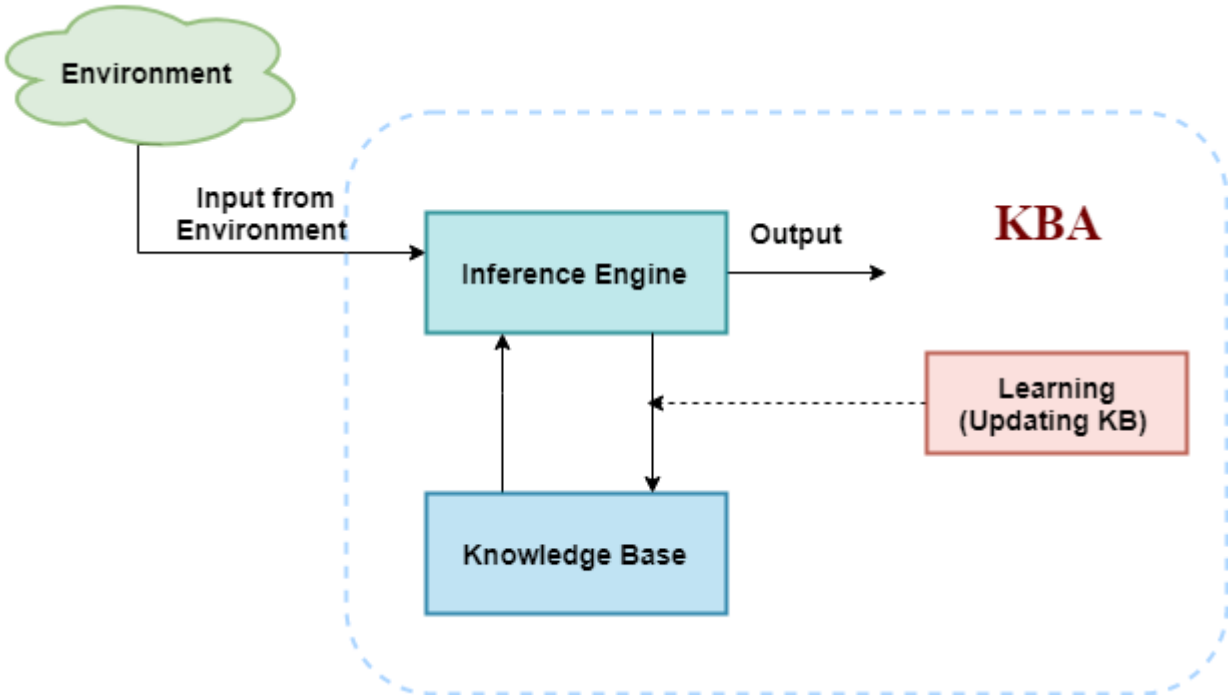
Knowledge-Based Agent in Artificial intelligence

- 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**

Forward Chaining:
Forward Chaining the Inference Engine goes through all the facts, conditions and derivations before deducing the outcome i.e When based on available data a decision is taken then the process is called as Forwarding chaining, It works from an initial state and reaches to the goal(final decision).
Example:
A
A -> B
B

He is running.
If he is running, he sweats.
He is sweating.

Backward Chaining:
In this, the inference system knows the final decision or goal, this system starts from the goal and works backwards to determine what facts must be asserted so that the goal can be achieved, i.e it works from goal(final decision) and reaches the initial state.
Example:
B

A -> B
A

He is sweating.
If he is running, he sweats.
He is running.

Operations Performed by KBA

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

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. **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. **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.

What is knowledge representation?

Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. **But how machines do all these things comes under knowledge representation and reasoning**. Hence we can describe Knowledge representation as following:

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.
- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.
- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

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).

Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

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 declarative sentences.
- 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.

The relation between knowledge and intelligence:

Knowledge of real-worlds plays a vital role in intelligence and same for creating artificial intelligence. Knowledge plays an important role in demonstrating intelligent behavior in AI agents. An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

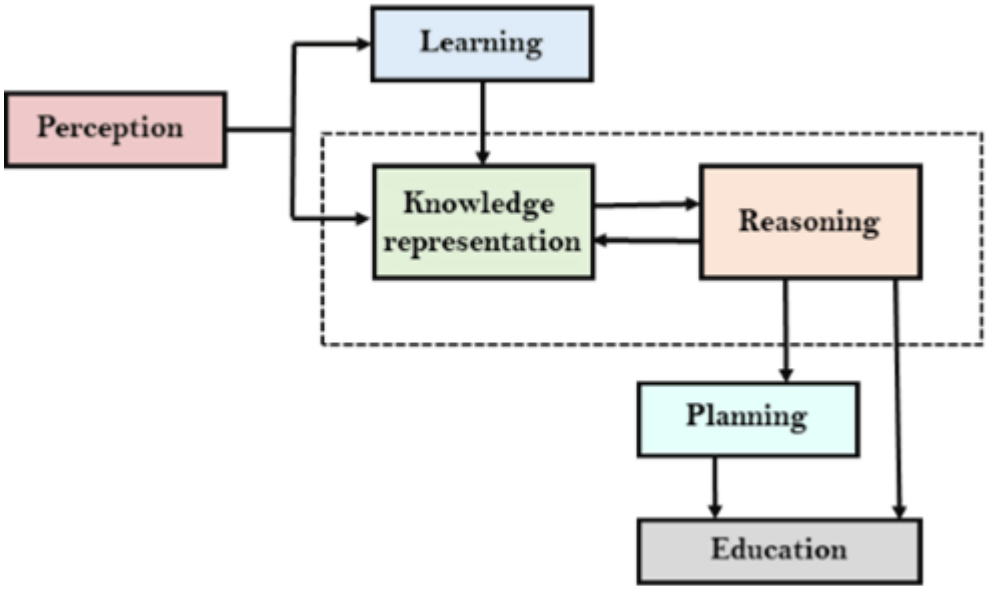
Let's suppose if you met some person who is speaking in a language which you don't know, then how you will able to act on that. The same thing applies to the intelligent behavior of the agents.

