

Artificial Intelligence (AI)

What Is Artificial Intelligence (AI)?

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving.

KEY TAKEAWAYS

- Artificial intelligence refers to the simulation of human intelligence in machines.
- The goals of artificial intelligence include learning, reasoning, and perception.
- AI is being used across different industries including finance and healthcare.
- Weak AI tends to be simple and single-task oriented, while strong AI carries on tasks that are more complex and human-like.

Categorization of Artificial Intelligence

Artificial intelligence can be divided into two different categories: weak and strong. [Weak artificial intelligence](#) embodies a system designed to carry out one particular job. Weak AI systems include video games such as the chess example from above and personal assistants such as Amazon's Alexa and Apple's Siri. You ask the assistant a question, it answers it for you.

[Strong artificial intelligence](#) systems are systems that carry on the tasks considered to be human-like. These tend to be more complex and complicated systems. They are programmed to handle situations in which they may be required to problem solve without having a person intervene. These kinds of systems can be found in applications like self-driving cars or in hospital operating rooms.

Search Algorithms in AI

Artificial Intelligence is the study of building agents that act rationally. Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.

- A search problem consists of:
 - **Problem Space** – It is the environment in which the search takes place. (A set of states and set of operators to change those states)
 - **Problem Instance** – It is Initial state + Goal state.
 - **Problem Space Graph** – It represents problem state. States are shown by nodes and operators are shown by edges.
 - **Depth of a problem** – Length of a shortest path or shortest sequence of operators from Initial State to goal state.
 - **Space Complexity** – The maximum number of nodes that are stored in memory.
 - **Time Complexity** – The maximum number of nodes that are created.

- **Admissibility** – A property of an algorithm to always find an optimal solution.
- **Branching Factor** – The average number of child nodes in the problem space graph.
- **Depth** – Length of the shortest path from initial state to goal state.

Properties of Search Algorithms:

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

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

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 four main types:

- Breadth-first search

- Depth-first search
- Iterative deepening depth-first search
- Bidirectional Search

Breadth-first Search:

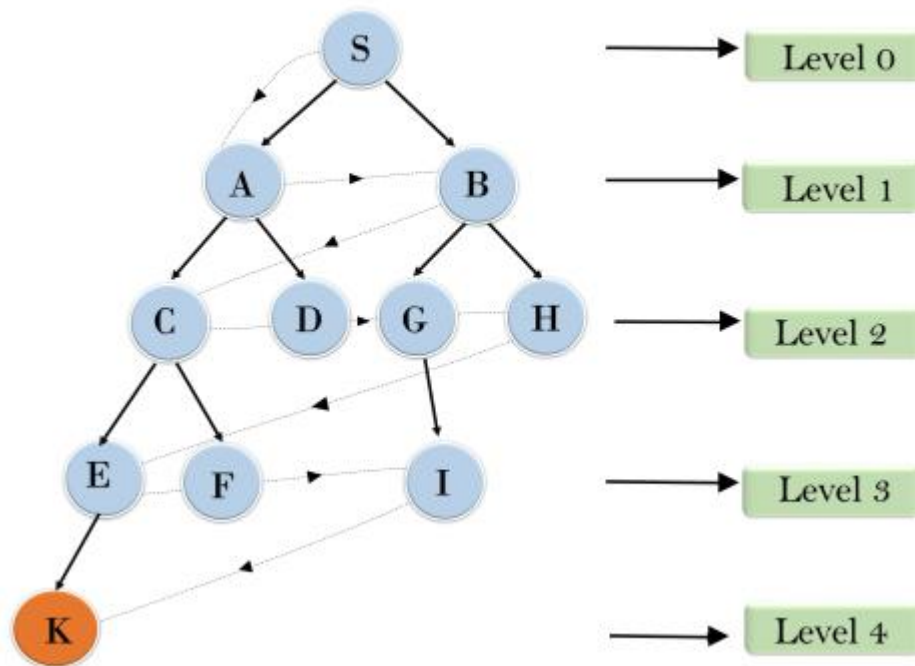
- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before 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.

