UNIT 1

Artificial Intelligence

"It is a branch of computer science by which we can create intelligent machines which can behave like a human, think like humans, and able to make decisions."

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems. Specific applications of AI include expert systems, natural language processing, speech recognition and machine vision.

Application of Artificial Intelligence include:

1. AI in Astronomy
2. AI in Healthcare
3. AI in Gaming
4. AI in Finance
5. AI in Data Security
6. AI in Social Media
7. AI in Travel & Transport
8. AI in Automotive Industry
9. AI in Robotics:
10. AI in Entertainment
11. AI in Agriculture
12. AI in E-commerce
13. AI in education:

Goals of Artificial Intelligence

Following are the main goals of Artificial Intelligence:

- Replicate human intelligence
- Solve Knowledge-intensive tasks
- An intelligent connection of perception and action
- Building a machine which can perform tasks that requires human intelligence such as:
    - Proving a theorem
    - Playing chess
    - Plan some surgical operation
    - Driving a car in traffic
- Creating some system which can exhibit intelligent behaviour, learn new things by itself, demonstrate, explain, and can advise to its user.

Approaches for AI

An algorithm is a kind of container, and it provides a box for storing a method to solve a particular kind of problem. Algorithms process data through a series of well-defined states. States do not need to be deterministic, but states are defined nonetheless. The goal is to create an output that solves a

problem. The algorithm receives input that helps define the output in some cases, but the focus is always on the output.

Algorithms must express transitions between states using a well-defined and formal language that the computer can understand. In processing data and solving a problem, the algorithm defines, refines, and performs a function. The function is always specific to the type of problem being addressed by the algorithm.

Each of the five tribes has a different technique and strategy for solving those problems resulting in unique algorithms. The combination of these algorithms should eventually lead to the master algorithm, which will solve any problem. The following discussion provides an overview of the five main algorithmic techniques.

1: Symbolic logic

One of the ancient tribes, the Symbolists, believed that knowledge could be gained by working on symbols (signs that stand for a certain meaning or event) and drawing rules from them.

2: Symbolic reasoning

One of the earliest tribes, the symbolists, believed that knowledge could be obtained by operating on symbols (signs that stand for a certain meaning or event) and deriving rules from them.

By putting together complex rules systems, you could attain a logical deduction of the result you wanted to know; thus, the symbolists shaped their algorithms to produce rules from data. In symbolic logic, *deduction* expands the scope of human knowledge, while *induction* increases the level of human knowledge. Induction usually opens up new areas of exploration, whereas deduction explores those areas.

3: The connections are based on the neurons of the brain.

The Connectionists are perhaps the most famous of the five tribes. This tribe attempts to reproduce brain functions by using silicon instead of neurons. Essentially, each of the neurons (built as an algorithm that models the real-world counterpart) solves a small piece of the problem, and using multiple neurons in parallel solves the problem as a whole.

The goal is to keep changing the weights and biases until the actual output matches the target output. The artificial neuron fires up and transmits its solution to the next neuron in line. The solution produced by just one neuron is a part of the whole solution. Each neuron sends information to the next neuron until the neurons make up the final output. Such a method proved most effective in human-like tasks such as recognizing objects, understanding written and spoken language and interacting with humans.

4: Evolutionary algorithms that test variation

The revolutionaries relied on the principles of evolution to solve problems. In other words, this strategy is based on the existence of the fittest (removing any solutions that do not match the desired output). A fitness function determines the feasibility of each function in solving a problem.

Using a tree structure, the solution method finds the best solution based on the function output. The winner of each level of development has to create tasks for the next level.

The idea is that the next level will get closer to solving the problem but may not solve it completely, which means that another level is needed. This particular tribe relies heavily on recursion and languages that strongly support recursion to solve problems. An interesting output of this strategy has been algorithms that evolve: one generation of algorithms creates the next generation.

5: Bayesian Approximation

A group of Bayesian scientists recognized that uncertainty was the dominant aspect of the view. Learning was not assured but rather occurred as a continuous update of previous assumptions that became more accurate. This notion inspired Bayesians to adopt statistical methods and, in particular, derivations from Bayes' theorem, which help you calculate probabilities in specific situations (for example, by looking at a card of a certain *seed,* pseudo -The starting value for a random sequence, after three other cards of the same seed are drawn from a deck).

6: Systems that learn by analogy

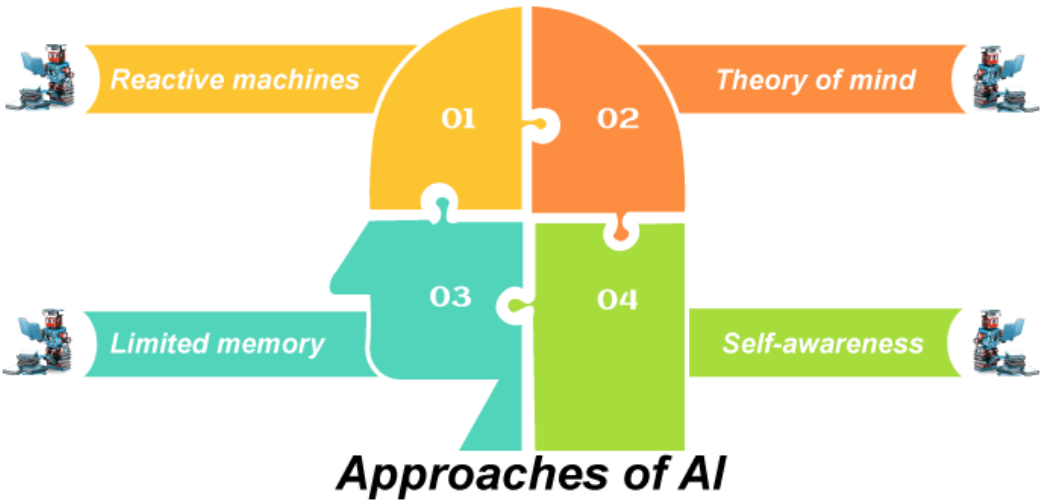
Analysts use kernel machines to recognize patterns in the data. By recognizing the pattern of a set of inputs and comparing it to known outputs, you can create a problem solution. The goal is to use equality to determine the best solution to a problem. It is the kind of reasoning that determines whether a particular solution was used in a particular situation at a prior time. Using that solution for similar situations should also work.