As we can see in below diagram, there is one decision maker which act by sensing the environment and using knowledge. But if the knowledge part will not present then, it cannot display intelligent behaviour.

AI knowledge cycle:

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

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



The above diagram is showing how an AI system can interact with the real world and what components help it to show intelligence. AI system has Perception component by which it retrieves information from its environment. It can be visual, audio or another form of sensory

input. The learning component is responsible for learning from data captured by Perception comportment. In the complete cycle, the main components are knowledge representation and Reasoning. These two components are involved in showing the intelligence in machine-like humans. These two components are independent with each other but also coupled together. The planning and execution depend on analysis of Knowledge representation and reasoning.

Approaches to knowledge representation:

There are mainly four approaches to knowledge representation, which are given below:

1. Simple relational knowledge:

- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This approach has little opportunity for inference.

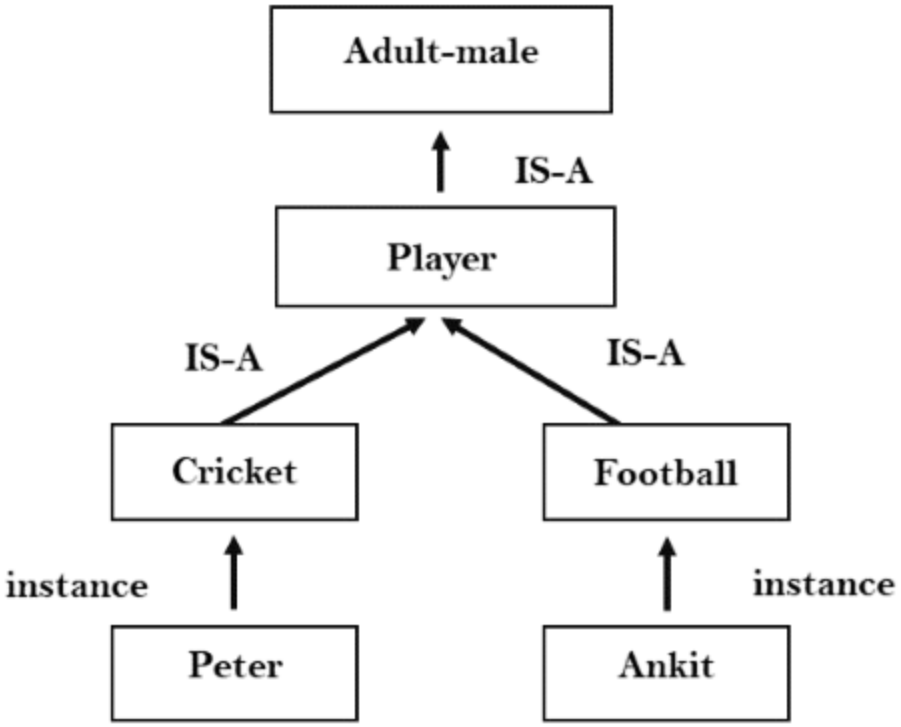
Example: The following is the simple relational knowledge representation.

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

2. Inheritable knowledge:

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- All classes should be arranged in a generalized form or a hierarchal manner.
- In this approach, we apply inheritance property.
- Elements inherit values from other members of a class.
- This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.
- Every individual frame can represent the collection of attributes and its value.
- In this approach, objects and values are represented in Boxed nodes.
- We use Arrows which point from objects to their values.

- **Example:**



3. Inferential knowledge:

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.
- **Example:** Let's suppose there are two statements:
    1. Marcus is a man
    2. All men are mortal
       Then it can represent as;
       a. man(Marcus)
       b. ∀x = man (x) ----------> mortal (x)

4. Procedural knowledge:

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- But it is not necessary that we can represent all cases in this approach.

Requirements for knowledge Representation system:

A good knowledge representation system must possess the following properties.

1. Representational Accuracy:
KR system should have the ability to represent all kind of required knowledge.

2. Inferential Adequacy:
KR system should have ability to manipulate the representational structures to produce new knowledge corresponding to existing structure.

3. Inferential Efficiency:
The ability to direct the inferential knowledge mechanism into the most productive directions by storing appropriate guides.

4. Acquisitional efficiency- The **ability to acquire the new knowledge** easily using automatic methods.

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:

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

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**.

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:

a) It is Sunday.

b) The Sun rises from West (False proposition)

c) 3+3= 7(False proposition)

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**.
- 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:

- **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**

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	P∨ Q	P∨Q→¬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:
  1. **All the students are intelligent.**
  2. **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:

15.3M

244

Hello Java Program for Beginners

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).

