Chapter 1 – Introduction to Artificial Intelligence

Since the invention of computers or machines, their capability to perform various tasks went on growing exponentially. Humans have developed the power of computer systems in terms of their diverse working domains, their increasing speed, and reducing size with respect to time. A branch of Computer Science named Artificial Intelligence pursues creating the computers or machines as intelligent as human beings.

1. What is Artificial Intelligence?

According to the father of Artificial Intelligence, John McCarthy, it is ***"The science and engineering of making intelligent machines, especially intelligent computer programs".***

Artificial Intelligence is a way of making a computer, a computer-controlled robot, or a software think intelligently, in the similar manner the intelligent humans think.

AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

1.1 Goals of Artificial Intelligence

- To Create Expert Systems − The systems which exhibit intelligent behavior, learn, demonstrate, explain, and advice its users.
- To Implement Human Intelligence in Machines − Creating systems that understand, think, learn, and behave like humans.

1.2 What is Artificial Intelligence 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.

1.3 Programming with and without Artificial Intelligence

Programming without AI	Programming with AI
A computer program without AI can answer the **specific** questions it is meant to solve.	A computer program with AI can answer the **generic** questions it is meant to solve.
Modification in the program leads to change in its structure.	AI programs can absorb new modifications by putting highly independent pieces of information together. Hence you can modify even a minute piece of information of program without affecting its structure.
Modification is not quick and easy. It may lead to affecting the program adversely.	Quick and Easy program modification.

1.4 Applications of Artificial Intelligence

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.

1.5 Objectives of Artificial Intelligence

Artificial Intelligence is a science and engineering to develop intelligent machines that works like human to do tasks such as speech recognition, decision-making, reasoning, learning, problem solving and translation between languages.

In simple words, it is developed to make human life easy in different aspects and perform enormous works with more accuracy. It has become an intelligent part of today's industry which is crucial as data is growing Big. It can perform tasks such as identifying patterns in data more effectively than humans thus making business more profitable. Knowledge Engineering and Machine learning is a core part of AI. Some of the objectives of AI are:

➢ Know the difficulties that arise from attempting to define AI.

- Know the three areas of research of AI, and give examples of problems from each area.
- Understand in a general way how a neural network is designed and trained.
- Know the components of a formal system.
- Understand how depth first, breadth first, and bi-directional searches are performed.
- Use evaluation functions to expedite the search process.

1.6 Approaches of Artificial Intelligence

Following are the approaches to the goal of AI:

- Computer systems that act like humans,
- Programs that simulate the human mind,
- Knowledge representation and mechanistic reasoning, and
- Intelligent or rational agent design.

The first two approaches focus on studying humans and how they solve problems, while the latter two approaches focus on studying real-world problems and developing rational solutions regardless of how a human would solve the same problems.

Programming a computer to act like a human is a difficult task and requires that the computer system be able to understand and process commands in natural language, store knowledge, retrieve and process that knowledge in order to derive conclusions and make decisions, learn to adapt to new situations, perceive objects through computer vision, and have robotic capabilities to move and manipulate objects. Although this approach was inspired by the Turing Test, most programs have been developed with the goal of enabling computers to interact with humans in a natural way rather than passing the Turing Test.

The Turing Test is used as a theoretical standard to determine whether a human judge can distinguish via a conversation with one machine and one human which a human is and which is a machine. If a machine can trick the human judge into thinking it is human then it passes the Turing Test

1.7 History of Artificial Intelligence

The concept of AI began around 1943 and became a field of study in 1956 at

Dartmouth. AI is not limited to the Computer Sciences disciplines, but can be seen in countless disciplines such as Mathematics, Philosophy, Economics, Neuroscience, psychology and various other areas. The areas of interest in the Computer Science and Engineering field are focused on how we can build more efficient computers. Great advancements have been made in the area of hardware and software. Here is the history of AI during 20th century:

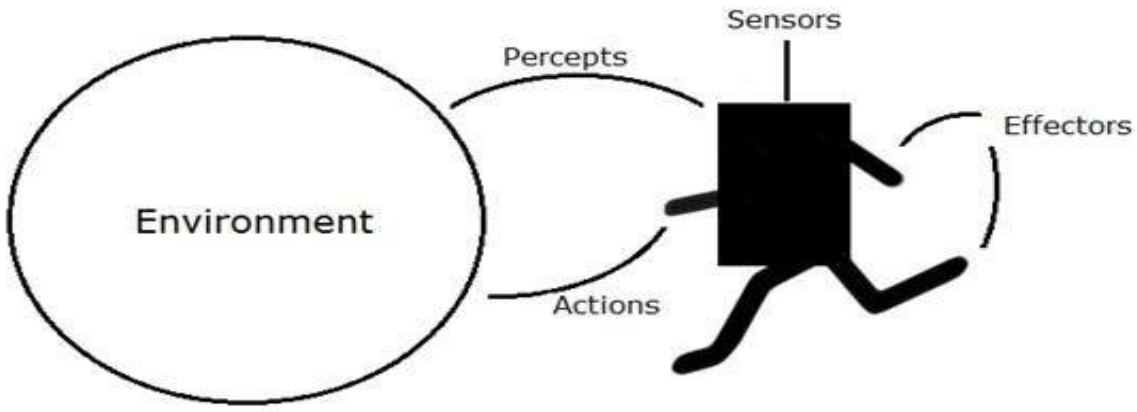
Year	Milestone / Innovation
1923	Karel Čapek play named “Rossum's Universal Robots” (RUR) opens in London, first use of the word "robot" in English.
1943	Foundations for neural networks laid.
1945	Isaac Asimov, a Columbia University alumni, coined the term *Robotics*.
1950	Alan Turing introduced Turing Test for evaluation of intelligence and published *Computing Machinery and Intelligence.* Claude Shannon published *Detailed Analysis of Chess Playing* as a search.
1956	John McCarthy coined the term *Artificial Intelligence*. Demonstration of the first running AI program at Carnegie Mellon University.
1958	John McCarthy invents LISP programming language for AI.
1964	Danny Bobrow's dissertation at MIT showed that computers can understand natural language well enough to solve algebra word problems correctly.
1965	Joseph Weizenbaum at MIT built *ELIZA*, an interactive problem that carries on a dialogue in English.
1969	Scientists at Stanford Research Institute Developed *Shakey*, a robot, equipped with locomotion, perception, and problem solving.
1973	The Assembly Robotics group at Edinburgh University built *Freddy*, the Famous Scottish Robot, capable of using vision to locate and assemble models.
1979	The first computer-controlled autonomous vehicle, Stanford Cart, was built.
1985	Harold Cohen created and demonstrated the drawing program, *Aaron*.

1990	Major advances in all areas of AI − • Significant demonstrations in machine learning • Case-based reasoning • Multi-agent planning • Scheduling • Data mining, Web Crawler • natural language understanding and translation • Vision, Virtual Reality • Games
1997	The Deep Blue Chess Program beats the then world chess champion, Garry Kasparov.
2000	Interactive robot pets become commercially available. MIT displays *Kismet*, a robot with a face that expresses emotions. The robot *Nomad* explores remote regions of Antarctica and locates meteorites.

Chapter 2 – Introduction to Intelligent Agents

2.1 An **agent** is anything that can perceive its environment through **sensors** and acts upon that environment through **effectors.**

- A **human agent** has sensory organs such as eyes, ears, nose, tongue and skin parallel to the sensors, and other organs such as hands, legs, mouth, for effectors.
- A **robotic agent** replaces cameras and infrared range finders for the sensors, and various motors and actuators for effectors.
- A **software agent** has encoded bit strings as its programs and actions.



An intelligent agent is an AI hardware and/or software system with some degree of autonomy and the capacity to make decisions and take actions. Intelligent agents are more advanced than conventional agents whose actions are completely programmed. An agent program might, for example, use human-defined parameters to search a knowledge base or the Internet and organize that information for presentation to the user.

Intelligent agents, at their most complexes, include physical systems, such as AI-equipped android (humanoid) robots. Although it has not yet been demonstrated, some believe that a future system could not only look like a human but could also replicate human cognitive powers in artificial general intelligence (AGI) or even overwhelmingly surpass it in artificial super intelligence (ASI). At the lower end of the scale lie more task-specific software programs, such as expert systems that work similarly to conventional agent programs but nevertheless some degree of autonomy and physical agents that take in sensor data and act on it.

2.2 Agent Terminology

- Performance Measure of Agent − It is the criteria, which determines how successful an agent is.
- Behavior of Agent − It is the action that agent performs after any given sequence of percepts.
- Percept − It is agent's perceptual inputs at a given instance.
- Percept Sequence − It is the history of all that an agent has perceived till date.
- Agent Function − It is a map from the precept sequence to an action.

Agent Rationality

Rationality is nothing but status of being reasonable, sensible, and having good sense of judgment.

Rationality is concerned with expected actions and results depending upon what the agent has perceived. Performing actions with the aim of obtaining useful information is an important part of rationality.

Ideal Rational Agent

An ideal rational agent is the one, which is capable of doing expected actions to maximize its performance measure, on the basis of:

- Its percept sequence
- Its built-in knowledge base

Rationality of an agent depends on the following:

- The **performance measures**, which determine the degree of success.
- Agent's **Percept Sequence** till now.
- The agent's **prior knowledge about the environment**.
- The **actions** that the agent can carry out.

A rational agent always performs right action, where the right action means the action that causes the agent to be most successful in the given percept sequence. The problem the agent solves is characterized by Performance Measure, Environment, Actuators, and Sensors (PEAS).

2.3 Structure of Intelligent Agents

Agent's structure can be viewed as:

- Agent = Architecture + Agent Program
- Architecture = the machinery that an agent executes on.
- Agent Program = an implementation of an agent function.

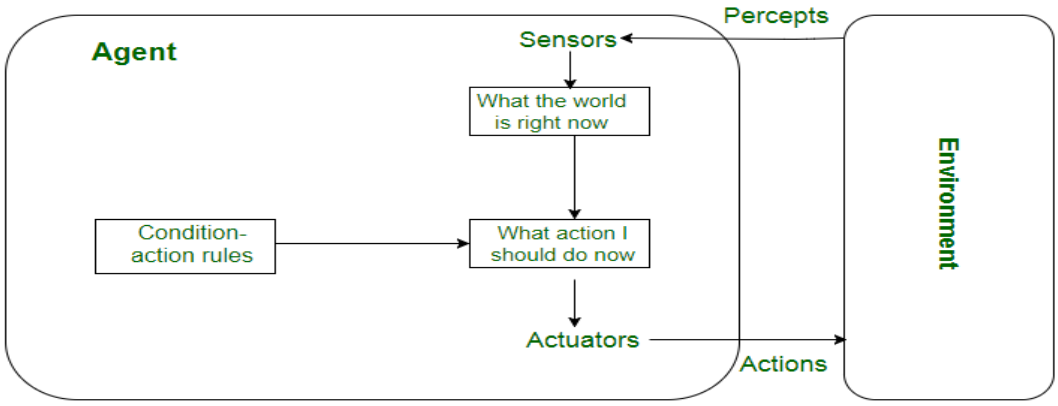
Types of Agents

Agents can be grouped into four classes based on their degree of perceived intelligence and capability:

- Simple Reflex Agents
- Model-Based Reflex Agents
- Goal-Based Agents
- Utility-Based Agents

Simple Reflex Agents

Simple reflex agents ignore the rest of the percept history and act only on the basis of the **current percept**. Percept history is the history of all that an agent has perceived till date. The agent function is based on the **condition-action rule**. A condition-action rule is a rule that maps a state i.e, condition to an action. If the condition is true, then the action is taken, else not. This agent function only succeeds when the environment is fully observable. For simple reflex agents operating in partially observable environments, infinite loops are often unavoidable. It may be possible to escape from infinite loops if the agent can randomize its actions. Problems with Simple reflex agents are:

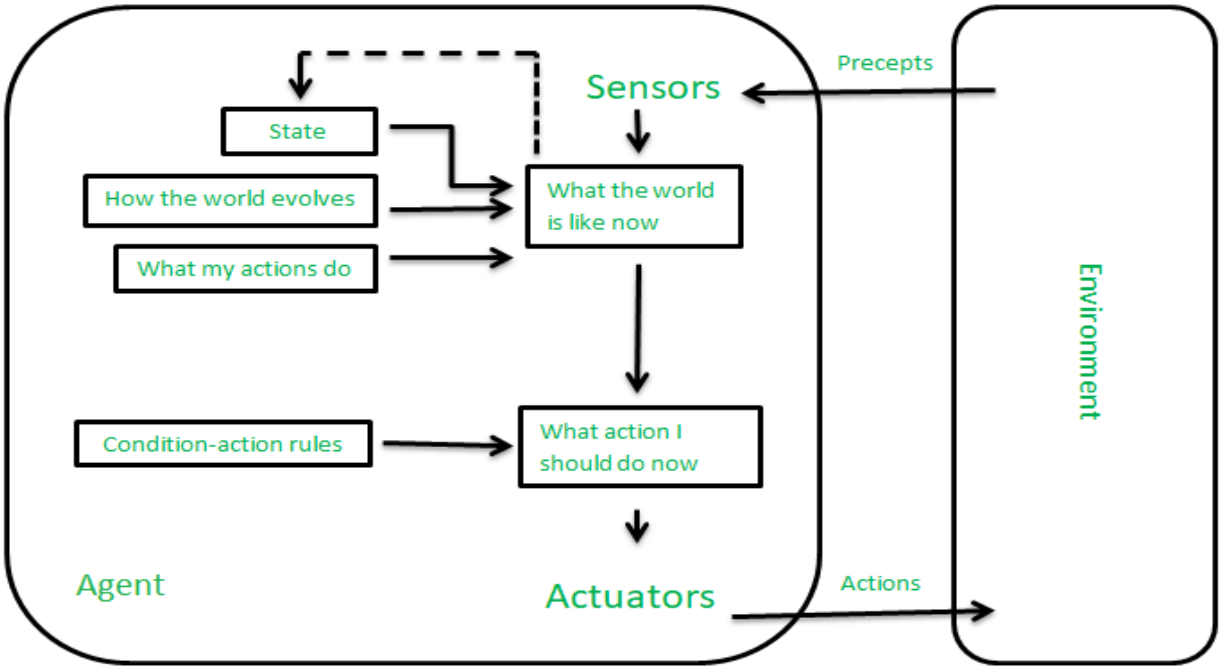


- They choose actions only based on the current percept.
- They are rational only if a correct decision is made only on the basis of current precept.
- Their environment is completely observable.
- Very limited intelligence.
- No knowledge of non-perceptual parts of state.
- Usually too big to generate and store.
- If there occurs any change in the environment, then the collections of rules need to be updated.

