AI TECHNIQUE –

**What is AI Technique?**

In the real world, the knowledge has some unwelcomed properties −

* Its volume is huge, next to unimaginable.
* It is not well-organized or well-formatted.
* It keeps changing constantly.

AI Technique is a manner to organize and use the knowledge efficiently in such a way that −

* It should be perceivable by the people who provide it.
* It should be easily modifiable to correct errors.
* It should be useful in many situations though it is incomplete or inaccurate.

AI techniques elevate the speed of execution of the complex program it is equipped with.

Applications of AI

AI has been dominant in various fields such as −

* **Gaming** − AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.
* **Natural Language Processing** − It is possible to interact with the computer that understands natural language spoken by humans.
* **Expert Systems** − There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.
* **Vision Systems** − These systems understand, interpret, and comprehend visual input on the computer. For example,
  + A spying aeroplane takes photographs, which are used to figure out spatial information or map of the areas.
  + Doctors use clinical expert system to diagnose the patient.
  + Police use computer software that can recognize the face of criminal with the stored portrait made by forensic artist.
* **Speech Recognition** − Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human’s noise due to cold, etc.
* **Handwriting Recognition** − The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.
* **Intelligent Robots** − Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment.

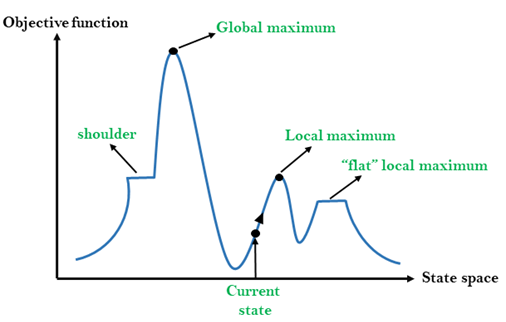
HILL CLIMBING ALGORITHM –

* 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.

**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.



**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:

1. If it is goal state, then return success and quit.
2. Else if it is better than the current state then assign new state as a current state.
3. 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.

1. Let SUCC be a state such that any successor of the current state will be better than it.
2. For each operator that applies to the current state:
3. Apply the new operator and generate a new state.
4. Evaluate the new state.
5. If it is goal state, then return it and quit, else compare it to the SUCC.
6. If it is better than SUCC, then set new state as SUCC.
7. If the SUCC is better than the current state, then set current state to SUCC.

* **Step 3:** Exit.

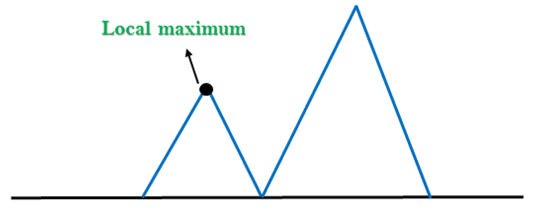
**3. Stochastic hill climbing:**

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

Problems in Hill Climbing Algorithm:

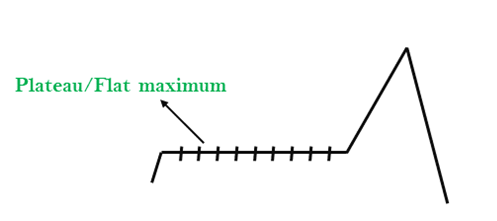
**1. Local Maximum:** A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

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



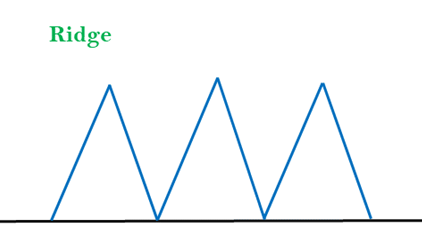
**2. Plateau:** A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

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



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

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



**Simulated Annealing:**

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

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

STATE SPACE SEARCH –

**State Space Search in Artificial Intelligence**

**State space search** is a problem-solving technique used in Artificial Intelligence (AI) to find the solution path from the initial state to the goal state by exploring the various states. The state space search approach searches through all possible states of a problem to find a solution. It is an essential part of Artificial Intelligence and is used in various applications, from game-playing algorithms to natural language processing.

**Introduction**

A **state space** is a way to mathematically represent a problem by defining all the possible states in which the problem can be. This is used in search algorithms to represent the initial state, goal state, and current state of the problem. Each state in the state space is represented using a set of variables.

The **efficiency** of the search algorithm greatly depends on the size of the state space, and it is important to choose an appropriate representation and search strategy to search the state space efficiently.

