

## 5. Iterative deepening depth-first Search:

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

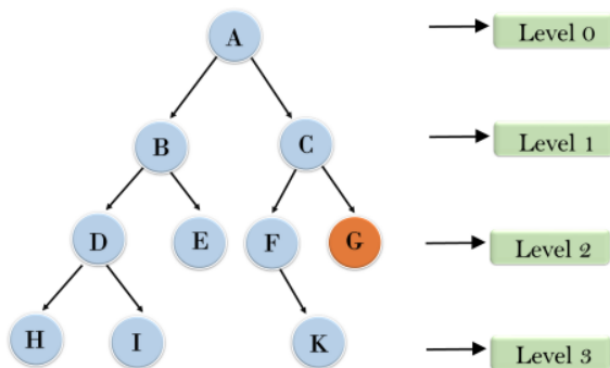
This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

### Advantages:

- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

### Iterative deepening depth first search



1'st Iteration-----> A

2'nd Iteration----> A, B, C

3'rd Iteration----->A, B, D, E, C, F, G

4'th Iteration----->A, B, D, H, I, E, C, F, K, G

In the fourth iteration, the algorithm will find the goal node.

## 6. Bidirectional Search Algorithm:

**Bidirectional search algorithm** runs two simultaneous searches, one from initial state called as forward-search and other from goal node called as backward-search, to find the goal node. Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex. The search stops when these two graphs intersect each other.

**Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.**

### **Advantages:**

- Bidirectional search is fast.
- Bidirectional search requires less memory

### **Disadvantages:**

- Implementation of the bidirectional search tree is difficult.
- **In bidirectional search, one should know the goal state in advance.**

### **Example:**

**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 is 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:**

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

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.

## 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.

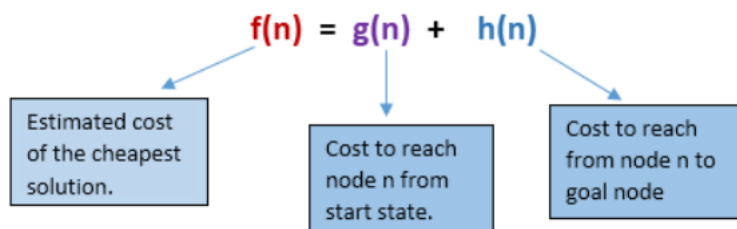
$$f(n) = g(n).$$

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

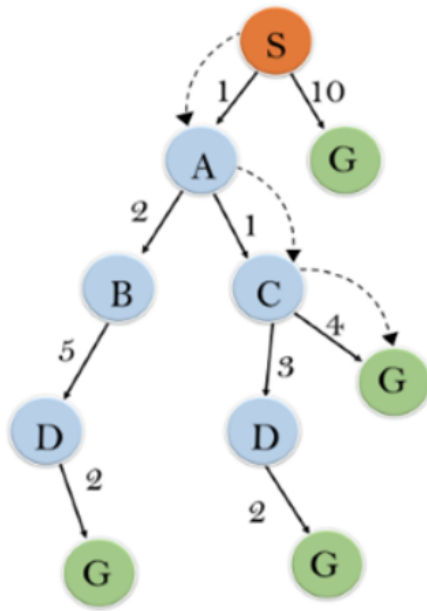
## 2.) A\* Search Algorithm:

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.



**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:**  $\{(S \rightarrow A \rightarrow C \rightarrow G, 6), (S \rightarrow A \rightarrow C \rightarrow D, 11), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$

**Iteration 4** will give the final result, as  $S \rightarrow A \rightarrow C \rightarrow G$  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  $f(n)$

**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.

Constraint Satisfaction Problems (CSPs) are computational problems defined as a set of objects whose state must satisfy a set of constraints. The goal is to find a solution that satisfies all the constraints.

**Components of a Constraint Satisfaction Problem:**

1. **Variables:** Represent the unknowns or entities that need to be assigned values to satisfy the constraints. Each variable has a domain of possible values it can take.
2. **Domains:** Define the set of possible values that variables can take. Domains can be discrete, continuous, finite, or infinite, depending on the problem.
3. **Constraints:** Specify the restrictions or conditions that must be satisfied by the variables' values. Constraints define relationships between variables and can limit the valid combinations of values.

---

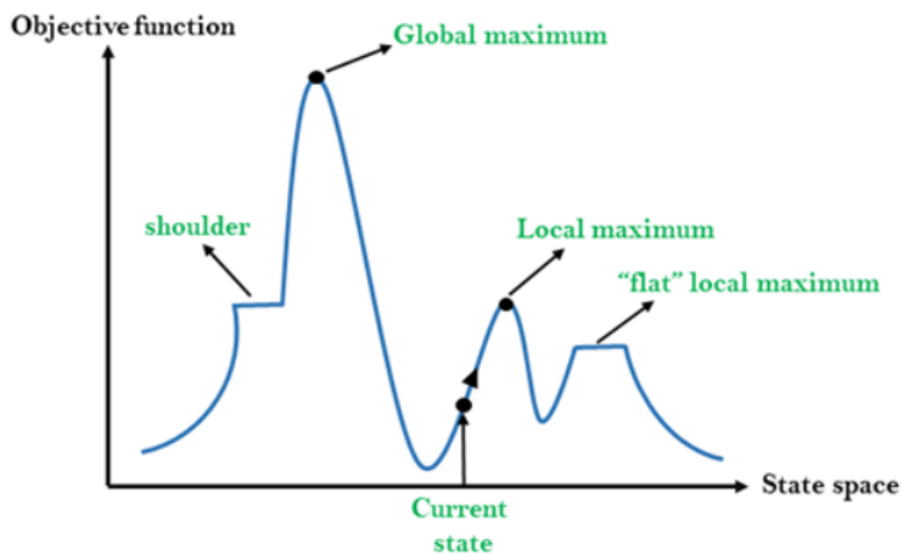
## Hill Climbing Algorithm in Artificial Intelligence

- 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.