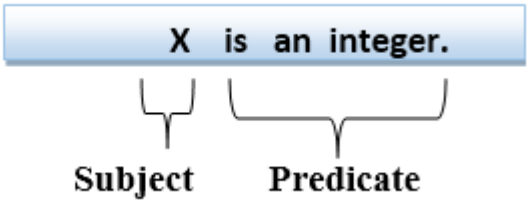
Complex Sentences:

- 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:
    1. **Universal Quantifier, (for all, everyone, everything)**
    2. **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.

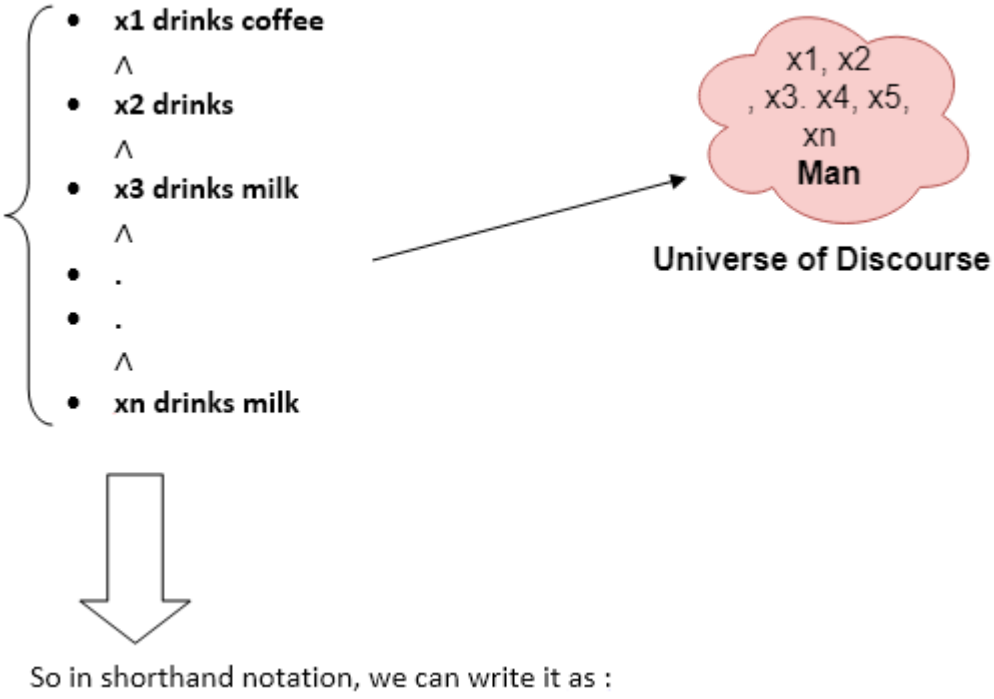
If x is a variable, then ∀x is read as:

- **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:



∀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.

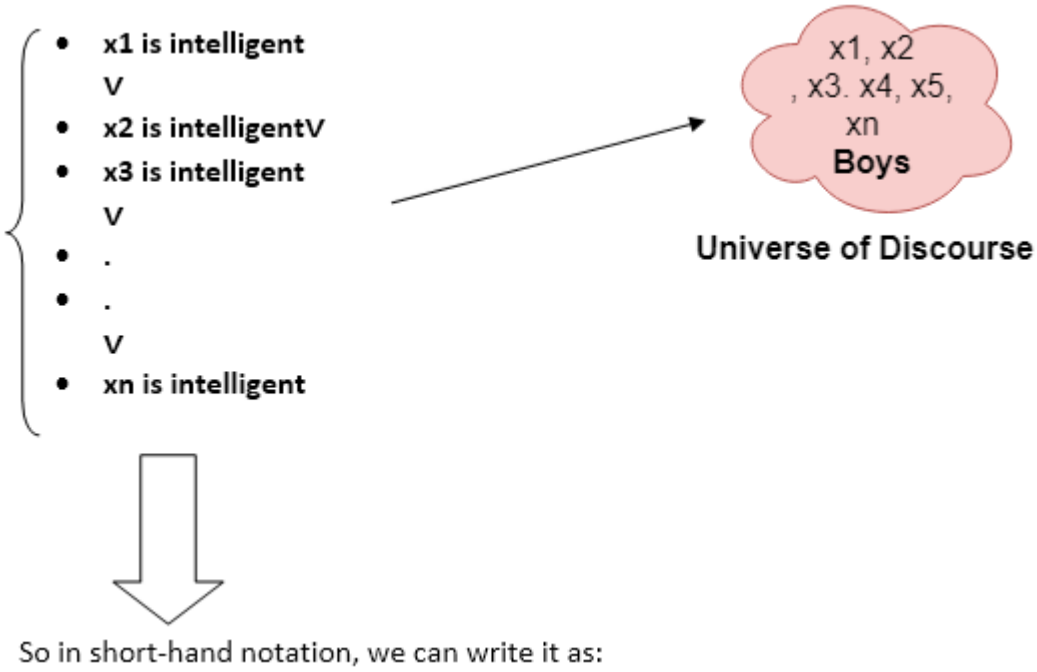
It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

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. 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)].

