

What is Artificial Intelligence?

In today's world, technology is growing very fast, and we are getting in touch with different new technologies day by day.

Here, one of the booming technologies of computer science is Artificial Intelligence which is ready to create a new revolution in the world by making intelligent machines. The Artificial Intelligence is now all around us. It is currently working with a variety of subfields, ranging from general to specific, such as self-driving cars, playing chess, proving theorems, playing music, Painting, etc.

AI is one of the fascinating and universal fields of Computer science which has a great scope in future. AI holds a tendency to cause a machine to work as a human.

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.

It is believed that AI is not a new technology, and some people says that as per Greek myth, there were Mechanical men in early days which can work and behave like humans.

Why Artificial Intelligence?

Before Learning about Artificial Intelligence, we should know that what is the importance of AI and why should we learn it. Following are some main reasons to learn about AI:

- 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.

What Comprises to Artificial Intelligence?

Artificial Intelligence is not just a part of computer science even it's so vast and requires lots of other factors which can contribute to it. 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

Every technology has some disadvantages, and the same goes for Artificial intelligence. Being so advantageous technology still, it has some disadvantages which we need to keep in our mind while creating an AI system. 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.

Prerequisite

Before learning about Artificial Intelligence, you must have the fundamental knowledge of following so that you can understand the concepts easily:

- Any computer language such as C, C++, Java, Python, etc. (knowledge of Python will be an advantage)
- Knowledge of essential Mathematics such as derivatives, probability theory, etc.

Application of AI

Artificial Intelligence has various applications in today's society. It is becoming essential for today's time because it can solve complex problems with an efficient way in multiple industries, such as Healthcare, entertainment, finance, education, etc. AI is making our daily life more comfortable and fast.

Following are some sectors which have the application of Artificial Intelligence:

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

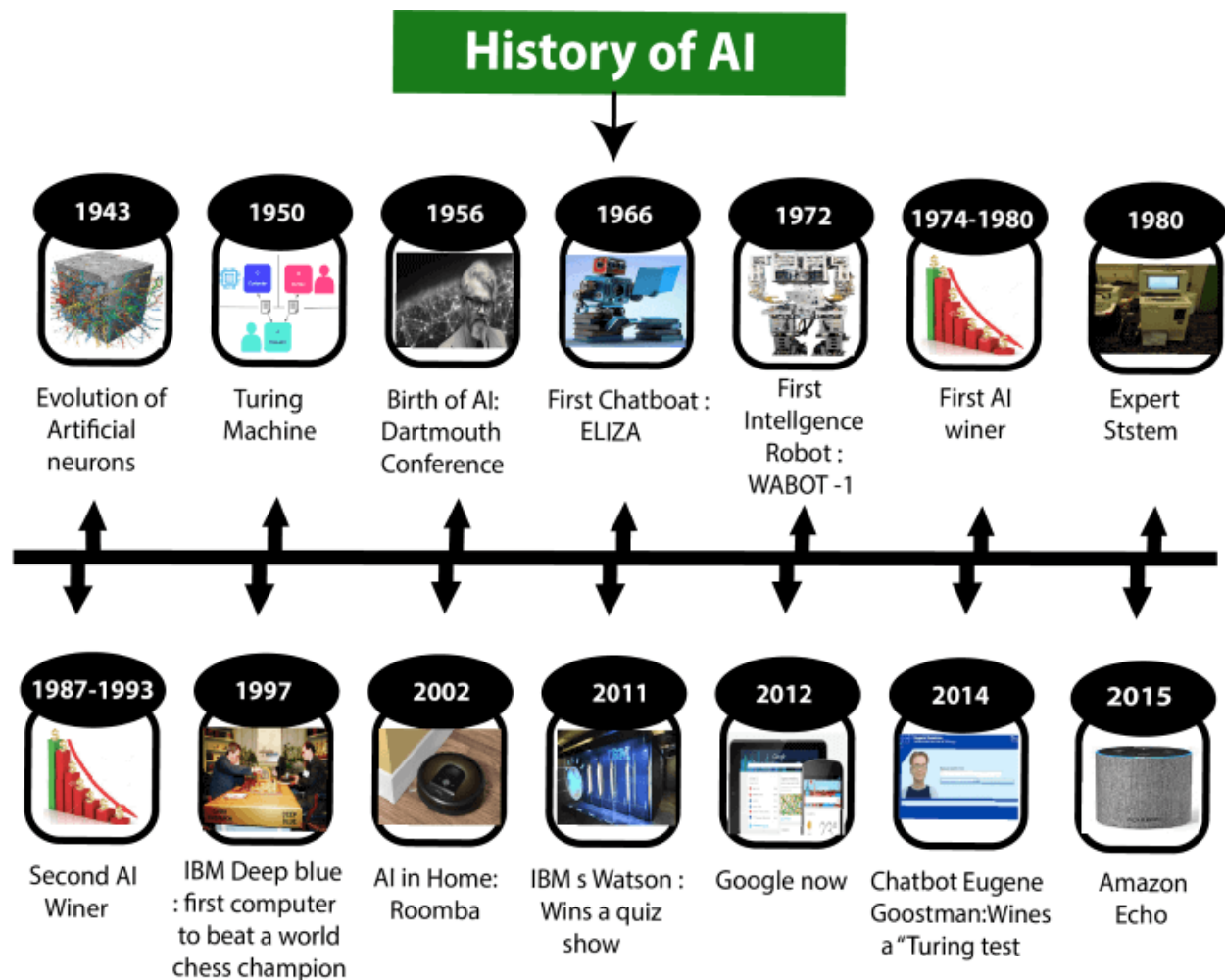
- AI is providing a competitive edge to the e-commerce industry, and it is becoming more demanding in the e-commerce business. AI is helping shoppers to discover associated products with recommended size, color, or even brand.

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

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



Maturation of Artificial Intelligence (1943-1952)

- **Year 1943:** The first work which is now recognized as AI was done by Warren McCulloch and Walter Pitts 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.

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 perceives 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.

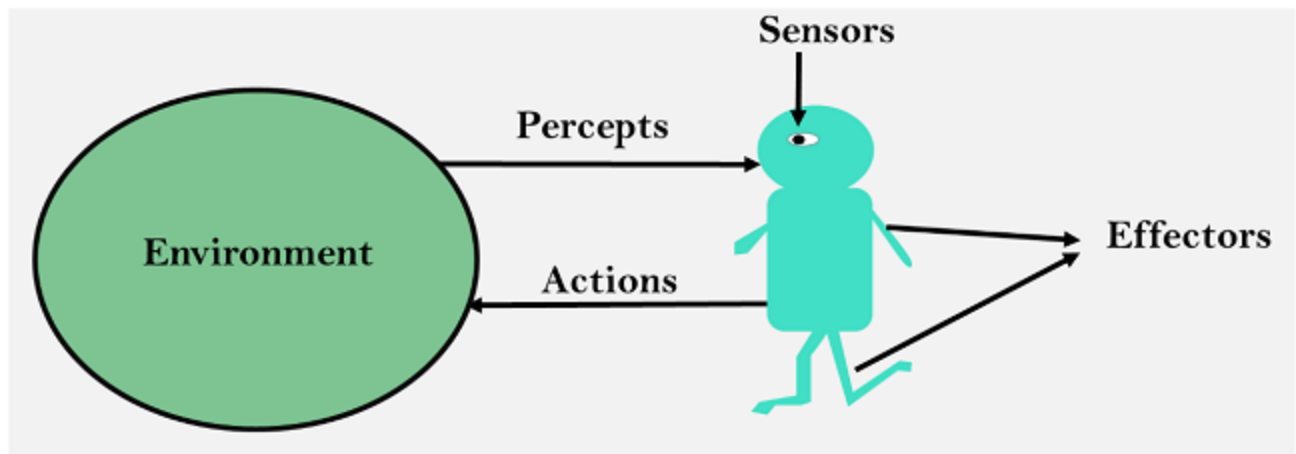
Hence the world around us is full of agents such as thermostat, cellphone, camera, and even we are also agents.

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.

Note: Rational agents in AI are very similar to intelligent agents.

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	<ul style="list-style-type: none">◦ Healthy patient◦ Minimized cost	<ul style="list-style-type: none">◦ Patient◦ Hospital◦ Staff	<ul style="list-style-type: none">◦ Tests◦ Treatments	Keyboard (Entry of symptoms)
2. Vacuum Cleaner	<ul style="list-style-type: none">◦ Cleanness◦ Efficiency◦ Battery life◦ Security	<ul style="list-style-type: none">◦ Room◦ Table◦ Wood floor◦ Carpet◦ Various obstacles	<ul style="list-style-type: none">◦ Wheels◦ Brushes◦ Vacuum Extractor	<ul style="list-style-type: none">◦ Camera◦ Dirt detection sensor◦ Cliff sensor◦ Bump Sensor◦ Infrared Wall Sensor
3. Part -picking Robot	<ul style="list-style-type: none">◦ Percentage of parts in correct bins.	<ul style="list-style-type: none">◦ Conveyor belt with parts,◦ Bins	<ul style="list-style-type: none">◦ Jointed Arms◦ Hand	<ul style="list-style-type: none">◦ Camera◦ Joint angle sensors.

Problem-solving agent

- The problem-solving agent performs precisely by defining problems and its several solutions.
- According to psychology, “*a problem-solving refers to a state where we wish to reach to a definite goal from a present state or condition.*”
- According to computer science, *a problem-solving is a part of artificial intelligence which encompasses a number of techniques such as algorithms, heuristics to solve a problem.*

- Therefore, a problem-solving agent is a **goal-driven agent** and focuses on satisfying the goal.
- **Steps performed by Problem-solving agent**
- **Goal Formulation:** It is the first and simplest step in problem-solving. It organizes the steps/sequence required to formulate one goal out of multiple goals as well as actions to achieve that goal. Goal formulation is based on the current situation and the agent's performance measure (discussed below).
- **Problem Formulation:** It is the most important step of problem-solving which decides what actions should be taken to achieve the formulated goal. There are following five components involved in problem formulation:
 - **Initial State:** It is the starting state or initial step of the agent towards its goal.
 - **Actions:** It is the description of the possible actions available to the agent.
 - **Transition Model:** It describes what each action does.
 - **Goal Test:** It determines if the given state is a goal state.
 - **Path cost:** It assigns a numeric cost to each path that follows the goal. The problem-solving agent selects a cost function, which reflects its performance measure. Remember, **an optimal solution has the lowest path cost among all the solutions.**
- **Note:** Initial state, actions, and transition model together define the **state-space** of the problem implicitly. State-space of a problem is a set of all states which can be reached from the initial state followed by any sequence of actions. The state-space forms a directed map or graph where nodes are the states, links between the nodes are actions, and the path is a sequence of states connected by the sequence of actions.
- **Search:** It identifies all the best possible sequence of actions to reach the goal state from the current state. It takes a problem as an input and returns solution as its output.
- **Solution:** It finds the best algorithm out of various algorithms, which may be proven as the best optimal solution.
- **Execution:** It executes the best optimal solution from the searching algorithms to reach the goal state from the current state.

Properties of Search Algorithms:

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

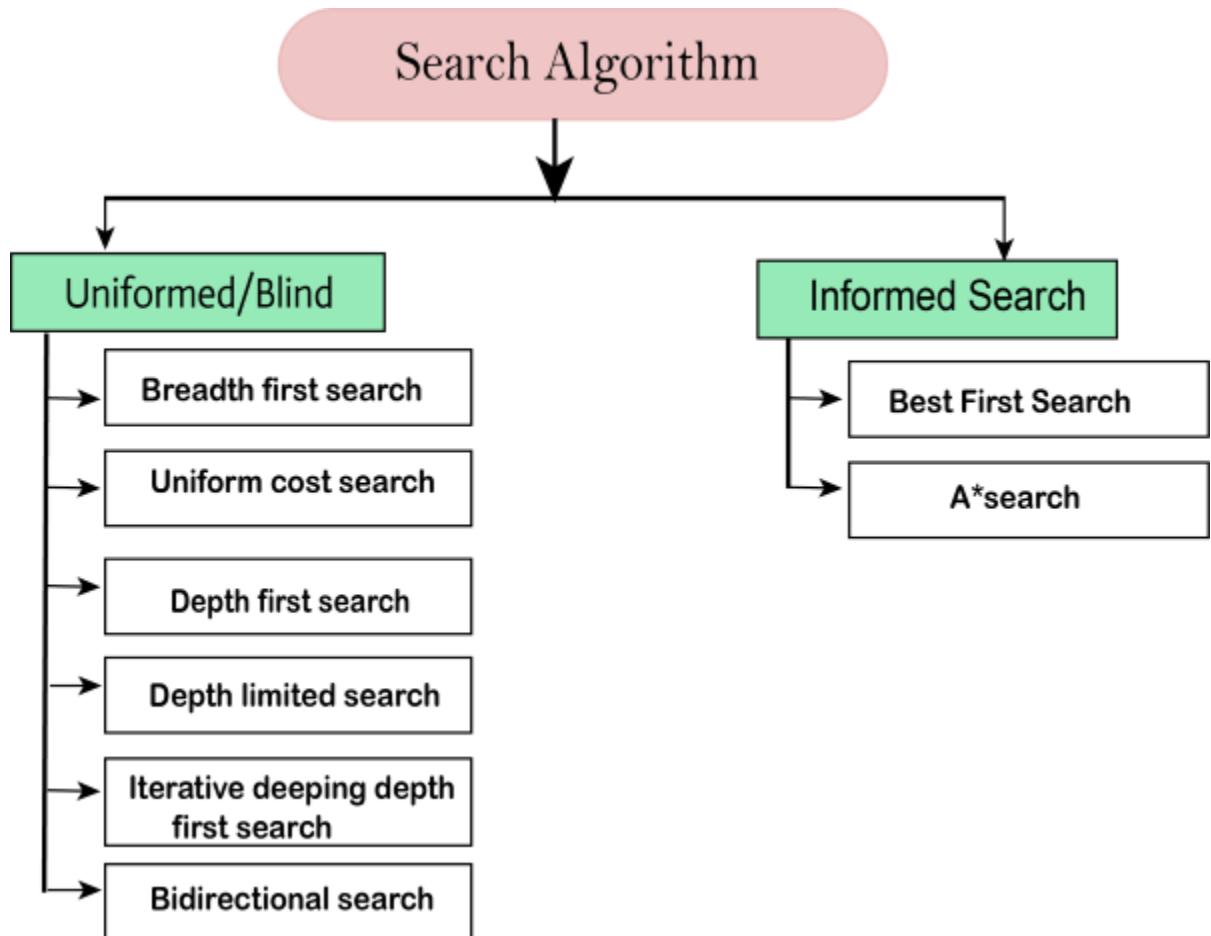
Optimality: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.

