Unit-1

Artificial Intelligence is composed of two words **Artificial** and **Intelligence**, where Artificial defines *"man-made,"* and intelligence defines *"thinking power"*, hence AI means *"a man-made thinking power."*

So, we can define AI as:

> "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 exists when a machine can have human based skills such as learning, reasoning, and solving problems

With Artificial Intelligence you do not need to preprogram a machine to do some work, despite that you can create a machine with programmed algorithms which can work with own intelligence, and that is the awesomeness of AI.

Why Artificial Intelligence?

- With the help of AI, you can create such software or devices which can solve real-world problems very easily and with accuracy such as health issues, marketing, traffic issues, etc.
- With the help of AI, you can create your personal virtual Assistant, such as Cortana, Google Assistant, Siri, etc.
- With the help of AI, you can build such Robots which can work in an environment where survival of humans can be at risk.
- AI opens a path for other new technologies, new devices, and new Opportunities.

Goals of Artificial Intelligence

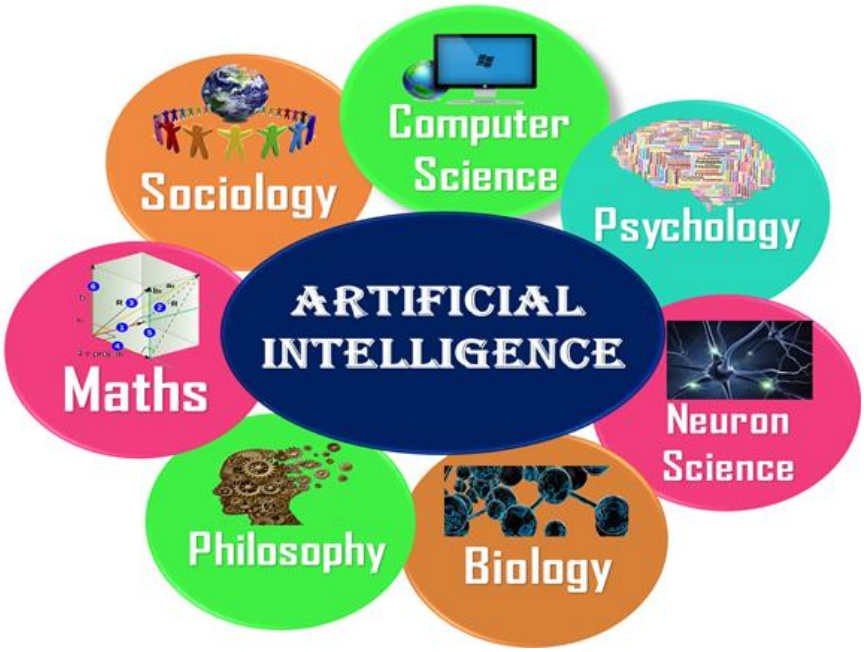
Following are the main goals of Artificial Intelligence:

1. Replicate human intelligence
2. Solve Knowledge-intensive tasks
3. An intelligent connection of perception and action
4. 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
5. Creating some system which can exhibit intelligent behavior, learn new things by itself, demonstrate, explain, and can advise to its user.

To create the AI first we should know that how intelligence is composed, so the Intelligence is an intangible part of our brain which is a combination of **Reasoning, learning, problem-solving perception, language understanding, etc**.

To achieve the above factors for a machine or software Artificial Intelligence requires the following discipline:

- Mathematics
- Biology
- Psychology
- Sociology
- Computer Science
- Neurons Study
- Statistics



Advantages of Artificial Intelligence

Following are some main advantages of Artificial Intelligence:

- **High Accuracy with less errors:** AI machines or systems are prone to less errors and high accuracy as it takes decisions as per pre-experience or information.
- **High-Speed:** AI systems can be of very high-speed and fast-decision making, because of that AI systems can beat a chess champion in the Chess game.
- **High reliability:** AI machines are highly reliable and can perform the same action multiple times with high accuracy.
- **Useful for risky areas:** AI machines can be helpful in situations such as defusing a bomb, exploring the ocean floor, where to employ a human can be risky.

- **Digital Assistant:** AI can be very useful to provide digital assistant to the users such as AI technology is currently used by various E-commerce websites to show the products as per customer requirement.
- **Useful as a public utility:** AI can be very useful for public utilities such as a self-driving car which can make our journey safer and hassle-free, facial recognition for security purpose, Natural language processing to communicate with the human in human-language, etc.

Disadvantages of Artificial Intelligence

Following are the disadvantages of AI:

- **High Cost:** The hardware and software requirement of AI is very costly as it requires lots of maintenance to meet current world requirements.
- **Can't think out of the box:** Even we are making smarter machines with AI, but still they cannot work out of the box, as the robot will only do that work for which they are trained, or programmed.
- **No feelings and emotions:** AI machines can be an outstanding performer, but still it does not have the feeling so it cannot make any kind of emotional attachment with human, and may sometime be harmful for users if the proper care is not taken.
- **Increase dependency on machines:** With the increment of technology, people are getting more dependent on devices and hence they are losing their mental capabilities.
- **No Original Creativity:** As humans are so creative and can imagine some new ideas but still AI machines cannot beat this power of human intelligence and cannot be creative and imaginative.

Application of AI

1. AI in Astronomy

- Artificial Intelligence can be very useful to solve complex universe problems. AI technology can be helpful for understanding the universe such as how it works, origin, etc.

2. AI in Healthcare

- In the last, five to ten years, AI becoming more advantageous for the healthcare industry and going to have a significant impact on this industry.
- Healthcare Industries are applying AI to make a better and faster diagnosis than humans. AI can help doctors with diagnoses and can inform when patients are worsening so that medical help can reach to the patient before hospitalization.

3. AI in Gaming

- AI can be used for gaming purpose. The AI machines can play strategic games like chess, where the machine needs to think of a large number of possible places.

4. AI in Finance

- AI and finance industries are the best matches for each other. The finance industry is implementing automation, chatbot, adaptive intelligence, algorithm trading, and machine learning into financial processes.

5. AI in Data Security

- The security of data is crucial for every company and cyber-attacks are growing very rapidly in the digital world. AI can be used to make your data more safe and secure. Some examples such as AEG bot, AI2 Platform,are used to determine software bug and cyber-attacks in a better way.

6. AI in Social Media

- Social Media sites such as Facebook, Twitter, and Snapchat contain billions of user profiles, which need to be stored and managed in a very efficient way. AI can organize and manage massive amounts of data. AI can analyze lots of data to identify the latest trends, hashtag, and requirement of different users.

7. AI in Travel & Transport

- AI is becoming highly demanding for travel industries. AI is capable of doing various travel related works such as from making travel arrangement to suggesting the hotels, flights, and best routes to the customers. Travel industries are using AI-powered chatbots which can make human-like interaction with customers for better and fast response.

8. AI in Automotive Industry

- Some Automotive industries are using AI to provide virtual assistant to their user for better performance. Such as Tesla has introduced TeslaBot, an intelligent virtual assistant.
- Various Industries are currently working for developing self-driven cars which can make your journey more safe and secure.

9. AI in Robotics:

- Artificial Intelligence has a remarkable role in Robotics. Usually, general robots are programmed such that they can perform some repetitive task, but with the help of AI, we can create intelligent robots which can perform tasks with their own experiences without pre-programmed.
- Humanoid Robots are best examples for AI in robotics, recently the intelligent Humanoid robot named as Erica and Sophia has been developed which can talk and behave like humans.

10. AI in Entertainment

- We are currently using some AI based applications in our daily life with some entertainment services such as Netflix or Amazon. With the help of ML/AI algorithms, these services show the recommendations for programs or shows.

11. AI in Agriculture

- Agriculture is an area which requires various resources, labor, money, and time for best result. Now a day's agriculture is becoming digital, and AI is emerging in this field. Agriculture is applying AI as agriculture robotics, solid and crop monitoring, predictive analysis. AI in agriculture can be very helpful for farmers.

12. AI in E-commerce

- AI is providing a competitive edge to the e-commerce industry, and it is becoming more demanding in the e-commerce business. AI is helping shoppers to discover associated products with recommended size, color, or even brand.

13. AI in education:

- AI can automate grading so that the tutor can have more time to teach. AI chatbot can communicate with students as a teaching assistant.
- AI in the future can be work as a personal virtual tutor for students, which will be accessible easily at any time and any place.

History of Artificial Intelligence

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."

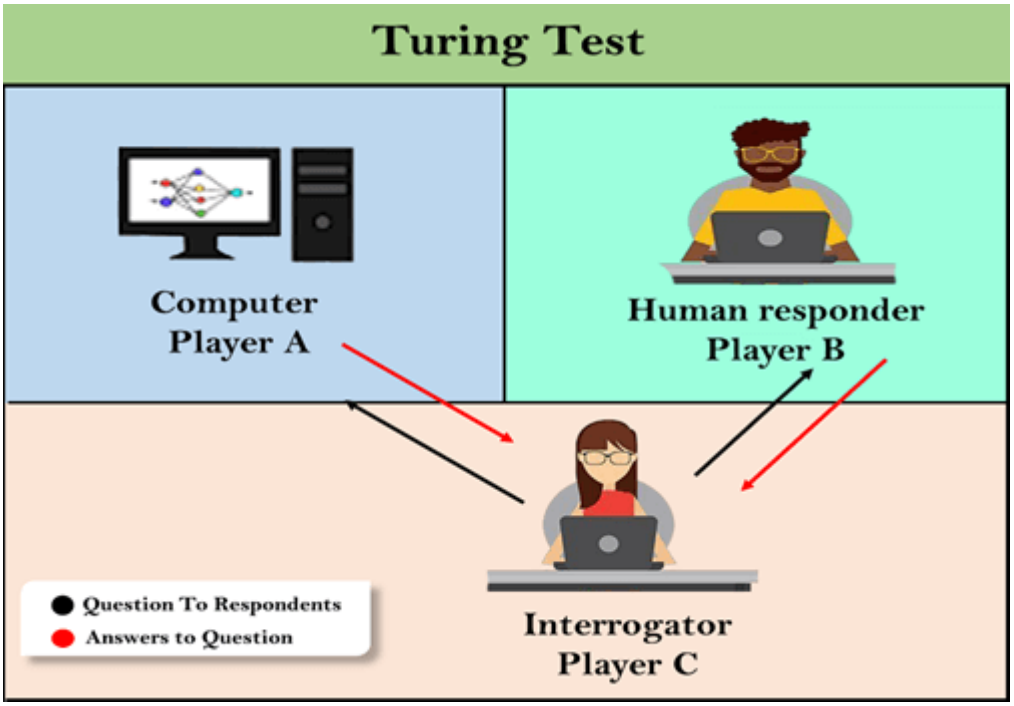
- **Year 2018:** The "Project Debater" from IBM debated on complex topics with two master debaters and also performed extremely well.
- Google has demonstrated an AI program "Duplex" which was a virtual assistant and which had taken hairdresser appointment on call, and lady on other side didn't notice 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.

Turing Test in AI

In 1950, Alan Turing introduced a test to check whether a machine can think like a human or not, this test is known as the Turing Test. In this test, Turing proposed that the computer can be said to be an intelligent if it can mimic human response under specific conditions.

Turing Test was introduced by Turing in his 1950 paper, "Computing Machinery and Intelligence," which considered the question, "Can Machine think?"



The Turing test is based on a party game "Imitation game," with some modifications. This game involves three players in which one player is Computer, another player is human responder, and the third player is a human Interrogator, who is isolated from other two players and his job is to find that which player is machine among two of them.

Consider, Player A is a computer, Player B is human, and Player C is an interrogator. Interrogator is aware that one of them is machine, but he needs to identify this on the basis of questions and their responses.

The conversation between all players is via keyboard and screen so the result would not depend on the machine's ability to convert words as speech.

The test result does not depend on each correct answer, but only how closely its responses like a human answer. The computer is permitted to do everything possible to force a wrong identification by the interrogator.

The questions and answers can be like:

Interrogator: Are you a computer?

PlayerA (Computer): No

Interrogator: Multiply two large numbers such as (256896489*456725896)

Player A: Long pause and give the wrong answer.

In this game, if an interrogator would not be able to identify which is a machine and which is human, then the computer passes the test successfully, and the machine is said to be intelligent and can think like a human.

"In 1991, the New York businessman Hugh Loebner announces the prize competition, offering a $100,000 prize for the first computer to pass the Turing test. However, no AI program to till date, come close to passing an undiluted Turing test".

Chatbots to attempt the Turing test:

ELIZA: ELIZA was a Natural language processing computer program created by Joseph Weizenbaum. It was created to demonstrate the ability of communication between machine and humans. It was one of the first chatterbots, which has attempted the Turing Test.

Parry: Parry was a chatterbot created by Kenneth Colby in 1972. Parry was designed to simulate a person with **Paranoid schizophrenia**(most common chronic mental disorder). Parry was described as "ELIZA with attitude." Parry was tested using a variation of the Turing Test in the early 1970s.

Eugene Goostman: Eugene Goostman was a chatbot developed in Saint Petersburg in 2001. This bot has competed in the various number of Turing Test. In June 2012, at an event, Goostman won the competition promoted as largest-ever Turing test content, in which it has convinced 29% of judges that it was a human.Goostman resembled as a 13-year old virtual boy.

The Chinese Room Argument:

There were many philosophers who really disagreed with the complete concept of Artificial Intelligence. The most famous argument in this list was "**Chinese Room**."

In the year **1980, John Searle** presented "**Chinese Room**" thought experiment, in his paper "**Mind, Brains, and Program**," which was against the validity of Turing's Test. According

to his argument, "**Programming a computer may make it to understand a language, but it will not produce a real understanding of language or consciousness in a computer**."

He argued that Machine such as ELIZA and Parry could easily pass the Turing test by manipulating keywords and symbol, but they had no real understanding of language. So it cannot be described as "thinking" capability of a machine such as a human.

Features required for a machine to pass the Turing test:

- **Natural language processing:** NLP is required to communicate with Interrogator in general human language like English.
- **Knowledge representation:** To store and retrieve information during the test.
- **Automated reasoning:** To use the previously stored information for answering the questions.
- **Machine learning:** To adapt new changes and can detect generalized patterns.
- **Vision (For total Turing test):** To recognize the interrogator actions and other objects during a test.
- **Motor Control (For total Turing test):** To act upon objects if requested3.

Cognitive science is the **interdisciplinary, scientific study of the mind and its processes** with input from linguistics, psychology, neuroscience, philosophy, computer science/artificial intelligence, and anthropology. It examines the nature, the tasks, and the functions of cognition (in a broad sense).

The term "***Artificial Intelligence***" refers to the simulation of human intelligence processes by machines, especially **computer systems**. It also includes **Expert systems, voice recognition, machine vision,** and **natural language processing (NLP).**

AI programming focuses on three cognitive aspects, such as learning, reasoning, and self-correction.

- Learning Processes
- Reasoning Processes
- Self-correction Processes

Learning Processes

This part of AI programming is concerned with gathering data and creating rules for transforming it into useful information. The rules, which are also called algorithms, offer computing devices with step-by-step instructions for accomplishing a particular job.

Reasoning Processes

This part of AI programming is concerned with selecting the best algorithm to achieve the desired result.

Self-Correction Processes

This part of AI programming aims to fine-tune algorithms regularly in order to ensure that they offer the most reliable results possible.

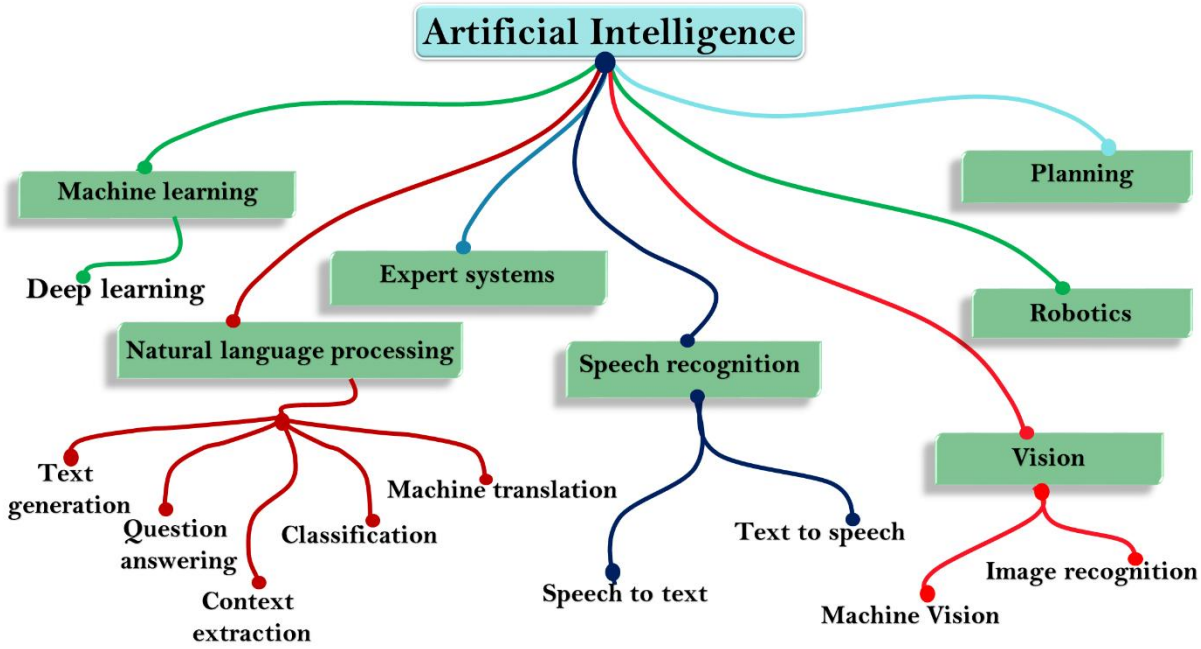
We can define AI as, "*Artificial Intelligence is a branch of computer science that deals with developing intelligent machines which can behave like human, think like human, and has ability to take decisions by their own***."**

Examples of AI-Artificial Intelligence

The following are the examples of AI-Artificial Intelligence:

1. Google Maps and Ride-Hailing Applications
2. Face Detection and recognition
3. Text Editors and Autocorrect
4. Chatbots
5. E-Payments
6. Search and Recommendation algorithms
7. Digital Assistant
8. Social media
9. Healthcare
10. Gaming
11. Online Ads-Network
12. Banking and Finance
13. Smart Home devices

14. Security and Surveillance
15. Smart Keyboard App
16. Smart Speaker
17. E-Commerce
18. Smart Email Apps
19. Music and Media Streaming Service
20. Space Exploration



`

A syllogism is **a form of deductive argument where the conclusion follows from the truth of two (or more) premises**. A deductive argument moves from the general to the specific and opposes inductive arguments that move from the specific to the general:1. All mammals are animals.

An example of a syllogism is "**All mammals are animals**. All elephants are mammals. Therefore, all elephants are animals." In a syllogism, the more general premise is called the major premise ("All mammals are animals").

Agents in Artificial Intelligence

Artificial intelligence is defined as the study of rational agents. A rational agent could be anything that makes decisions, as a person, firm, machine, or software. It carries out an action with the best outcome after considering past and current percepts(agent's perceptual inputs at a given instance). An AI system is composed of an **agent and its environment**. The agents act in their environment. The environment may contain other agents.
An agent is anything that can be viewed as :

- perceiving its environment through **sensors** and
- acting upon that environment through **actuators**

Agents in Artificial Intelligence

An AI system can be defined as the study of the rational agent and its environment. The agents sense the environment through sensors and act on their environment through actuators. An AI agent can have mental properties such as knowledge, belief, intention, etc.

What is an Agent?

An agent can be anything that perceive its 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.

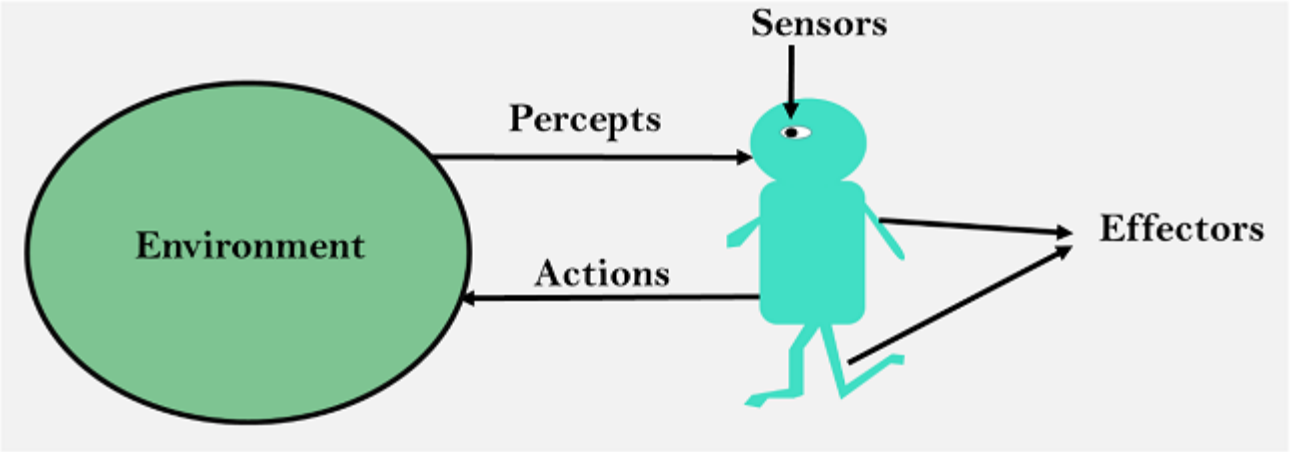
Hence the world around us is full of agents such as thermostat, cellphone, camera, and even we are also agents.

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.



Intelligent Agents:

An intelligent agent is an autonomous entity which act upon an environment using sensors and actuators for achieving goals. An intelligent agent may learn from the environment to achieve their goals. A thermostat is an example of an intelligent agent.

Following are the main four rules for an AI agent:

- **Rule 1:** An AI agent must have the ability to perceive the environment.
- **Rule 2:** The observation must be used to make decisions.
- **Rule 3:** Decision should result in an action.
- **Rule 4:** The action taken by an AI agent must be a rational action.

Rational Agent:

A rational agent is an agent which has clear preference, models uncertainty, and acts in a way to maximize its performance measure with all possible actions.

A rational agent is said to perform the right things. AI is about creating rational agents to use for game theory and decision theory for various real-world scenarios.

For an AI agent, the rational action is most important because in AI reinforcement learning algorithm, for each best possible action, agent gets the positive reward and for each wrong action, an agent gets a negative reward.

Rationality:

The rationality of an agent is measured by its performance measure. Rationality can be judged on the basis of following points:

- Performance measure which defines the success criterion.
- Agent prior knowledge of its environment.

- Best possible actions that an agent can perform.
- The sequence of percepts.

> *Note: Rationality differs from Omniscience because an Omniscient agent knows the actual outcome of its action and act accordingly, which is not possible in reality.*

Structure of an AI Agent

The task of AI is to design an agent program which implements the agent function. The structure of an intelligent agent is a combination of architecture and agent program. It can be viewed as:

1. Agent = Architecture + Agent program

Following are the main three terms involved in the structure of an AI agent:

Architecture: Architecture is machinery that an AI agent executes on.

Agent Function: Agent function is used to map a percept to an action.

1. $f:P^* \rightarrow A$

Agent program: Agent program is an implementation of agent function. An agent program executes on the physical architecture to produce function f.

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.

Example of Agents with their PEAS representation

Agent	Performance measure	Environment	Actuators	Sensors
1. Medical Diagnose	○ Healthy patient ○ Minimized cost	○ Patient ○ Hospital ○ Staff	○ Tests ○ Treatments	Keyboard (Entry of symptoms)
2. Vacuum Cleaner	○ Cleanness ○ Efficiency ○ Battery life ○ Security	○ Room ○ Table ○ Wood floor ○ Carpet ○ Various obstacles	○ Wheels ○ Brushes ○ Vacuum Extractor	○ Camera ○ Dirt detection sensor ○ Cliff sensor ○ Bump Sensor ○ Infrared Wall Sensor
3. Part - picking Robot	○ Percentage of parts in correct bins.	○ Conveyor belt with parts, ○ Bins	○ Jointed Arms ○ Hand	○ Camera ○ Joint angle sensors.

`

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.

Types of AI Agents

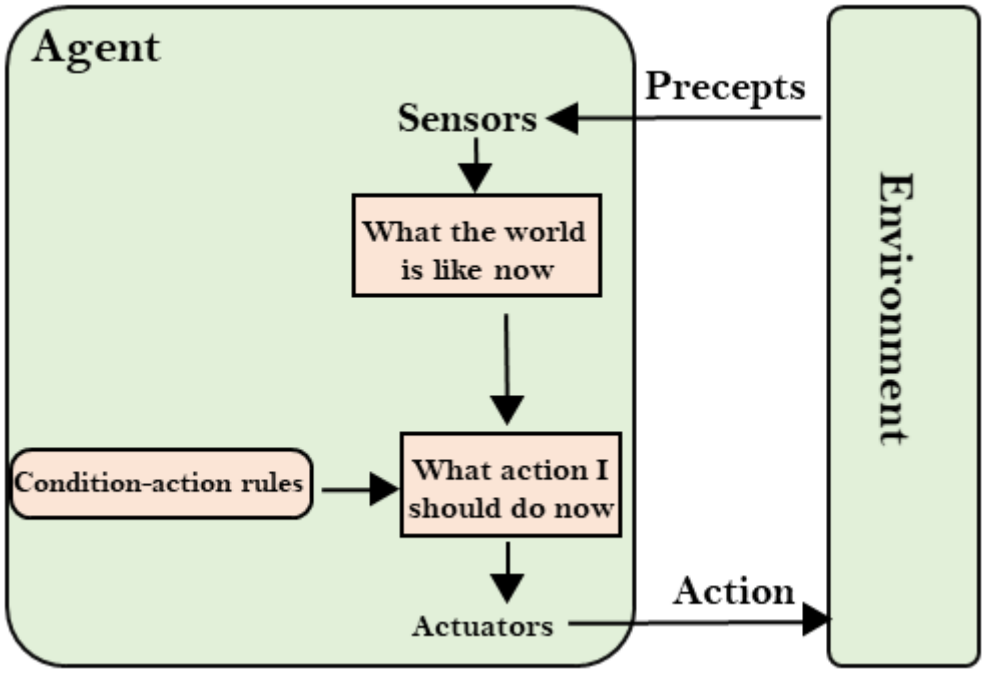
Agents can be grouped into five classes based on their degree of perceived intelligence and capability. All these agents can improve their performance and generate better action over the time. These are given below:

- Simple Reflex Agent
- Model-based reflex agent
- Goal-based agents
- Utility-based agent
- Learning agent

1. Simple Reflex agent:

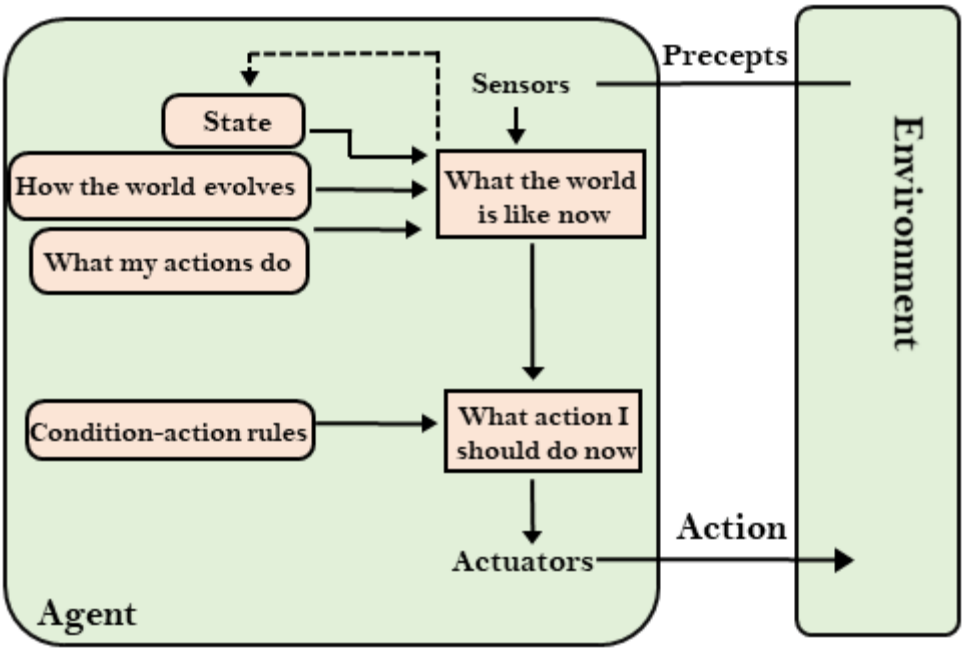
- The Simple reflex agents are the simplest agents. These agents take decisions on the basis of the current percepts and ignore the rest of the percept history.
- These agents only succeed in the fully observable environment.
- The Simple reflex agent does not consider any part of percepts history during their decision and action process.

- The Simple reflex agent works on Condition-action rule, which means it maps the current state to action. Such as a Room Cleaner agent, it works only if there is dirt in the room.
- Problems for the simple reflex agent design approach:
    - They have very limited intelligence
    - They do not have knowledge of non-perceptual parts of the current state
    - Mostly too big to generate and to store.
    - Not adaptive to changes in the environment.



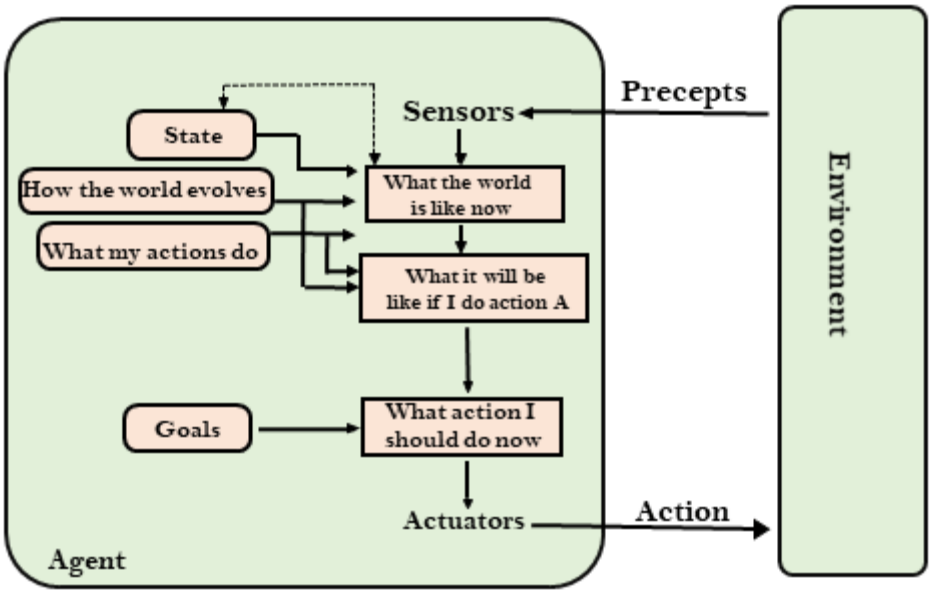
2. Model-based reflex agent

- The Model-based agent can work in a partially observable environment, and track the situation.
- A model-based agent has two important factors:
    - **Model:** It is knowledge about "how things happen in the world," so it is called a Model-based agent.
    - **Internal State:** It is a representation of the current state based on percept history.
- These agents have the model, "which is knowledge of the world" and based on the model they perform actions.
- Updating the agent state requires information about:
    a. How the world evolves
    b. How the agent's action affects the world.



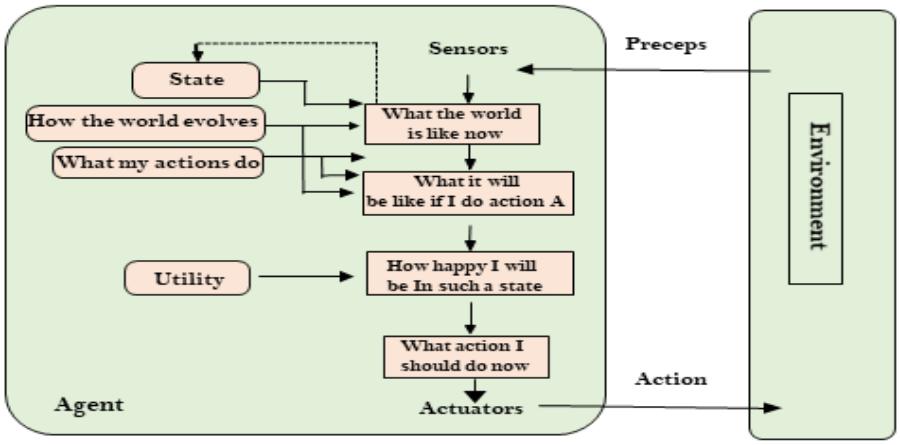
3. Goal-based agents

- The knowledge of the current state environment is not always sufficient to decide for an agent to what to do.
- The agent needs to know its goal which describes desirable situations.
- Goal-based agents expand the capabilities of the model-based agent by having the "goal" information.
- They choose an action, so that they can achieve the goal.
- These agents may have to consider a long sequence of possible actions before deciding whether the goal is achieved or not. Such considerations of different scenario are called searching and planning, which makes an agent proactive.



4. Utility-based agents

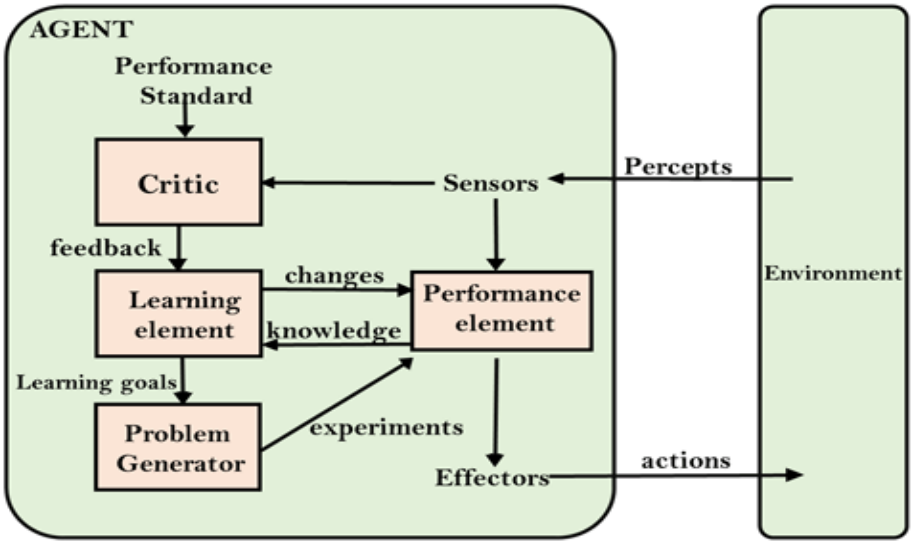
- These agents are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.
- Utility-based agent act based not only goals but also the best way to achieve the goal.
- The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.
- The utility function maps each state to a real number to check how efficiently each action achieves the goals.



5. Learning Agents

- A learning agent in AI is the type of agent which can learn from its past experiences, or it has learning capabilities.
- It starts to act with basic knowledge and then able to act and adapt automatically through learning.
- A learning agent has mainly four conceptual components, which are:
    a. **Learning element:** It is responsible for making improvements by learning from environment
    b. **Critic:** Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.
    c. **Performance element:** It is responsible for selecting external action
    d. **Problem generator:** This component is responsible for suggesting actions that will lead to new and informative experiences.

Hence, learning agents are able to learn, analyze performance, and look for new ways to improve the performance.



Unit-2

knowledge representation

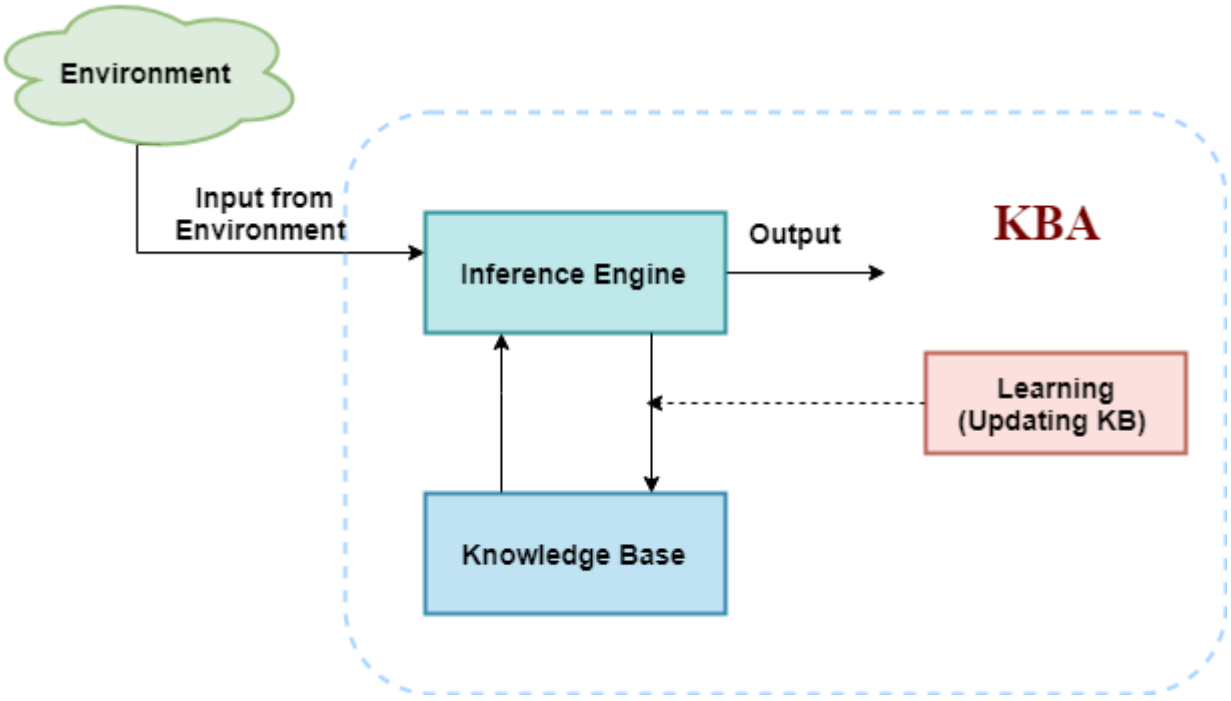
Knowledge-Based Agent in Artificial intelligence

- 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. **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:

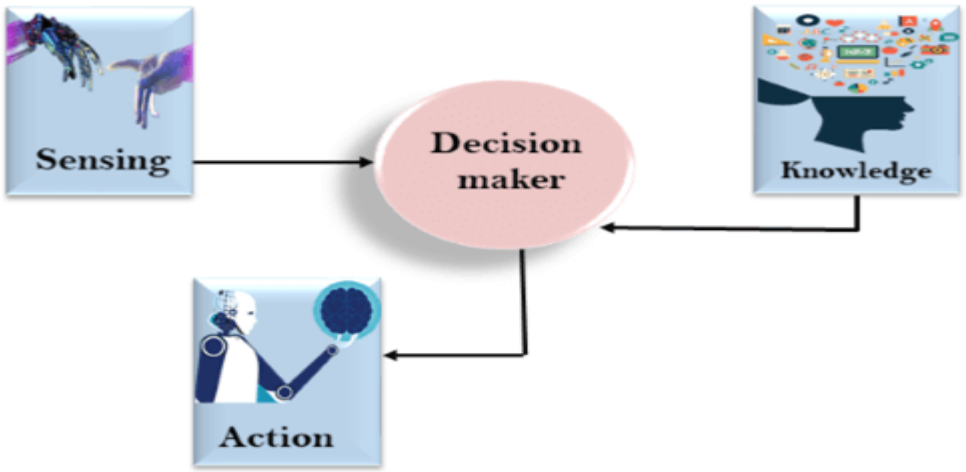
- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

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.

