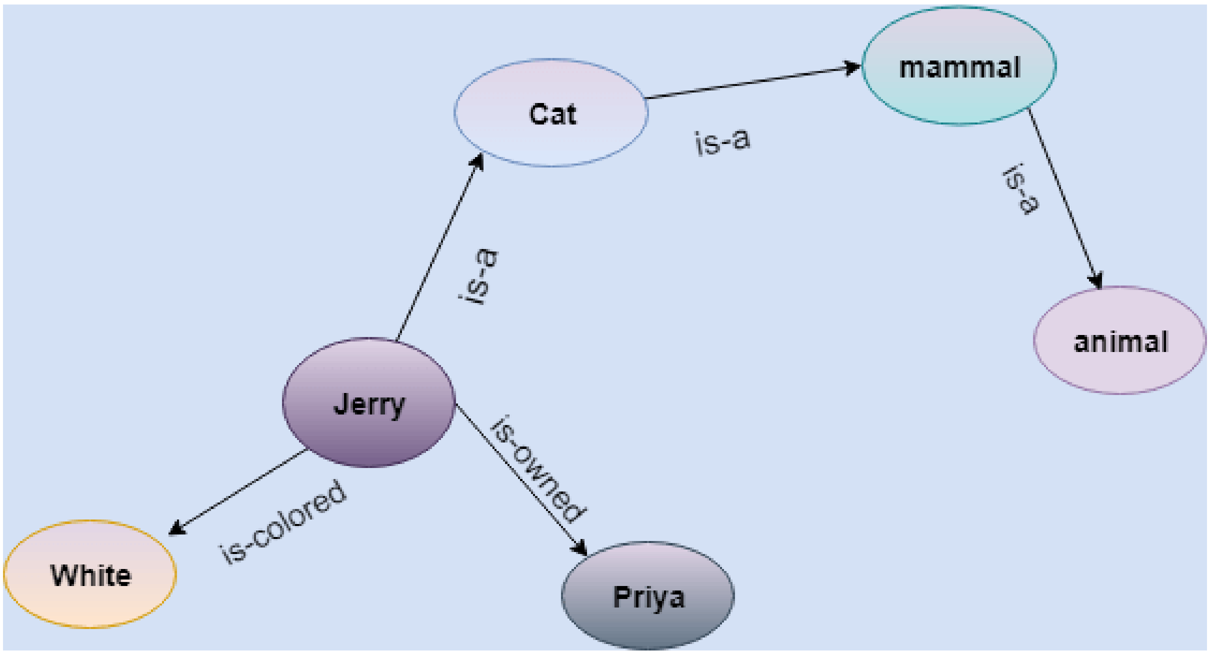
This representation consist of mainly two types of relations:

a. IS-A relation (Inheritance)
b. Kind-of-relation

Example: Following are some statements which we need to represent in the form of nodes and arcs.

Statements:

a. Jerry is a cat.
b. Jerry is a mammal
c. Jerry is owned by Priya.
d. Jerry is brown colored.
e. All Mammals are animal.



In the above diagram, we have represented the different type of knowledge in the form of nodes and arcs. Each object is connected with another object by some relation.

Drawbacks in Semantic representation:

1. Semantic networks take more computational time at runtime as we need to traverse the complete network tree to answer some questions. It might be possible in the worst case scenario that after traversing the entire tree, we find that the solution does not exist in this network.
2. Semantic networks try to model human-like memory (Which has 1015 neurons and links) to store the information, but in practice, it is not possible to build such a vast semantic network.
3. These types of representations are inadequate as they do not have any equivalent quantifier, e.g., for all, for some, none, etc.
4. Semantic networks do not have any standard definition for the link names.
5. These networks are not intelligent and depend on the creator of the system.

Advantages of Semantic network:

1. Semantic networks are a natural representation of knowledge.
2. Semantic networks convey meaning in a transparent manner.
3. These networks are simple and easily understandable.

3. Frame Representation

A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world. Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations. It consists of a collection of slots and slot values. These slots may be of any type and sizes. Slots have names and values which are called facets.

Facets: The various aspects of a slot is known as **Facets**. Facets are features of frames which enable us to put constraints on the frames. Example: IF-NEEDED facts are called when data of any particular slot is needed. A frame may consist of any number of slots, and a slot may include any number of facets and facets may have any number of values. A frame is also known as **slot-filter knowledge representation** in artificial intelligence.

Frames are derived from semantic networks and later evolved into our modern-day classes and objects. A single frame is not much useful. Frames system consist of a collection of frames which are connected. In the frame, knowledge about an object or event can be stored together in the knowledge base. The frame is a type of technology which is widely used in various applications including Natural language processing and machine visions

Example: 1

Let's take an example of a frame for a book

Slots	Filters
Title	Artificial Intelligence
Genre	Computer Science
Author	Peter Norvig
Edition	Third Edition
Year	1996
Page	1152

Example 2:

Let's suppose we are taking an entity, Peter. Peter is an engineer as a profession, and his age is 25, he lives in city London, and the country is England. So following is the frame representation for this:

Slots	Filter
Name	Peter
Profession	Doctor
Age	25
Marital status	Single
Weight	78

Advantages of frame representation:

1. The frame knowledge representation makes the programming easier by grouping the related data.
2. The frame representation is comparably flexible and used by many applications in AI.
3. It is very easy to add slots for new attribute and relations.
4. It is easy to include default data and to search for missing values.
5. Frame representation is easy to understand and visualize.