One of the most recognizable outputs of this tribe is the recommendation system. For example, when you buy a product on Amazon, the recommendation system comes up with other related products that you might want to buy.

The ultimate goal of machine learning is to combine the techniques and strategies adopted by the five tribes to form a single master algorithm that can learn anything. Of course, achieving that goal is a long way off, yet scientists like Pedro Domingos are currently working toward that goal.

What are the Different Types of Artificial Intelligence Approaches?

While everything seems green and sunny to a non-specialist, there is a lot of technology to build AI systems. There are four types of artificial intelligence approaches based on how machines behave - reactive machines, limited memory, theory of mind, and self-awareness.



1. Reactive machines

These machines are the most basic form of AI applications. Examples of reactive machines are **Deep Blue, IBM's** chess-playing supercomputer, and the same computer that defeated the then-grand master of the world.

AI teams do not use training sets to feed the machines or store subsequent data for future references. Based on the move made by the opponent, the machine decides/predicts the next move.

2. Limited memory

These machines belong to the Category **II** category of **AI applications**, and Self-propelled cars are the perfect example. Over time, these machines are fed with data and trained on the speed and direction of other **cars, lane markings, traffic lights, curves of roads**, and other important factors.

3. Theory of mind

It is where we are struggling to make this concept work. However, we are not there yet. Theory of mind is the concept where bots will understand and react to human emotions, thoughts.

If AI-powered machines are always mingling and moving around with us, then understanding human behavior is imperative. And then, it is necessary to react to such behaviors accordingly.

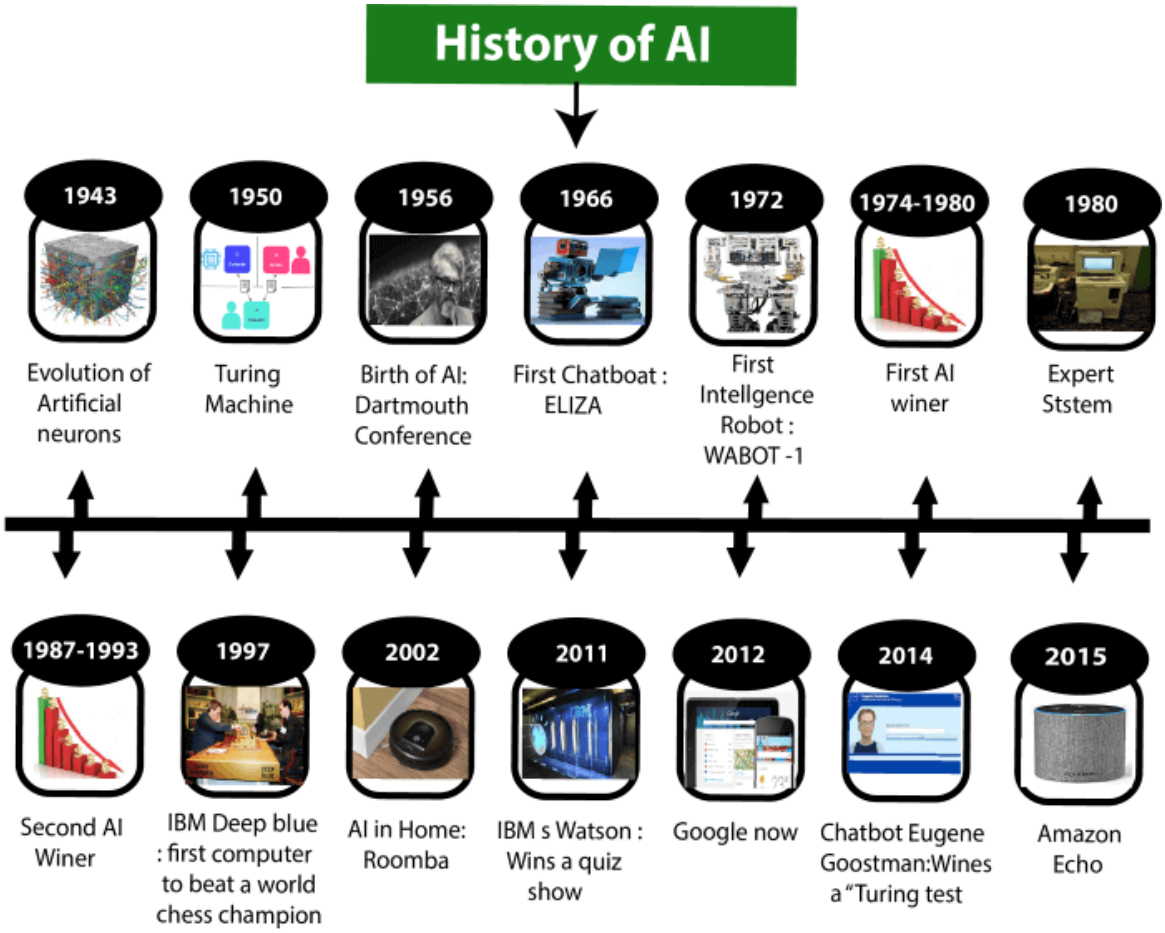
4. Self-awareness

These machines are an extension of class III type AI, and it is a step ahead of understanding human emotions. It is the stage where AI teams build machines with self-awareness factors programmed into them.

When someone is honking the horn from behind, the machines must sense the emotion, and only then do they understand what it feels like when they horn someone from behind.

History of Artificial Intelligence

Artificial Intelligence is not a new word and not a new technology for researchers. This technology is much older than you would imagine. Even there are the myths of Mechanical men in Ancient Greek and Egyptian Myths. Following are some milestones in the history of AI which defines the journey from the AI generation to till date development.



Maturation of Artificial Intelligence (1943-1952)

- **Year 1943:** The first work which is now recognized as AI was done by Warren McCulloch and Walter pits in 1943. They proposed a model of **artificial neurons**.
- **Year 1949:** Donald Hebb demonstrated an updating rule for modifying the connection strength between neurons. His rule is now called **Hebbian learning**.
- **Year 1950:** The Alan Turing who was an English mathematician and pioneered Machine learning in 1950. Alan Turing publishes **"Computing Machinery and Intelligence"** in which he proposed a test. The test can check the machine's ability to exhibit intelligent behavior equivalent to human intelligence, called a **Turing test**.