One of the most well-known **state space search algorithms** is the A algorithm. Other commonly used state space search algorithms include **breadth-first search (BFS)**, **depth-first search (DFS)**, **hill climbing**, **simulated annealing**, and **genetic algorithms**.

**Features of State Space Search**

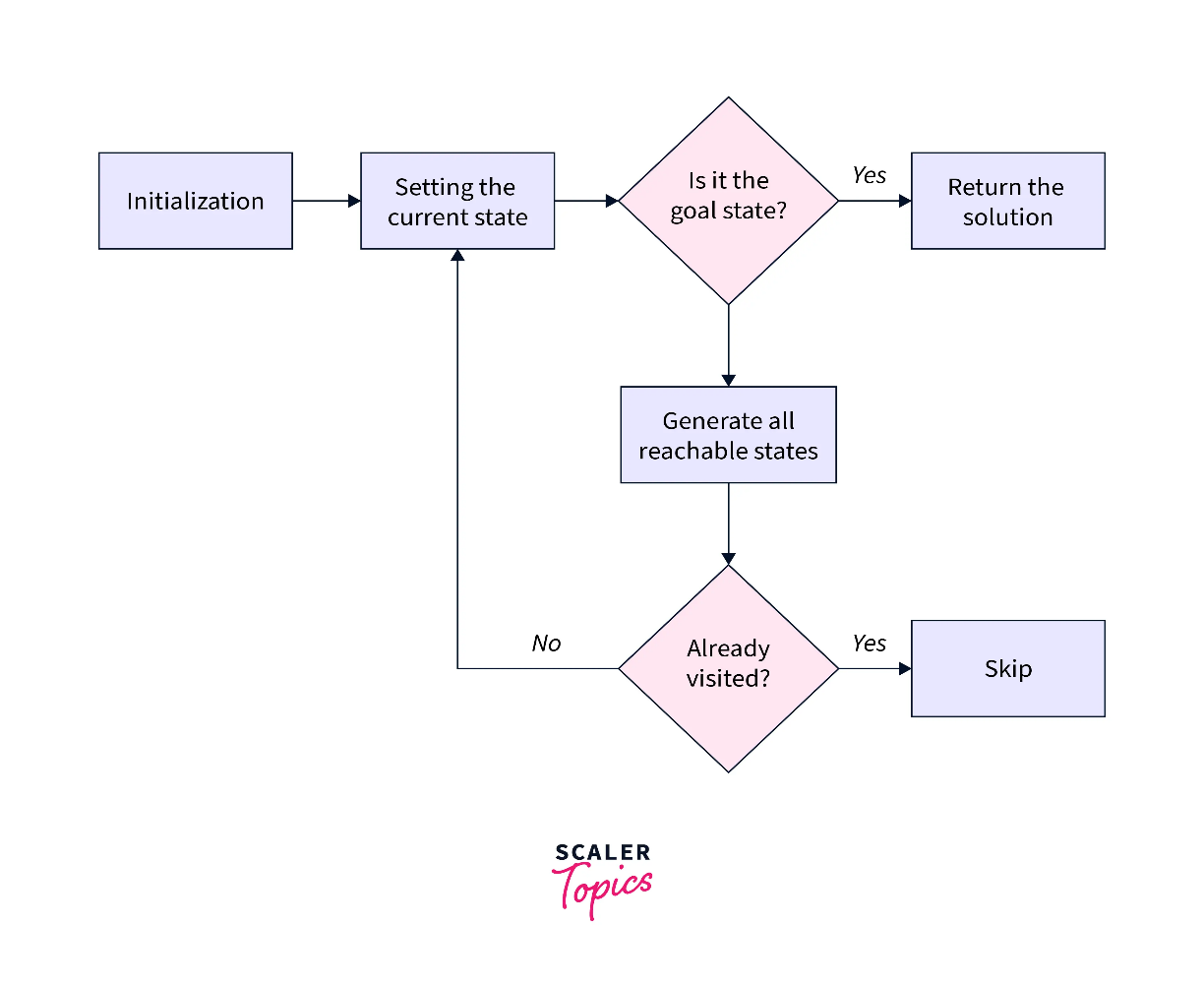
**State space search** has several features that make it an effective problem-solving technique in Artificial Intelligence. These features include:

* **Exhaustiveness:**  
  State space search explores all possible states of a problem to find a solution.
* **Completeness:**  
  If a solution exists, state space search will find it.
* **Optimality:**  
  Searching through a state space results in an optimal solution.
* **Uninformed and Informed Search:**  
  State space search in artificial intelligence can be classified as uninformed if it provides additional information about the problem.