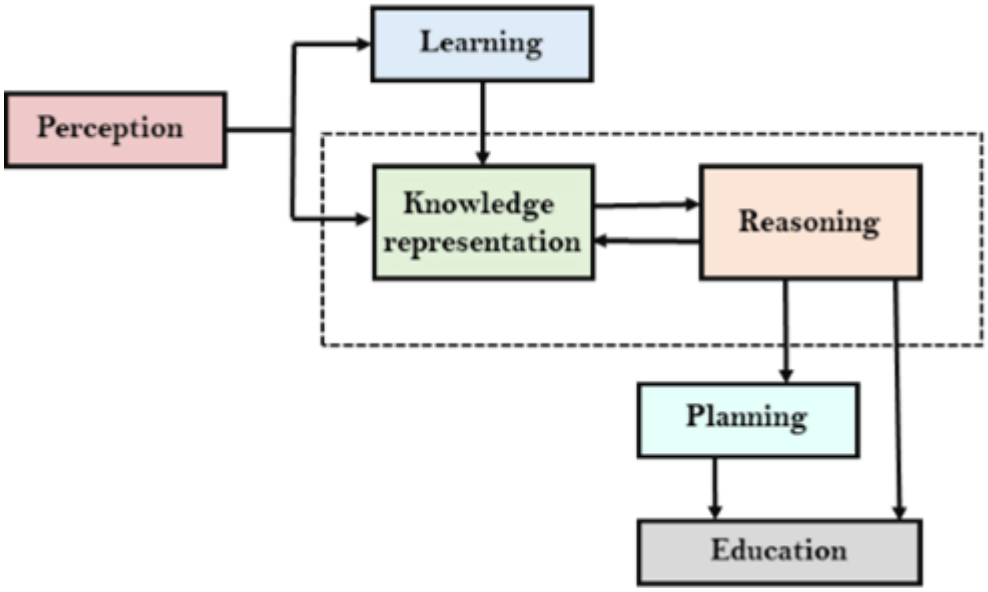
As we can see in below diagram, there is one decision maker which act by sensing the environment and using knowledge. But if the knowledge part will not present then, it cannot display intelligent behavior.



AI knowledge cycle:

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

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



The above diagram 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 givenbelow:

1. Simple relational knowledge:

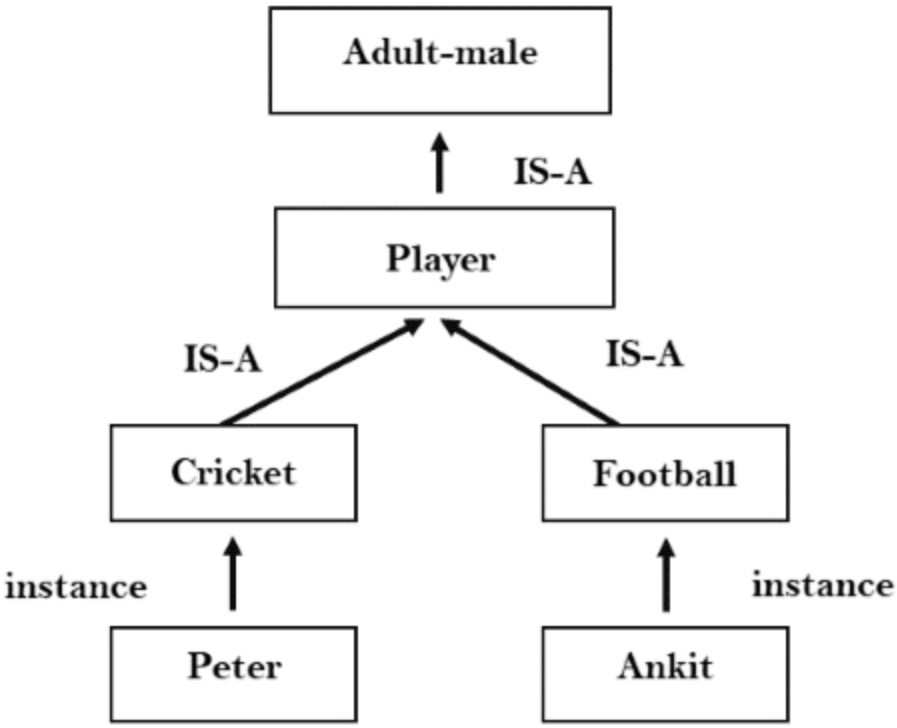
- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This 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:
    a. Marcus is a man
    b. All men are mortal
    Then it can represent as;

    man(Marcus)
    ∀x = man (x) ----------> mortal (x)s

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**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

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**.
- 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:

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:

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 ¬A∨ B and A→B, are identical hence A is Equivalent to B

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

Properties of Operators:

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

Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic.
  Example:
  a. **All the girls are intelligent.**
  b. **Some apples are sweet.**

  Propositional logic has limited expressive power.

  In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

Rules of Inference in Artificial intelligence

Inference:

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

Inference rules:

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

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

- **Implication:** It is one of the logical connectives which can be represented as P → 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: $\frac{P \rightarrow Q, \quad \sim Q}{\sim P}$

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

Proof by Truth table:

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

3. Hypothetical Syllogism:

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

Example:

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

Proof by truth table:

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

4. Disjunctive Syllogism:

The Disjunctive syllogism rule state that if P∨Q 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. ==>P∨Q
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 P∨Q 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. ==> (P∨Q)

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:

Following are the basic elements of FOL syntax:

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

Atomic sentences:

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

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

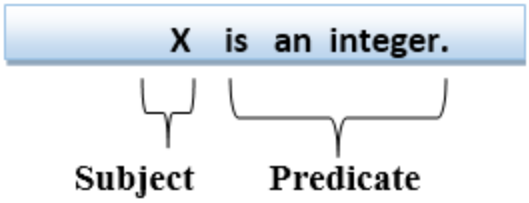
Complex Sentences:

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

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

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

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



Quantifiers in First-order logic:

- 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 "→".*

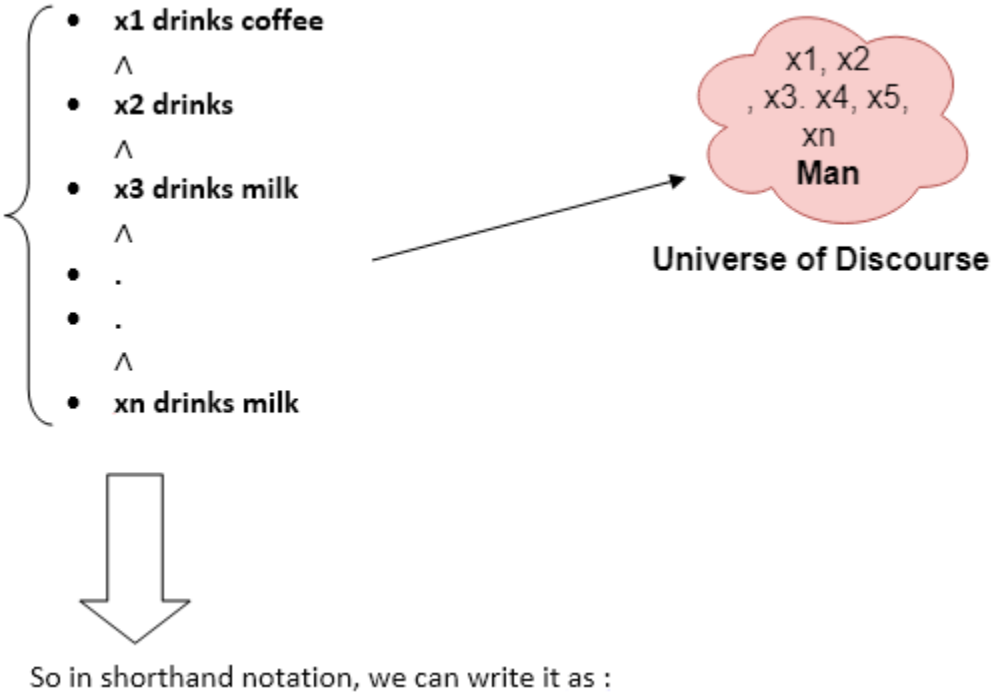
If x is a variable, then ∀x is read as:

- **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:



∀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.

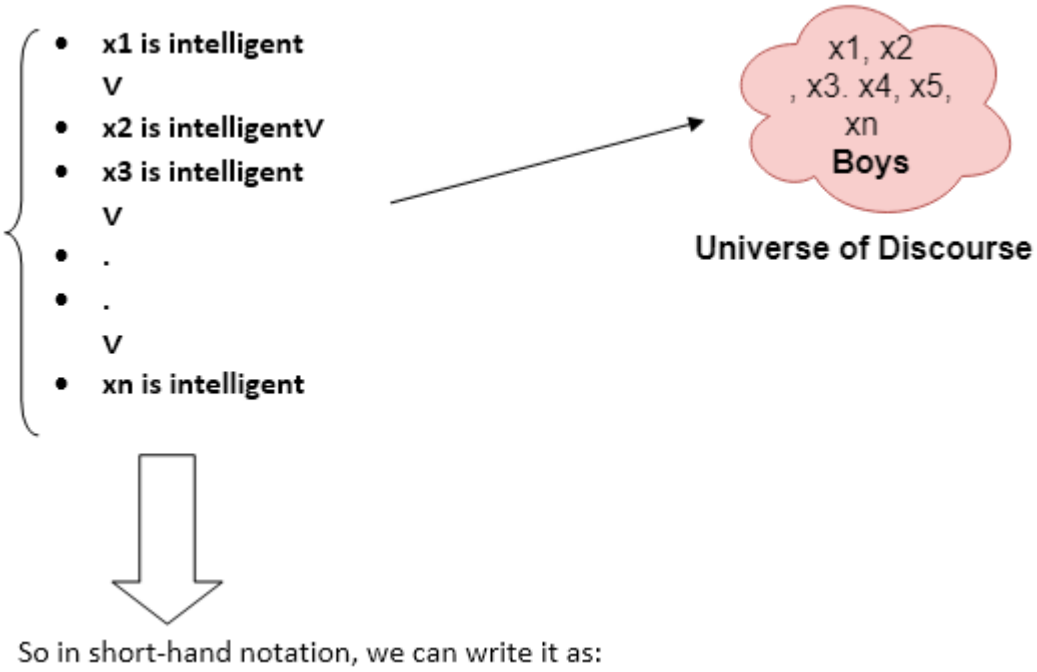
Note: In Existential quantifier we always use AND or Conjunction symbol (∧).

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)].

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**".

> *Note: First-order logic is capable of expressing facts about some or all objects in the universe.*

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: $\frac{\exists x\, P(x)}{P(c)}$

4. Existential introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the property P.
- It can be represented as: $\frac{P(c)}{\exists x P(x)}$

- **Example: Let's say that,**
  "Priyanka got good marks in English."
  "Therefore, someone got good marks in English."

What is Unification?

- Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
- It takes two literals as input and makes them identical using substitution.
- Let L_1 and L_2 be two atomic sentences and σ be a unifier such that, $\mathbf{L_1\sigma = L_2\sigma}$, then it can be expressed as **UNIFY($\mathbf{L_1}$, $\mathbf{L_2}$)**.
- **Example: Find the MGU for Unify{King(x), King(John)}**

Let L_1 = King(x), L_2 = King(John),

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

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

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

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

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

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

Conditions for Unification:

Following are some basic conditions for unification:

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
- Number of Arguments in both expressions must be identical.

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

Unification Algorithm:

Algorithm: Unify(L_1, L_2)

Step. 1: If L_1 or L_2 is a variable or constant, then:
a) If L_1 or L_2 are identical, then return NIL.
b) Else if $L_1$is a variable,
a. then if L_1 occurs in L_2, then return FAILURE
b. Else return {(L_2/ L_1)}.
c) Else if L_2 is a variable,
a. If L_2 occurs in L_1 then return FAILURE,
b. Else return {(L_1/ L_2)}.
d) Else return FAILURE.
Step.2: If the initial Predicate symbol in L_1 and L_2 are not same, then return FAILURE.
Step. 3: IF L_1 and L_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 L_1.
a) Call Unify function with the ith element of L_1 and ith element of L_2, and put the result into S.
b) If S = failure then returns Failure
c) If S ≠ NIL then do,
a. Apply S to the remainder of both L1 and L2.
b. SUBST= APPEND(S, SUBST).
Step.6: Return SUBST.

Implementation of the Algorithm

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

Step.2: Recursively unify atomic sentences:

a. Check for Identical expression match.
b. If one expression is a variable v_i, and the other is a term t_i which does not contain variable v_i, then:
a. Substitute t_i / v_i in the existing substitutions
b. Add t_i /v_i to the substitution setlist.
c. If both the expressions are functions, then function name must be similar, and the number of arguments must be the same in both the expression.

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

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

Sol: S_0 => Here, L_1 = p(f(a), g(Y)), and L_2 = p(X, X)
SUBST θ= {f(a) / X}
S1 => L_1 = p(f(a), g(Y)), and L_2 = p(f(a), f(a))
SUBST θ= {f(a) / g(y)}, **Unification failed**.

Unification is not possible for these expressions.

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

Here, L_1 = p(b, X, f(g(Z))) , and L_2 = p(Z, f(Y), f(Y))
S_0 => { p(b, X, f(g(Z))); p(Z, f(Y), f(Y))}
SUBST θ={b/Z}

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

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

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

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

Here, L_1 = {p (X, X), and L_2 = p (Z, f(Z))
S_0 => {p (X, X), p (Z, f(Z))}
SUBST θ= {X/Z}
S1 => {p (Z, Z), p (Z, f(Z))}
SUBST θ= {f(Z) / Z}, **Unification Failed**.

Hence, unification is not possible for these expressions.

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

Here, L_1 = {prime(11) , and L_2 = prime(y)}
S_0 => {prime(11) , prime(y)}
SUBST θ= {11/y}

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

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

Here, L_1 = Q(a, g(x, a), f(y)), and L_2 = Q(a, g(f(b), a), x)
S_0 => {Q(a, g(x, a), f(y)); Q(a, g(f(b), a), x)}
SUBST θ= {f(b)/x}
S_1 => {Q(a, g(f(b), a), f(y)); Q(a, g(f(b), a), f(b))}

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

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

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

Here, L_1 = knows(Richard, x), and L_2 = knows(Richard, John)
S_0 => { knows(Richard, x); knows(Richard, John)}
SUBST θ= {John/x}
S_1 => { knows(Richard, John); knows(Richard, John)}, **Successfully Unified.**
Unifier: {John/x}.

Clausal Form in Deductive Databases

In **clausal form**, the formula is made up of a number of clauses, where each **clause** is composed of a number of *literals* connected by OR logical connectives only.
A formula can have the following *quantifiers*:

1. **Universal quantifier –**
   It can be understood as – "For all x, P(x) holds", meaning P(x) is true for every object x in the universe.
   Example: All trucks has wheels.

2. **Existential quantifier –**
   It can be understood as – "There exists an x such that P(x)", meaning P(x) is true for at least one object x of the universe.
   Example: Someone cares for you.

A **clausal form** formula must be transformed into another formula with the following characteristics :

- All variables in the formula are universally quantified. Hence, it is not necessary to include the universal quantifiers explicitly for all. The quantifiers are removed, and all variables in the formula are implicitly quantified by the universal quantifier.

- As the formula is made up of a number of clauses, and each clause is composed of a number of literals connected by OR logical connectives only. Hence, each clause is a *disjunction* of literals.

- To form a formula, the clauses themselves are connected by AND logical connectives only. Hence, clausal form of a formula is a *conjunction of clauses*.

Any formula can be converted into clausal form.

Example :
Literals can be positive literals or negative literals. For the forms of the individual clauses where each of is a disjunction of literals. For the clause form:

NOT(P1) *OR* NOT(P2) *OR* ..... *OR* NOT(Pn) *OR* Q1 *OR* Q2 *OR* ..... *OR* Qm

The above clause has **n** negative literals and **m** positive literals. This clause can be transformed into the following equivalent logical formula:

P1 *AND* P2 *AND* ..... *AND* Pn => Q1 *OR* Q2 *OR* ..... *OR* Qm
where '=>' is the implies symbol.

If all the p literals (i = 1, 2, …,) are true, the 2nd formula is true only if at least one of the Q's is true, which is the meaning of the (implies) symbol. For 1st formula, if all the P literals (i = 1, 2, …, n) are true, their negations are all false; so in this case it is true only if at least one of the Q's is true.

Thus the above two formulas are equivalent, hence their truth values are always the same.

Resolution in FOL

Resolution

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

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

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

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

The resolution inference rule:

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

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

Example:

We can resolve two clauses which are given below:

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

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

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

[Animal (g(x) V ¬ Kills(f(x), x)].

Resolution is one kind of proof technique that works this way - (i) select two clauses that contain conflicting terms (ii) combine those two clauses and (iii) cancel out the conflicting terms.

Steps for Resolution:

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

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

Example:

a. **John likes all kind of food.**
b. **Apple and vegetable are food**
c. **Anything anyone eats and not killed is food.**
d. **Anil eats peanuts and still alive**
e. **Harry eats everything that Anil eats.**
   Prove by resolution that:
f. **John likes peanuts.**

Step-1: Conversion of Facts into FOL

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

a. ∀x: food(x) → likes(John, x)
b. food(Apple) ∧ food(vegetables)
c. ∀x ∀y: eats(x, y) ∧ ¬ killed(x) → food(y)
d. eats (Anil, Peanuts) ∧ alive(Anil).
e. ∀x : eats(Anil, x) → eats(Harry, x)
f. ∀x: ¬ killed(x) → alive(x) } **added predicates.**
g. ∀x: alive(x) →¬ killed(x) }
h. likes(John, Peanuts)

John likes all kind of food.

Apple and vegetable are food

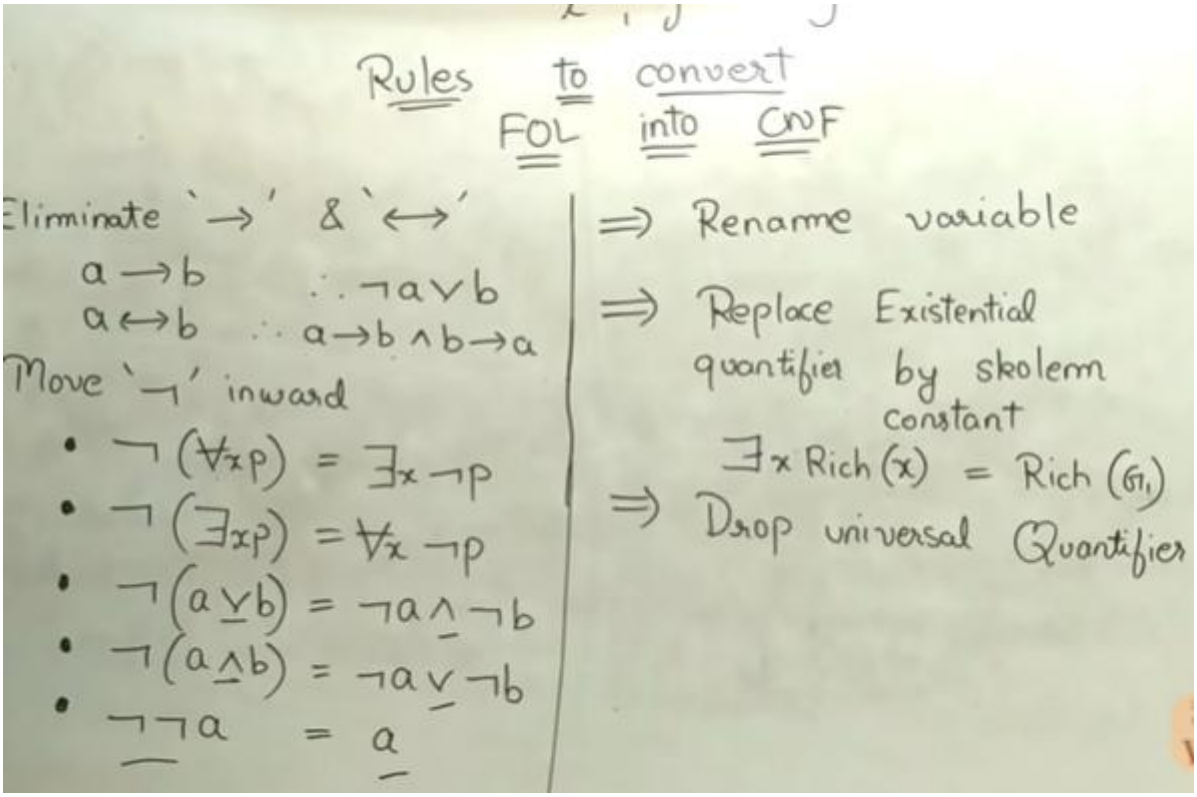
Anything(Y) anyone(X) eats and not killed is food.

Anil eats peanuts and still alive

Harry eats everything that Anil eats.
John likes peanuts.

Step-2: Conversion of FOL into CNF

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



- **Eliminate all implication (→) and rewrite: negate(a)V b**
    a. ∀x ¬ food(x) V likes(John, x)
    b. food(Apple) Λ food(vegetables)
    c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
    d. eats (Anil, Peanuts) Λ alive(Anil)
    e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
    f. ∀x¬ [¬ killed(x)] V alive(x)
    g. ∀x ¬ alive(x) V ¬ killed(x)

a. ∀x: food(x) → likes(John, x)
b. food(Apple) Λ food(vegetables)
c. ∀x ∀y: eats(x, y) Λ ¬ killed(x) → food(y)
d. eats (Anil, Peanuts) Λ alive(Anil).
e. ∀x : eats(Anil, x) → eats(Harry, x)
f. ∀x: ¬ killed(x) → alive(x) } **added predicates.**
g. ∀x: alive(x) →¬ killed(x) }
h. likes(John, Peanuts)

h. likes(John, Peanuts).

Move negation (¬)inwards and rewrite

a. ∀x ¬ food(x) V likes(John, x)
b. food(Apple) Λ food(vegetables)
c. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
d. eats (Anil, Peanuts) Λ alive(Anil)
e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
f. ∀x killed(x) V alive(x)
g. ∀x ¬ alive(x) V ¬ killed(x)
h. likes(John, Peanuts).

Rename variables or standardize variables

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

Eliminate existential instantiation quantifier by elimination.

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

Drop Universal quantifiers.

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

a. ¬ food(x) V likes(John, x)
b. food(Apple)
c. food(vegetables)
d. ¬ eats(y, z) V killed(y) V food(z)
e. eats (Anil, Peanuts)
f. alive(Anil)
g. ¬ eats(Anil, w) V eats(Harry, w)
h. killed(g) V alive(g)
i. ¬ alive(k) V ¬ killed(k)

j. likes(John, Peanuts).

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

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

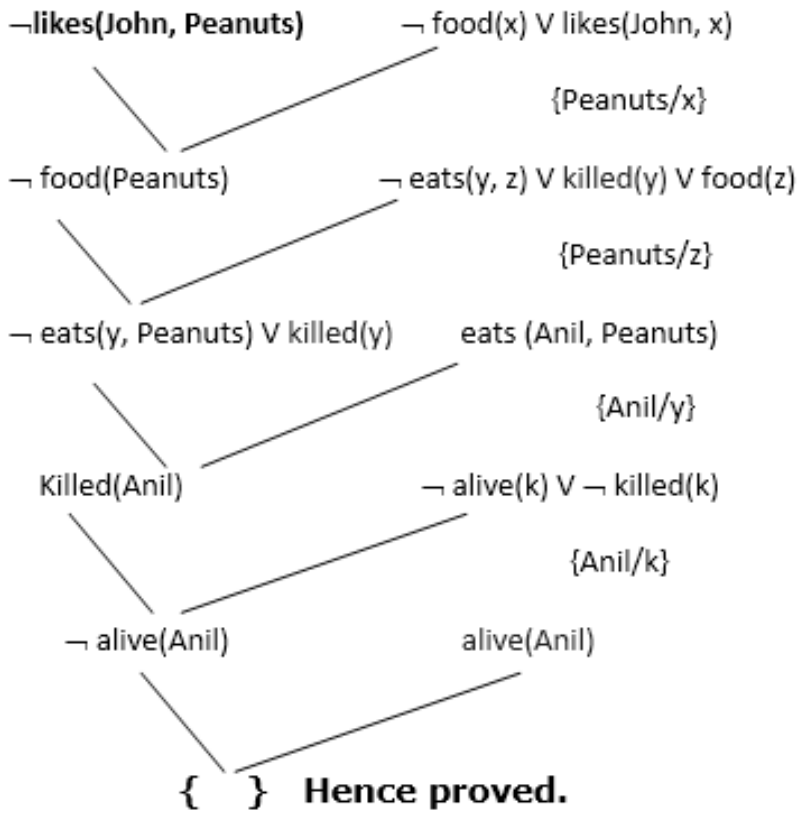
Step-3: Negate the statement to be proved

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

Step-4: Draw Resolution graph:

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

s:



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

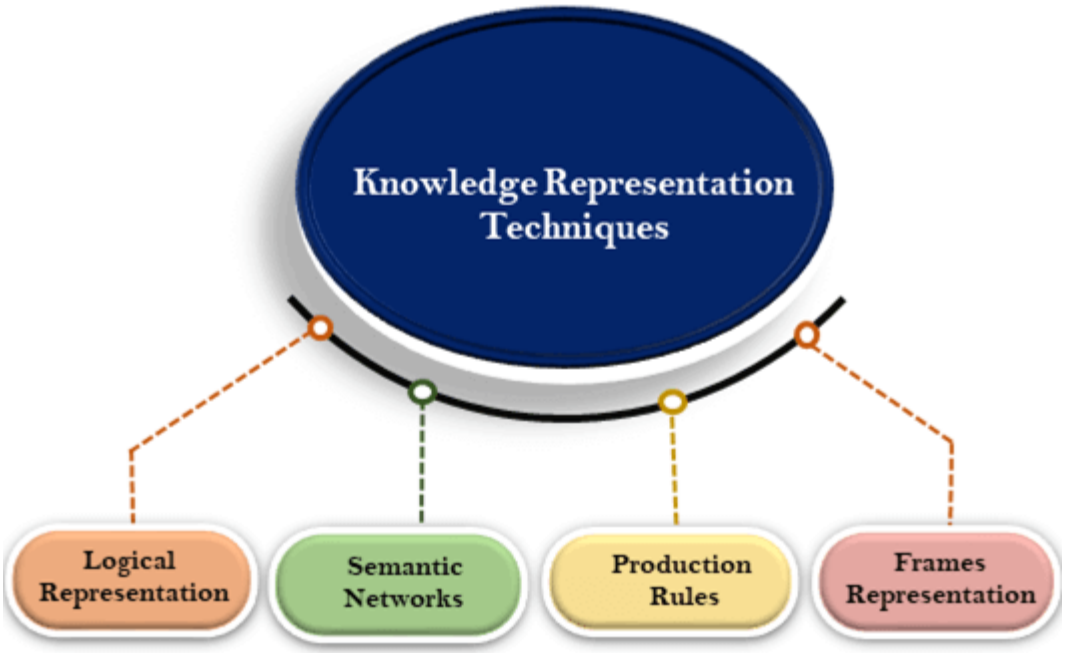
Explanation of Resolution graph:

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

Techniques of knowledge representation

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

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



1. Logical Representation

Logical representation is a language with some concrete rules which deals with propositions and has no ambiguity in representation. Logical representation means drawing a conclusion based on various conditions. This representation lays down some important communication rules. It consists of precisely defined syntax and semantics which supports the sound inference. Each sentence can be translated into logics using syntax and semantics.

Syntax:

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

Semantics:

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

Logical representation can be categorised into mainly two logics:

a. Propositional Logics

b. Predicate logics

Advantages of logical representation:

1. Logical representation enables us to do logical reasoning.

2. Logical representation is the basis for the programming languages.

Disadvantages of logical Representation:

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

2. Semantic Network Representation

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

This representation consist of mainly two types of relations:

a. IS-A relation (Inheritance)

b. Kind-of-relation

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

Statements:

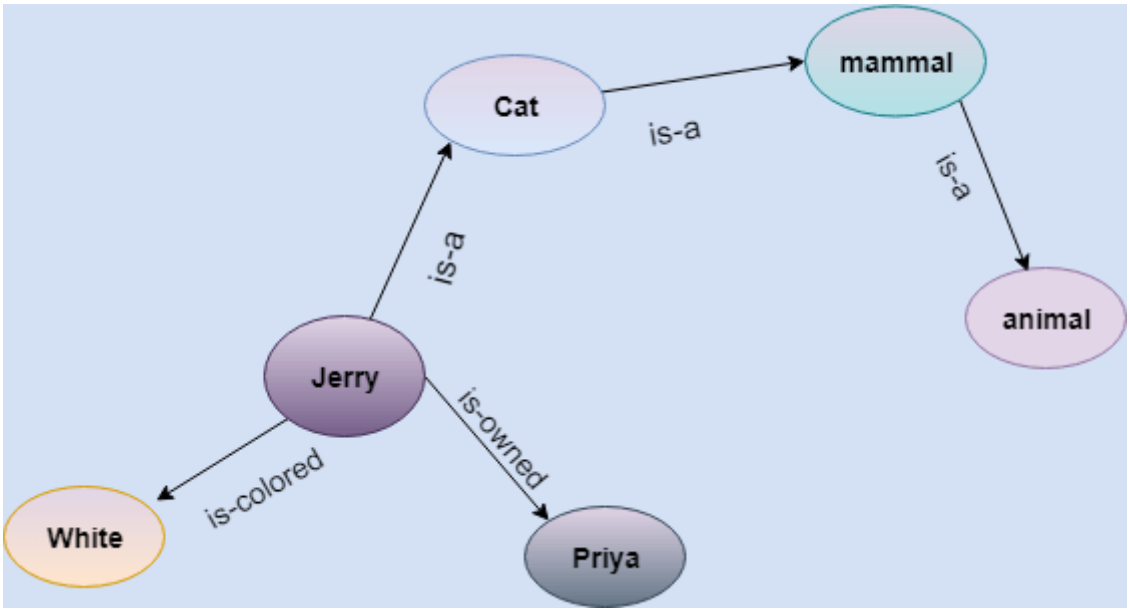
a. Jerry is a cat.

b. Jerry is a mammal

c. Jerry is owned by Priya.

d. Jerry is brown colored.

e. All Mammals are animal.



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

Drawbacks in Semantic representation:

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

Advantages of Semantic network:

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

3. Frame Representation

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

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

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

Example: 1

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

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

Example 2:

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

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

Advantages of frame representation:

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

Disadvantages of frame representation:

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

4. Production Rules

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

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

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

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

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

Example:

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

Advantages of Production rule:

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

Disadvantages of Production rule:

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

Types of Knowledge Representation

Knowledge can be represented in different ways. The structuring of knowledge and how designers might view it, as well as the type of structures used internally are considered. Different knowledge representation techniques are

a. Logic

b. Semantic Network

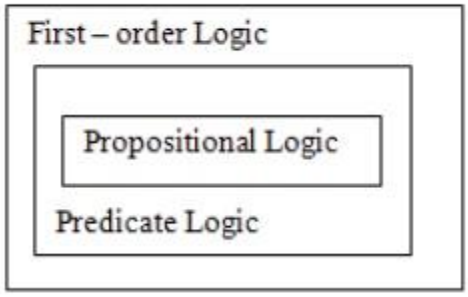
c. Frame
d. Conceptual Graphs

e. Conceptual Dependency

f. Script

Logic

A logic is a formal language, with precisely defined syntax and semantics, which supports sound inference. Different logics exist, which allow you to represent different kinds of things, and which allow more or less efficient inference. The logic may be different types like propositional logic, predicate logic, temporal logic, description logic etc. But representing something in logic may not be very natural and inferences may not be efficient.



Figure

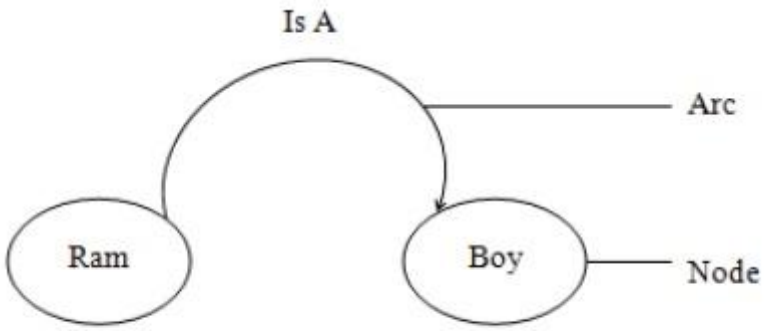
Semantic Network

A semantic network is a graphical knowledge representation technique. This knowledge representation system is primarily on network structure. The semantic networks were basically developed to model human memory. A semantic net consists of nodes connected by

arcs. The arcs are defined in a variety of ways, depending upon the kind of knowledge being represented.

The main idea behind semantic net is that the meaning of a concept comes, from the ways in which it is connected to other concepts. The semantic network consists of different nodes and arcs. Each node should contain the information about objects and each arc should contain the relationship between objects. Semantic nets are used to find relationships among objects by spreading activation about from each of two nodes and seeing where the activation met this process is called intersection search.

For example: Ram is a boy.



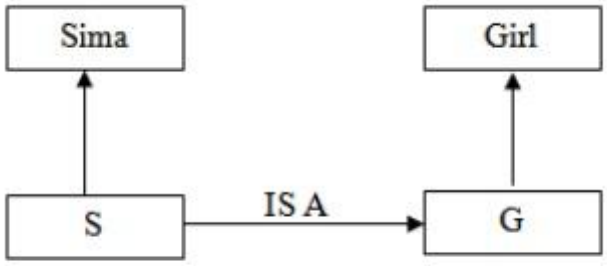
Figure

Semantic network by using Instances

The semantic network based knowledge representation mechanism is useful where an object or concept is associated with many attributes and where relationships between objects are important. Semantic nets have also been used in natural language research to represent complex sentences expressed in English. The semantic representation is useful because it provides a standard way of analyzing the meaning of sentence. It is a natural way to represent relationships that would appear as ground instances of binary predicates in predicate logic. In this case we can create one instance of each object. In instance based semantic net representations some keywords are used like: IS A, INSTANCE, AGENT, HAS-PARTS etc.

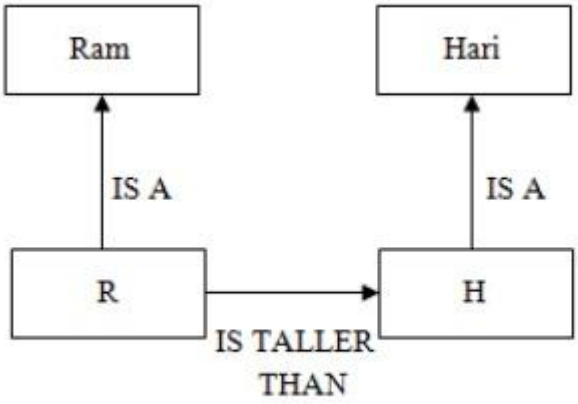
Consider the following examples:

1. Suppose we have to represent the sentence “Sima is a girl”.

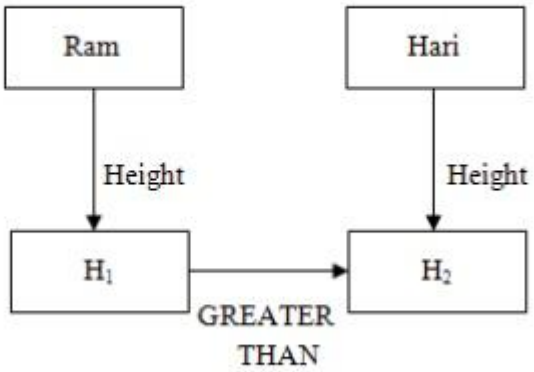


Figure

2. Ram is taller than Hari



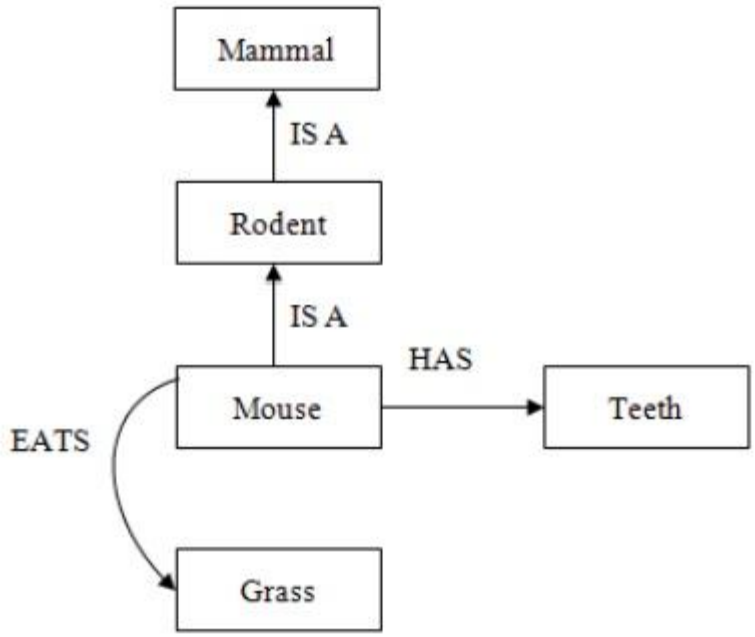
It can also be represented as



(b)

3. "Mouse is a Rodent and Rodent is a mammal. Mouse ha s teeth and etas grass". Check whether the sentence mammal has teeth is valid or not.]

(c)



Partitioned Semantic Network

Some complex sentences are there which cannot be represented by simple semantic nets and for this we have to follow the technique partitioned semantic networks. Partitioned semantic net allow for

1. Propositions to be made without commitment to truth.

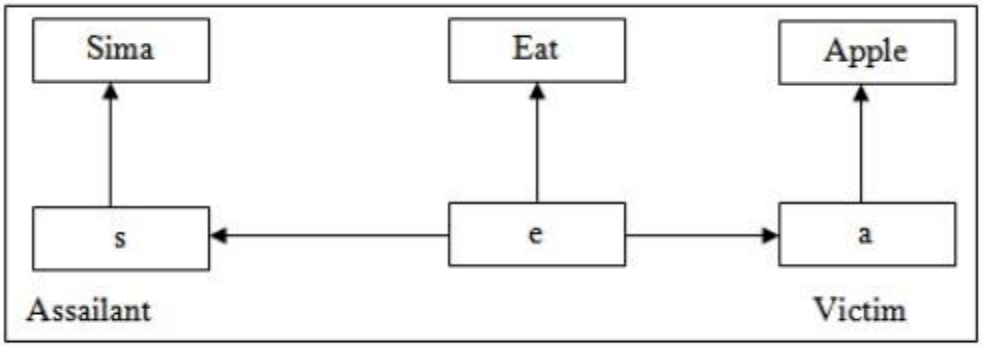
2. Expressions to be quantified.

In partitioned semantic network, the network is broken into spaces which consist of groups of nodes and arcs and regard each space as a node.

Let us consider few examples.