The birth of Artificial Intelligence (1952-1956)

- **Year 1955:** An Allen Newell and Herbert A. Simon created the "first artificial intelligence program"Which was named as **"Logic Theorist"**. This program had proved 38 of 52 Mathematics theorems, and find new and more elegant proofs for some theorems.

- **Year 1956:** The word "Artificial Intelligence" first adopted by American Computer scientist John McCarthy at the Dartmouth Conference. For the first time, AI coined as an academic field.

At that time high-level computer languages such as FORTRAN, LISP, or COBOL were invented. And the enthusiasm for AI was very high at that time.

The golden years-Early enthusiasm (1956-1974)

- **Year 1966:** The researchers emphasized developing algorithms which can solve mathematical problems. Joseph Weizenbaum created the first chatbot in 1966, which was named as ELIZA.
- **Year 1972:** The first intelligent humanoid robot was built in Japan which was named as WABOT-1.

The first AI winter (1974-1980)

- The duration between years 1974 to 1980 was the first AI winter duration. AI winter refers to the time period where computer scientist dealt with a severe shortage of funding from government for AI researches.
- During AI winters, an interest of publicity on artificial intelligence was decreased.

A boom of AI (1980-1987)

- **Year 1980:** After AI winter duration, AI came back with "Expert System". Expert systems were programmed that emulate the decision-making ability of a human expert.
- In the Year 1980, the first national conference of the American Association of Artificial Intelligence **was held at Stanford University**.

The second AI winter (1987-1993)

- The duration between the years 1987 to 1993 was the second AI Winter duration.
- Again Investors and government stopped in funding for AI research as due to high cost but not efficient result. The expert system such as XCON was very cost effective.

The emergence of intelligent agents (1993-2011)

- **Year 1997:** In the year 1997, IBM Deep Blue beats world chess champion, Gary Kasparov, and became the first computer to beat a world chess champion.
- **Year 2002:** for the first time, AI entered the home in the form of Roomba, a vacuum cleaner.
- **Year 2006:** AI came in the Business world till the year 2006. Companies like Facebook, Twitter, and Netflix also started using AI.

Deep learning, big data and artificial general intelligence (2011-present)

- **Year 2011:** In the year 2011, IBM's Watson won jeopardy, a quiz show, where it had to solve the complex questions as well as riddles. Watson had proved that it could understand natural language and can solve tricky questions quickly.
- **Year 2012:** Google has launched an Android app feature "Google now", which was able to provide information to the user as a prediction.
- **Year 2014:** In the year 2014, Chatbot "Eugene Goostman" won a competition in the infamous "Turing test."
- **Year 2018:** The "Project Debater" from IBM debated on complex topics with two master debaters and also performed extremely well.
- Google has demonstrated an AI program "Duplex" which was a virtual assistant and which had taken hairdresser appointment on call, and lady on other side didn't notice that she was talking with the machine.

Now AI has developed to a remarkable level. The concept of Deep learning, big data, and data science are now trending like a boom. Nowadays companies like Google, Facebook, IBM, and Amazon are working with AI and creating amazing devices. The future of Artificial Intelligence is inspiring and will come with high intelligence.

Intelligent agents

What is an Agent?

An agent can be anything that perceiveits environment through sensors and act upon that environment through actuators. An Agent runs in the cycle of **perceiving**, **thinking**, and **acting**. An agent can be:

- **Human-Agent:** A human agent has eyes, ears, and other organs which work for sensors and hand, legs, vocal tract work for actuators.
- **Robotic Agent:** A robotic agent can have cameras, infrared range finder, NLP for sensors and various motors for actuators.
- **Software Agent:** Software agent can have keystrokes, file contents as sensory input and act on those inputs and display output on the screen.

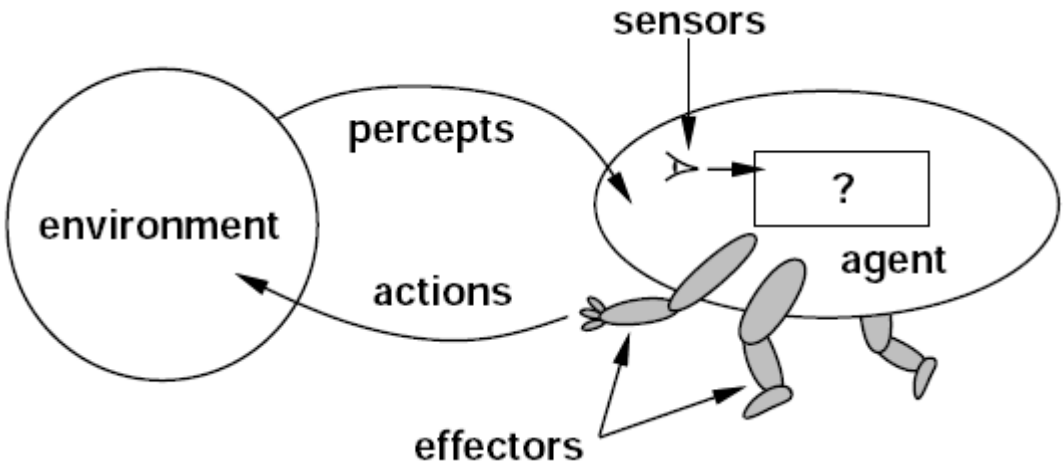
Hence the world around us is full of agents such as thermostat, cellphone, camera, and even we are also agents.

Before moving forward, we should first know about sensors, effectors, and actuators.

Sensor: Sensor is a device which detects the change in the environment and sends the information to other electronic devices. An agent observes its environment through sensors.

Actuators: Actuators are the component of machines that converts energy into motion. The actuators are only responsible for moving and controlling a system. An actuator can be an electric motor, gears, rails, etc.

Effectors: Effectors are the devices which affect the environment. Effectors can be legs, wheels, arms, fingers, wings, fins, and display screen.



Types of intelligent agents

Types of intelligent agents are defined by their range of capabilities and degree of intelligence:

- **Reflex agents:** These agents function in a current state, ignoring past history. Responses are based on the event-condition-action rule (ECA rule) where a user initiates an event and the agent refers to a list of pre-set rules and pre-programmed outcomes.
- **Model-based agents:** These agents choose an action in the same way as a reflex agent, but they have a more comprehensive view of the environment. A model of the world is programmed into the internal system that incorporates the agent's history.
- **Goal-based agents:** These agents expand upon the information model-based agents store by also including goal information, or information about desirable situations.
- Utility-based agents: These agents are similar to goal-based agents but provide an extra utility measurement which rates each possible scenario on its desired result and chooses the action that maximizes the outcome. Rating criteria examples could be the probability of success or the resources required.
- **Learning agents:** These agents have the ability to gradually improve and become more knowledgeable about an environment over time through an additional learning element. The learning element will use feedback to determine how performance elements should be changed to improve gradually.