In contrast, informed search uses additional information, such as heuristics, to guide the search process.

**Steps in State Space Search**

The steps involved in state space search are as follows:

* 
* To begin the search process, we set the current state to the initial state.
* We then check if the current state is the goal state. If it is, we terminate the algorithm and return the result.
* If the current state is not the goal state, we generate the set of possible successor states that can be reached from the current state.
* For each successor state, we check if it has already been visited. If it has, we skip it, else we add it to the queue of states to be visited.
* Next, we set the next state in the queue as the current state and check if it's the goal state. If it is, we return the result. If not, we repeat the previous step until we find the goal state or explore all the states.
* If all possible states have been explored and the goal state still needs to be found, we return with no solution.

**State Space Representation**

State space Representation involves defining an INITIAL STATE and a GOAL STATE and then determining a sequence of actions, called states, to follow.

* State:  
  A state can be an Initial State, a Goal State, or any other possible state that can be generated by applying rules between them.
* Space:  
  In an AI problem, space refers to the exhaustive collection of all conceivable states.
* Search:  
  This technique moves from the beginning state to the desired state by applying good rules while traversing the space of all possible states.
* Search Tree:  
  To visualize the search issue, a search tree is used, which is a tree-like structure that represents the problem. The initial state is represented by the root node of the search tree, which is the starting point of the tree.
* Transition Model:  
  This describes what each action does, while Path Cost assigns a cost value to each path, an activity sequence that connects the beginning node to the end node. The optimal option has the lowest cost among all alternatives.

**Example of State Space Search**

The 8-puzzle problem is a commonly used example of a state space search. It is a sliding puzzle game consisting of 8 numbered tiles arranged in a 3x3 grid and one blank space. The game aims to rearrange the tiles from their initial state to a final goal state by sliding them into the blank space.

To represent the state space in this problem, we use the nine tiles in the puzzle and their respective positions in the grid. Each state in the state space is represented by a 3x3 array with values ranging from 1 to 8, and the blank space is represented as an empty tile.

The initial state of the puzzle represents the starting configuration of the tiles, while the goal state represents the desired configuration. Search algorithms utilize the state space to find a sequence of moves that will transform the initial state into the goal state.

This algorithm guarantees a solution but can become very slow for larger state spaces. Alternatively, other algorithms, such as **A search**, use heuristics to guide the search more efficiently.

Our objective is to move from the current state to the target state by sliding the numbered tiles through the blank space. Let's look closer at reaching the target state from the current state.

To summarize, our approach involved exhaustively exploring all reachable states from the current state and checking if any of these states matched the target state.

**Applications of State Space Search**

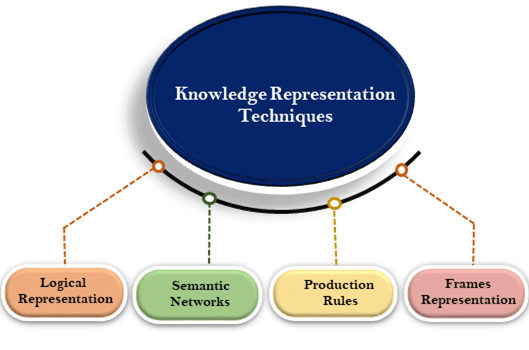
* State space search algorithms are used in various fields, such as robotics, game playing, computer networks, operations research, bioinformatics, cryptography, and supply chain management. In artificial intelligence, state space search algorithms can solve problems like **pathfinding**, **planning**, and **scheduling**.
* They are also useful in planning robot motion and finding the best sequence of actions to achieve a goal. In games, state space search algorithms can help determine the best move for a player given a particular game state.
* **State space search algorithms** can optimize routing and resource allocation in computer networks and operations research.
* In **Bioinformatics**, state space search algorithms can help find patterns in biological data and predict protein structures.
* In **Cryptography**, state space search algorithms are used to break codes and find cryptographic keys.

UNIT – 2

**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:

1. Propositional Logics
2. Predicate logics

Note: We will discuss Prepositional Logics and Predicate logics in later chapters.

Advantages of logical representation:

1. Logical representation enables us to do logical reasoning.
2. Logical representation is the basis for the programming languages.