Disadvantages of frame representation:

1. In frame system inference mechanism is not be easily processed.
2. Inference mechanism cannot be smoothly proceeded by frame representation.
3. Frame representation has a much generalized approach.

4. Production Rules

Production rules system consist of (**condition, action**) pairs which mean, "If condition then action". It has mainly three parts:

- The set of production rules
- Working Memory
- The recognize-act-cycle

In production rules agent checks for the condition and if the condition exists then production rule fires and corresponding action is carried out. The condition part of the rule determines which rule may be applied to a problem. And the action part carries out the associated problem-solving steps. This complete process is called a recognize-act cycle.

The working memory contains the description of the current state of problems-solving and rule can write knowledge to the working memory. This knowledge match and may fire other rules.

If there is a new situation (state) generates, then multiple production rules will be fired together, this is called conflict set. In this situation, the agent needs to select a rule from these sets, and it is called a conflict resolution.

Example:

- **IF (at bus stop AND bus arrives) THEN action (get into the bus)**
- **IF (on the bus AND paid AND empty seat) THEN action (sit down).**
- **IF (on bus AND unpaid) THEN action (pay charges).**
- **IF (bus arrives at destination) THEN action (get down from the bus).**

Advantages of Production rule:

1. The production rules are expressed in natural language.
2. The production rules are highly modular, so we can easily remove, add or modify an individual rule.

Disadvantages of Production rule:

1. Production rule system does not exhibit any learning capabilities, as it does not store the result of the problem for the future uses.
2. During the execution of the program, many rules may be active hence rule-based production systems are inefficient.

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
$\wedge$	AND	Conjunction	$A \wedge B$
$\vee$	OR	Disjunction	$A \vee B$
$\rightarrow$	Implies	Implication	$A \rightarrow B$
$\Leftrightarrow$	If and only if	Biconditional	$A \Leftrightarrow B$
$\neg$ or $\sim$	Not	Negation	$\neg A$ or $\neg B$

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	$\neg$ P
True	**False**
False	**True**

For Conjunction:

P	Q	$P \wedge Q$
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

For disjunction:

P	Q	$P \vee Q$.
True	True	**True**
False	True	**True**
True	False	**True**
False	False	**False**

For Implication:

P	Q	$P \rightarrow Q$
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

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.

Rules of Inference in Artificial intelligence

Inference:

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

Inference rules:

Inference rules are the templates for generating valid arguments. Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal.

In inference rules, the implication among all the connectives plays an important role. Following are some terminologies related to inference rules:

- **Implication:** It is one of the logical connectives which can be represented as $P \rightarrow Q$. It is a Boolean expression.
- **Converse:** The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as $Q \rightarrow P$.
- **Contrapositive:** The negation of converse is termed as contrapositive, and it can be represented as $\neg Q \rightarrow \neg P$.
- **Inverse:** The negation of implication is called inverse. It can be represented as $\neg P \rightarrow \neg Q$.

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

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

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

Types of Inference rules:

1. Modus Ponens:

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

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

Example:

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
Statement-2: "I am sleepy" ==> P
Conclusion: "I go to bed." ==> Q.
Hence, we can say that, if P→ Q is true and P is true then Q will be true.

Proof by Truth table:

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

2. Modus Tollens:

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

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

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
Statement-2: "I do not go to the bed."==> ~Q
Statement-3: Which infers that "**I am not sleepy**" => ~P

Proof by Truth table:

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

3. Hypothetical Syllogism:

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

Example:

Statement-1: If you have my home key then you can unlock my home. **P→Q**
Statement-2: If you can unlock my home then you can take my money. **Q→R**
Conclusion: If you have my home key then you can take my money. **P→R**

Proof by truth table:

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

4. Disjunctive Syllogism:

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

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

Example:

Statement-1: Today is Sunday or Monday. ==>PVQ
Statement-2: Today is not Sunday. ==> ¬P
Conclusion: Today is Monday. ==> Q

Proof by truth-table:

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

5. Addition:

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

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

Example:

Statement: I have a vanilla ice-cream. ==> P
Statement-2: I have Chocolate ice-cream.
Conclusion: I have vanilla or chocolate ice-cream. ==> (PVQ)

Proof by Truth-Table:

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

6. Simplification:

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

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

Proof by Truth-Table:

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

7. Resolution:

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

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

Proof by Truth-Table:

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

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:
  a. **Syntax**
  b. **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).

