**GURUNANAK INSTITUTE OF TECHNOLOGY**

**157/F, Nilgunj Road, Panihati**
**Kolkata -700114**
**Website: www.gnit.ac.in**
**Email: info.gnit@jisgroup.org**

**Approved by A.I.C.T.E., New Delhi**

**Affiliated to MAKAUT, West Bengal**

**Online Course Ware (OCW)**

# Course: Artificial Intelligence

**Course Level: Undergraduate**

**Credit: 3**

**Prepared by:**

**Mr. Amrut Ranjan Jena (CSE)**

**Mr. Nirupam Saha (CSE)**

# ARTIFICIAL INTELLIGENCE

# INTRODUCTION

## OVERVIEW OF ARTIFICIAL INTELLIGENCE

Artificial intelligence (AI) has become a common term this days. Almost everyone from every domain seems to be aware of this term. It is almost impossible to avoid its importance in our daily life. Its presence is visible in every part of our life. For example it is visible in the applications especially in mobile apps we use on daily basis, it is visible in the movies that the film industries made. We can see its presence while we made purchases on e-commerce sites, or we are availing the facilities of medical systems. Manufacturing of an intelligent robot is possible today due to advance application of AI. Even, today the latest model of mobiles are coming equipped with a separate processing unit named as AI processor to manage the use of limited resources of a mobile device intelligently and what not.

The history of AI is very rich though it is one of the newest fields in science and engineering. Even, its presence can be seen long before the development of modern digital computers. Actually its inception had taken place while Gottfried Leibniz and Blaise Pascal constructed the mechanical machine for calculation and following it Charles Babbage introduced the very first machine with the capability of storing and manipulating symbols. However, the credit for the name "Artificial Intelligence" goes to John McCarthy who along with Marvin Minsky and Claude Shanon, organized the Dartmouth Conference in 1956. This conference was a major turning point that helps to refuel the research on AI to move forward. But it is to be mentioned that Alan Turing was the first to carry out substantial research in the field now known as Artificial Intelligence or AI though he originally termed it as "Machine Intelligence".

### WHAT IS AI?

It is important to understand that AI doesn't really have anything to do with human intelligence despite the fact that some simulators try to mimic the human intelligence. However, some definitions of AI focus on human intelligence while the others on hard problems. Let us see some of such definitions:

According to Herbert Simon "We call programs 'intelligent', if they exhibit behaviours that would be regarded intelligent if they were exhibited by human beings".

In the words of Avron Barr and Edward Feigenbaum - "Physicists ask what kind of place this universe is and seek to characterize its behavior systematically. Biologists ask what it means for a physical system to be living. We (in AI) wonder what kind of information-processing system can ask such questions."

Elaine Rich explained that – "AI is the study of techniques for solving exponentially hard problems in polynomial time by exploiting knowledge about the problem domain."

And according to Eugene Charniak and Drew McDermott – “AI is the study of mental faculties through the use of computational model.”

John Haugeland described the researches in this field as follows – “The fundamental goal of this research is not merely to mimic intelligence or produce some clever fake. “AI” wants the genuine article; *machines with minds*.”

A better way to describe AI is to categorize it in the following four ways:

**Thinking Humanly:** When a computer thinks as a human, it performs tasks that require intelligence from a human to succeed, such as driving a car. To determine whether a program thinks like a human, you must have some method of determining how humans think, which the cognitive modeling approach defnes. This model relies on three techniques: introspection, psychological testing and brain imaging.

**Acting Humanly:** When a computer acts like a human, it best reflects the Turing test, in which the computer succeeds when differentiation between the computer and a human isn’t possible.

**Thinking Rationally:** Studying how humans think using some standard enables the creation of guidelines that describe typical human behaviors. A person is considered rational when following these behaviors within certain levels of deviation. A computer that thinks rationally relies on the recorded behaviors to create a guide as to how to interact with an environment based on the data at hand. The goal of this approach is to solve problems logically, when possible.

**Acting Rationally:** Studying how humans act in given situations under specifc constraints enables you to determine which techniques are both efcient and effective. A computer that acts rationally relies on the recorded actions to interact with an environment based on conditions, environmental factors, and existing data.

Finally we can say, AI is about “*Methods and algorithms by constraints exposed by representation that support models targeted at thinking, perception and action*.”

TURING TEST

Consider the following setting. There are two rooms, A and B. One of the rooms contains a computer. The other contains a human. The interrogator is outside and does not know which one is a computer. He can ask questions through a teletype and receives answers from both A and B. The interrogator needs to identify whether A or B are humans. To pass the Turing test, the machine has to fool the interrogator into believing that it is human. For more details on the Turing test visit the site http://cogsci.ucsd.edu/~asaygin/tt/ttest.html

## PROBLEMS OF AI

Generally, problems, for which straightforward mathematical /logical algorithms are not readily available and which can be solved by intuitive approach only, are called AI problems. The 4-puzzle problem, for instance, is an ideal AI Problem. There is no formal algorithm for its realization, i.e., given a starting and a goal state, one cannot say prior to execution of the tasks the sequence of steps required to get the goal from the starting state. Such problems are called the ideal *AI problems*. The well known water-jug problem, the Travelling Salesperson Problem (TSP), and the n-Queen problem are typical examples of the classical AI problems. Among the non-classical AI problems, the diagnosis problems and the pattern classification problem need special mention. For solving an AI problem, one may employ both AI and non-AI algorithms.

For better understanding we can divide the problems of AI into two categories, common AI problems and expert AI problems. For example, common AI problems may include identification of people, objects/things, communication using natural languages, smartly moving around on the road to avoid traffic and other obstacles etc. these are the kind of problems which can be solved through regular practice and intelligence. Whereas, expert AI problems includes those which demands specialized skills such as solving difficult mathematical or logical problems, defining strategies or solutions for games requiring higher and complex level of logical thinking and reasoning power, medical diagnosis etc. What is interesting here is that computer systems are able to solve many sophisticated and expert level AI problem quite efficiently although many times they failed to solve regular or common problems which are easily solvable with human intelligence without applying any sophisticated AI techniques.