Model Based Reflex Agent

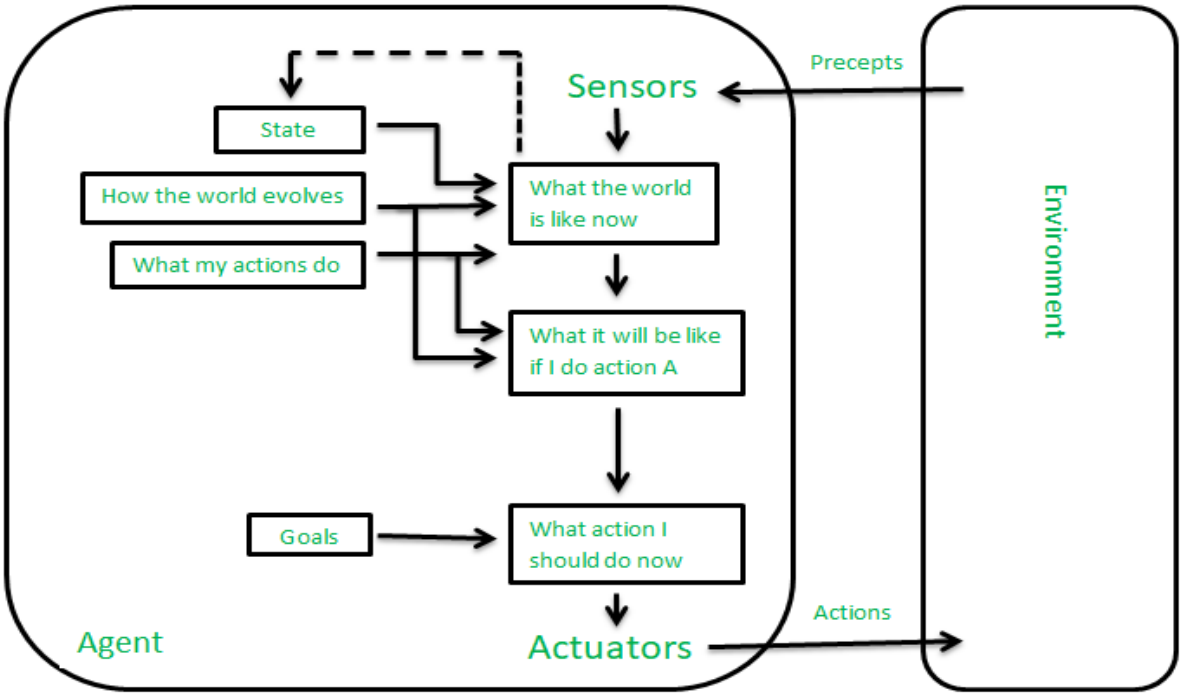
It works by finding a rule whose condition matches the current situation. A model-based agent can handle **partially observable environments** by use of model about the world. The agent has to keep track of **internal state** which is adjusted by each percept and that depends on the percept history. The current state is stored inside the agent which maintains some kind of structure describing the part of the world which cannot be seen. Updating the state requires the information about:

- how the world evolves in-dependently from the agent, and
- how the agent actions affects the world.



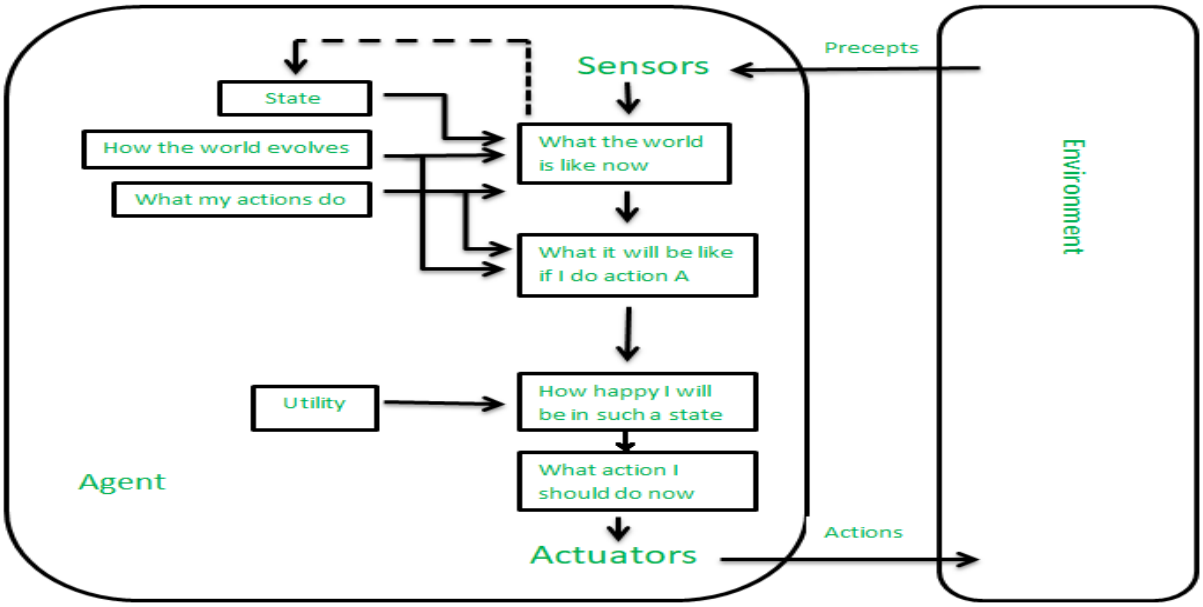
Goal Based Agent

These kinds of agents take decision based on how far they are currently from their **goal** (description of desirable situations). Their every action is intended to reduce its distance from goal. This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state. The knowledge that supports its decisions is represented explicitly and can be modified, which makes these agents more flexible. They usually require search and planning. The goal based agent's behavior can easily be changed.



Utility Based Agents

The agents which are developed having their end uses as building blocks are called utility based agents. When there are multiple possible alternatives, then to decide which one is best, utility based agents are used. They choose actions based on a **preference (utility)** for each state. Sometimes achieving the desired goal is not enough. We may look for quicker, safer, cheaper trip to reach a destination. Agent happiness should be taken into consideration. Utility describes how **"happy"** the agent is. Because of the uncertainty in the world, a utility agent chooses the action that maximizes the expected utility. A utility function maps a state onto a real number which describes the associated degree of happiness.



Goals are inadequate when:

- There are conflicting goals, out of which only few can be achieved.
- Goals have some uncertainty of being achieved and you need to weigh likelihood of success against the importance of a goal.

2.4 Nature of Environment

Some programs operate in the entirely **artificial environment** confined to keyboard input, database, computer file systems and character output on a screen.

In contrast, some software agents (software robots or softbots) exist in rich, unlimited softbots domains. The simulator has a **very detailed, complex environment**. The software agent needs to choose from a long array of actions in real time. A softbot designed to scan the online preferences of the customer and show interesting items to the customer works in the **real** as well as an **artificial** environment.

The most famous **artificial environment** is the **Turing Test environment**, in which one real and other artificial agent are tested on equal ground. This is a very challenging environment as it is highly difficult for a software agent to perform as well as a human.

Turing Test

The success of an intelligent behavior of a system can be measured with Turing Test. Two persons and a machine to be evaluated participate in the test. Out of the two persons, one plays the role of the tester. Each of them sits in different rooms. The

tester is unaware of who is machine and who is a human. He interrogates the questions by typing and sending them to both intelligences, to which he receives typed responses.

This test aims at fooling the tester. If the tester fails to determine machine's response from the human response, then the machine is said to be intelligent.

Properties of Environment

The environment has multifold properties:

- **Discrete / Continuous** − If there are a limited number of distinct, clearly defined, states of the environment, the environment is discrete (For example, chess); otherwise it is continuous (For example, driving).
- **Observable / Partially Observable** − If it is possible to determine the complete state of the environment at each time point from the percepts it is observable; otherwise it is only partially observable.
- **Static / Dynamic** − If the environment does not change while an agent is acting, then it is static; otherwise it is dynamic.
- **Single agent / Multiple agents** − The environment may contain other agents which may be of the same or different kind as that of the agent.
- **Accessible / Inaccessible** − If the agent's sensory apparatus can have access to the complete state of the environment, then the environment is accessible to that agent.
- **Deterministic / Non-deterministic** − If the next state of the environment is completely determined by the current state and the actions of the agent, then the environment is deterministic; otherwise it is non-deterministic.
- **Episodic / Non-episodic** − In an episodic environment, each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself. Subsequent episodes do not depend on the actions in the previous episodes. Episodic environments are much simpler because the agent does not need to think ahead.

Chapter 3 – Problem Solving

3.1 Classical Approach of Problem Solving

The classical approach to solving a problem is pretty simple; given a problem at hand use hit and trial method to check for various solutions to that problem. This hit and trial approach usually works well for trivial problems and is referred to as the classical approach to problem solving.

Consider the maze searching problem where the mouse travels though one path and finds that the path leads to a dead end, it then back tracks somewhat and goes along some other path and again finds that there is no way to proceed. It goes on performing such search, trying different solutions to solve the problem until a sequence of turns in the maze takes it to the cheese. Hence, of all the solutions the mouse tries, the one that reached the cheese was the one that solved the problem.

Consider that a child is to switch on the light in a dark room. He sees the switchboard having a number of buttons on it. He presses one, nothing happens, he presses the second one, the fan gets on, he goes on trying different buttons till at last the room gets lighted and his problem gets solved. Consider another situation when we have to open a combinational lock of a briefcase. It is a lock which probably most of you would have seen where we have different numbers and we adjust the individual dials/digits to obtain a combination that opens the lock. However, if we don't know the correct combination of digits that open the lock, we usually try 0-0-0, 7-7-7, 7-8-6 or any such combination for opening the lock. We are solving this problem in the same manner as the child did in the light switch example.

All this discussion has one thing in common. That different intelligent species use a similar approach to solve the problem at hand. This approach is essentially the classical way in which intelligent species solve problems. Technically we call this hit and trial approaches the "Generate and Test" approach.

Components of Problem Solving

There are four components of Problem solving such as Problem Statement, Goal State, Solution Space and Operators.

- **Problem Statement**

This is the very essential component whereby we get to know what exactly the problem at hand is. The two major things that we get to know about the problem is the

Information about what is to be done and constraints to which our solution should comply. For example we might just say that given infinite amount of time, one will be able to solve any problem he wishes to solve. But the constraint "infinite amount of time" is not a practical one. Hence whenever addressing a problem we have to see that how much time shall out solution take at max. Time is not the only constraint. Availability of resources, and all the other parameters laid down in the problem statement actually tells us about all the rules that have to be followed while solving a problem. For example, taking the same example of the mouse, are problem statements that will tell us things like, the mouse has to reach the cheese as soon as possible and in case it is unable to find a path within an hour, it might die of hunger. The statement might as well tell us that the mouse is located in the lower left corner of the maze and the cheese in the top left corner, the mouse can turn left, right and might or might not be allowed to move backward and things like that. Thus it is the problem statement that gives us a feel of what exactly to do and helps us start thinking of how exactly things will work in the solution.

- **Problem Solution/Goal State**

  While solving a problem, this should be known that what will be our ultimate aim. That is what should be the output of our procedure in order to solve the problem. For example in the case of mouse, the ultimate aim is to reach the cheese. The state of word when mouse will be beside the cheese and probably eating it defines the aim. This state of word is also referred to as the Goal State or the state that represents the solution of the problem.

- **Solution Space**

  In order to reach the solution we need to check various strategies. We might or might not follow a systematic strategy in all the cases. Whatever we follow, we have to go through a certain amount of states of nature to reach the solution. For example when the mouse was in the lower left corner of the maze, represents a state i.e. the start state. When it was stuck in some corner of the maze represents a state. When it was stuck somewhere else represents another state. When it was travelling on a path represents some other state and finally when it reaches the cheese represents a state called the goal state. The set of the start state, the goal state and all the intermediate states constitutes something which is called a solution space.

- **Operators (Travelling in the Solution Space)**

  We have to travel inside this solution space in order to find a solution to our problem. The travelling inside a solution space requires something called as “operators”. In case of the mouse example, turn left, turn right, and go straight are the operators which help us travelling inside the solution space. In short the action that takes us from one state to the other is referred to as an operator. So while solving a problem we should clearly know that what are the operators that we can use in order to reach the goal state from the starting state. The sequence of these operators is actually the solution to our problem.