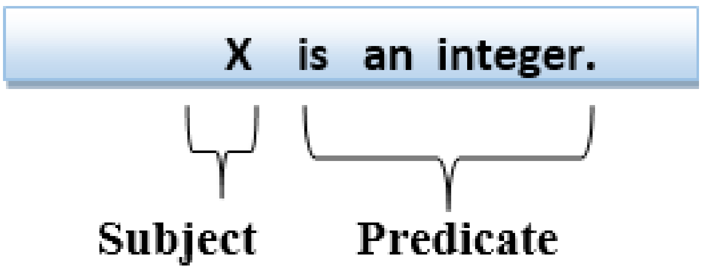
Complex Sentences:

- 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:
    a. **Universal Quantifier, (for all, everyone, everything)**

b. **Existential quantifier, (for some, at least one).**

Universal Quantifier:

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

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

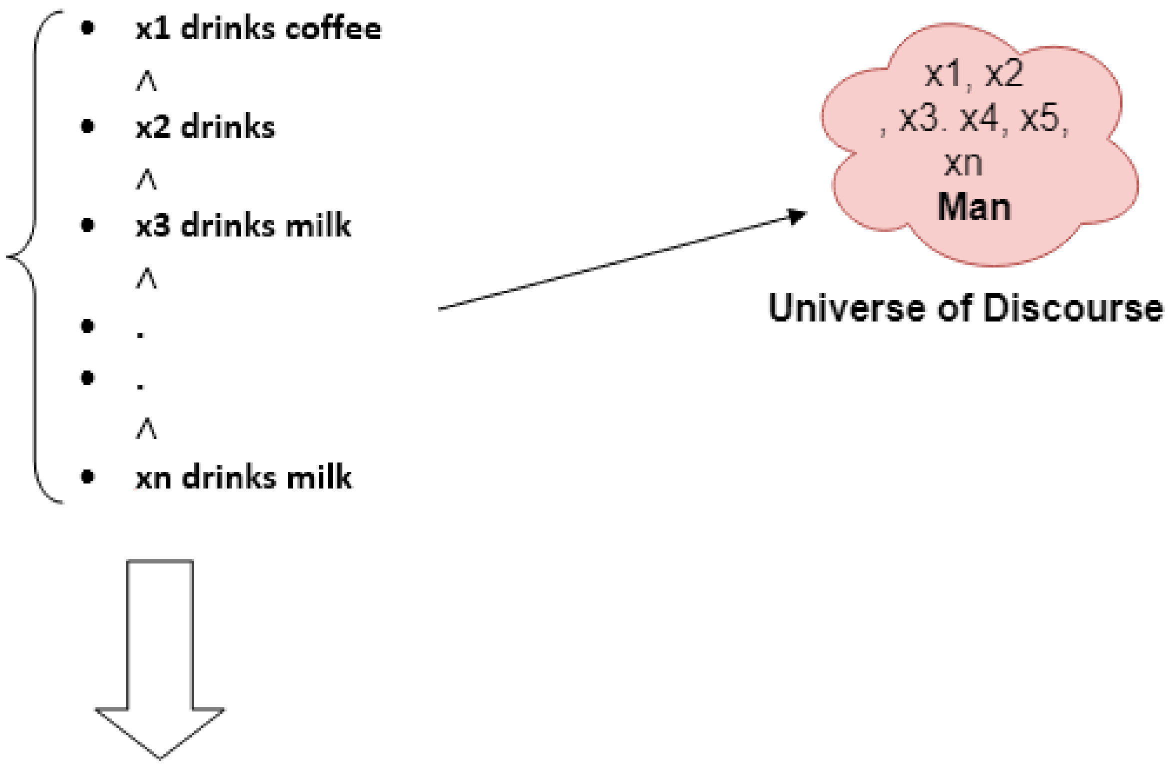
If x is a variable, then ∀x is read as:

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

Example:

All man drink coffee.

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



So in shorthand notation, we can write it as :

∀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.

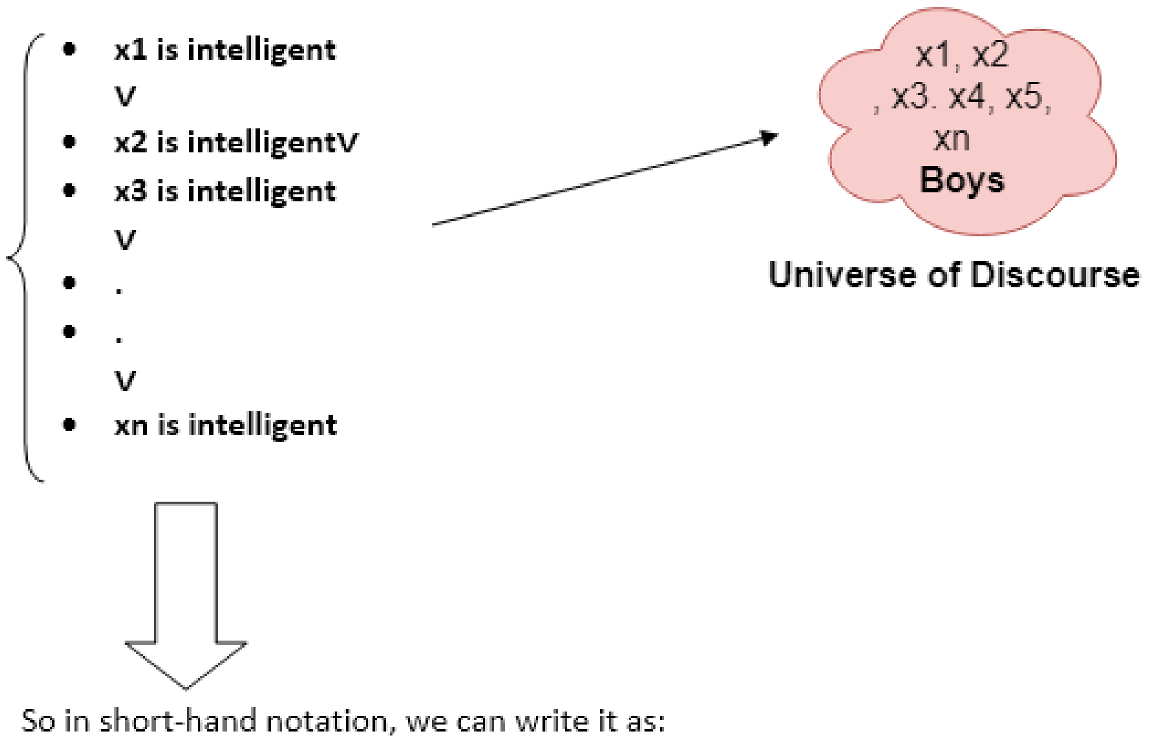
It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