## AI TECHNIQUES

The subject of AI spans a wide horizon. It deals with the various kinds of knowledge representation schemes, different techniques of intelligent search, various methods for resolving uncertainty of data and knowledge, different schemes for automated machine learning and many others. Among the application areas of AI, we have Expert systems, Game-playing, and Theorem-proving, Natural language processing, Image recognition, Robotics and many others. The subject of AI has been enriched with a wide discipline of knowledge from Philosophy, Psychology, Cognitive Science, Computer Science, Mathematics and Engineering. All the researches in AI makes one thing clear, that intelligence requires knowledge. Most of the time knowledge possesses some less desirable properties, including:

- It is voluminous.
- It is hard to characterize accurately.
- It is constantly changing.
- It differs from data by being organized in a way that corresponds to the ways it will be used.

Based on all these information it can be concluded that AI technique is a method that exploits knowledge that should be represented in such a way that,

- The knowledge captures generalization.
- It should be perceivable by the people who provide it.
- It should be easily modifiable to correct errors and to reflect changes.
- It should be useful in many situations though it is incomplete or inaccurate.
- It can be applied on itself to narrow down the range of possibilities for better processing and consideration.

Whatsoever, an AI problem is solvable without applying AI techniques although it may not assure perfect or at least a good solution or result. Similarly, non AI problems can also be solved by applying AI techniques.

**TIC - TAC - TOE PROBLEM**

Two people play Tic Tac Toe with paper and pencil. One player is X and the other player is O. Players take turns placing their X or O. If a player gets three of their marks on the board in a row, column or one of the two diagonals, they win. When the board fills up with neither player winning, the game ends in a draw.

From AI point of view, the problem of playing Tic-Tac-Toe will be formulated as follows:

The start state is all blank squares out of 9 squares. Player 1 can play in one square. As

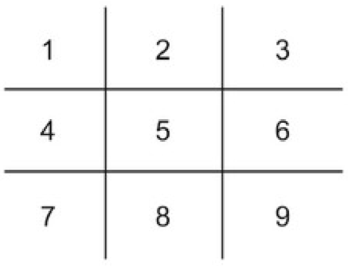


**Fig. 1.1:** Element positions of tic-tac-toe

the game proceeds, blank squares remain the choice, which can be marked by the players. The data structure used to represent the board is a 9-element vector, with element position shown in Fig. 1.1:

Board position: = {1,2,3,4,5,6,7,8,9}
An element contains the value 0, if the corresponding square is blank; 1, if it is filled with “O” and 2, if it is filled with “X”.
Hence starting state is {0,0,0,0,0,0,0,0,0}
The goal state or winning combination will be board position having “O” or “X” separately in

the combination of {1,2,3}, {4,5,6}, {7,8,9},{1,4,7},{2,5,8},{3,6,9}, {1,5,9}, { 3,5,7}) element values. Hence two goal states can be
{2,0,1,1,2,0,0,0,2} and {2,2,2,0,1,0,1,0,0}. These values correspond to the goal states shown in the figure.The start and goal state are shown in Fig. 1.2.

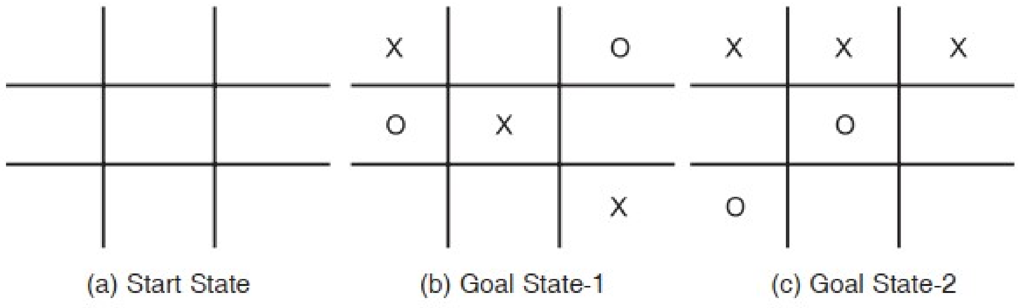


**Fig. 1.2:** Start and goal states of tic-tac-toe

Any board position satisfying this condition would be declared as win for corresponding player. The valid transitions of this problem are simply putting ‘1’ or ‘2’ in any of the element position containing 0. In practice, all the valid moves are defined and stored. While selecting a move it is taken from this store. In this game, valid transition table will be a vector (having 39 entries), having 9 elements in each.

# INTELLIGENT AGENTS

## AGENTS & ENVIRONMENT

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators. The current percept or a sequence of percepts can influence the actions of an agent. We use the term **percept** to refer to the agent's perceptual inputs at any given instant. An agent's **percept sequence** is the complete history of everything the agent has ever perceived. In general, an agent's choice of action at any given instant can depend on the entire percept sequence observed to date, but not on anything it hasn't perceived. The agent can change the environment through actuators or effectors. An operation involving an actuator is called an action.

We can also consider the agents as autonomous systems. They would be persistent, goal oriented, pro-active and would sense the situations and surroundings. They would maintain the social ability by communicating with the owners and the other agents. For an agent to act out its decision, it must be embodied in some environment.