Data Acquisition

- Data acquisition is the process of taking measurements of real-world physical occurrences using signals and digitizing them so that a computer and software may alter them.

SIMPLE REFLEX AGENT

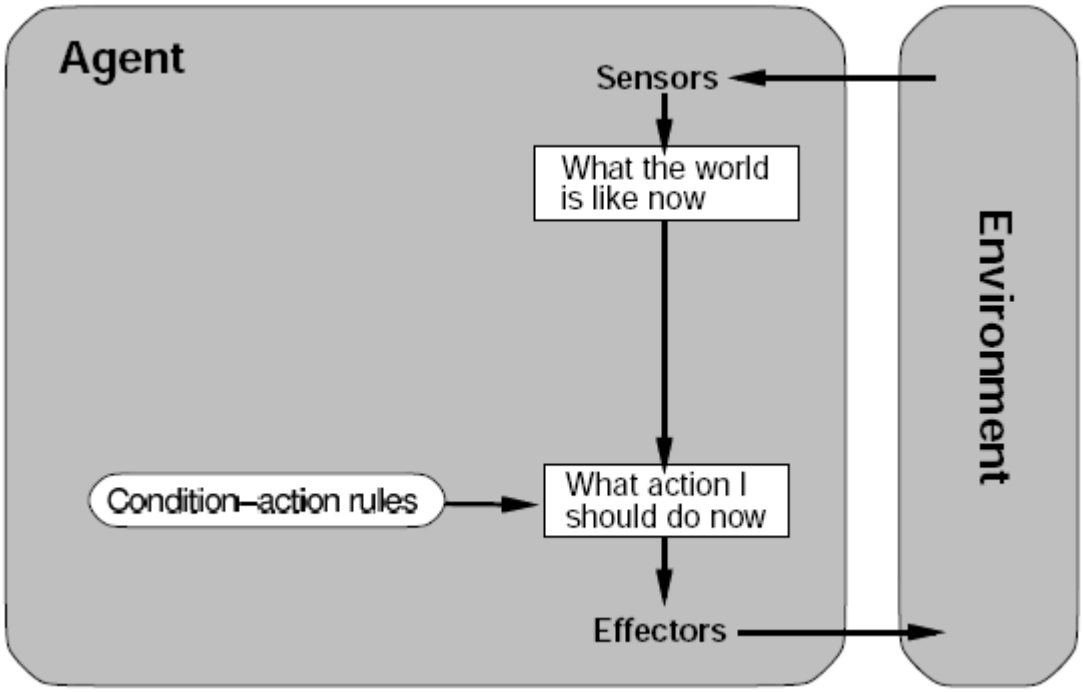
A **simple reflex agent** is the simplest of all the agent programs. Decisions or actions are taken based on the current percept, which does not depend on the rest of the percept history. These agents react only to the predefined rules. It works best when the environment is fully observable.

Example

The vacuum agent is a simple reflex agent because the decision is based only on the current location, and whether the place contains dirt.

A simple reflex agent comprises the following parts:

- **Agent**: The agent is the one who performs actions on the environment.
- **Sensors**: Sensors are the things that sense the environment. They are devices that measure physical property.
- **Actuators**: Actuators are devices that convert energy into motion.
- **Environment**: The environment includes the surroundings of the agent.



This agent selects actions based on the agents current perception or the world and not based on past perceptions.

This kind of connection where only one possibility is acted upon is called a condition-action rule, written as:

if hand is in fire **then** pull away hand

The simple reflex agent has a library of such rules so that if a certain situation should arise and it is in the set of Condition-action rules the agent will know how to react with minimal reasoning.

These agents are simple to work with but have very limited intelligence, such as picking up 2 rock samples.

```
function SIMPLE-REFLEX-AGENT(percept) returns action
  static: rules, a set of condition-action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  return action
```

Above is the logical representation of a Simple reflex agent. This will only work if the environment is fully observable.

PEAS Representation

PEAS is a type of model on which an AI agent works upon. When we define an AI agent or rational agent, then we can group its properties under PEAS representation model. It is made up of four words:

- **P:** Performance measure
- **E:** Environment
- **A:** Actuators
- **S:** Sensors

Here performance measure is the objective for the success of an agent's behavior.

PEAS for self-driving cars:

Let's suppose a self-driving car then PEAS representation will be:

Performance: Safety, time, legal drive, comfort

Environment: Roads, other vehicles, road signs, pedestrian

Actuators: Steering, accelerator, brake, signal, horn

Sensors: Camera, GPS, speedometer, odometer, accelerometer, sonar.

Agent Environment in AI

An environment is everything in the world which surrounds the agent, but it is not a part of an agent itself. An environment can be described as a situation in which an agent is present.

The environment is where agent lives, operate and provide the agent with something to sense and act upon it. An environment is mostly said to be non-feministic.

Features of Environment

As per Russell and Norvig, an environment can have various features from the point of view of an agent:

1. Fully observable vs Partially Observable
2. Static vs Dynamic
3. Discrete vs Continuous
4. Deterministic vs Stochastic
5. Single-agent vs Multi-agent
6. Episodic vs sequential
7. Known vs Unknown
8. Accessible vs Inaccessible

1. Fully observable vs Partially Observable:

- If an agent sensor can sense or access the complete state of an environment at each point of time then it is **a fully observable** environment, else it is **partially observable**.
- A fully observable environment is easy as there is no need to maintain the internal state to keep track history of the world.
- An agent with no sensors in all environments then such an environment is called as **unobservable**.

2. Deterministic vs Stochastic:

- If an agent's current state and selected action can completely determine the next state of the environment, then such environment is called a deterministic environment.
- A stochastic environment is random in nature and cannot be determined completely by an agent.
- In a deterministic, fully observable environment, agent does not need to worry about uncertainty.

3. Episodic vs Sequential:

- In an episodic environment, there is a series of one-shot actions, and only the current percept is required for the action.
- However, in Sequential environment, an agent requires memory of past actions to determine the next best actions.