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.

Inference in First-Order Logic

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

Substitution:

Substitution is a fundamental operation performed on terms and formulas. It occurs in all inference systems in first-order logic. The substitution is complex in the presence of quantifiers in FOL. If we write **F[a/x]**, so it refers to substitute a constant "**a**" in place of variable "**x**".

Equality:

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

Example: Brother (John) = Smith.

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

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

FOL inference rules for quantifier:

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

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

1. Universal Generalization:

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

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

2. Universal Instantiation

- Universal instantiation is also called as universal elimination or UI is a valid inference rule. It can be applied multiple times to add new sentences.
- The new KB is logically equivalent to the previous KB.
- As per UI, **we can infer any sentence obtained by substituting a ground term for the variable**.
- The UI rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from **∀ x P(x) for any object in the universe of discourse**.
- It can be represented as: $\frac{\forall x\, P(x)}{P(c)}$.

Example:1.

IF "Every person like ice-cream"=> $\forall x\ P(x)$ so we can infer that "John likes ice-cream" => $P(c)$

Example: 2.

Let's take a famous example,

"All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL:

$\forall x$ king(x) ∧ greedy (x) → Evil (x),

So from this information, we can infer any of the following statements using Universal Instantiation:

- **King(John) ∧ Greedy (John) → Evil (John),**
- **King(Richard) ∧ Greedy (Richard) → Evil (Richard),**
- **King(Father(John)) ∧ Greedy (Father(John)) → Evil (Father(John)),**

3. Existential Instantiation:

- Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.
- It can be applied only once to replace the existential sentence.
- The new KB is not logically equivalent to old KB, but it will be satisfiable if old KB was satisfiable.
- This rule states that one can infer P(c) from the formula given in the form of $\exists x\ P(x)$ for a new constant symbol c.
- The restriction with this rule is that c used in the rule must be a new term for which P(c) is true.
- It can be represented as: $\dfrac{\exists x\ P(x)}{P(c)}$

Example:

From the given sentence: **$\exists x$ Crown(x) ∧ OnHead(x, John),** So we can infer: **Crown(K) ∧ OnHead(K, John),** as long as K does not appear in the knowledge base.

- The above used K is a constant symbol, which is called **Skolem constant**.
- The Existential instantiation is a special case of **Skolemization process**.

4. Existential introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the property P.
- It can be represented as: $\dfrac{P(c)}{\exists x P(x)}$
- **Example: Let's say that,**
  "Priyanka got good marks in English."
  "Therefore, someone got good marks in English."

Generalized Modus Ponens Rule:

For the inference process in FOL, we have a single inference rule which is called Generalized Modus Ponens. It is lifted version of Modus ponens.

Generalized Modus Ponens can be summarized as, " P implies Q and P is asserted to be true, therefore Q must be True."

According to Modus Ponens, for atomic sentences **pi, pi', q**. Where there is a substitution θ such that SUBST **(θ, pi',) = SUBST(θ, pi)**, it can be represented as:

$$\frac{p1', p2', ..., pn', (p1 \land p2 \land ... \land pn \Rightarrow q)}{SUBST(\theta, q)}$$

Example:

We will use this rule for Kings are evil, so we will find some x such that x is king, and x is greedy so we can infer that x is evil.

1. Here let say, p1' is king(John)   p1 is king(x)
2. p2' is Greedy(y)   p2 is Greedy(x)
3. θ is {x/John, y/John}   q is evil(x)
4. SUBST(θ,q).

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_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}.

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 $p \wedge q \rightarrow k$.

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.