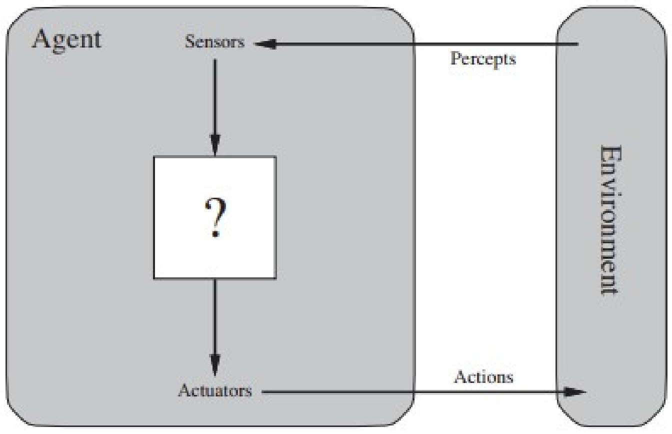


**Fig. 2.1:** Agents interacting with environment

An agent can be looked upon as a system that implements a mapping from percept sequences to actions. A performance measure has to be used in order to evaluate an agent. An autonomous agent decides autonomously which action to take in the current situation to maximize progress towards its goals. The agent function for an artificial agent will be implemented by an **agent program**. It is important to keep these two ideas distinct. The agent function is an abstract mathematical description; the agent program is a concrete implementation, running within some physical system.

## 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. The most famous artificial environment is the Turing Test environment, in which one real and other artificial agents 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.

In spite of the difficulty of knowing exactly where the environment ends and the agent begins in some cases, it is useful to be able to classify AI environments because it can predict how difficult the task of the AI will be. Russell and Norvig (2009) introduce seven ways to classify AI environments, which can be remembered with the mnemonic "D-SOAKED." They are:

- **Deterministicness** (deterministic or stochastic or Non-deterministic): An environment is deterministic if the next state is perfectly predictable given knowledge of the previous state and the agent's action.
- **Staticness** (static or dynamic): Static environments do not change while the agent deliberates.
- **Observability** (full or partial): A fully observable environment is one in which the agent has access to all information in the environment relevant to its task.
- **Agency** (single or multiple): If there is at least one other agent in the environment, it is a multi-agent environment. Other agents might be apathetic, cooperative, or competitive.
- **Knowledge** (known or unknown): An environment is considered to be "known" if the agent understands the laws that govern the environment's behavior. For example, in chess, the agent would know that when a piece is "taken" it is removed from the game. On a street, the agent might know that when it rains, the streets get slippery.
- **Episodicness** (episodic or sequential): Sequential environments require memory of past actions to determine the next best action. Episodic environments are a series of one-shot actions, and only the current (or recent) percept is relevant. An AI that looks at radiology images to determine if there is a sickness is an example of an episodic environment. One image has nothing to do with the next.
- **Discreteness** (discrete or continuous or ): A discrete environment has fixed locations or time intervals. A continuous environment could be measured quantitatively to any level of precision.

In each case, the job of the AI (and for the programmer making the AI) is easier if the first of the two options is the best descriptor for each category. That is, an AI that has a much more difficult job if it works in an environment that is stochastic, dynamic, partially observable, multi-agent, unknown, sequential, and continuous.

## STRUCTURE OF AGENTS

The job of AI is to design an agent program that implements the agent function — the mapping from percepts to actions. We assume this program will run on some sort of computing device with physical sensors and actuators — we call this the architecture:

*agent = architecture + program .*

*architecture = the machinery that an agent executes on.*

*agent program = an implementation of an agent function.*

### AGENT PROGRAM

The agent program takes just the current percept as input because nothing more is available from the environment; if the agent's actions need to depend on the entire percept sequence, the agent will have to remember the percepts.

### AGENT ARCHITECTURE

There are various agent architectures are available. Such as:

- Table based agent
- Percept based agent or reflex agent
- Subsumption Architecture
- State-based Agent or model-based reflex agent
- Goal-based Agent
- Utility-based Agent
- Learning Agent

**Table based agent.** In table based agent the action is looked up from a table based on information about the agent's percepts. A table is simple way to specify a mapping from percepts to actions. The mapping is implicitly defined by a program. The mapping may be implemented by a rule based system, by a neural network or by a procedure. There are several disadvantages to a table based system. The tables may become very large. Learning a table may take a very long time, especially if the table is large. Such systems usually have little autonomy, as all actions are pre-determined.

**Percept based agent or reflex agent.** In percept based agents, information comes from **sensors** i.e. **percepts** which changes the agents current **state of the world** and triggers **actions** through the **effectors.**

These kinds of agents are also called reactive agents or stimulus-response agents. They have no notion of history. The current state is as the sensors see it right now. The action is based on the current percepts only.

The percept base agents are efficient but they don't have any internal representation for reasoning inference. Besides this, these agents don't follow any strategic planning or learning. Finally they are not good for multiple opposing goals.

**Subsumption Architecture.** This architecture is based on reactive systems, also known as Brook's architecture. The main features of Brooks' architecture are as follows:

1. There is no explicit knowledge representation.
2. Behaviour is distributed, not centralized
3. Response to stimuli is reflexive.
4. The design is bottom up, and complex behaviours are fashioned from the combination of simpler underlying ones.
5. Individual agents are simple.

The Subsumption Architecture built in layers. There are different layers of behaviour. The higher layers can override lower layers. Each activity is modeled by a finite state machine. The subsumption architecture can be illustrated by Brooks' Mobile Robot example.

**State-based Agent or model-based reflex agent.** State based agents differ from percept based agents in that such agents maintain some sort of state based on the percept sequence received so far. The state is updated regularly based on what the agent senses, and the agent's actions. Keeping track of the state requires that the agent has knowledge about how the world evolves, and how the agent's actions affect the world.

Thus a state based agent works as follows:

- information comes from **sensors** – **percepts**
- based on this, the agent changes the current **state of the world**
- based on **state of the world** and **knowledge (memory)**, it triggers **actions** through the **effectors**

**Goal-based Agent.** Knowing something about the current state of the environment is not always enough to decide
what to do. As well as a current state description, the agent needs some sort of **goal** information that describes situations that are desirable. The agent program can combine this with the model (the same information as was used in the model based reflex agent) to choose actions that achieve the goal. Sometimes goal-based action selection is straightforward—for example, when goal satisfaction results immediately from a single action. Sometimes it will be more tricky—for

example, when the agent has to consider long sequences of twists and turns in order to find a way to achieve the goal. **Search** and **planning** are the subfields of AI devoted to finding action sequences that achieve the agent's goals.