4. Single-agent vs Multi-agent

- If only one agent is involved in an environment, and operating by itself then such an environment is called single agent environment.
- However, if multiple agents are operating in an environment, then such an environment is called a multi-agent environment.
- The agent design problems in the multi-agent environment are different from single agent environment.

5. Static vs Dynamic:

- If the environment can change itself while an agent is deliberating then such environment is called a dynamic environment else it is called a static environment.
- Static environments are easy to deal because an agent does not need to continue looking at the world while deciding for an action.
- However for dynamic environment, agents need to keep looking at the world at each action.
- Taxi driving is an example of a dynamic environment whereas Crossword puzzles are an example of a static environment.

6. Discrete vs Continuous:

- If in an environment there are a finite number of percepts and actions that can be performed within it, then such an environment is called a discrete environment else it is called continuous environment.
- A chess gamecomes under discrete environment as there is a finite number of moves that can be performed.
- A self-driving car is an example of a continuous environment.

7. Known vs Unknown

- Known and unknown are not actually a feature of an environment, but it is an agent's state of knowledge to perform an action.
- In a known environment, the results for all actions are known to the agent. While in unknown environment, agent needs to learn how it works in order to perform an action.

- It is quite possible that a known environment to be partially observable and an Unknown environment to be fully observable.

8. Accessible vs Inaccessible

- If an agent can obtain complete and accurate information about the state's environment, then such an environment is called an Accessible environment else it is called inaccessible.
- An empty room whose state can be defined by its temperature is an example of an accessible environment.
- Information about an event on earth is an example of Inaccessible environment.

Search Algorithms in Artificial Intelligence

Search algorithms are one of the most important areas of Artificial Intelligence. This topic will explain all about the search algorithms in AI.

Problem-solving agents:

In Artificial Intelligence, Search techniques are universal problem-solving methods. **Rational agents** or **Problem-solving agents** in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result. Problem-solving agents are the goal-based agents and use atomic representation. In this topic, we will learn various problem-solving search algorithms.

Search Algorithm Terminologies:

- **Search:** Searchingis a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
    a. **Search Space:** Search space represents a set of possible solutions, which a system may have.
    b. **Start State:** It is a state from where agent begins **the search**.
    c. **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.
- **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
- **Actions:** It gives the description of all the available actions to the agent.
- **Transition model:** A description of what each action do, can be represented as a transition model.
- **Path Cost:** It is a function which assigns a numeric cost to each path.
- **Solution:** It is an action sequence which leads from the start node to the goal node.
- **Optimal Solution:** If a solution has the lowest cost among all solutions.

Properties of Search Algorithms:

Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:

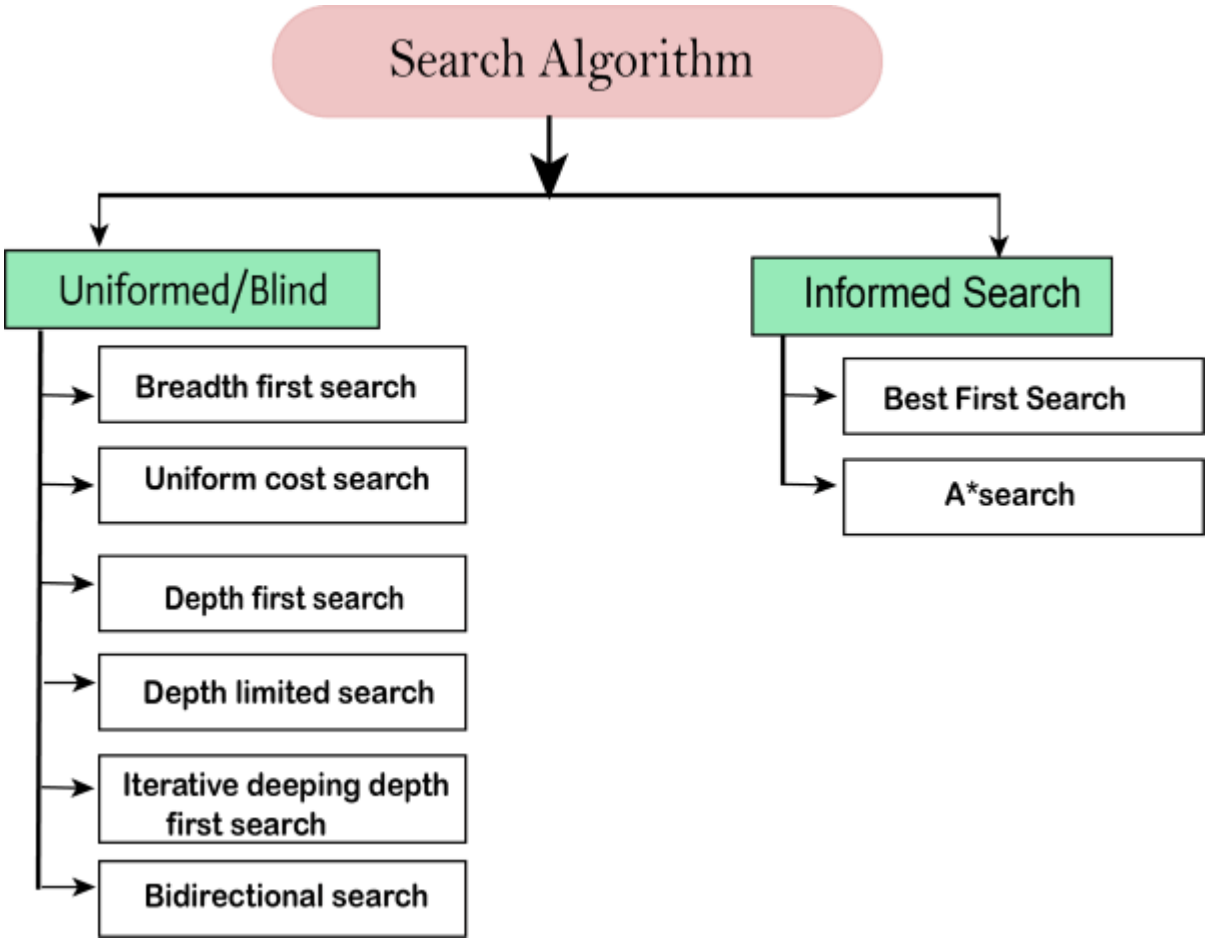
Completeness: A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.