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

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

Types of search algorithms

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



Uninformed/Blind Search:

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

It can be divided into five main types:

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

- Bidirectional Search

Informed Search

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

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

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

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

1. Greedy Search
2. A* Search

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
3. Depth-limited Search
4. Iterative deepening depth-first search
5. **Uniform cost search**
6. **Bidirectional 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:

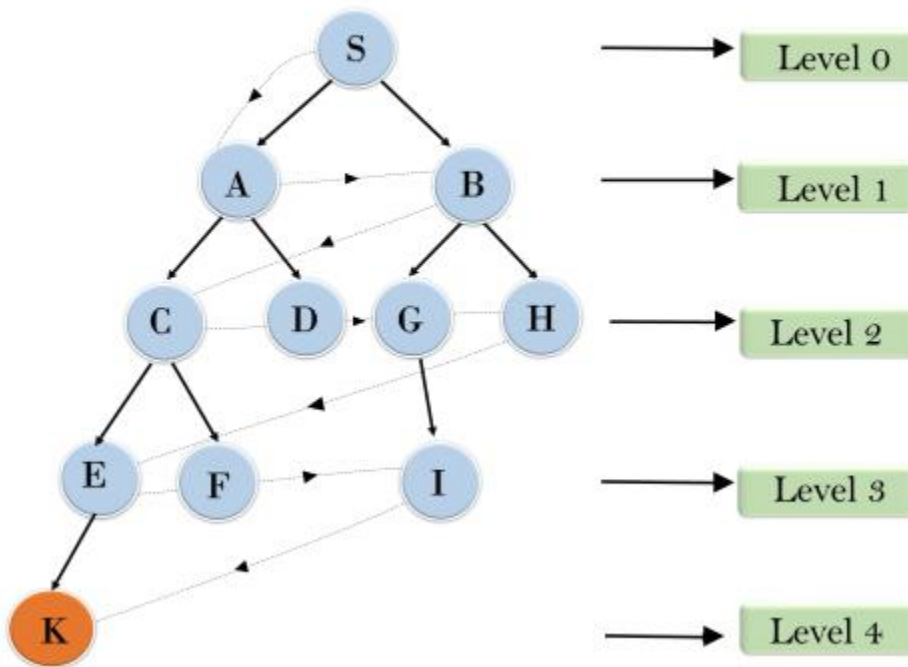
- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

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

Breadth First Search



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.

$$T(b) = 1 + b^1 + b^2 + \dots + b^d = O(b^{d+1})$$

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 is a 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.

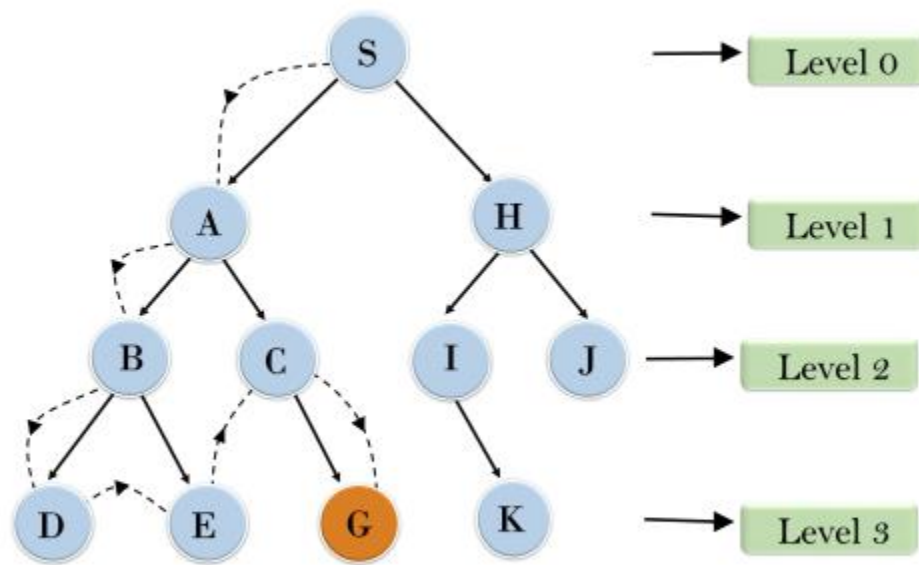
Example:

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.

Depth First Search



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:

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

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. $h(n) \leq h^*(n)$

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 until 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. $f(n) = g(n) + h(n)$

Where, $h(n)$ = 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:

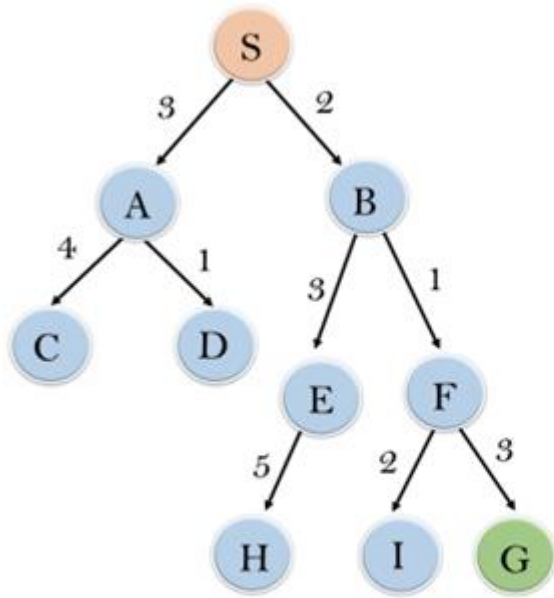
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

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.

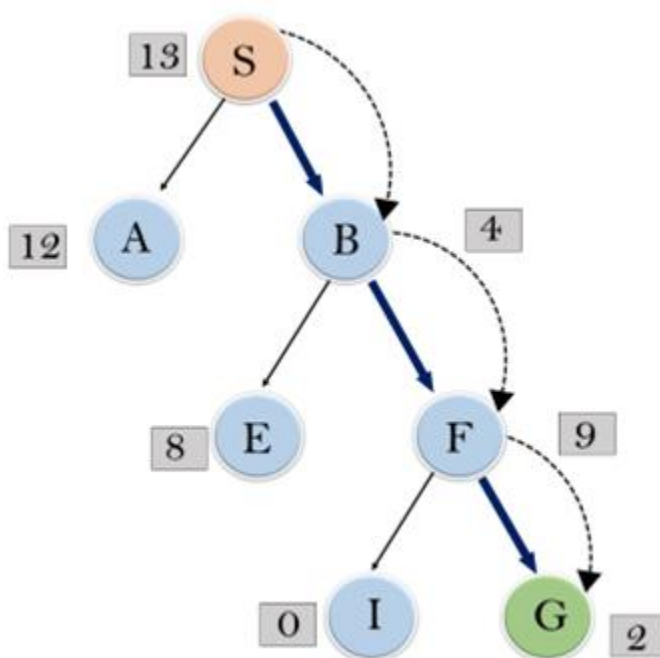
Example:

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.

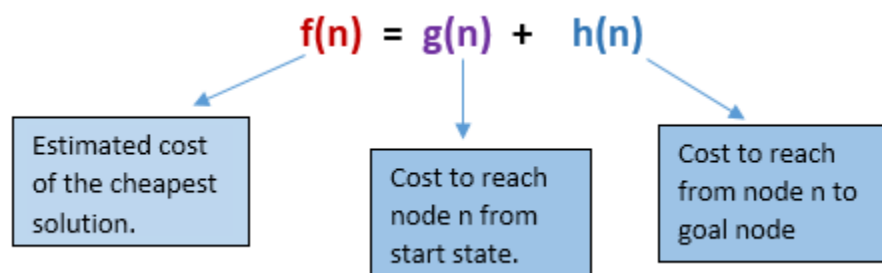
Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

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:

Step 1: 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.

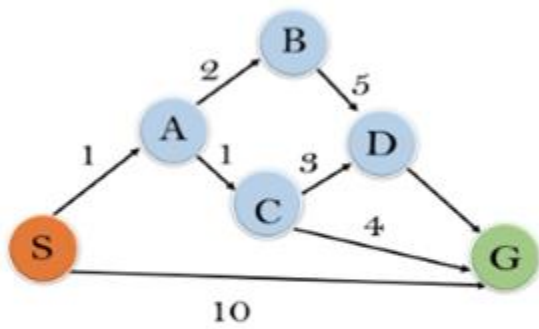
Disadvantages:

- 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:

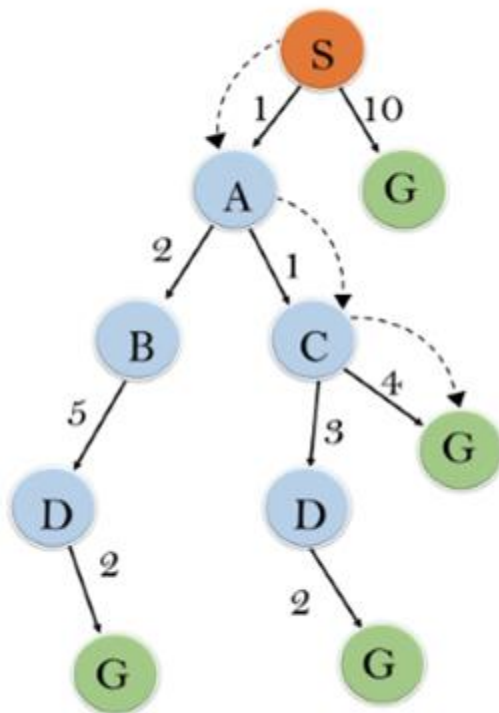
In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the $f(n)$ of each state using the formula $f(n) = g(n) + h(n)$, where $g(n)$ is the cost to reach any node from start state.

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: $\{(S \rightarrow A, 4), (S \rightarrow G, 10)\}$

Iteration2: $\{(S \rightarrow A \rightarrow C, 4), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$

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 $f(n) \leq l_i$

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:

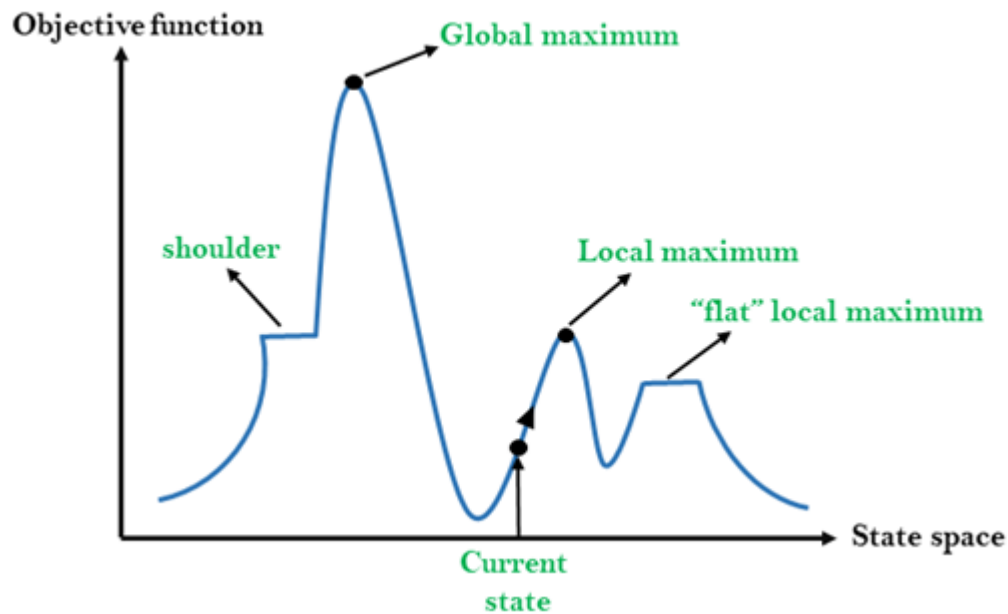
Following are some main features of Hill Climbing Algorithm:

- **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.

- o **Step 5:** Exit.

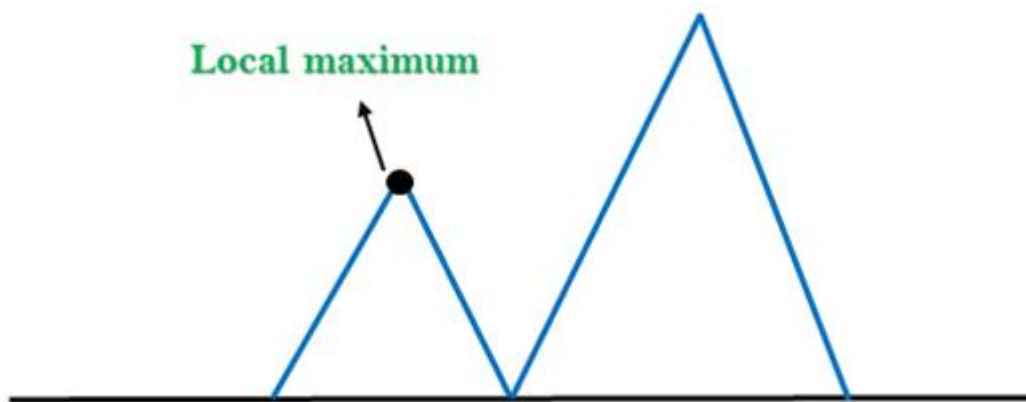
3. Stochastic hill climbing:

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.