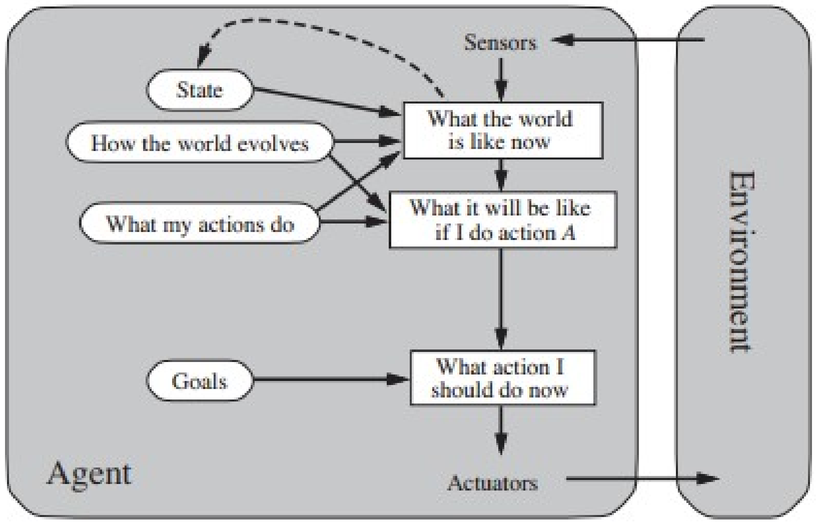

Sensors
State
What the world is like now
How the world evolves
What my actions do
What it will be like if I do action *A*
Goals
What action I should do now
Agent
Actuators
Environment

**Fig. 2.2:** Structure of Goal based agent

Although the goal-based agent appears less efficient, it is more flexible because the knowledge that supports its decisions is represented explicitly and can be modified. For the reflex agent, on the other hand, we would have to rewrite many condition–action rules. The goal-based agent's behavior can easily be changed to go to a different destination, simply by specifying that destination as the goal. The reflex agent's rules for when to turn and when to go straight will work only for a single destination; they must all be replaced to go somewhere new.

**Utility-based Agent.** Goals alone are not enough to generate high-quality behavior in most environments. Goals just provide a crude binary distinction between "happy" and "unhappy" states. A more general performance measure should allow a comparison of different world states according to exactly how happy they would make the agent. Because "happy" does not sound very scientific, economists and computer scientists use the term **utility** instead. An agent's **utility function** is essentially an internalization of the performance measure. If the internal utility function and the external performance measure are in agreement, then an agent that chooses actions to maximize its utility will be rational according to the external performance measure. A rational utility-based agent chooses the action that maximizes the **expected utility** of the action outcomes—that is, the utility the agent expects to derive, on average, given the probabilities and utilities of each outcome.

Utility based agents provide a more general agent framework. In case that the agent has multiple goals, this framework can accommodate different preferences for the different goals. Such systems are characterized by a utility function that maps a state or a sequence of states to a real valued utility. The agent acts so as to maximize expected utility.

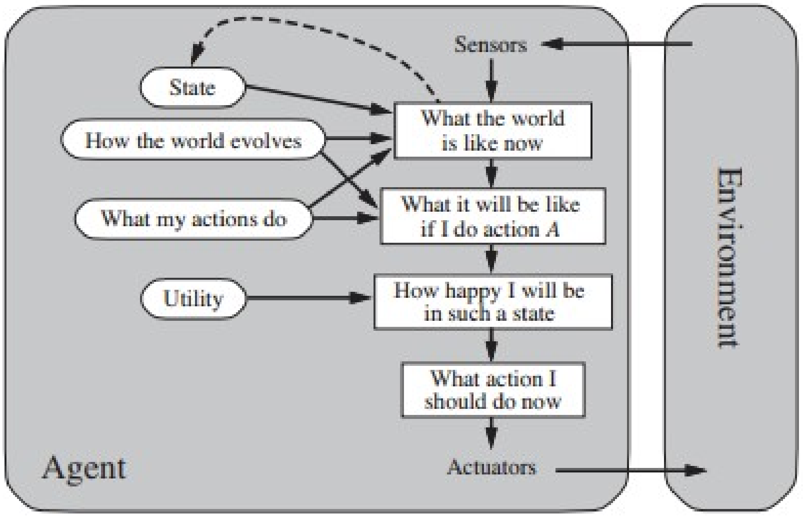

Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action A
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

Fig. 2.3: A model-based, utility-based agent.

**Learning Agent.** Learning allows an agent to operate in initially unknown environments. The learning element modifies the performance element. Learning is required for true autonomy. Alan Turing in his one famous paper proposed the method to build learning machines and then to teach them.   In many areas of AI, this is now the preferred method for creating state-of-the-art systems. Learning has another advantage, as we noted earlier: it allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow.

A learning agent can be divided into four conceptual components, as shown in Figure 2.4. The most important distinction is between the **learning element**, which is responsible for making improvements, and the **performance element**, which is responsible for selecting external actions. The learning element uses feedback from the **critic** on how the agent is doing and determines how the performance element should be modified to do better in the future. The design of the learning element depends very much on the design of the performance element. The last component of the learning agent is the **problem generator**. It is responsible for suggesting actions that will lead to new and informative experiences.