Time Complexity: Time complexity is a measure of time for an algorithm to complete its task.

Space Complexity: It is the maximum storage space required at any point during the search, as the complexity of the problem.

Types of search algorithms

Based on the search problems we can classify the search algorithms into uninformed (Blind search) search and informed search (Heuristic search) algorithms.



Uninformed/Blind Search:

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes. Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search.It examines each node of the tree until it achieves the goal node.

It can be divided into five main types:

- Breadth-first search
- Uniform cost search
- Depth-first search
- Iterative deepening depth-first search
- Bidirectional Search

Informed Search

Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called a Heuristic search.

A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.

Informed search can solve much complex problem which could not be solved in another way.

An example of informed search algorithms is a traveling salesman problem.

1. Greedy Search
2. A* Search

Basis of comparison	Informed search	Uninformed search
Basic knowledge	Uses knowledge to find the steps to the solution.	No use of knowledge
Efficiency	Highly efficient as consumes less time and cost.	Efficiency is mediatory
Cost	Low	Comparatively high
Performance	Finds the solution more quickly.	Speed is slower than the informed search.
Algorithms	Heuristic depth-first and breadth-first search, and A* search	Depth-first search, breadth-first search, and lowest cost first search

Blind search:

- it is totally brute in nature because it doesn't have any domain specific knowledge.
- it is a very lengthy process
- it is also called uninformed or Brute Force search.
- large memory is used.
- the search process remembers all the unwanted nodes which are no use for the search process.
- it doesn't use any special function for searching.
- example: depth first search and breadth first search.

Heuristic search:

- they use domain-specific knowledge to do the search process.

- by the use of heuristic the search process is reduced.
- this is called informed search.
- no time is wasted in this type of search.
- no large memory is used.
- heuristic functions are used for searching.
- example: hill climbing,best first search and A* and AO*.

UNIT 2

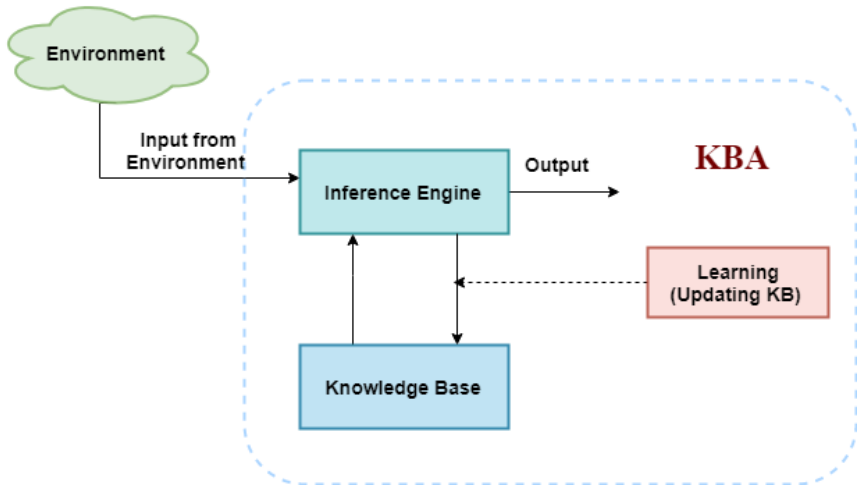
Knowledge-Based Agent in Artificial intelligence

- 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.

Propositional logic in Artificial intelligence

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

Example:

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

Following are some basic facts about propositional logic:

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

Syntax of propositional logic:

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

a. **Atomic Propositions**

b. **Compound propositions**

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

Example:

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

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

Example:

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

Logical Connectives:

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

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

Following is the summarized table for Propositional Logic Connectives:

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

Truth Table:

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

For Negation:

P	¬ P
True	**False**
False	**True**

For Conjunction:

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

For disjunction:

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

For Implication:

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

For Biconditional:

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

Truth table with three propositions:

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

P	Q	R	¬R	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

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

Logical equivalence:

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

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

A	B	¬A	¬AV 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 → 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 → P.
- **Contrapositive:** The negation of converse is termed as contrapositive, and it can be represented as ¬ Q → ¬ P.
- **Inverse:** The negation of implication is called inverse. It can be represented as ¬ P → ¬ 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: $\frac{P \rightarrow Q, \quad P}{\therefore Q}$

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: $\frac{P \rightarrow Q, \ \sim Q}{\sim P}$

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: $\frac{P \vee Q, \ \neg P}{Q}$

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 $\frac{P \vee Q,\ \neg P \wedge R}{Q \vee R}$

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:

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).

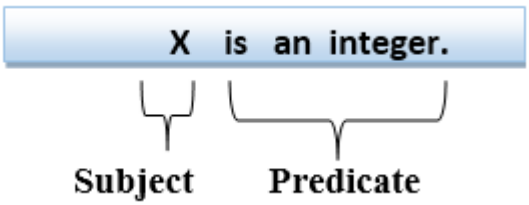
Complex Sentences:

- 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.

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

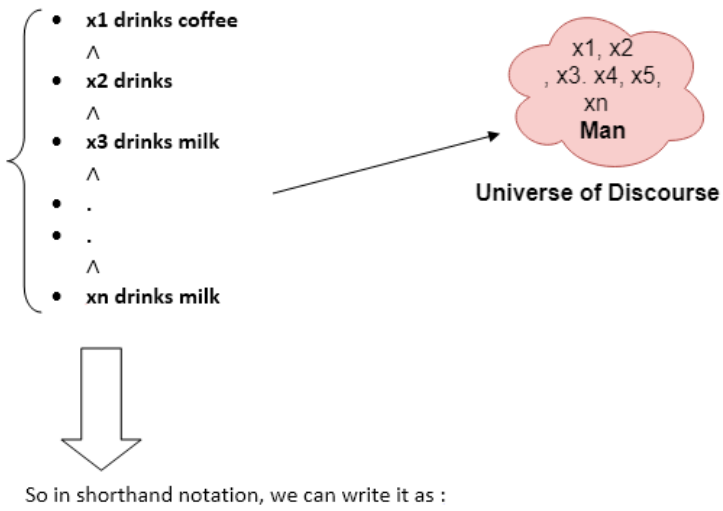
If x is a variable, then ∀x is read as:

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

Example:

All man drink coffee.