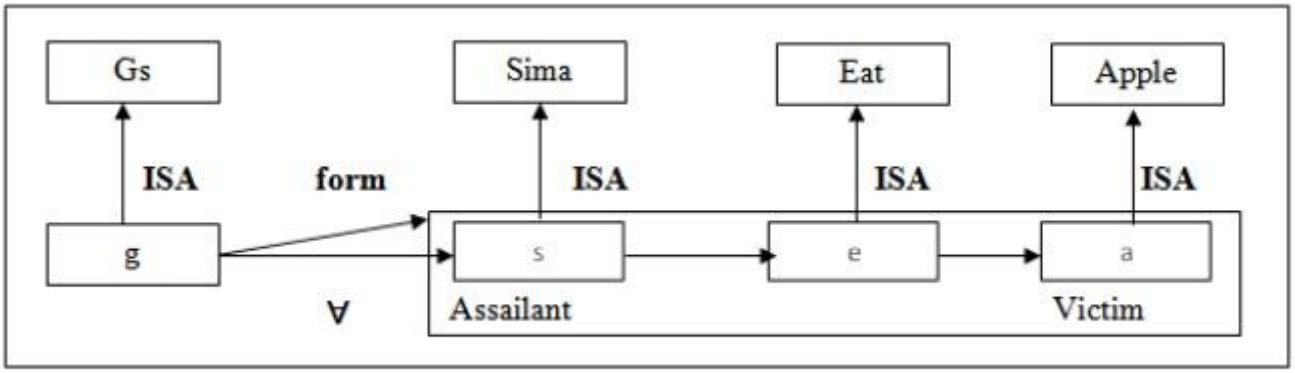
Draw the partitioned semantic network structure for the followings:

a) Sima is eating an apple.



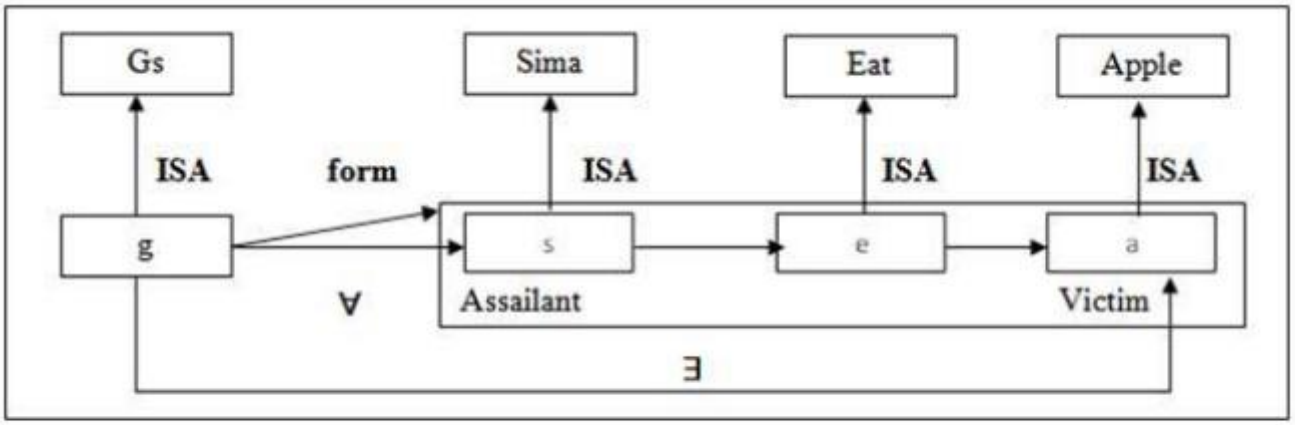
Figure

b) All Sima are eating an apple.



Figure

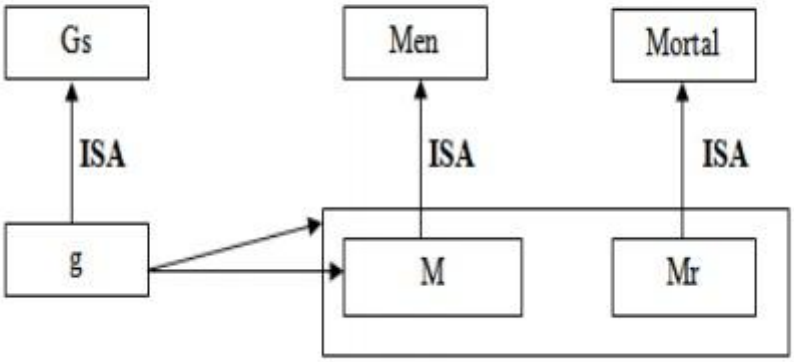
c) All Sima are eating some apple.



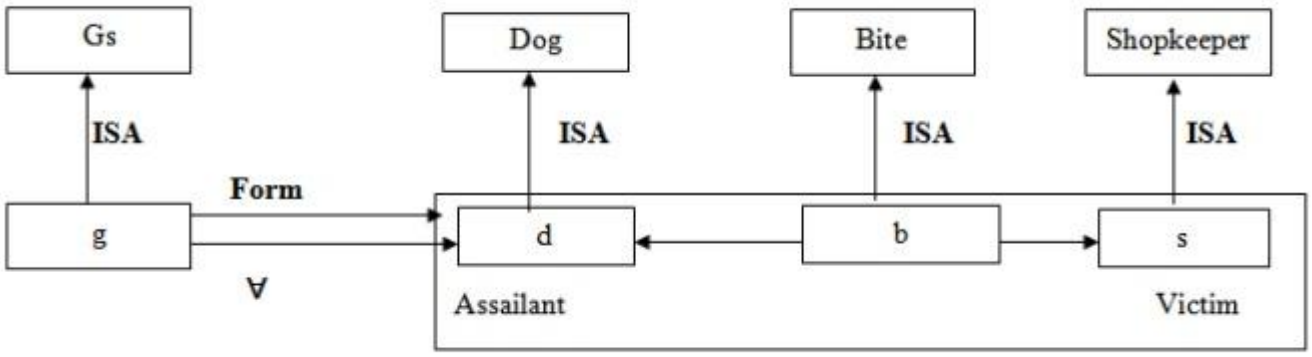
Figure

d) All men are mortal

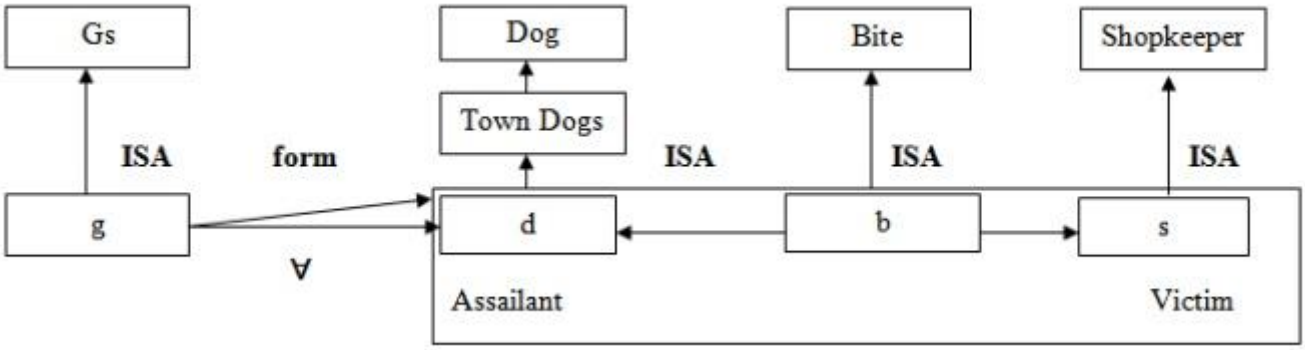
Figure



e) Every dog has bitten a shopkeeper



f) Every dog in town has bitten a shopkeeper.



NOTE: On the above semantic network structures, the instance "IS A" is used. Also two terms like assailant and victim are used. Assailant means "by which the work is done" and that of victim refers t o "on which the work is applied". Another term namely GS, which refers to General Statement. For GS, make a node g which is an instance of Gs. Every element will have at least two attributes. Firstly, a form that states which a relation is being asserted. Secondly, one or more for all (□) or there exists (□) connections which represent universally quantifiable variables.

FRAME

A frame is a collection of attributes and associated values that describe some entity in the world. Frames are general record like structures which consist of a collection of slots and slot values. The slots may be of any size and type. Slots typically have names and values or subfields called facets. Facets may also have names and any number of values. A frame may have any number of slots, a slot may have any number of facets, each with any number of values. A slot contains information such as attribute value pairs, default values, condition for filling a slot, pointers to other related frames and procedures that are activated when needed for different purposes. Sometimes a frame describes an entity in some absolute sense, sometimes it represents the entity from a particular point of view. A single frame taken alone is rarely useful. We build frame systems out of collection of frames that are connected to each other by virtue of the fact that the value of an attribute of one frame may be another frame. Each frame should start with an open parenthesis and closed with a closed parenthesis.

Syntax of a frame

(<frame name>
(<slot1> (<facet1> <value 1>.....<value n_1>)
(<facet2> <value1>.....<value n_2>)
.
.
.
.
.
(<facet n> <value1>..... <value n_n>))
(<slot 2> (<facet1> <value 1>.....<value n_1>)
(<facet2><value2>.....<value n_2>)
.
.
))

Let us consider the below examples.

1) Create a frame of the person Ram who is a doctor. He is of 40. His wife name is Sita. They have two children Babu and Gita. They live in 100 kps street in the city of Delhi in India. The zip code is 756005.

(Ram

(PROFESSION (VALUE Doctor))

(AGE (VALUE 40))

(WIFE (VALUE Sita))

(CHILDREN (VALUE Bubu, Gita))

(ADDRESS

(STREET (VALUE 100 kps))

(CITY(VALUE Delhi))

(COUNTRY(VALUE India))

(ZIP (VALUE 756005))))

2) Create a frame of the person Anand who is a chemistry professor in RD Women's College. His wife name is Sangita having two children Rupa and Shipa.

(Anand

(PROFESSION (VALUE Chemistry Professor))

(ADDRESS (VALUE RD Women's College))

(WIFE (VALUE Sangita))

(CHILDREN(VALUE RupaShipa)))

3) Create a frame of the person Akash who has a white maruti car of LX-400 Model. It has 5 doors. Its weight is 225kg, capacity is 8, and mileage is 15 km /lit.

(Akash

(CAR (VALUE Maruti))

(COLOUR (VALUE White))

(MODEL (VALUE LX-400))

(DOOR (VALUE 5))

(WEIGHT (VALUE 225kg))

(CAPACITY (VALUE 8))

(MILAGE (VALUE 15km/lit)))

The frames can be attached with another frame and can create a network of frames. The main task of action frame is to provide the facility for procedural attachment and help in reasoning process. Reasoning using frames is done by instantiation. Instantiation process begins, when the given situation is matched with frames that are already in existence. The reasoning process tries to match the current problem state with the frame slot and assigns them values.The values assigned to the slots depict a particular situation and by this, the reasoning process moves towards a goal. The reasoning process can be defined as filling slot values in frames.

Conceptual Graphs

It is a knowledge representation technique which consists of basic concepts and the relationship between them. As the name indicates, it tries to capture the concepts about the events and represents them in the form of a graph. A concept may be individual or generic. An individual concept has a type field followed by a reference field. For example person : Ram. Here person indicates type and Ram indicates reference.

An individual concept should be represented within a rectangle in graphical representation and within a square bracket in linear representation. The generic concept should be represented within an oval in graphical representation and within a parenthesis in linear representation. Conceptual graph is a basic building block for associative network. Concepts like AGENT, OBJECT, INSTRUMENT, PART are obtained from a collection of standard concepts. New concepts and relations can be defined from these basic ones. These are also basic building block for associative network. A linear conceptual graph is an elementary form of this structure. A single conceptual graph is roughly equivalent to a graphical diagram of a natural language sentence where the words are depicted as concepts and relationships.

Consider an example

“Ram is eating an apple “

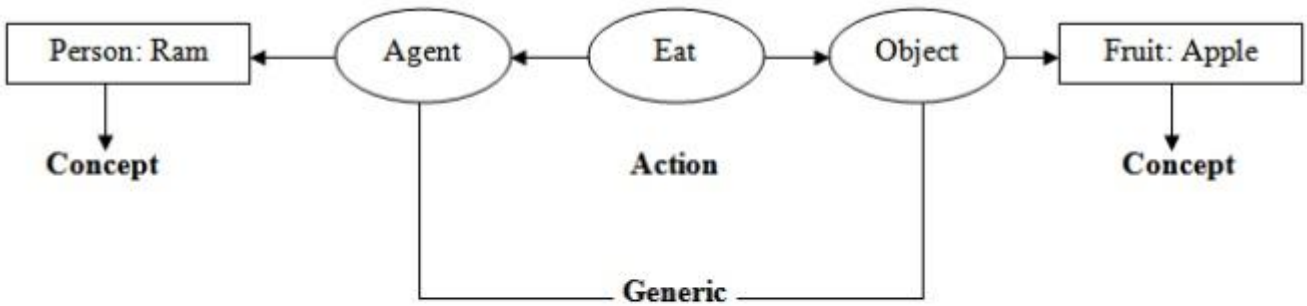
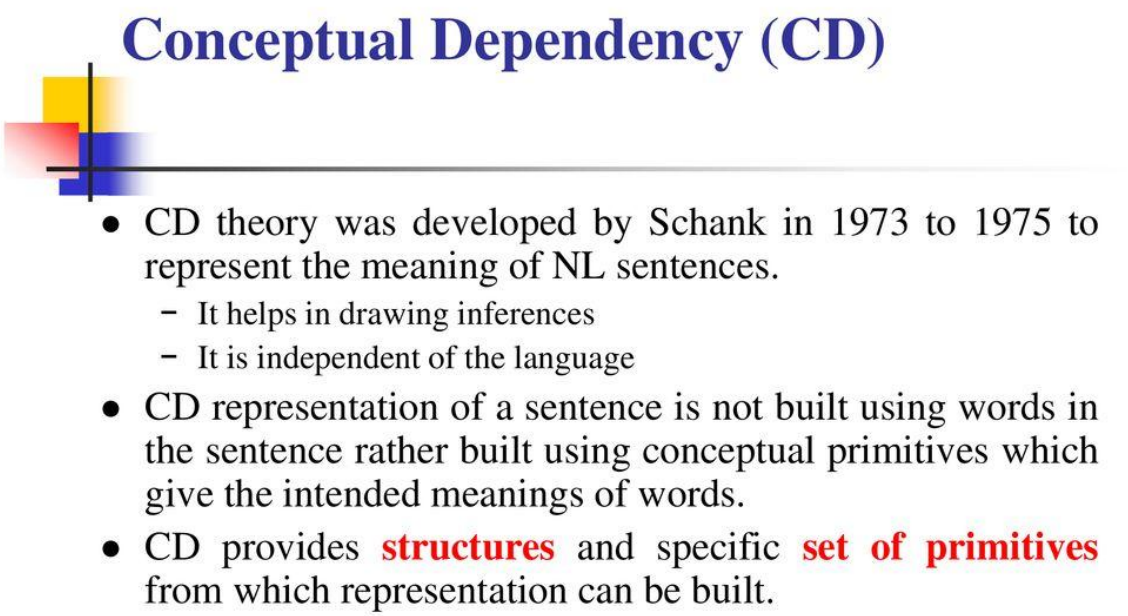


Figure Graphical Representation

[Person: Ram]←——(Agent)←— [Eat]——→(Object)——→[Fruit: Apple]

Conceptual Dependency



It is an another knowledge representation technique in which we can represent any kind of knowledge. It is based on the use of a limited number of primitive concepts and rules of formation to represent any natural language statement. Conceptual dependency theory is based on the use of knowledge representation methodology was primarily developed to understand and represent natural language structures. The conceptual dependency structures were originally developed by Roger C SChank in 1977.

If a computer program is to be developed that can understand wide phenomenon represented by natural languages, the knowledge representation should be powerful enough to represent these concepts. The conceptual dependency representation captures maximum concepts to provide canonical form of meaning of sentences. Generally there are four primitives from which the conceptual dependency structure can be described. They are

a. ACTS : Actions

b. PPs : Objects (Picture Producers)

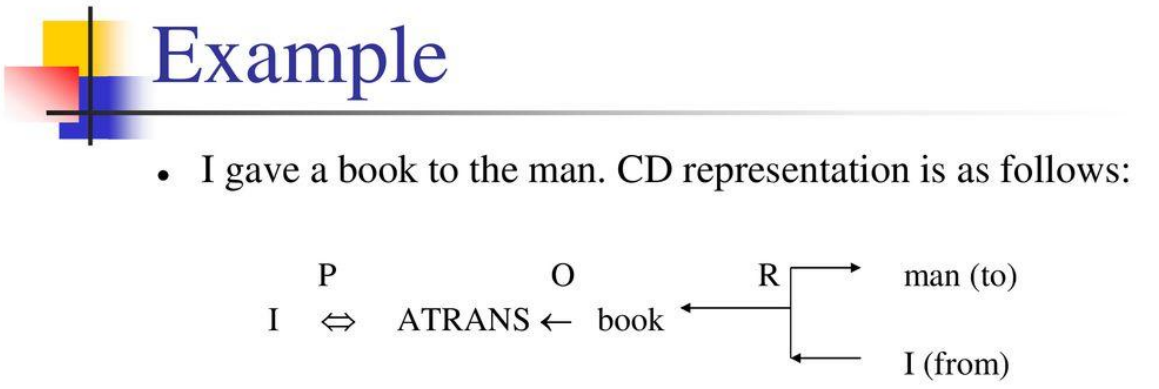
c. AAs : Modifiers of Actions (Action Aiders)

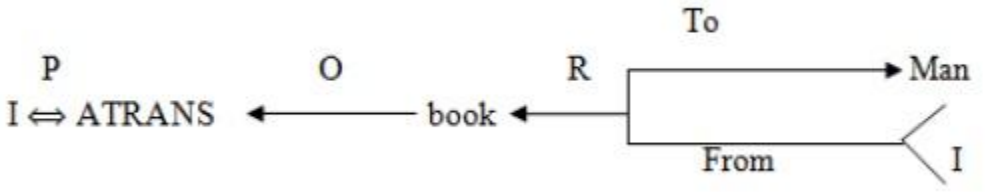
d. Pas : Modifiers of PPs (Picture Aiders)

e. TS : Time of action

Conceptual dependency provides both a structure and a specific set of primitives at a particular level of granularity, out of which representation of particular pieces of information can be constructed.

For example





Where ←: Direction of dependency

Double arrow indicates two way link between actor and action.

P: Past Tense

ATRANS: One of the primitive acts used by the theory

O: The objective case relation

R: Recipient case Relation

In CD, representation of actions are built from a set of primitive acts.

1) **ATRANS:** Transfer of an abstract relationship (give, accept, take)

2) **PTRANS:** Transfer the physical location of an object (Go, Come, Run, Walk)

3) **MTRANS:** Transfer the mental information (Tell)

4) **PROPEL:** Application of physical force to an object (push, pull, throw)

5) **MOVE:** Movement of a body part by its owner (kick).

6) **GRASP:** Grasping of an object by an action (clutch)

7) **INGEST:** Ingestion of an object by an animal (eat)

8)

EXPEL: Expel from an animal body (cry)

9) **MBUILD:** Building new information out of old (decide)

10) **SPEAK:** Production of sounds (say)

ATTEND: Focusing of a sense organ towards a stimulus (Listen)

The main goal of CD representation is to capture the implicit concept of a sentence and make it explicit. In normal representation of the concepts, besides actor and object, other concepts of time, location, source and destination are also mentioned. Following conceptual tenses are used in CD representation.

1) O: Object case relationship
2) R: Recipient case relationship
3) P : Past
4) F : Future
5) Nil : Present
6) T : Transition
7) Ts : Start Transition
8) Tf : Finisher Transition

9) K : Continuing

10) ? : Interrogative

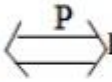
11) / : Negative

12) C : Conditional

Rule 1: PP  ACT

It describes the relationship between an actor and an event, he/she causes.

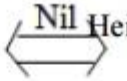
E.g. Ram ran

Ram  PTRANS

Where P: Past Tense

Rule 2: PP  PA

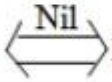
It describes the relationship between a PP and PA where the PA indicates one characteristics of PP. E.g. Ram is tall

Ram  Tall or Ram  Height (> Average)

Rule 3: PP  PP

It describes the relationship between two PPs where one PP is defined by other.

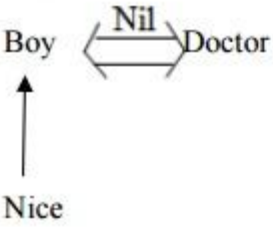
E.g. Ram is a doctor

Ram  Doctor

Rule 4: PP  PA or PA  PP

It describes the relationship between the PP and PA, where PA indicates one attributes of PP.

E.g. A nice boy is a doctor



Rule 5: PP

It describes the relationship between 3 PP's where one PP is the owner of another PP.

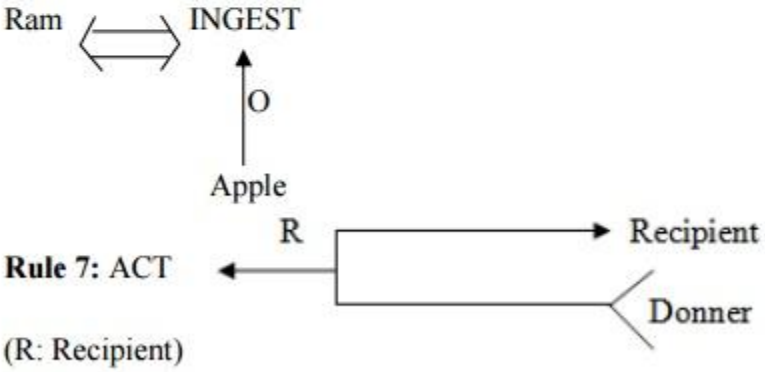
E.g. Ram's Cat

Cat

Ram

Rule 6: Act ← O PP Where O: Object

It describes the relationship between the PP and ACT. Where PP indicates the object of that action. E.g Ram is eating an apple.

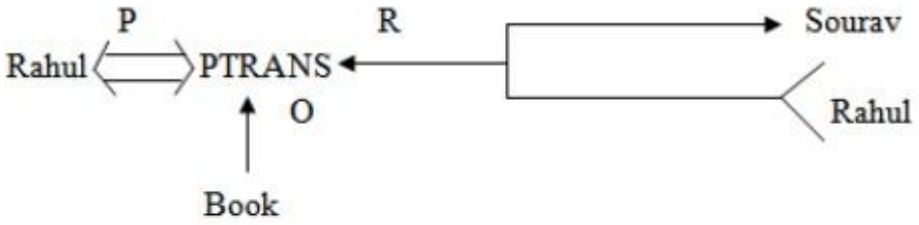


Rule 7: ACT

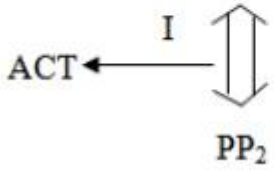
(R: Recipient)

Here one PP describes the recipient and another PP describes the donner

E.g. Rahul gave a book to sourav.



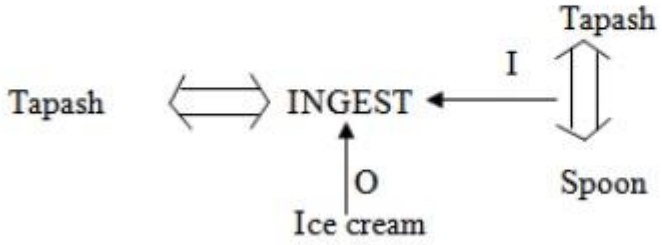
Rule 8:



(I: Instrument used in the action)

Here PP_1 indicates the agent and PP_2 indicates the object that is used in the action.

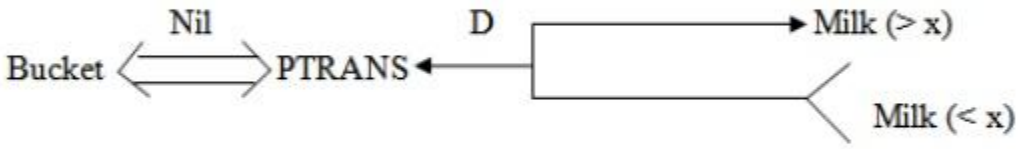
E.g. Tapash ate the ice cream with the spoon.



Rule 9:

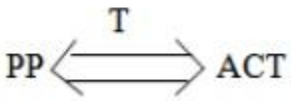
Here D indicates destination, PP_1 indicates destination and PP_2 indicates the source.

E.g. the bucket is filled with milk.



xindicates the average milk and the source i.e. bucket is dry which is hidden.

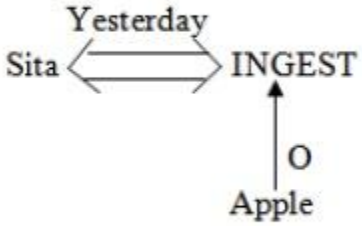
Rule 10:



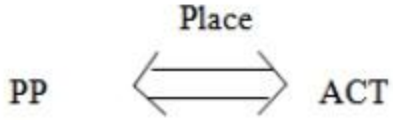
(T: Time)

It describes the relationship between a conceptualization and the time at which the event is described occurs.

E.g. Sita ate the apple yesterday.

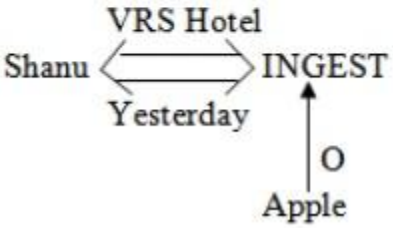


Rule 11:

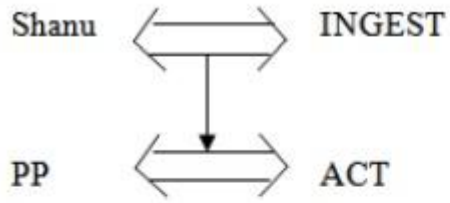


It describes the relationship between a conceptualization and the place at which it is occurred.

E.g. Shanu ate the apple at VRS hotel yesterday

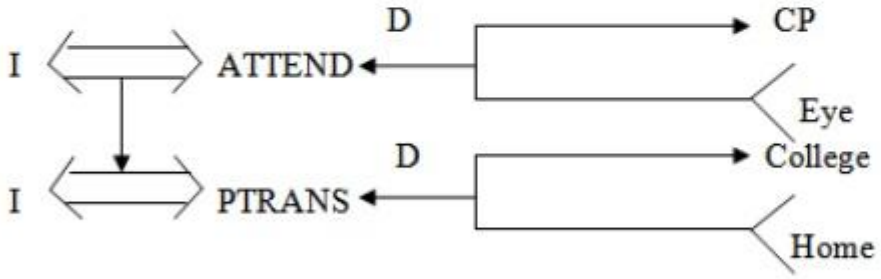


Rule 12:



It describes the relationship between one conceptualization with another.

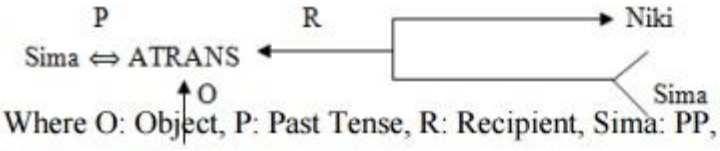
E.g. while I was going to college, I saw a snake



(Where CP: Conscious Processor i.e. the combination of all sense organs like eye, ear, nose etc.)

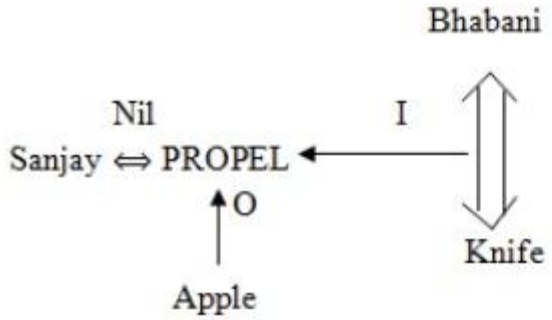
By using the above rules we can represent any sentence. Let us visualize few examples on conceptual dependency.

1) Sima gave a book to Niki

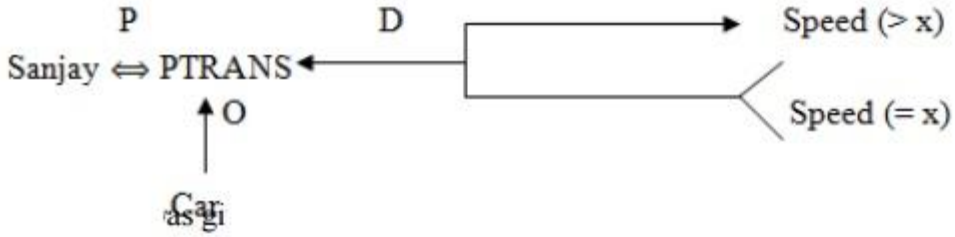


Where O: Object, P: Past Tense, R: Recipient, Sima: PP, Book: PP, Niki: PP, ATRANS: give

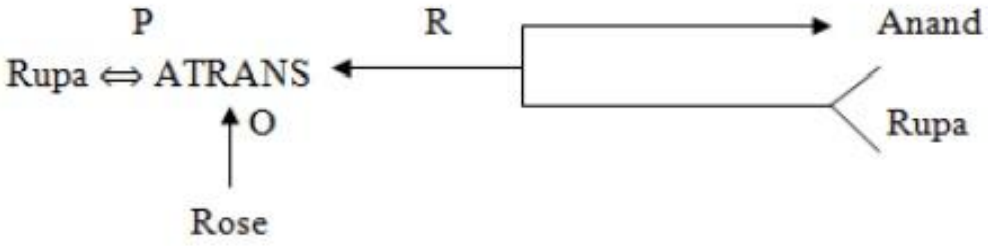
2) Bhabani cuts an apple with a knife



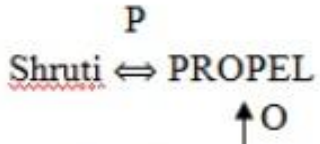
3) Sanjay drove the car fast



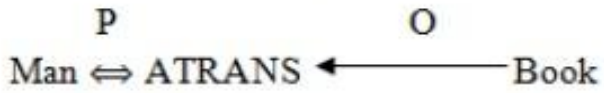
4) The rose was given by Rupa to Anand



5) Shruti pushed the door.



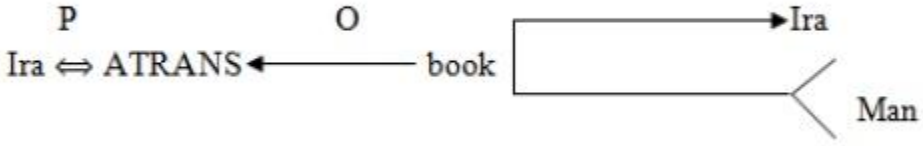
6) The man took a book



Here man is the doctor and book is the object of the action took.

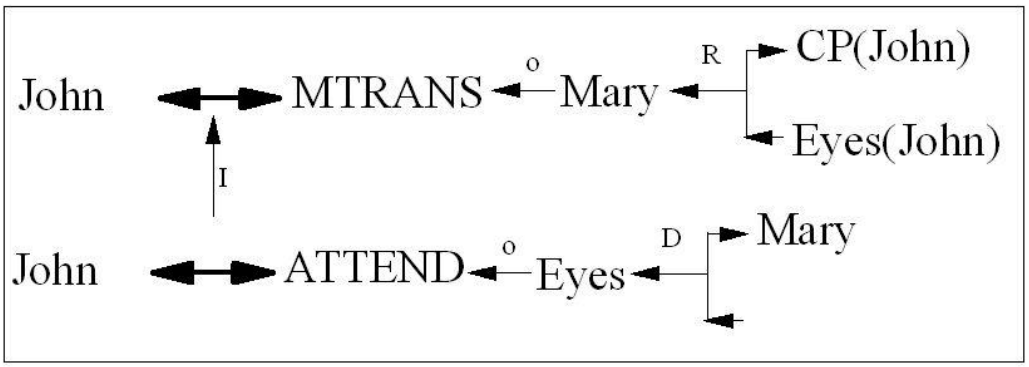
7) My grandfather told me a story

8) Ira gave the man a dictionary



CD Examples

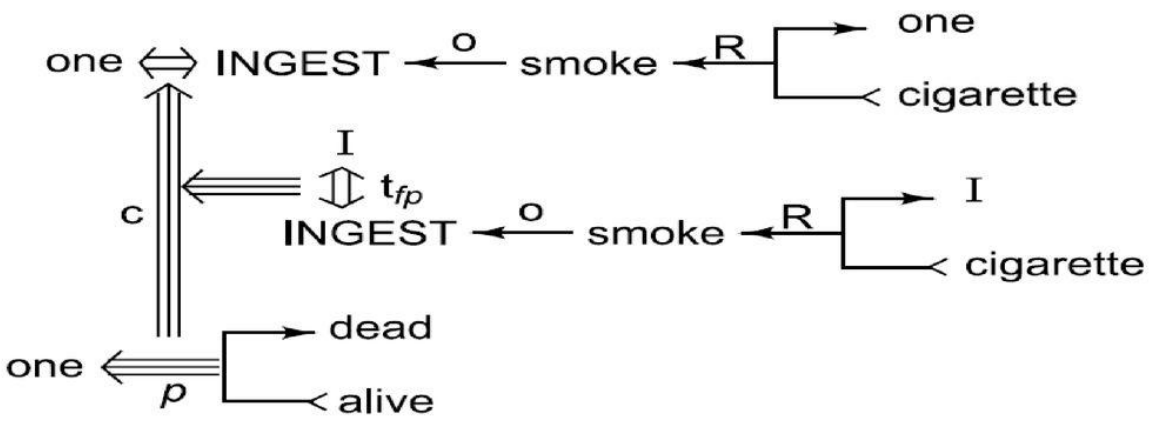
- John saw Mary.



Here, CP is the "Cognitive Processor" of John, or John's brain.

Using Conceptual Tenses

- "Since smoking can kill you, I stopped."



241

Fig 7.9 Conceptual dependency representing "John ate the egg" (Schank and Rieger 1974).

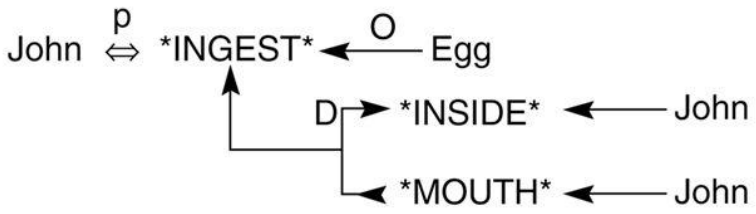
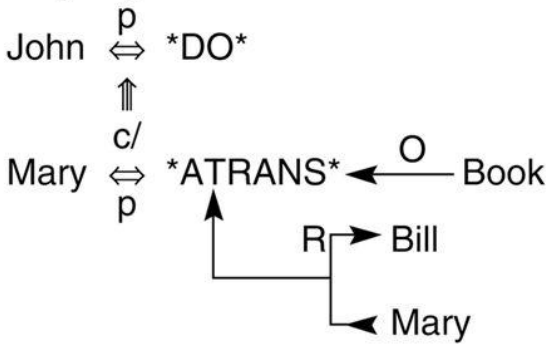


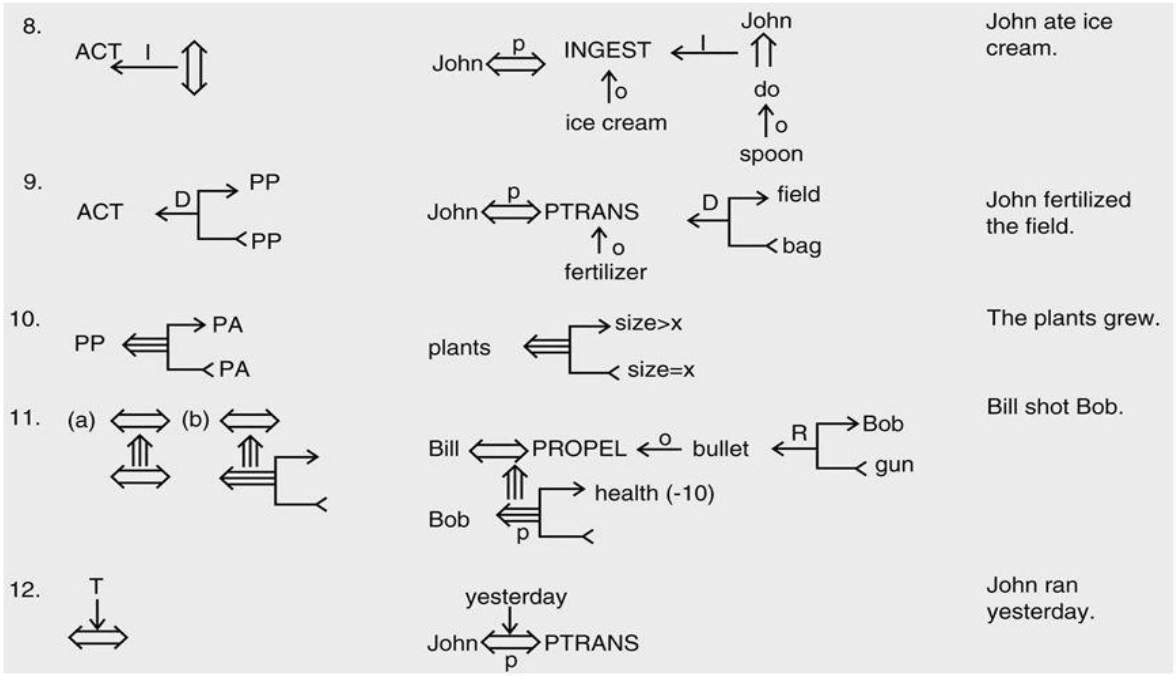
Fig 7.10 Conceptual dependency representation of the sentence "John prevented Mary from giving a book to Bill" (Schank and Rieger 1974).



Examples with the basic conceptual dependencies

1.	PP ⟺ ACT	John ⟺ PTRANS	John ran.
2.	PP ⟺ PA	John ⟺ height (>average)	John is tall.
3.	PP ⟺ PP	John ⟺ doctor	John is a doctor.
4.	PP ↑ PA	boy ↑ nice	A nice boy
5.	PP ⇑ PP	dog ⇑ POSS-BY John	John's dog
6.	ACT ←o PP	John ⟺p PROPEL ←o cart	John pushed the cart.
7.	ACT ←R PP, PP	John ⟺p ATRANS ↑o book, R → John, Mary	John took the book from Mary.

Examples with the basic conceptual dependencies (cont'd)



SCRIPT

It is an another knowledge representation technique. Scripts are frame like structures used to represent commonly occurring experiences such as going to restaurant, visiting a doctor. A script is a structure that describes a stereotyped sequence of events in a particular context. A script consist of a set of slots. Associated with each slot may be some information about what kinds of values it may contain as well as a default value to be used if no other information is available. Scripts are useful because in the real world, there are no patterns to the occurrence of events. These patterns arise because of clausal relationships between events. The events described in a script form a giant casual chain. The beginning of the chain is the set of entry conditions which enable the first events of the script to occur. The end of the chain is the set of results which may enable later events to occur. The headers of a script can all serve as indicators that the script should be activated.

Once a script has been activated, there are a variety of ways in which it can be useful in interpreting a particular situation. A script has the ability to predict events that has not

explicitly been observed. An important use of scripts is to provide a way of building a single coherent interpretation from a collection of observation. Scripts are less general structures than are frames and so are not suitable for representing all kinds of knowledge. Scripts are very useful for representing the specific kinds of knowledge for which they were designed.

A script has various components like:

1) **Entry condition**: It must be true before the events described in the script can occur. E.g. in a restaurant script the entry condition must be the customer should be hungry and the customer has money.

2) **Tracks:** It specifies particular position of the script e.g. In a supermarket script the tracks may be cloth gallery, cosmetics gallery etc.