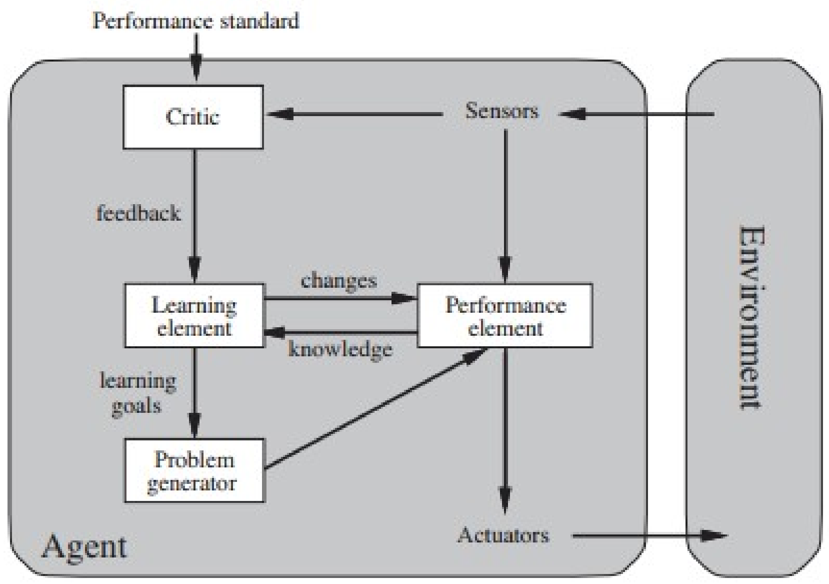


**Fig. 2.4:** A general learning agent.

# PROBLEM SOLVING

## PROBLEMS

A **problem** can be defined formally by five components:

- The **initial state** that the agent starts in. For example, the initial state for our agent in Romania might be described as *In(Arad)*.
- A description of the possible **actions** available to the agent. Given a particular state *s*, ACTIONS(s) returns the set of actions that can be executed in s. We say that each of these actions is **applicable** in *s*. For example, from the state *In(Arad)*, the applicable actions are {*Go(Sibiu), Go(Timisoara), Go(Zerind)*}.
- A description of what each action does; the formal name for this is the **transition model**, specified by a function RESULT(s, a) that returns the state that results from doing action a in state *s*. We also use the term **successor** to refer to any state reachable from a given state by a single action.2 For example, we have

  *RESULT(In(Arad), Go(Zerind)) = In(Zerind)* .

- The **goal test**, which determines whether a given state is a goal state.
- A **path cost** function that assigns a numeric cost to each path.

To build a system to solve a particular problem, we need to do four things:

1. Define the problem precisely. This definition must include precise specifications of what the initial situation(s) will be as well as what final situations constitute acceptable solutions to the problem.
2. Analyze the problem. A few very important features can have an immense impact on the appropriateness of various possible techniques for solving the problem.
3. Isolate and represent the task knowledge that is necessary to solve the problem.
4. Choose the best problem-solving technique(s) and apply it (them) to the particular problem.

Searching is the universal technique of problem solving in AI. Problem solving requires two prime considerations: first representation of the problem by an appropriately organized state space and then testing the existence of a well-defined goal state in that space.

## PROBLEM SPACE & SEARCH

### PROBLEM STATE SPACE

Together, the initial state, actions, and transition model implicitly define the **state space** of the problem—the set of all states reachable from the initial state by any sequence of actions. The state space forms a directed network or **graph** in which the nodes are states and the links between nodes are actions. (The map of Romania shown in Figure 3.1 can be interpreted as a state-space graph if we view each road as standing for two driving actions, one in each direction.) A **path** in the state space is a sequence of states connected by a sequence of actions.

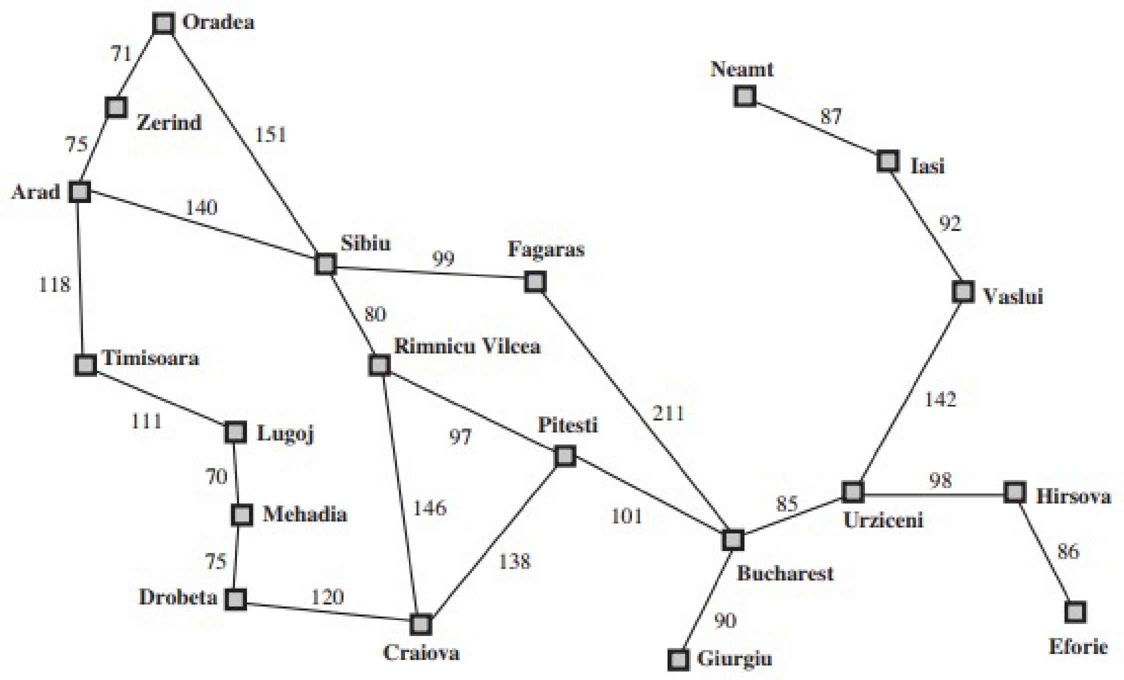


**Fig. 3.1:** A simplified road map of part of Romania.

PRODUCTION SYSTEM

‘Production systems’ is one of the oldest techniques of knowledge representation. A production system includes a knowledge base, represented by production rules, a working memory to hold the matching patterns of data that causes the rules to fire and an interpreter, also called the inference engine, that decides which rule to fire, when more than one of them are concurrently firable.