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"=> ∀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:

∀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 ∃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: $\frac{\exists x\, P(x)}{P(c)}$

Example:

From the given sentence: **∃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 generalization

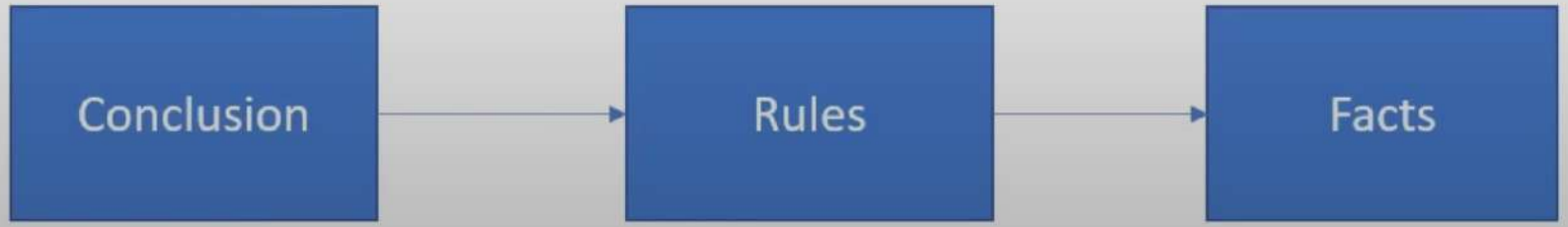
- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which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

Forward Chaining

- Forward chaining starts with the available data and uses inference rules to extract more data until a goal is reached.
- **"What will happen next?"**
- It is a bottom up approach.
- Forward chaining is data driven because the reasoning starts from a set of data.

Backward Chaining

- Backward chaining starts from goal and proceeds backward to determine the set of rules that match the goal.
- **"Why did this happen?"**
- It is top down approach.
- Forward chaining is goal driven
- Example: diagnose bacterial infections.



Example

- The law says that it is a crime for an American to sell weapons to hostile nations. Country Q, an enemy of America, has some missiles, and all of its missiles were sold to it by Jack, who is American.
- Prove that **"Jack is criminal".**

- it is a crime for an American to sell weapons to hostile nations

  $American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \Rightarrow Criminal(x)$
- Q has some missiles

  $\exists x\ Owns(Q,x) \land Missile(x)$

  $Owns(Q,M1)$

  $Missile(M1)$

Conversion into FOL

- all of its missiles were sold to it by Jack

  $\forall x\ \text{Missile}(x) \wedge \text{Owns}(Q,x) \Rightarrow \text{Sells}(\text{Jack},x,Q)$

  $\text{Missile}(x) \wedge \text{Owns}(Q,x) \Rightarrow \text{Sells}(\text{Jack},x,Q)$
- Missiles are weapons

  $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$
- Enemies of America are hostile

  $\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(Q)$
- Jack, who is American

  $\text{American}(\text{Jack})$
- The country Q, an enemy of America

  $\text{Enemy}(Q,\text{America})$

Forward Chaining->Facts,Rules,Conclusion

1. American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
2. Owns(Q,M1)
3. Missile(M1)
4. Missile(x) ∧ Owns(Q,x) ⇒ Sells(Jack,x,Q)
5. Missile(x) ⇒ Weapon(x)
6. Enemy(x, America) ⇒ Hostile(Q)
7. American(Jack)
8. Enemy(Q,America)

Step 1-Forward Chaining

Owns(Q,M1)

Missile(M1) => **Known Facts**

American(Jack)

Enemy(Q, America)

American(Jack) | Missile(M1) | Owns(A,M1) | Enemy(A, America)

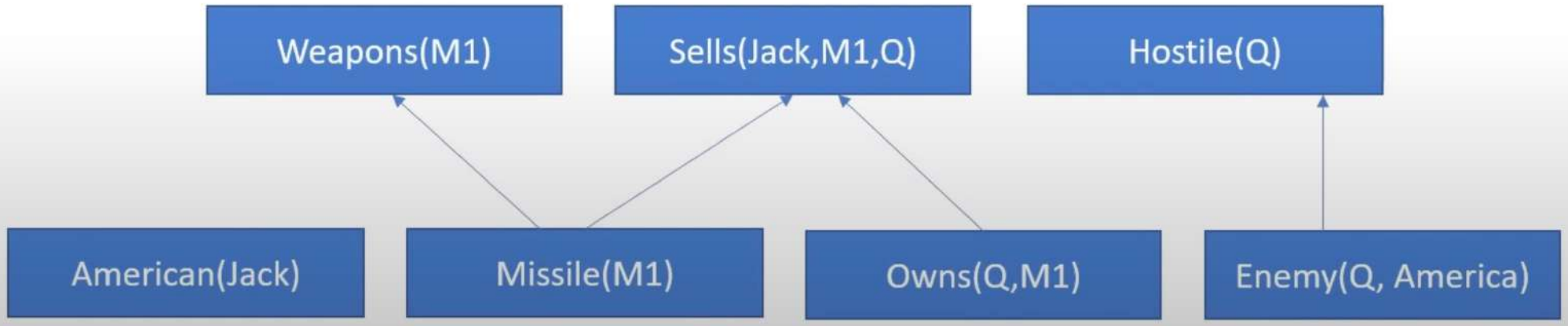
Forward Chaining->Facts,Rules,Conclusion

1. American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
2. Owns(Q,M1)
3. Missile(M1)
4. Missile(x) ∧ Owns(Q,x) ⇒ Sells(Jack,x,Q)
5. Missile(x) ⇒ Weapon(x)
6. Enemy(x, America) ⇒ Hostile(Q)
7. American(Jack)
8. Enemy(Q,America)

Step 2-Forward Chaining

- Missile(x) ⇒ Weapon(x)
- ∀x Missile(x) ∧ Owns(Q,x) ⇒ Sells(Jack,x,Q)
- Enemy(x, America) ⇒ Hostile(Q)