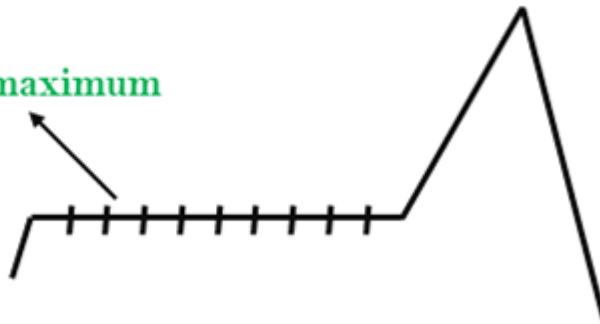
Solution: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.



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.

Plateau/Flat maximum



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.

Ridge



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.

Adversarial Search

Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.

- In previous topics, we have studied the search strategies which are only associated with a single agent that aims to find the solution which often expressed in the form of a sequence of actions.
- But, there might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.
- The environment with more than one agent is termed as **multi-agent environment**, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.
- So, **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games.**
- Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

	Deterministic	Chance Moves
Perfect information	Chess, Checkers, go, Othello	Backgammon, monopoly
Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

Types of Games in AI:

- **Perfect information:** A game with the perfect information is that in which agents can look into the complete board. Agents have all the information about

the game, and they can see each other moves also. Examples are Chess, Checkers, Go, etc.

- **Imperfect information:** If in a game agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information, such as tic-tac-toe, Battleship, blind, Bridge, etc.
- **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examples are chess, Checkers, Go, tic-tac-toe, etc.
- **Non-deterministic games:** Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck. This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed. Such games are also called as stochastic games. Example: Backgammon, Monopoly, Poker, etc.

Note: In this topic, we will discuss deterministic games, fully observable environment, zero-sum, and where each agent acts alternatively.

Mini-Max Algorithm in Artificial Intelligence

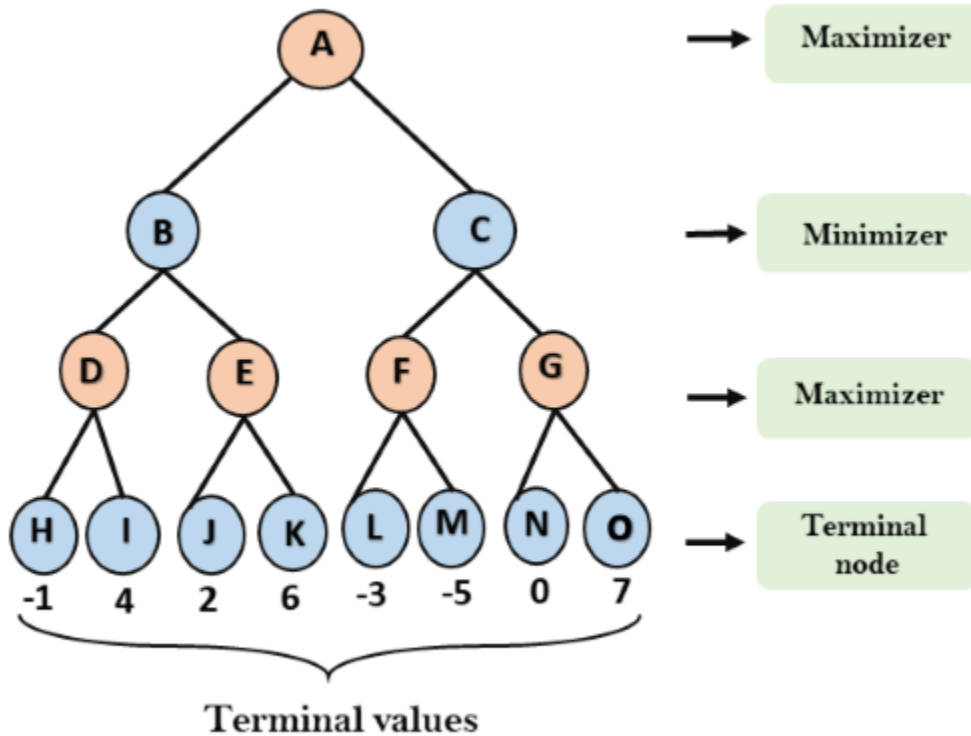
- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly 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.

Working of Min-Max Algorithm:

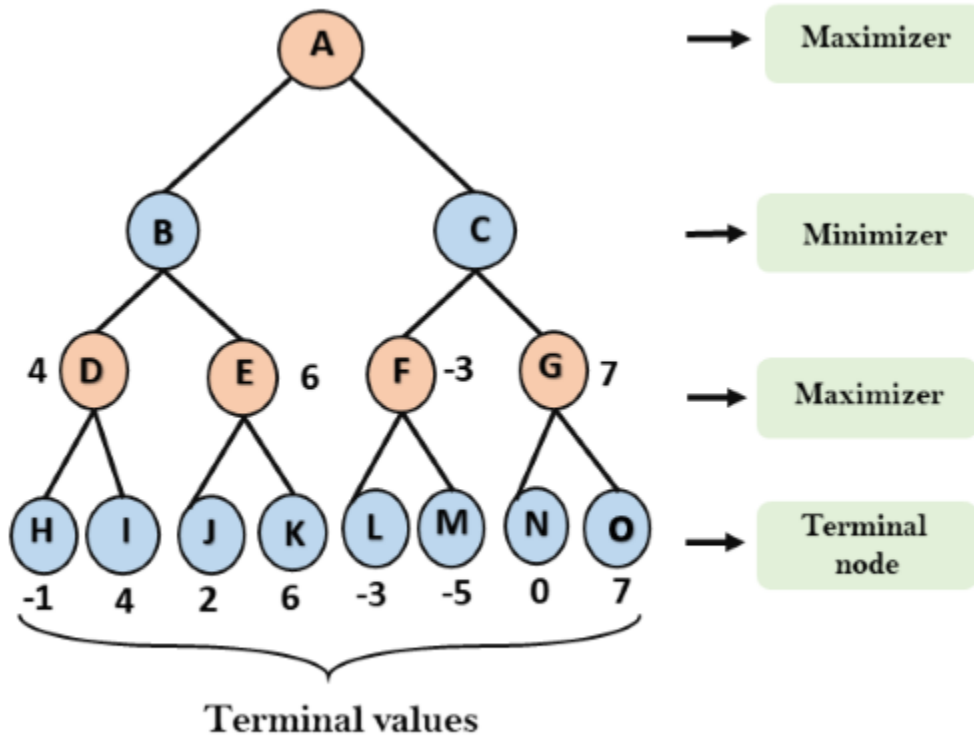
- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum 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.



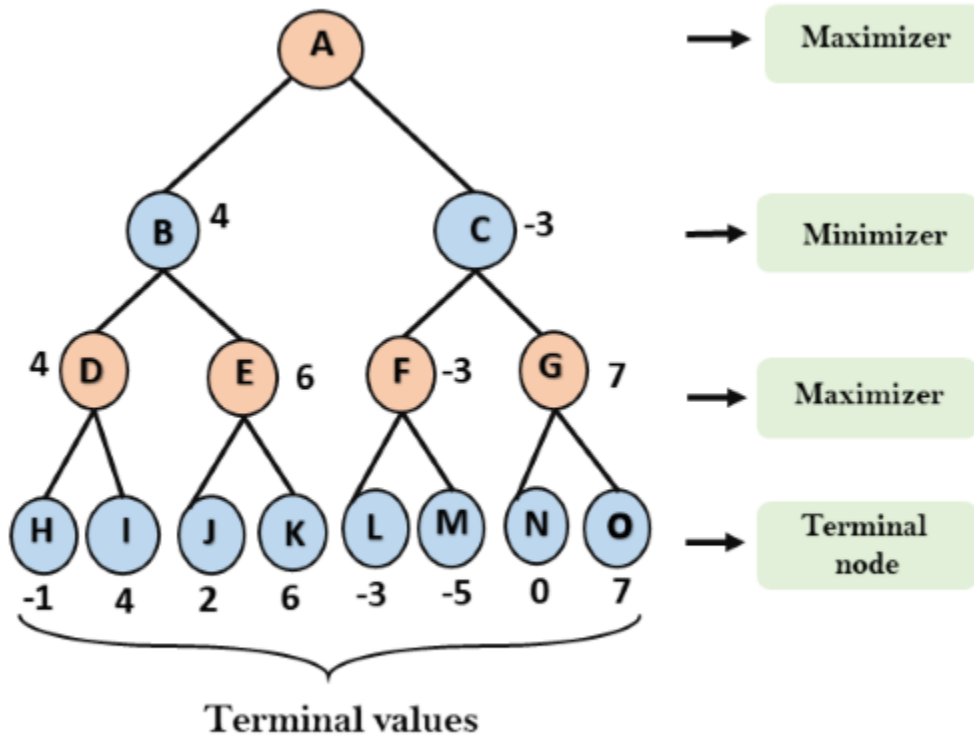
Step 2: Now, first we find the utilities value for the Maximizer, its initial value is $-\infty$, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

- For node D $\max(-1, -\infty) \Rightarrow \max(-1, 4) = 4$
- For Node E $\max(2, -\infty) \Rightarrow \max(2, 6) = 6$
- For Node F $\max(-3, -\infty) \Rightarrow \max(-3, -5) = -3$
- For node G $\max(0, -\infty) = \max(0, 7) = 7$



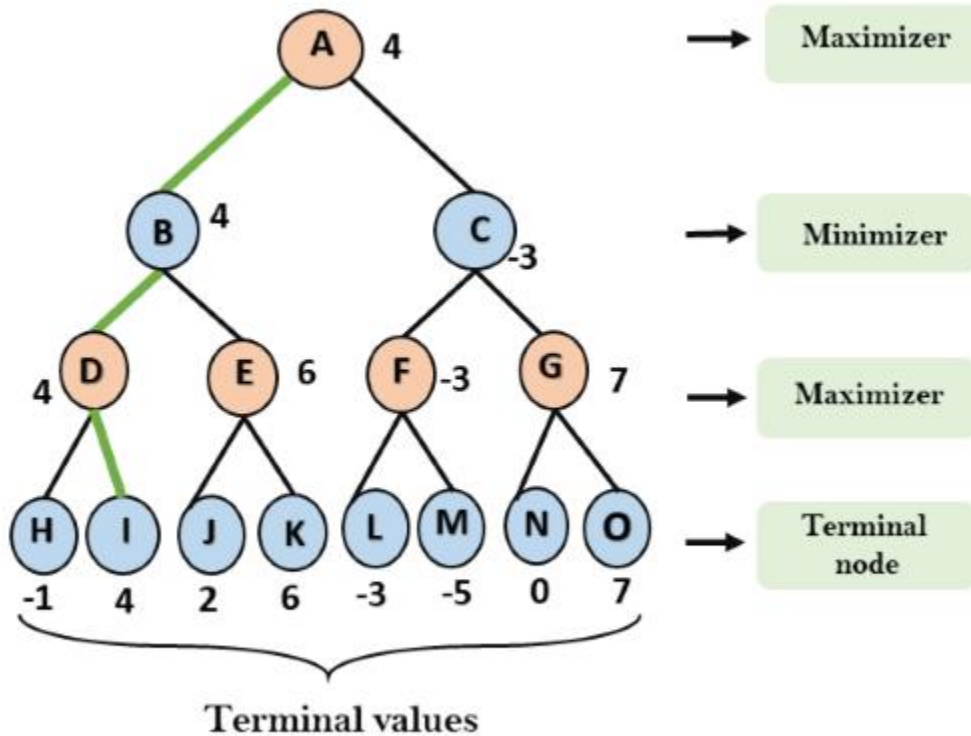
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with $+\infty$, and will find the 3rd layer node values.

- For node B = $\min(4, 6) = 4$
- For node C = $\min(-3, 7) = -3$



Step 3: 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** which we have discussed in the next topic.

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 $-\infty$.
 - 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 $+\infty$.
- 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.

Note: To better understand this topic, kindly study the minimax algorithm.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is:

1. $\alpha \geq \beta$

Key points about alpha-beta pruning:

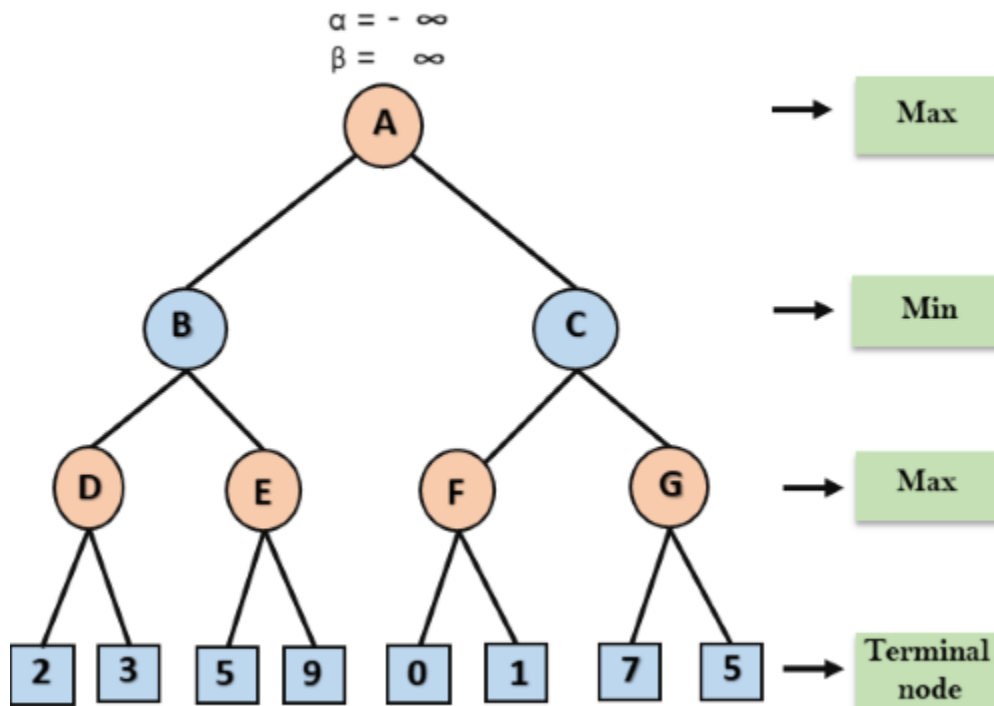
- The Max player will only update the value of alpha.

- 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.