3.2 Problem Solving Agents

There are four general steps for problem solving which are also referred as Goal based agents. Here we will take an example of travelling from Teppi to Addis Ababa.

- **Goal Formulation**
  - Declaring the Goal.
  - Ignoring the some actions.
  - Limits the objective that agent is trying to achieve.
  - Goal can be defined as set of world states.
- **Problem Formulation**
  - What actions and states to consider given the goal.
  - It can change the world states size, enormously.
  - It depends on how the agent is connected to its environment.
- **Search**
  - Now, our agent knows the goal and actions.
  - Which action to choose (with map or with-out map)?
  - What action should be chosen in a state with unknown value?
  - Process of looking for such a sequence is called Search.
  - Search Algorithm takes problem as input and returns a solution in form of an action sequence.
- **Execute**
  - After finding a suitable action sequence, the actions can be carried out.
  - To solve a problem: “Formulate, Search, and Execute”.

3.3 Search Strategies Algorithm

Searching is the universal technique of problem solving in AI. There are some single-player games such as tile games, Sudoku, crossword, etc. The search algorithms help you to search for a particular position in such games.

Single Agent Path finding Problems

The games such as 3X3 eight-tile, 4X4 fifteen-tile, and 5X5 twenty four tile puzzles are single-agent-path-finding challenges. They consist of a matrix of tiles with a blank tile. The player is required to arrange the tiles by sliding a tile either vertically or horizontally into a blank space with the aim of accomplishing some objective.

The other examples of single agent path finding problems are Travelling Salesman Problem, Rubik's Cube, and Theorem Proving.

Brute Force Search Strategies

They are most simple, as they do not need any domain-specific knowledge. They work fine with small number of possible states.

Requirements −

- State description
- A set of valid operators
- Initial state
- Goal state description

This search strategy includes Breadth First Search, Depth First Search, Bidirectional Search, Uniform Cost Search, and Iterative Deepening Depth First Search.

- **Breadth First Search**

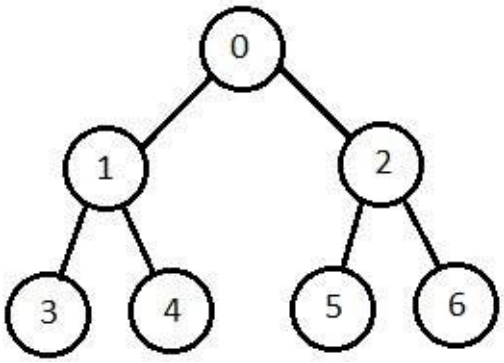
It starts from the root node, explores the neighboring nodes first and moves towards the next level neighbors. It generates one tree at a time until the solution is found. It can be implemented using FIFO (First In First Out) queue data structure. This method provides shortest path to the solution.

If branching factor (average number of child nodes for a given node) = b and depth = d, then number of nodes at level d = b^d.

The total no of nodes created in worst case is $b + b^2 + b^3 + \ldots + b^d$.

Disadvantage − Since each level of nodes is saved for creating next one, it consumes a lot of memory space. Space requirement to store nodes is exponential.

Its complexity depends on the number of nodes. It can check duplicate nodes.

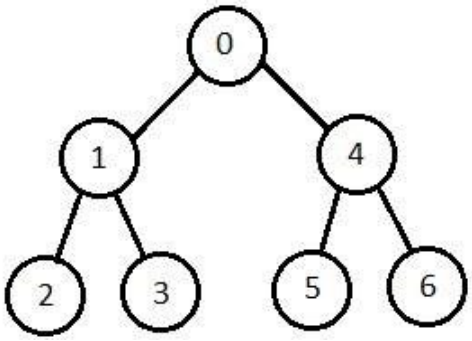


- **Depth First Search**

  It is implemented in recursion with LIFO (Last In First Out) stack data structure. It creates the same set of nodes as Breadth-First method, only in the different order. As the nodes on the single path are stored in each iteration from root to leaf node, the space requirement to store nodes is linear. With branching factor *b*and depth as *m*, the storage space is *bm.*

  Disadvantage − This algorithm may not terminate and go on infinitely on one path. The solution to this issue is to choose a cut-off depth. If the ideal cut-off is *d*, and if chosen cut-off is lesser than *d*, then this algorithm may fail. If chosen cut-off is more than *d*, then execution time increases.

  Its complexity depends on the number of paths. It cannot check duplicate nodes.



- **Bidirectional Search**

  It searches forward from initial state and backward from goal state till both meet to identify a common state.

  The path from initial state is concatenated with the inverse path from the goal state. Each search is done only up to half of the total path.

- **Uniform Cost Search**

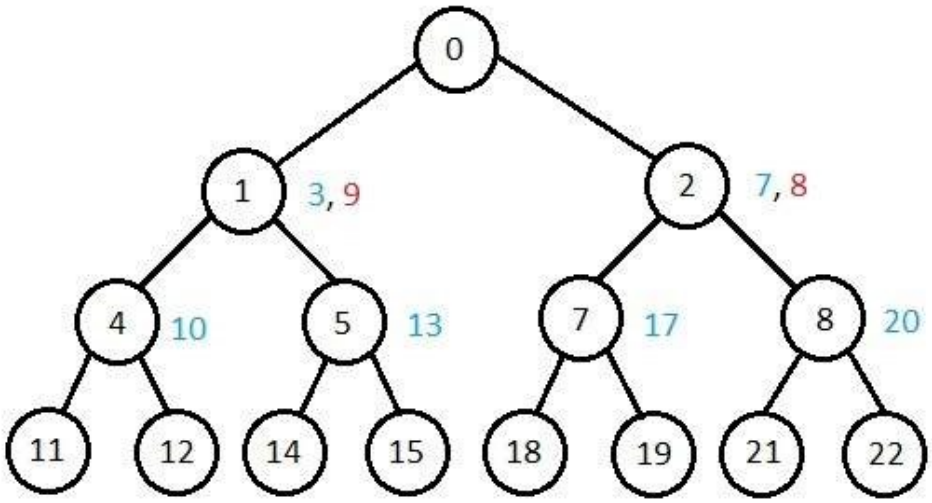
  Sorting is done in increasing cost of the path to a node. It always expands the least cost node. It is identical to Breadth First search if each transition has the same cost.

  It explores paths in the increasing order of cost.

  Disadvantage − There can be multiple long paths with the cost ≤ C*. Uniform Cost search must explore them all.

- **Iterative Deepening Depth First Search**

  It performs depth-first search to level 1, starts over, executes a complete depth-first search to level 2, and continues in such way till the solution is found. It never creates a node until all lower nodes are generated. It only saves a stack of nodes. The algorithm ends when it finds a solution at depth *d*. The number of nodes created at depth *d* is b^d and at depth *d-1* is $b^{d-1.}$



Comparision of Various Algorithms Complexities

Criterion	Breadth First	Depth First	Bidirectional	Uniform Cost	Interactive Deepening
Time	b^d	b^m	$b^{d/2}$	b^d	b^d
Space	b^d	b^m	$b^{d/2}$	b^d	b^d
Optimality	Yes	No	Yes	Yes	Yes
Completeness	Yes	No	Yes	Yes	Yes

3.4 Informed (Heuristic) Search Strategies

To solve large problems with large number of possible states, problem-specific knowledge needs to be added to increase the efficiency of search algorithms.

- **Heuristic Evaluation Functions**

They calculate the cost of optimal path between two states. A heuristic function for sliding-tiles games is computed by counting number of moves that each tile makes from its goal state and adding these numbers of moves for all tiles.

- **Pure Heuristic Search**

It expands nodes in the order of their heuristic values. It creates two lists, a closed list for the already expanded nodes and an open list for the created but unexpanded nodes.

In each iteration, a node with a minimum heuristic value is expanded; all its child nodes are created and placed in the closed list. Then, the heuristic function is applied to the child nodes and they are placed in the open list according to their heuristic value. The shorter paths are saved and the longer ones are disposed.

- **A * Search**

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.

f(n) = g(n) + h(n), where

→ g(n) the cost (so far) to reach the node

→ h(n) estimated cost to get from the node to the goal

→ f(n) estimated total cost of path through n to goal. It is implemented using priority queue by increasing f (n).

- **Greedy Best First Search**

It expands the node that is estimated to be closest to goal. It expands nodes based on f(n) = h(n). It is implemented using priority queue.

Disadvantage − It can get stuck in loops. It is not optimal.

3.5 Local Search Algorithms

They start from a prospective solution and then move to a neighbouring solution. They can return a valid solution even if it is interrupted at any time before they end.

Hill-Climbing Search

It is an iterative algorithm that starts with an arbitrary solution to a problem and attempts to find a better solution by changing a single element of the solution incrementally. If the change produces a better solution, an incremental change is taken as a new solution. This process is repeated until there are no further improvements.

function Hill-Climbing (problem), returns a state that is a local maximum.

```
inputs: problem, a problem
local variables: current, a node
                 neighbor, a node
current <-Make_Node(Initial-State[problem])
loop
  do neighbor <- a highest_valued successor of current
    if Value[neighbor] ≤ Value[current] then
    return State[current]
    current <- neighbor

end
```

Disadvantage − This algorithm is neither complete, nor optimal.

- **Local Beam Search**

In this algorithm, it holds k number of states at any given time. At the start, these states are generated randomly. The successors of these k states are computed with the help of objective function. If any of these successors is the maximum value of the objective function, then the algorithm stops.

Otherwise the (initial k states and k number of successors of the states = 2k) states are placed in a pool. The pool is then sorted numerically. The highest k states are selected as new initial states. This process continues until a maximum value is reached.

function BeamSearch (*problem, k*), returns a solution state.

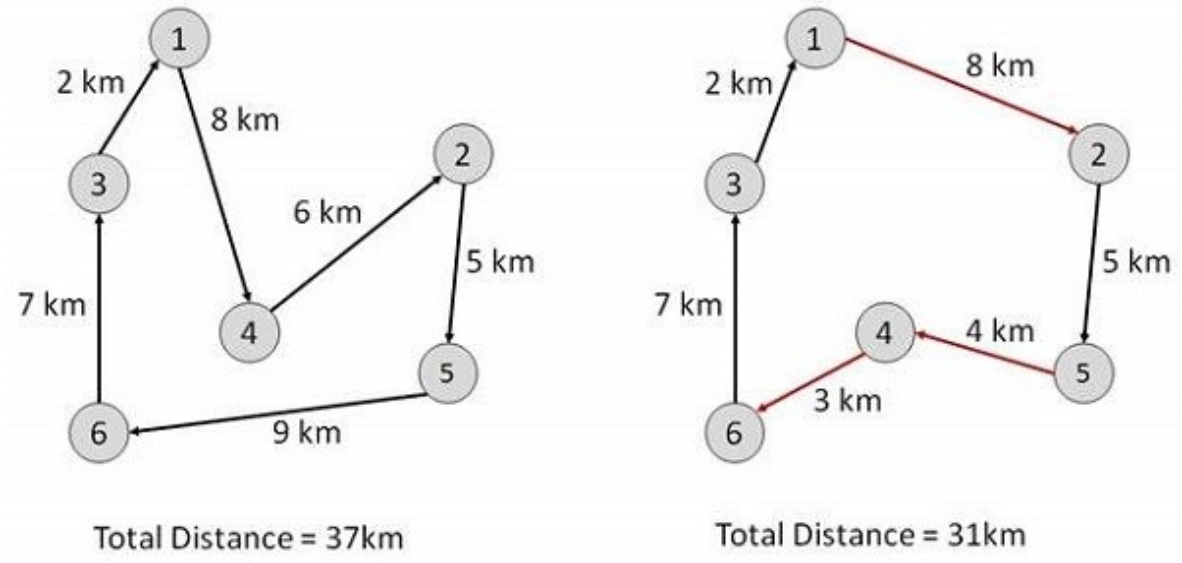
```
start with k randomly generated states
loop
  generate all successors of all k states
  if any of the states = solution, then return the state
  else select the k best successors
end
```

- **Travelling Salesman Problem**

The traveling salesman problem (TSP) is an algorithmic problem tasked with finding the shortest route between a set of points and locations that must be visited. In the problem statement, the points are the cities a salesperson might visit. The salesman‘s goal is to keep both the travel costs and the distance traveled as low as possible.

In this algorithm, the objective is to find a low-cost tour that starts from a city, visits all cities en-route exactly once and ends at the same starting city.

```
Start
  Find out all (n -1)! Possible solutions, where n is the total number of cities.
  Determine the minimum cost by finding out the cost of each of these (n -1)! solutions.
  Finally, keep the one with the minimum cost.
end
```



- **Simulated Annealing**

Annealing is the process of heating and cooling a metal to change its internal structure for modifying its physical properties. When the metal cools, its new structure is seized, and the metal retains its newly obtained properties. In simulated annealing process, the temperature is kept variable.

We initially set the temperature high and then allow it to ‘cool' slowly as the algorithm proceeds. When the temperature is high, the algorithm is allowed to accept worse solutions with high frequency.

Chapter 4 – Knowledge and Reasoning

4.1 Introduction to Statement / Propositions

In order to make arguments rigorous, we need to develop a language in which we can express sentences in such a way that brings out their logical structure. The language we begin with is the language of propositional logic. It is based on propositions, or declarative sentences which one can, in principle, argue as being true or false. Examples of declarative sentences are:

1) The sum of the numbers 3 and 5 equals 8.
2) Bontu reacted violently to Abebe's accusations.
3) Every even natural number >2 is the sum of two prime numbers.
4) All Ethiopians like Thibs with Injira.
5) Albert Camus ´etait un ´ecrivain fran， cais.
6) Die W¨urde des Menschen ist unantastbar.