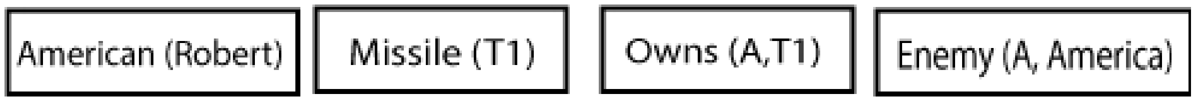
Facts Conversion into FOL:

- 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:

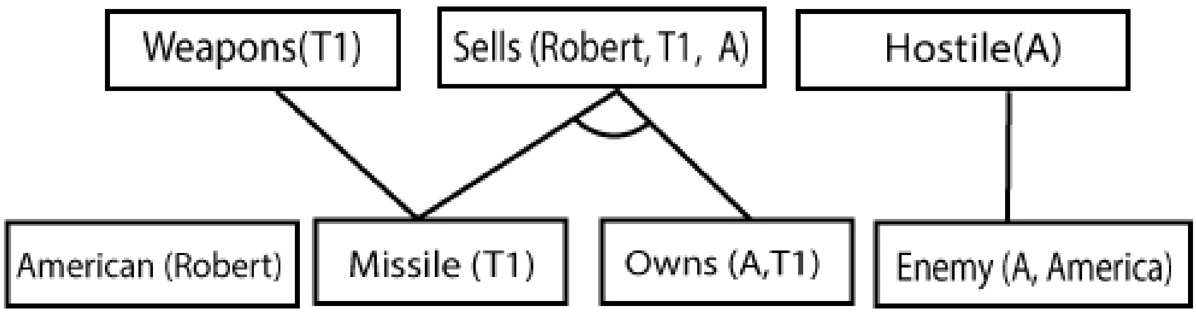
At the second step, we will see those facts which infer from available facts and with satisfied premises.

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

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

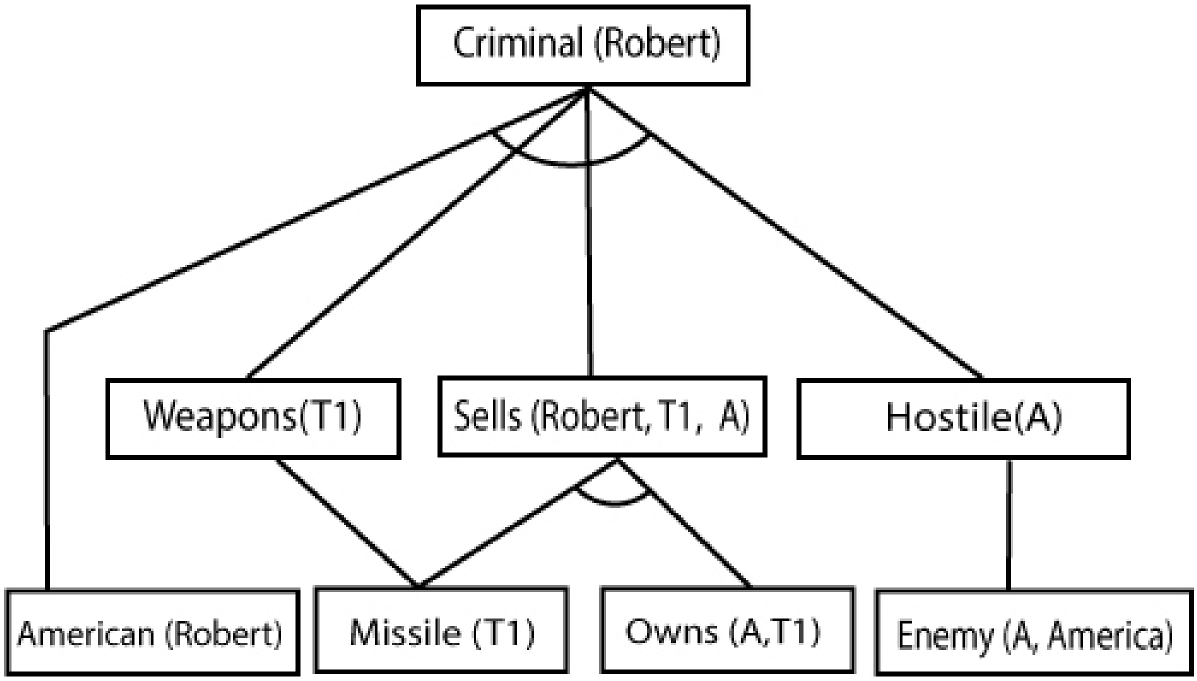
Rule-(4) satisfy with the substitution {p/T1}, **so Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

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.