Working of Alpha-Beta Pruning:

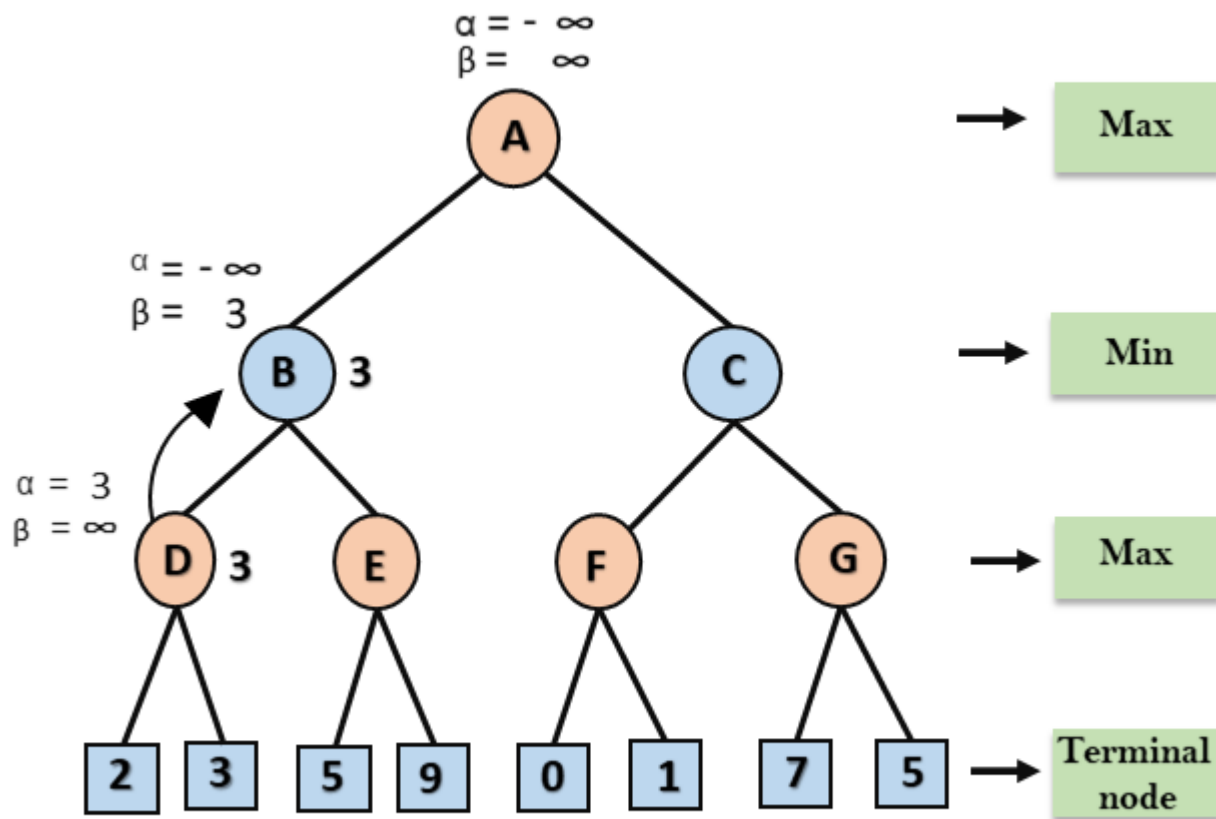
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

Step 1: At the first step the, Max player will start first move from node A where $\alpha = -\infty$ and $\beta = +\infty$, these value of alpha and beta passed down to node B where again $\alpha = -\infty$ and $\beta = +\infty$, and Node B passes the same value to its child D.



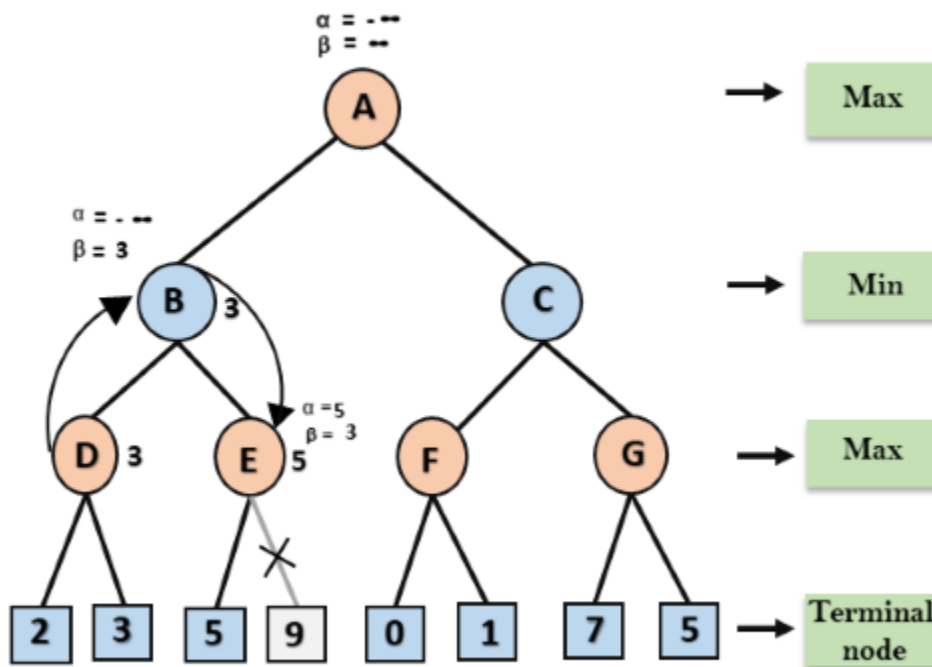
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max $(2, 3) = 3$ will be the value of α at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now $\beta = +\infty$, will compare with the available subsequent nodes value, i.e. $\min(\infty, 3) = 3$, hence at node B now $\alpha = -\infty$, and $\beta = 3$.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of $\alpha = -\infty$, and $\beta = 3$ will also be passed.

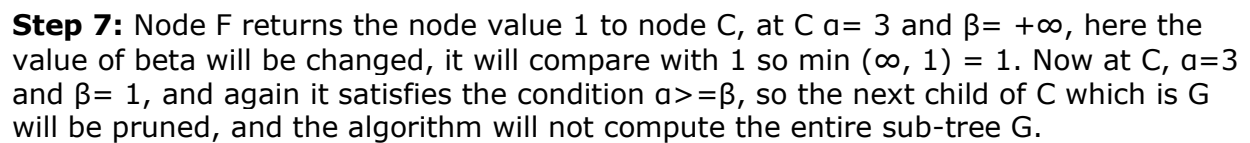
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so $\max(-\infty, 5) = 5$, hence at node E $\alpha = 5$ and $\beta = 3$, where $\alpha \geq \beta$, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.

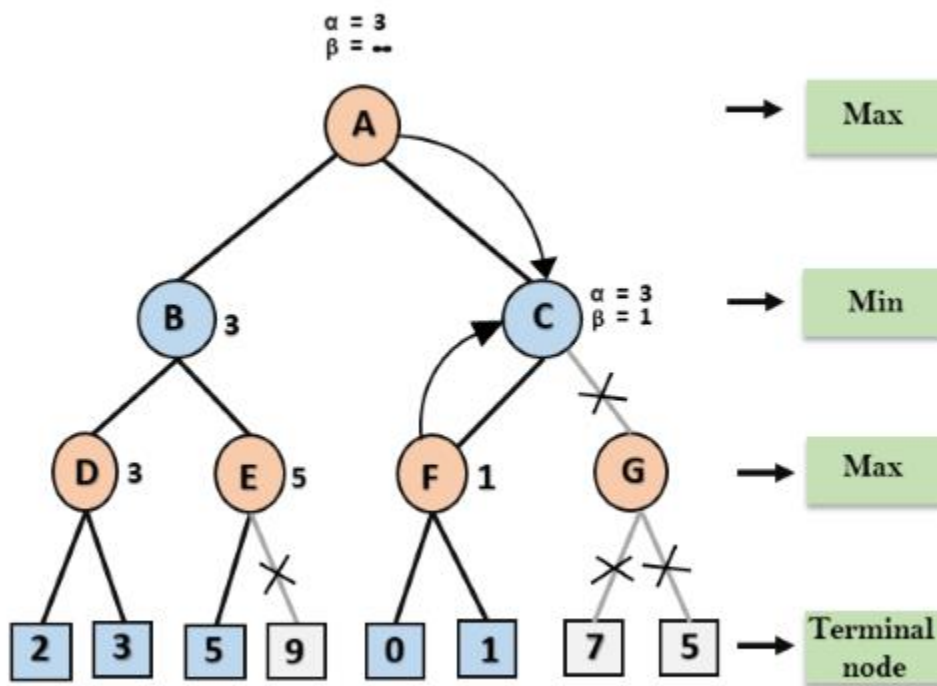


Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as $\max(-\infty, 3) = 3$, and $\beta = +\infty$, these two values now passes to right successor of A which is Node C.

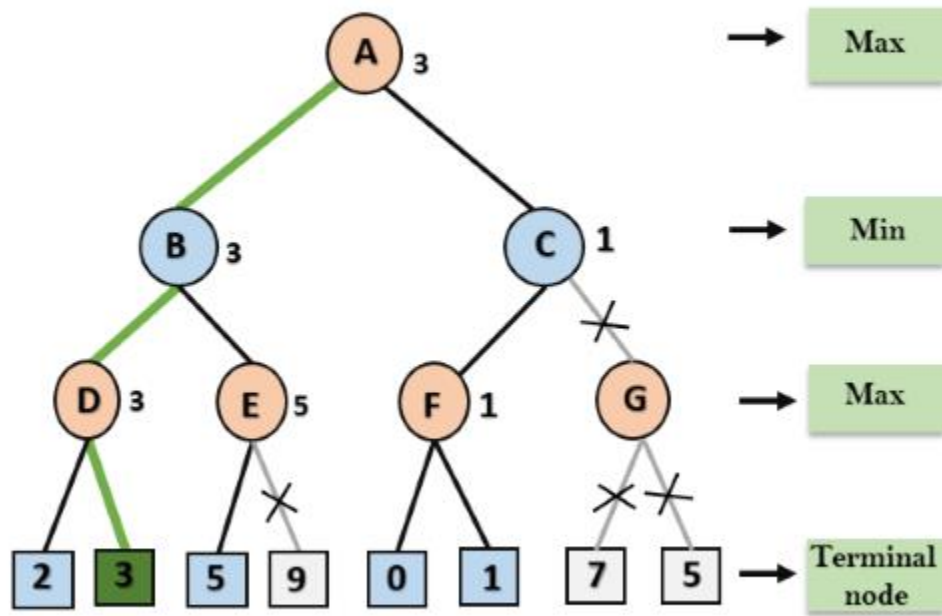
At node C, $\alpha = 3$ and $\beta = +\infty$, and the same values will be passed on to node F.

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 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 showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



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 parser 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 .

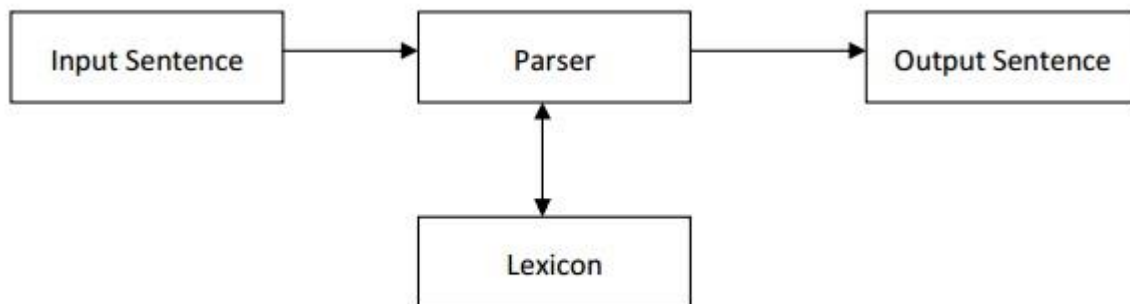


Figure Parsing Technique

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

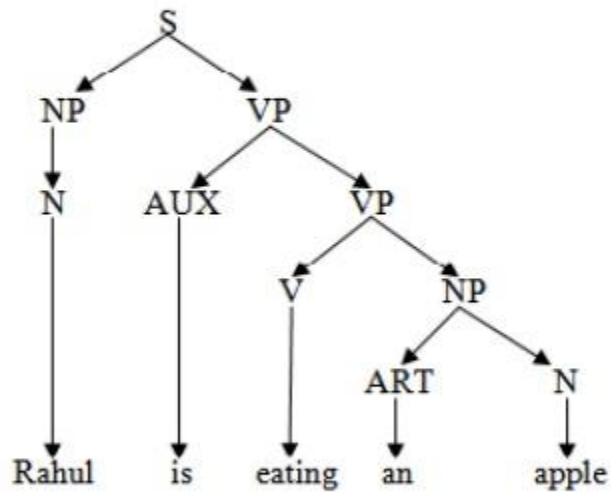


Figure Example of Top down Parsing

Example 2: The small tree shades the new house by the stream.