These sentences are all declarative, because they are in principle capable of being declared 'true', or 'false'.

Sentence (1) can be tested by appealing to basic facts about arithmetic (and by tacitly assuming an Arabic, decimal representation of natural numbers).

Sentence (2) is a bit more problematic. In order to give it a truth value, we need to know who Bontu and Abebe are and perhaps to have a reliable account from someone who witnessed the situation described. In principle, e.g., if we had been at the scene, we feel that we would have been able to detect Bontu's violent reaction, provided that it indeed occurred in that way.

Sentence (3), known as Goldbach's conjecture, seems straightforward on the face of it. Clearly, a fact about all even numbers >2 is either true or false. But to this day nobody knows whether sentence (3) expresses a truth or not. It is even not clear whether this could be shown by some finite means, even if it were true. However, in this text we will be content with sentences as soon as they can, in principle, attain some truth value regardless of whether this truth value reflects the actual state of affairs suggested by the sentence in question.

Sentence (4) seems a bit silly, although we could say that if Ethiopians eat Thibs then

all of them will either like Injira on it or not. Because someone may like bread, ambasha or rus with Thibs.

Sentences (5) and (6) are fine if you happen to read French and German a bit. Thus, declarative statements can be made in any natural, or artificial, language. The kind of sentences we won't consider here are non-declarative ones, like

- Could you please pass me the salt?
- Ready, steady, go!
- May fortune come your way?

4.2 Logical Connectives

Definitions: A logical connective (also called a logical operator) is a symbol or word used to connect two or more sentences (of either a formal or a natural language) in a grammatically valid way, such that the sense of the compound sentence produced depends only on the original sentences. In terms of logical form, statements are built from simpler statements using *logical connectives*. The basic connectives of sentential logic are:

(a) Negation ("not"), denoted $\sim$.

(b) *Conjunction* ("and"), denoted $\wedge$.

(c) *Disjunction* ("or"), denoted $\vee$.

(d) *Conditional* ("if-then" or "implication"), denoted $\rightarrow$.

(e) *Biconditional* ("if and only if" or "double implication"), denoted $\leftrightarrow$.

Note: You may see different symbols used by other people. For example, some people use $\neg$ for negation. And $\Rightarrow$ is sometimes used for the conditional, in which case $\Leftrightarrow$ is used for the biconditional.

Example 1: **Represent the following statements using logical connectives.**

Given statements P and Q, we can combine them with various connectives.

(a) P or not Q. $P \vee \sim Q$

(b) If P and R, then Q. $(P \wedge R) \rightarrow Q$

(c) P if and only if (Q and R). $P \leftrightarrow (Q \wedge R)$

(d) Not P and not Q. $\sim P \wedge \sim Q$

(e) It is not the case that if P, then Q. $\sim (P \rightarrow Q)$

(f) If P and Q, then R or S. $(P \wedge Q) \rightarrow (R \vee S)$

Other words or phrases may occur in statements. Here's a list of some of them and how they are translated.

Phrase	Logical translation
P, but Q	$P \wedge Q$
Either P or Q	$P \vee Q$
P or Q, but not both	$(P \vee Q) \wedge \sim (P \wedge Q)$
P if Q	$Q \rightarrow P$
P is necessary for Q	$Q \rightarrow P$
P is sufficient for Q	$P \rightarrow Q$
P only if Q	$P \rightarrow Q$
P is equivalent to Q	$P \leftrightarrow Q$
P whenever Q	$Q \rightarrow P$

Consider the word "but", for example. If I say "Tesfaye is here, but Bontu is there", I mean that Tesfaye is here *and* Bontu is there. My intention is that both of the statements should be true. That is the same as what I mean when I say "Tesfaye is here and Bontu is there".

In practice, mathematicians tend to a small set of phrases over and over. It doesn't make for exciting reading, but it allows the reader to concentrate on the meaning of what is written. For example, a mathematician will usually say "if Q, then P", rather than the logically equivalent "P whenever Q". The second statement is less familiar, and therefore more apt to be misunderstood.

This is a good time to discuss the way the word "or" is used in mathematics. When you say "I'll have dinner at Armahic Hotel **or** at Kanain Hotel", you *probably* mean "or" in its *exclusive* sense: You'll have dinner either at Armahic Hotel *or* you'll have dinner at

Kanain Hotel, *but not both*. The "but not both" is what makes this an *exclusive or*.

Mathematicians use "or" in the *inclusive* sense. When "or" is used in this way, "I'll have dinner at Armahic Hotel or at Kanain Hotel " means you'll have dinner at Armahic Hotel *or* you'll have dinner at Kanain Hotel, *or possibly both*. Obviously, I'm not *guaranteeing* that both will occur; I'm just not ruling it out.

Example 2: Translate the following statements into logical notation, using the following symbols:

S = "The **Shiiroo** is hot."

L = "The **laslasa** is cold."

P = "The **pizza** will be delivered."

(a) "The **Shiiroo** is hot and the **pizza** will not be delivered." $S \wedge \sim P$

(b) "If the **laslasa** is cold, then the **pizza** will be delivered." $L \rightarrow P$

(c) "Either the **laslasa** is cold or the **pizza** won't be delivered." $L \vee \sim P$

(d) "If the **pizza** won't be delivered, then both the **Shiiroo** is hot and the **laslasa** is cold."

$\sim P \rightarrow (S \wedge L)$

(e) "The **laslasa** isn't cold if and only if the **Shiiroo** isn't hot." $\sim L \leftrightarrow \sim S$

(f) "The **pizza** will be delivered only if the **laslasa** is cold." $P \rightarrow L$

(g) "The **Shiiroo** is hot and the **laslasa** isn't cold, but the **pizza** will be delivered."

$S \wedge \sim L \wedge P$

4.3 Introduction to Propositional Logic

Definition: A proposition or statement is a sentence which is either true or false. If a proposition is true, then we say its truth value is true, and if a proposition is false, we say its truth value is false.

Are these propositions?

1. The sun is shining.
2. The sum of two prime numbers is even.
3. 3+4=7
4. It rained in Teppi on February 01, 2019
5. Is it raining?
6. Come to class!
7. The moon is made of green cheese.

Definition: A propositional variable represents an arbitrary proposition. We represent propositional variables with uppercase letters.

What is an Argument?

Definition: An argument consists of a sequence of statements called premises and a statement called a conclusion. An argument is valid if the conclusion is true whenever the premises are all true.

Example-1: My program won't compile or it produces a division by 0 error. My program does not produce a division by 0 error. Therefore my program will not compile. Now: Rewrite this argument in its general form by defining appropriate propositional variables. This is one example of an argument form that is called disjunctive syllogism.

Example-2: Another disjunctive syllogism example Ladybugs are purple or green. Ladybugs are not green. Therefore ladybugs are purple.

Let P be: Ladybugs are purple. Let Q be: Ladybugs are green. Rewritten argument: P or Q not Q Therefore P This argument is valid, but it isn't very meaningful since P and Q are not true.

Logic format - *"if it was so, it might be, and if it were so, it would be; but as it isn't, it aren't. That's logic!"*

Prepositional Logic

To develop a language in which we can express sentences in such a way that brings out their logical structure. The language we begin with is the language of propositional logic. It is based on propositions,

Let us assign certain distinct symbols p, q, r, . . ., or sometimes p1, p2, p3, . . . to each of these atomic sentences and we can then code up more complex sentences in a compositional way. For example, given the atomic sentences

p: 'I won the lottery last week.'

q: 'I purchased a lottery ticket.'

r: 'I won last week's sweepstakes.'

We can form more complex sentences according to the rules below:

¬: The negation of p is denoted by ¬p and expresses 'I did not win the lottery last week,' or equivalently 'It is not true that I won the lottery last week.'

∨: Given p and r we may wish to state that at least one of them is true: 'I won the lottery last week, or I won last week's sweepstakes;' we denote this declarative sentence by p ∨ r and call it the disjunction of p and r.

∧: Dually, the formula p ∧ r denotes the rather fortunate conjunction of p and r: 'Last week I won the lottery and the sweepstakes.'

→: Last, but definitely not least, the sentence 'If I won the lottery last week, then I purchased a lottery ticket.' expresses an implication between p and q, suggesting that q is a logical consequence of p. We write p → q for that. We call p the assumption of p → q and q its conclusion. Of course, we are entitled to use these rules of constructing propositions repeatedly.

For example, we are now in a position to form the proposition p ∧ q → ¬r ∨ q which means that 'if p and q then not r or q'. You might have noticed a potential ambiguity in this reading. One could have argued that this sentence has the structure 'p is the case and if q then . . . ' A computer would require the insertion of brackets, as in (p ∧ q) → ((¬r) ∨ q).

The Syntax of Propositional Logic

The expressions of this language are called well-formed formulas or wffs.

1) Every propositional symbol is a wff. (We use p, p1, p2, p3,. . . , q, q1, q2, q3 . . . as propositional symbols.)

2) For any wff W, the following is also a wff: ¬W

3) For any two wffs W1 and W2, the following are also wffs: W1 ∧ W2, W1 ∨ W2, W1 ⇒ W2, W1 ⇔ W2 .

4) Any wff may be enclosed in parentheses and will still be a wff.

5) No other string of symbols is a wff.

Rules (2) and (3) form compound wffs from other wffs.

The symbols ¬,∧,∨,⇒ and ⇔ are the connectives. Here are examples of wffs: Propositional Logic Statements

p	q	p_1
$(\neg p)$	$(\neg q)$	$(\neg(\neg q))$
$(p \wedge p)$	$(p \vee q)$	$(p_1 \Rightarrow (\neg q_1))$
$(p \Leftrightarrow (q_1 \vee q_2))$	$((p_1 \wedge p_2) \vee (q_1 \wedge q_2))$	$(\neg((p_1 \wedge p_2) \vee (q_1 \wedge q_2)))$

4.4 Tautologies & contradiction

Tautology: A tautology is a proposition that is always true. Example: For a given atomic sentence, p and q , p ∨ ¬p

Contradiction: A contradiction is a proposition that is always false. Example: For a given atomic sentence, p and q, p ∧ ¬p

Contingency: Contingency is a proposition that is neither a tautology nor a contradiction. Example: For a given atomic sentence p and q, p ∨ q → ¬r

Validity : Validity is closely related to *logical truth*. A sentence is a logical truth if it is a tautology. A tautology is a statement which is true solely in virtue of its logical structure. Any logically valid argument can be recast as a logically true sentence.

Example

Consider the following argument:

- If it is raining, the road will be wet
- It is raining
- Therefore, the road is wet.

In symbolic logic this can be expressed in the following way:

- If X, then Y
- X
- Therefore, Y

Where 'X' is "it is raining" and 'Y' is "the road is wet", with the first two statements being the *premises* and the final statement the *conclusion*

Logical Validity considers only the *structure* of the argument, not if the argument is actually true. In the above argument, it is valid, because the structure of the argument is true. Another argument which has the same structure is the following:

- If there is creation there must be a creator
- There is creation
- Therefore, there is a creator.

The above argument is logically valid, again, meaning that the argument is true based only on its *structure*.

ARGUMENTS AND PROOFS

Definition: An argument is a sequence of statements called premises, plus a statement called the conclusion. A valid argument is an argument such that the conclusion is true whenever the premises are all true.

Note: An argument has the following form:

$(P1 \wedge P2 \wedge ... \wedge Pn) \rightarrow C$.

Example: Argument:

If it is Saturday, then it will rain.

It is Saturday.

Therefore it will rain.

Is this a valid argument? Let's come up with the general form.

P: It is Saturday.

Q: It will rain.

Premises: $P \rightarrow Q$, P

Conclusion: Q

Typically we write the argument like this:

$P \rightarrow Q$

P

$\therefore$ Q

Note: the symbol $\therefore$ stands for "therefore".

Note: We can see via a truth table that, for propositions P and Q, $((P \rightarrow Q) \wedge P) \rightarrow Q$ is a tautology: from Truth table:

Modus Ponens

This general form of argument, the rule of inference called modus ponens:

$P \rightarrow Q$

P

$\therefore$ Q is valid since whenever the premises are true, the conclusion is also true.

Example: If it snows, then the schools will be closed. It snows.

Therefore the schools are closed.

P: It snows.

Q: The schools will be closed.

If $P \rightarrow Q$ and P are true, then Q is true, by modus ponens.

Chapter 5 – Uncertain Knowledge and Reasoning

5.1 Introduction to Predicate Logic

The propositional logic is not powerful enough to represent all types of assertions that are used in computer science and mathematics, or to express certain types of relationship between propositions such as equivalence. For example, the statement "**x is greater than 1**", where x is a variable, is not a proposition because you cannot tell whether it is true or false unless you know the value of x. Thus the propositional logic cannot deal with such sentences.

Also the pattern involved in the following logical equivalences cannot be captured by the propositional logic:

"Not all birds fly" is equivalent to "Some birds don't fly".

"Not all integers are even" is equivalent to "Some integers are not even".

"Not all cars are expensive" is equivalent to "Some cars are not expensive",

Each of those propositions is treated independently of the others in propositional logic. For example, if P represents "Not all birds fly" and Q represents "Some integers are not even", then there is no mechanism in propositional logic to find out whether or not P is equivalent to Q. Hence to be used in inferencing, each of these equivalences must be listed individually rather than dealing with a general formula.