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



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.

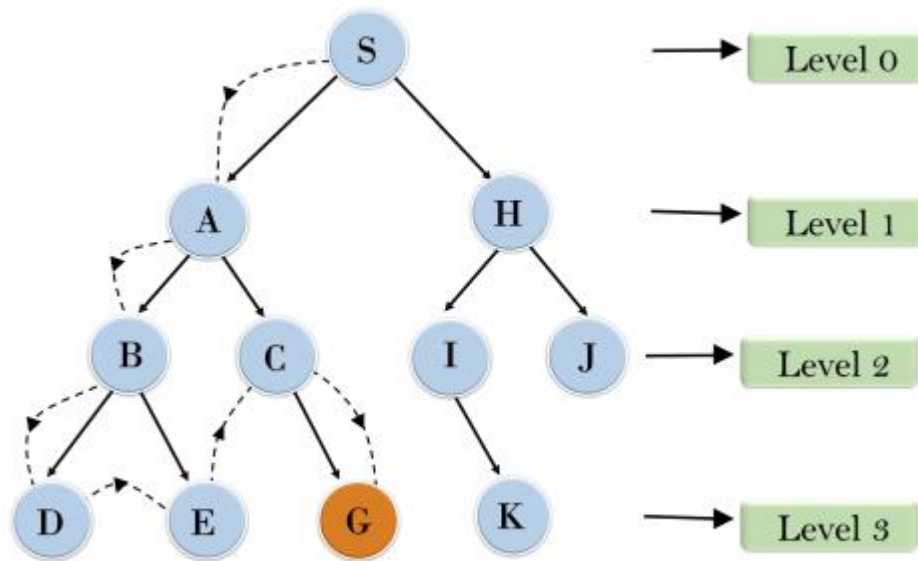
Example:

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.

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. **Best First Search Algorithm(Greedy search)**
2. A* Search

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.

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:

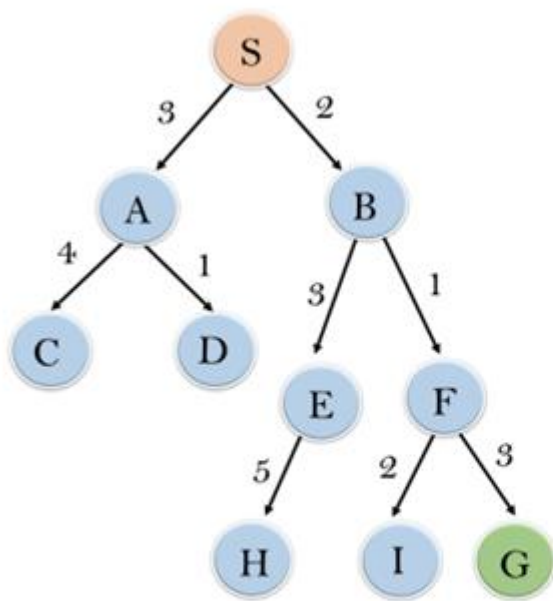
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

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.

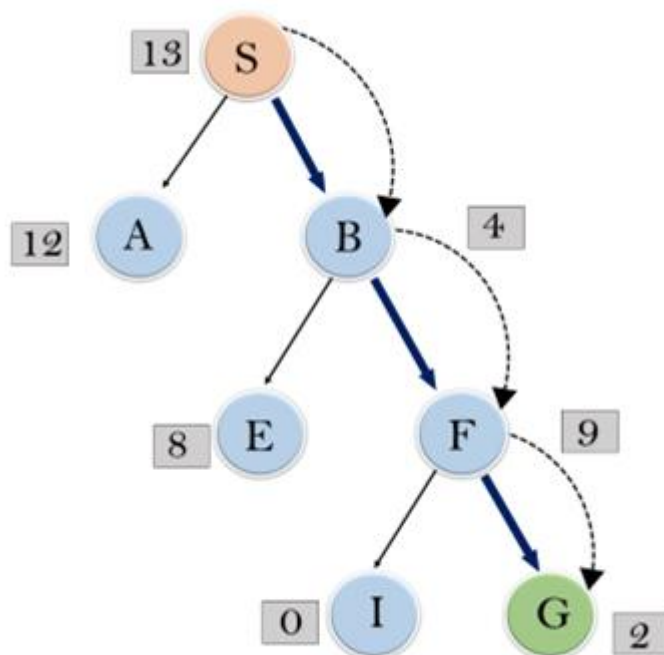
Example:

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.

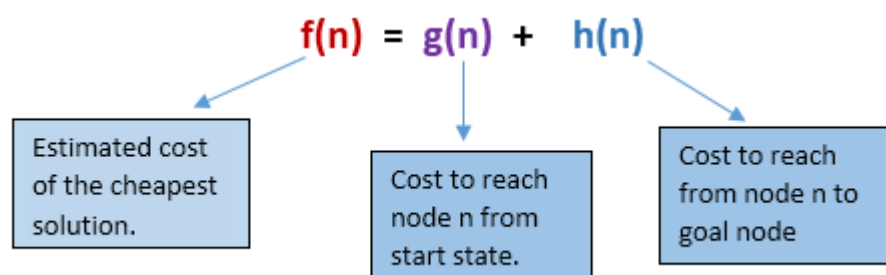
Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

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.

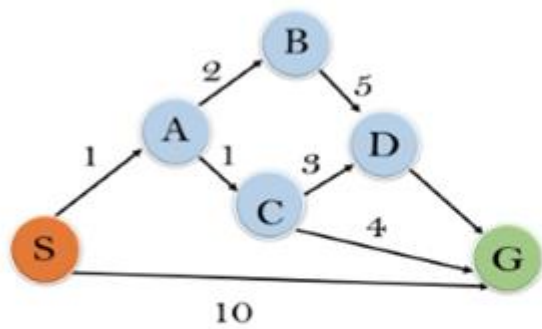
Disadvantages:

- 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:

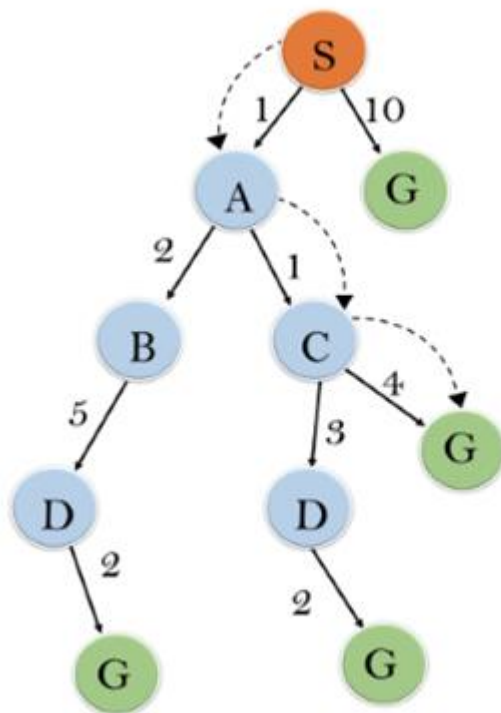
In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the $f(n)$ of each state using the formula $f(n) = g(n) + h(n)$, where $g(n)$ is the cost to reach any node from start state.

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

Iteration 4 will give the final result, as **S→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.

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).