Symbolical Representation

S → 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

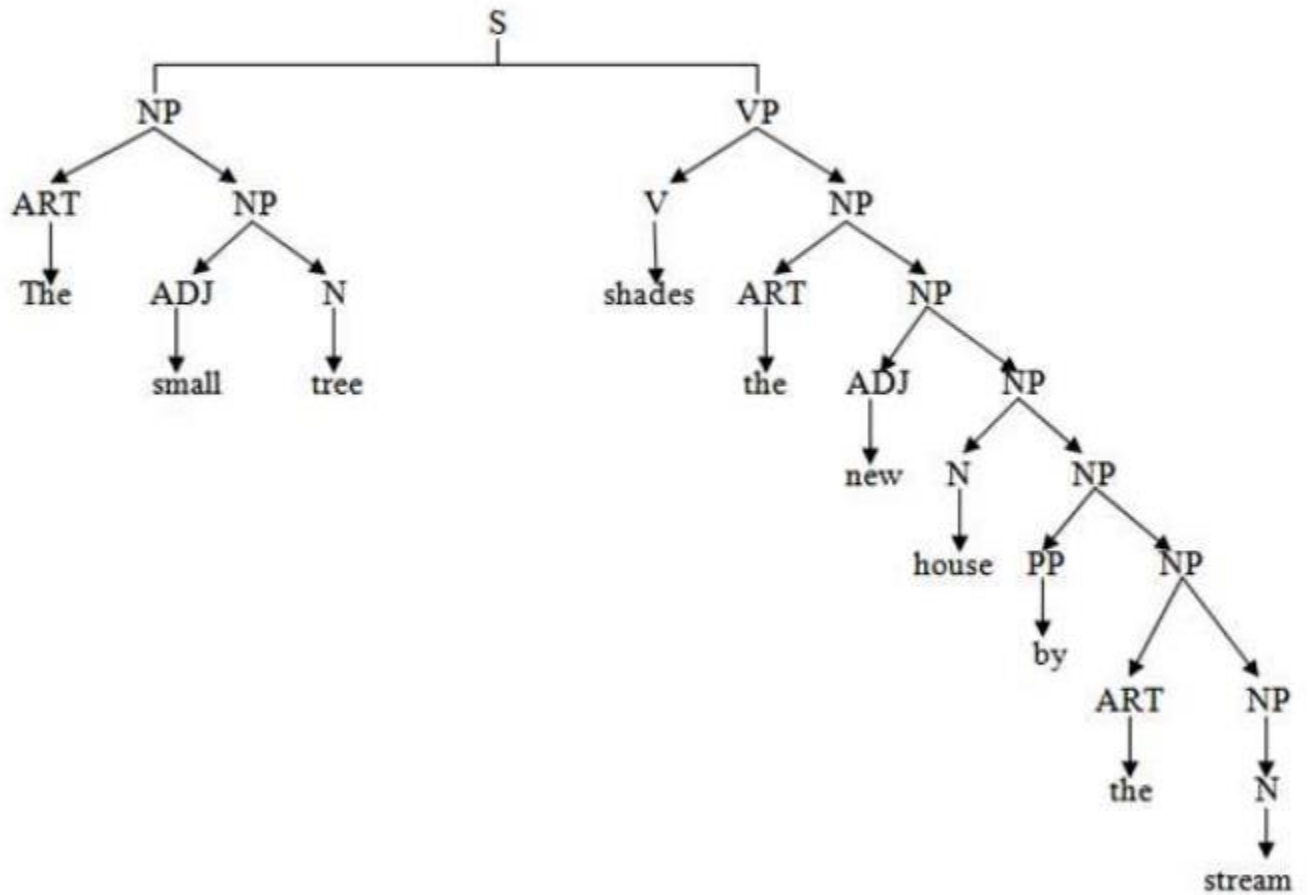


Figure Top down Parsing

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 either 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 “empty 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

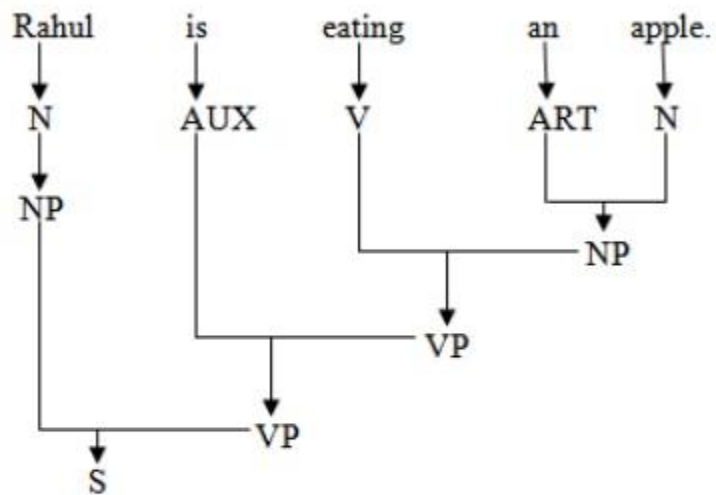


Figure Examples of Bottom up Parsing

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

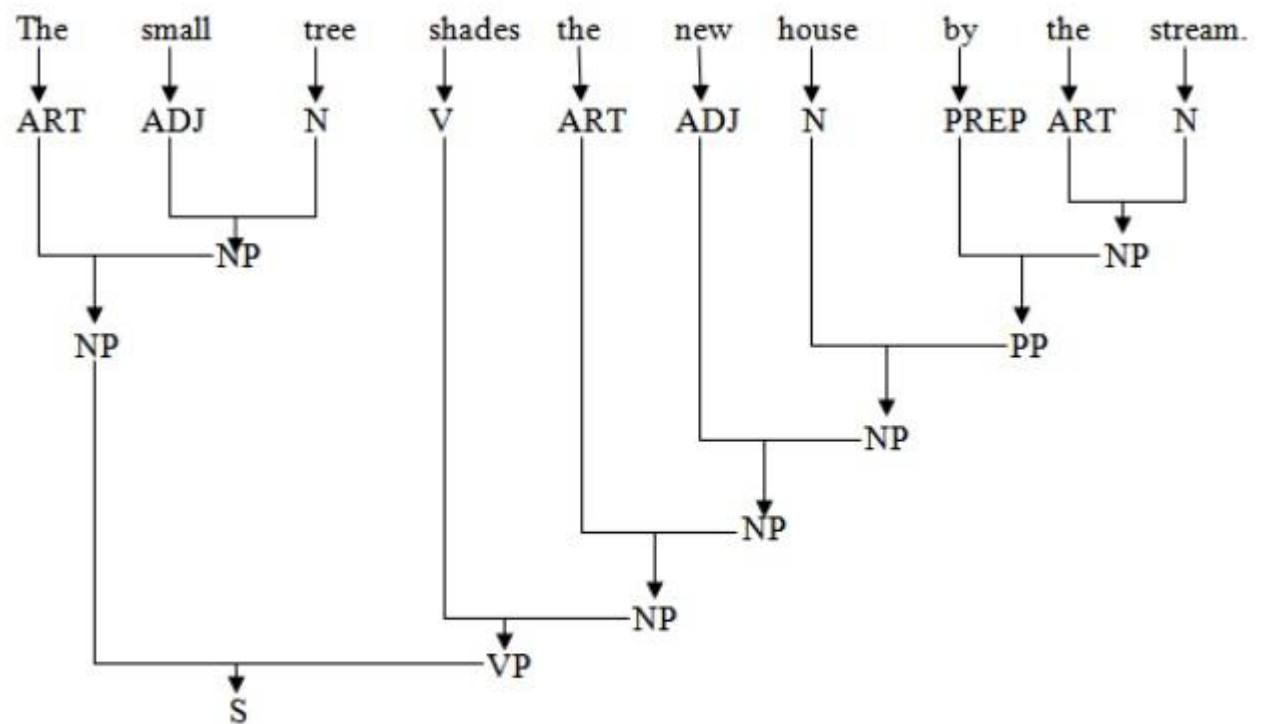


Figure Example of Bottom up Parsing

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.

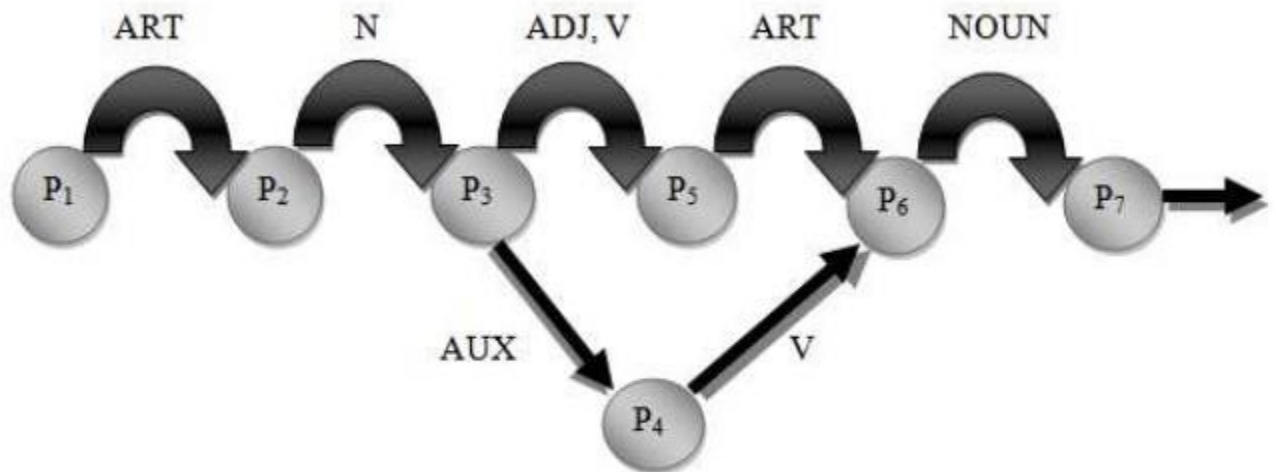


Figure A deterministic Network

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.

Context Free Grammar (CFG)

The grammar in which each production has exactly one terminal symbol in its left hand side and at least one symbol at the right hand side is called context free grammar. A CFG is a four tuple where

Σ : Finite non empty set of terminals, the alphabet.

V : Finite non empty set of grammar variables (categories or non terminal symbols)

Such as $\Sigma \cap V = \emptyset$

S : Starting symbol ($S \in V$)

P : Finite set of production rules, each of the form $A \rightarrow \alpha$, where

$$A \in V \text{ and } \alpha \in (V \cup \Sigma)^*$$

Each terminal symbol in a grammar denotes a language. The non terminals are written in capital letters and terminals are written in small letters. Some properties of CFG formalism are

→ Concatenation is the only string combination operation.

□

→ Phrase structure is the only syntactic relationship.

□

→ The terminal symbols have no properties.

□

→ Non terminal symbols are atomic.

□

→ Most of the information encoded in a grammar lies in the production rules.

□

→ Any attempt of extending the grammar with semantics requires extra means.

□

→ Concatenation is not necessarily the only way by which phrases may be combined to yield other phrases.

□

→ Even if concatenation is the sole string operation, other syntactic relationships are being put forward.

For example we can write the followings:

```
S → NP   VP
NP → ART  N
VP → V    NP
ART → a
N → Cat
N → Meat
V → Eat
```

Transformational Grammar

These are the grammars in which the sentence can be represented structurally into two stages. Obtaining different structures from sentences having the same meaning

is undesirable in language understanding systems. Sentences with the same meaning should always correspond to the same internal knowledge structures. In one stage the basic structure of the sentence is analyzed to determine the grammatical constituent parts and in the second stage just the vice versa of the first one. This reveals the surface structure of the sentence, the way the sentence is used in speech or in writing. Alternatively, we can also say that application of the transformation rules can produce a change from passive voice to active voice and vice versa. Let us see the structure of a sentence as given below.

1. Ram is eating an apple (In Active Voice)

Figure (Transformational grammar Tree representation of Active voice to sentence formation)

2. An apple is being eaten by Ram (In Passive Voice)

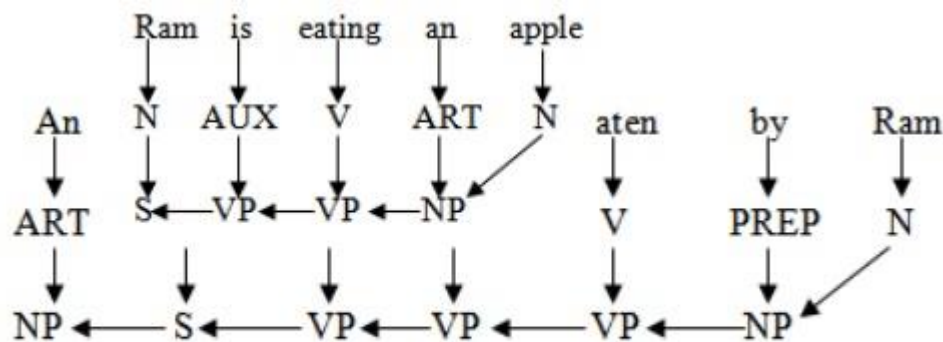


Figure Passive voice to sentence formation

Both of the above sentences are two different sentences but they have same meaning. Thus it is an example of a transformational grammar. These grammars were never widely used in computational models of natural language. The applications of this grammar are changing of voice (Active to Passive and Passive to Active) change a question to declarative form etc.

TRANSITION NETWORK

It is a method to represent the natural languages. It is based on applications of directed graphs and finite

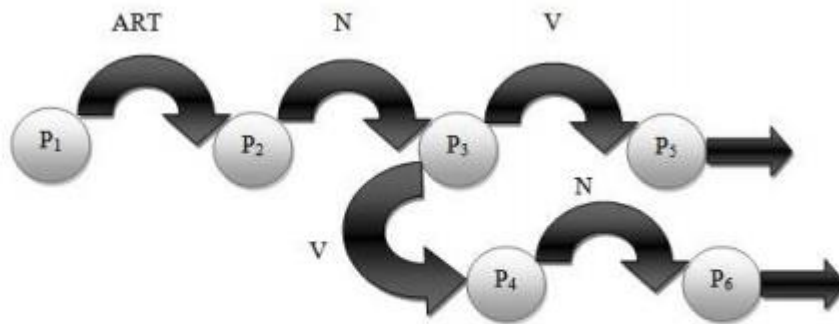


Figure Non-Deterministic Parsing Network

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.

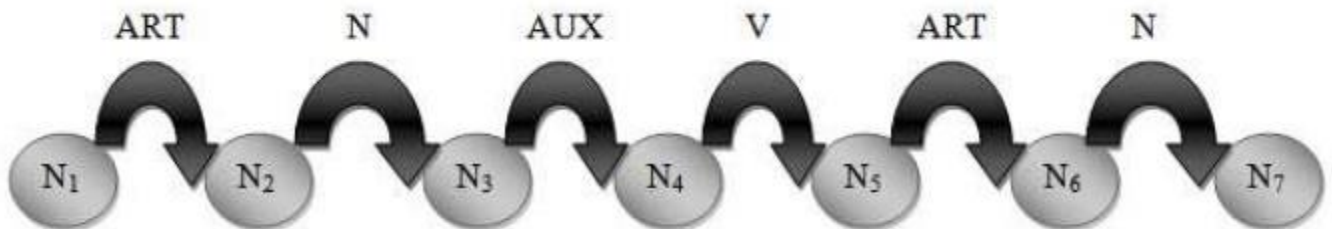


Figure Transition Network

The transition from N₁ to N₂ will be made if an article is the first input symbol. If successful, state N₂ is entered. The transition from N₂ to N₃ can be made if a noun is found next. If successful, state N₃ is entered. The transition from N₃ to N₄ can be made if an auxiliary is found and so on. Suppose consider a sentence “A boy is eating a banana”. So if the sentence is parsed in the above transition network then, first ‘A’ is an article. So successful transition to the node N₁ to N₂. 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.