The production system is a model of computation that can be applied to implement search algorithms and model human problem solving. Such problem solving knowledge can be packed up in the form of little quanta called productions. A production is a rule consisting of a situation recognition part and an action part. A production is a situation-action pair in which the left side is a list of things to watch for and the right side is a list of things to do so. When productions are used in deductive systems, the situation that trigger productions are specified combination of facts. The actions are restricted to being assertion of new facts deduced directly from the triggering combination. Production systems may be called premise conclusion pairs rather than situation action pair.

A production system consists of:

- A set of rules, each consisting of a left side (a pattern) that determines the applicability of the rule and a right side that describes the operation to be performed if the rule is applied.
- One or more knowledge/databases that contain appropriate information for the particular task. Some parts of the database may be permanent, while other parts of it may pertain only to the solution of the current problem. The information in these databases may be structured in a appropriate way.
- A control strategy that specifies order in which the rules will be compared to the database of rules and a way of resolving the conflicts that arise when several rules match simultaneously.
- A rule applier, which checks the capability of rule by matching the content state with the left hand side of the rule and finds the appropriate rule from database of rules.

The important roles played by production systems include a powerful knowledge representation scheme. A production system not only represents knowledge but also action. It acts as a bridge between AI and expert systems. Production system provides a language in which the representation of expert knowledge is very natural.

FEATURES OF PRODUCTION SYSTEM

Some of the main features of production system are:

Expressiveness and intuitiveness: In real world, many times situation comes like “if this happen-you will do that”, “if this is so-then this should happen” and many more. The production rules essentially tell us what to do in a given situation.

- *Simplicity:* The structure of each sentence in a production system is unique and uniform as they use “IF-THEN” structure. This structure provides simplicity in knowledge representation. This feature of production system improves the readability of production rules.
- *Modularity:* This means production rule code the knowledge available in discrete pieces. Information can be treated as a collection of independent facts which may be added or deleted from the system with essentially no deletetious side effects.
- *Modifiability:* This means the facility of modifying rules. It allows the development of production rules in a skeletal form first and then it is accurate to suit a specific application.
- *Knowledge intensive:* The knowledge base of production system stores pure knowledge. This part does not contain any type of control or programming information. Each production rule is normally written as an English sentence; the problem of semantics is solved by the very structure of the representation.

DISADVANTAGES OF PRODUCTION SYSTEM

- *Opacity:* This problem is generated by the combination of production rules. The opacity is generated because of less prioritization of rules. More priority to a rule has the less opacity.
- *Inefficiency:* During execution of a program several rules may active. A well devised control strategy reduces this problem. As the rules of the production system are large in number and they are hardly written in hierarchical manner, it requires some forms of complex search through all the production rules for each cycle of control program.
- *Absence of learning:* Rule based production systems do not store the result of the problem for future use. Hence, it does not exhibit any type of learning capabilities. So for each time for a particular problem, some new solutions may come.
- *Conflict resolution:* The rules in a production system should not have any type of conflict operations. When a new rule is added to a database, it should ensure that it does not have any conflicts with the existing rules.

## PROBLEM CHARACTERISTICS

A problem may have different aspects of representation and explanation. In order to choose the most appropriate method for a particular problem, it is necessary to analyze the problem along several key dimensions. Some of the main key features of a problem are given below.

- Is the problem decomposable into a set of independent smaller or easier sub problems?
- Can solution steps be ignored or at least undone if they prove unwise?
- Is the problem's universe predictable?
- Is a good solution to the problem obvious without comparison to all the other possible solutions?
- Is the desired solution a state of the world or a path to a state?
- Is a large amount of knowledge absolutely required to solve the problem, or is knowledge important only to constrain the search?
- Will the solution of the problem required interaction between the computer and the person?

The above characteristics of a problem are called as 7-problem characteristics under which the solution must take place.

## ISSUES IN THE DESIGN OF SEARCH PROGRAMS

Every search process can be viewed as traversal of a tree structure in which each node represents a problem state and each are represents a relationship between the states represented by the node it connects. The search process must find a path or paths through the tree that connects an initial state with one or more final states. The tree that must be searched could, in principle, be constructed in its entirety from the rules that define allowable moves in the problem space. But, in practice, most of it never is. It is too large and most of it need never be explored. Instead of first building the tree explicitly and then searching it, most search programs represent the tree implicitly in the rules and generate explicitly only those parts that they decide to explore.

Following are some of the issues that arise in all general purpose search techniques:

- Direction in which to conduct the search. We can search forward through the state space from the start state to goal state, or we can search backward from the goal.
- Production systems typically spend most of their time looking for rules to apply, so it is critical to have efficient procedures for matching rules against states.
- How to represent each node of the search processs (the knowledge representation problem and the frame problem).