Hence we need more powerful logic to deal with these and other problems. The predicate logic is one of such logic and it addresses these issues among others. To cope with deficiencies of propositional logic we introduce two new features: predicates and quantifiers.

A **predicate** is a verb phrase template that describes a property of objects, or a relationship among objects represented by the variables. Sometimes it is also called as Predicate Calculus and First Order Logic. It shifts the focus from logical relations that holds between sentences to those that holds within sentences. A Predications consist of a Predicate and a set of arguments such as given below.

$$\textbf{Predicate (arg}_1\textbf{, ….arg}_n\textbf{)}$$

For example, the sentences "The car Tom is driving is blue", "The sky is blue", and "The

cover of this book is blue" come from the template "is blue" by placing an appropriate noun/noun phrase in front of it. The phrase **"is blue"** is a predicate and it describes the property of being blue. Predicates are often given a **name**. For example any of "is_blue", "Blue" or "B" can be used to represent the predicate "is blue" among others. If we adopt B as the name for the predicate "is_blue", sentences that assert an object is blue can be represented as "B(x)", where x represents an arbitrary object.

B (X) reads as “x is blue”.

Similarly the sentences "John gives the book to Mary", "Jim gives a loaf of bread to Tom", and "Jane gives a lecture to Mary" are obtained by substituting an appropriate object for variables *x*, *y*, and *z* in the sentence "*x* gives *y* to *z*". The template "... gives ... to ..." is a predicate and it describes a relationship among three objects. This predicate can be represented by Give(x, y, z) or G(x, y, z), for example.

Note: The sentence "John gives the book to Mary" can also be represented by another predicate such as "gives a book to". Thus if we use B(x, y) to denote this predicate, "John gives the book to Mary" becomes B(John, Mary). In that case, the other sentences, "Jim gives a loaf of bread to Tom", and "Jane gives a lecture to Mary", must be expressed with other predicates.

Examples

Daniel is a student at MTU.
What is the subject? What is the predicate?

Example 1: We can form two different predicates here.
Let P(x) be “x is a student at MTU”.
Let Q(x, y) be “x is a student at y”.

Definition: A predicate is a property that a variable or a finite collection of variables can have. A predicate becomes a proposition when specific values are assigned to the variables. P(x1, x2, ..., xn) is called a predicate of n variables or n arguments.
Example 2: She lives in the city.

P(x,y): x lives in y.

P(Ayantu, Teppi) is a proposition: Ayantu lives in Teppi.

Domains and Truth Sets

Definition: The domain or universe or universe of discourse for a predicate variable is the set of values that may be assigned to the variable.

Definition: If P(x) is a predicate and x has domain U, the truth set of P(x) is the set of all elements t of U such that P(t) is true,

Example: U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

P(x): “x is even”.

The truth set is: {2, 4, 6, 8, 10}

5.2 Quantification

It is a Logical expression or an expression, as “all” or “some,” that indicates the quantity of a proposition, especially a modifier, which indicates the quantity of something.

The Universal Quantifier: ∀

Turn predicates into propositions by assigning values to all variables:

Predicate P(x): “x is even”

Proposition P(6): “6 is even”

The other way to turn a predicate into a proposition: add a quantifier like “all” or “some” that indicates the number of values for which the predicate is true.

Definition: The symbol ∀ is called the universal quantifier. The universal quantification of P(x) is the statement “P(x) for all values x in the universe”, which is written in logical notation as:

∀xP(x)

Ways to read ∀xP(x):

For every x, P(x)

For every x, P(x) is true

For all x, P(x)

A predicate with variables is not a proposition. For example, the statement $x > 1$ with

variable *x* over the universe of real numbers is neither true nor false since we don't know what *x* is. It can be true or false depending on the value of *x*. For *x* > *1* to be a proposition either we substitute a specific number for *x* or change it to something like "There is a number *x* for which *x* > *1* holds", or "For every number *x*, *x* > *1* holds".

The Universal Quantifier

The expression: ∀*x P(x)*, denotes the **universal quantification** of the atomic formula P(x). Translated into the English language, the expression is understood as: "*For all x, P(x)* holds", "*for each x, P(x)* holds" or "*for every x, P(x)* holds". ∀ is called the **universal quantifier**, and ∀*x* means all the objects x in the universe. If this is followed by *P(x)* then the meaning is that *P(x)* is true for every object *x* in the universe. For example, "All cars have wheels" could be transformed into the propositional form, ∀ *x P(x)*, where:

- *P(x)* is the predicate denoting: ***x* has wheels**, and
- the universe of discourse is only populated by cars.

The Existential Quantifier

The expression: ∃ *xP(x)*, denotes the **existential quantification** of *P(x)*. Translated into the English language, the expression could also be understood as: "There exists an *x* such that *P(x)*" or "There is at least one *x* such that *P(x)*" ∃ is called the **existential quantifier**, and ∃ *x* means at least one object *x* in the universe. If this is followed by *P(x)* then the meaning is that *P(x)* is true for at least one object *x* of the universe. For example, "*Someone loves you*" could be transformed into the propositional form, ∃ *x P(x)*, where:

- *P(x)* is the predicate meaning: ***x* loves you**,
- The universe of discourse contains (but is not limited to) all living creatures.

How to read quantified formulas

When reading quantified formulas in English, **read them from left to right.** ∀***x*** can be read as "for every object ***x*** in the universe the following holds" and ∃ ***x*** can be read as "there exists an object ***x*** in the universe which satisfies the following" or "for some

object ***x*** in the universe the following holds". Those do not necessarily give us good English expressions. But they are where we can start. Get the correct reading first then polish your English without changing the truth values.

For example, let the universe be the set of airplanes and let ***F(x, y)*** denote "***x*** flies faster than ***y***". Then $\forall x \forall y$ ***F(x, y)*** can be translated initially as "For every airplane ***x*** the following holds: ***x*** is faster than every (any) airplane ***y***". In simpler English it means "Every airplane is faster than every airplane (including itself!)". $\forall x \exists$ ***y F(x, y)*** can be read initially as "For every airplane ***x*** the following holds: for some airplane ***y***, ***x*** is faster than ***y***".

In simpler English it means "Every airplane is faster than some airplane". $\exists$ ***x*** $\forall$ ***y F(x, y)*** represents "There exist an airplane ***x*** which satisfies the following: (or such that) for every airplane ***y***, ***x*** is faster than ***y***".

In simpler English it says "There is an airplane which is faster than every airplane" or "Some airplane is faster than every airplane". $\exists x \exists$ ***y F(x, y)*** reads "For some airplane ***x*** there exists an airplane ***y*** such that ***x*** is faster than ***y***", which means "Some airplane is faster than some airplane".

Order of Application of Quantifiers

When more than one variables are quantified in a wff such as $\exists$ ***y*** $\forall$ ***x P(x, y)***, they are applied from the inside, that is, the one closest to the atomic formula is applied first. Thus $\exists$ ***y*** $\forall$ ***x P(x, y)*** reads $\exists$ ***y [*** $\forall$ ***x P(x, y)]***, and we say "there exists an ***y*** such that for every ***x***, ***P(x, y)*** holds" or "for some ***y***, ***P(x, y)*** holds for every ***x***".

5.3 Inference in Predicate Logic

Predicate logic is more powerful than propositional logic. It allows one to reason about properties and relationships of individual objects. In predicate logic, one can use some **inference rules**.

The following four rules describe when and how the universal and existential quantifiers can be added to or deleted from an assertion.

Universal Instantiation:

Where *c* is some arbitrary element of the universe.

$\forall$ x P(X)

P(c)

Universal Generalization:

P(c)

$\forall$ x P(x)

Where *P(c)* holds for every element c of the universe of discourse.

Existential Instantiation:

$\exists$ x P(x)

P(c)

Where *c* is some element of the universe of discourse, It is not arbitrary but must be one for which P(c) is true.

Where *c* is some element of the universe of discourse. It is not arbitrary but must be one for which P(c) is true.

Existential Generalization:

P(c)

$\exists$ x P(x)

Where *c* is an element of the universe.

Negation of Quantified Statement

Another important inference rule is the following:

$\neg \exists$ x P(x) $\Leftrightarrow \forall$ x $\neg$ P(x)

This, for example, shows that if P(x) represents x is happy and the universe is the set of people, then "There does not exist a person who is happy" is equivalent to "Everyone is not happy". Thus the left side can be substituted for the right side whenever necessary in reasoning and vice versa.

Example:

As an example of inference using these rules, let us consider the following reasoning: **A cheque is void if it has not been cashed for 30 days. This cheque has not been cashed for 30 days. Therefore this cheque is void. You cannot cash a cheque which is void. Therefore you cannot cash this cheque. We now have a cheque which cannot be cashed.**

This can be put into symbolic form using the following predicates assuming the universe is the set of all objects:

C*(*x*):** ***x is a cheque.

T*(*x*):** ***x has been cashed within 30 days.

V*(*x*):** ***x is void.

S*(*x*):** ***x can be cashed.

***This_* cheque** represents a specific object in the universe which corresponds to "this cheque".

$\forall x [[C(x) \land \neg T(x)] \rightarrow V(x)]$

$\neg T(\ This\_cheque\ )$

$V(\ This\_cheque\ )$

$\forall x [[C(x) \land V(x)] \rightarrow \neg S(x)]$

$\neg S(\ This\_cheque)$

$\exists x [C(x) \land \neg S(x)]$

Here the reasoning proceeds as follows:

From $\forall x [[C(x) \land \neg T(x)] \rightarrow V(x)]$ by Universal Instantiation

$[[C(\ This\_cheque) \land \neg T(\ This\_cheque)] \rightarrow V(\ This\_cheque)]$

Since This_cheque is a cheque and $\neg T(\ This\_cheque)$,

$[C(\ This\_cheque) \land \neg T(\ This\_cheque)]$ holds.

Hence

$[[C(\ This\_cheque) \land \neg T(\ This\_cheque)] \rightarrow V(\ This\_cheque)]$

$[C(\ This\_cheque) \land \neg T(\ This\_cheque)]$

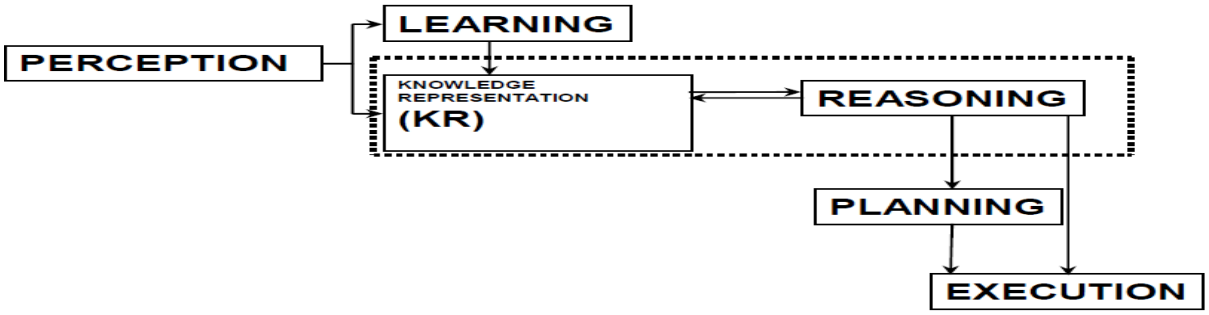
$V(\ This\_cheque)$

5.4 Knowledge Representation and Reasoning

Artificial Intelligence Cycle

Almost all AI systems have the following components in general:

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



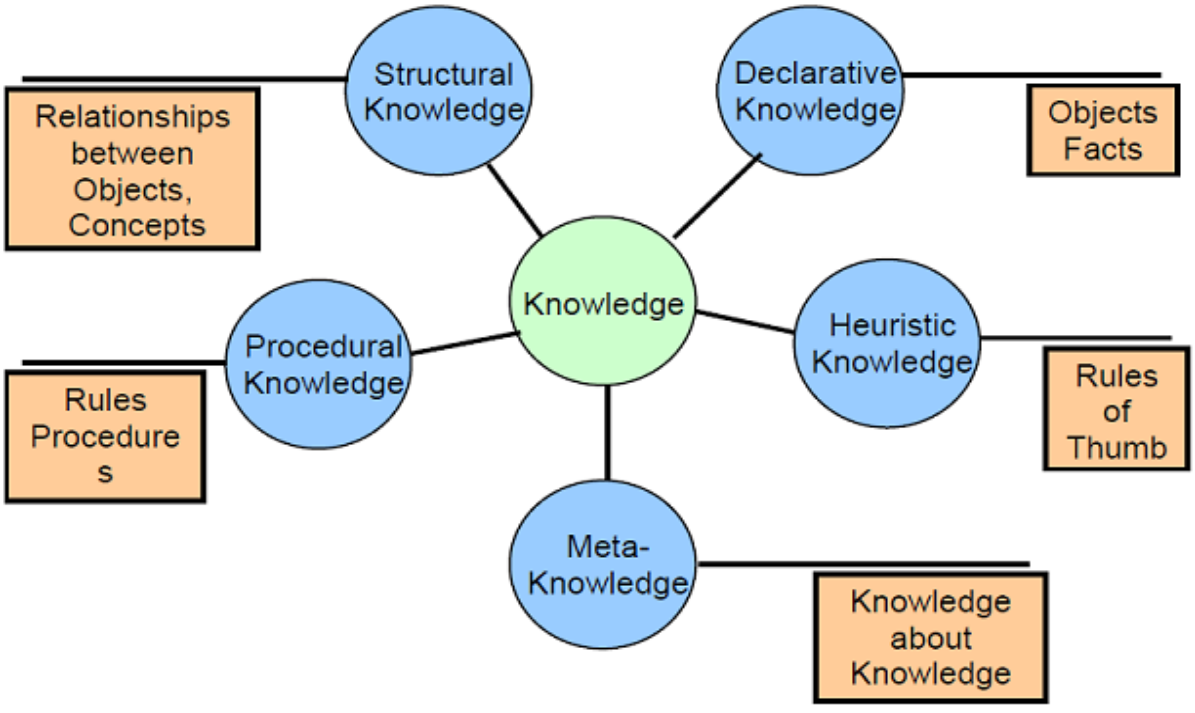
An AI system has a perception component that allows the system to get information from its environment. As with human perception, this may be visual, audio or other forms of sensory information. The system must then form a meaningful and useful representation of this information internally. This knowledge representation maybe static or it may be coupled with a learning component that is adaptive and draws trends from the perceived data.