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.

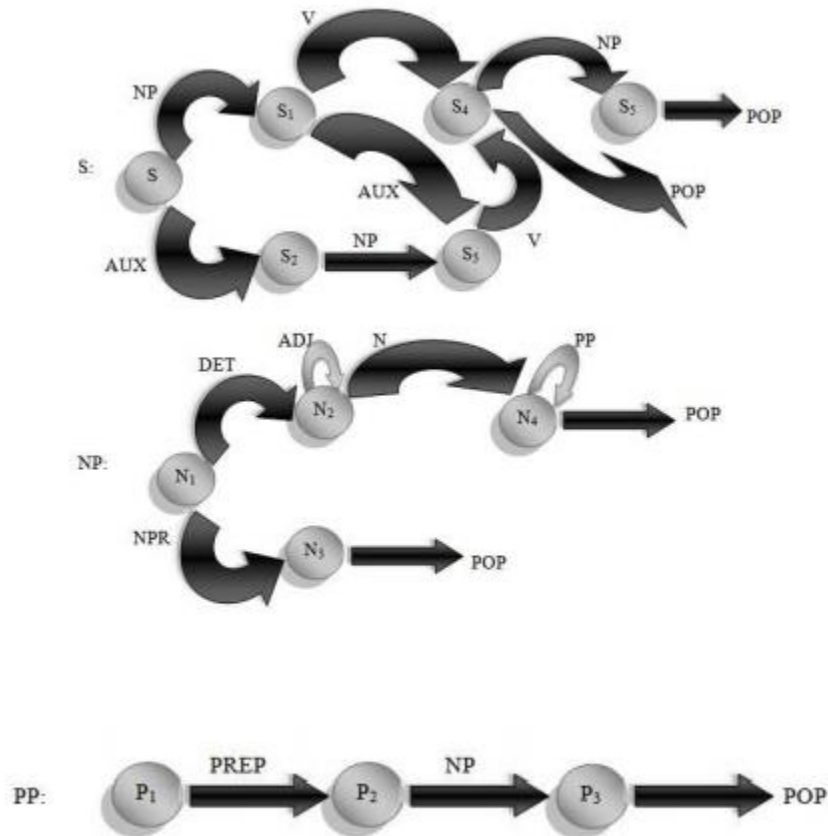


Figure **ATN Structure**

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.

Case Grammars (FILLMORE's Grammar)

Case grammars use the functional relationships between noun phrases and verbs to conduct the more deeper case of a sentence. Generally in our English sentences, the difference between different forms of a sentence is quite negligible. In early 1970's Fillmore gave some idea about different cases of a English sentence. He extended the transformational grammars of Chomsky by focusing more on the semantic aspects of view of a sentence. In case grammars a sentence id defined as being composed of a preposition P, a modality constituent M, composed of mood, tense, aspect, negation and so on. Thus we can represent a sentence like

$$S \rightarrow M + P$$

Where P - Set of relationships among verbs and noun phrases i.e. P = (C=Case)

M - Modality constituent

For example consider a sentence "Ram did not eat th e apple".

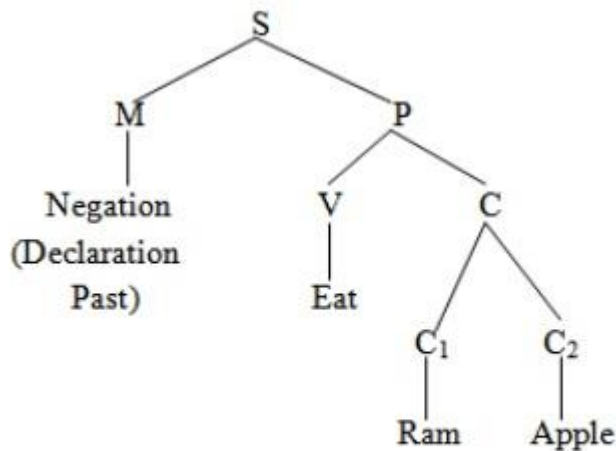


Figure Case Grammar Tree Representation

The tree representation for a case grammar will identify the words by their modality and case. The cases may be related to the actions performed by the agents, the location and direction of actions. The cases may also be instrumental and objective. For example “Ram cuts the apple by a knife”. Here knife is an instrumental case. In fig 8.5 the modality constituent is the negation part, eat is the verb and Ram, apple are nouns which are under the case C₁ and C₂ respectively. Case frames are provided for verbs to identify allowable cases. They give the relationships which are required and which are optional.

bhgggfdgdfg

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:

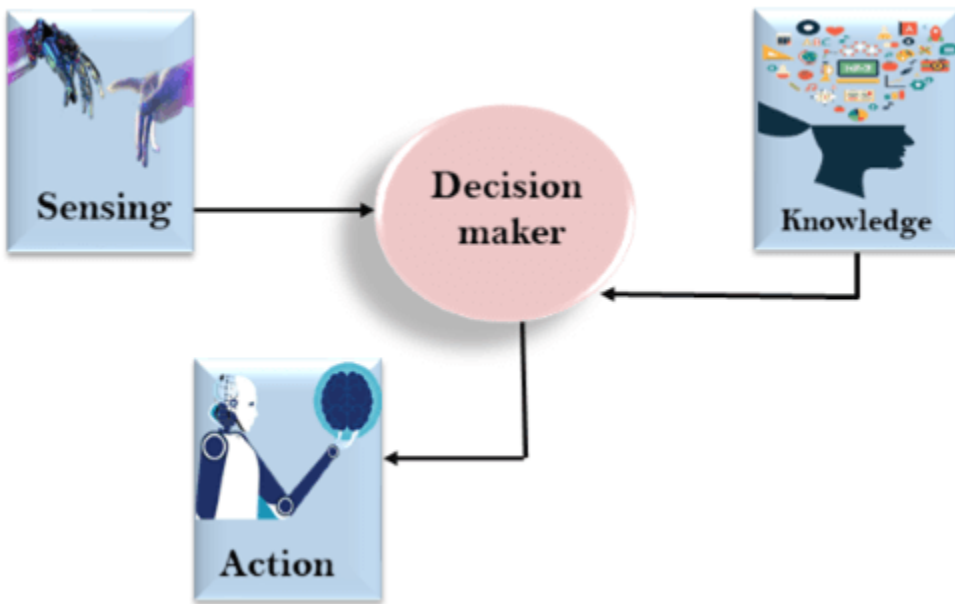
- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

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 be 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 acts by sensing the environment and using knowledge. But if the knowledge part will not be present then, it cannot display intelligent behavior.

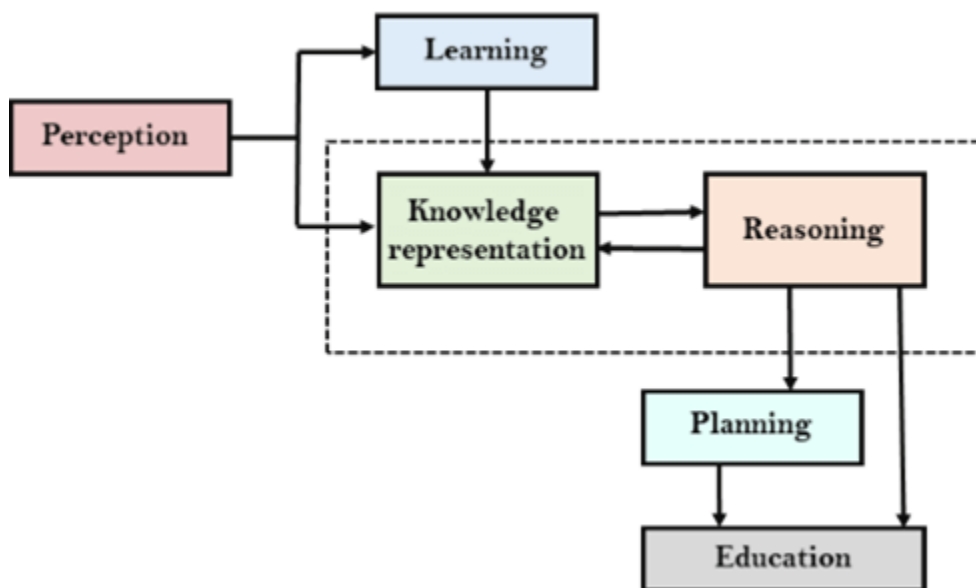


AI

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 component. In the complete cycle, the main components are knowledge representation and Reasoning. These two components are involved in showing the intelligence in machine-like humans. These two components are independent with each other but also coupled together. The planning and execution depend on analysis of Knowledge representation and reasoning.

Approaches to knowledge representation:

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

1. Simple relational knowledge:

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

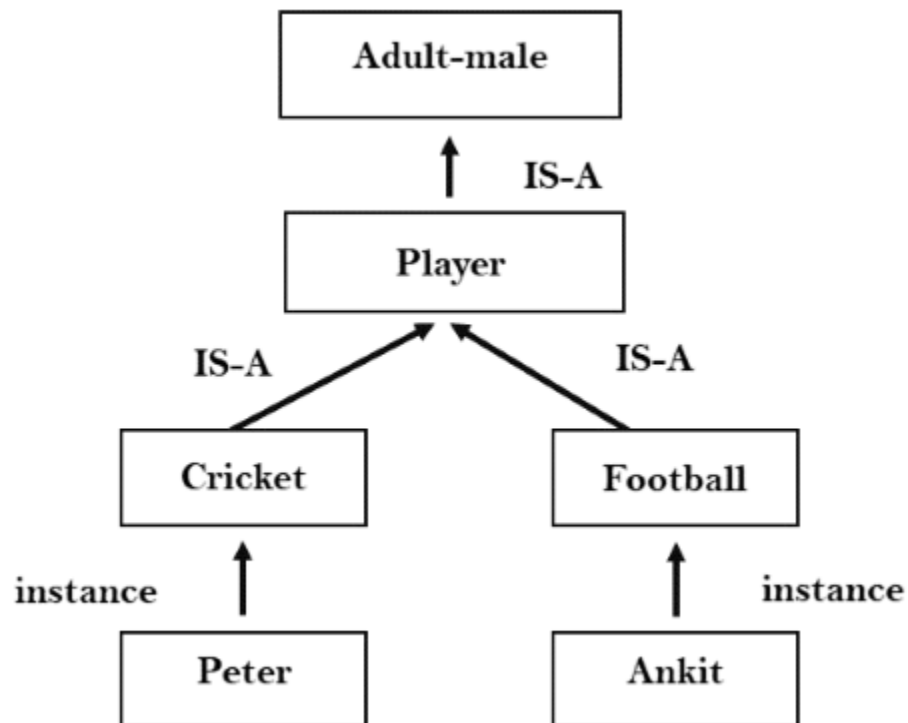
Example: The following is the simple relational knowledge representation.

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

2. Inheritable knowledge:

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- All classes should be arranged in a generalized form or a hierarchical 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)

$\forall x = \text{man}(x) \text{ -----} \rightarrow \text{mortal}(x)$

4. Procedural knowledge:

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- 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.

Requirements for knowledge Representation system:

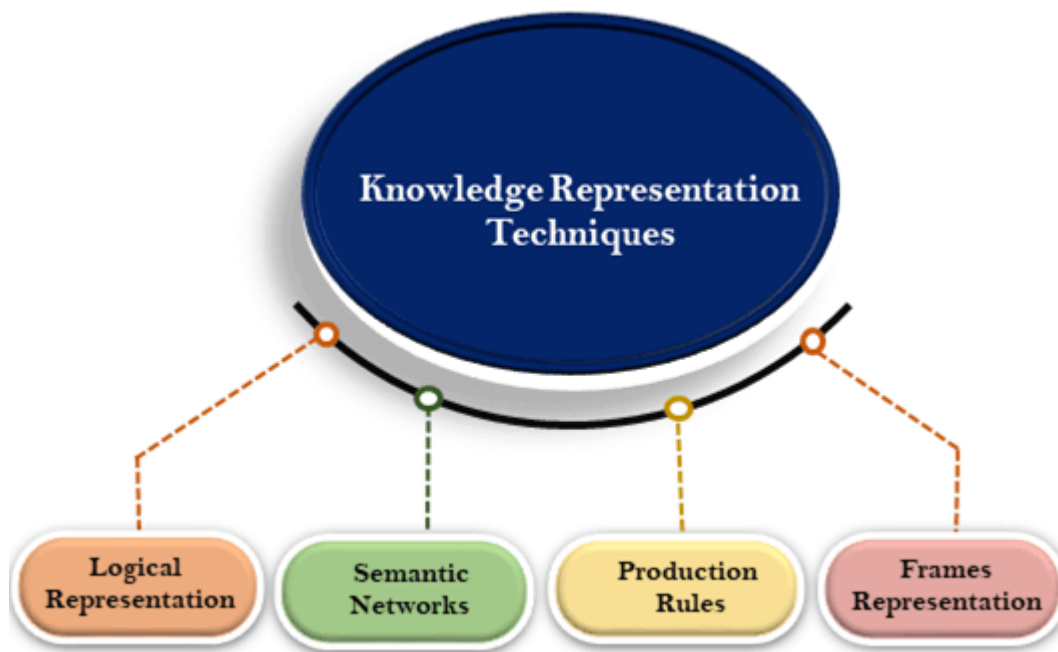
A good knowledge representation system must possess the following properties.

1. **1. Representational Accuracy:**
KR system should have the ability to represent all kind of required knowledge.
2. **2. Inferential Adequacy:**
KR system should have ability to manipulate the representational structures to produce new knowledge corresponding to existing structure.
3. **3. Inferential Efficiency:**
The ability to direct the inferential knowledge mechanism into the most productive directions by storing appropriate guides.
4. **4. Acquisitional efficiency-** The ability to acquire the new knowledge easily using automatic methods.