3) **Result:** It must be satisfied or true after the events described in the script have occurred. e.g. In a restaurant script the result must be true if the customer is pleased. The customer has less money.

4) **Probs:** It describes the inactive or dead participants in the script e.g. In a supermarket script, the probes may be clothes, sticks, doors, tables, bills etc.

5) Roles: It specifies the various stages of the script. E.g. In a restaurant script the scenes may be entering, ordering etc.

Now let us look on a movie script description according to the above component.

a) Script name : Movie

b) Track : CINEMA HALL

c) Roles : Customer(c), Ticket seller(TS), Ticket Checker(TC), Snacks Sellers (SS)

d) Probes : Ticket, snacks, chair, money, Ticket, chart

e) Entry condition : The customer has money

The customer has interest to watch movie.

6) Scenes:

a. SCENE-1 (Entering into the cinema hall)

CPTRANS C into the cinema hall

C ATTEND eyes towards the ticket counter C PTRANS C towards the ticket counters C ATTEND eyes to the ticket chart

C MBUILD to take which class ticket C MTRANS TS for ticket

C ATRANS money to TS

TS ATRANS ticket to C

b. SCENE-2 (Entering into the main ticket check gate)

C PTRANS C into the queue of the gate C ATRANS ticket to TC

TC ATTEND eyes onto the ticket

TC MBUILD to give permission to C for entering into the hall

TC ATRANS ticket to C

C PTRANS C into the picture hall.

c. **SCENE-3 (Entering into the picture hall)**

CATTEND eyes into the chair

TC SPEAK where to sit

C PTRANS C towards the sitting position

C ATTEND eyes onto the screen

d. SCENE-4 (Ordering snacks)

C MTRANS SS for snacks

SS ATRANS snacks to C

C ATRANS money to SS

C INGEST snacks **e. SCENE-5 (Exit)**

C ATTEND eyes onto the screen till the end of picture

C MBUILD when to go out of the hall

C PTRANS C out of the hall

7) Result:

The customer is happy

The customer has less money

Example 2: Write a script of visiting a doctor in a hospital

1) SCRIPT_NAME : Visiting a doctor

2) TRACKS : Ent specialist

3) ROLES : Attendant (A), Nurse(N), Chemist (C), Gatekeeper(G), Counter clerk(CC), Receptionist(R), Patient(P), Ent specialist Doctor (D), Medicine Seller (M).

4) PROBES : Money, Prescription, Medicine, Sitting chair, Doctor's table, Thermometer, Stetho scope, writing pad, pen, torch, stature.

5) ENTRY CONDITION: The patient need consultation. Doctor's visiting time on.

6) SCENES:

a. SCENE-1 (Entering into the hospital)

PPTRANS P into hospital

P ATTEND eyes towards ENT department

P PTRANS P into ENT department

P PTRANS P towards the sitting chair

b. SCENE-2 (Entering into the Doctor's Room)

P PTRANS P into doctor’s room

P MTRANS P about the diseases

P SPEAK D about the disease

D MTRANS P for blood test, urine test

D ATRANS prescription to P

P PTRANS prescription to P.

P PTRANS P for blood and urine test

c. SCENE-3 (Entering into the Test Lab)

P PTRANS P into the test room

P ATRANS blood sample at collection room

P ATRANS urine sample at collection room

P ATRANS the examination reports

d. SCENE-4 (Entering to the Doctor’s room with Test reports)

P ATRANS the report to D

D ATTEND eyes into the report

D MBUILD to give the medicines

D SPEAK details about the medicine to P

P ATRANS doctor’s fee

P PTRANS from doctor’s room

e. SCENE-5 (Entering towards medicine shop)

P PTRANS P towards medicine counter

P ATRANS Prescription to M

M ATTEND eyes into the prescription

M MBUILD which medicine to give

M ATRANS medicines to P

P ATRANS money to M

P PTRANS P from the medicine shop

7) RESULT:

The patient has less money

Patient has prescription and medicine.

Advantages And Disadvantages Of Different Knowledge Representation

Sl. No.	Scheme	Advantages	Disadvantages
1	Production rules	• Simple syntax • Easy to understand • Simple interpreter • Highly Modular • Easy to add or modify	• Hard to follow Hierarchies • Inefficient for large systems • Poor at representing structured descriptive knowledge.
2	Semantic	• Easy to follow hierarchy • Easy to trace associations • Flexible	• Meaning attached to nodes might be ambiguous • Exception handling is difficult • Difficult to program
3	Frame	• Expressive Power • Easy to set up slots for new properties and relations • Easy to create specialized	• Difficult to program • Difficult for inference • Lack of inexpensive software

		procedures	
4	Script	• Ability to predict events • A single coherent interpretation may be build up from a collection of observations	• Less general than frames • May not be suitable to represent all kinds of knowledge
5	Formal Logic	• Facts asserted independently of use • Assurance that only valid consequence are asserted • Completeness	• Separation of representation and processing • Inefficient with large data sets • Very slow with large knowledge bases

Unit-3

Uncertainty:

Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge

representation, we might write A→B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.

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

Causes of uncertainty:

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

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

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.

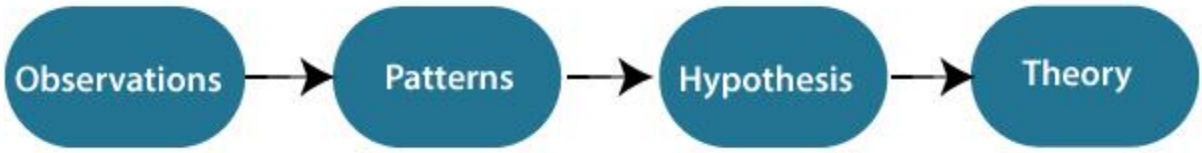
In inductive reasoning, we use historical data or various premises to generate a generic rule, for which premises support the conclusion.

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**.

Difference between Inductive and Deductive reasoning

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

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

Comparison Chart:

Basis for comparison	**Deductive Reasoning**	**Inductive Reasoning**
Definition	Deductive reasoning is the form of valid reasoning, to deduce new information or conclusion from known related facts and information.	Inductive reasoning arrives at a conclusion by the process of generalization using specific facts or data.
Approach	Deductive reasoning follows a top-down approach.	Inductive reasoning follows a bottom-up approach.
Starts from	Deductive reasoning starts from Premises.	Inductive reasoning starts from the Conclusion.

Validity	In deductive reasoning conclusion must be true if the premises are true.	In inductive reasoning, the truth of premises does not guarantee the truth of conclusions.
Usage	Use of deductive reasoning is difficult, as we need facts which must be true.	Use of inductive reasoning is fast and easy, as we need evidence instead of true facts. We often use it in our daily life.
Process	Theory→ hypothesis→ patterns→confirmation.	Observations-→patterns→hypothesis→Theory.
Argument	In deductive reasoning, arguments may be valid or invalid.	In inductive reasoning, arguments may be weak or strong.
Structure	Deductive reasoning reaches from general facts to specific facts.	Inductive reasoning reaches from specific facts to general facts.

Truth Maintenance System (TMS):

The logics are known as monotonic logics. The conclusions derived using such logics are valid deductions, and they remain so for all times. Adding new axioms increases the amount of knowledge contained in the knowledge base. Therefore, the set of facts and inferences in such systems can only grow larger; they cannot be reduced; that is, they increase monotonically.

The form of reasoning referred to above, on the other hand, is non-monotonic. New facts become known which can contradict and can invalidate the old knowledge. The old knowledge is retracted causing other dependent knowledge to become invalid, thereby requiring further retractions. The retractions lead to a shrinkage or growth of knowledge base, called non-monotonic growth in the knowledge, at times.

This can be illustrated by a real-life situation. Suppose a young boy Sahu enjoys seeing movie in a cinema hall on the first day of its release. He insists upon his grand father, Mr.

Girish in accompanying him. Mr. Girish has agreed to accompany Sahu there on the following Friday evening. On the Thursday, when forecasts predicted heavy snow.

Now, believing the weather would discourage most senior citizens, Girish changed his mind of joining Mr. Sahu. But, unexpectedly, on the given Friday, the forecasts proved to be false; so Mr. Girish once again went to see movie. This is the case of non-monotonic reasoning.

It is not reasonable to expect that all the knowledge needed for a set of tasks could be acquired, validated, and loaded into the system at the outset. More typically, the initial knowledge will be incomplete, contain redundancies, inconsistencies, and other sources of uncertainty. Even if it were possible to assemble complete, valid knowledge initially, it probably would not remain valid forever, more so in a continually changing environment.

In an attempt to model real-world, commonsense reasoning, researchers have proposed extensions and alternatives to traditional logics such as Predicate Logic and First Order Predicate Logic. The extensions accommodate such real time forms of uncertainty and non-monotony as experienced by our subject, Mr. Girish.

We now give a description of Truth maintenance systems (TMS), which have been implemented to permit a form of non-monotonic reasoning by permitting the addition of changing (even contradictory) statements to a knowledge base. Truth maintenance system (also known as belief revision system) is a companion component to inference system.

The main object of the TMS is the maintenance of the knowledge base used by the problem solving system and not to perform any inference. As such, it frees the problem solver from any concerns of knowledge consistency check when new knowledge gets added or deleted and allows it to concentrate on the problem solution aspects.

The TMS also gives the inference component the latitude to perform non-monotonic inferences. When new discoveries are made, this more recent information can displace the previous conclusions which are no longer valid.

In this way, the set of beliefs available to the problem solver will continue to be current and consistent.

Fig. 7.1 illustrates the role played by the TMS as a part of the problem solving system. The Inference Engine (IE) from the expert system or decision support system solves domain specific problems based on its current belief set, maintained by the TMS. The updating process is incremental. After each inference, information is exchanged between the two components the IE and the TMS.

The IE tells the TMS what deductions it has made. The TMS, in turn, asks questions about current beliefs and reasons for failure of earlier statements. It maintains a consistent set of beliefs for the IE to work with when the new knowledge is added or removed.

For example, suppose the knowledge base (KB) contained only the propositions P and $P \rightarrow Q$, and modus ponens. From this, the IE would rightfully conclude Q and add this conclusion to the KB. Later, if it was learned that $\sim P$ become true it would be added to the KB resulting in P becoming false leading to a contradiction. Consequently, it would be necessary to remove P to eliminate the inconsistency. But, with P now removed, Q is no longer a justified belief. It too should be removed. This type of belief revision is the job of the TMS.
Actually, the TMS does not discard conclusions like Q as suggested. That could be wasteful, since P may again become valid, which would require that Q and facts justified by Q be re-derived. Instead, the TMS maintains dependency records for all such conclusions.

These records determine which set of beliefs are current and are to be used by the IE. Thus, Q would be removed from the current belief set by making appropriate updates to the records and not by erasing Q. Since Q would not be lost, its re-derivation would not be necessary if and when P became valid once again.

The TMS maintains complete records of reasons or justifications for beliefs. Each proposition or statement having at least one valid justification is made a part of the current belief set. Statements lacking acceptable justifications are excluded from this set.

When a contradiction is discovered, the statements responsible for the contradiction are identified and an appropriate one is retracted. This in turn may result in other reactions and additions. The procedure used to perform this process is called dependency directed back tracking which will be explained shortly.

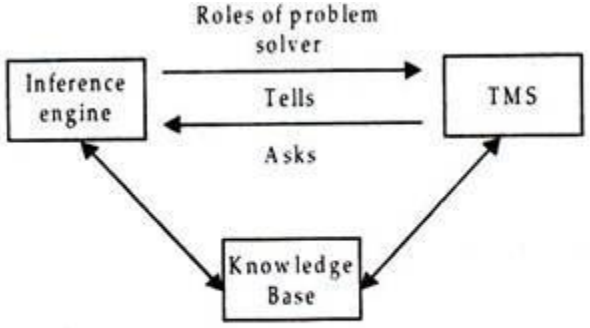


Fig. 7.1. *Architecture of the problem solver with a TMS.*

The TMS maintains records to show retractions and additions so that the IE will always know its current belief set. The records are maintained in the form of a dependency network. The nodes in the network represent KB entries such as premises, conclusions, inference rules etc.

To the nodes are also attached justifications which represent the inference steps from which the node was derived. Nodes in the belief set must have valid justifications. A premise is a fundamental belief which is assumed to be always true and need no justifications. In fact, they form a base from which all other currently active nodes can be explained in terms of valid justification.

There are two types of justification records maintained for nodes:
1. Support Lists (SL) and

2. Conceptual Dependencies (CP).

SLs are more common. They provide the supporting justifications for nodes. The SL contains two lists of other dependent node names, an in-list and an out-list. It has the form.

SL <in-list> <out-list>

For a node to be active (labeled as IN the belief set), its SL must have at least one valid node in its in-list, and all nodes named in its out-list, if any, must be marked OUT of the belief set. For example, a current belief set which represents that Oosho is a non-flying bird (an ostrich) might have the nodes and justifications listed in Table 7.1.

Table 7.1. *Example nodes in a dependency network*

Node	Status	Meaning	Support list	Commends (justification)
nl	IN	Oosho is a bird	SL ( ) ( ))	a premise
n2	OUT	Oosho can fly	(SL (nl) (n3)	unjustified belief
n3	IN	Oosho can't fly	(SL (n5) (n4))	justified belief
n4	OUT	Oosho has wings	(SL ( ) ( ))	retracted premise
n5	IN	Oosho is an Ostrich	(SL ( ) ( ))	a premise

Each IN-node given in Table 7.1, is part of the current belief set. Nodesn 1 and n5 are premises. They have empty support lists since they do not require justifications. Node n2, the belief that Oosho can fly is out because n3; a valid node, is in the out-list of n2.

Suppose later on it is discovered that Oosho is not an ostrich, thereby causing n 5 to be retraced as OUT. Then n3, which depends on n5, must be also retracted. This, in turn,

changes the status of n2 to be a justified node. The resultant belief set is now that the bird Oosho can fly.

For representation of a belief network the symbol conventions shown in Fig. 7.2 are quite often used.

The meaning of the nodes shown in the figure are:
(1) A premise is a true proposition requiring no justification,

(2) An assumption is a proposition which is held true because there is no evidence against that,

(3) A datum is either a currently assumed or IE derived belief, and

(4) Justifications are the belief nodes consisting of supporting antecedent node links and a consequent node link.

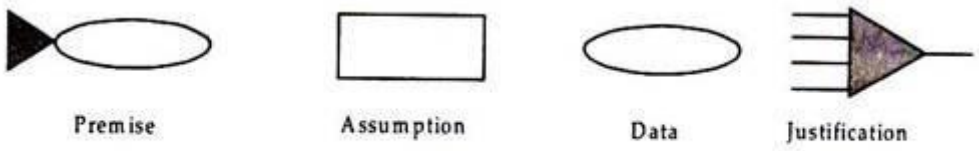


Fig. 7.2. *Meanings in belief network.*

An example of a typical network representation is given in Fig. 7.3. Nodes T, U, and W are OUT since they lack needed support from P. If the node labeled P is made IN for some reason, the TMS would update the network by propagating the "in ness" support provided by node P to make T, U, and WIN.

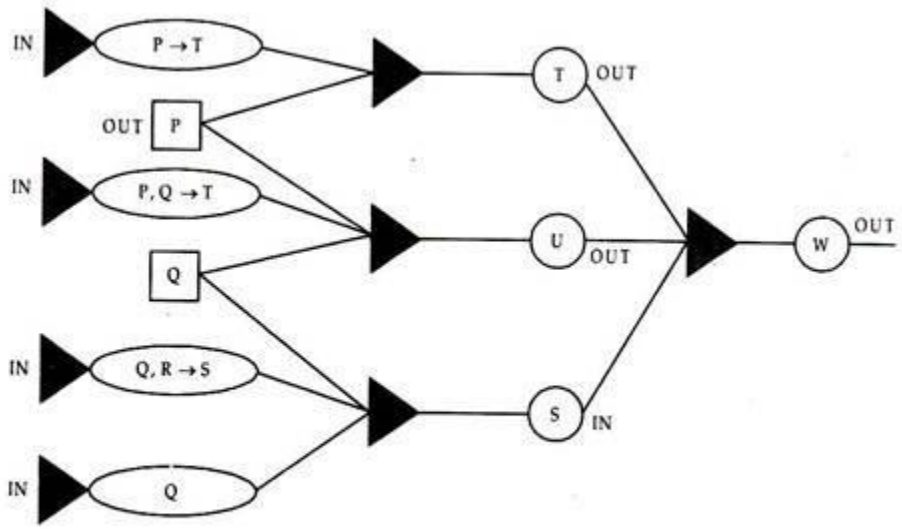


Fig. 7.3. *A skeleton belief network*

When a contradiction is discovered, the TMS locates the source of the contradiction and corrects it by retracting one of the contributing sources. It does this by checking the support lists of the contradictory node and going directly to the source of the contradiction.

It goes directly to the source by examining the dependency structure supporting the justification and fixing the annoying nodes, ('guilty' premises), thereby dismantling the justification for the contradiction.

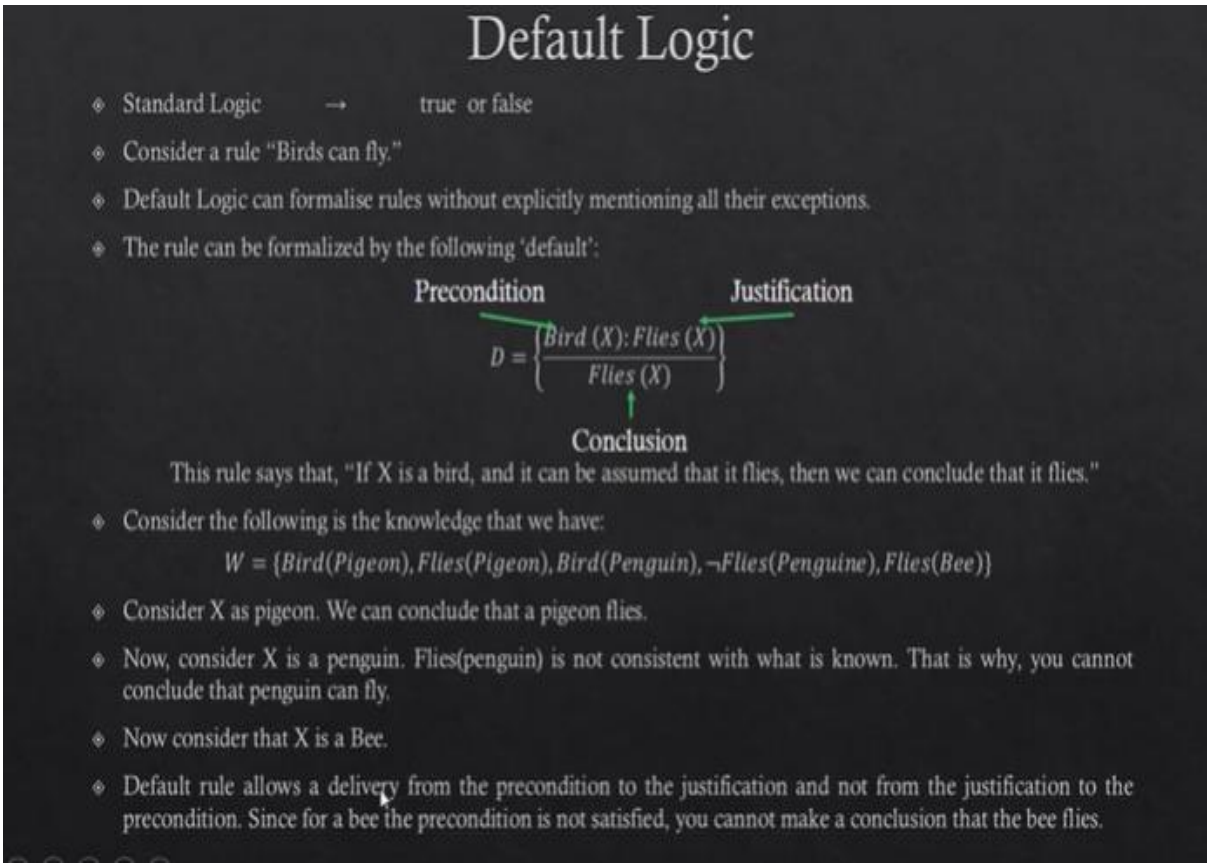
This is in contrast to the chronological backtracking approach mentioned many times, which would search a deduction tree sequentially, node-by-node until the contradictory node is reached. Backtracking directly to the node causing the contradiction is known as Dependency-Directed Backtracking (DDB). This is clearly a more efficient search strategy than chronological backtracking. By backtracking directly to the source of a contradiction extra search time is saved.

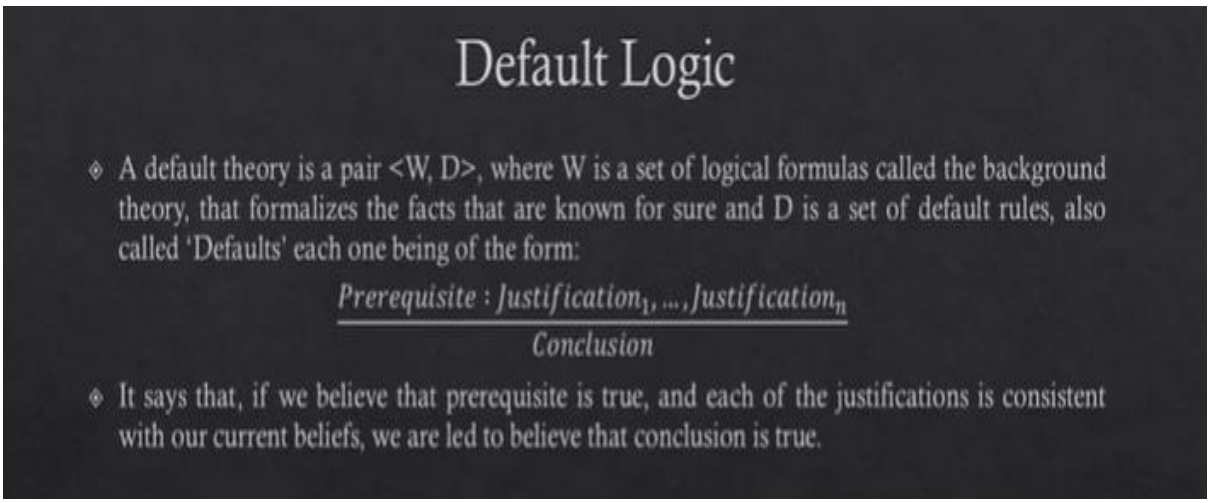
TMS discussed above concentrates on maintaining a single, consistent world, model. However, sometimes it is convenient to perform reasoning in the context of different hypothetical worlds, which may or may not resemble the way the world actually is. For example, in doing diagnosis, it is often worthwhile to assume that a certain fault has occurred and then make predictions on the basis of this assumption and see if they are backed up by evidence.

This strategy is particularly useful if there are a large number of hypothesis competing to account for the observations, with the possibility that a composite hypothesis may be required to cover all of them. Another domain of application might be arrangement problems/where the hypothetical worlds represent different ways of arranging objects to satisfy a set of constraints.

To maintain multiple contexts more sophisticated systems are Logic-based TSM and assumption based TMS among others. Other more sophisticated systems are Logic-based TMS (LTMS), assumption-based TMS (ATMS) among others.

Default Logic





For example, if x is an adult and it is consistent to assume that x can drive, then infer that x can drive. Formula:

$$\frac{\text{ADULT}(x) : \text{MDRIVE}(x)}{\text{DRIVE}(x)}$$

Here, M is the consistency operator.

Default rules are especially useful in hierarchial KBs. Because the default rules are transitive, property inheritance becomes possible. For example, in a heirarchy of living things, any animal could inherit the property has-heart from the rule

$$\forall x \ \text{ANIMAL}(x) \rightarrow \text{HAS-HEART}(x)$$

Transitivity can also be a problem in KBs with many default rules. Rule interactions can make representations very complex. Therefore caution is needed in implementing such systems.

Closed World Assumption

Another form of assumption, made with regard to incomplete knowledge, is more global in nature than single defaults. This type of assumption is useful in applications where most of the facts are known, and it is, therefore, reasonable to assume that if a proposition cannot be proven, it is false. This is known as the closed world assumption (CWA) with failure as negation (failure to prove a statement F results in assuming its negation, $\neg F$). This means that in a KB if the ground literal $P(a)$ is not provable, then $\neg P(a)$ is assumed to hold.

Closed World Assumption

- **Closed World Assumption** is an approach for dealing with incompleteness by assuming that, anything that is not contained within the knowledge base is false.
- Let us consider another example where the knowledge base consists of 5 cities, A, B, C, D and E, and the following information is available:
  - City A is connected to city B
  - City B is connected to city C
  - City C is connected to city D
  - City D is connected to city E
- Then the extended knowledge base will consists of statements that mention which is not connected to which city.
- Is city A connected to city E? → No!
- Our extended database says that, city A is not connected to city E.
- This would be the Non-Monotonic Inference.

Limitations of the Closed World Assumption

1. Knowledge is not realistically "closable".
   - For example in the ABC Murder story W
     - A or B or C committed the crime
     - A did not commit the crime
     - B did not commit the crime
   - This was our closed world. C's proof was not present in the knowledge base.
2. Syntax problem: Conjunction can be handled by CWA. But not Disjunction.

Knowledge Base: single(Jim) ∨ single(Pam)	
Is Jim single? →The answer cannot be yes	Is Pam single? →The answer cannot be yes
¬single(Jim)	¬single(Pam)

   - So the extended database would look like this
     - Single(Jim) V Single(Pam)
     - ¬ single(Jim)
     - ¬ single(Pam)
   - This extended data base is inconsistent!
   - This example shows that **Disjunctions cannot be handled by the Closed World Assumption.**

Probabilistic reasoning:

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

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

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

Need of probabilistic reasoning in AI:

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.

- When an unknown error occurs during an experiment.

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

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

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

Probability: Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

1. $0 \leq P(A) \leq 1$, where P(A) is the probability of an event A.
1. $P(A) = 0$, indicates total uncertainty in an event A.
1. $P(A) = 1$, indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

$$\text{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

- $P(\neg A)$ = probability of a not happening event.
- $P(\neg A) + P(A) = 1$.

Event: Each possible outcome of a variable is called an event.

Sample space: The collection of all possible events is called sample space.

Random variables: Random variables are used to represent the events and objects in the real world.

Prior probability: The prior probability of an event is probability computed before observing new information.

Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

Conditional probability:

Conditional probability is a probability of occurring an event when another event has already happened.

Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

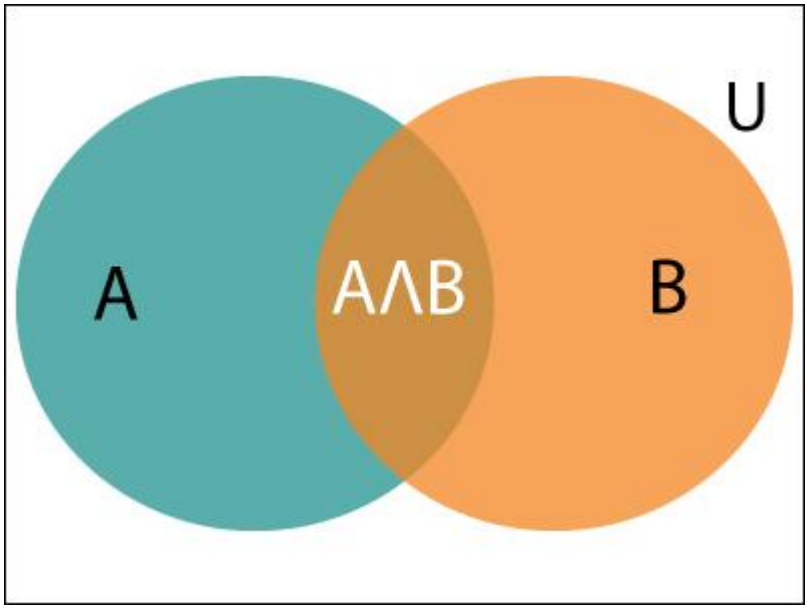
Where P($A \wedge B$)= Joint probability of a and B

P(B)= Marginal probability of B.

If the probability of A is given and we need to find the probability of B, then it will be given as:

$$P(B|A) = \frac{P(A \wedge B)}{P(A)}$$

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of **P($A \wedge B$) by P(B)**.



Example:

In a class, there are 70% of the students who like English and 40% of the students who likes English and mathematics, and then what is the percent of students those who like English also like mathematics?

Solution:

Let, A is an event that a student likes Mathematics

B is an event that a student likes English.

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

Hence, 57% are the students who like English also like Mathematics.

Bayes' theorem in Artificial intelligence

Bayes' theorem:

Bayes' theorem is also known as **Bayes' rule, Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.

In probability theory, it relates the conditional probability and marginal probabilities of two random events.

Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.

It is a way to calculate the value of P(B|A) with the knowledge of P(A|B).

Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.

Example: If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.

Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:

As from product rule we can write:

1. $P(A \wedge B) = P(A|B)\, P(B)$ or

Similarly, the probability of event B with known event A:

1. $P(A \wedge B) = P(B|A)\, P(A)$

Equating right hand side of both the equations, we will get:

$$P(A|B) = \frac{P(B|A)\, P(A)}{P(B)} \quad \text{....(a)}$$

The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.

It shows the simple relationship between joint and conditional probabilities. Here,

P(A|B) is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.

P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.

P(A) is called the **prior probability**, probability of hypothesis before considering the evidence

P(B) is called **marginal probability**, pure probability of an evidence.

In the equation (a), in general, we can write P (B) = P(A)*P(B|Ai), hence the Bayes' rule can be written as:

$$P(A_i|B) = \frac{P(A_i)*P(B|A_i)}{\sum_{i=1}^{k} P(A_i)*P(B|A_i)}$$

Where A_1, A_2, A_3,....., A_n is a set of mutually exclusive and exhaustive events.

Applying Bayes' rule:

Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(*B*), and P(A). This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one. Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

$$P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause})\,P(\text{cause})}{P(\text{effect})}$$

Example-1:

Question: what is the probability that a patient has diseases meningitis with a stiff neck?

Given Data:

A doctor is aware that disease meningitis causes a patient to have a stiff neck, and it occurs 80% of the time. He is also aware of some more facts, which are given as follows:

- The Known probability that a patient has meningitis disease is 1/30,000.
- The Known probability that a patient has a stiff neck is 2%.

Let a be the proposition that patient has stiff neck and b be the proposition that patient has meningitis. , so we can calculate the following as:

P(a|b) = 0.8

P(b) = 1/30000

P(a)= .02

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)} = \frac{0.8*(\frac{1}{30000})}{0.02} = 0.001333333.$$

Hence, we can assume that 1 patient out of 750 patients has meningitis disease with a stiff neck.

Example-2:

Question: From a standard deck of playing cards, a single card is drawn. The probability that the card is king is 4/52, then calculate posterior probability P(King|Face), which means the drawn face card is a king card.

Solution:

$$P(king|face) = \frac{P(Face|king)*P(King)}{P(Face)} \quad .....(i)$$

P(king): probability that the card is King= 4/52= 1/13

P(face): probability that a card is a face card= 3/13

P(Face|King): probability of face card when we assume it is a king = 1

Putting all values in equation (i) we will get:

$$P(king|face) = \frac{1*(\frac{1}{13})}{(\frac{3}{13})} = 1/3,$$ **it is a probability that a face card is a king card.**

Application of Bayes' theorem in Artificial intelligence:

Following are some applications of Bayes' theorem:

- It is used to calculate the next step of the robot when the already executed step is given.
- Bayes' theorem is helpful in weather forecasting.
- It can solve the Monty Hall problem.

Bayesian Belief Network in artificial intelligence

Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as:

"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**.

Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

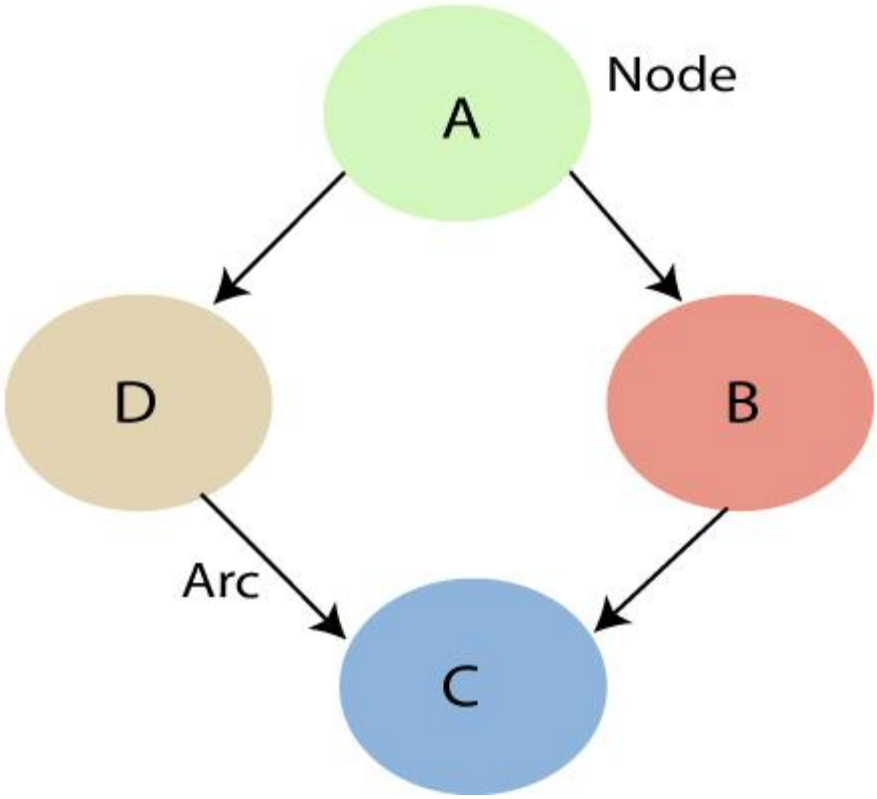
Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- **Directed Acyclic Graph**
- **Table of conditional probabilities.**

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram**.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:



- Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.
- **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph.
  These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other
  - **In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.**
  - **If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.**
  - **Node C is independent of node A.**

> *Note: The Bayesian network graph does not contain any cyclic graph. Hence, it is known as a **directed acyclic graph or DAG**.*

The Bayesian network has mainly two components:

- **Causal Component**

- **Actual numbers**

Each node in the Bayesian network has condition probability distribution **$P(X_i | Parent(X_i))$**, which determines the effect of the parent on that node.

Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables x1, x2, x3,....., xn, then the probabilities of a different combination of x1, x2, x3.. xn, are known as Joint probability distribution.

$P[x_1, x_2, x_3,....., x_n]$, it can be written as the following way in terms of the joint probability distribution.

$= P[x_1 | x_2, x_3,....., x_n]P[x_2, x_3,....., x_n]$

$= P[x_1 | x_2, x_3,....., x_n]P[x_2|x_3,....., x_n]....P[x_{n-1}|x_n]P[x_n].$

In general for each variable Xi, we can write the equation as:

$$P(X_i|X_{i-1},....., X_1) = P(X_i | Parents(X_i))$$

Explanation of Bayesian network:

Let's understand the Bayesian network through an example by creating a directed acyclic graph:

Example: Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

Problem:

Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.

Solution:

- The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly

affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.

- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- The conditional distributions for each node are given as conditional probabilities table or CPT.
- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
- In CPT, a boolean variable with k boolean parents contains 2^K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

List of all events occurring in this network:

- **Burglary (B)**
- **Earthquake(E)**
- **Alarm(A)**
- **David Calls(D)**
- **Sophia calls(S)**

We can write the events of problem statement in the form of probability: **P[D, S, A, B, E]**, can rewrite the above probability statement using joint probability distribution:

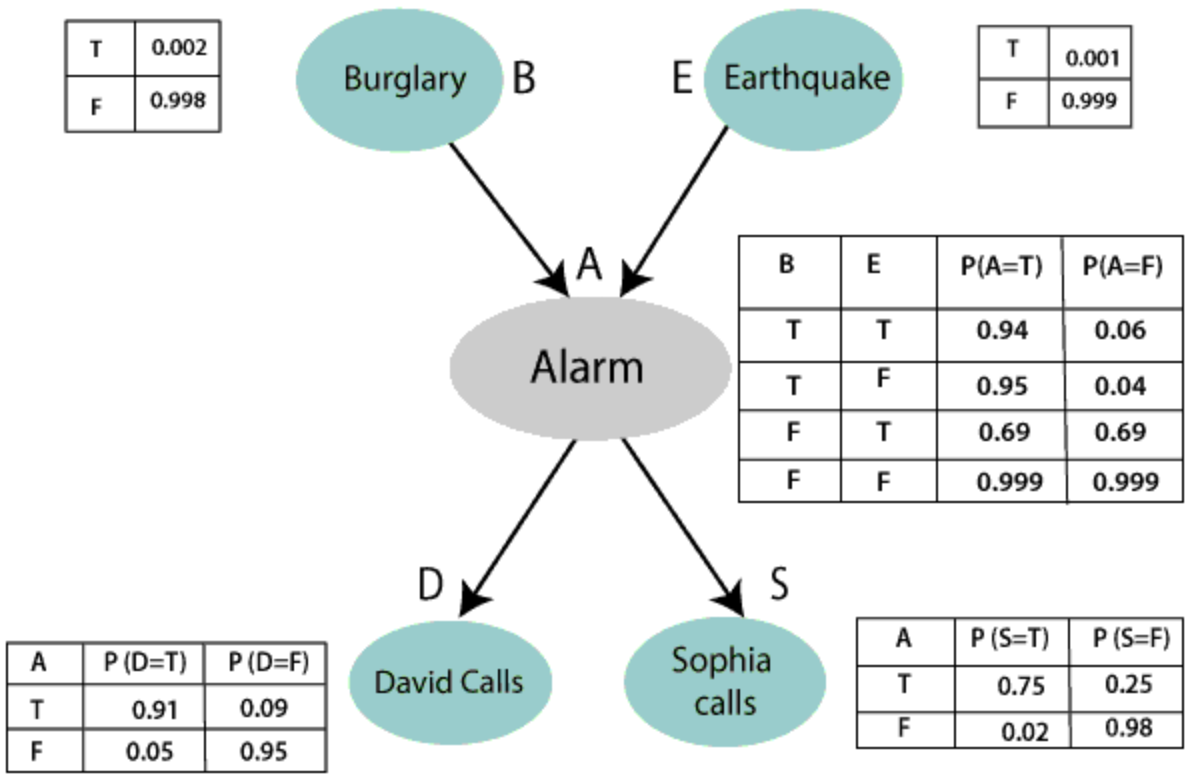
P[D, S, A, B, E]= P[D | S, A, B, E]. P[S, A, B, E]

=P[D | S, A, B, E]. P[S | A, B, E]. P[A, B, E]

= P [D| A]. P [S| A, B, E]. P[A, B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B |E]. P[E]



Let's take the observed probability for the Burglary and earthquake component:

P(B= True) = 0.002, which is the probability of burglary.

P(B= False)= 0.998, which is the probability of no burglary.

P(E= True)= 0.001, which is the probability of a minor earthquake

P(E= False)= 0.999, Which is the probability that an earthquake not occurred.

We can provide the conditional probabilities as per the below tables:

Conditional probability table for Alarm A:

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	E	P(A= True)	P(A= False)
True	True	0.94	0.06
True	False	0.95	0.04

False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

P(S, D, A, ¬B, ¬E) = P (S|A) *P (D|A)*P (A|¬B ^ ¬E) *P (¬B) *P (¬E).

= 0.75* 0.91* 0.001* 0.998*0.999

= 0.00068045.

Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

The semantics of Bayesian Network:

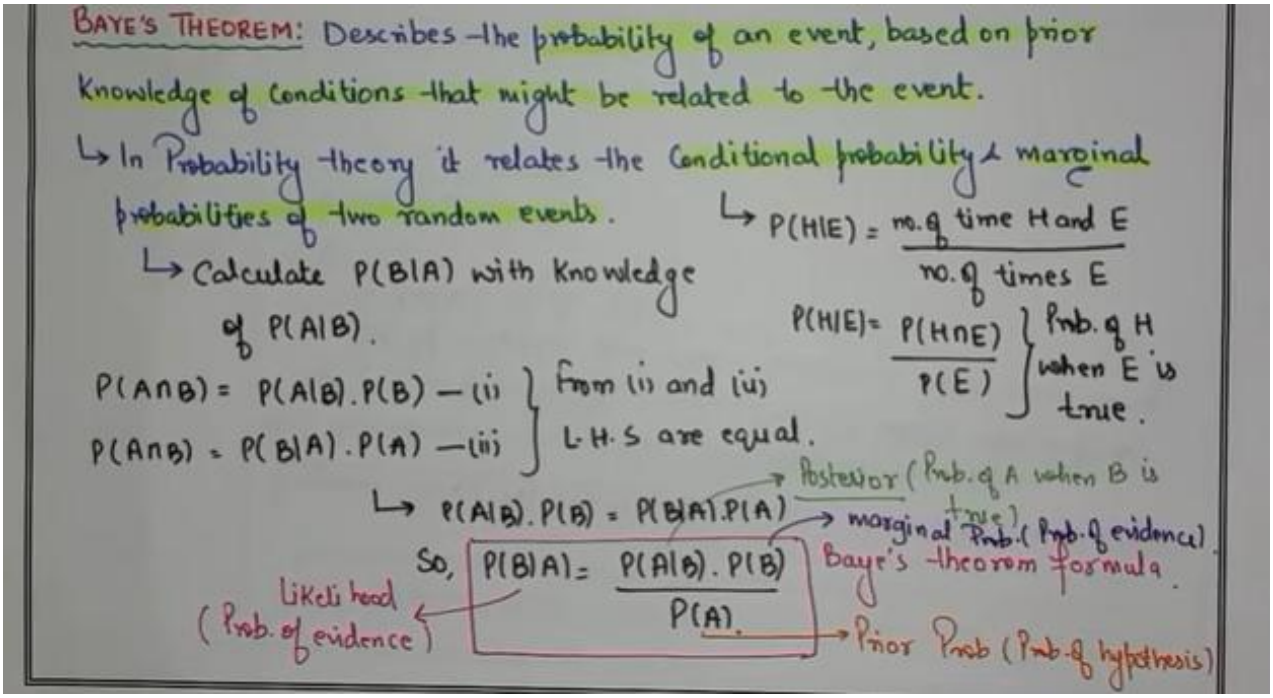
There are two ways to understand the semantics of the Bayesian network, which is given below:

1. To understand the network as the representation of the Joint probability distribution.

It is helpful to understand how to construct the network.

2. To understand the network as an encoding of a collection of conditional independence statements.

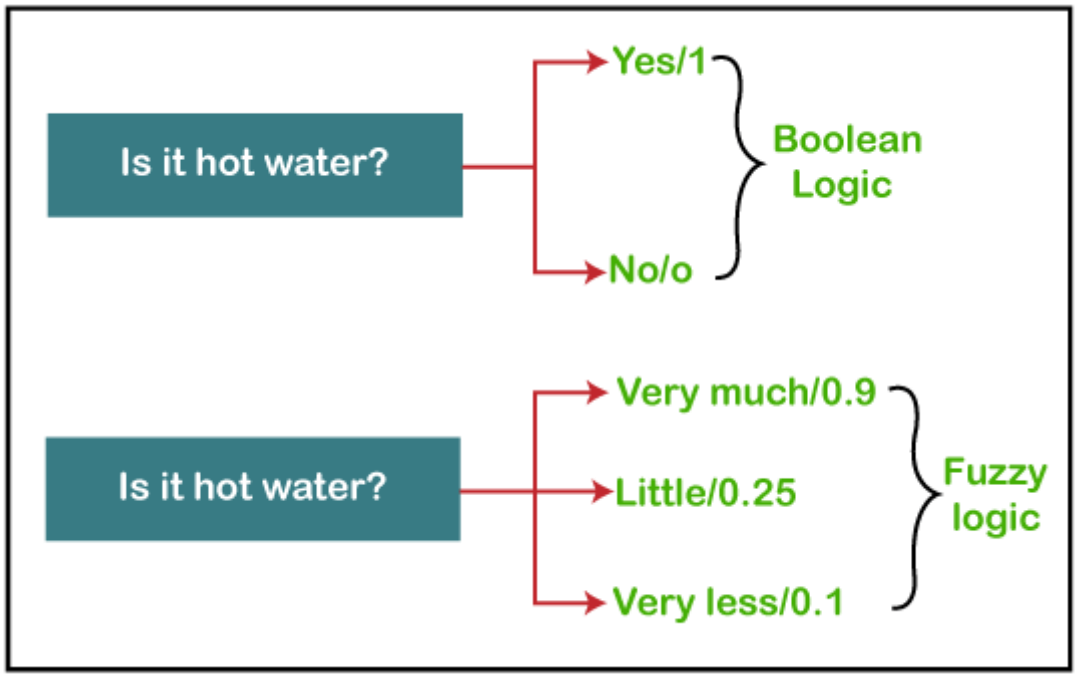
It is helpful in designing inference procedure.



What is Fuzzy Logic?

The '**Fuzzy**' word means the things that are not clear or are vague. Sometimes, we cannot decide in real life that the given problem or statement is either true or false. At that time, this concept provides many values between the true and false and gives the flexibility to find the best solution to that problem.

Example of Fuzzy Logic as comparing to Boolean Logic



Fuzzy logic contains the multiple logical values and these values are the truth values of a variable or problem between 0 and 1. This concept was introduced by **Lotfi Zadeh** in **1965** based on the **Fuzzy Set Theory**. This concept provides the possibilities which are not given by computers, but similar to the range of possibilities generated by humans.

In the Boolean system, only two possibilities (0 and 1) exist, where 1 denotes the absolute truth value and 0 denotes the absolute false value. But in the fuzzy system, there are multiple possibilities present between the 0 and 1, which are partially false and partially true.

CERTAINLY YES
POSSIBLY YES
CANNOT SAY
POSSIBLY NO

CERTAINLY NO

Implementation

- It can be implemented in systems with various sizes and capabilities ranging from small micro-controllers to large, networked, workstation-based control systems.
- It can be implemented in hardware, software, or a combination of both.

Why do we use Fuzzy Logic?

Generally, we use the fuzzy logic system for both commercial and practical purposes such as:

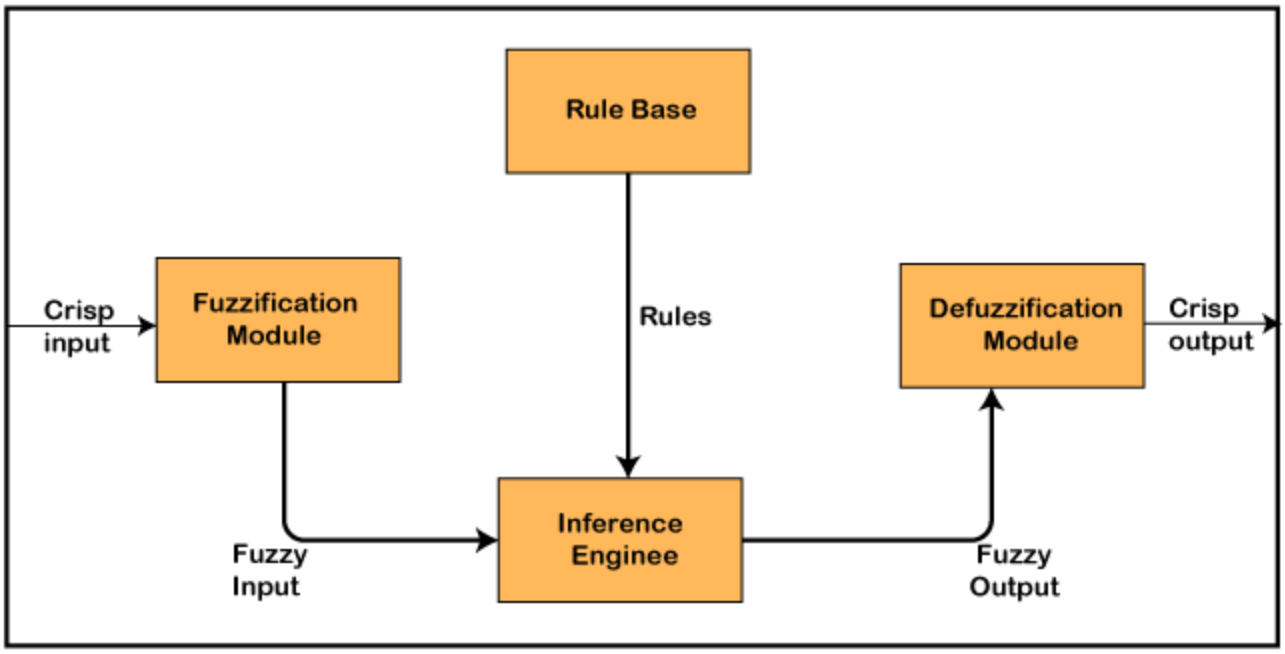
- It **controls machines** and **consumer products**
- If not accurate reasoning, it at least provides **acceptable reasoning**
- This helps in dealing with the **uncertainty in engineering**

Architecture of a Fuzzy Logic System

In the architecture of the **Fuzzy Logic** system, each component plays an important role. The architecture consists of the different four components which are given below.

1. Rule Base
2. Fuzzification
3. Inference Engine
4. Defuzzification

Following diagram shows the architecture or process of a Fuzzy Logic system:



1. Rule Base

Rule Base is a component used for storing the set of rules and the If-Then conditions given by the experts are used for controlling the decision-making systems. There are so many updates that come in the Fuzzy theory recently, which offers effective methods for designing and tuning of fuzzy controllers. These updates or developments decreases the number of fuzzy set of rules.

2. Fuzzification

Fuzzification is a module or component for transforming the system inputs, i.e., it converts the crisp number into fuzzy steps. The crisp numbers are those inputs which are measured by the sensors and then fuzzification passed them into the control systems for further processing. This component divides the input signals into following five states in any Fuzzy Logic system:

- Large Positive (LP)
- Medium Positive (MP)
- Small (S)
- Medium Negative (MN)
- Large negative (LN)

3. Inference Engine

This component is a main component in any Fuzzy Logic system (FLS), because all the information is processed in the Inference Engine. It allows users to find the matching degree between the current fuzzy input and the rules. After the matching degree, this system determines which rule is to be added according to the given input field. When all rules are fired, then they are combined for developing the control actions.

4. Defuzzification

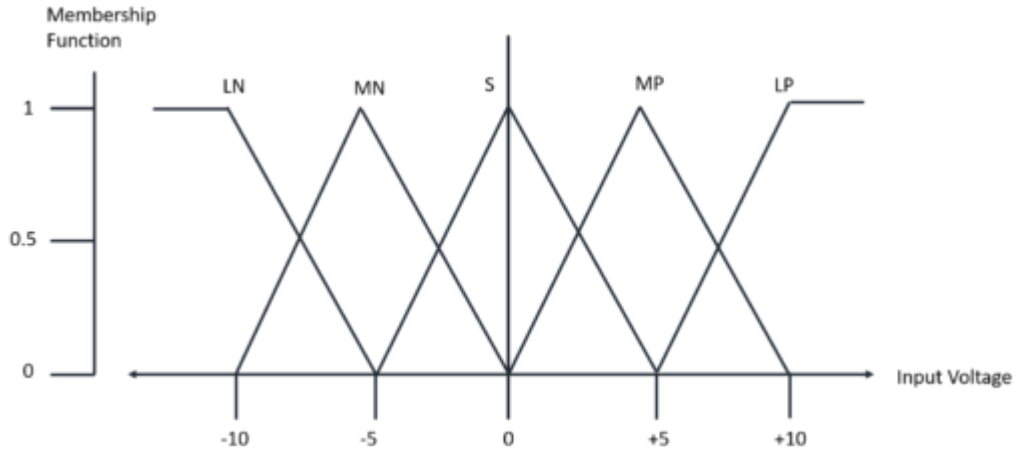
Defuzzification is a module or component, which takes the fuzzy set inputs generated by the **Inference Engine**, and then transforms them into a crisp value. It is the last step in the process of a fuzzy logic system. The crisp value is a type of value which is acceptable by the user. Various techniques are present to do this, but the user has to select the best one for reducing the errors.

Membership Function

The membership function is a function which represents the graph of fuzzy sets, and allows users to quantify the linguistic term. It is a graph which is used for mapping each element of x to the value between 0 and 1.

This function is also known as indicator or characteristics function.

This function of Membership was introduced in the first papers of fuzzy set by **Zadeh**. For the Fuzzy set B, the membership function for X is defined as: $\mu B:X \rightarrow [0,1]$. In this function X, each element of set B is mapped to the value between 0 and 1. This is called a degree of membership or membership value.

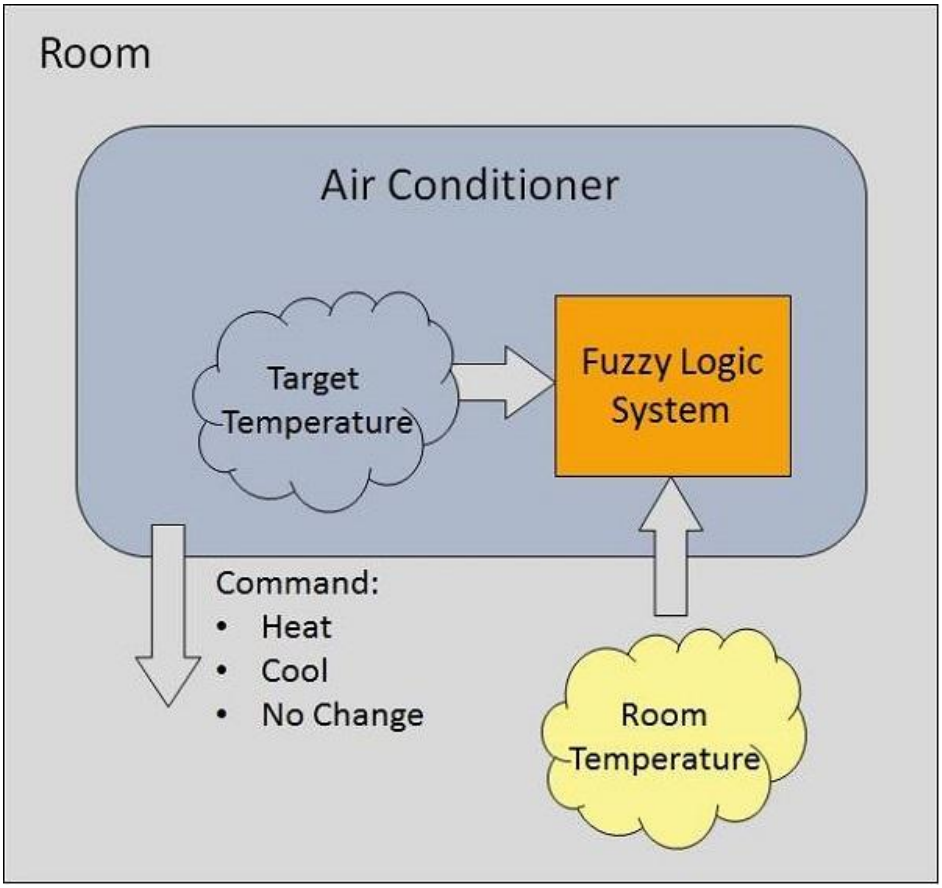


The triangular membership function shapes are most common among various other membership function shapes such as trapezoidal, singleton, and Gaussian.

Here, the input to 5-level fuzzifier varies from -10 volts to +10 volts. Hence the corresponding output also changes.

Example of a Fuzzy Logic System

Let us consider an air conditioning system with 5-level fuzzy logic system. This system adjusts the temperature of air conditioner by comparing the room temperature and the target temperature value.



Algorithm

- Define linguistic Variables and terms (start)
- Construct membership functions for them. (start)
- Construct knowledge base of rules (start)
- Convert crisp data into fuzzy data sets using membership functions. (fuzzification)
- Evaluate rules in the rule base. (Inference Engine)
- Combine results from each rule. (Inference Engine)
- Convert output data into non-fuzzy values. (defuzzification)

Development

Step 1 − Define linguistic variables and terms

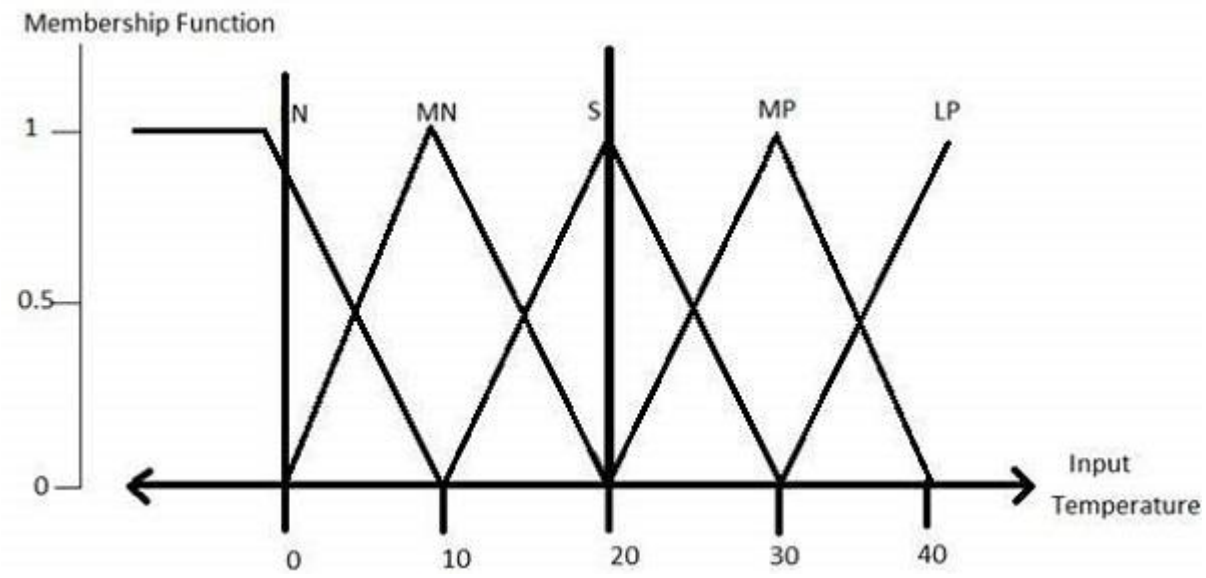
Linguistic variables are input and output variables in the form of simple words or sentences. For room temperature, cold, warm, hot, etc., are linguistic terms.

Temperature (t) = {very-cold, cold, warm, very-warm, hot}

Every member of this set is a linguistic term and it can cover some portion of overall temperature values.

Step 2 − Construct membership functions for them

The membership functions of temperature variable are as shown −



Step3 − Construct knowledge base rules

Create a matrix of room temperature values versus target temperature values that an air conditioning system is expected to provide.

RoomTemp. /Target	Very_Cold	Cold	Warm	Hot	Very_Hot
Very_Cold	No_Change	Heat	Heat	Heat	Heat
Cold	Cool	No_Change	Heat	Heat	Heat
Warm	Cool	Cool	No_Change	Heat	Heat
Hot	Cool	Cool	Cool	No_Change	Heat
Very_Hot	Cool	Cool	Cool	Cool	No_Change

Build a set of rules into the knowledge base in the form of IF-THEN-ELSE structures.

Sr. No.	Condition	Action

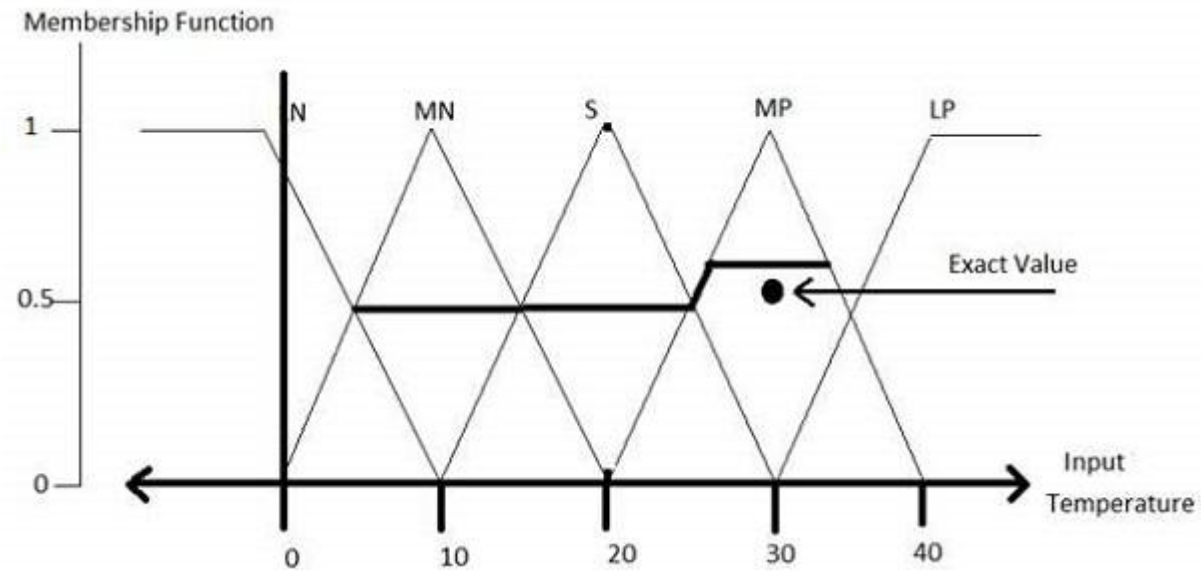
1	IF temperature=(Cold OR Very_Cold) AND target=Warm THEN	Heat
2	IF temperature=(Hot OR Very_Hot) AND target=Warm THEN	Cool
3	IF (temperature=Warm) AND (target=Warm) THEN	No_Change

Step 4 − Obtain fuzzy value

Fuzzy set operations perform evaluation of rules. The operations used for OR and AND are Max and Min respectively. Combine all results of evaluation to form a final result. This result is a fuzzy value.

Step 5 − Perform defuzzification

Defuzzification is then performed according to membership function for output variable.



Application Areas of Fuzzy Logic

The key application areas of fuzzy logic are as given −

Automotive Systems

- Automatic Gearboxes
- Four-Wheel Steering
- Vehicle environment control

Consumer Electronic Goods

- Hi-Fi Systems
- Photocopiers
- Still and Video Cameras

- Television

Domestic Goods

- Microwave Ovens
- Refrigerators
- Toasters
- Vacuum Cleaners
- Washing Machines

Environment Control

- Air Conditioners/Dryers/Heaters
- Humidifiers

Classical and Fuzzy Set Theory

To learn about classical and Fuzzy set theory, firstly you have to know about what is set.

Set

A set is a term, which is a collection of unordered or ordered elements. Following are the various examples of a set:

1. A set of all-natural numbers
2. A set of students in a class.
3. A set of all cities in a state.
4. A set of upper-case letters of the alphabet.

Types of Set:

There are following various categories of set:

1. Finite
2. Empty
3. Infinite
4. Proper
5. Universal

6. Subset
7. Singleton
8. Equivalent Set
9. Disjoint Set

Classical Set

It is a type of set which collects the distinct objects in a group. The sets with the crisp boundaries are classical sets. In any set, each single entity is called an element or member of that set.

Mathematical Representation of Sets

Any set can be easily denoted in the following two different ways:

1. Roaster Form: This is also called as a tabular form. In this form, the set is represented in the following way:

```
Set_name = { element1, element2, element3, ....., element N}
```

The elements in the set are enclosed within the brackets and separated by the commas.

Following are the two examples which describes the set in Roaster or Tabular form:

Example 1:

```
Set of Natural Numbers: N={1, 2, 3, 4, 5, 6, 7, .....,n).
```

Example 2:

```
Set of Prime Numbers less than 50: X={2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47}.
```

2. Set Builder Form: Set Builder form defines a set with the common properties of an element in a set. In this form, the set is represented in the following way:

```
A = {x:p(x)}
```

The following example describes the set in the builder form:

```
The set {2, 4, 6, 8, 10, 12, 14, 16, 18} is written as:
B = {x:2 ≤ x < 20 and (x%2) = 0}
```

Operations on Classical Set

Following are the various operations which are performed on the classical sets:

1. Union Operation
2. Intersection Operation
3. Difference Operation
4. Complement Operation

1. Union:

This operation is denoted by (A U B). A U B is the set of those elements which exist in two different sets A and B. This operation combines all the elements from both the sets and make a new set. It is also called a Logical OR operation.

It can be described as:

$A \cup B = \{ x \mid x \in A \text{ OR } x \in B \}$.

Example:

Set A = {10, 11, 12, 13}, Set B = {11, 12, 13, 14, 15}, then $A \cup B$ = {10, 11, 12, 13, 14, 15}

2. Intersection

This operation is denoted by ($A \cap B$). $A \cap B$ is the set of those elements which are common in both set A and B. It is also called a Logical OR operation.

It can be described as:

$A \cap B = \{ x \mid x \in A \text{ AND } x \in B \}$.

Example:

Set A = {10, 11, 12, 13}, Set B = {11, 12, 14} then $A \cap B$ = {11, 12}

3. Difference Operation

This operation is denoted by (A - B). A-B is the set of only those elements which exist only in set A but not in set B.

It can be described as:

$A - B = \{ x \mid x \in A \text{ AND } x \notin B \}$.

4. Complement Operation: This operation is denoted by (A`). It is applied on a single set. A` is the set of elements which do not exist in set A.

It can be described as:

$A' = \{x|x \notin A\}$.

Properties of Classical Set

There are following various properties which play an essential role for finding the solution of a fuzzy logic problem.

1. Commutative Property:

This property provides the following two states which are obtained by two finite sets A and B:

$$A \cup B = B \cup A$$
$$A \cap B = B \cap A$$

2. Associative Property:

This property also provides the following two states but these are obtained by three different finite sets A, B, and C:

$$A \cup (B \cup C) = (A \cup B) \cup C$$
$$A \cap (B \cap C) = (A \cap B) \cap C$$

3. Idempotency Property:

This property also provides the following two states but for a single finite set A:

$$A \cup A = A$$
$$A \cap A = A$$

4. Absorption Property

This property also provides the following two states for any two finite sets A and B:

$$A \cup (A \cap B) = A$$
$$A \cap (A \cup B) = A$$

5. Distributive Property:

This property also provides the following two states for any three finite sets A, B, and C:

A∪ (B ∩ C) = (A ∪ B)∩ (A ∪ C)

A∩ (B ∪ C) = (A∩B) ∪ (A∩C)

6. Identity Property:

This property provides the following four states for any finite set A and Universal set X:

A ∪ φ =A

A ∩ X = A

A ∩ φ = φ

A ∪ X = X

7. Transitive property

This property provides the following state for the finite sets A, B, and C:

If A ⊆ B ⊆ C, then A ⊆ C

8. Ivolution property

This property provides following state for any finite set A:

$$\overline{\overline{A}} = A$$

9. De Morgan's Law

This law gives the following rules for providing the contradiction and tautologies:

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

Fuzzy Set

The set theory of classical is the subset of Fuzzy set theory. Fuzzy logic is based on this theory, which is a generalisation of the classical theory of set (i.e., crisp set) introduced by Zadeh in 1965.

A fuzzy set is a collection of values which exist between 0 and 1. Fuzzy sets are denoted or represented by the tilde (~) character. The sets of Fuzzy theory were introduced in 1965 by Lofti A. Zadeh and Dieter Klaua. In the fuzzy set, the partial membership also exists. This theory released as an extension of classical set theory.

This theory is denoted mathematically as A fuzzy set (Ã) is a pair of U and M, where U is the Universe of discourse and M is the membership function which takes on values in the interval [0, 1]. The universe of discourse (U) is also denoted by Ω or X.

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) | x \in X\}$$

Operations on Fuzzy Set

Given Ã and B are the two fuzzy sets, and X be the universe of discourse with the following respective member functions:

$$\mu_{\tilde{A}}(x) \text{ and } \mu_{\tilde{B}}(x)$$

The operations of Fuzzy set are as follows:

1. Union Operation: The union operation of a fuzzy set is defined by:

$\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$

Example:

Let's suppose A is a set which contains following elements:

$A = \{(X_1, 0.6), (X_2, 0.2), (X_3, 1), (X_4, 0.4)\}$

And, B is a set which contains following elements:

$B = \{(X_1, 0.1), (X_2, 0.8), (X_3, 0), (X_4, 0.9)\}$

then,

$A \cup B = \{(X_1, 0.6), (X_2, 0.8), (X_3, 1), (X_4, 0.9)\}$

Because, according to this operation

For X_1

$\mu_{A \cup B}(X_1) = \max(\mu_A(X_1), \mu_B(X_1))$

$\mu_{A \cup B}(X_1) = \max(0.6, 0.1)$

$\mu_{A \cup B}(X_1) = 0.6$

For X_2

$\mu_{A\cup B}(X_2) = \max(\mu_A(X_2), \mu_B(X_2))$

$\mu_{A\cup B}(X_2) = \max(0.2, 0.8)$

$\mu_{A\cup B}(X_2) = 0.8$

For X_3

$\mu_{A\cup B}(X_3) = \max(\mu_A(X_3), \mu_B(X_3))$

$\mu_{A\cup B}(X_3) = \max(1, 0)$

$\mu_{A\cup B}(X_3) = 1$

For X_4

$\mu_{A\cup B}(X_4) = \max(\mu_A(X_4), \mu_B(X_4))$

$\mu_{A\cup B}(X_4) = \max(0.4, 0.9)$

$\mu_{A\cup B}(X_4) = 0.9$

2. Intersection Operation: The intersection operation of fuzzy set is defined by:

$\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x))$

Example:

Let's suppose A is a set which contains following elements:

$A = \{(X_1, 0.3), (X_2, 0.7), (X_3, 0.5), (X_4, 0.1)\}$

And, B is a set which contains following elements:

$B = \{(X_1, 0.8), (X_2, 0.2), (X_3, 0.4), (X_4, 0.9)\}$

then,

$A\cap B = \{(X_1, 0.3), (X_2, 0.2), (X_3, 0.4), (X_4, 0.1)\}$

Because, according to this operation

For X_1

$\mu_{A\cap B}(X_1) = \min(\mu_A(X_1), \mu_B(X_1))$

$\mu_{A\cap B}(X_1) = \min(0.3, 0.8)$

$\mu_{A\cap B}(X_1) = 0.3$

For X_2

$\mu_{A\cap B}(X_2) = \min(\mu_A(X_2), \mu_B(X_2))$

$\mu_{A\cap B}(X_2) = \min(0.7, 0.2)$

$\mu_{A\cap B}(X_2) = 0.2$

For X_3

$\mu_{A\cap B}(X_3) = \min(\mu_A(X_3), \mu_B(X_3))$

$\mu_{A\cap B}(X_3) = \min(0.5, 0.4)$

$\mu_{A\cap B}(X_3) = 0.4$

For X_4

$\mu_{A\cap B}(X_4) = \min(\mu_A(X_4), \mu_B(X_4))$

$\mu_{A\cap B}(X_4) = \min(0.1, 0.9)$

$\mu_{A\cap B}(X_4) = 0.1$

3. Complement Operation: The complement operation of fuzzy set is defined by:

$\mu_{\bar{A}}(x) = 1-\mu_A(x)$,

Example:

Let's suppose A is a set which contains following elements:

$A = \{(X_1, 0.3), (X_2, 0.8), (X_3, 0.5), (X_4, 0.1)\}$

then,

$\bar{A} = \{(X_1, 0.7), (X_2, 0.2), (X_3, 0.5), (X_4, 0.9)\}$

Because, according to this operation

For X_1

$\mu_{\bar{A}}(X_1) = 1-\mu_A(X_1)$

$\mu_{\bar{A}}(X_1) = 1 - 0.3$

$\mu_{\bar{A}}(X_1) = 0.7$

For X_2

$\mu_{\bar{A}}(X_2) = 1-\mu_A(X_2)$

$\mu_{\bar{A}}(X_2) = 1 - 0.8$

$\mu_{\bar{A}}(X_2) = 0.2$

For X_3

$\mu_{\bar{A}}(X_3) = 1-\mu_A(X_3)$

$\mu_{\bar{A}}(X_3) = 1 - 0.5$

$\mu_{\bar{A}}(X_3) = 0.5$

For X_4

$\mu_{\bar{A}}(X_4) = 1-\mu_A(X_4)$

$\mu_{\bar{A}}(X_4) = 1 - 0.1$

$\mu_{\bar{A}}(X_4) = 0.9$

Classical Set Theory	Fuzzy Set Theory
1. This theory is a class of those sets having sharp boundaries.	1. This theory is a class of those sets having un-sharp boundaries.
2. This set theory is defined by exact boundaries only 0 and 1.	2. This set theory is defined by ambiguous boundaries.
3. In this theory, there is no uncertainty about the boundary's location of a set.	3. In this theory, there always exists uncertainty about the boundary's location of a set.
4. This theory is widely used in the design of digital systems.	4. It is mainly used for fuzzy controllers.

Applications of Fuzzy Logic

Following are the different application areas where the Fuzzy Logic concept is widely used:

1. It is used in **Businesses** for decision-making support system.

2. It is used in **Automative systems** for controlling the traffic and speed, and for improving the efficiency of automatic transmissions. **Automative systems** also use the shift scheduling method for automatic transmissions.
3. This concept is also used in the **Defence** in various areas. Defence mainly uses the Fuzzy logic systems for underwater target recognition and the automatic target recognition of thermal infrared images.
4. It is also widely used in the **Pattern Recognition and Classification** in the form of Fuzzy logic-based recognition and handwriting recognition. It is also used in the searching of fuzzy images.
5. Fuzzy logic systems also used in **Securities**.
6. It is also used in **microwave oven** for setting the lunes power and cooking strategy.
7. This technique is also used in the area of **modern control systems** such as expert systems.
8. **Finance** is also another application where this concept is used for predicting the stock market, and for managing the funds.
9. It is also used for controlling the brakes.
10. It is also used in the **industries of chemicals** for controlling the ph, and chemical distillation process.
11. It is also used in the **industries of manufacturing** for the optimization of milk and cheese production.
12. It is also used in the vacuum cleaners, and the timings of washing machines.
13. It is also used in heaters, air conditioners, and humidifiers.

Advantages of Fuzzy Logic

Fuzzy Logic has various advantages or benefits. Some of them are as follows:

1. The methodology of this concept works similarly as the human reasoning.
2. Any user can easily understand the structure of Fuzzy Logic.
3. It does not need a large memory, because the algorithms can be easily described with fewer data.

4. It is widely used in all fields of life and easily provides effective solutions to the problems which have high complexity.
5. This concept is based on the set theory of mathematics, so that's why it is simple.
6. It allows users for controlling the control machines and consumer products.
7. The development time of fuzzy logic is short as compared to conventional methods.
8. Due to its flexibility, any user can easily add and delete rules in the FLS system.

Disadvantages of Fuzzy Logic

Fuzzy Logic has various disadvantages or limitations. Some of them are as follows:

1. The run time of fuzzy logic systems is slow and takes a long time to produce outputs.
2. It **cannot recognize** machine learning as-well-as neural network type patterns
3. The possibilities produced by the fuzzy logic system are not always accurate.
4. Fuzzy logics are not suitable for those problems that require high accuracy.
5. The systems of a Fuzzy logic need a lot of testing for verification and validation.
6. Setting exact, fuzzy rules and, membership functions is a **difficult task**

Unit-4

PROBLEM CHARACTERISTICS

A problem may have different aspects of representation and explanation. In order to choose the most appropriate method for a particular problem, it is necessary to analyze the problem along several key dimensions. Some of the main key features of a problem are given below.

Is the problem decomposable into set of sub problems?

Can the solution step be ignored or undone?

Is the problem universally predictable?

Is a good solution to the problem obvious without comparison to all the possible solutions? Is the desire solution a state of world or a path to a state?

Is a large amount of knowledge absolutely required to solve the problem?

Will the solution of the problem required interaction between the computer and the person?

The above characteristics of a problem are called as 7-problem characteristics under which the solution must take place.

What is Production System in Artificial Intelligence?

A production system is based on a set of rules about behavior. These rules are a basic representation found helpful in expert systems, automated planning, and action selection.

Production system or production rule system is a computer program typically used to provide some form of artificial intelligence, which consists primarily of a set of rules about behavior but it also includes the mechanism necessary to follow those rules as the system responds to states of the world.

Components of Production System

The major components of Production System in Artificial Intelligence are:

- **Global Database:** The global database is the central data structure used by the production system in Artificial Intelligence.
- **Set of Production Rules:** The production rules operate on the global database. Each rule usually has a precondition that is either satisfied or not by the global database. If the precondition is satisfied, the rule is usually be applied. The application of the rule changes the database.
- **A Control System:** The control system then chooses which applicable rule should be applied and ceases computation when a termination condition on the database is satisfied. If multiple rules are to fire at the same time, the control system resolves the conflicts.

Features of Production System in Artificial Intelligence

The main features of the production system include:

1. Simplicity: The structure of each sentence in a production system is unique and uniform as they use the "IF-THEN" structure. This structure provides simplicity in knowledge representation. This feature of the production system improves the readability of production rules.

2. Modularity: This means the production rule code the knowledge available in discrete pieces. Information can be treated as a collection of independent facts which may be added or deleted from the system with essentially no deleterious side effects.

3. Modifiability: This means the facility for modifying rules. It allows the development of production rules in a skeletal form first and then it is accurate to suit a specific application.

4. Knowledge-intensive: The knowledge base of the production system stores pure knowledge. This part does not contain any type of control or programming information. Each production rule is normally written as an English sentence; the problem of semantics is solved by the very structure of the representation.

Control/Search Strategies

How would you decide which rule to apply while searching for a solution for any problem? There are certain requirements for a good control strategy that you need to keep in mind, such as:

- The first requirement for a good control strategy is that it should **cause motion**.
- The second requirement for a good control strategy is that it should be **systematic**.
- Finally, it must be **efficient** in order to find a good answer.

Production System Rules

Production System rules can be classified as:

- **Deductive Inference Rules**
- **Abductive Inference Rules**

You can represent the knowledge in a production system as a set of rules along with a control system and database. It can be written as:

If(Condition) Then (Condition)

The production rules are also known as condition-action, antecedent-consequent, pattern-action, situation-response, feedback-result pairs.