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



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.

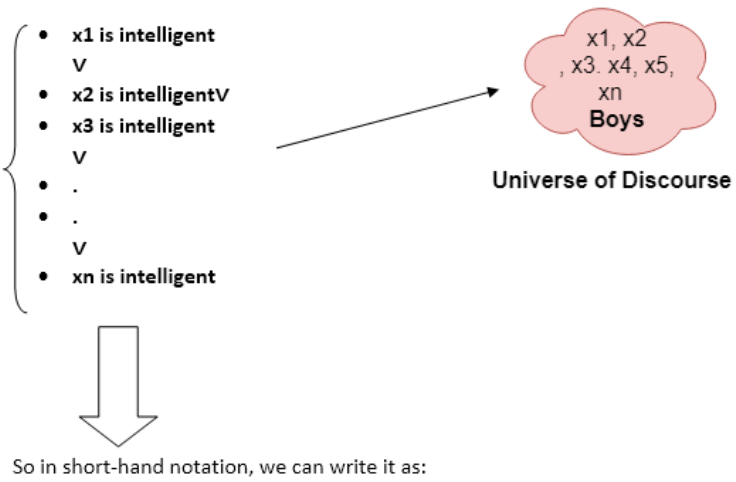
> *Note: In Existential quantifier we always use AND or Conjunction symbol (∧).*

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

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

Example:

Some boys are intelligent.



∃x: boys(x) ∧ intelligent(x)

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

Points to remember:

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

Properties of Quantifiers:

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

Some Examples of FOL using quantifier:

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

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

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

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

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

Free and Bound Variables:

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

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

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

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

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

What is Unification?

- Unification is the process of finding a substitute that makes two separate logical atomic expressions identical. The substitution process is necessary for unification.
- It accepts two literals as input and uses substitution to make them identical.
- Let Ψ_1 and Ψ_2 be two atomic sentences, and be a unifier such that $\boldsymbol{\Psi_{1\sigma} = \Psi_{2\sigma}}$, then **UNIFY($\Psi_1$, Ψ_2)**can be written.
- **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 both equations will be equivalent if this substitution is used.

- For unification, the UNIFY algorithm is employed, which takes two atomic statements and returns a unifier for each of them (If any exist).
- All first-order inference techniques rely heavily on unification.
- If the expressions do not match, the result is failure.
- The replacement variables are referred to as MGU (Most General Unifier).
  - **E.g.** Let's say there are two different expressions, **P(x, y), and P(a, f(z))**.
    In this case, we must make both of the preceding assertions identical. For this, we'll make a substitute.
    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.
    The first expression will be identical to the second expression with both replacements, and the substitution set will be: [**a/x** , f(z)/y].

Conditions for Unification

The following are some fundamental requirements for unification:

- Atoms or expressions with various predicate symbols can never be united.
- Both phrases must have the same number of arguments.
- If two comparable variables appear in the same expression, unification will fail.

Unification Algorithm:

Algorithm: Unify(Ψ_1, Ψ_2)

Step. 1: If Ψ_1 **or** Ψ_2 **is** a variable **or** constant, **then**:
  a) If Ψ_1 **or** Ψ_2 are identical, **then return** NIL.
  b) Else **if** Ψ_1 **is** a variable,
    a. **then if** Ψ_1 occurs **in** Ψ_2, **then return** FAILURE
    b. Else **return** { (Ψ_2/ Ψ_1)}.
  c) Else **if** Ψ_2 **is** a variable,
    a. If Ψ_2 occurs **in** Ψ_1 **then return** FAILURE,
    b. Else **return** {(Ψ_1 / Ψ_2)}.
  d) Else **return** FAILURE.
Step.2: If the initial Predicate symbol **in** Ψ_1 **and** Ψ_2 are **not** same, **then return** FAILURE.
Step. 3: IF Ψ_1 **and** Ψ_2 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** Ψ_1.
a) Call Unify **function with** the ith element **of** Ψ_1 **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.

mplementation of the Algorithm

Step 1: Begin by making the substitute set empty.
Step 2: Unify atomic sentences in a recursive manner:
a.Check for expressions that are identical.
b.If one expression is a variable $v\Psi_i$, and the other is a term ti which does not contain variable vi, 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.
Find the most general unifier for each pair of the following atomic statements (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}
S_1 => 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))}
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}
S_1 => {p (Z, Z), p (Z, f(Z))}
SUBST θ= {f(Z) / Z}, **Unification Failed.**
Therefore, unification is not possible for these expressions.

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}
SUBST θ={f(Y) /X}

S_2 => { p(b, f(Y), f(g(b))); p(b, f(Y), f(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)}
S 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 ∧ q → k.

A. Forward Chaining

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

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

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.

- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

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

Example:

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

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

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

Facts Conversion into FOL:

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)

  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)
- Country A has some missiles. **?p Owns(A, p) ∧ Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.

  Owns(A, T1) .....(2)

  Missile(T1) .....(3)
- All of the missiles were sold to country A by Robert.

  ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) .....(4)
- Missiles are weapons.

  Missile(p) → Weapons (p) .....(5)
- Enemy of America is known as hostile.

  Enemy(p, America) →Hostile(p) .....(6)
- Country A is an enemy of America.

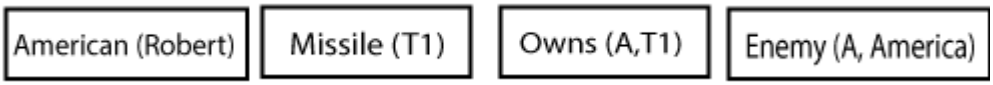
  Enemy (A, America) .....(7)
- Robert is American

  American(Robert). .....(8)

Forward chaining proof:

Step-1:

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



Step-2:

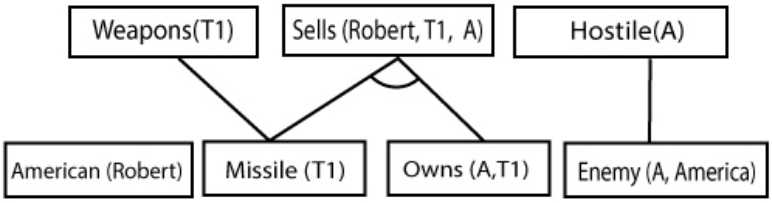
At the second step, we will see those facts which infer from available facts and with satisfied premises.