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.

Note: Do not be confused with logical representation and logical reasoning as logical representation is a representation language and reasoning is a process of thinking logically.

2. Semantic Network Representation

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

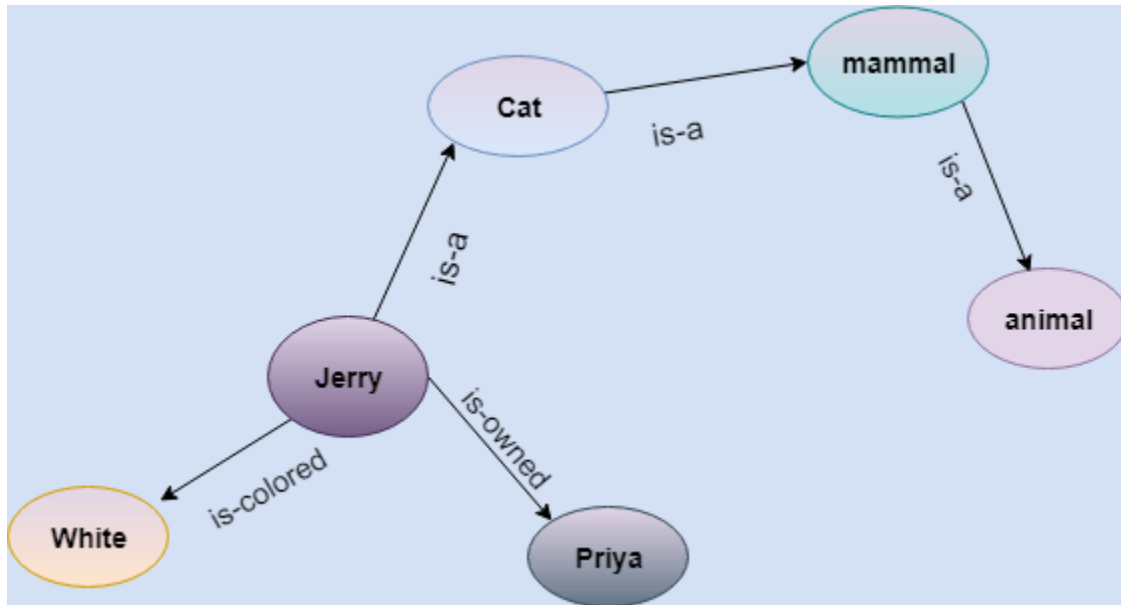
This representation consist of mainly two types of relations:

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

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

Statements:

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



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

Drawbacks in Semantic representation:

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

Advantages of Semantic network:

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

3. Frame Representation

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

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

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

Example: 1

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

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

Example 2:

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

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

Advantages of frame representation:

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

Disadvantages of frame representation:

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

4. Production Rules

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

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

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

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

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

Example:

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

Advantages of Production rule:

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

Disadvantages of Production rule:

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

Propositional logic in Artificial intelligence

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

Example:

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

Following are some basic facts about propositional logic:

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

Syntax of propositional logic:

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

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

Example:

- 1. a) $2+2$ is 4 , it is an atomic proposition as it is a **true** fact.
- 2. b) "The Sun is cold" is also a proposition as it is a **false** fact.
 - o **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 $\neg P$ is called negation of P. A literal can be either Positive literal or negative literal.
- 2. **Conjunction:** A sentence which has \wedge connective such as, $P \wedge Q$ is called a conjunction.

Example: Rohan is intelligent and hardworking. It can be written as,

P= Rohan is intelligent,

Q= Rohan is hardworking. $\rightarrow P \wedge Q$.

- 3. **Disjunction:** A sentence which has \vee connective, such as $P \vee Q$. 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 **$P \vee Q$** .

4. **Implication:** A sentence such as $P \rightarrow Q$, 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 $P \rightarrow Q$

5. **Biconditional:** A sentence such as **$P \Leftrightarrow Q$ 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 $P \Leftrightarrow Q$.

Following is the summarized table for Propositional Logic Connectives:

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

Truth Table:

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

Following are the truth table for all logical connectives:

For Negation:

P	$\neg P$
True	False
False	True

For Conjunction:

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

For disjunction:

P	Q	$P \vee Q$
True	True	True
False	True	True
True	False	True
False	False	False

For Implication:

P	Q	$P \rightarrow Q$
True	True	True
True	False	False
False	True	True
False	False	True

For Biconditional:

P	Q	$P \leftrightarrow Q$
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	$\neg R$	$P \vee Q$	$P \vee Q \rightarrow \neg R$
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 $\neg R \vee Q$, It can be interpreted as $(\neg R) \vee Q$.

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 $A \Leftrightarrow B$. In below truth table we can see that column for $\neg A \vee B$ and $A \rightarrow B$, are identical hence A is Equivalent to B

A	B	$\neg A$	$\neg A \vee B$	$A \rightarrow B$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Properties of Operators:

- **Commutativity:**
 - $P \wedge Q = Q \wedge P$, or
 - $P \vee Q = Q \vee P$.
- **Associativity:**
 - $(P \wedge Q) \wedge R = P \wedge (Q \wedge R)$,
 - $(P \vee Q) \vee R = P \vee (Q \vee R)$
- **Identity element:**
 - $P \wedge \text{True} = P$,
 - $P \vee \text{True} = \text{True}$.
- **Distributive:**
 - $P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$.
 - $P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$.
- **DE Morgan's Law:**
 - $\neg (P \wedge Q) = (\neg P) \vee (\neg Q)$
 - $\neg (P \vee Q) = (\neg P) \wedge (\neg Q)$.
- **Double-negation elimination:**
 - $\neg (\neg P) = P$.

Limitations of Propositional logic:

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

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

First-Order Logic in Artificial intelligence

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

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

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

First-Order logic:

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

- As a natural language, first-order logic also has two main parts:
 - 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	$\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$
Equality	$=$
Quantifier	\forall, \exists

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: \Rightarrow Brothers(Ravi, Ajay).
Chinky is a cat: \Rightarrow cat (Chinky).

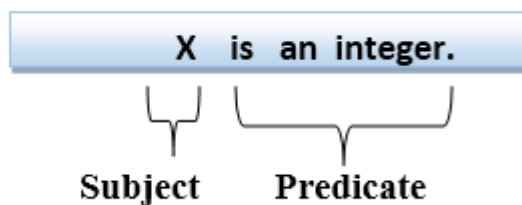
Complex Sentences:

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

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

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

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



Quantifiers in First-order logic:

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

Universal Quantifier:

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

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

Note: In universal quantifier we use implication " \rightarrow ".

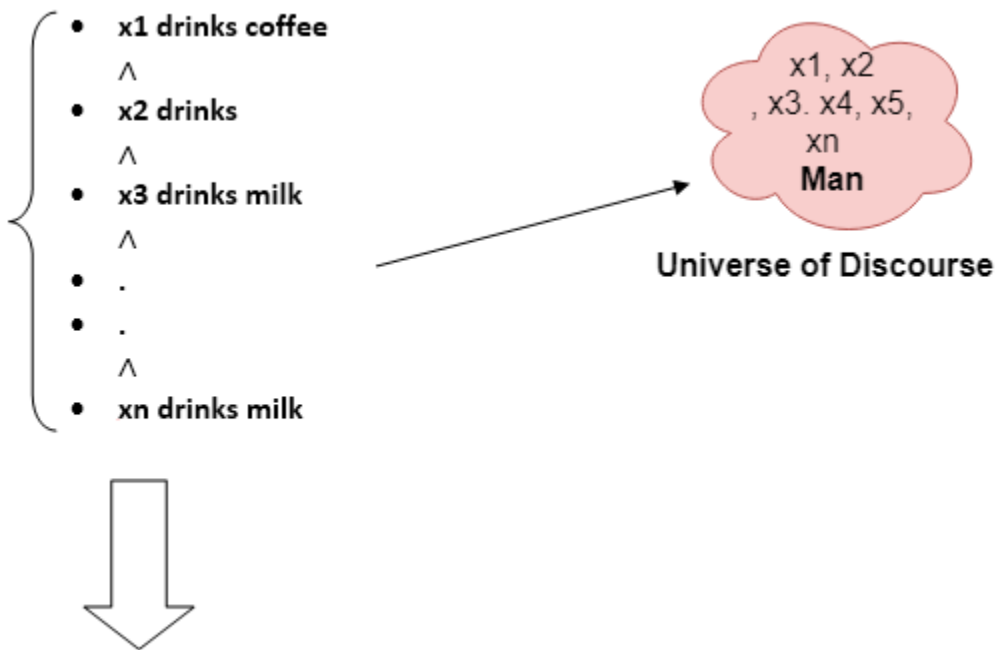
If x is a variable, then $\forall x$ is read as:

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

Example:

All man drink coffee.

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



So in shorthand notation, we can write it as :

$\forall x \text{ man}(x) \rightarrow \text{drink}(x, \text{coffee}).$

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 \exists , which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

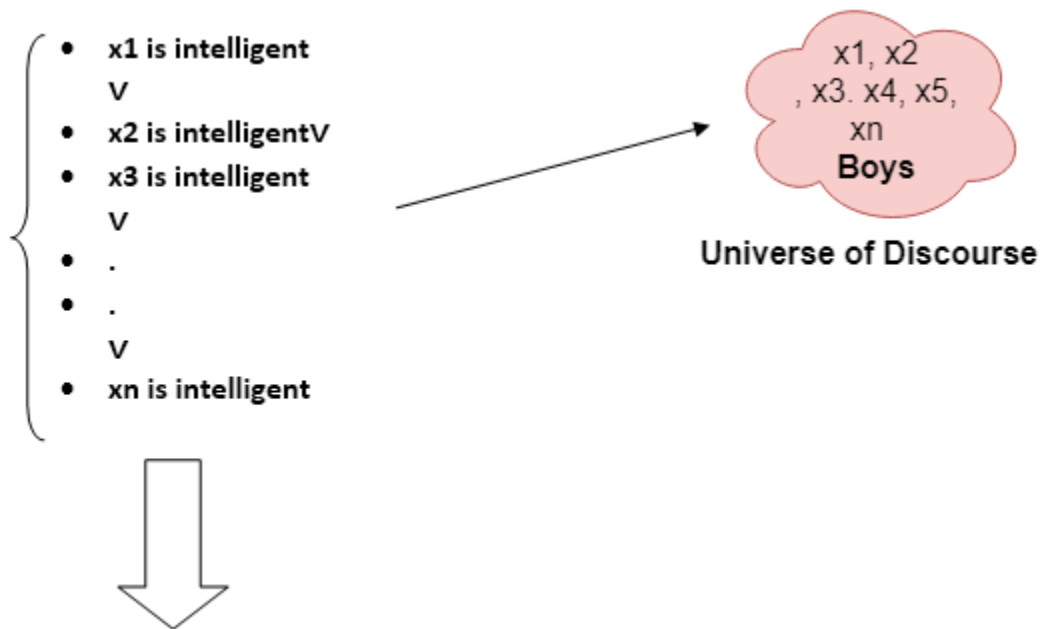
Note: In Existential quantifier we always use AND or Conjunction symbol (\wedge).

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

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

Example:

Some boys are intelligent.



So in short-hand notation, we can write it as:

$\exists x: \text{boys}(x) \wedge \text{intelligent}(x)$

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 \forall is implication \rightarrow .
- The main connective for existential quantifier \exists is and \wedge .

Properties of Quantifiers:

- In universal quantifier, $\forall x \forall y$ is similar to $\forall y \forall x$.
- In Existential quantifier, $\exists x \exists y$ is similar to $\exists y \exists x$.
- $\exists x \forall y$ is not similar to $\forall y \exists x$.

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.

$$\forall x \text{ bird}(x) \rightarrow \text{fly}(x).$$

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 \forall , and it will be represented as follows:

$$\forall x \text{ man}(x) \rightarrow \text{respects}(x, \text{parent}).$$

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 \exists , and it will be represented as:

$$\exists x \text{ boys}(x) \rightarrow \text{play}(x, \text{cricket}).$$

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 \forall with negation, so following representation for this:

$$\neg \forall (x) [\text{student}(x) \rightarrow \text{like}(x, \text{Mathematics}) \wedge \text{like}(x, \text{Science})].$$

5. Only one student failed in Mathematics.

In this question, the predicate is "**failed(x, y),**" where **x= student, and y= subject.**

Since there is only one student who failed in Mathematics, so we will use following representation for this:

$$\exists (x) [\text{student}(x) \rightarrow \text{failed}(x, \text{Mathematics}) \wedge \forall (y) [\neg (x=y) \wedge \text{student}(y) \rightarrow \neg \text{failed}(y, \text{Mathematics})]].$$

Free and Bound Variables:

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

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

Example: $\forall x \exists (y)[P(x, y, z)],$ 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: $\forall x [A(x) B(y)],$ 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: $\neg(x=y)$ which is equivalent to $x \neq y$.

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 $\forall x P(x)$.

- 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 $\forall x P(x)$ "**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 $\forall x P(x)$ **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" $\Rightarrow \forall x P(x)$ so we can infer that "John likes ice-cream" $\Rightarrow P(c)$

Example: 2.

Let's take a famous example,

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

$\forall x \text{ king}(x) \wedge \text{greedy}(x) \rightarrow \text{Evil}(x),$

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

- **$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \rightarrow \text{Evil}(\text{John}),$**
- **$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \rightarrow \text{Evil}(\text{Richard}),$**

- **King(Father(John)) \wedge Greedy (Father(John)) \rightarrow Evil (Father(John)),**

3. Existential Instantiation:

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

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

- It can be represented as:

Example:

From the given sentence: **$\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$,**

So we can infer: **$\text{Crown}(K) \wedge \text{OnHead}(K, \text{John})$** , as long as K does not appear in the knowledge base.

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

4. Existential introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P , then we can infer that there exists something in the universe which has the property P .

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

- It can be represented as:

- **Example: Let's say that,**

"Priyanka got good marks in English."

"Therefore, someone got good marks in English."