Classes of Production System in Artificial Intelligence

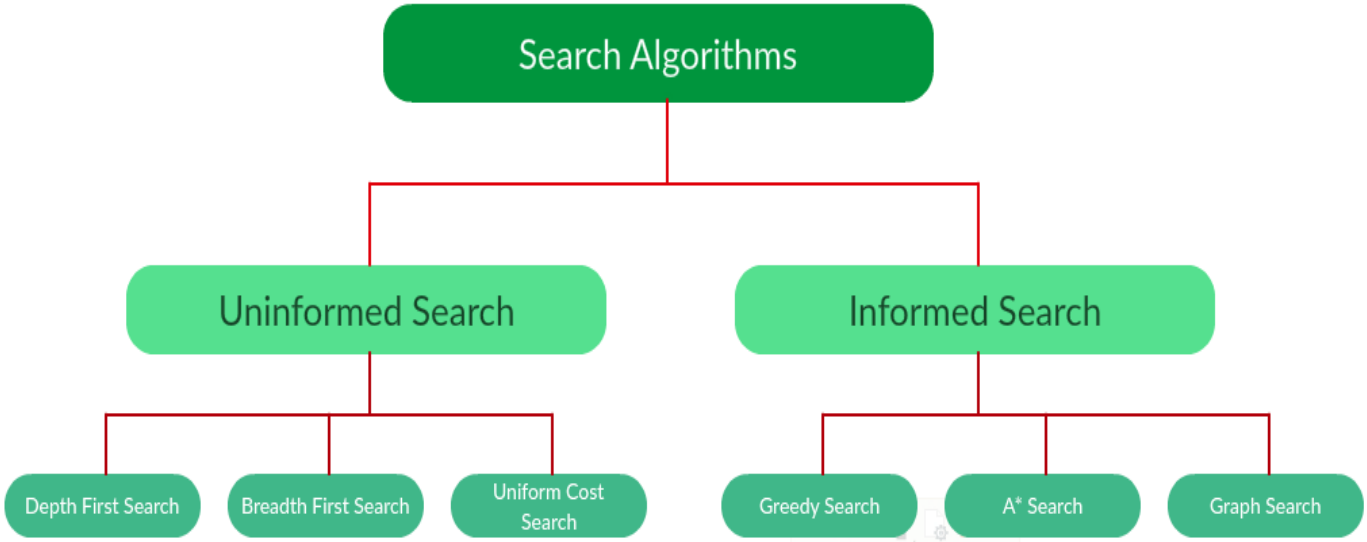
There are four major classes of Production System in Artificial Intelligence:

- **Monotonic Production System**: It's a production system in which the application of a rule never prevents the later application of another rule, that could have also been applied at the time the first rule was selected.
- **Partially Commutative Production System**: It's a type of production system in which the application of a sequence of rules transforms state X into state Y, then any permutation of those rules that is allowable also transforms state x into state Y. Theorem proving falls under the monotonic partially communicative system.
- **Non-Monotonic Production Systems**: These are useful for solving ignorable problems. These systems are important from an implementation standpoint because they can be implemented without the ability to backtrack to previous states when it is discovered that an incorrect path was followed. This production system increases efficiency since it is not necessary to keep track of the changes made in the search process.
- **Commutative Systems**: These are usually useful for problems in which changes occur but can be reversed and in which the order of operation is not critical. Production systems that are not usually not partially commutative are useful for many problems in which irreversible changes occur, such as chemical analysis. When dealing with such systems, the order in which operations are performed is very important and hence correct decisions must be made at the first attempt itself.

- A search problem consists of:
    - **A State Space.** Set of all possible states where you can be.
    - **A Start State.** The state from where the search begins.
    - **A Goal Test.** A function that looks at the current state returns whether or not it is the goal state.
- The **Solution** to a search problem is a sequence of actions, called the **plan** that transforms the start state to the goal state.
- This plan is achieved through search algorithms.

Types of search algorithms:

There are far too many powerful search algorithms out there to fit in a single article. Instead, this article will discuss *six* of the fundamental search algorithms, divided into *two* categories, as shown below.



Uninformed Search Algorithms

Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

Following are the various types of uninformed search algorithms:

1. **Breadth-first Search**
2. **Depth-first Search**

1. Breadth-first Search:

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure.

Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

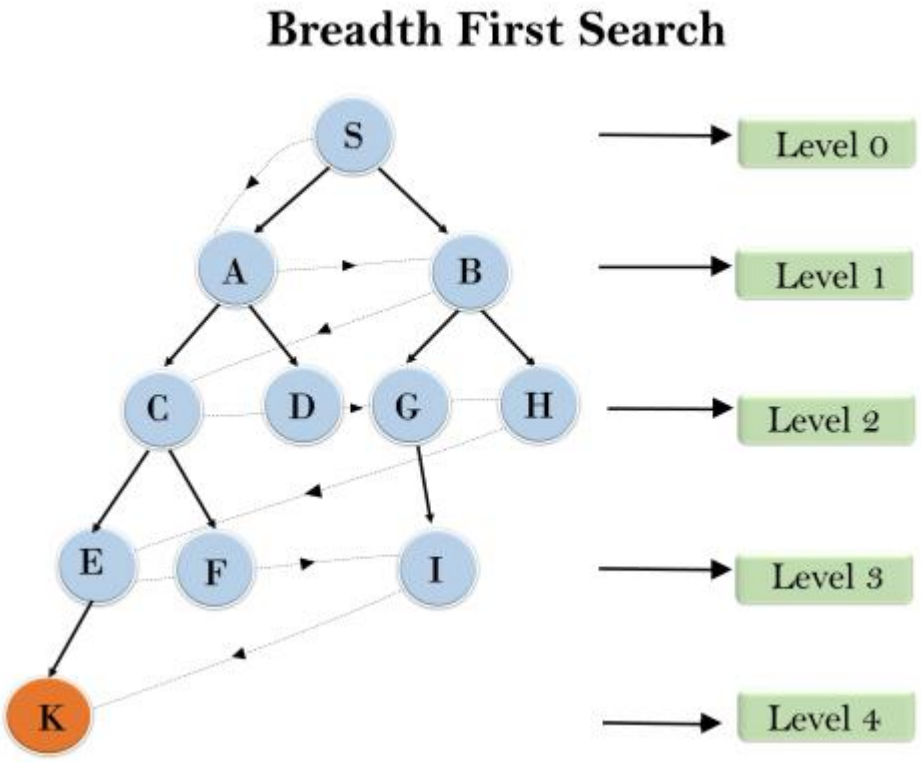
Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Example:

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

1. S---> A--->B----->C--->D----->G--->H--->E----->F----->I----->K



Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

$T(b) = 1+b^2+b^3+.....+ b^d = O(b^d)$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

2. Depth-first Search

- Depth-first search isa recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

> *Note: Backtracking is an algorithm technique for finding all possible solutions using recursion.*

Advantage:

- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage:

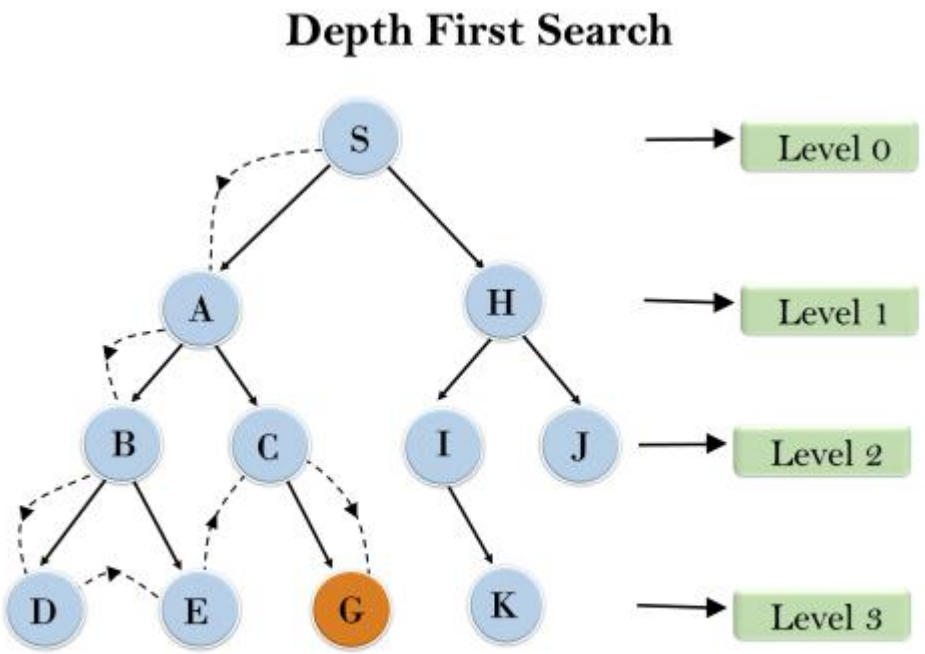
- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

Example:

In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

Root node--->Left node ----> right node.

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$T(n)= 1+ n^2+ n^3 +.....+ n^m=O(n^m)$

Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

Informed Search Algorithms

So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space. But informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.

The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

Heuristics function: Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

Admissibility of the heuristic function is given as:

1. h(n) <= h*(n)

Here h(n) is heuristic cost, and h*(n) is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

Pure Heuristic Search:

Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.

On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues unit a goal state is found.

In the informed search we will discuss two main algorithms which are given below:

- **Best First Search Algorithm(Greedy search)**
- **A* Search Algorithm**

1.) Best-first Search Algorithm (Greedy Search):

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising

node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

1. f(n)= g(n).

   Were, h(n)= estimated cost from node n to the goal.

   The greedy best first algorithm is implemented by the priority queue.

   Best first search algorithm:

   - **Step 1:** Place the starting node into the OPEN list.
   - **Step 2:** If the OPEN list is empty, Stop and return failure.
   - **Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
   - **Step 4:** Expand the node n, and generate the successors of node n.
   - **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
   - **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
   - **Step 7:** Return to Step 2.

   Advantages:

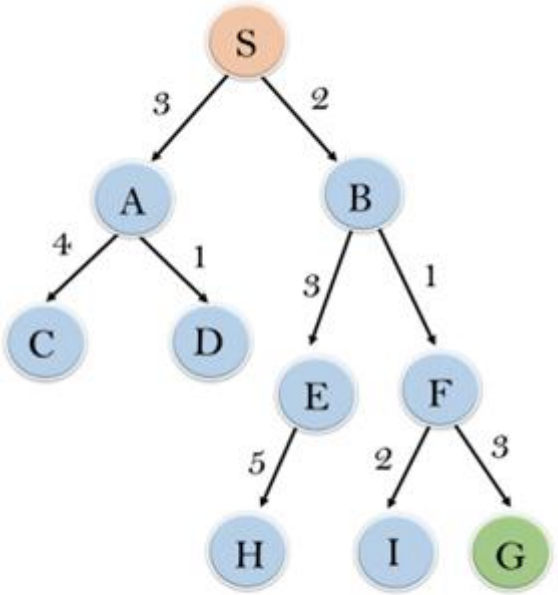
   - Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
   - This algorithm is more efficient than BFS and DFS algorithms.

   Disadvantages:

   - It can behave as an unguided depth-first search in the worst case scenario.
   - It can get stuck in a loop as DFS.
   - This algorithm is not optimal.

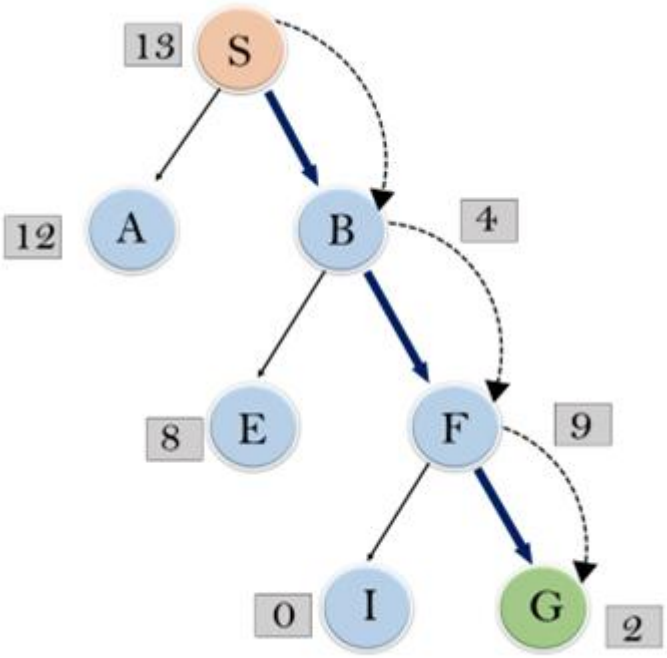
Example:

Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.



Expand the nodes of S and put in the CLOSED list

Initialization: Open [A, B], Closed [S]

Iteration 1: Open [A], Closed [S, B]

Iteration 2: Open [E, F, A], Closed [S, B]
: Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F]
: Open [I, E, A], Closed [S, B, F, G]

Hence the final solution path will be: **S----> B----->F----> G**

Time Complexity: The worst case time complexity of Greedy best first search is $O(b^m)$.

Space Complexity: The worst case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.

Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

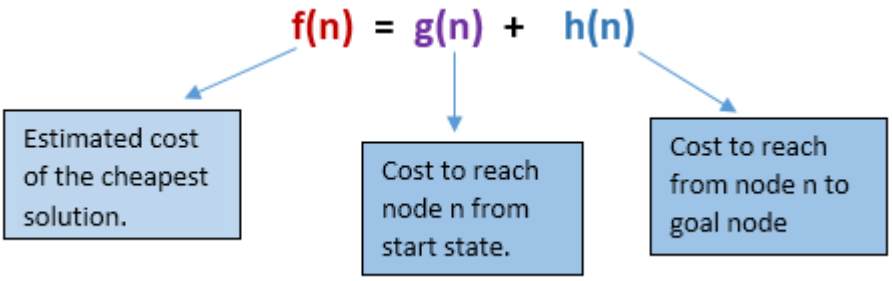
Optimal: Greedy best first search algorithm is not optimal.

2.) A* Search Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search

algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



> At each point in the search space, only those node is expanded which have the lowest value of f(n), and the algorithm terminates when the goal node is found.

Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

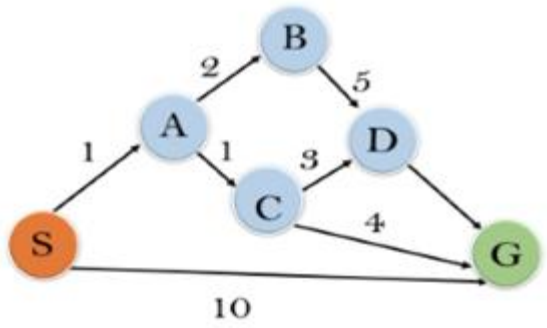
- It does not always produce the shortest path as it mostly based on heuristics and approximation.

- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Example:

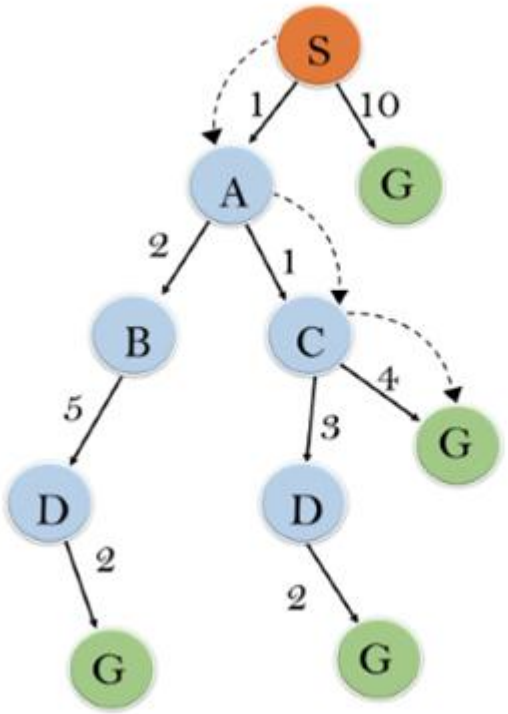
In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.

Here we will use OPEN and CLOSED list.



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution:



Initialization: {(S, 5)}

Iteration1: {(S--> A, 4), (S-->G, 10)}

Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n)<="" li="">

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **O(b^d)**

Hill Climbing Algorithm in Artificial Intelligence

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.
- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

Features of Hill Climbing:

Following are some main features of Hill Climbing Algorithm:

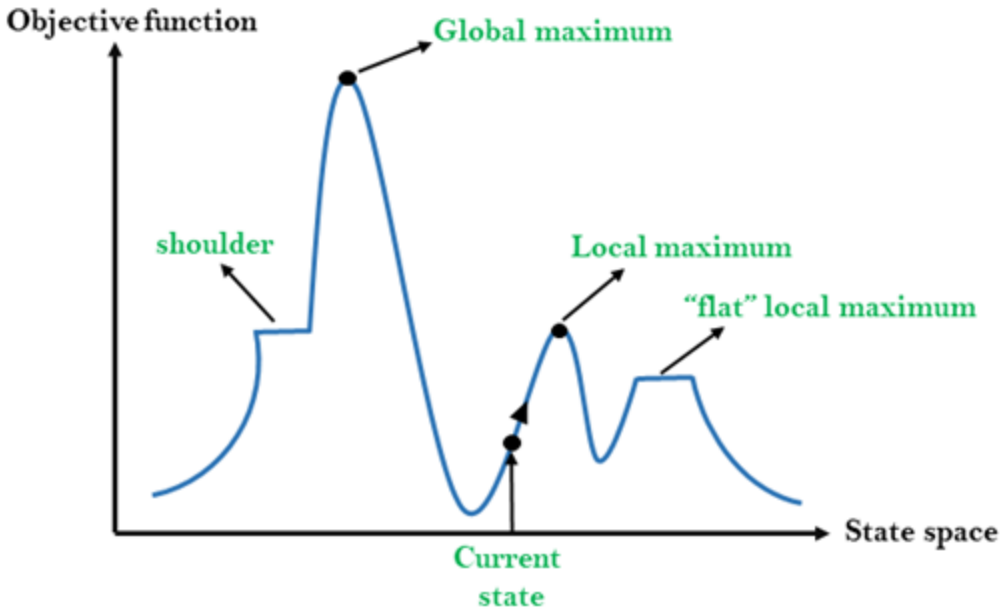
- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.

- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.
- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

State-space Diagram for Hill Climbing:

The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.

On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis. If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum. If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.



Different regions in the state space landscape:

Local Maximum: Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.

Global Maximum: Global maximum is the best possible state of state space landscape. It has the highest value of objective function.

Current state: It is a state in a landscape diagram where an agent is currently present.

Flat local maximum: It is a flat space in the landscape where all the neighbor states of current states have the same value.

Shoulder: It is a plateau region which has an uphill edge.

Types of Hill Climbing Algorithm:

- Simple hill Climbing:
- Steepest-Ascent hill-climbing:
- Stochastic hill Climbing:

1. Simple Hill Climbing:

Simple hill climbing is the simplest way to implement a hill climbing algorithm. **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state**. It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state. This algorithm has the following features:

- Less time consuming
- Less optimal solution and the solution is not guaranteed

Algorithm for Simple Hill Climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
    a. If it is goal state, then return success and quit.
    b. Else if it is better than the current state then assign new state as a current state.
    c. Else if not better than the current state, then return to step2.

  Step 5: Exit.

2. Steepest-Ascent hill climbing:

The steepest-Ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.

a. Let SUCC be a state such that any successor of the current state will be better than it.

b. For each operator that applies to the current state:

    a. Apply the new operator and generate a new state.

    b. Evaluate the new state.

    c. If it is goal state, then return it and quit, else compare it to the SUCC.

    d. If it is better than SUCC, then set new state as SUCC.

    e. If the SUCC is better than the current state, then set current state to SUCC.

Step 5: Exit.

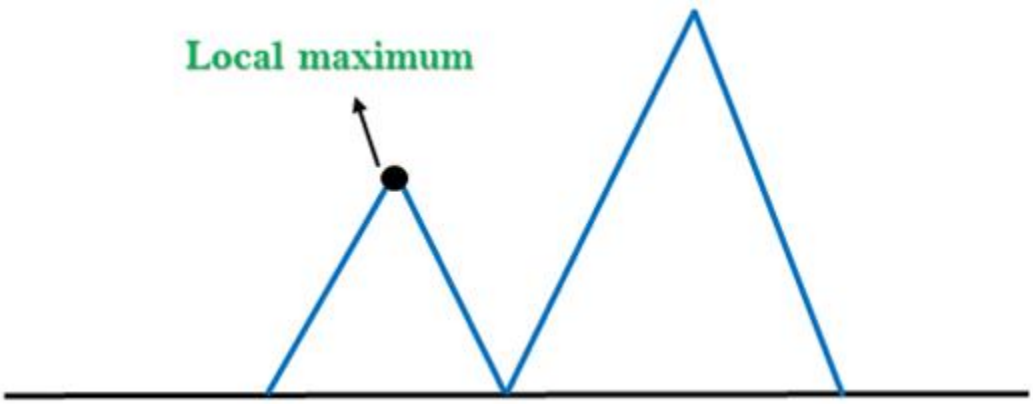
3. Stochastic hill climbing:

Stochastic hill climbing does not examine for all its neighbor before moving. Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.

Problems in Hill Climbing Algorithm:

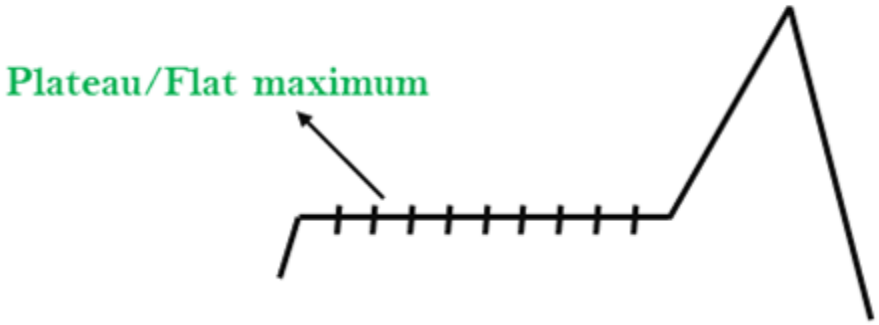
1. Local Maximum: A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

Solution: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.



2. Plateau: A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

Solution: The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.



3. Ridges: A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.

Solution: With the use of bidirectional search, or by moving in different directions, we can improve this problem.

Simulated Annealing:

A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum. And if algorithm applies a random walk, by moving a successor, then it may complete but not efficient. **Simulated Annealing** is an algorithm which yields both efficiency and completeness.

In mechanical term **Annealing** is a process of hardening a metal or glass to a high temperature then cooling gradually, so this allows the metal to reach a low-energy crystalline state. The same process is used in simulated annealing in which the algorithm picks a random move, instead of picking the best move. If the random move improves the state, then it follows the same path. Otherwise, the algorithm follows the path which has a probability of less than 1 or it moves downhill and chooses another path.

Means-Ends Analysis in Artificial Intelligence

- We have studied the strategies which can reason either in forward or backward, but a mixture of the two directions is appropriate for solving a complex and large problem. Such a mixed strategy, make it possible that first to solve the major part of a problem and then go back and solve the small problems arise during combining the big parts of the problem. Such a technique is called **Means-Ends Analysis**.
- Means-Ends Analysis is problem-solving techniques used in Artificial intelligence for limiting search in AI programs.
- It is a mixture of Backward and forward search technique.
- The MEA technique was first introduced in 1961 by Allen Newell, and Herbert A. Simon in their problem-solving computer program, which was named as General Problem Solver (GPS).
- The MEA analysis process centered on the evaluation of the difference between the current state and goal state.

How means-ends analysis Works:

The means-ends analysis process can be applied recursively for a problem. It is a strategy to control search in problem-solving. Following are the main Steps which describes the working of MEA technique for solving a problem.

a. First, evaluate the difference between Initial State and final State.

b. Select the various operators which can be applied for each difference.

c. Apply the operator at each difference, which reduces the difference between the current state and goal state.

Operator Subgoaling

In the MEA process, we detect the differences between the current state and goal state. Once these differences occur, then we can apply an operator to reduce the differences. But sometimes it is possible that an operator cannot be applied to the current state. So we create the subproblem of the current state, in which operator can be applied, such type of backward chaining in which operators are selected, and then sub goals are set up to establish the preconditions of the operator is called **Operator Subgoaling**.

Algorithm for Means-Ends Analysis:

Let's we take Current state as CURRENT and Goal State as GOAL, then following are the steps for the MEA algorithm.

- **Step 1:** Compare CURRENT to GOAL, if there are no differences between both then return Success and Exit.
- **Step 2:** Else, select the most significant difference and reduce it by doing the following steps until the success or failure occurs.

a. Select a new operator O which is applicable for the current difference, and if there is no such operator, then signal failure.

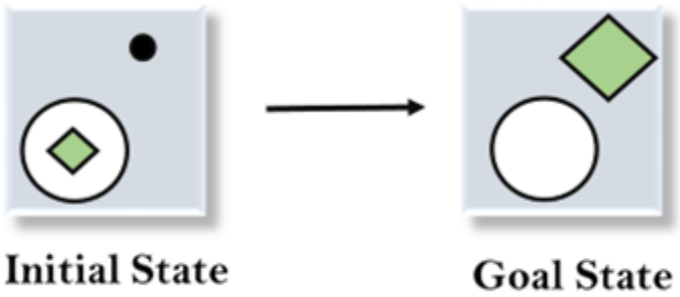
b. Attempt to apply operator O to CURRENT. Make a description of two states.
i) O-Start, a state in which O?s preconditions are satisfied.
ii) O-Result, the state that would result if O were applied In O-start.

c. If
(First-Part <------ MEA (CURRENT, O-START)
And
(LAST-Part <----- MEA (O-Result, GOAL), are successful, then signal Success and return the result of combining FIRST-PART, O, and LAST-PART.

The above-discussed algorithm is more suitable for a simple problem and not adequate for solving complex problems.

Example of Mean-Ends Analysis:

Let's take an example where we know the initial state and goal state as given below. In this problem, we need to get the goal state by finding differences between the initial state and goal state and applying operators.



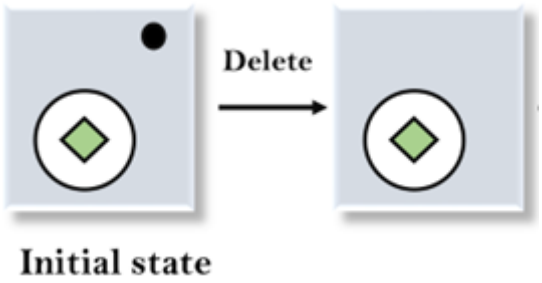
Solution:

To solve the above problem, we will first find the differences between initial states and goal states, and for each difference, we will generate a new state and will apply the operators. The operators we have for this problem are:

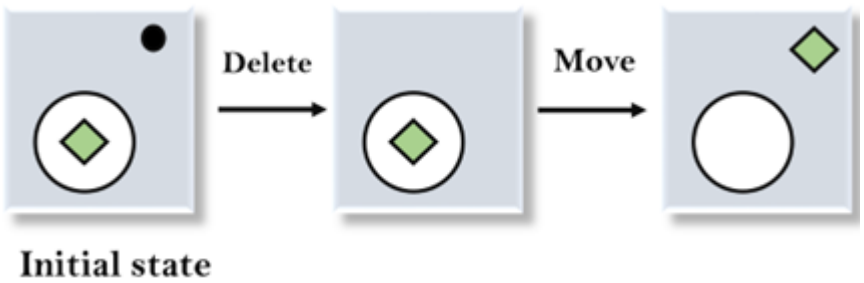
- **Move**
- **Delete**
- **Expand**

1. Evaluating the initial state: In the first step, we will evaluate the initial state and will compare the initial and Goal state to find the differences between both states.

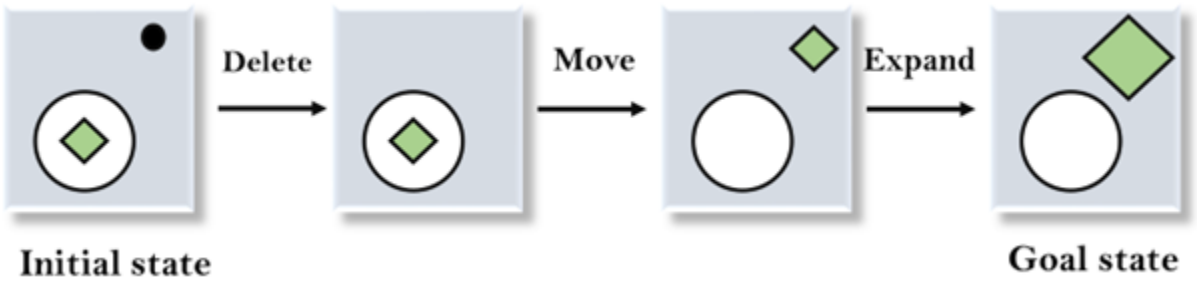
2. Applying Delete operator: As we can check the first difference is that in goal state there is no dot symbol which is present in the initial state, so, first we will apply the **Delete operator** to remove this dot.



3. Applying Move Operator: After applying the Delete operator, the new state occurs which we will again compare with goal state. After comparing these states, there is another difference that is the square is outside the circle, so, we will apply the **Move Operator**.



4. Applying Expand Operator: Now a new state is generated in the third step, and we will compare this state with the goal state. After comparing the states there is still one difference which is the size of the square, so, we will apply **Expand operator**, and finally, it will generate the goal state.



Water jug problem in Artificial Intelligence

In the **water jug problem in Artificial Intelligence**, we are provided with two jugs: one having the capacity to hold 3 gallons of water and the other has the capacity to hold 4 gallons of water. There is no other measuring equipment available and the jugs also do not have any kind of marking on them. So, the agent's task here is to fill the 4-gallon jug with 2 gallons of water by using only these two jugs and no other material. Initially, both our jugs are empty.

So, to solve this problem, following set of rules were proposed:

Production rules for solving the water jug problem

Here, let ***x*** denote the 4-gallon jug and ***y*** denote the 3-gallon jug.

S.No.	Initial State	Condition	Final state	Description of action taken
1.	(x,y)	If x<4	(4,y)	Fill the 4 gallon jug completely
2.	(x,y)	if y<3	(x,3)	Fill the 3 gallon jug completely
3.	(x,y)	If x>0	(x-d,y)	Pour some part from the 4 gallon jug
4.	(x,y)	If y>0	(x,y-d)	Pour some part from the 3 gallon jug
5.	(x,y)	If x>0	(0,y)	Empty the 4 gallon jug

6.	(x,y)	If y>0	(x,0)	Empty the 3 gallon jug
7.	(x,y)	If (x+y)<7	(4, y-[4-x])	Pour some water from the 3 gallon jug to fill the four gallon jug
8.	(x,y)	If (x+y)<7	(x-[3-y],y)	Pour some water from the 4 gallon jug to fill the 3 gallon jug.
9.	(x,y)	If (x+y)<4	(x+y,0)	Pour all water from 3 gallon jug to the 4 gallon jug
10.	(x,y)	if (x+y)<3	(0, x+y)	Pour all water from the 4 gallon jug to the 3 gallon jug

The listed production rules contain all the actions that could be performed by the agent in transferring the contents of jugs. But, to solve the water jug problem in a minimum number of moves, following set of rules in the given sequence should be performed:

Solution of water jug problem according to the production rules:

S.No.	**4 gallon jug contents**	**3 gallon jug contents**	**Rule followed**
1.	0 gallon	0 gallon	Initial state
2.	0 gallon	3 gallons	Rule no.2
3.	3 gallons	0 gallon	Rule no. 9
4.	3 gallons	3 gallons	Rule no. 2
5.	4 gallons	2 gallons	Rule no. 7
6.	0 gallon	2 gallons	Rule no. 5
7.	2 gallons	0 gallon	Rule no. 9

On reaching the 7^{th} attempt, we reach a state which is our goal state. Therefore, at this state, our problem is solved.

Eight Puzzle problem

We also know the **eight puzzle problem** by the name of **N puzzle problem** or **sliding puzzle problem.**

N-puzzle that consists of N tiles (N+1 titles with an empty tile) where N can be 8, 15, 24 and so on.

In our example **N = 8.** (that is **square root of (8+1) = 3 rows and 3 columns**).

In the same way, if we have N = 15, 24 in this way, then they have Row and columns as follow **(square root of (N+1) rows and square root of (N+1) columns).**

That is if **N=15** than number of rows and columns= 4, and if **N= 24** number of rows and columns= 5.

So, basically in these types of problems we have given a **initial state or initial configuration (Start state) and a Goal state or Goal Configuration.**

Here We are solving a problem **of 8 puzzle** that is **a 3x3 matrix.**

Initial state

1	**2**	**3**
	4	**6**
7	**5**	**8**

Goal state

1	**2**	**3**
4	**5**	**6**
7	**8**	

Solution:
The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state.

Rules of solving puzzle

Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile.

The empty space can only **move in four directions** (Movement of empty space)

1. Up
2. Down
3. Right or
4. Left

The empty space **cannot move diagonally** and can take **only one step at a time**.

All possible move of a Empty tile

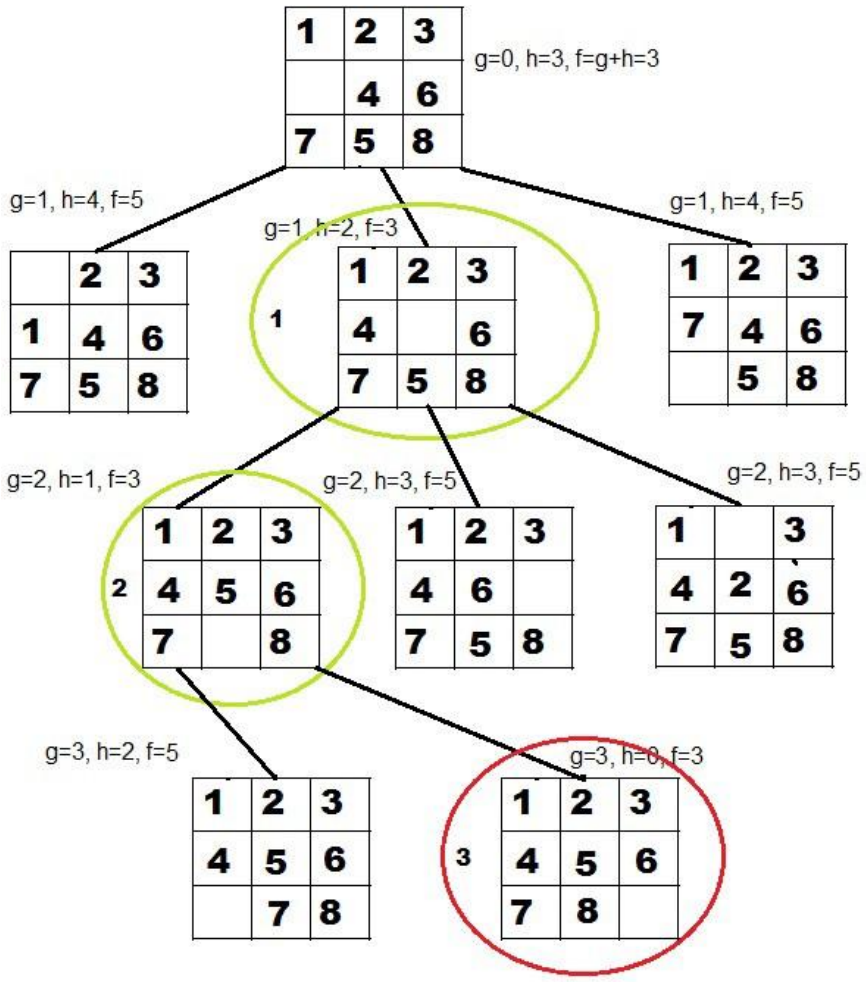
o- Position total possible moves are **(2)**, **x - position** total possible moves are **(3)** and

#-position total possible moves **are (4)**

Let's solve the problem with **Heuristic Search** that is **Informed Search (A* , Best First Search (Greedy Search))**

To solve the problem with Heuristic search or informed search we have to calculate Heuristic values of each node to calculate **cost function. (f=g+h)**

Initial state **Goal state**



Note: See the initial state and goal state carefully all values except (4,5 and 8) are at their respective places. **so, the heuristic value for first node is 3.**(Three values are misplaced to reach the goal). And l**et's take actual cost (g) according to depth.**

Note: Because of quick solution time complexity is less than that of Uninformed search but optimal solution not possible.

Unit- 5
Game Playing in Artificial Intelligence

Game Playing is an important domain of artificial intelligence. Games don't require much knowledge; the only knowledge we need to provide is the rules, legal moves and the conditions of winning or losing the game.

Both players try to win the game. So, both of them try to make the best move possible at each turn. Searching techniques like BFS(Breadth First Search) are not accurate for this as the branching factor is very high, so searching will take a lot of time. So, we need another search procedures that improve –

- **Generate procedure** so that only good moves are generated.
- **Test procedure** so that the best move can be explored first.

The most common search technique in game playing is Minimax search procedure. It is depth-first depth-limited search procedure. It is used for games like chess and tic-tac-toe.

Minimax algorithm uses two functions –

MOVEGEN : It generates all the possible moves that can be generated from the current position.

STATICEVALUATION : It returns a value depending upon the goodness from the viewpoint otwo-player
This algorithm is a two player game, so we call the first player as PLAYER1 and second player as PLAYER2. The value of each node is backed-up from its children. For PLAYER1 the backed-up value is the maximum value of its children and for PLAYER2 the backed-up value is the minimum value of its children. It provides most promising move to PLAYER1, assuming that the PLAYER2 has make the best move. It is a recursive algorithm, as same procedure occurs at each level.

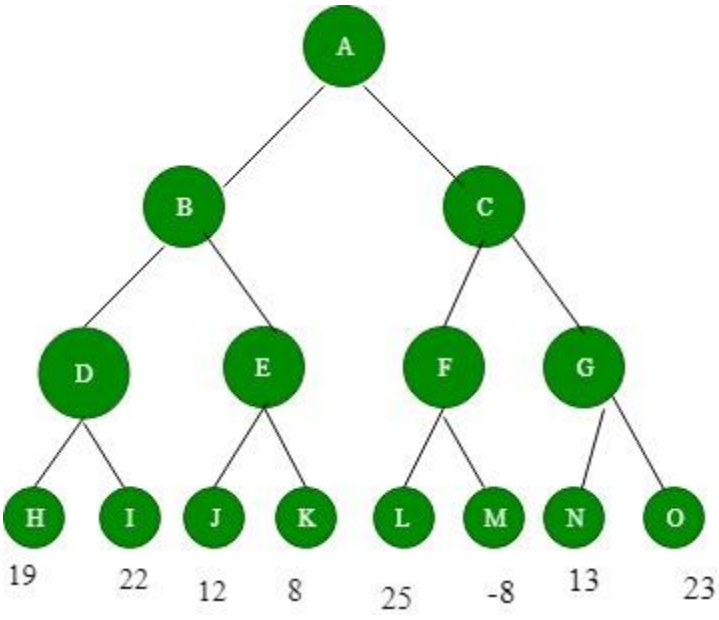


Figure 1: Before backing-up of values

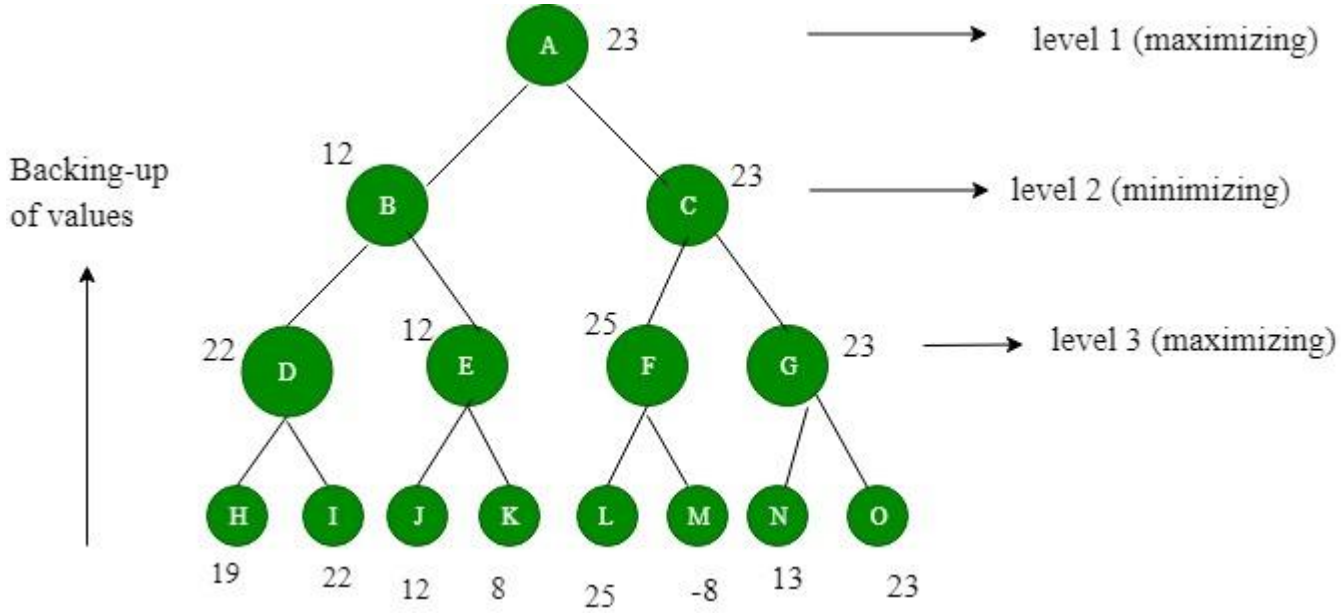


Figure 2: After backing-up of values

We assume that PLAYER1 will start the game. 4 levels are generated. The value to nodes H, I, J, K, L, M, N, O is provided by STATICEVALUATION function. Level 3 is maximizing level, so all nodes of level 3 will take maximum values of their children. Level 2 is minimizing level, so all its nodes will take minimum values of their children. This process continues. The value of A is 23. That means A should choose C move to win.