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

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

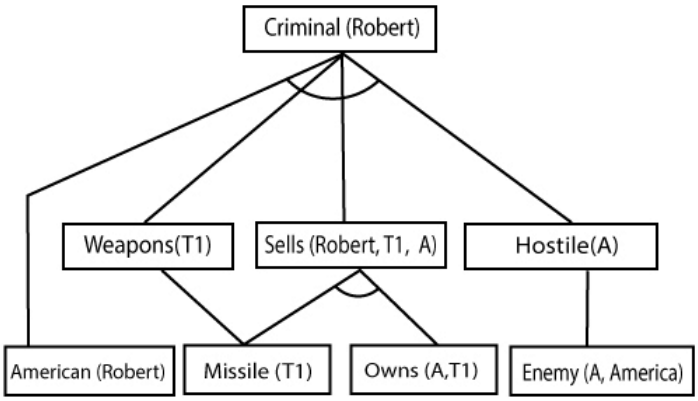
Rule-(4) satisfy with the substitution {p/T1}, **so Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

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



Step-3:

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



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

B. Backward Chaining:

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

Properties of backward chaining:

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

Example:

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

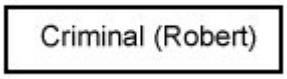
- **American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)**
  Owns(A, T1) .....(2)
- **Missile(T1)**
- **?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) .....(4)**
- **Missile(p) → Weapons (p) .....(5)**
- **Enemy(p, America) →Hostile(p) .....(6)**
- **Enemy (A, America) .....(7)**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

Backward-Chaining proof:

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

Step-1:

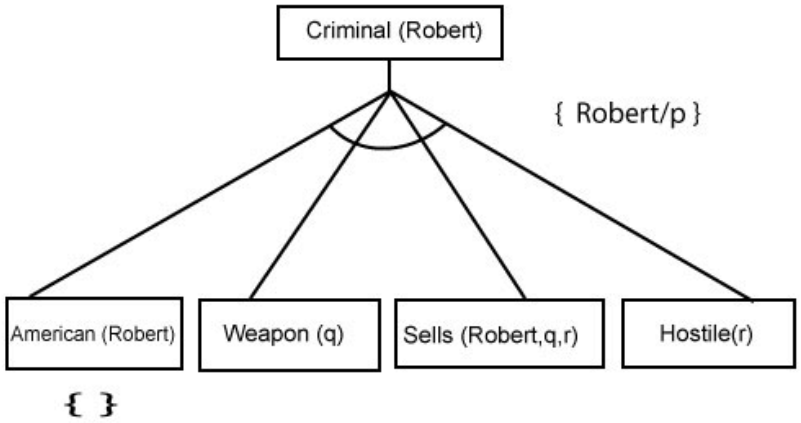
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.



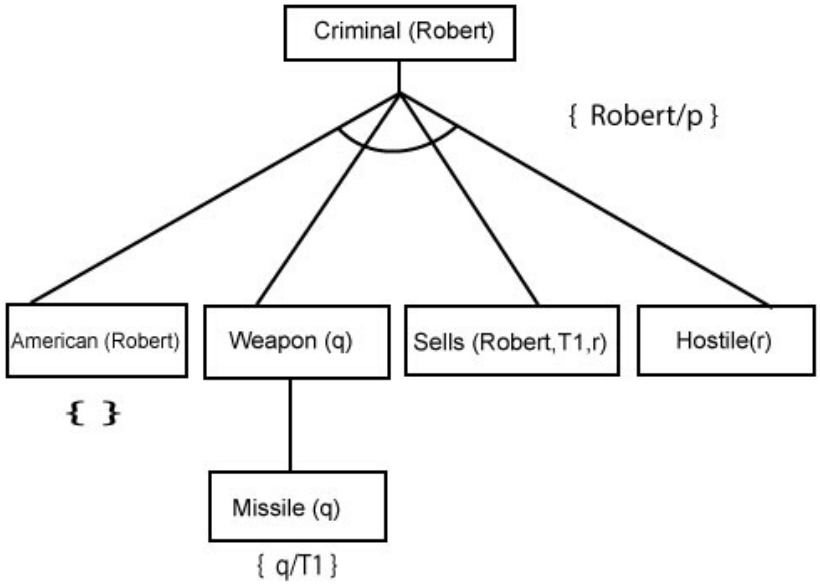
Step-2:

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

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

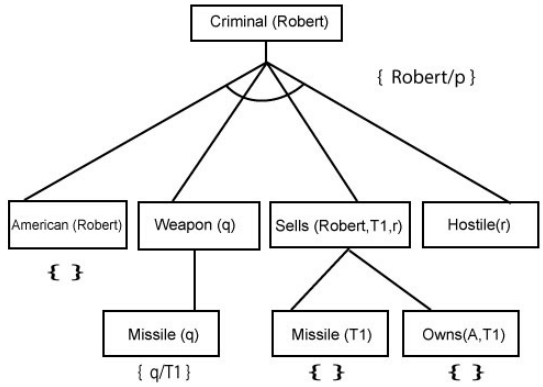


Step-3:t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



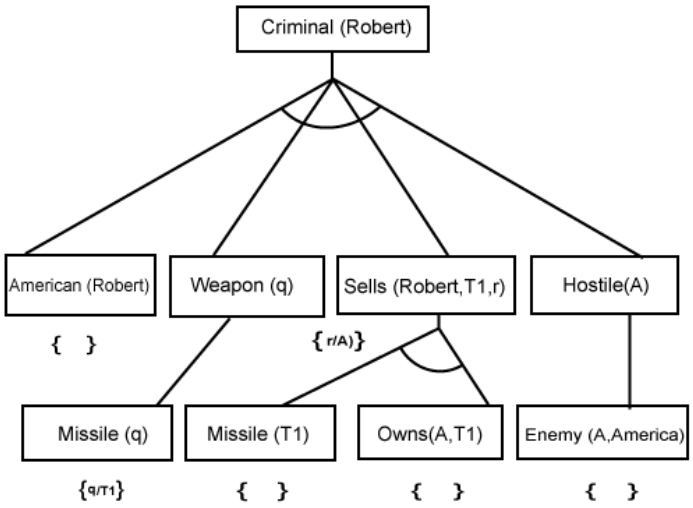
Step-4:

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



Step-5:

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



Difference between backward chaining and forward chaining

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.