Knowledge representation (KR) and reasoning are closely coupled components which are basically tied to each other. A representation scheme is not meaningful on its own; it must be useful and helpful in achieve certain tasks. The same information may be represented in many different ways, depending on how you want to use that information. For example, in mathematics, if we want to solve problems about ratios, we would most likely use algebra, but we could also use simple hand drawn symbols. To say half of something, you could use **0.5 x** or you could draw a picture of the object with half of it coloured differently. Both would convey the same information but the former is more compact and useful in complex scenarios where you want to perform reasoning on the information. It is important at this point to understand how knowledge representation and reasoning are interdependent components, and as AI system designer, you have to consider this relationship when coming up with any solution.

Knowledge and Its Type

Knowledge is referred as the "Understanding of a subject area". A well-focused subject area is referred to as a knowledge domain, for example, medical domain, engineering domain, business domain, etc. If we analyze the various types of knowledge we use in everyday life, we can broadly define knowledge to be one of the following categories:

- **Procedural knowledge**: Describes how to do things, provides a set of directions of how to perform certain tasks, e.g., how to drive a car.
- **Declarative knowledge**: It describes objects, rather than processes. What is known about a situation, e.g. it is sunny today, and cherries are red.
- **Meta knowledge**: Knowledge about knowledge, e.g., the knowledge that blood pressure is more important for diagnosing a medical condition than eye color.
- **Heuristic knowledge**: Rule-of-thumb, e.g. if I start seeing shops, I am close to the market.
    - Heuristic knowledge is sometimes called shallow knowledge.
    - Heuristic knowledge is empirical as opposed to deterministic
- **Structural knowledge**: Describes structures and their relationships. e.g. how the various parts of the car fit together to make a car, or knowledge structures in terms of concepts, sub concepts, and objects.



5.5 Reasoning

As we have learned knowledge representation, we will look at mechanisms to reason on the knowledge once we have represented it using some logical scheme. Reasoning is the process of deriving logical conclusions from given facts. Reasoning is referred as "the process of working with knowledge, facts and problem solving strategies to draw conclusions".

Types of Reasoning

- **Deductive Reasoning**

  Deductive reasoning, as the name implies, is based on deducing new information from logically related known information. A deductive argument offers assertions that lead automatically to a conclusion.

  Example: If there is dry wood, oxygen and a spark, there will be a fire.

  Given: There is dry wood, oxygen and a spark

  We can deduce: There will be a fire.

- **Inductive Reasoning**

  Inductive reasoning is based on forming, or inducing a 'generalization' from a limited set of observations. Example:

  Observation: All the crows that I have seen in my life are black.

  Conclusion: All crows are black.

  Thus the essential difference between deductive and inductive reasoning is that inductive reasoning is based on experience while deductive reasoning is based on rules, hence the latter will always be correct.

- **Abductive Reasoning**

  Deduction is exact in the sense that deductions follow in a logically provable way from the axioms. Abduction is a form of deduction that allows for plausible (reasonable) inference, i.e. the conclusion might be wrong, Example:

  Implication: She carries an umbrella if it is raining

  Axiom: she is carrying an umbrella

  Conclusion: It is raining

  This conclusion might be false, because there could be other reasons that she is carrying an umbrella, for instance she might be carrying it to protect herself from the sun.

- **Analogical Reasoning**

  Analogical reasoning works by drawing analogies between two situations, looking for similarities and differences, e.g. when you say driving a truck is just like driving a car, by analogy you know that there are some similarities in the driving mechanism, but you also know that there are certain other distinct characteristics of each.

- **Common-Sense Reasoning**

  Common-sense reasoning is an informal form of reasoning that uses rules gained through experience or what we call rules-of-thumb. It operates on heuristic knowledge and heuristic rules.

- **Non-Monotonic Reasoning**

  Non-Monotonic reasoning is used when the facts of the case are likely to change after some time, Example:

  Rule: IF the wind blows

  THEN the curtains sway

  When the wind stops blowing, the curtains should sway no longer. However, if we use monotonic reasoning, this would not happen. The fact that the curtains are swaying would be retained even after the wind stopped blowing. In non monotonic reasoning, we have a "truth maintenance system". It keeps track of what caused a fact to become true. If the cause is removed, that fact is removed (retracted) also.

5.6 Knowledge Based System

A knowledge-based system (KBS) is a computer system which generates and utilizes knowledge from different sources, data and information. These systems aid in solving problems, especially complex ones, by utilizing artificial intelligence concepts. These systems are mostly used in problem-solving procedures and to support human learning, decision making and actions.

Knowledge-based systems are considered to be a major branch of artificial intelligence. They are capable of making decisions based on the knowledge residing in them, and can understand the context of the data that is being processed.

Knowledge-based systems broadly consist of an interface engine and knowledge base.

The interface engine acts as the search engine, and the knowledge base acts as the knowledge repository. Learning is an essential component of knowledge-based systems and simulation of learning helps in the betterment of the systems. Knowledge-based systems can be broadly classified as CASE-based systems, intelligent tutoring systems, expert systems, hypertext manipulation systems and databases with intelligent user interface.

Compared to traditional computer-based information systems, knowledge-based systems have many advantages. They can provide efficient documentation and also handle large amounts of unstructured data in an intelligent fashion. Knowledge-based systems can aid in expert decision making and allow users to work at a higher level of expertise and promote productivity and consistency. These systems are considered very useful when expertise is unavailable, or when data needs to be stored for future usage or needs to be grouped with different expertise at a common platform, thus providing large-scale integration of knowledge. Finally, knowledge-based systems are capable of creating new knowledge by referring to the stored content.

5.7 Facts

Facts are a basic block of knowledge (the atomic units of knowledge). They represent declarative knowledge (they declare knowledge about objects). A proposition is the statement of a fact. Each proposition has an associated truth value. It may be either true or false. In AI, to represent a fact, we use a proposition and its associated truth value, for example:

Proposition A: It is raining

Proposition B: I have an umbrella

Proposition C: I will go to school

Types of Facts

Single Valued or Multiple Valued

Facts may be single-valued or multi-valued, where each fact (attribute) can take one or more than one values at the same time, for example an individual can only have one eye color, but may have many cars. So the value of attribute cars may contain more than one value.

Uncertain Facts

Sometimes we need to represent uncertain information in facts. These facts are called uncertain facts, e.g. it will probably be sunny today. We may chose to store numerical certainty values with such facts that tell us how much uncertainty there is in the fact.

Fuzzy Facts

Fuzzy facts are ambiguous in nature, e.g. the book is heavy/light. Here it is unclear what heavy means because it is a subjective description. Fuzzy representation is used for such facts. While defining fuzzy facts, we use certainty factor values to specify value of "truth". We will look at fuzzy representation in more detail later.

Object Attribute Value Triplets

Object-Attribute Value Triplets or OAV triplets are a type of fact composed of three parts; object, attribute and value. Such facts are used to assert a particular property of some object.

Example 1: Khalid's Eye Color is Brown.

Object: Khalid

Attribute: Eye Color

Value: Brown

Example 2: Khalid's daughter is Sara.

Object: Khalid

Attribute: Daughter

Value: Sara

5.8 Quantifying Uncertainty

Let's consider an example of uncertain reasoning as diagnosing a dental patient's toothache.

Diagnosis—whether for medicine, automobile repair, or whatever—almost always involves uncertainty. Let us try to write rules for dental diagnosis using propositional logic, so that we can see how the logical approach breaks down.

Consider the following simple rule:

$$\textbf{Toothache} \Rightarrow \textbf{Cavity}$$

The problem is that this rule is wrong. Not all patients with toothaches have cavities; some of them have gum disease, a sore, or one of several other problems:

$$\textbf{Toothache} \Rightarrow \textbf{Cavity} \vee \textbf{Gum Problem} \vee \textbf{Sore} \ldots$$

Unfortunately, in order to make the rule true, we have to add an almost unlimited list of possible problems. We could try turning the rule into a causal rule:

Cavity ⇒ Toothache

But this rule is not right either; not all cavities cause pain. The only way to fix the rule is to make it logically exhaustive: to augment the left-hand side with all the qualifications required for a cavity to cause a toothache. The connection between toothaches and cavities is just not a logical consequence in either direction. This is typical of the medical domain, as well as most other judgmental domains: law, business, design, automobile repair, gardening, dating, and so on. The agent's knowledge can at best provide only a degree of belief in the relevant sentences. Our main tool for dealing with degrees of belief is ***probability theory.***

5.9 Probabilistic Reasoning

Probabilistic reasoning is the concept of using logic and probability to handle uncertain situations. An example of Probabilistic Reasoning is using past situations and statistics to predict an outcome.

Bayes Theorem

One of the most significant developments in the probability field has been the development of Bayesian decision theory which has proved to be of immense help in making decisions under uncertain conditions. The Bayes Theorem was developed by a British Mathematician Rev. Thomas Bayes. The probability given under Bayes theorem is also known by the name of inverse probability, posterior probability or revised probability. This theorem finds the probability of an event by considering the given sample information; hence the name posterior probability.

Bayes theorem describes the probability of an event based on other information that might be relevant. Essentially, you are estimating a probability, but then updating that estimate based on other things that you know. This is something that you already do every day in real life. For instance, if your friend is supposed to pick you up to go out to dinner, you might have a mental estimate of if she will be on time, be 15 minutes late, or be a half hour late. That would be your starting probability. Bayes theorem is a formal way of doing that.

The equation for Bayes theorem is:

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

Where,

A & B are events.

P(A) and P(B) are the probabilities of A and B without regard for each other.

P(A|B) is the conditional probability, the probability of A given that B is true.

P(B|A) is the probability of B given that A is true.

To solve Bayes theorem we should follow some steps as:

1) Determine what we want the probability of, and what we are observing
2) Estimate initial probabilities for all of the possible answers
3) For each of the initial possible answers, assume it is true and calculate the probability of getting our observation with that possibility being true
4) Multiply the initial probabilities (Step 2) by the probabilities based on our observation (Step 3) for each of the initial possible answers
5) Normalize the results (divide the each probability by the sum of the total probabilities so that the new total probability is 1)
6) Repeat Steps 2-5 over and over for each new observation

Example 1:

Suppose that a test for using a particular drug is 99% sensitive and 99% specific. That is, the test will produce 99% true positive results for drug users and 99% true negative results for non-drug users. Suppose that 0.5% of people are users of the drug. What is the probability that a randomly selected individual with a positive test is a drug user?

$$\begin{aligned} P(\text{User} \mid +) &= \frac{P(+ \mid \text{User})P(\text{User})}{P(+)} \\ &= \frac{P(+ \mid \text{User})P(\text{User})}{P(+ \mid \text{User})P(\text{User}) + P(+ \mid \text{Non-user})P(\text{Non-user})} \\ &= \frac{0.99 \times 0.005}{0.99 \times 0.005 + 0.01 \times 0.995} \\ &\approx 33.2\% \end{aligned}$$

Even if an individual tests positive, it is more likely that they do not use the drug than that they do. This is because the number of non-users is large compared to the number of users. The number of false positives outweighs the number of true positives. For example, if 1000 individuals are tested, there are expected to be 995 non-users and 5 users. From the 995 non-users, 0.01 × 995 ≃ 10 false positives are expected. From the 5 users, 0.99 × 5 ≈ 5 true positives are expected. Out of 15 positive results, only 5 are genuine.

The importance of specificity in this example can be seen by calculating that even if sensitivity is raised to 100% and specificity remains at 99% then the probability of the person being a drug user only rises from 33.2% to 33.4%, but if the sensitivity is held at 99% and the specificity is increased to 99.5% then the probability of the person being a drug user rises to about 49.9%.

Example 2:

The entire output of a factory is produced on three machines. The three machines account for 20%, 30%, and 50% of the factory output. The fraction of defective items produced is 5% for the first machine; 3% for the second machine; and 1% for the third machine. If an item is chosen at random from the total output and is found to be defective, what is the probability that it was produced by the third machine?

Once again, the answer can be reached without recourse to the formula by applying the conditions to any hypothetical number of cases. For example, if 100,000 items are produced by the factory, 20,000 will be produced by Machine A, 30,000 by Machine B, and 50,000 by Machine C. Machine A will produce 1000 defective items, Machine B 900, and Machine C 500. Of the total 2400 defective items, only 500, or 5/24 were produced by Machine C.