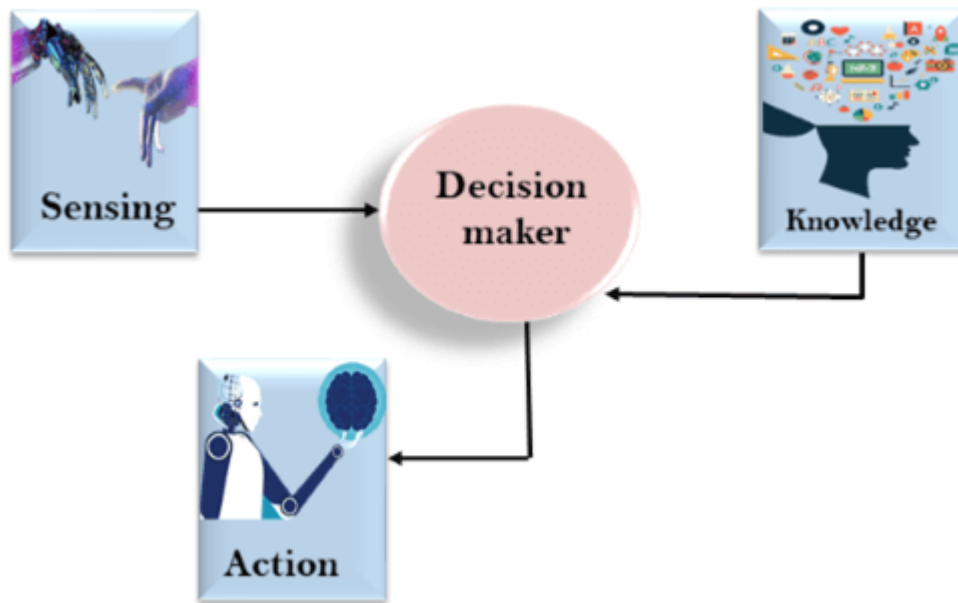
Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

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 behaviour in AI agents. An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

Let's suppose if you met some person who is speaking in a language which you don't know, then how you will able to act on that. The same thing applies to the intelligent behaviour of the agents.

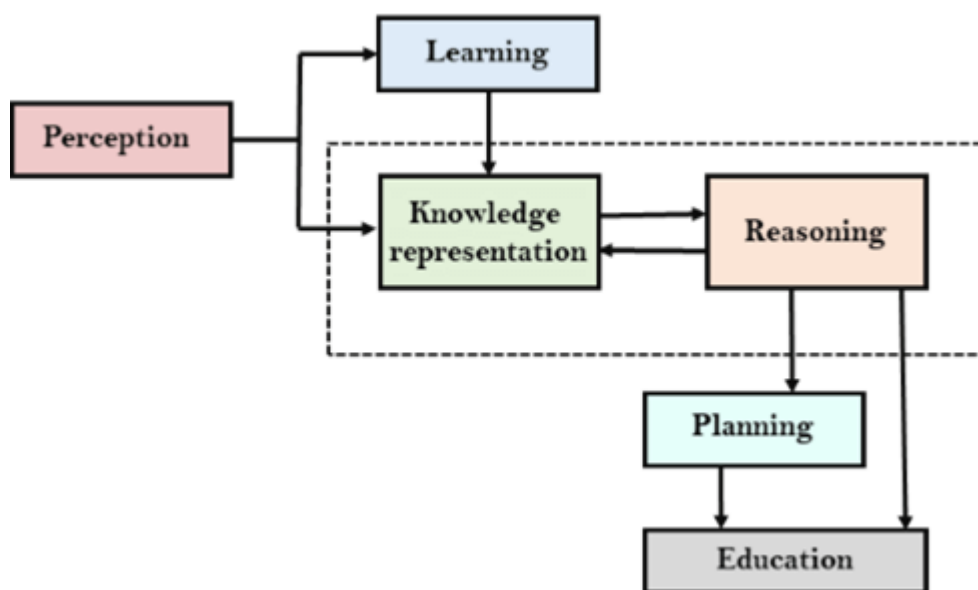
As we can see in below diagram, there is one decision maker which act by sensing the environment and using knowledge. But if the knowledge part will not present then, it cannot display intelligent behavior.