=> Rules that satisfy the facts



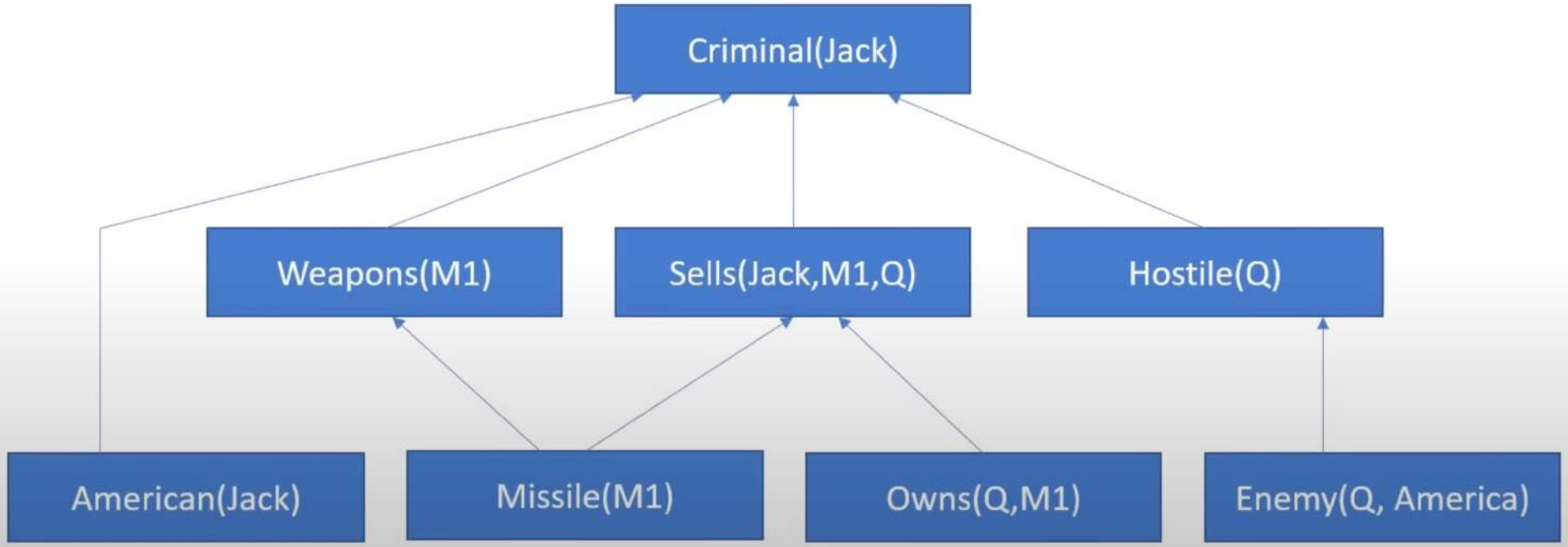
Forward Chaining->Facts,Rules,Conclusion

1. American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
2. Owns(Q,M1)
3. Missile(M1)
4. Missile(x) ∧ Owns(Q,x) ⇒ Sells(Jack,x,Q)
5. Missile(x) ⇒ Weapon(x)
6. Enemy(x, America) ⇒ Hostile(Q)
7. American(Jack)
8. Enemy(Q,America)

Step 3-Forward Chaining

- American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Proof



Backward Chaining
Conclusion,Rules,Facts

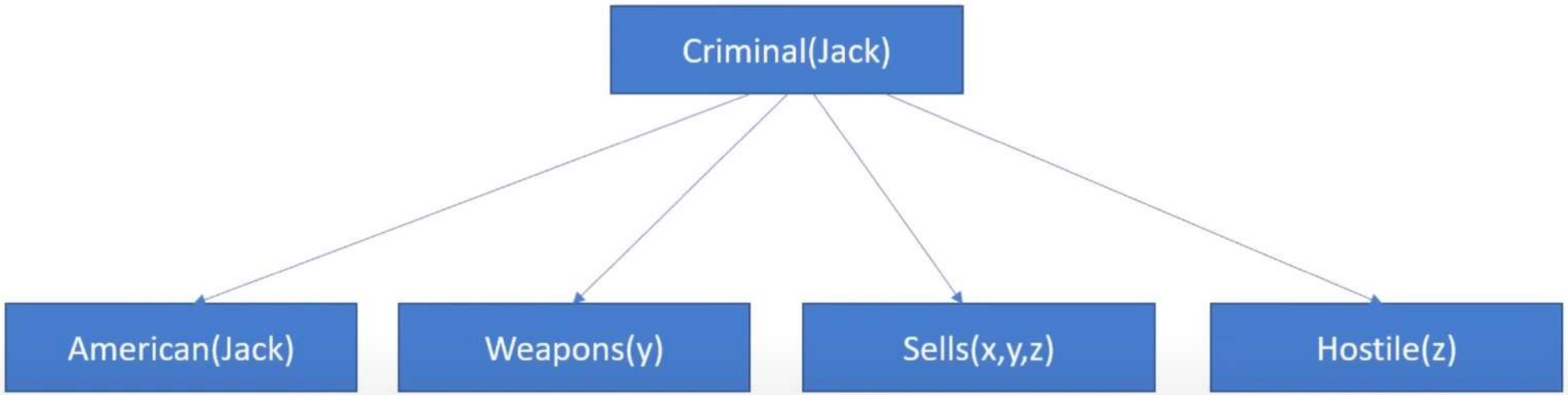
1. $American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$
2. $Owns(Q,M1)$
3. $Missile(M1)$
4. $Missile(x) \wedge Owns(Q,x) \Rightarrow Sells(Jack,x,Q)$
5. $Missile(x) \Rightarrow Weapon(x)$
6. $Enemy(x, America) \Rightarrow Hostile(Q)$
7. $American(Jack)$
8. $Enemy(Q,America)$