# SEARCH TECHNIQUES

## SOLVING PROBLEMS BY SEARCHING

### PROBLEM SOLVING AGENTS

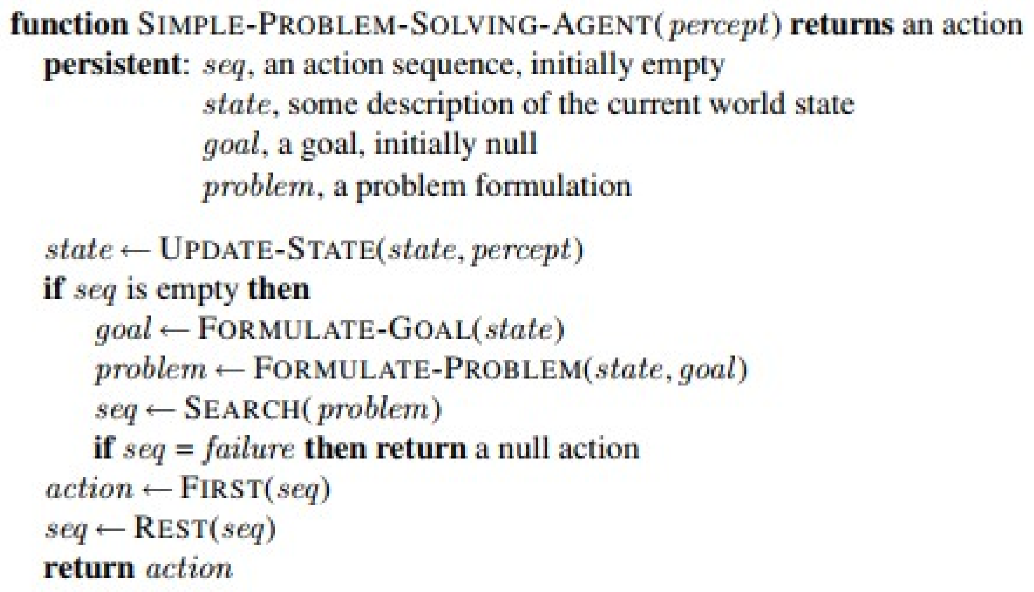
Problem solving in AI may be characterized as a systematic search through a range of possible actions in order to reach some predefined goal or solution. The problem solving agents decide what to do by finding sequence of action that lead to desirable states.

Intelligent agents are supposed to maximize their performance measure. achieving this is sometimes simplified if the agent can adopt a **goal** and aim at satisfying it. Goals help organize behavior by limiting the objectives that the agent is trying to achieve and hence the actions it needs to consider. **Goal formulation**, based on the current situation and the agent's performance measure, is the first step in problem solving. Whereas, **Problem formulation** is the process of deciding what actions and states to consider, given a goal. If the agent has three different path to reach its goal then may be the agent will not know which of its possible actions is best, because it does not yet know enough about the state that results from taking each action. If the agent has no additional information—i.e., if the environment is **unknown** then it is has no choice but to try one of the actions at random. In such situation, an agent with several immediate options of unknown value can decide what to do by first examining future actions that eventually lead to states of known value.

### SEARCHING FOR SOLUTIONS

The process of looking for a sequence of actions that reaches the goal is called **search**. A search algorithm takes a problem as input and returns a **solution** in the form of an action sequence. Once a solution is found, the actions it recommends can be carried out. This is called the **execution** phase. Thus, we have a simple "formulate, search, execute" design for the agent, as shown in Figure 4.1.