AI knowledge cycle:

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

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



The above diagram is showing how an AI system can interact with the real world and what components help it to show intelligence. AI system has Perception component by which it retrieves information from its environment.

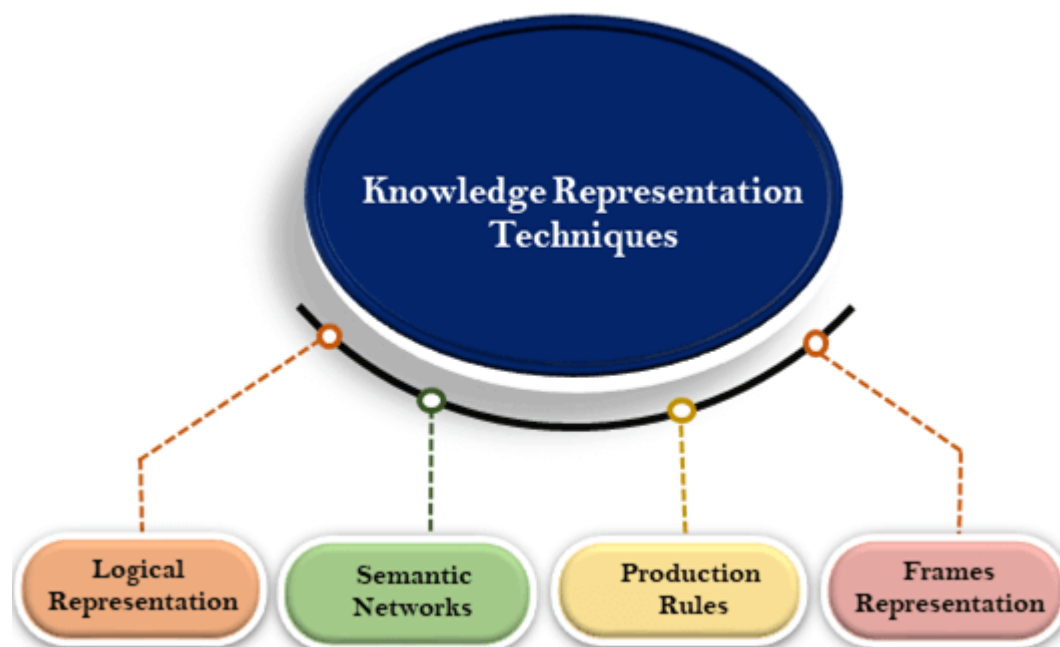
It can be visual, audio or another form of sensory input. The learning component is responsible for learning from data captured by Perception component. In the complete cycle, the main components are knowledge representation and Reasoning.

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.

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:

- Propositional Logics
- 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.

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.

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

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**

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).**

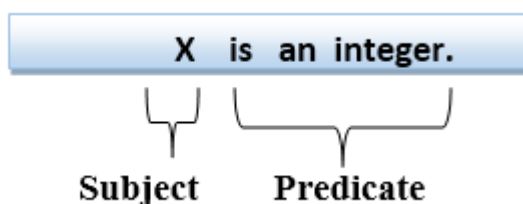
Complex Sentences:

- 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:
 - a. **Universal Quantifier, (for all, everyone, everything)**
 - b. **Existential quantifier, (for some, at least one).**

Universal Quantifier:

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

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

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

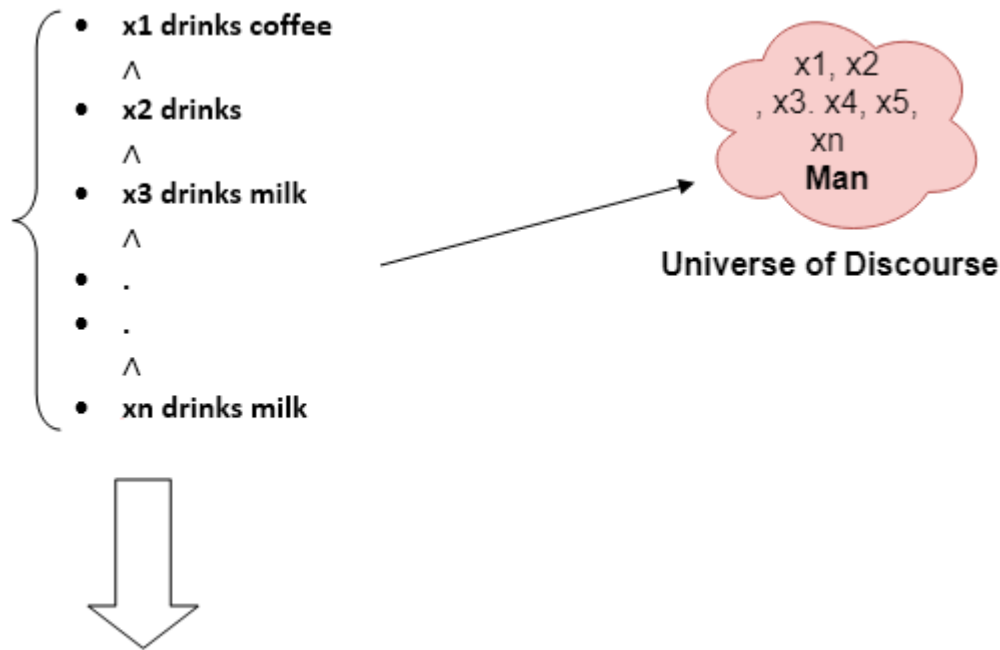
If x is a variable, then $\forall x$ is read as:

- **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.

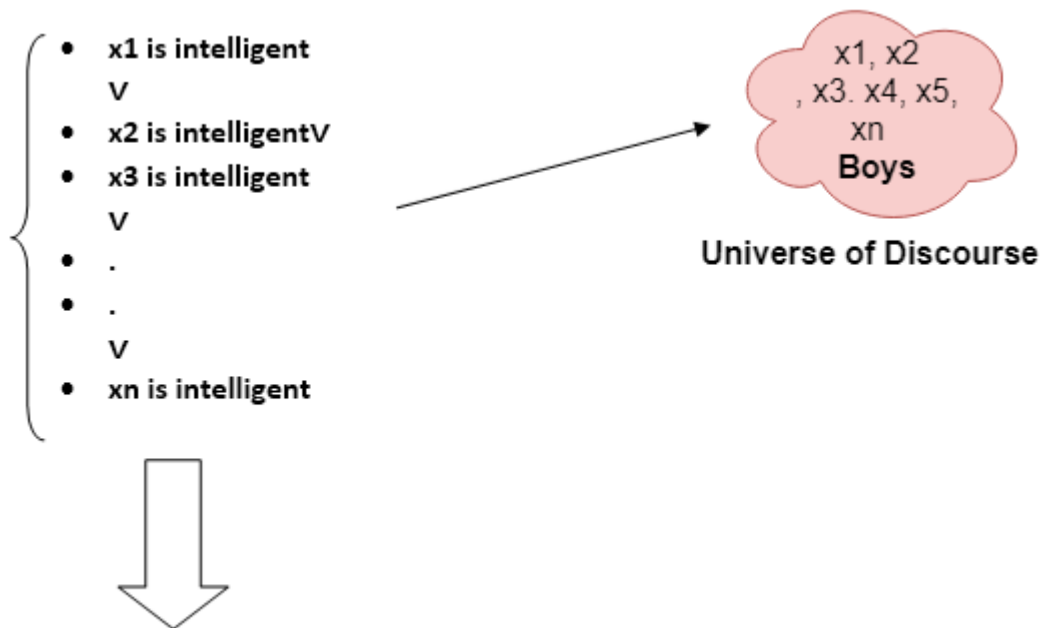
Note: In Existential quantifier we always use AND or Conjunction symbol (\wedge).

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:

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

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.