A solution is as follows. Let X_i denote the event that a randomly chosen item was made by the i^{th} machine (for i = A,B,C). Let Y denote the event that a randomly chosen item is defective. Then, we are given the following information:

$$P(X_A) = 0.2, \quad P(X_B) = 0.3, \quad P(X_C) = 0.5.$$

If the item was made by the first machine, then the probability that it is defective is 0.05; that is, $P(Y \mid X_A) = 0.05$. Overall, we have

$$P(Y \mid X_A) = 0.05, \quad P(Y \mid X_B) = 0.03, \quad P(Y \mid X_C) = 0.01.$$

To answer the original question, we first find *P*(Y). That can be done in the following way:

$$P(Y) = \sum_i P(Y \mid X_i)P(X_i) = (0.05)(0.2) + (0.03)(0.3) + (0.01)(0.5) = 0.024.$$

Hence 2.4% of the total output of the factory is defective.

We are given that *Y* has occurred, and we want to calculate the conditional probability of X_C. By Bayes' theorem,

$$P(X_C \mid Y) = \frac{P(Y \mid X_C)P(X_C)}{P(Y)} = \frac{0.01 \cdot 0.50}{0.024} = \frac{5}{24}$$

Given that the item is defective, the probability that it was made by the third machine is only 5/24. Although machine C produces half of the total output, it produces a much smaller fraction of the defective items. Hence the knowledge that the item selected was defective enables us to replace the prior probability $P(X_C) = 1/2$ by the smaller posterior probability $P(X_C \mid Y) = 5/24$.

Probabilistic Reasoning over Time

In the Bayesian theorem we have seen static situations where each random variable gets a single fixed value in a single problem. Now we consider the problem of describing probabilistic environments that evolve over time. Examples: Robot localization, tracking speech etc.

Hidden Markov Model

The hidden Markov model can be represented as the simplest dynamic Bayesian network. The current state is conditionally independent of all the other states given the state in the previous time step.

The Hidden Markov Model (HMM) provides a framework for modelling daily rainfall occurrences and amounts on multi-site rainfall networks. The HMM fits a model to observed rainfall records by introducing a small number of discrete rainfall states. These states allow a diagnostic interpretation of observed rainfall variability in terms of a few rainfall patterns. They are not directly observable, or '*hidden*' from the observer.

The time sequence of which state is active on each day follows a Markov chain. Thus, the state which is active '*today*' depends only on the state which was active '*yesterday*' according to transition probabilities.

The HMM also allows you to *simulate* rainfall at each of the station locations, such that key statistical properties (eg. rainfall probabilities, dry/wet spell lengths) of the simulated rainfall match those of the observed rainfall records. This can be useful for generating large numbers of synthetic realizations of rainfall for input into statistical analysis, or input into a crop simulation model.

5.10 Making Simple Decision

- In many cases, our knowledge of the world is incomplete (not enough information) or uncertain (sensors are unreliable).
- Often, rules about the domain are incomplete or even incorrect - in the qualification problem, for example, what are the preconditions for an action?
- We have to act in spite of this!
- Drawing conclusions under uncertainty.

Example:

- Goal: Be in Campus at 8:20 AM to give a lecture.
- There are several plans that achieve the goal:
  - P1: Get up at 7:00, take the bus at 7:30 AM, arrive at 8:00.
  - P2: Get up at 7:30 AM, walk to campus and arrive at 8:00.
- All these plans are correct, but
  - They imply different costs and different probabilities of actually achieving the goal.
  - P2 eventually is the plan of choice, since giving a lecture is very important, and the success rate of P1 is only 90-95%.

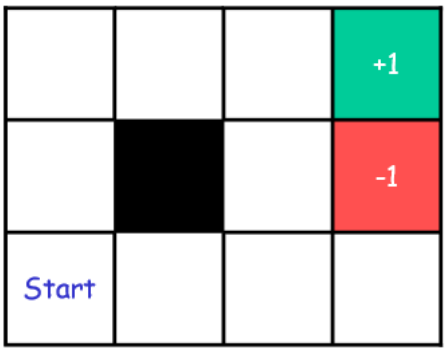
Uncertainty and Rational Decisions

- We have a choice of actions (or plans).
- These can lead to different solutions with different probabilities.
- The actions have different (subjective) costs.
- The results have different (subjective) utilities.
- It would be rational to choose the action with the maximum expected total utility!

Decision Theory = Utility Theory + Probability Theory

5.11 Making Complex Decisions

- The agent's utility now depends on a sequence of decisions.
- In the following 4 × 3 grid environment the agent makes a decision to move (U, R, D, L) at each time step.
- When the agent reaches one of the goal states, it terminates.
- The environment is fully observable — the agent always knows where it is



- If the environment were deterministic, a solution would be easy: the agent will always reach +1 with moves [U, U, R, R, R]
- Because actions are unreliable, a sequence of moves will not always lead to the desired outcome
- Let each action achieve the intended effect with probability 0.8 but with probability 0.1 the action moves the agent to either of the right angles to the intended direction
- If the agent bumps into a wall, it stays in the same square
- Now the sequence [U, U, R, R, R] leads to the goal state with probability 0.85 = 0.32768
- In addition, the agent has a small chance of reaching the goal by accident going the other way around the obstacle with a probability 0.14 × 0.8, for a grand total of 0.32776
- A transition model specifies outcome probabilities for each action in each possible state
- Let P(s' | s, a) denote the probability of reaching state s' if action a is done in state s
- The transitions are Markovian in the sense that the probability of reaching s'

depends only on s and not the earlier states

- We still need to specify the utility function for the agent
- The decision problem is sequential, so the utility function depends on a sequence of states — an environment history — rather than on a single state
- For now, we will simply stipulate that is each state s, the agent receives a reward R(s), which may be positive or negative
- For our particular example, the reward is -0.04 in all states except in the terminal states
- The utility of an environment history is just (for now) the sum of rewards received
- If the agent reaches the state +1, e.g., after ten steps, its total utility will be 0.6
- The small negative reward gives the agent an incentive to reach [4, 3] quickly
- A sequential decision problem for a fully observable environment with
  - A Markovian transition model and
  - Additive rewards

  is called a Markov decision problem (MDP)

Chapter 6 – Learning

6.1 Introduction to Learning

Learning can be described as normally a relatively permanent change that occurs in behavior as a result of experience. Learning occurs in various regimes. For example, it is possible to learn to open a lock as a result of trial and error; possible to learn how to use a word processor as a result of following particular instructions.

Once the internal model of what ought to happen is set, it is possible to learn by practicing the skill until the performance converges on the desired model. One begins by paying attention to what needs to be done, but with more practice, one will need to monitor only the trickier parts of the performance. Automatic performance of some skills by the brain points out that the brain is capable of doing things in parallel i.e. one part is devoted to the skill whilst another part mediates conscious experience. The definition of Learning is as follow:

- "Learning denotes changes in a system that ... enables a system to do the same task more efficiently the next time." By Herbert Simon.
- "Learning is constructing or modifying representations of what is being experienced." By Ryszard Michalski.
- "Learning is making useful changes in our minds." By Marvin Minsky.

What is Machine Learning

The goal of machine learning is to build computer systems that can learn from their experience and adapt to their environments. Obviously, learning is an important aspect or component of intelligence. There are both theoretical and practical reasons to support such a claim. Some people even think intelligence is nothing but the ability to learn, though other people think an intelligent system has a separate "learning mechanism" which improves the performance of other mechanisms of the system.

In simple words, we can say that machine learning is the competency of the software to perform a single or series of tasks intelligently without being programmed for those activities. This is part of Artificial Intelligence**.** Normally, the software behaves the way the programmer programmed it; while machine learning is going one step further by making the software capable of accomplishing intended tasks by using statistical analysis and predictive analytics techniques.

Why do we want Machine Learning?

One response to the idea of AI is to say that computers can not think because they only do what their programmers tell them to do. However, it is not always easy to tell what a particular program will do, but given the same inputs and conditions it will always produce the same outputs. If the program gets something right once it will always get it right. If it makes a mistake once it will always make the same mistake every time it runs. In contrast to computers, humans learn from their mistakes; attempt to work out why things went wrong and try alternative solutions. Also, we are able to notice similarities between things, and therefore can generate new ideas about the world we live in. Any intelligence, however artificial or unknown, that did not learn would not be much of intelligence. So, machine learning is a prerequisite for any mature programme of artificial intelligence.

Three Phases of Machine Learning

- Training: a training set of examples of correct behavior is analyzed and some representation of the newly learnt knowledge is stored. This is often some form of rules.
- Validation: the rules are checked and, if necessary, additional training is given. Sometimes additional test data are used, but instead of using a human to validate the rules, some other automatic knowledge based component may be used. The role of tester is often called the critic.
- Application: the rules are used in responding to some new situations.

These phases may not be distinct. For example, there may not be an explicit validation phase; instead, the learning algorithm guarantees some form of correctness. Also in some circumstances, systems learn "on the job", that is, the training and application phases overlap.

How does Machine Learning Works?

Machine Learning is a technique which works intelligently by using some complex algorithms and set of predefined rules. It uses the past data to read the patterns and then based on the analysis it generates the relevant data or performs the intended task abiding the defined rules and algorithms.

For example, whenever we typed in something on the search bar, the search engines

uses this machine learning technique to display the related contents. It intelligently reads the vast data on the web, indexes, ranking, and based on the defined rules and algorithm it displays the search results. Let's discuss the basic steps involved in machine learning techniques.

- The first step is to gathering the past data from the various sources such as excel files, text files or any other provider. The quality of related data is the foundation of learning for the software, the more relevant the data, the better learning for the software.
- The next step is data preparation. The programmer spends time on preparing the quality data and fixing the data issues. The missing data should be rectified to improve the data quality.
- Develop the data model with appropriate algorithms. The models are developed and tested with relevant algorithms which enable the software to read the correct data sets intelligently in the real world.
- Next step is to testing the model's accuracy using the data sets used while developing the models and also the data sets which were not used while developing the models. It shows the model's performance and accuracy with the optimum level of precisions.
- Finally, check and improve the performance by using various data sets which were not used earlier. That shows the software's capabilities to read new data sets based on the rules defined in the programmed algorithms.

6.2 Learning Techniques

Rote Learning

In this kind of learning there is no prior knowledge. When a computer stores a piece of data, it is performing an elementary form of learning. This act of storage presumably allows the program to perform better in the future. Examples of correct behavior are stored and when a new situation arises it is matched with the learnt examples. The values are stored so that they are not re-computed later. One of the earliest game-playing programs is checkers program. This program learned to play checkers well enough to beat its creator/designer.

Deductive Learning

Deductive learning works on existing facts and knowledge and deduces new knowledge

from the old. This is best illustrated by giving an example. For example, assume:

A = B

B = C

Then we can deduce with much confidence that:

C = A

Arguably, deductive learning does not generate "new" knowledge at all, it simply memorizes the logical consequences of what is known already. This implies that virtually all mathematical research would not be classified as learning "new" things. However, regardless of whether this is termed as new knowledge or not, it certainly makes the reasoning system more efficient.

Inductive Learning

Inductive learning takes examples and generalizes rather than starting with existing knowledge. For example, having seen many cats, all of which have tails, one might conclude that all cats have tails. This is an unsound step of reasoning but it would be impossible to function without using induction to some extent. In many areas it is an explicit assumption. There is scope of error in inductive reasoning, but still it is a useful technique that has been used as the basis of several successful systems.

One major subclass of inductive learning is concept learning. This takes examples of a concept and tries to build a general description of the concept. Very often, the examples sare described using attribute-value pairs. The example of inductive learning given here is that of a fish. Look at the table below:

	Cat	**Dog**	**Whale**	**Cow**
Swims	No	No	Yes	No
Has Lungs	Yes	Yes	Yes	Yes
Is a Fish	No	No	No	No

In the above example, there are various ways of generalizing from examples of fish and non-fish. The simplest description can be that a fish is something that does not have lungs. No other single attribute would serve to differentiate the fish.

6.3 Applied Learning

Solving Real World Problems with Learning

We do not yet know how to make computers learn nearly as well as people learn. However, algorithms have been developed that are effective for certain types of learning tasks, and many significant commercial applications have begun to appear. For problems such as speech recognition, algorithms based on machine learning outperform all other approaches that have been attempted to date. In other emergent fields like computer vision and data mining, machine learning algorithms are being used to recognize faces and to extract valuable information and knowledge from large commercial databases respectively. Some of the applications that use learning algorithms include:

- Spoken digits and word recognition
- Handwriting recognition
- Driving autonomous vehicles
- Path finders
- Intelligent homes
- Intrusion detectors
- Intelligent refrigerators, tvs, vacuum cleaners
- Computer games
- Humanoid robotics

This is just the glimpse of the applications that use some intelligent learning components. The current era has applied learning in the domains ranging from agriculture to astronomy to medical sciences.

General Model of Learning Agent (Pattern Recognition)

Any given learning problem is primarily composed of three things:

- Input
- Processing unit
- Output

The input is composed of examples that help the learner learn the underlying problem concept. Suppose we were to build the learner for recognizing spoken digits. We would ask some of our friends to record their sounds for each digit [0 to 9]. Positive examples of digit ‘1’ would be the spoken digit ‘1’, by the speakers. Negative examples for digit ‘1’

would be all the rest of the digits. For our learner to learn the digit '1', it would need positive and negative examples of digit '1' in order to truly learn the difference between digits '1' and the rest. The processing unit is the learning agent in our focus of study.