Backward Chaining-Conclusion,Rules,Facts

Backward Chaining
Conclusion,Rules,Facts

1. American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
2. Owns(Q,M1)
3. Missile(M1)
4. Missile(x) ∧ Owns(Q,x) ⇒ Sells(Jack,x,Q)
5. Missile(x) ⇒ Weapon(x)
6. Enemy(x, America) ⇒ Hostile(Q)
7. American(Jack)
8. Enemy(Q,America)

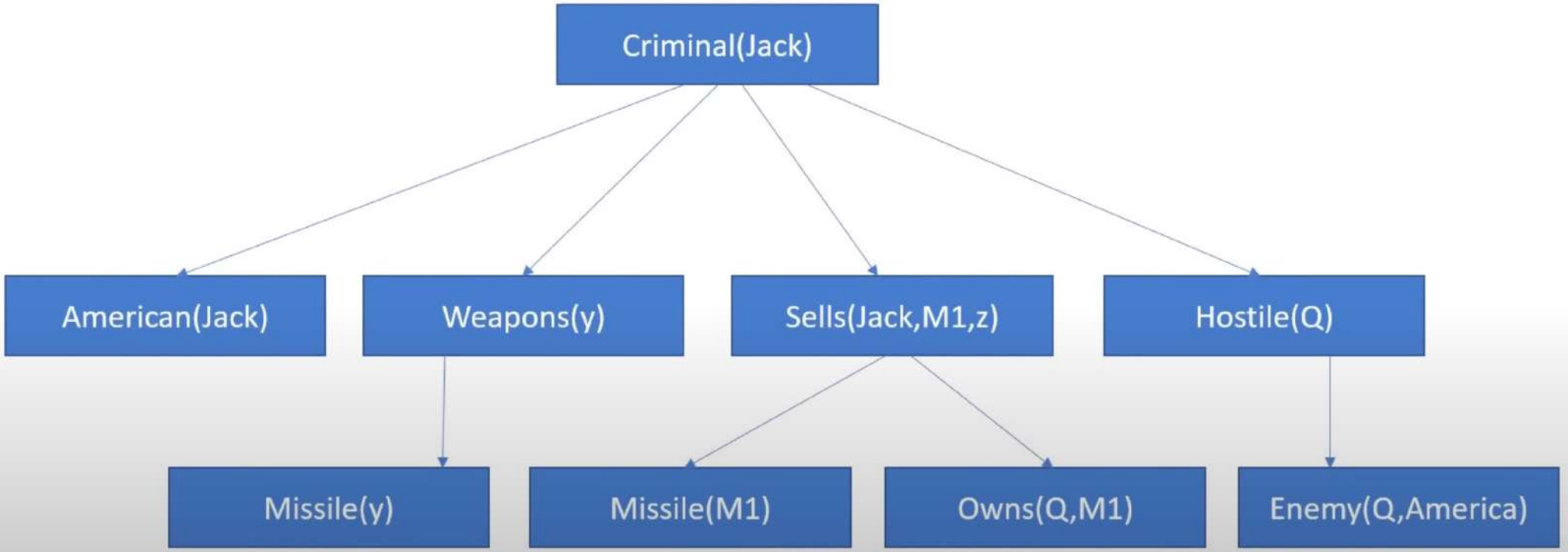
Backward Chaining-Conclusion,Rules,Facts



Backward Chaining
Conclusion,Rules,Facts

1. $American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$
2. $Owns(Q,M1)$
3. $Missile(M1)$
4. $Missile(x) \wedge Owns(Q,x) \Rightarrow Sells(Jack,x,Q)$
5. $Missile(x) \Rightarrow Weapon(x)$
6. $Enemy(x, America) \Rightarrow Hostile(Q)$
7. $American(Jack)$
8. $Enemy(Q,America)$

Backward Chaining-Conclusion,Rules,Facts



Unification

- Unification is a kind of binding logic between two or more variables.
- In propositional logic it is easy to determine that two literals can not both be true at the same time.

  Eg. Man(Sunil) and ¬Man(Sunil) is a contradiction

  while Man(Sunil) and ¬Man(Arun) is not .
- In predicate logic, this matching process is more complicated, since bindings of variables must be considered.
- In Predicate logic in order to determine contradiction we need a matching procedure that compares two literals and discovers whether there exist a set of substitutions that make them identical. **Unification algorithms**

Unification...

- The goal of unification is to make two expression look like identical by using substitution
- It means the meaning of the sentence should not be changed, but it should be expressed in multiple ways.
- The **UNIFY** algorithm in unification takes two sentences as input and then returns a unifier if one exists:
  - Substitution means replacing one variable with another term.
  - It takes two literals as input and makes them identical using substitution.
  - It returns fail if the expressions do not match with each other.

UNIFY(p, q)= θ where SUBST(θ , p) = SUBST(θ, q).

Example #1

- Let's say there are two different expressions,

  P(x, y), and P(a, f(z)).
- we need to make both above statements identical to each other.
- 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]**.