Mini-Max Algorithm in Artificial Intelligence

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Pseudo-code for MinMax Algorithm:

```
1. function minimax(node, depth, maximizingPlayer) is
2. if depth ==0 or node is a terminal node then
3. return static evaluation of node
4.
5. if MaximizingPlayer then      // for Maximizer Player
6. maxEva= -infinity
7.  for each child of node do
8.  eva= minimax(child, depth-1, false)
9. maxEva= max(maxEva,eva)       //gives Maximum of the values
10. return maxEva
11.
12. else                    // for Minimizer player
13.  minEva= +infinity
14.  for each child of node do
15.  eva= minimax(child, depth-1, true)
16.  minEva= min(minEva, eva)       //gives minimum of the values
17.  return minEva
```

Initial call:

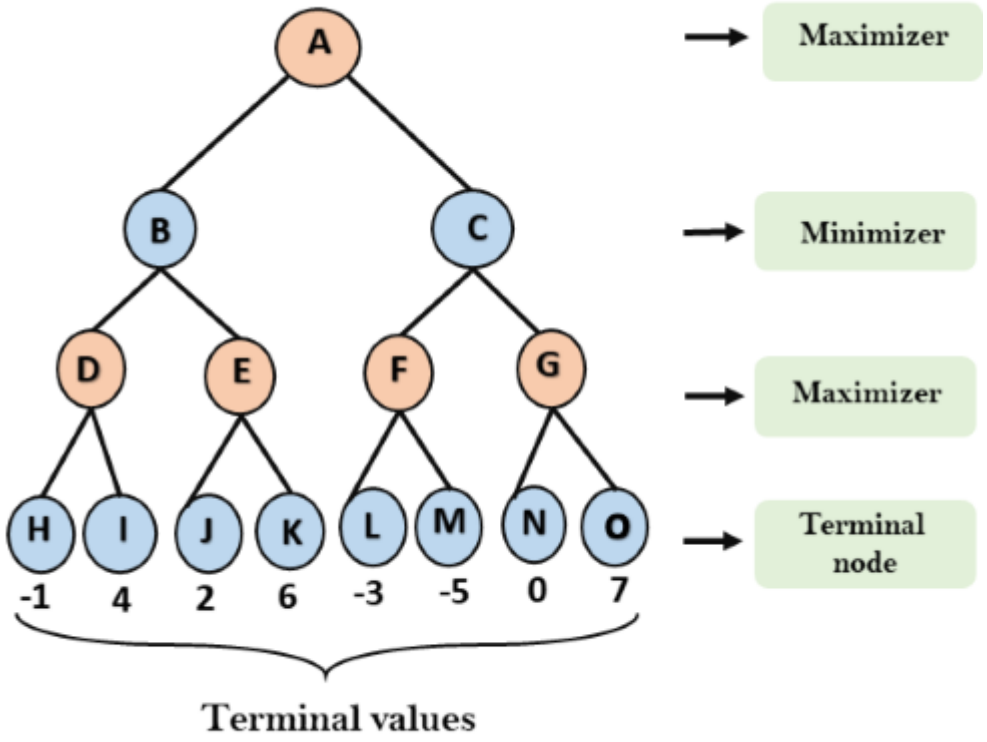
Minimax(node, 3, true)

Working of Min-Max Algorithm:

- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

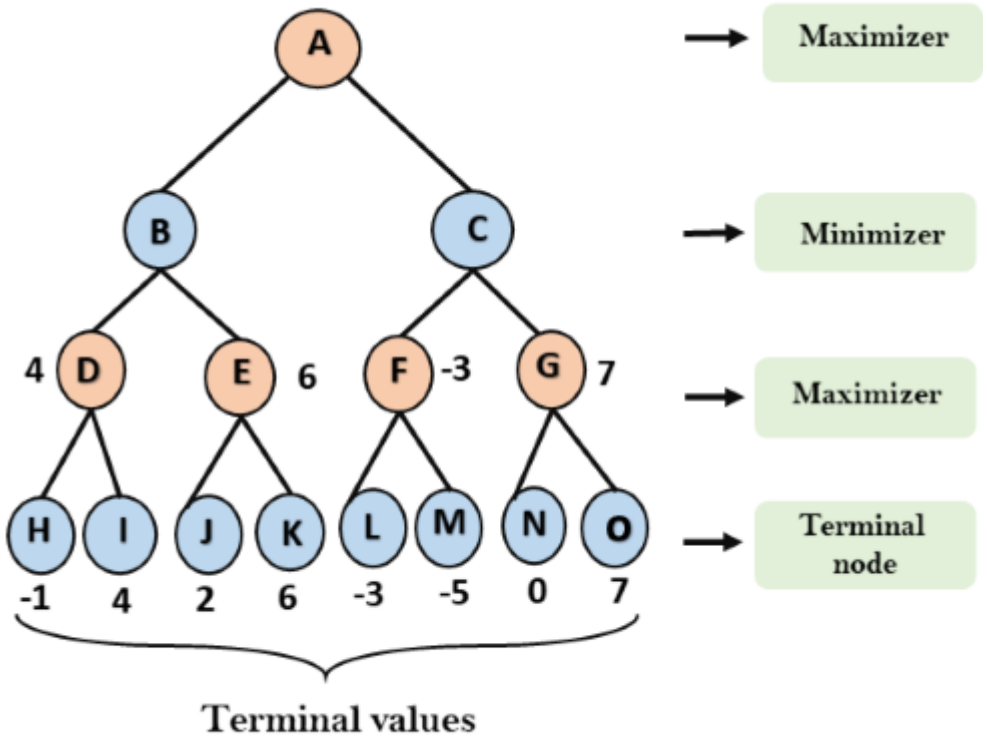
Step-1: In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial

value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



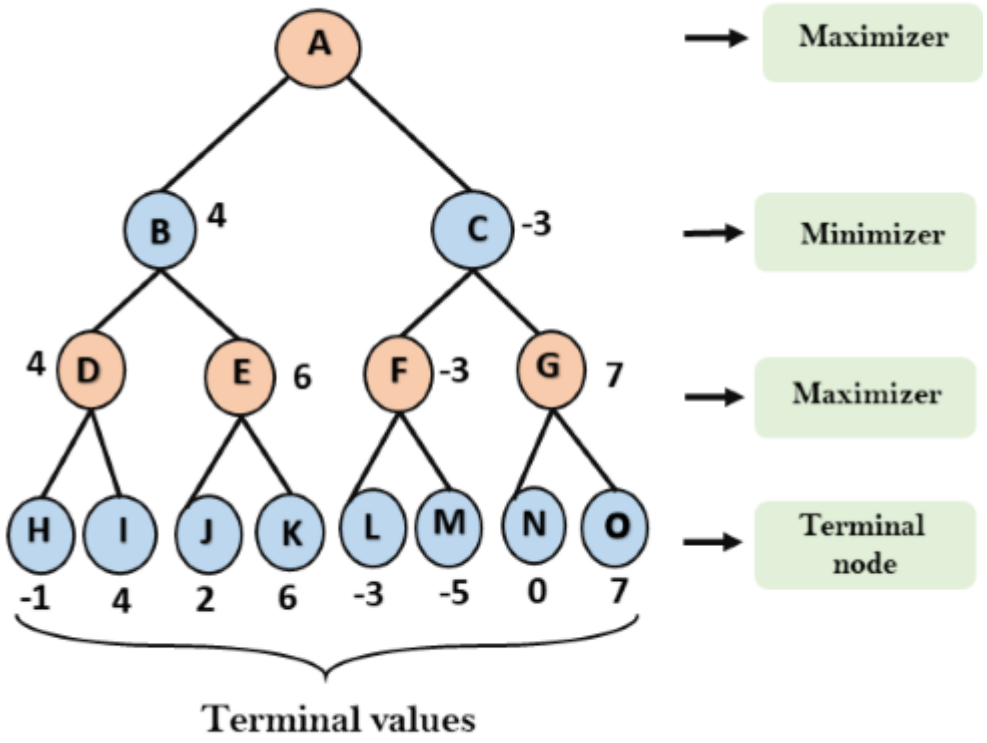
Step 2: Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

- For node D max(-1,- -∞) => max(-1,4)= 4
- For Node E max(2, -∞) => max(2, 6)= 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) = max(0, 7) = 7



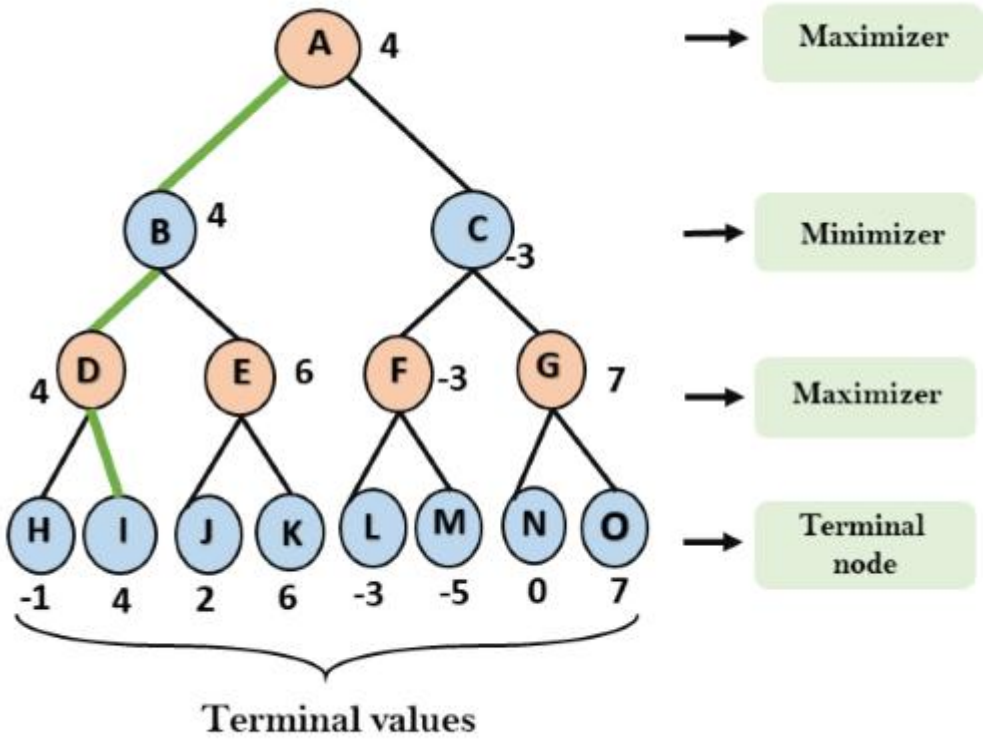
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

- For node B= min(4,6) = 4
- For node C= min (-3, 7) = -3



Step 4: Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

- For node A max(4, -3)= 4



That was the complete workflow of the minimax two player game.

Properties of Mini-Max algorithm:

- **Complete-** Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- **Optimal-** Min-Max algorithm is optimal if both opponents are playing optimally.
- **Time complexity-** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is **$O(b^m)$**, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- **Space Complexity-** Space complexity of Mini-max algorithm is also similar to DFS which is **O(bm)**.

Limitation of the minimax Algorithm:

The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc. This type of games has a huge branching factor, and the player has lots of choices to decide. This limitation of the minimax algorithm can be improved from **alpha-beta pruning**.

Alpha-Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:
    a. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞.
    b. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞.

The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is:

1. $\alpha >= \beta$

Key points about alpha-beta pruning:

- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

Pseudo-code for Alpha-beta Pruning:

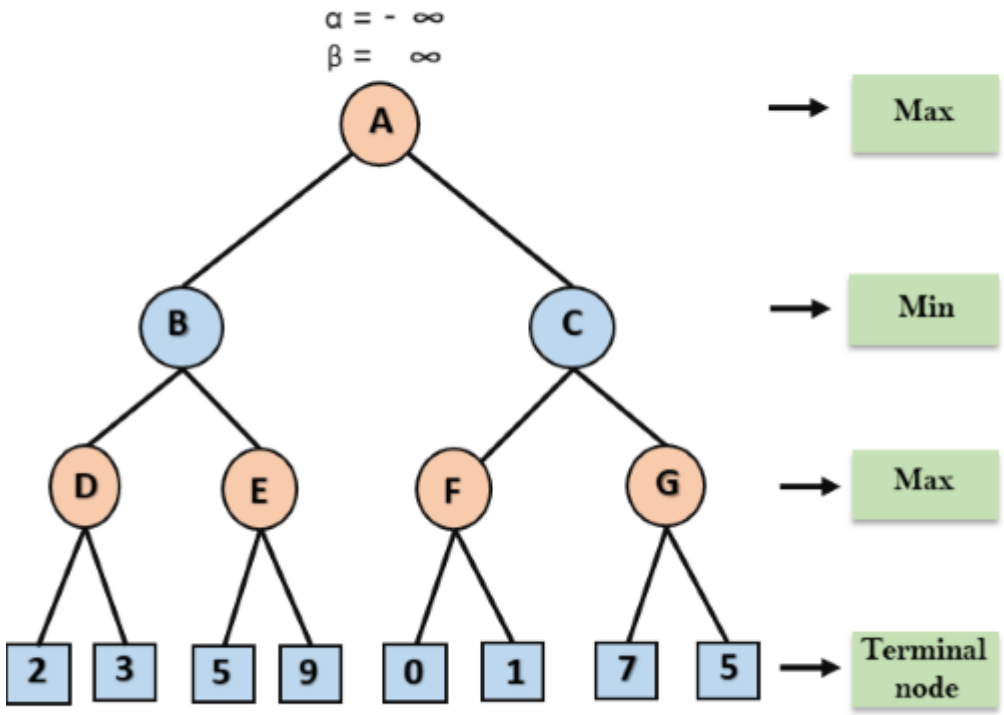
```
1. function minimax(node, depth, alpha, beta, maximizingPlayer) is
2. if depth ==0 or node is a terminal node then
3. return static evaluation of node
```

```
4.
5.  if MaximizingPlayer then      // for Maximizer Player
6.     maxEva= -infinity
7.     for each child of node do
8.     eva= minimax(child, depth-1, alpha, beta, False)
9.    maxEva= max(maxEva, eva)
10.   alpha= max(alpha, maxEva)
11.    if beta<=alpha
12.  break
13.  return maxEva
14.
15. else                    // for Minimizer player
16.    minEva= +infinity
17.    for each child of node do
18.    eva= minimax(child, depth-1, alpha, beta, true)
19.    minEva= min(minEva, eva)
20.    beta= min(beta, eva)
21.     if beta<=alpha
22.   break
23.  return minEva
```

Working of Alpha-Beta Pruning:

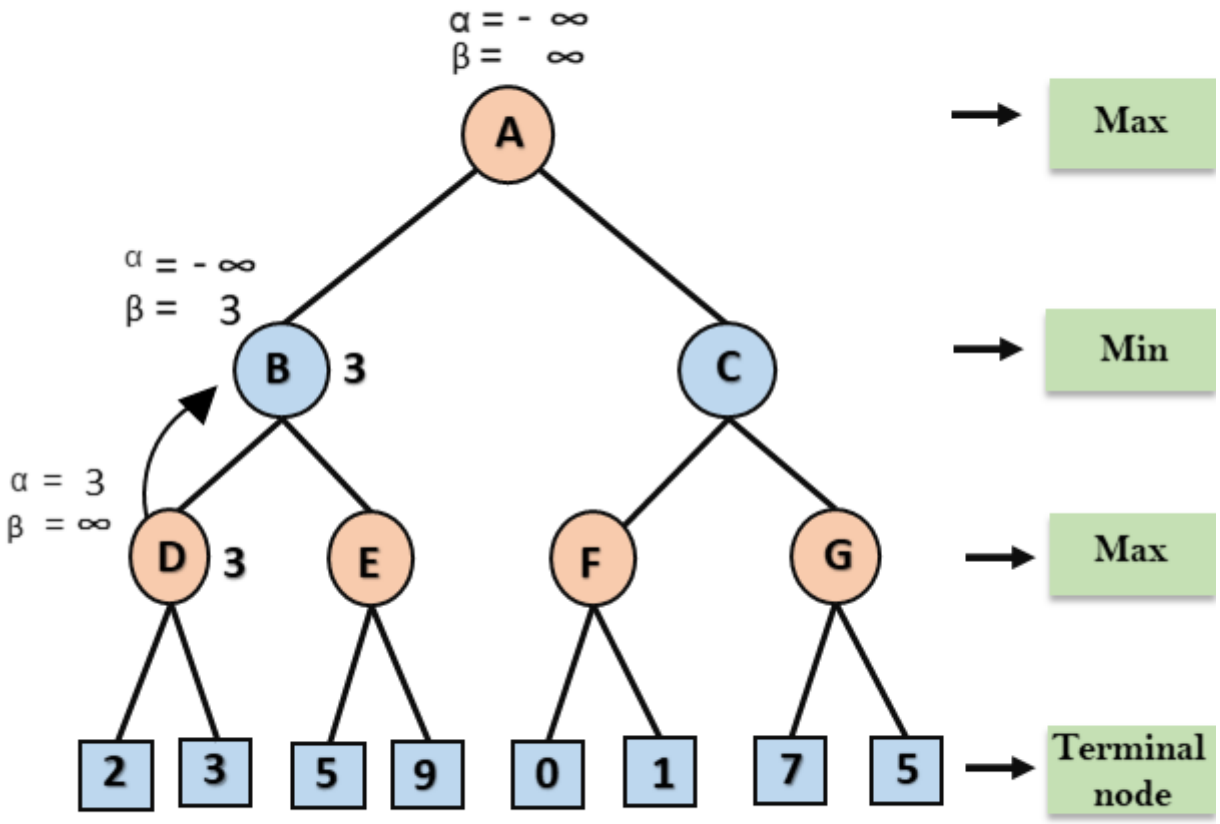
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

Step 1: At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



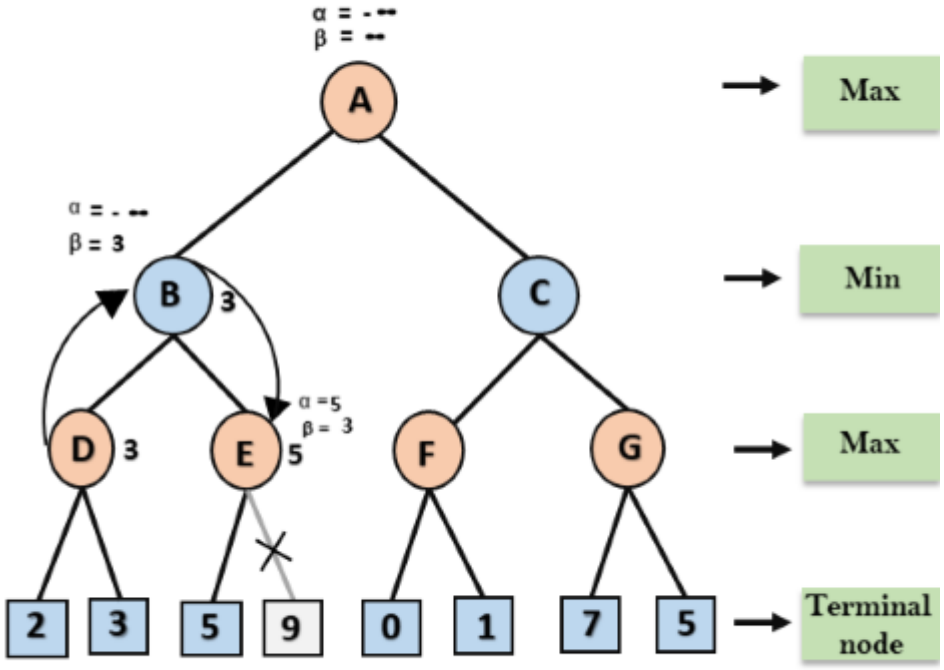
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

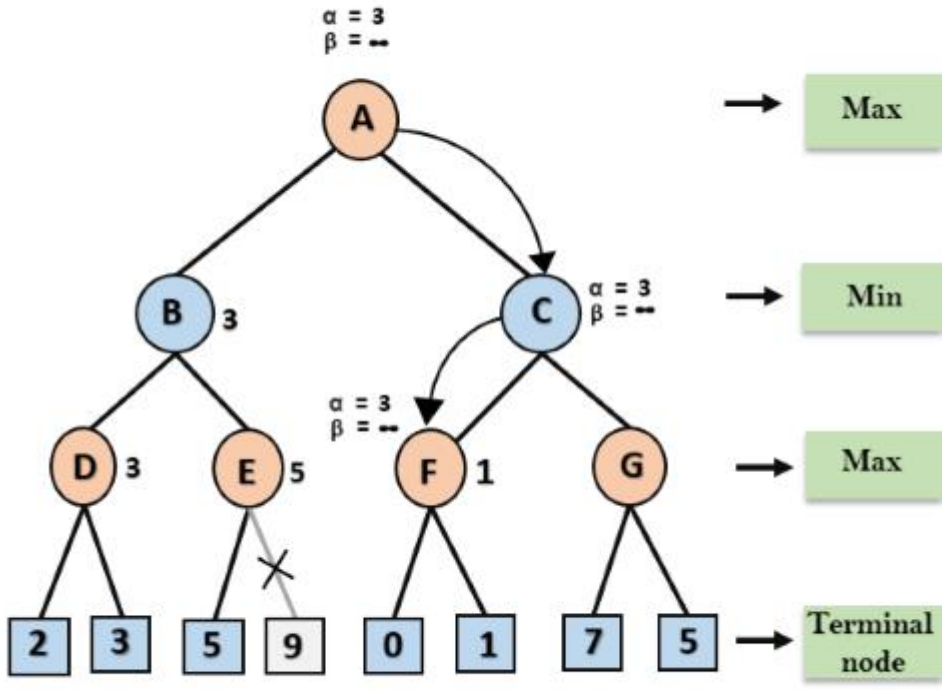
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



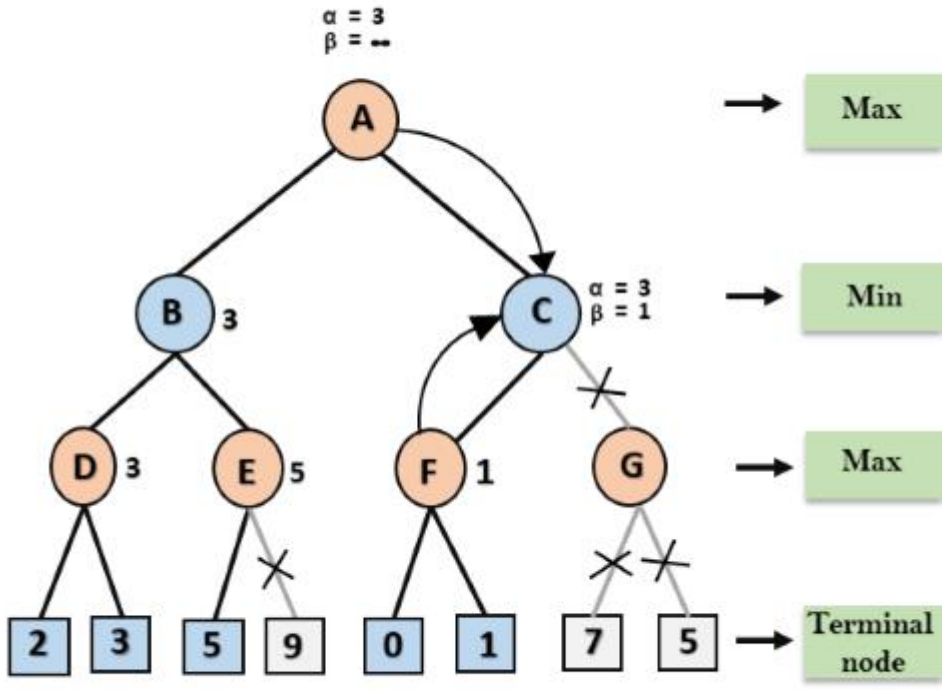
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

At node C, α=3 and β= +∞, and the same values will be passed on to node F.

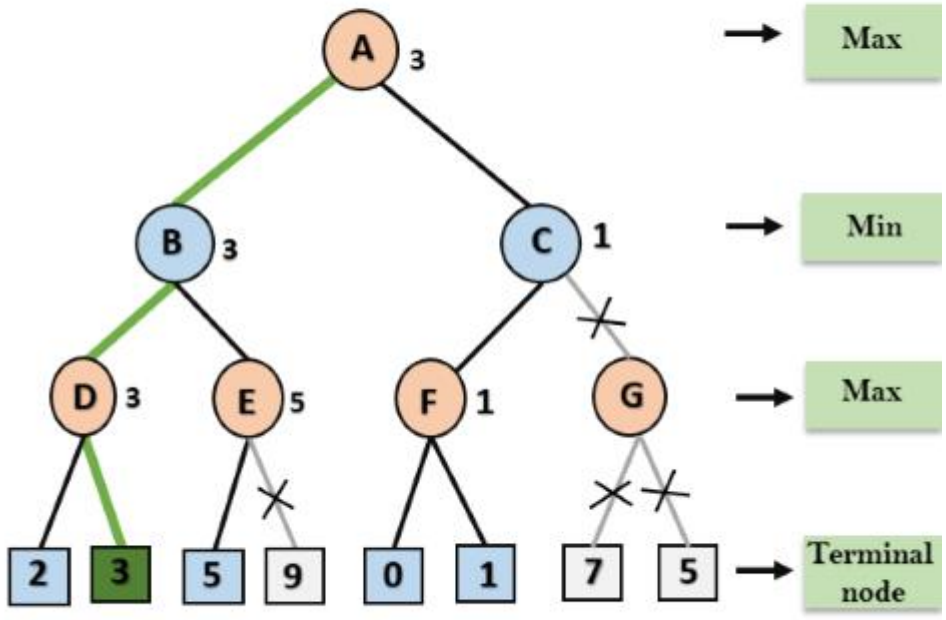
Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



Step 7: Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



Move Ordering in Alpha-Beta pruning:

The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.

It can be of two types:

- **Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.
- **Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.

Rules to find good ordering:

Following are some rules to find good ordering in alpha-beta pruning:

- Occur the best move from the shallowest node.

- Order the nodes in the tree such that the best nodes are checked first.
- Use domain knowledge while finding the best move. Ex: for Chess, try order: captures first, then threats, then forward moves, backward moves.
- We can bookkeep the states, as there is a possibility that states may repeat.

Unit- 6

AI - Natural Language Processing

Natural Language Processing (NLP) refers to AI method of communicating with an intelligent systems using a natural language such as English.

Processing of Natural Language is required when you want an intelligent system like robot to perform as per your instructions, when you want to hear decision from a dialogue based clinical expert system, etc.

The field of NLP involves making computers to perform useful tasks with the natural languages humans use. The input and output of an NLP system can be −

- Speech
- Written Text

Components of NLP

There are two components of NLP as given −

Natural Language Understanding (NLU)

Understanding involves the following tasks −

- Mapping the given input in natural language into useful representations.
- Analyzing different aspects of the language.

Natural Language Generation (NLG)

It is the process of producing meaningful phrases and sentences in the form of natural language from some internal representation.

It involves −

- **Text planning** − It includes retrieving the relevant content from knowledge base.
- **Sentence planning** − It includes choosing required words, forming meaningful phrases, setting tone of the sentence.
- **Text Realization** − It is mapping sentence plan into sentence structure.

The NLU is harder than NLG.

Difficulties in NLU

NL has an extremely rich form and structure.

It is very ambiguous. There can be different levels of ambiguity −

- **Lexical ambiguity** − It is at very primitive level such as word-level.
- For example, treating the word “board” as noun or verb?
- **Syntax Level ambiguity** − A sentence can be parsed in different ways.
- For example, “He lifted the beetle with red cap.” − Did he use cap to lift the beetle or he lifted a beetle that had red cap?
- **Referential ambiguity** − Referring to something using pronouns. For example, Rima went to Gauri. She said, “I am tired.” − Exactly who is tired?
- One input can mean different meanings.
- Many inputs can mean the same thing.

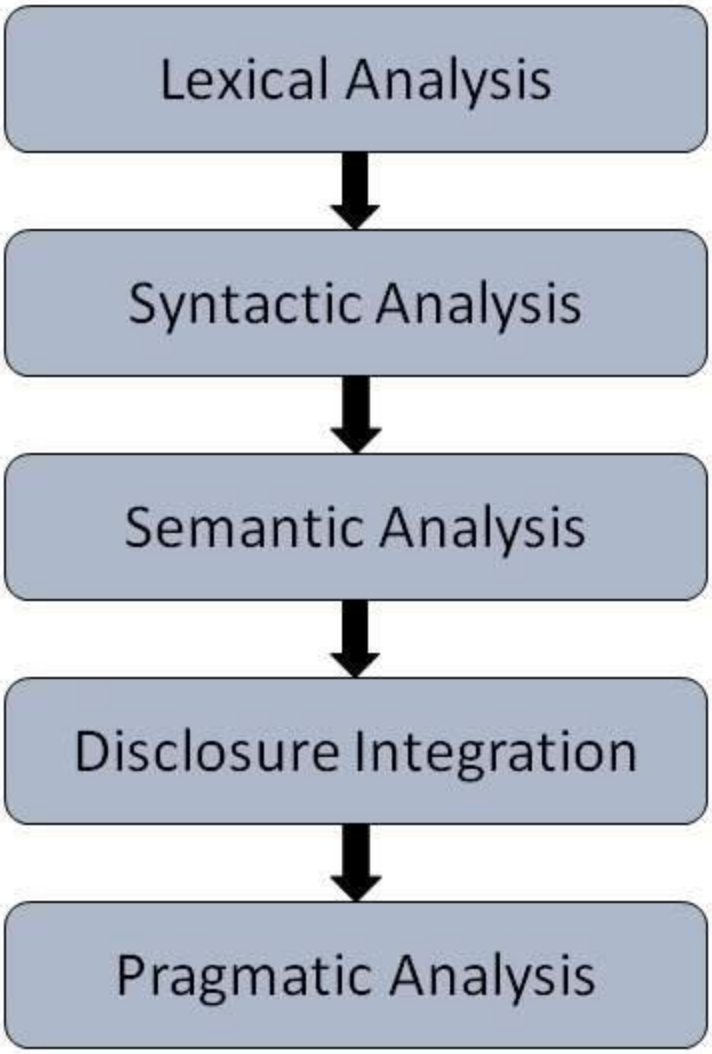
NLP Terminology

- **Phonology** − It is study of organizing sound systematically.
- **Morphology** − It is a study of construction of words from primitive meaningful units.
- **Morpheme** − It is primitive unit of meaning in a language.
- **Syntax** − It refers to arranging words to make a sentence. It also involves determining the structural role of words in the sentence and in phrases.
- **Semantics** − It is concerned with the meaning of words and how to combine words into meaningful phrases and sentences.
- **Pragmatics** − It deals with using and understanding sentences in different situations and how the interpretation of the sentence is affected.
- **Discourse** − It deals with how the immediately preceding sentence can affect the interpretation of the next sentence.
- **World Knowledge** − It includes the general knowledge about the world.

Steps in NLP

There are general five steps −

- **Lexical Analysis** − It involves identifying and analyzing the structure of words. Lexicon of a language means the collection of words and phrases in a language. Lexical analysis is dividing the whole chunk of txt into paragraphs, sentences, and words.
- **Syntactic Analysis (Parsing)** − It involves analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words. The sentence such as “The school goes to boy” is rejected by English syntactic analyzer.



- **Semantic Analysis** − It draws the exact meaning or the dictionary meaning from the text. The text is checked for meaningfulness. It is done by mapping syntactic structures and objects in the task domain. The semantic analyzer disregards sentence such as “hot ice-cream”.
- **Discourse Integration** − The meaning of any sentence depends upon the meaning of the sentence just before it. In addition, it also brings about the meaning of immediately succeeding sentence.
- **Pragmatic Analysis** − During this, what was said is re-interpreted on what it actually meant. It involves deriving those aspects of language which require real world knowledge.

Applications of NLP

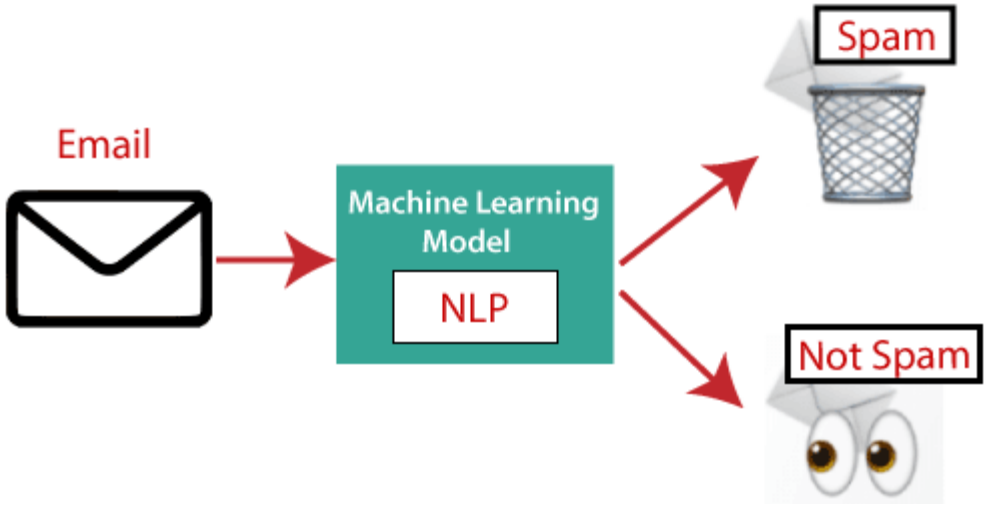
There are the following applications of NLP -

1. Question Answering

Question Answering focuses on building systems that automatically answer the questions asked by humans in a natural language.

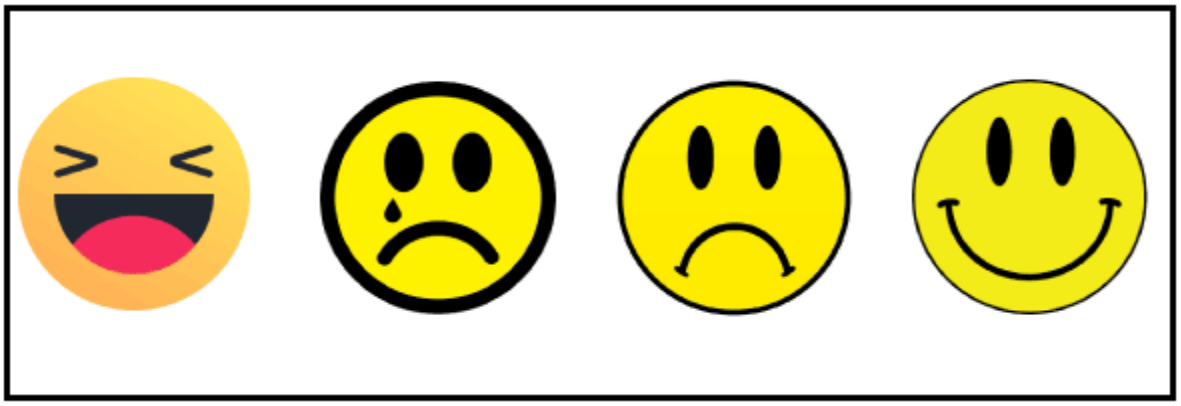
2. Spam Detection

Spam detection is used to detect unwanted e-mails getting to a user's inbox.



3. Sentiment Analysis

Sentiment Analysis is also known as **opinion mining**. It is used on the web to analyse the attitude, behaviour, and emotional state of the sender. This application is implemented through a combination of NLP (Natural Language Processing) and statistics by assigning the values to the text (positive, negative, or natural), identify the mood of the context (happy, sad, angry, etc.)



4. Machine Translation

Machine translation is used to translate text or speech from one natural language to another natural language.

Example: Google Translator

5. Spelling correction

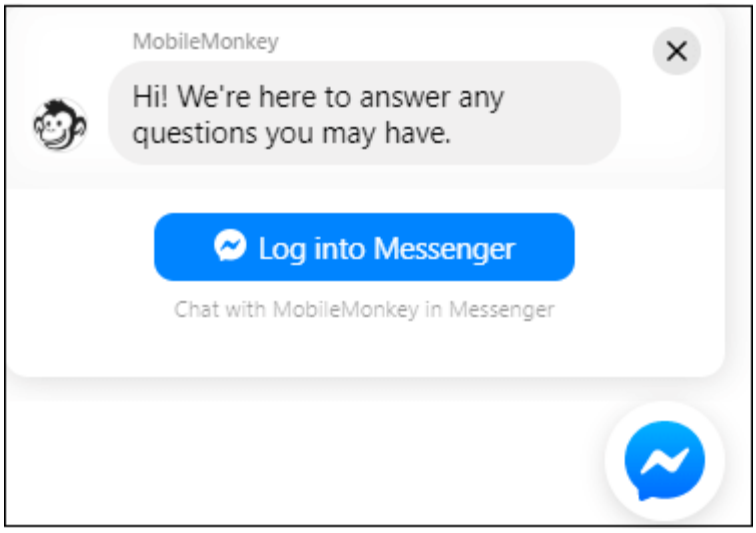
Microsoft Corporation provides word processor software like MS-word, PowerPoint for the spelling correction.

6. Speech Recognition

Speech recognition is used for converting spoken words into text. It is used in applications, such as mobile, home automation, video recovery, dictating to Microsoft Word, voice biometrics, voice user interface, and so on.

7. Chatbot

Implementing the Chatbot is one of the important applications of NLP. It is used by many companies to provide the customer's chat services.



8. Information extraction

Information extraction is one of the most important applications of NLP. It is used for extracting structured information from unstructured or semi-structured machine-readable documents.

9. Natural Language Understanding (NLU)

It converts a large set of text into more formal representations such as first-order logic structures that are easier for the computer programs to manipulate notations of the natural language processing.

Advantages of NLP

- NLP helps users to ask questions about any subject and get a direct response within seconds.
- NLP offers exact answers to the question means it does not offer unnecessary and unwanted information.
- NLP helps computers to communicate with humans in their languages.