## 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.



## Types of Hill Climbing Algorithm:

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

## 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.

---

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.



## Types of 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

- **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.

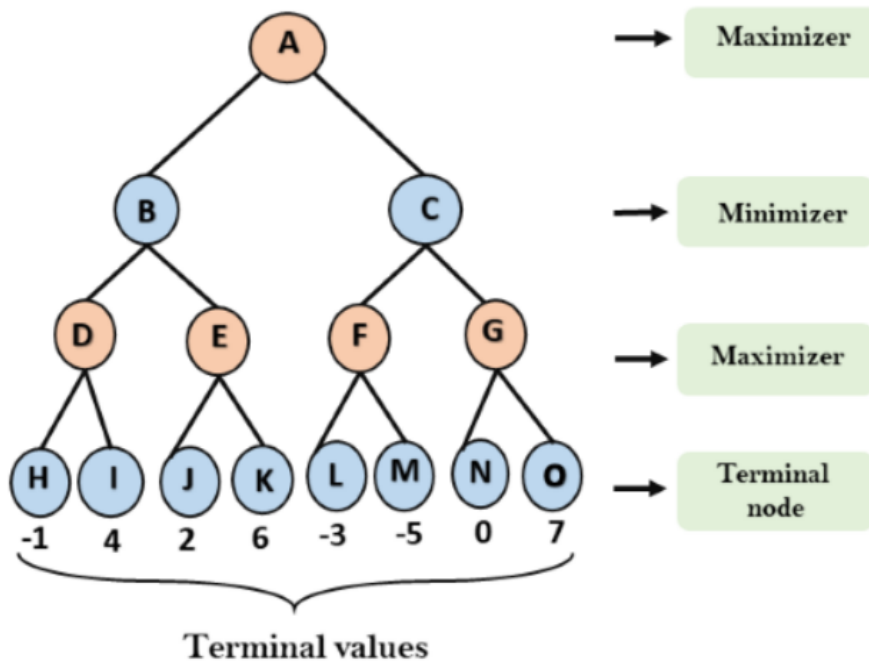
# Mini-Max Algorithm in Artificial Intelligence

- 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.

## 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 =  $-\infty$ , and minimizer will take next turn which has worst-case initial value =  $+\infty$ .



## 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  $+\infty$ .

## Module-3

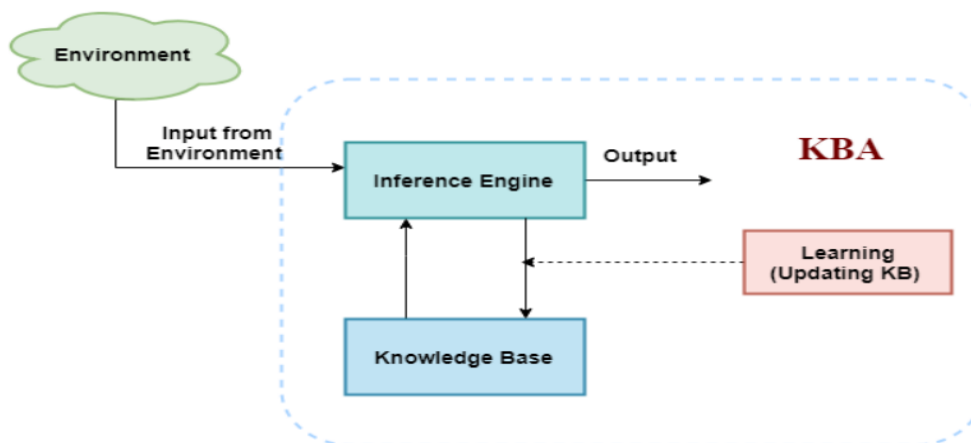
# Knowledge-Based Agent in Artificial intelligence

- An intelligent agent needs **knowledge** about the real world for taking decisions and **reasoning** to act efficiently.
- Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently.**
- Knowledge-based agents are composed of two main parts:
  - **Knowledge-base and**
  - **Inference system.**

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

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

The architecture of knowledge-based agent:



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

## Why use a knowledge base?

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

## Inference system

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

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

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

# The Wumpus World in Artificial intelligence

## Wumpus world:

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

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

# Propositional logic in Artificial intelligence

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

## Example:

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

## Following are some basic facts about propositional logic:

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for representing a proposition, such as 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.





## 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**

## Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

<b>Constant</b>	1, 2, A, John, Mumbai, cat,....
<b>Variables</b>	x, y, z, a, b,....
<b>Predicates</b>	Brother, Father, >,....
<b>Function</b>	sqrt, LeftLegOf, ....
<b>Connectives</b>	$\wedge$ , $\vee$ , $\neg$ , $\Rightarrow$ , $\Leftrightarrow$
<b>Equality</b>	$=$
<b>Quantifier</b>	$\forall$ , $\exists$

## 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:

S. No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
8.	It operates in the forward direction.	It operates in the backward direction.