Disadvantages of logical Representation:

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

**2. Semantic Network Representation**

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

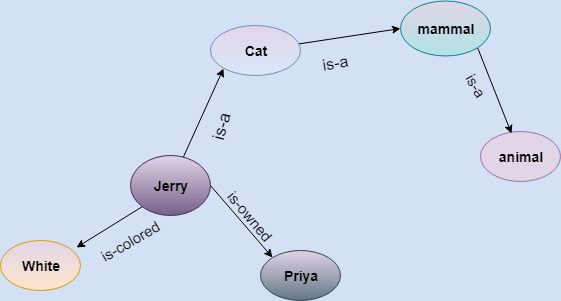
This representation consist of mainly two types of relations:

1. IS-A relation (Inheritance)
2. Kind-of-relation

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

Statements:

1. Jerry is a cat.
2. Jerry is a mammal
3. Jerry is owned by Priya.
4. Jerry is brown colored.
5. 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.

What is Forward Reasoning?

**Forward reasoning** is a process in artificial intelligence that finds all the possible solutions of a problem based on the initial data and facts. Thus, the forward reasoning is a data-driven task as it begins with new data. The main objective of the forward reasoning in AI is to find a conclusion that would follow. It uses an opportunistic type of approach.

Forward reasoning flows from incipient to the consequence. The inference engine searches the knowledge base with the given information depending on the constraints. The precedence of these constraints have to match the current state.

In forward reasoning, the first step is that the system is given one or more constraints. The rules are then searched for in the knowledge base for every constraint. The rule that fulfils the condition is selected. Also, every rule can generate a new condition from the conclusion which is obtained from the invoked one. This new conditions can be added and are processed again.

The step ends if no new conditions exist. Hence, we can conclude that forward reasoning follows the top-down approach.

What is Backward Reasoning?

**Backward reasoning** is the reverse process of the forward reasoning in which a goal or hypothesis is selected and it is analyzed to find the initial data, facts, and rules. Therefore, the backward reasoning is a goal driven task as it begins with conclusions or goals that are uncertain. The main objective of the backward reasoning is to find the facts that support the conclusions.

Backward reasoning uses a conservative type of approach and flows from consequence to the incipient. The system helps to choose a goal state and reasons in a backward direction. The first step in the backward reasoning is that the goal state and rules are selected. Then, sub-goals are made from the selected rule, which need to be satisfied for the goal state to be true.

The initial conditions are set such that they satisfy all the sub-goals. Also, the established states are matched to the initial state provided. If the condition is fulfilled, the goal is the solution, otherwise the goal is rejected. Therefore, backward reasoning follows bottom-up technique.

Backward reasoning is also known as a **decision-driven** or **goal-driven**inference technique because the system selects a goal state and reasons in the backward direction.

Difference between Forward and Backward Reasoning in AI

The following are the important differences between Forward and Backward Reasoning in AI −

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Forward Reasoning** | **Backward Reasoning** |
| 1. | It is a data-driven task. | It is a goal driven task. |
| 2. | It begins with new data. | It begins with conclusions that are uncertain. |
| 3. | The objective is to find a conclusion that would follow. | The objective is to find the facts that support the conclusions. |
| 4. | It uses an opportunistic type of approach. | It uses a conservative type of approach. |
| 5. | It flows from incipient to the consequence. | It flows from consequence to the incipient. |
| 6. | Forward reasoning begins with the initial facts. | Backward reasoning begins with some goal (hypothesis). |
| 7. | Forward reasoning tests all the rules. | Backward reasons tests some rules. |
| 8. | Forward reasoning is a bottom-up approach. | Backward reasoning is a top-down approach. |
| 9. | Forward reasoning can produce an infinite number of conclusion. | Backward reasoning produces a finite number of conclusions. |
| 10. | In the forward reasoning, all the data is available. | In the backward reasoning, the data is acquired on demand. |
| 11. | Forward reasoning has a small number of initial states but a large number of conclusions. | Backward reasoning has a smaller number of goals and a larger number of rules. |
| 12. | In forward reasoning, the goal formation is difficult. | In backward reasoning, it is easy to form a goal. |
| 13. | Forward reasoning works in forward direction to find all the possible conclusions from facts. | Backward reasoning work in backward direction to find the facts that justify the goal. |
| 14. | Forward reason is suitable to answer the problems such as planning, control, monitoring, etc. | Backward reasoning is suitable for diagnosis like problems. |

RESOLUTION –

**Resolution :**

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

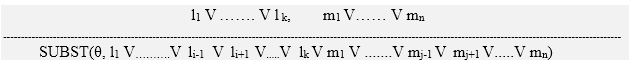
Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.