- It is very time efficient.
- Most of the companies use NLP to improve the efficiency of documentation processes, accuracy of documentation, and identify the information from large databases.

Disadvantages of NLP

A list of disadvantages of NLP is given below:

- NLP may not show context.
- NLP is unpredictable
- NLP may require more keystrokes.
- NLP is unable to adapt to the new domain, and it has a limited function that's why NLP is built for a single and specific task only.

Artificial Intelligence: Grammars and Languages

The types of grammars that exist are Noam Chomsky invented a hierarchy of grammars.

The hierarchy consists of four main types of grammars.

Chomsky Hierarchy

Chomsky Hierarchy represents the class of languages that are accepted by the different machine. The category of language in Chomsky's Hierarchy is as given below:

1. Type 0 known as Unrestricted Grammar.
2. Type 1 known as Context Sensitive Grammar.
3. Type 2 known as Context Free Grammar.
4. Type 3 Regular Grammar.

Type 0 Grammar:

Type 0 grammar is known as Unrestricted grammar. There is no restriction on the grammar rules of these types of languages. These languages can be efficiently modeled by Turing machines.

For example:

1. bAa → aa
2. S → s

Type 1 Grammar:

Type 1 grammar is known as Context Sensitive Grammar. The context sensitive grammar is used to represent context sensitive language. The context sensitive grammar follows the following rules:

- The context sensitive grammar may have more than one symbol on the left hand side of their production rules.
- The number of symbols on the left-hand side must not exceed the number of symbols on the right-hand side.
- The rule of the form $A \rightarrow \varepsilon$ is not allowed unless A is a start symbol. It does not occur on the right-hand side of any rule.
- The Type 1 grammar should be Type 0. In type 1, Production is in the form of $V \rightarrow T$

Where the count of symbol in V is less than or equal to T.

For example:

1. $S \rightarrow AT$
2. $T \rightarrow xy$
3. $A \rightarrow a$

Type 2 Grammar:

Type 2 Grammar is known as Context Free Grammar. Context free languages are the languages which can be represented by the context free grammar (CFG). Type 2 should be type 1. The production rule is of the form

1. $A \rightarrow \alpha$

Where A is any single non-terminal and is any combination of terminals and non-terminals.

For example:

1. $A \rightarrow aBb$
2. $A \rightarrow b$
3. $B \rightarrow a$

Type 3 Grammar:

Type 3 Grammar is known as Regular Grammar. Regular languages are those languages which can be described using regular expressions. These languages can be modeled by NFA or DFA.

Type 3 is most restricted form of grammar. The Type 3 grammar should be Type 2 and Type 1. Type 3 should be in the form of

1. V → T*V / T*

   For example:

1. A → xy

   The simplest grammars are used to define regular languages.

   A regular language is one that can be described or understood by a finite state automaton. Such languages are very simplistic and allow sentences such as “aaaaabbbbbb.” Recall that a finite state automaton consists of a finite number of states, and rules that define how the automaton can transition from one state to another.

   A finite state automaton could be designed that defined the language that consisted of a string of one or more occurrences of the letter a. Hence, the following strings would be valid strings in this language:

   aaa

   a

   aaaaaaaaaaaaaaaa

   Regular languages are of interest to computer scientists, but are not of great interest to the field of natural language processing because they are not powerful enough to represent even simple formal languages, let alone the more complex natural languages.

   Sentences defined by a regular grammar are often known as regular expressions. The grammar that we defined above using rewrite rules is a context-free grammar.

   It is context free because it defines the grammar simply in terms of which word types can go together—it does not specify the way that words should agree with each.

   A stale dog climbs Mount Rushmore.

   It also, allows the following sentence, which is not grammatically correct:

   Chickens eats.

A context-free grammar can have only at most one terminal symbol on the right-hand side of its rewrite rules.

Rewrite rules for a context-sensitive grammar, in contrast, can have more than one terminal symbol on the right-hand side. This enables the grammar to specify number, case, tense, and gender agreement.

Each context-sensitive rewrite rule must have at least as many symbols on the right-hand side as it does on the left-hand side.

Rewrite rules for context-sensitive grammars have the following form:

$$A\ X\ B \rightarrow A\ Y\ B$$

which means that in the context of A and B, X can be rewritten as Y.

Each of A, B, X, and Y can be either a terminal or a nonterminal symbol.

Context-sensitive grammars are most usually used for natural language processing because they are powerful enough to define the kinds of grammars that natural languages use. Unfortunately, they tend to involve a much larger number of rules and are a much less natural way to describe language, making them harder for human developers to design than context free grammars.

The final class of grammars in Chomsky's hierarchy consists of recursively enumerable grammars (also known as unrestricted grammars).

A recursively enumerable grammar can define any language and has no restrictions on the structure of its rewrite rules. Such grammars are of interest to computer scientists but are not of great use in the study of natural language processing.

PARSING PROCESS

Parsing is the term used to describe the process of automatically building syntactic analysis of a sentence in terms of a given grammar and lexicon. The resulting syntactic analysis may be used as input to a process of semantic interpretation. Occasionally, parsing is also used to include both syntactic and semantic analysis. The parsing process is done by the parser. The parsing performs grouping and labeling of parts of a sentence in a way that displays their relationships to each other in a proper way.

The parser is a computer program which accepts the natural language sentence as input and generates an output structure suitable for analysis. The lexicon is a dictionary of words where each word contains some syntactic, some semantic and possibly some pragmatic information.

The entry in the lexicon will contain a root word and its various derivatives. The information in the lexicon is needed to help determine the function and meanings of the words in a sentence. The basic parsing technique is shown in figure .

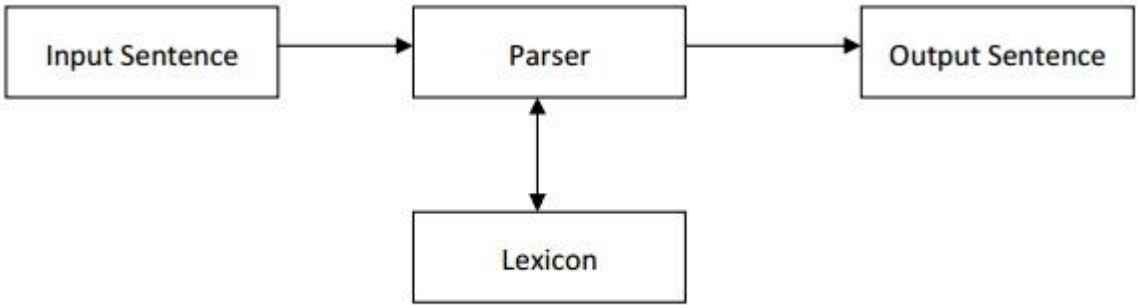


Figure Parsing Technique

Generally in computational linguistics the lexicon supplies paradigmatic information about words including part of speech labels, irregular plurals and sub categorization information for verbs. Traditionally, lexicons were quite small and were constructed largely by hand. The additional information being added to the lexicon increase the complexity of the lexicon. The organization and entries of a lexicon will vary from one implementation to another but they are usually made up of variable length data structures such as lists or records arranged in alphabetical order. The word order may also be given in terms of usage frequency so that frequently used words like “a”, “the” and “an” will appear at the beginning of the list facilitating the search. The entries in a lexicon could be grouped and given word category (by articles, nouns, pronouns, verbs, adjectives, adverbs and so on) and all words contained within the lexicon listed within the categories to which they belong. The entries are like a, an (determiner), be (verb), boy, stick, glass (noun), green, yellow, red (adjectives), I, we, you, he, she, they (pronouns) etc.

In most contemporary grammatical formalisms, the output of parsing is something logically equivalent to a tree, displaying dominance and precedence relations between constituents of a sentence. Parsing algorithms are usually designed for classes of grammar rather than tailored towards individual grammars.

Types of Parsing

The parsing technique can be categorized into two types such as

1. Top down Parsing

2. Bottom up Parsing

Let us discuss about these two parsing techniques and how they will work for input sentences.

1 Top down Parsing

Top down parsing starts with the starting symbol and proceeds towards the goal. We can say it is the process of construction the parse tree starting at the root and proceeds towards the leaves. It is a strategy of analyzing unknown data relationships by hypothesizing general parse tree structures and then considering whether the known fundamental structures are compatible with the hypothesis. In top down parsing words of the sentence are replaced by their categories like verb phrase (VP), Noun phrase (NP), Preposition phrase (PP), Pronoun (PRO) etc. Let us consider some examples to illustrate top down parsing. We will consider both the symbolical representation and the graphical representation. We will take the words of the sentences and reach at the complete sentence. For parsing we will consider the previous symbols like PP, NP, VP, ART, N, V and so on. Examples of top down parsing are LL (Left-to-right, left most derivation), recursive descent parser etc.

Example 1: Rahul is eating an apple.

Symbolical Representation

S → NP VP

□ N VP (∴ NP □ N)

□ N AUX VP (□ VP □ AUX VP)

□ N AUX V NP (□ VP □ V NP)

□□ □UX V ART N (□□P □ ART N)

□□ □UX V ART apple

□ N AUX V an apple

□ N AUX eating an apple

□□ is eating an apple

□ Rahul is eating an apple.

Graphical Representation

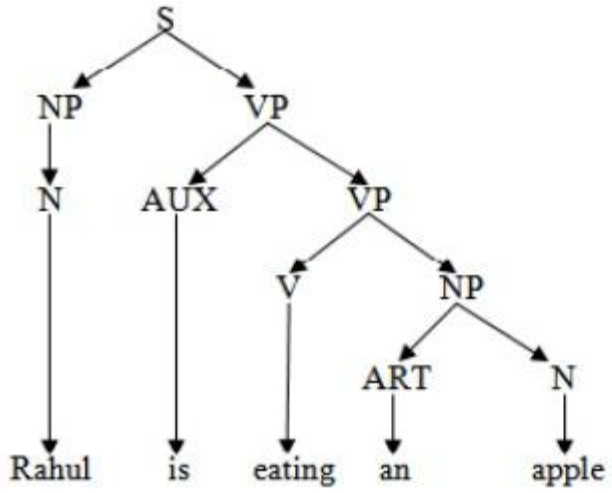


Figure Example of Top down Parsing

Example 2: The small tree shades the new house by the stream.

Symbolical Representation

Ş□NP VP

□ ART NP VP

□ The ADJ N VP

□ The small N V NP

□ The small tree V ART NP

□ The small tree shades ART ADJ NP

□ The small tree shades the ADJ N NP

□ The small tree shades the new N PREP N

□ The small tree shades the new house PREP ART N

□ The small tree shades the new house by ART N

□ The small tree shades the new house by the N

□ The small tree shades the new house by the stream.

Graphical Representation

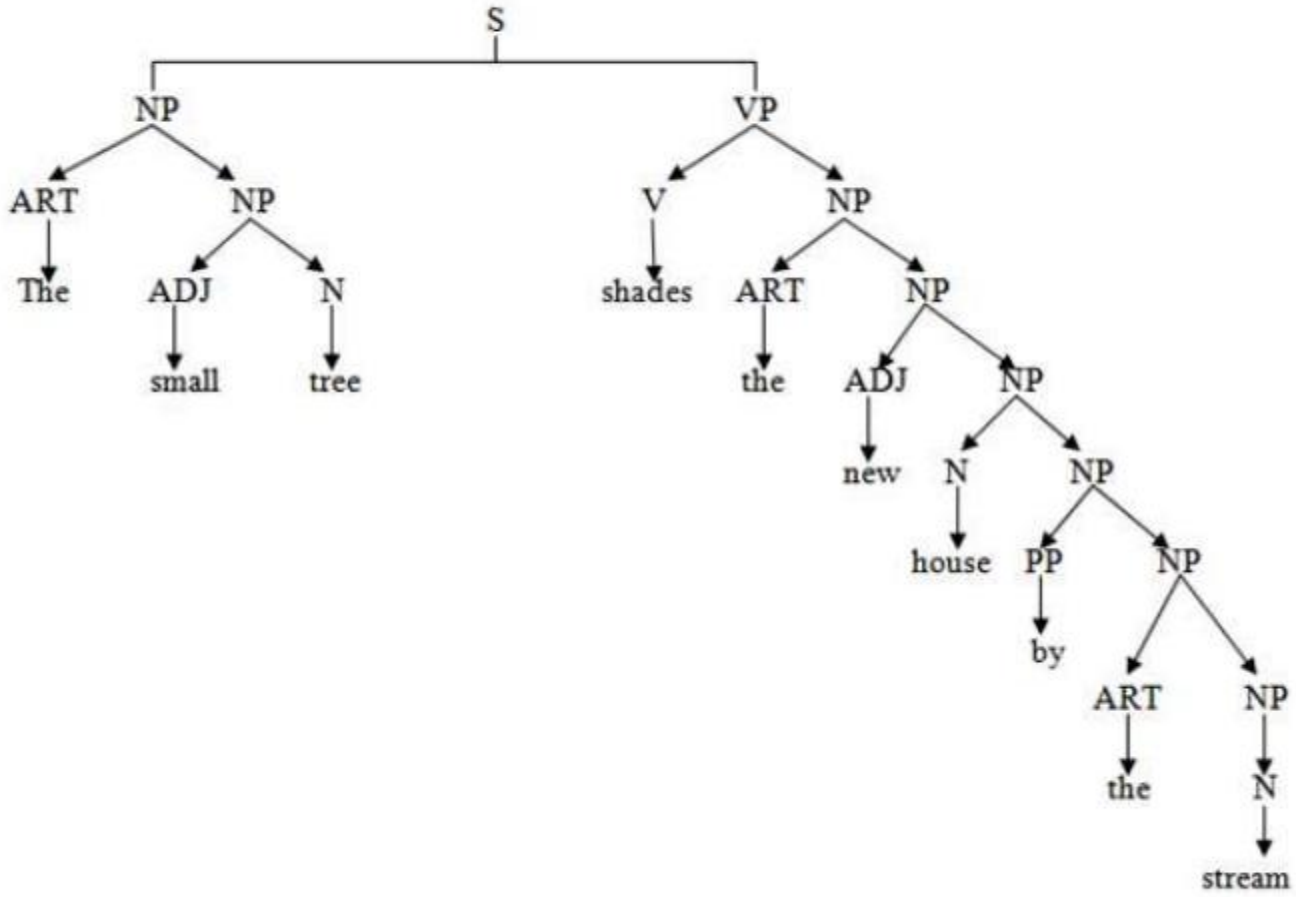


Figure Top down Parsing

2. Bottom up Parsing

In this parsing technique the process begins with the sentence and the words of the sentence is replaced by their relevant symbols. This process was first suggested by Yngve (1955). It is also called shift reducing parsing. In bottom up parsing the construction of parse tree starts at the leaves and proceeds towards the root. Bottom up parsing is a strategy for analyzing unknown data relationships that attempts to identify the most fundamental units first and then to infer higher order structures for them. This process occurs in the analysis of both natural languages and computer languages. It is common for bottom up parsers to take the form of general parsing engines that can wither parse or generate a parser for a specific programming language given a specific of its grammar.

A generalization of this type of algorithm is familiar from computer science LR (k) family can be seen as shift reduce algorithms with a certain amount ("K" words) of look ahead to determine for a set of possible states of the parser which action to take. The sequence of actions from a given grammar can be pre-computed to give a 'parsing table' saying whether a shift or reduce is to be performed and which state to go next. Generally bottom up algorithms are more efficient than top down algorithms, one particular phenomenon that they deal with only clumsily are "e mpty rules": rules in which the right hand side is the empty string. Bottom up parsers find instances of such rules applying at every possible point in the input which can lead to much wasted effort. Let us see some examples to illustrate the bottom up parsing.

Example-1:	Rahul	is	eating	an	apple.
	□□	is	eating	an	apple.
	□N	AUX	eating	an	apple.
	□N	AUX	V	an	apple.
	□N	AUX	V	ART	apple.
	□N	AUX	V	ART	N
	□N	AUX	V	NP	
	□N	AUX	VP		
	□N	VP			
	□NP	VP			
	□S				

Graphical Representation

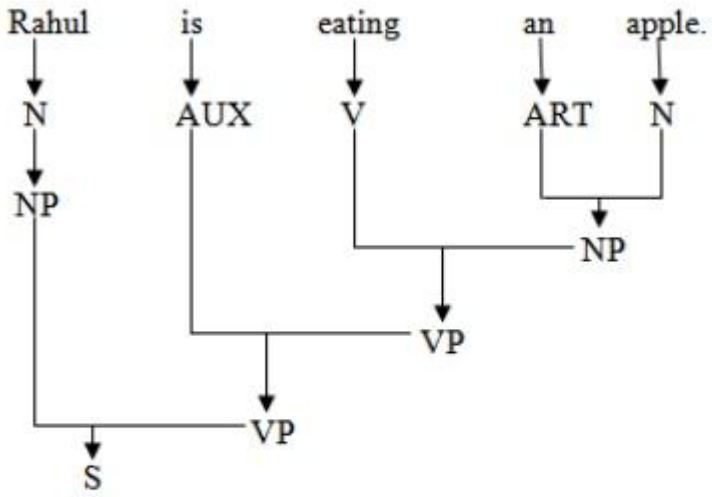


Figure Examples of Bottom up Parsing

Example-2:

□The small tree shades the new house by the stream

□ART small tree shades the new house by the stream

□ART ADJ tree shades the new house by the stream

□ART ADJ N shades the new house by the stream

□ART ADJ N V the new house by the stream

□ART ADJ N V ART new house by the stream

□ART ADJ N V ART ADJ house by the stream

□ART ADJ N V ART ADJ N by the stream

□ART ADJ N V ART ADJ N PREP the stream

□ART ADJ N V ART ADJ N PREP ART stream

□ART ADJ N V ART ADJ N PREP ART N

□ART ADJ N V ART ADJ N PREP NP

□ART ADJ N V ART ADJ N PP

□ART ADJ N V ART ADJ N PP

□ART ADJ N V ART ADJ NP

□ART ADJ N V ART NP

□ART ADJ N V NP

□ART ADJ N VP

□ART NP VP

□NP VP

□S

Graphical Representation

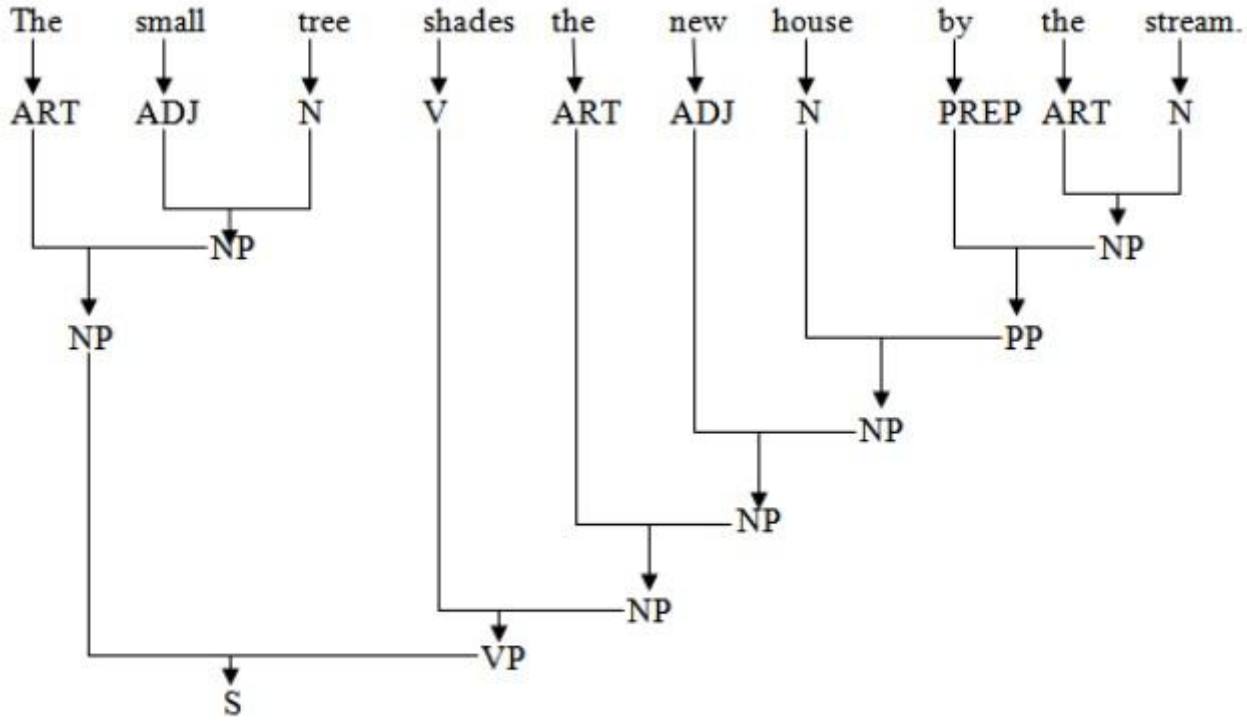


Figure Example of Bottom up Parsing

Example-2:

The small tree shades the new house by the stream

ART small tree shades the new house by the stream

ART ADJ tree shades the new house by the stream

ART ADJ N shades the new house by the stream

ART ADJ N V the new house by the stream

ART ADJ N V ART new house by the stream

ART ADJ N V ART ADJ house by the stream

ART ADJ N V ART ADJ N by the stream

ART ADJ N V ART ADJ N PREP the stream

ART ADJ N V ART ADJ N PREP ART stream

ART ADJ N V ART ADJ N PREP ART N

ART ADJ N V ART ADJ N PREP NP

ART ADJ N V ART ADJ N PP

ART ADJ N V ART ADJ NP

ART ADJ N V ART NP

ART ADJ N V NP

ART ADJ N VP

ART NP VP

NP VP

S

Deterministic Parsing

A deterministic parser is one which permits only one choice for each word category. That means there is only one replacement possibility for every word category. Thus, each word has a different test conditions. At each stage of parsing always the correct choice is to be taken. In deterministic parsing back tracking to some previous positions is not possible. Always the parser has to move forward. Suppose the parser some form of incorrect choice, then the parser will not proceed forward. This situation arises when one word satisfies more than one word categories, such as noun and verb or adjective and verb. The deterministic parsing network is shown in figure.

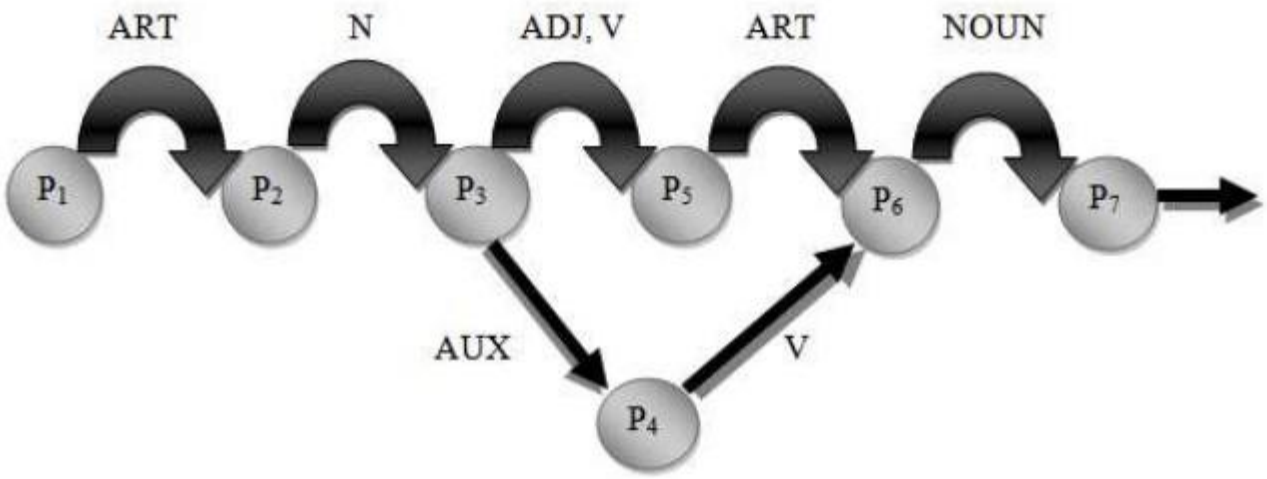


Figure A deterministic Network

Non-Deterministic Parsing

The non deterministic parsing allows different arcs to be labeled with the some test. Thus, they can uniquely make the choice about the next arc to be taken. In non deterministic parsing, the back tracking procedure can be possible. Suppose at some extent of point, the parser does not find the correct word, then at that stage it may backtracks to some of its previous nodes and then start parsing. But the parser has to guess about the proper constituent and then backtrack if the guess is later proven to be wrong. So comparative to deterministic parsing, this procedure may be helpful for a number of sentences as it can backtrack at any point of state. A non deterministic parsing network is shown in figure.

TRANSITION NETWORK

It is a method to represent the natural languages. It is based on applications of directed graphs and finite

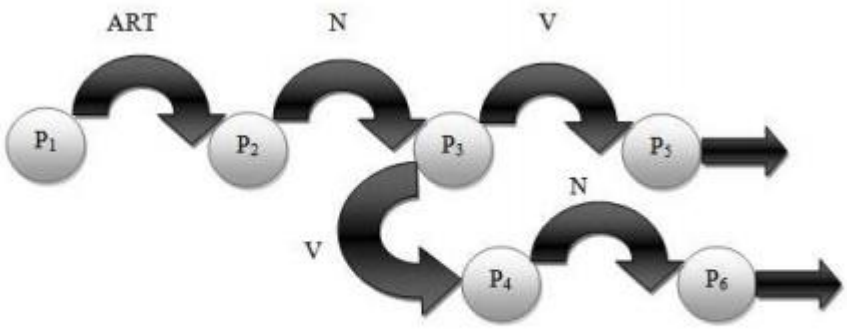


Figure Non-Deterministic Parsing Network

state automata. The transition network can be constructed by the help of some inputs, states and outputs. A transition network may consist of some states or nodes, some labeled arcs from one state to the next state through which it will move. The arc represents the rule or some conditions upon which the transition is made from one state to another state. For example, a transition network is used to recognize a sentence consisting of an article, a noun, an auxiliary, a verb, an article, a noun would be represented by the transition network as follows.

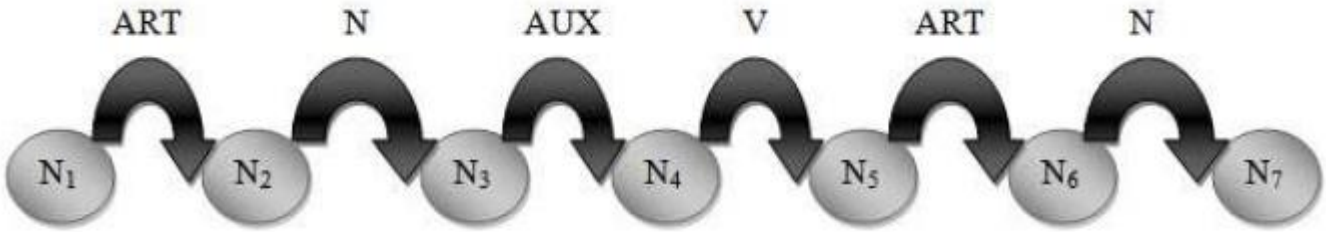


Figure Transition Network

The transition from N_1 to N_2 will be made if an article is the first input symbol. If successful, state N_2 is entered. The transition from N_2 to N_3 can be made if a noun is found next. If successful, state N_3 is entered. The transition from N_3 to N_4 can be made if an auxiliary is found and so on. Suppose consider a sentence "A boy is eating a banana". So if the sent ence is parsed in the above transition network then, first 'A' is an article. So successful transition t o the node N_1 to N_2. Then boy is a noun (so N_2 to N_3), "is" is an auxiliary (N_5 to N_6) and finally "banana" is a noun (N_6 to N_7) is done successfully. So the above sentence is successfully parsed in the transition network.

TYPES OF TRANSITION NETWORK

There are generally two types of transition networks like

1. Recursive Transition networks (RTN)

2. Augmented Transition networks (ATN)

Let us focus on these two transition networks and their structure for parsing a sentence.

1. Recursive Transition Networks (RTN)

RTNs are considered as development for finite state automata with some essential conditions to take the recursive complexion for some definitions in consideration. A recursive transition network consists of nodes (states) and labeled arcs (transitions). It permits arc labels to refer to other networks and they in turn may refer back to the referring network rather than just permitting word categories. It is a modified version of transition network. It allows arc labels that refer to other networks rather than word category. A recursive transition network can have 5 types of arcs (Allen's, JM's) like

1) **CAT:** Current word must belong to category.

2) **WORD:** Current word must match label exactly.

3) **PUSH:** Named network must be successfully traversed.

4) **JUMP:** Can always be traversed.

5) **POP:** Can always be traversed and indicates that input string has been accepted by the network. In RTN, one state is specified as a start state. A string is accepted by an RTN if a POP arc is reached and all the input has been consumed. Let us consider a sentence "The stone was dark black".

Here **The:** ART

Stone: ADJ NOUN

Was: VERB

Dark: ADJ

Black: ADJ NOUN

The RTN structure is given in figure

The RTN structure is given in figure

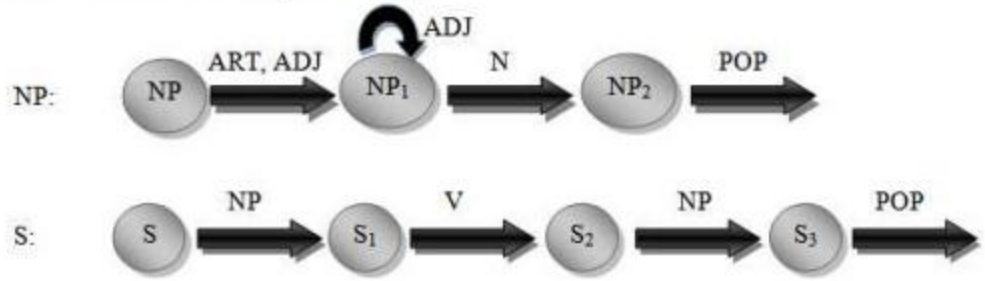


Figure RTN Structure

So we can parse the sentence through the RTN structure as follows.

Current	Word	Return	Arc	Backup States
S	1	-------	NP	----------
NP	1	S_1	ART	----------
NP_1	2	S_1	NOUN	(NP_1, 3, S_1)
NP_2	3	S_1	POP	(NP_1, 3, S_1) (NP_2, 4, S_1)
S_1	3	-------	VERB	(NP_1, 3, S_1) (NP_2, 4, S_1)
S_2	4	-------	NP	(NP_1, 3, S_1) (NP_2, 4, S_1)
NP	4	S_3	NONE	(NP_1, 3, S_1) (NP_2, 4, S_1)
NP_1	3	S_1	NOUN	(NP_2, 4, S_1)
NP_2	4	S_1	POP	(NP_2, 4, S_1)
S_1	4	-------	VERB	(NP_2, 4, S_1)
S_2	5	-------	NP	(NP_2, 4, S_1)
NP	5	S_3	ADJ	(NP_2, 4, S_1)
NP_1	6	S_3	NOUN	(NP_2, 4, S_1) (NP_1, 7, S_3)
NP_2	7	S_3	POP	(NP_2, 4, S_1) (NP_1, 7, S_3)
NP_2	7	S_3	POP	(NP_2, 4, S_1) (NP_1, 7, S_3)
S_3	7	-------	POP	(NP_2, 4, S_1) (NP_1, 7, S_3)

Finally as there are no words left so the parse is successful.

Also there is an another structure of RTN is described by William Woods (1970) is illustrated in figure. He described the total RTN structure into three parts like sentence (S), Noun Phrase (NP), Preposition Phrase (PP).

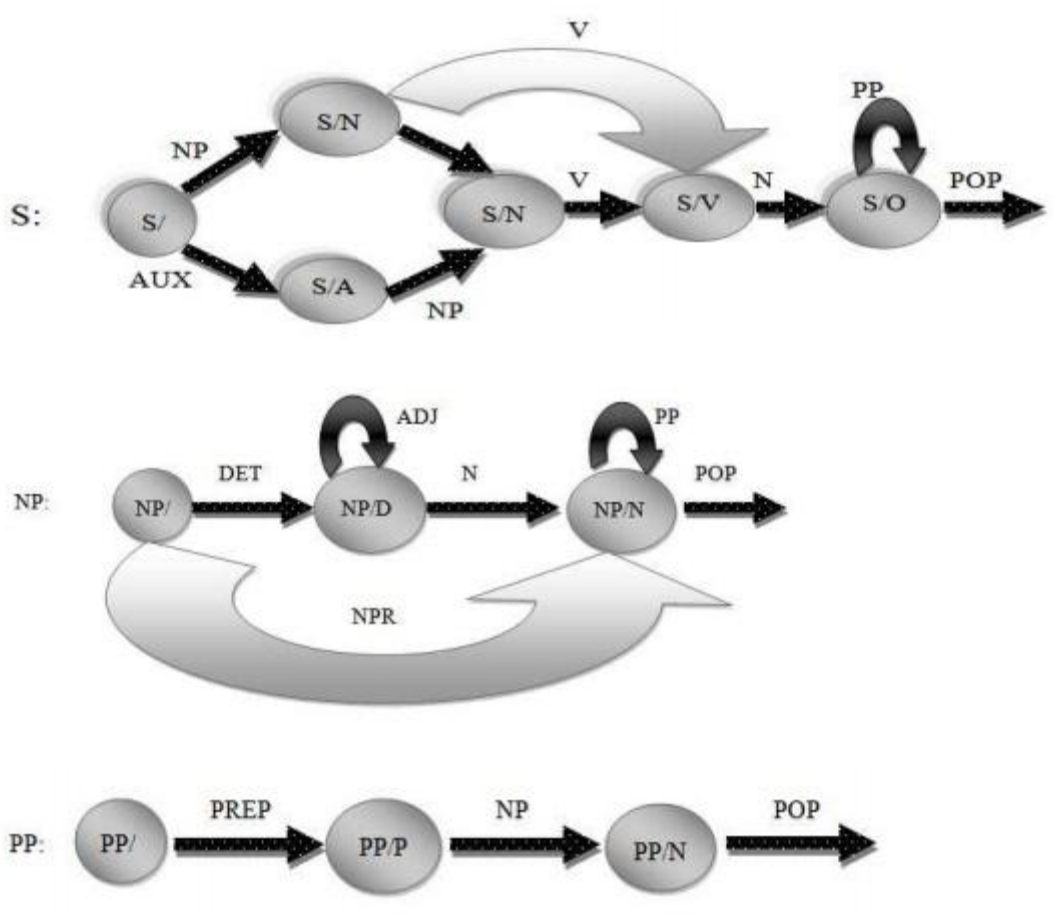


Figure RTN Structure

The number of sentences accepted by an RTN can be extended if backtracking is permitted when a failure occurs. This requires that states having alternative transitions be remembered until the parse progresses past possible failure points. In this way, if a failure occurs at some point, the interpreter can backtrack and try alternative paths. The disadvantage with this approach is that parts of a sentence may be parsed more than time resulting in excessive computations. During the traversal of an RTN, a record must be maintained of the word position in the input sentence and the current state and return nodes to be used as return points when control has been transformed to a lower level network.

2. Augmented Transition Network (ATN)

An ATN is a modified transition network. It is an extension of RTN. The ATN uses a top down parsing procedure to gather various types of information to be later used for understanding system. It produces the data structure suitable for further processing and capable of storing semantic details. An augmented transition network (ATN) is a recursive transition network that can perform tests and take actions during arc transitions. An ATN uses a set of registers to store information. A set of actions is defined for each arc and the actions can look at and modify the registers. An arc may have a test associated with it. The

arc is traversed (and its action) is taken only if the test succeeds. When a lexical arc is traversed, it is put in a special variable (*) that keeps track of the current word. The ATN was first used in LUNAR system. In ATN, the arc can have a further arbitrary test and an arbitrary action. The structure of ATN is illustrated in figure. Like RTN, the structure of ATN is also consisting of the substructures of S, NP and PP.

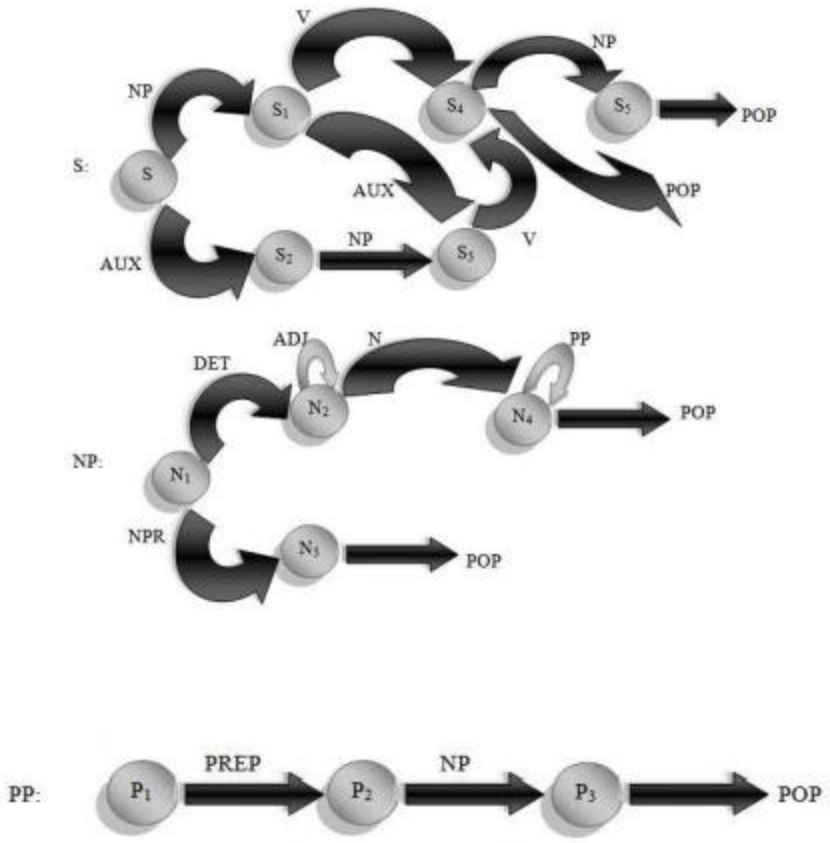


Figure ATN Structure

The ATN collects the sentence features for further analysis. The additional features that can be captured by the ATN are; subject NP, the object NP, the subject verb agreement, the declarative or interrogative mood, tense and so on. So we can conclude that ATN requires some more analysis steps compared to that of RTN. If these extra analysis tests are not performed, then there must some ambiguity in ATN. The ATN represents sentence structure by using a slot filter representation, which reflects more of the functional role of phrases in a sentence. For example, one noun phrase may be identified as “subject” (SUBJ) and another as the “object” of the verb. Wit hin noun phrases, parsing will also identify the determiner structure, adjectives, the noun etc. For the sentence “Ram ate an apple”, we can represent as in figure.

```
(S  SUBJ (NP   NAME    Ram)
         MAIN_V   ate
         TENSE    PAST
         OBJ (NP  DET   an
         HEAD     apple))
```

Figure Representation of sentence in ATN

The ATN maintains the information by having various registers like DET, ADJ and HEAD etc. Registers are set by actions that can be specified on the arcs. When the arc is followed, the specified action associated with it is executed. An ATN can recognize any language that a general purpose computer can recognize. The ATNs have been used successfully in a number of natural language systems as well as front ends for databases and expert systems.

Unit -7

https://interestingengineering.com/ethics-of-ai-benefits-and-risks-ofartificial-intelligence-systems
2. https://royalsocietypublishing.org/doi/full/10.1098/rsta.2018.0080
3. https://law-campbell.libguides.com/ld.php?content_id=58542260