Example #2

- **Given:** Knows(Ram,x). Is a predicate
- **Whom does Ram knows?**
- The UNIFY algorithm will search all the related sentences in the knowledge base, which could unify with Knows(Ram, x).

  UNIFY (Knows(Ram, x), Knows(Ram, Shyam))≡{x/Shyam}

  UNIFY (Knows{Ram,x}, Knows{y, Aakash})≡{x/Aakash, y/Ram}

  UNIFY (Knows{Ram,x}, Knows{x, Raman})≡fails. **Unifier is empty**
- The last one failed because we have used the same variable for two persons or the two sentences happen to use the same variable name, x
- unifications are attempted only with sentences that have some chance of unifying.
- For example, there is no point in trying to

  unify Knows(Ram, x) with Brother(Laxman, Ram).

Conditions for Unification

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.

  tryassassinate (Marcus, Caesar)

  hate (Marcus, Caesar)

- Number of Arguments in both expressions must be identical.

  hate(Marcus)

  hate (Marcus, Caesar)

- Unification will fail if there are two similar variables present in the same expression.

Resolution

- Its an inference rule used in both Propositional as well as First-order Predicate Logic in different ways.
- It is also called **Proof by Refutation**
- It is basically used for proving the satisfiability of a sentence.
- Proof by Refutation technique is used to prove the given statement. It's a iterative process :
  1. Select two clauses that contain conflicting terms.
  2. Combine those two clauses and cancel out the conflicting terms, yielding a new clause that has been inferred from them.
- The key idea is to use the knowledge base and negated goal to obtain null clause(which indicates contradiction).
- 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.

Steps in Resolution

1. Convert facts into First order logic (FOL)
2. Convert FOL into CNF (Conjuctive Normal Form)
3. Negate the statement to be proved , and add the result to the knowledge base.
4. Draw Resolution graph .
5. If empty clause (NIL) is produced , stop and report that original theorem is true .

Example #1

1. If It is sunny and warm day you will enjoy
2. If it is raining you will get wet
3. It is warm day
4. It is raining
5. It is sunny

Goal: You will enjoy

Prove: enjoy

Step 1 : Conversion to first order logic

- If It is sunny and warm day you will enjoy

  Sunny Λ warm → enjoy
- If it is raining you will get wet

  raining → wet
- It is warm day

  warm
- It is raining

  raining
- It is sunny

  Sunny

Step 2 : conversion to CNF

- **Sunny Λ Warm → enjoy**

  Eliminate implication:

  ¬(Sunny Λ warm) V enjoy

  Moving negation inside

  ¬Sunny V ¬warm V enjoy

- **raining → wet**

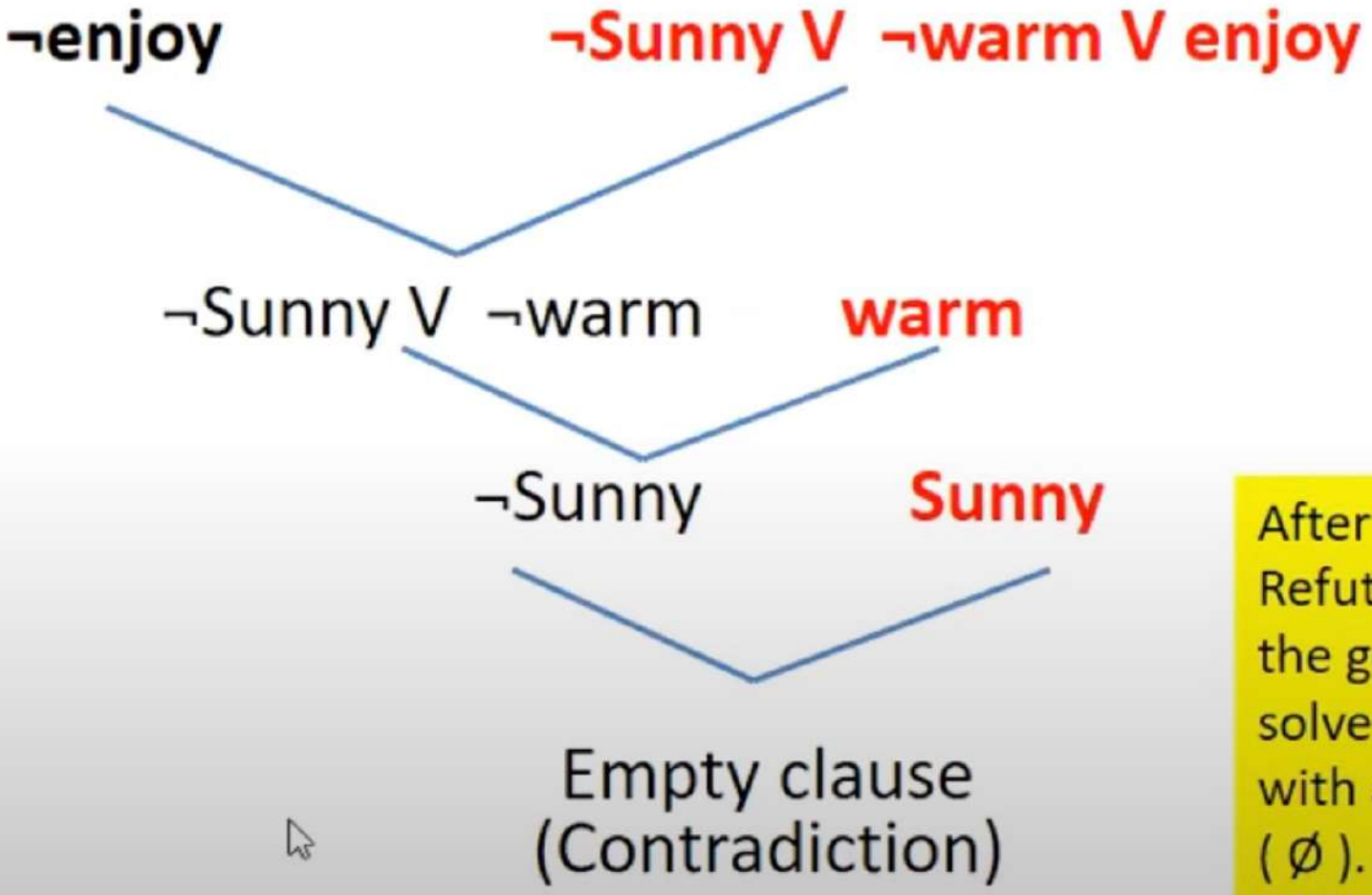
  Eliminate implication:

  ¬raining V wet

- **warm**
- **raining**
- **Sunny**

Step: 3 & 4 Resolution graph

- Negate the statement to be proved: **¬enjoy**
- Now take the statements one by one and create resolution graph.



After applying Proof by Refutation (Contradiction) on the goal, the problem is solved, and it has terminated with a **Null clause** (∅). **Hence, the goal is achieved.**

Example #2

- 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.

Step 1 : Conversion to first order logic

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.

- **Let P : The humidity is high**
- **Let Q: sky is cloudy**

The humidity is high or the sky is cloudy

P V Q

Conversion to first order logic...

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.

- **Let Q: sky is cloudy**
- **Let R: it will rain**

If the sky is cloudy, then it will rain

$Q \rightarrow R$

Conversion to first order logic...

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.

- **Let P : The humidity is high**
- **Let S: it is hot**

If the humidity is high, then it is hot

$P \rightarrow S$

Conversion to first order logic...

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.

- Let S: it is hot

It is not hot

¬S

Statements in first order logic

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.

- $P \vee Q$
- $Q \rightarrow R$
- $P \rightarrow S$
- $\neg S$

- **Goal:** It will rain. **(R)**

Convert it to CNF

- P V Q
- Q→R
- P→S
- ¬S

- P V Q
- ¬Q V R
- ¬P V S
- ¬S

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

Example #3

1. Sunil likes all kind of food.
2. Apple and vegetable are food
3. Anything anyone eats and not killed is food.
4. Anil eats peanuts and still alive
5. Sohan eats everything that Anil eats.

- **Prove by resolution that:**

  Sunil likes peanuts

Conversion of Facts/Statements into FOL

1. Sunil likes all kind of food.
2. Apple and vegetable are food
3. Anything anyone eats and not killed is food.
4. Anil eats peanuts and still alive
5. Sohan eats everything that Anil eats
6. Sunil likes peanuts.

1. $\forall x : food(x) \rightarrow likes(Sunil, x)$
2. $food(apple) \wedge food(vegetable)$
3. $\forall x \forall y : eats(x , y) \wedge \neg killed(x) \rightarrow food(y)$
4. $eats(Anil , Peanut) \wedge alive(Anil)$
5. $\forall x: eats(Anil, x) \rightarrow eats(Sohan , x)$
6. $Likes(Sunil, Peanut)$

- $\forall x : \neg killed(x) \rightarrow alive(x)$
- $\forall x : alive (x) \rightarrow \neg killed (x)$

(added predicates)

Conversion of FOL into CNF: Elimination of →

1. ∀x : food(x)→likes(Sunil, x)
2. food(apple) ∧ food(vegetable)
3. ∀x ∀y : eats(x , y) ∧ ¬ killed(x)→food(y)
4. eats(Anil , Peanut) ∧ alive(Anil)
5. ∀x: eats(Anil, x) → eats(Sohan , x)
6. ∀x : ¬ killed(x)→alive(x)
7. ∀x : alive (x)→¬killed (x)
8. Likes(Sunil, Peanut)

1. ∀x ¬ food(x) V likes(Sunil, 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(Sohan, x)
6. ∀x¬ [¬ killed(x)] V alive(x)
7. ∀x ¬ alive(x) V ¬ killed(x)
8. likes(Sunil, Peanuts)

Move negation (¬) inwards and rewrite

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

1. ∀x ¬ food(x) V likes(Sunil, 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(Sohan, x)
6. ∀x killed(x) V alive(x)
7. ∀x ¬ alive(x) V ¬ killed(x)
8. likes(Sunil, Peanuts).

Rename variables or standardize variables

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

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

Drop Universal quantifiers

1. ∀x ¬ food(x) V likes(Sunil, 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(Sohan, w)
6. ∀g killed(g) V alive(g)
7. ∀k ¬ alive(k) V ¬ killed(k)
8. likes(Sunil, Peanuts).

1. ¬ food(x) V likes(Sunil, 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(Sohan, w)
8. killed(g) V alive(g)
9. ¬ alive(k) V ¬ killed(k)
10. likes(Sunil, Peanuts)

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

Negate the statement to be proved

- Prove by resolution that: **Sunil likes peanuts.**

  likes(Sunil, Peanuts).

  ¬likes(Sunil, Peanuts)

Resolution graph