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

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

The resolution inference rule:

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



Where **li** and **mj** are complementary literals.

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

Example:

We can resolve two clauses which are given below:

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

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

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

**[Animal (g(x) V ￢ Kills(f(x), x)].**

Steps for Resolution:

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

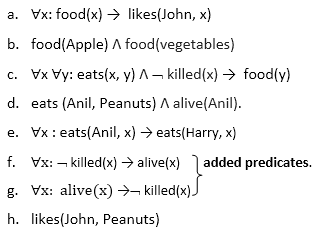
To better understand all the above steps, we will take an example in which we will apply resolution.

Example:

1. **John likes all kind of food.**
2. **Apple and vegetable are food**
3. **Anything anyone eats and not killed is food.**
4. **Anil eats peanuts and still alive**
5. **Harry eats everything that Anil eats.**  
   **Prove by resolution that:**
6. **John likes peanuts.**

**Step-1: Conversion of Facts into FOL**

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



**Step-2: Conversion of FOL into CNF**

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

* **Eliminate all implication (→) and rewrite**
  1. ∀x ¬ food(x) V likes(John, x)
  2. food(Apple) Λ food(vegetables)
  3. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
  4. eats (Anil, Peanuts) Λ alive(Anil)
  5. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  6. ∀x¬ [¬ killed(x) ] V alive(x)
  7. ∀x ¬ alive(x) V ¬ killed(x)
  8. likes(John, Peanuts).
* **Move negation (¬)inwards and rewrite**
  1. ∀x ¬ food(x) V likes(John, x)
  2. food(Apple) Λ food(vegetables)
  3. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
  4. eats (Anil, Peanuts) Λ alive(Anil)
  5. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  6. ∀x ¬killed(x) ] V alive(x)
  7. ∀x ¬ alive(x) V ¬ killed(x)
  8. likes(John, Peanuts).
* **Rename variables or standardize variables**
  1. ∀x ¬ food(x) V likes(John, x)
  2. food(Apple) Λ food(vegetables)
  3. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)
  4. eats (Anil, Peanuts) Λ alive(Anil)
  5. ∀w¬ eats(Anil, w) V eats(Harry, w)
  6. ∀g ¬killed(g) ] V alive(g)
  7. ∀k ¬ alive(k) V ¬ killed(k)
  8. likes(John, Peanuts).
* **Eliminate existential instantiation quantifier by elimination.**  
  In this step, we will eliminate existential quantifier ∃, and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.

Predicate Logic deals with predicates, which are propositions, consist of variables.

**Predicate Logic - Definition**

A predicate is an expression of one or more variables determined on some specific domain. A predicate with variables can be made a proposition by either authorizing a value to the variable or by quantifying the variable.

The following are some examples of predicates.

* Consider E(x, y) denote "x = y"
* Consider X(a, b, c) denote "a + b + c = 0"
* Consider M(x, y) denote "x is married to y."

**Quantifier:**

The variable of predicates is quantified by quantifiers. There are two types of quantifier in predicate logic - Existential Quantifier and Universal Quantifier.

Existential Quantifier:

If p(x) is a proposition over the universe U. Then it is denoted as ∃x p(x) and read as "There exists at least one value in the universe of variable x such that p(x) is true. The quantifier ∃ is called the existential quantifier.

There are several ways to write a proposition, with an existential quantifier, i.e.,

(∃x∈A)p(x)    or    ∃x∈A    such that p (x)    or    (∃x)p(x)    or    p(x) is true for some x ∈A.

Universal Quantifier:

If p(x) is a proposition over the universe U. Then it is denoted as ∀x,p(x) and read as "For every x∈U,p(x) is true." The quantifier ∀ is called the Universal Quantifier.

There are several ways to write a proposition, with a universal quantifier.

∀x∈A,p(x)    or    p(x), ∀x ∈A      Or    ∀x,p(x)    or    p(x) is true for all x ∈A.

Negation of Quantified Propositions:

When we negate a quantified proposition, i.e., when a universally quantified proposition is negated, we obtain an existentially quantified proposition,and when an existentially quantified proposition is negated, we obtain a universally quantified proposition.

The two rules for negation of quantified proposition are as follows. These are also called DeMorgan's Law.