6.4 Symbol Based Learning

Ours is a world of symbols. We use symbolic interpretations to understand the world around us. For instance, if we saw a ship, and were to tell a friend about its size, we will not say that we saw a 254.756 meters long ship, instead we'd say that we saw a 'huge' ship about the size of 'Eiffel tower'. And our friend would understand the relationship between the size of the ship and its hugeness with the analogies of the symbolic information associated with the two words used: 'huge' and 'Eiffel tower'.

Similarly, the techniques we are to learn now use symbols to represent knowledge and information. Let us consider a small example to help us see where we're headed. What if we were to learn the concept of a GOOD STUDENT? We would need to define, first of all some attributes of a student, on the basis of which we could tell apart the good student from the average. Then we would require some examples of good students and average students. To keep the problem simple we can label all the students who are "not good" (average, below average, satisfactory, bad) as NOT GOOD STUDENT. Let's say we choose two attributes to define a student: grade and class participation. Both the attributes can have either of the two values: High, Low. Our learner program will require some examples from the concept of a student, for instance:

1. Student (GOOD STUDENT): Grade (High) ^ Class Participation (High)

2. Student (GOOD STUDENT): Grade (High) ^ Class Participation (Low)

3. Student (NOT GOOD STUDENT): Grade (Low) ^ Class Participation (High)

4. Student (NOT GOOD STUDENT): Grade (Low) ^ Class Participation (Low)

As you can see the system is composed of symbolic information, based on which the learner can even generalize that a student is a GOOD STUDENT if his/her grade is high, even if the class participation is low:

Student (GOOD STUDENT): Grade (High) ^ Class Participation (?)

This is the final rule that the learner has learnt from the enumerated examples. Here the '?' means that the attribute class participation can have any value, as long as the grade is high.

6.5 Problem and Problem Spaces

In theoretical computer science there are two main branches of problems: **Tractable and Intractable.** Those problems that can be solved in polynomial time are termed as tractable; the other half is called intractable. The tractable problems are further divided into structured and complex problems. Structured problems are those which have defined steps through which the solution to the problem is reached. Complex problems usually don't have well-defined steps. Machine learning algorithms are particularly more useful in solving the complex problems like recognition of patterns in images or speech, for which it's hard to come up with procedural algorithms. The solution to any problem is a function that converts its inputs to corresponding outputs. To understand this concept let us discuss this example:

Let us consider the domain of HEALTH. The problem in this case is to distinguish between a sick and a healthy person. Suppose we have some domain knowledge; keeping a simplistic approach, we say that two attributes are necessary and sufficient to declare a person as healthy or sick. These two attributes are: Temperature (T) and Blood Pressure (BP). Any patient coming into the hospital can have three values for T and BP: High (H), Normal (N) and Low (L). Based on these values, the person is to be classified as Sick (SK). SK is a Boolean concept, SK = 1 means the person is sick and SK = 0 means person is healthy. So the concept to be learnt by the system is of Sick, i.e., SK=1.

Instance Space

How many distinct instances can the concept sick have? Since there are two attributes: T and BP, each having 3 values, there can be a total of 9 possible distinct instances in all. If we were to list these, we'll get the following table:

X	T	BP	SK
X1	L	L	-
X2	L	N	-
X3	L	H	-
X4	N	L	-
X5	N	N	-
X6	N	H	-
X7	H	L	-
X8	H	N	-
X9	H	H	-

This is the entire instance space, denoted by X, and the individual instances are denoted by xi. |X| gives us the size of the instance space, which in this case are 9.

|X| = 9

The set X is the entire data possibly available for any concept. However, sometimes in real world problems, we don't have the liberty to have access to the entire set X, instead we have a subset of X, known as training data, denoted by D, available to us, on the basis of which we make our learner learn the concept.

Concept Space

A concept is the representation of the problem with respect to the given attributes, for example, if we're talking about the problem scenario of concept SICK defined over the attributes T and BP, then the concept space is defined by all the combinations of values of SK for every instance x. One of the possible concepts for the concept SICK might be listed in the following table:

X	T	BP	SK
X1	L	L	0
X2	L	N	0
X3	L	H	1
X4	N	L	0
X5	N	N	0
X6	N	H	1
X7	H	L	1
X8	H	N	1
X9	H	H	1

But there are a lot of other possibilities besides this one. The question is: how many total concepts can be generated out of this given situation. The answer is: $2^{|X|}$.

Hypothesis Space

In this space the learner has to apply some hypothesis, which has either a search or the language bias to reduce the size of the concept space. This reduced concept space becomes the hypothesis space. For example, the most common language bias is that the hypothesis space uses the conjunctions (AND) of the attributes, i.e. H = <T, BP>

H is the denotive representation of the hypothesis space; here it is the conjunction of attribute T and BP. If written in English it would mean:

H = <T, BP>:

IF "Temperature" = T AND "Blood Pressure" = BP

THEN

H = 1

ELSE

H = 0

Version Space and Searching

Version space is a set of all the hypotheses that are consistent with all the training examples. When we are given a set of training examples D, it is possible that there might be more than one hypotheses from the hypothesis space that are consistent with all the training examples. By consistent we mean h(xi) = C(xi). That is, if the true output of a concept [c(xi)] is 1 or 0 for an instance, then the output by our hypothesis [h(xi)] is 1 or 0 as well, respectively. If this is true for every instance in our training set D, we can say that the hypothesis is consistent.

X	T	BP	SK
X1	H	H	1
X2	L	L	0
X3	N	N	0

6.6 Introduction to Neural Network

In the area of Computer Science a Neural Network is a system of programs and data structures that approximates the operation of the human brain. A neural network usually involves a large number of processors operating in parallel, each with its own small sphere of knowledge and access to data in its local memory. Typically, a neural network is initially "trained" or fed large amounts of data and rules about data relationships (for example, "A grandfather is older than a person's father"). A program can then tell the network how to behave in response to an external stimulus (for example, to input from a computer user who is interacting with the network) or can initiate activity on its own (within the limits of its access to the external world).

In making determinations, neural networks use several principles, including gradient-based training, fuzzy logic, genetic algorithms, and Bayesian methods. Neural networks are sometimes described in terms of knowledge layers, with, in general, more complex networks having deeper layers. Current applications of neural networks include: oil exploration data analysis, weather prediction, the interpretation of nucleotide sequences in biology labs, and the exploration of models of thinking and consciousness.

The term neural network was traditionally used to refer to a network or circuit of biological neurons. The modern usage of the term often refers to artificial neural networks, which are composed of artificial neurons or nodes. Thus the term has two distinct usages:

- ***Biological neural networks*** are made up of real biological neurons that are connected or functionally related in a nervous system. In the field of neuro-science, they are often identified as groups of neurons that perform a specific physiological function in laboratory analysis.
- ***Artificial neural networks*** are composed of interconnecting artificial neurons (programming constructs that mimic the properties of biological neurons). Artificial neural networks may either be used to gain an understanding of biological neural networks, or for solving artificial intelligence problems without necessarily creating a model of a real biological system. The real, biological nervous system is highly complex: artificial neural network algorithms attempt to abstract this complexity and focus on what may hypothetically matter most from an information processing point of view. Good performance (e.g. as measured by good predictive ability, low generalization error), or performance mimicking animal or human error patterns, can then be used as one source of evidence towards supporting the hypothesis that the abstraction really captured something important from the point of view of information processing in the brain. Another incentive for these abstractions is to reduce the amount of computation required to simulate artificial neural networks, so as to allow one to experiment with larger networks and train them on larger data sets.

Comparision between Human and Brain

	Biological Neural Networks	Computers
Speed	Fast (nanoseconds)	Slow (milliseconds)
Processing	Superior (massively parallel)	Inferior (Sequential mode)
Size & Complexity	10^{11} neurons, 10^{15} interconnections	Far few processing elements
Storage	Adaptable, interconnection strengths	Strictly replaceable
Fault tolerance	Extremely Fault tolerant	Inherently non fault tolerant
Control mechanism	Distributive control	Central control

While this clearly shows that the human information processing system is superior to conventional computers, but still it is possible to realize an artificial neural network which exhibits the above mentioned properties.

Directions and Classifications of Neural Network

The Neural Networks can roughly be classified into four different types of orientations which includes its applications to real-world problems.

- In **Cognitive science/Artificial intelligence**, the interest is in modeling intelligent behavior. The interest in neural networks is mostly to overcome the problems and pitfalls of classical and symbolic methods of modeling intelligence.
- **Neurobiological modeling** has the goal to develop models of biological neurons. Here, the exact properties of the neurons play an essential role. In most of these models there is a level of activation of an individual neuron.
- **Scientific modeling** uses neural networks as modeling tools. In physics, psychology, and sociology neural networks have been successfully applied.
- **Computer science** views neural networks as an interesting class of algorithms that has properties like noise and fault tolerance, and generalization ability– that make them suited for application to real-world problems.

Applications of Neural Network

Neural Networks are now applied in many areas of science. Here are some few examples:

- **Optimization*:*** Neural networks have been applied to almost many kind of optimization problem. Conversely, neural network learning can often be conceived as an optimization problem in which it will minimize a kind of error function.
- **Control:** Many complex control problems have been solved by neural networks. They are especially popular for robot control. Such as autonomous robots - like humanoids, that have to operate in real world environments that are characterized by higher level rapid change. Since biological neural networks have evolved for precisely these kinds of conditions, they are well suited for such types of tasks. Also, because in the real world generalization is crucial, neural networks is often the tool of choice for systems, in particular robots, having interacted with physical environments.

- **Signal Processing*:*** Neural networks have been used to distinguish mines from rocks using sonar signals, to detect sun eruptions, and to process speech signals. Speech processing techniques and statistical approaches involving hidden models are sometimes combined.
- **Pattern Recognition*:*** Neural networks have been widely used for pattern recognition purposes, from face recognition, to recognition of tumors in various types of scans, to identification of plastic explosives in luggage of aircraft passengers (which yield a particular gamma radiation patterns when subjected to a stream of thermal neurons), to recognition of hand-written zip-codes.
- **Stock market prediction:** The dream of every mathematician is to develop methods for predicting the development of stock prices. Neural networks, in combination with other methods, are often used in this area. However, it is an open question whether they have been really successful.

Learning Rule

Weights are modified by learning rules. The learning rules determine how experiences of a network use their influence on its future behavior. There are basically three types of learning rules.

- **Supervised Learning Rule:** The term supervised is used both in general as well as narrow technical sense. In the narrow technical sense supervised means the following. If for a certain input the corresponding output is known, the network is to learn the mapping from inputs to outputs. In supervised learning applications, the correct output must be known and provided to the learning algorithm. The task of the network is to find the mapping. The weights are changed depending on the magnitude of the error that the network produces at the output layer, if the error is larger than the weight changes more. This is why the term *error-correction learning* is also used. Examples are the *Perceptron learning rule*, the *delta rule*, and - most famous of all - *Backprpagation*.
- **Reinforcement Learning:** If the teacher only tells a student whether his answer is correct or not, but leaves the task of determining why the answer is correct or false to the student, which means *reinforcement learning*. The problem of attributing the error (or the success) to the right cause is called the credit assignment or blame assignment problem. It is fundamental to many learning

theories. There is also a more technical meaning of the term of *reinforcement learning* as it is used in the neural network literature. It is used to designate learning where a particular behavior is to be reinforced. Typically, the robot receives a positive reinforcement signal if the result was good, no reinforcement or a negative reinforcement signal if it was bad. If the robot has managed to pick up an object, or if it has managed to shoot the ball into the goal, it will get a positive reinforcement. Reinforcement learning is not tied to neural networks, there are many reinforcement learning algorithms in the field of machine learning in general.

- **Unsupervised Learning**: Mainly two categories of learning rules fall under this unsupervised learning as Hebbian learning and competitive learning. Hebbian learning establishes correlations such as if two nodes are active simultaneously the connection between them is strengthened. Hebbian learning has become popular because it is not very powerful as a learning mechanism. In industrial applications, Hebbian learning is not used. Competitive learning, in particular Kohonen networks is used to find clusters in data sets. In addition, they have been put to many industrial usages.

6.7 Introduction to Planning

The key in planning is to use logic in order to solve problem elegantly. Planning is an advanced form of problem solving which generates a sequence of operators that guarantee the goal. Furthermore, such sequence of operators or actions (commonly used in planning literature) is called a plan. STRIPS is one of the founding languages developed particularly for planning.

What is planning in Artificial Intelligence?

- The planning in Artificial Intelligence is about the decision making tasks performed by the robots or computer programs to achieve a specific goal.
- The execution of planning is about choosing a sequence of actions with a high likelihood to complete the specific task.

Planning VS Problem Solving

Planning introduces the following improvements with respect to problem solving:

- Each state is represented in predicate logic. De-facto representation of a state is the conjunction (AND) of predicates that are true in that state.

- The goal is also represented as states, i.e. conjunction of predicates.
- Each action (or operator) is associated with some logic preconditions that must be true for that action to be applied. Thus a planning system can avoid any action that is just not possible at a particular state.
- Each action is associated with an 'effect' or post-conditions. These post conditions specify the added and/or deleted predicates when the action is applied.
- The inference mechanism used is that of backward chaining so as to use only the actions and states that are really required to reach goal state.
- Optional: The sequence of actions (plan) is minimally ordered. Only those actions are ordered in a sequence when any other order will not achieve the desired goal. Therefore, planning allows partial ordering i.e. there can be two actions that are not in any order from each other because any particular order used amongst them will achieve the same goal.