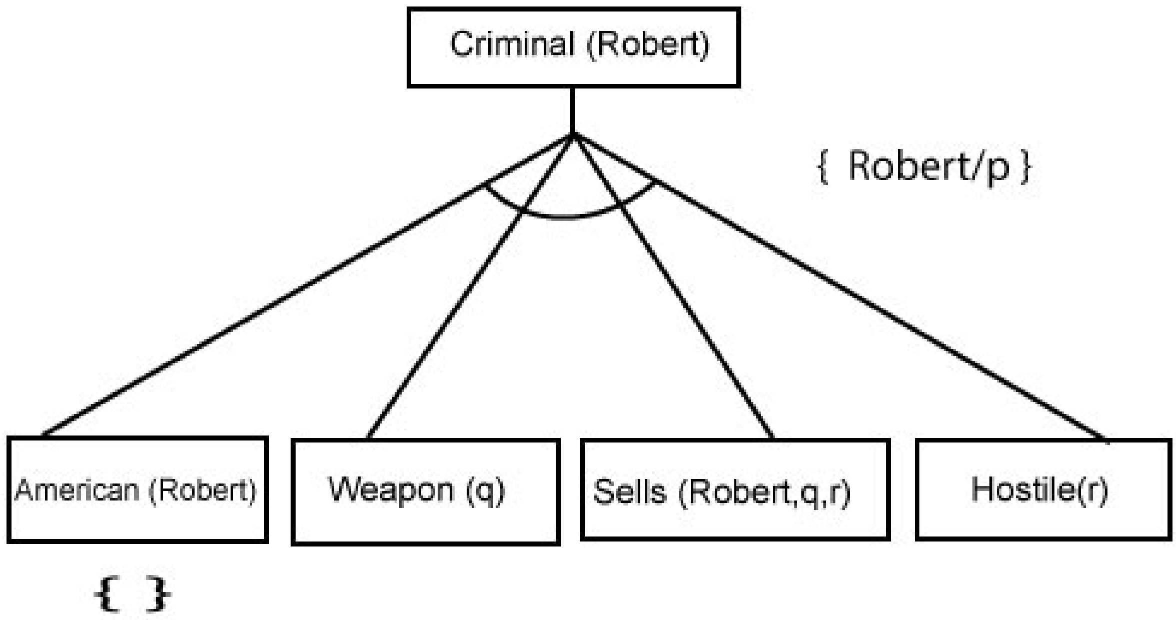
Step-1:

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.

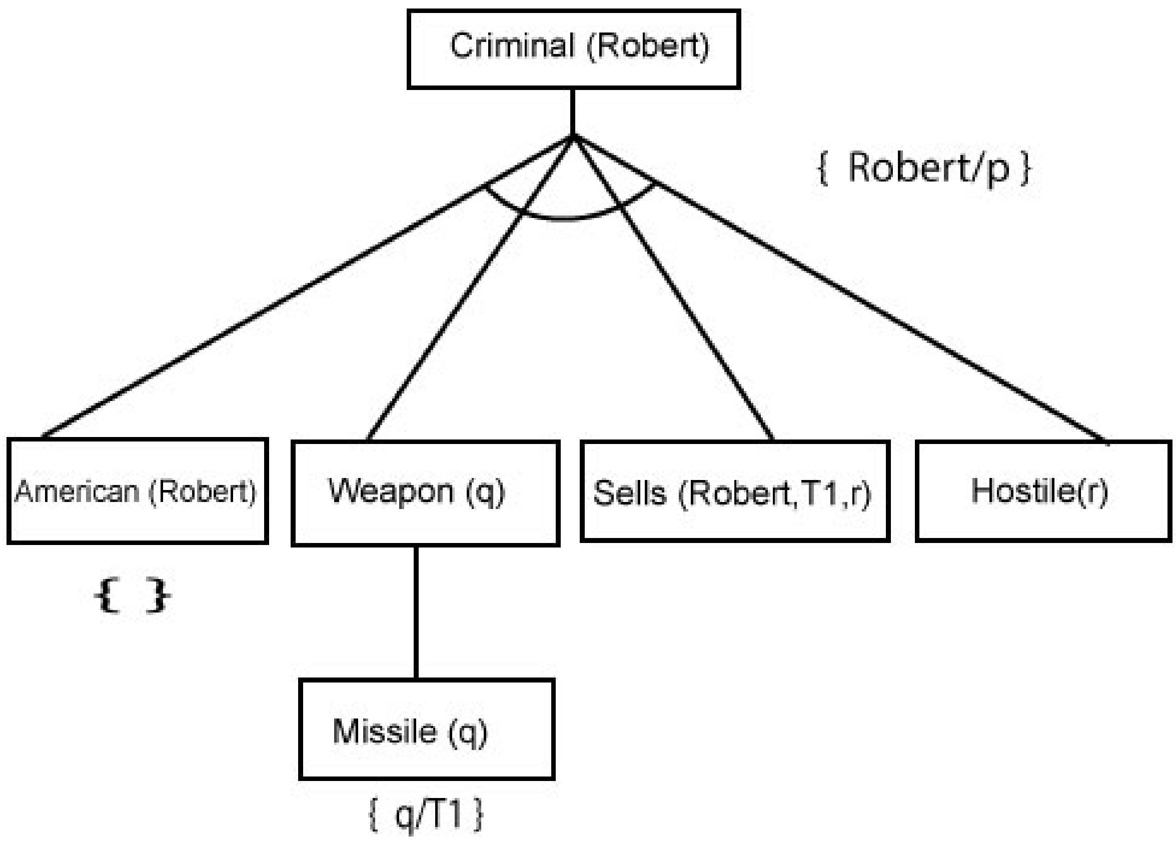
Step-2:

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.