**Example: Negate each of the following propositions:**

1.∀x p(x)∧ ∃ y q(y)

**Sol:** ~.∀x p(x)∧ ∃ y q(y))  
      ≅~∀ x p(x)∨∼∃yq (y)        (∴∼(p∧q)=∼p∨∼q)  
      ≅ ∃ x ~p(x)∨∀y∼q(y)

2. (∃x∈U) (x+6=25)

**Sol:** ~( ∃ x∈U) (x+6=25)  
      ≅∀ x∈U~ (x+6)=25  
      ≅(∀ x∈U) (x+6)≠25

3. ~( ∃ x p(x)∨∀ y q(y)

**Sol:** ~( ∃ x p(x)∨∀ y q(y))  
      ≅~∃ x p(x)∧~∀ y q(y)        (∴~(p∨q)= ∼p∧∼q)  
      ≅ ∀ x ∼ p(x)∧∃y~q(y))

Propositions with Multiple Quantifiers:

The proposition having more than one variable can be quantified with multiple quantifiers. The multiple universal quantifiers can be arranged in any order without altering the meaning of the resulting proposition. Also, the multiple existential quantifiers can be arranged in any order without altering the meaning of the proposition.

The proposition which contains both universal and existential quantifiers, the order of those quantifiers can't be exchanged without altering the meaning of the proposition, e.g., the proposition ∃x ∀ y p(x,y) means "There exists some x such that p (x, y) is true for every y."

**Example:** Write the negation for each of the following. Determine whether the resulting statement is true or false. Assume U = R.

1.∀ x ∃ m(x2<m)

**Sol:** Negation of ∀ x ∃ m(x2<m) is ∃ x ∀ m (x2≥m). The meaning of ∃ x ∀ m (x2≥m) is that there exists for some x such that x2≥m, for every m. The statement is true as there is some greater x such that x2≥m, for every m.

2. ∃ m∀ x(x2<m)

**Sol:** Negation of ∃ m ∀ x (x2<m) is ∀ m∃x (x2≥m). The meaning of ∀ m∃x (x2≥m) is that for every m, there exists for some x such that x2≥m. The statement is true as for every m, there exists for some greater x such that x2≥m.

Heuristic Search

Heuristic search is a problem-solving strategy that uses a heuristic function to guide the search for solutions in a more efficient manner than brute-force methods. It makes educated guesses or estimates about which paths or actions are more likely to lead to a solution, thus reducing the search space and computational effort.

Heuristic Function

A heuristic function (or heuristic evaluation function) is a function that estimates the cost or value of reaching a goal from a given state in a problem. It provides a way to evaluate how "good" or "promising" a particular state is in terms of reaching the goal. The heuristic function uses domain-specific knowledge to guide the search by providing a measure of how close a state is to the goal.

Admissible Heuristic: A heuristic is said to be admissible if it never overestimates the true cost to reach the goal from the current state.

Consistent (or Monotonic) Heuristic: A heuristic is consistent if the estimated cost from the current state to a successor plus the estimated cost from that successor to the goal is less than or equal to the estimated cost from the current state to the goal.

Heuristic Search Techniques

Heuristic search is like solving a puzzle by making smart guesses instead of checking every possibility. These techniques use shortcuts or rules of thumb to solve problems faster.

Common Techniques:

Greedy Best-First Search

What? Picks the path that looks closest to the goal.

Good For: Quick solutions.

Bad For: Might miss the best solution.

A Search\*

What? Uses both the current path cost and an estimate to the goal to pick the best path.

Good For: Finding the best solution if the estimate is good.

Bad For: Uses more memory than Greedy Search.

Iterative Deepening A\*

What? Uses A\* but checks paths in layers.

Good For: Best solutions with less memory.

Bad For: Slower than A\*.

Bidirectional Search

What? Searches from both the start and end until they meet.

Good For: Faster searches.

Bad For: Needs more memory and not always works.

Hill Climbing

What? Keeps choosing the best next step.

Good For: Simple problems.

Bad For: Might get stuck without the best solution.

Simulated Annealing

What? Sometimes takes worse steps to find better paths.

Good For: Finding better paths.

Bad For: Needs fine-tuning and can be slow.

Genetic Algorithms

What? Uses ideas from evolution to find good solutions.

Good For: Big problems.

Bad For: Needs special setup and can be complex.

Constraint Satisfaction Problems (CSP)

What? Solves problems step by step based on rules.

Good For: Many types of problems.

Bad For: Might need to redo steps.

In Simple Words:

Heuristic Search is like finding the best way through a maze by making smart guesses.

Advantages: These methods are faster than checking every option.

Disadvantages: They might not always find the very best solution and can need some tweaking to work best.