Generalized Modus Ponens Rule:

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

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

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

$$\frac{p1', p2', \dots, pn', (p1 \wedge p2 \wedge \dots \wedge pn \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

Example:

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

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

Forward Chaining and backward chaining in AI

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

Inference engine:

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

- a. **Forward chaining**
- b. **Backward chaining**

Horn Clause and Definite clause:

Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm. Logical inference

algorithms use forward and backward chaining approaches, which require KB in the form of the **first-order definite clause**.

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

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

Example: $(\neg p \vee \neg q \vee k)$. It has only one positive literal k .

It is equivalent to $p \wedge q \rightarrow k$.

A. Forward Chaining

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

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

Properties of Forward-Chaining:

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

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

Example:

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

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

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

Facts Conversion into FOL:

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

$$\text{American}(p) \wedge \text{weapon}(q) \wedge \text{sells}(p, q, r) \wedge \text{hostile}(r) \rightarrow \text{Criminal}(p)$$

...(1)
- Country A has some missiles. $\exists p \text{ Owns}(A, p) \wedge \text{Missile}(p)$. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.

$$\text{Owns}(A, T1) \quad \dots\dots(2)$$

$$\text{Missile}(T1) \quad \dots\dots(3)$$
- All of the missiles were sold to country A by Robert.

$$\exists p \text{ Missiles}(p) \wedge \text{Owns}(A, p) \rightarrow \text{Sells}(\text{Robert}, p, A) \quad \dots\dots(4)$$
- Missiles are weapons.

$$\text{Missile}(p) \rightarrow \text{Weapons}(p) \quad \dots\dots(5)$$
- Enemy of America is known as hostile.

$$\text{Enemy}(p, \text{America}) \rightarrow \text{Hostile}(p) \quad \dots\dots(6)$$
- Country A is an enemy of America.

$$\text{Enemy}(A, \text{America}) \quad \dots\dots(7)$$
- Robert is American

$$\text{American}(\text{Robert}). \quad \dots\dots(8)$$

Forward chaining proof:

Step-1:

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

American (Robert)	Missile (T1)	Owns (A,T1)	Enemy (A, America)
-------------------	--------------	-------------	--------------------

Step-2:

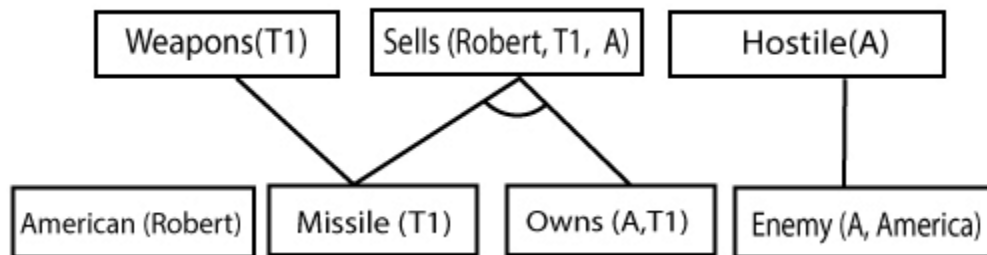
At the second step, we will see those facts which infer from available facts and with satisfied premises.

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

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

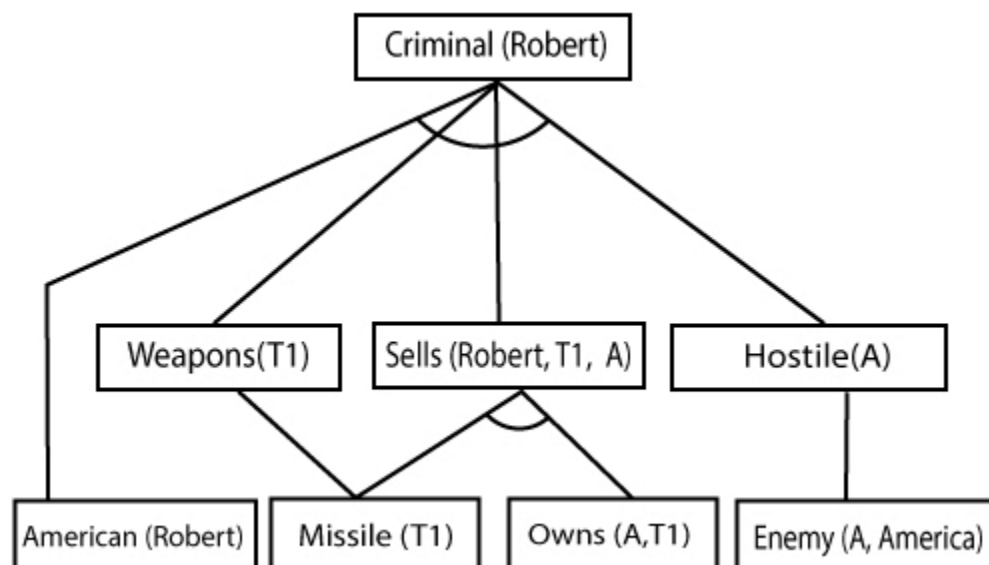
Rule-(4) satisfy with the substitution $\{p/T1\}$, **so Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

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



Step-3:

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



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

B. Backward Chaining:

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of

reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Properties of backward chaining:

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

Example:

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

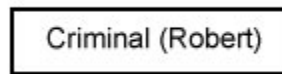
- **American (p) \wedge weapon(q) \wedge sells (p, q, r) \wedge hostile(r) \rightarrow Criminal(p)**
...(1)
- **Owns(A, T1)(2)**
- **Missile(T1)**
- **?p Missiles(p) \wedge Owns (A, p) \rightarrow Sells (Robert, p, A)(4)**
- **Missile(p) \rightarrow Weapons (p)(5)**
- **Enemy(p, America) \rightarrow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

Backward-Chaining proof:

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

Step-1:

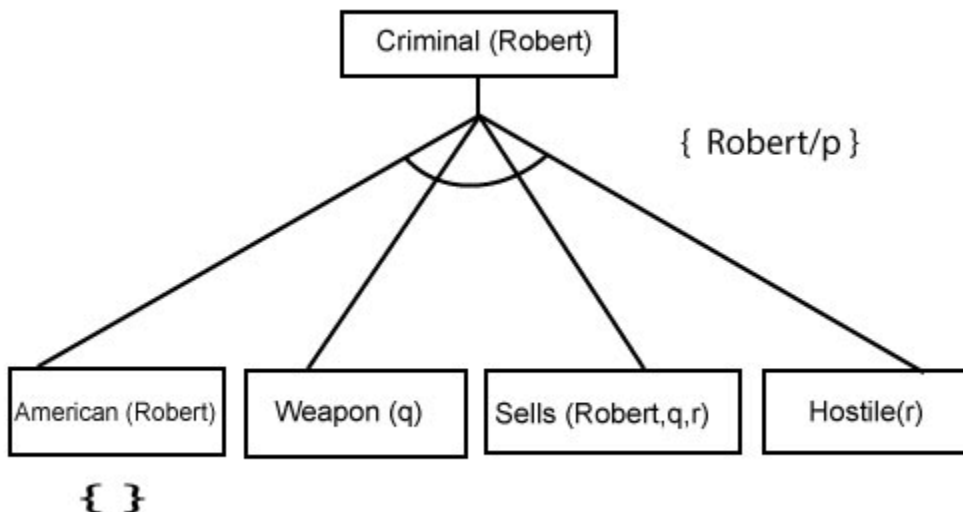
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.



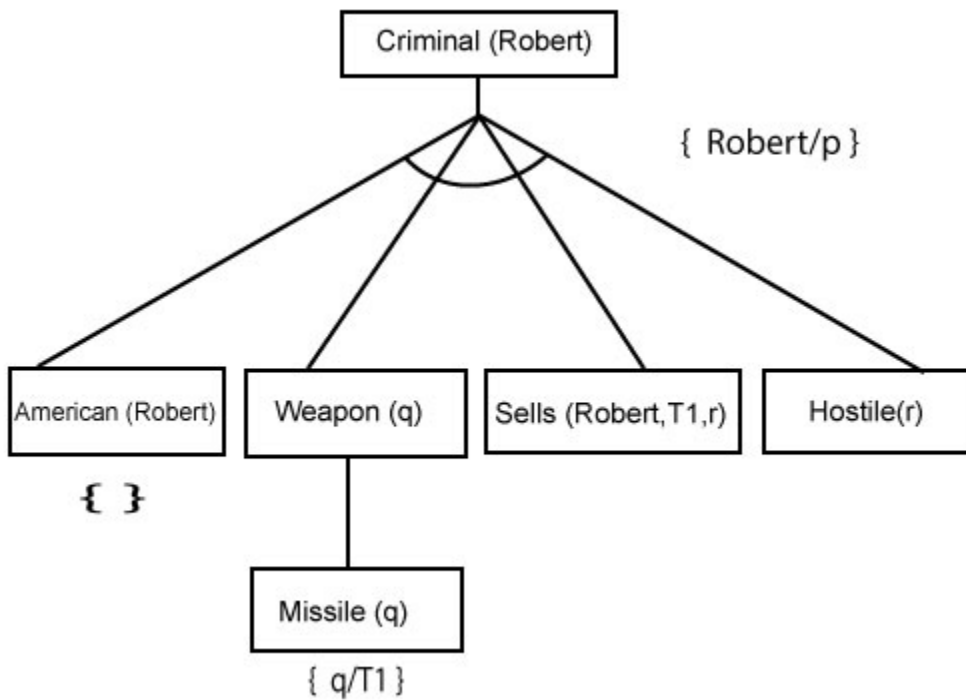
Step-2:

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

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

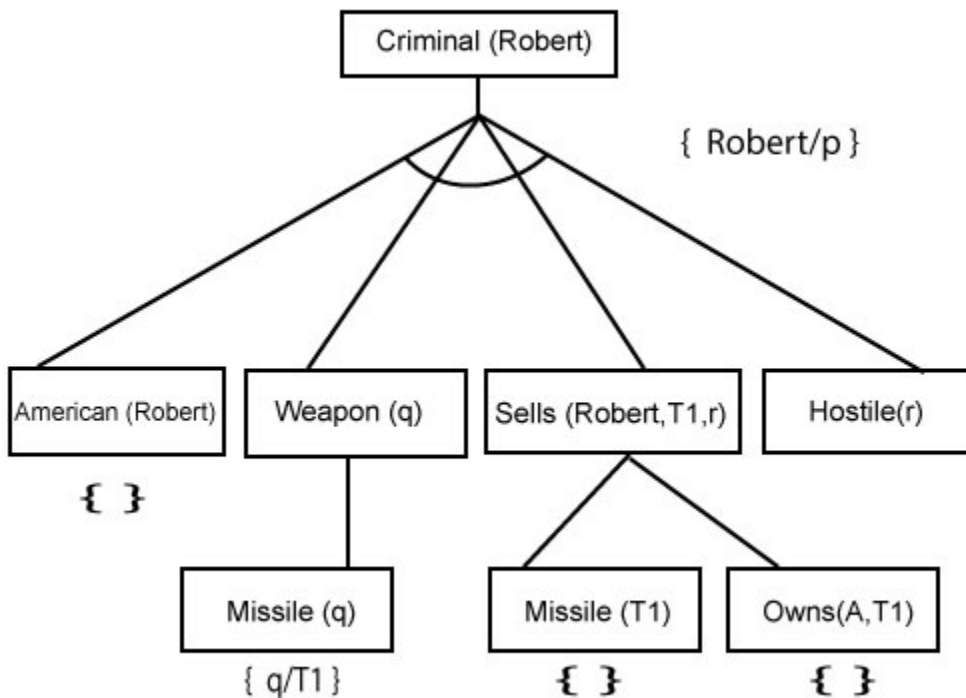


Step-3: At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q .



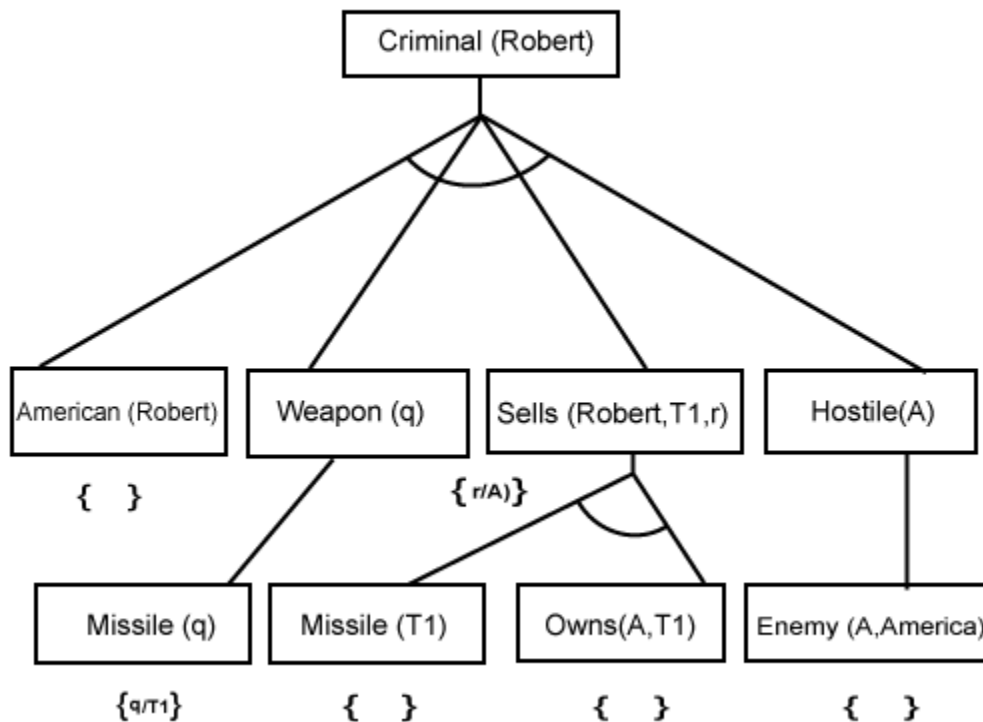
Step-4:

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



Step-5:

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



Difference between backward chaining and forward chaining

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

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

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

S. No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward using inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the	Backward chaining is suitable for diagnostic, prescription

	planning, monitoring, control, and interpretation application.	application.
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible
8.	It operates in the forward direction.	It operates in the backward direction.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.