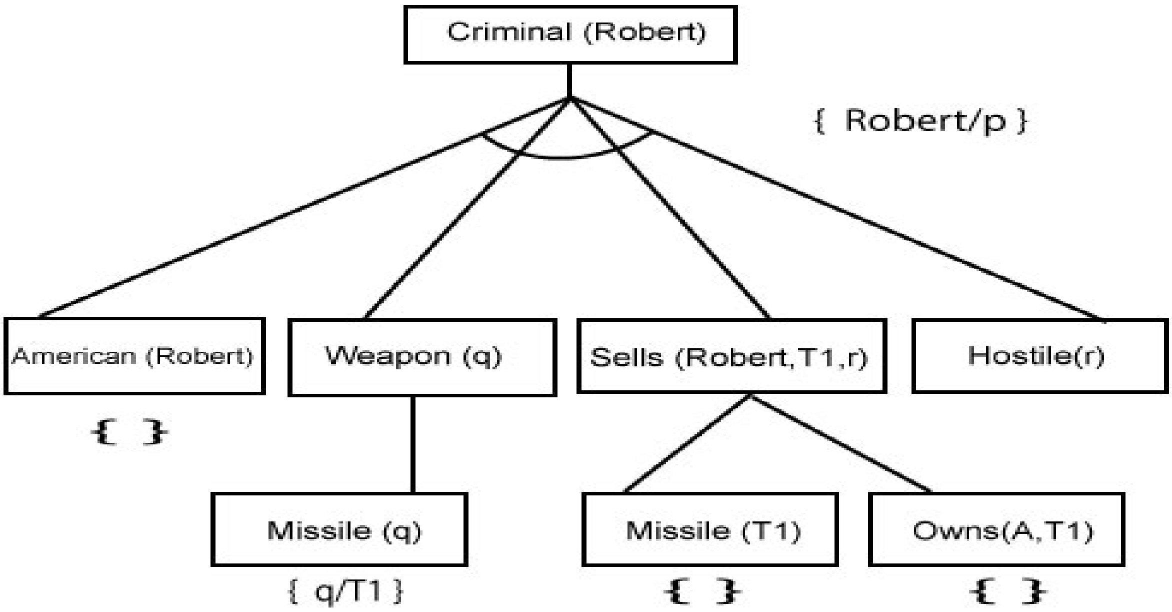


Step-3:t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



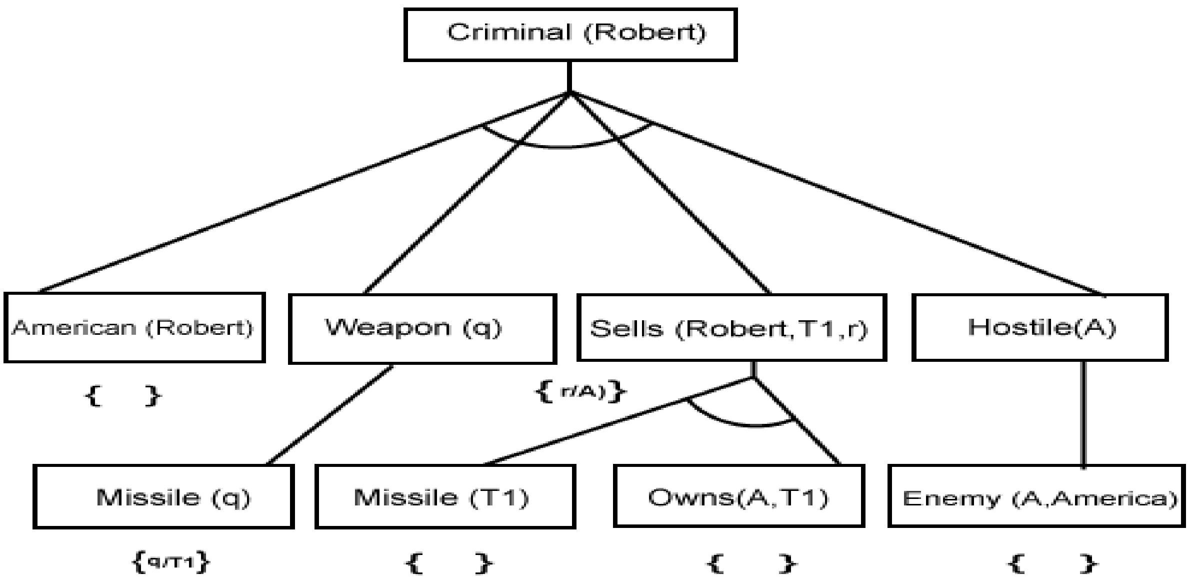
Step-4:

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.

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.

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

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. $\forall x\text{: food}(x) \rightarrow \text{likes}(\text{John}, x)$

b. $\text{food}(\text{Apple}) \land \text{food}(\text{vegetables})$

c. $\forall x \forall y\text{: eats}(x, y) \land \neg \text{killed}(x) \rightarrow \text{food}(y)$

d. $\text{eats}(\text{Anil}, \text{Peanuts}) \land \text{alive}(\text{Anil})$.

e. $\forall x : \text{eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$

f. $\forall x\text{: } \neg \text{killed}(x) \rightarrow \text{alive}(x)$ **added predicates.**

g. $\forall x\text{: alive}(x) \rightarrow \neg \text{killed}(x)$

h. $\text{likes}(\text{John}, \text{Peanuts})$

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**
  a. $\forall x \neg \text{food}(x) \lor \text{likes}(\text{John}, x)$
  b. $\text{food}(\text{Apple}) \land \text{food}(\text{vegetables})$
  c. $\forall x \forall y \neg [\text{eats}(x, y) \land \neg \text{killed}(x)] \lor \text{food}(y)$
  d. $\text{eats}(\text{Anil}, \text{Peanuts}) \land \text{alive}(\text{Anil})$
  e. $\forall x \neg \text{eats}(\text{Anil}, x) \lor \text{eats}(\text{Harry}, x)$
  f. $\forall x \neg [\neg \text{killed}(x)] \lor \text{alive}(x)$
  g. $\forall x \neg \text{alive}(x) \lor \neg \text{killed}(x)$

h. likes(John, Peanuts).

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

. $\forall x \neg$ food(x) V likes(John, x)

a. food(Apple) Λ food(vegetables)

b. $\forall x \forall y \neg$ eats(x, y) V killed(x) V food(y)

c. eats (Anil, Peanuts) Λ alive(Anil)

d. $\forall x \neg$ eats(Anil, x) V eats(Harry, x)

e. $\forall x$ ¬killed(x)] V alive(x)

f. $\forall x \neg$ alive(x) V ¬ killed(x)

g. likes(John, Peanuts).

- **Rename variables or standardize variables**

. $\forall x \neg$ food(x) V likes(John, x)

a. food(Apple) Λ food(vegetables)

b. $\forall y \forall z \neg$ eats(y, z) V killed(y) V food(z)

c. eats (Anil, Peanuts) Λ alive(Anil)

d. $\forall w \neg$ eats(Anil, w) V eats(Harry, w)

e. $\forall g$ ¬killed(g)] V alive(g)

f. $\forall k \neg$ alive(k) V ¬ killed(k)

g. 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.

. ¬ food(x) V likes(John, x)

a. food(Apple)

b. food(vegetables)

c. ¬ eats(y, z) V killed(y) V food(z)

d. eats (Anil, Peanuts)

e. alive(Anil)

f. ¬ eats(Anil, w) V eats(Harry, w)

g. killed(g) V alive(g)

h. ¬ alive(k) V ¬ killed(k)

i. likes(John, Peanuts).

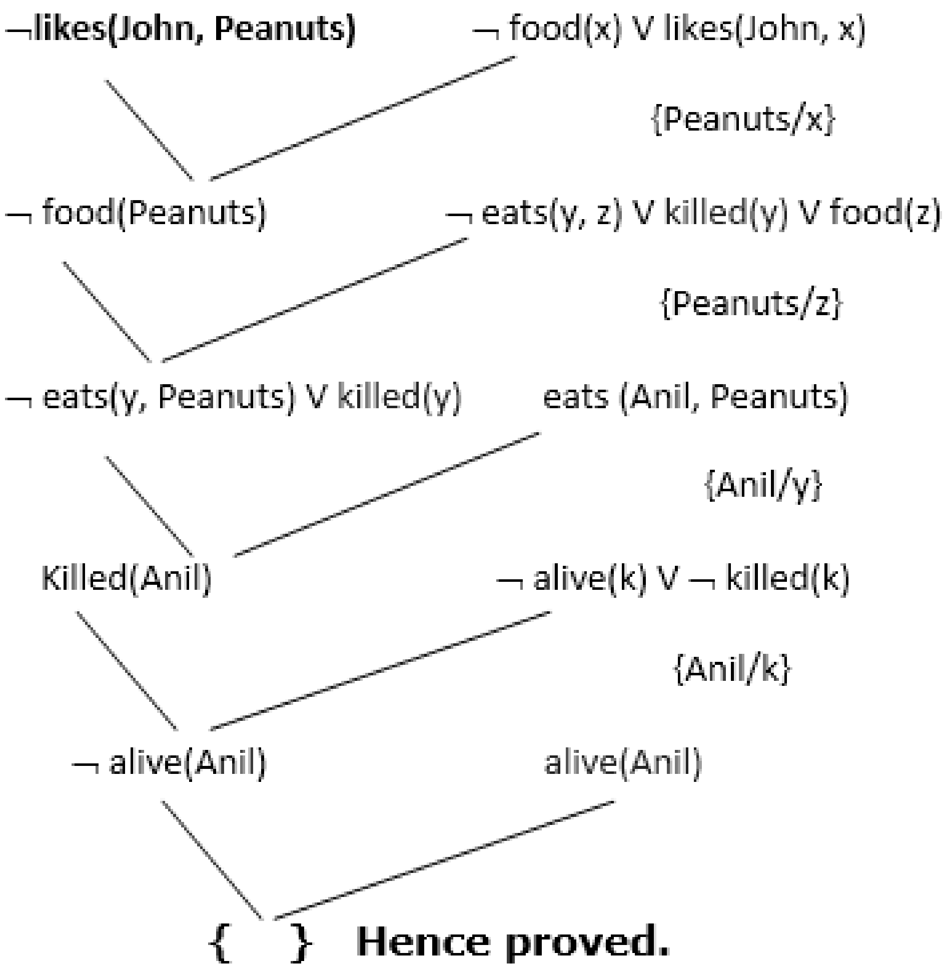
- **Distribute conjunction ∧ over disjunction ¬.**
  This step will not make any change in this problem.

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.