After formulating a goal and a problem to solve, the agent calls a search procedure to solve it. It then uses the solution to guide its actions, doing whatever the solution recommends as the next thing to do—typically, the first action of the sequence—and then removing that step from the sequence. Once the solution has been executed, the agent will formulate a new goal. It is to be noted that while the agent is executing the solution sequence it ignores its percepts when choosing an action because it knows in advance what they will be. According to the control theorists it is called open-loop system.



```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

**Fig. 4.1** A simple problem-solving agent. It first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. When this is complete, it formulates another goal and starts over.

## UNIFORM SEARCH STRATEGIES

Uniform search also known as blind search. The term means that the strategies have no additional information about states beyond that provided in the problem definition. All they can do is generate successors and distinguish a goal state from a non-goal state. Intuitively, these algorithms ignore where they are going until they find a goal and report success.

### BREADTH FIRST SEARCH (BFS)

Breadth first search is a general technique of traversing a graph. Breadth first search may use more memory but will always find the shortest path first. In this type of search the state space is represented in form of a tree. The solution is obtained by traversing through the tree. The nodes of the tree represent the start value or starting state, various intermediate states and the final state. In this search a queue data structure is used and it is level by level traversal. Breadth first search expands nodes in order of their distance from the root. It is a path finding algorithm that is capable of always finding the solution if one exists. The solution which is found is always the optional solution. This task is completed in a very memory intensive manner. Each node in the search tree is expanded in a breadth wise at each level.

**Concept:**

**Step 1:** Traverse the root node
**Step 2:** Traverse all neighbours of root node.
**Step 3:** Traverse all neighbours of neighbours of the root node.
**Step 4:** This process will continue until we are getting the goal node.

**Algorithm:**

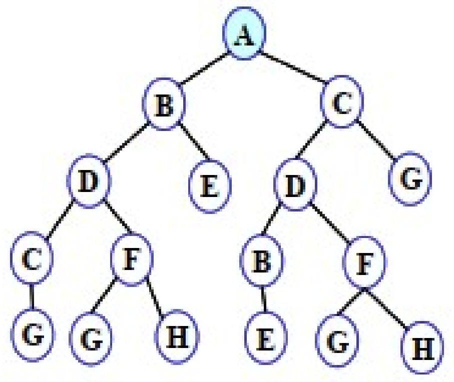
```
Breadth first search
Let fringe be a list containing the initial state
Loop
        If fringe is empty return failure
        Node ← remove-first (fringe)
        if  Node is a goal
                then return the path from initial state to Node
        else generate all successors of Node, and
                (merge the newly generated nodes into fringe)
                add generated nodes to the back of fringe
End Loop
```

Note that in breadth first search the newly generated nodes are put at the back of fringe or the OPEN list. What this implies is that the nodes will be expanded in a FIFO (First In First Out) order. The node that enters OPEN earlier will be expanded earlier. This amounts to expanding the shallowest nodes first.

**BFS illustrated:**

We will now consider the search space in Figure 1, and show how breadth first search works on this graph.

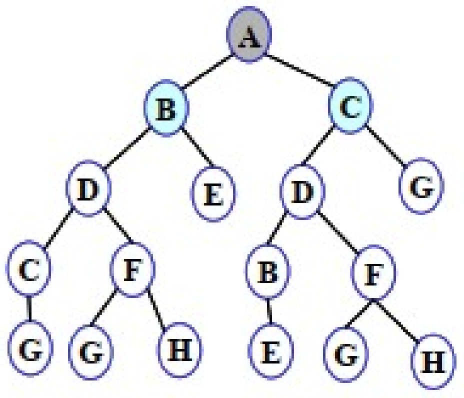
**Step 1:** Initially fringe contains only one node corresponding to the source state A.



(i)

FRINGE: A

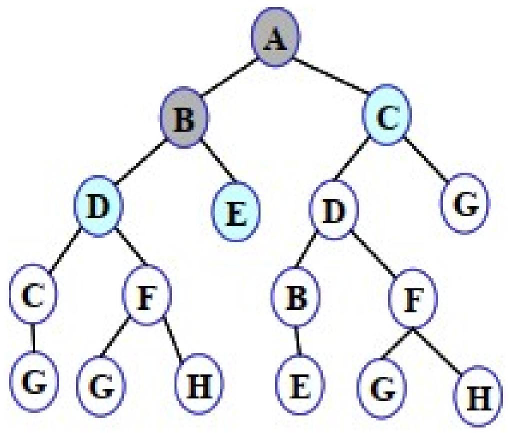
**Step 2:** A is removed from fringe. The node is expanded, and its children B and C are generated. They are placed at the back of fringe.



(ii)

FRINGE: B   C

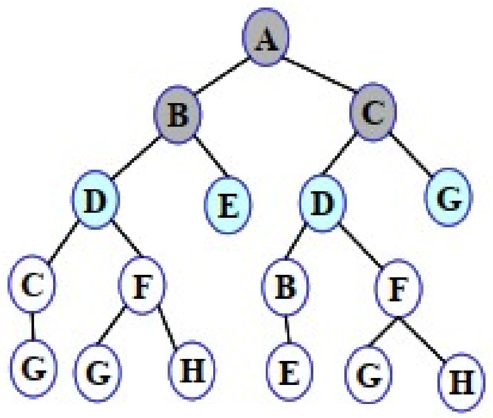
**Step 3:** Node B is removed from fringe and is expanded. Its children D, E are generated and put at the back of fringe.



(iii)

FRINGE: C  D   E

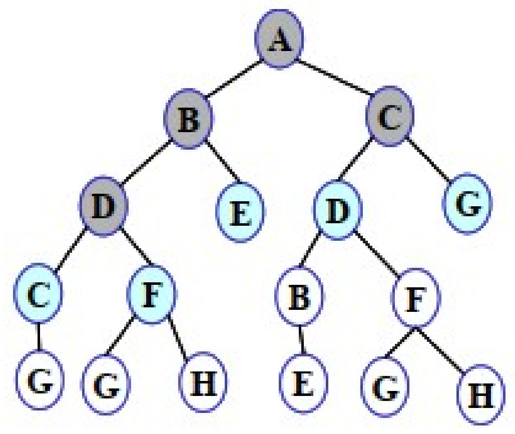
**Step 4:** Node C is removed from fringe and is expanded. Its children D and G are added to the back of fringe.



(iv)

FRINGE: D E D G

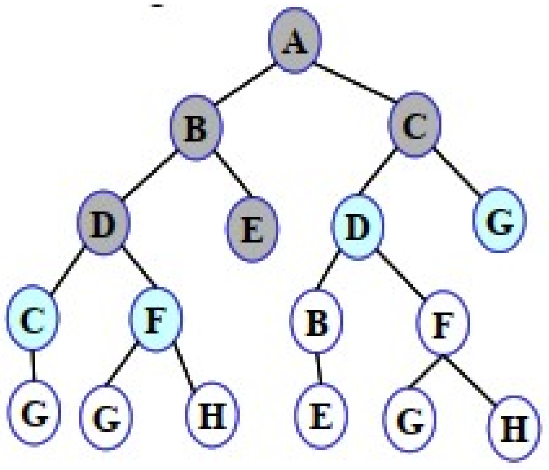
**Step 5:** Node D is removed from fringe. Its children C and F are generated and added to the back of fringe.



(v)

FRINGE: E D G C F

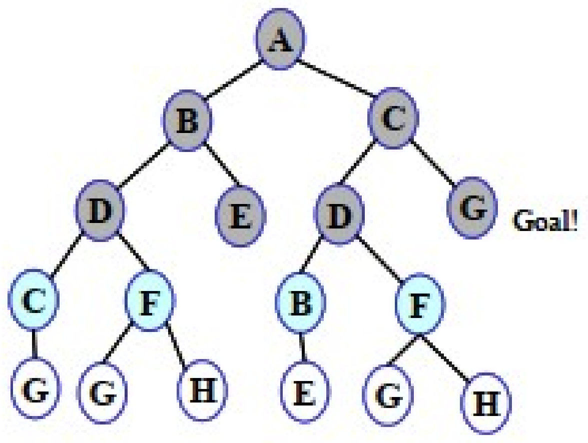
**Step 6:** Node E is removed from fringe. It has no children.



(vi)

FRINGE: D G C F

Step 7: D is expanded, B and F are put in OPEN.



(vii)

FRINGE: ***G*** C F B F

**Step 8:** G is selected for expansion. It is found to be a goal node. So the algorithm returns the path A C G by following the parent pointers of the node corresponding to G. The algorithm terminates.

We can easily see that it is *complete*—if the shallowest goal node is at some finite depth d, breadth-first search will eventually find it after generating all shallower nodes (provided the branching factor b is finite). The *shallowest* goal may not necessarily be the *optimal* one. If the path cost is a non-decreasing function of the depth of the node then we can say that the breadth-first search is optimal.

Imagine searching a uniform tree where every state has b successors. The root of the search tree generates b nodes at the first level, each of which generates b more nodes, for a total of $b^2$ at the second level. Each of *these* generates b more nodes, yielding $b^3$ nodes at the third level, and so on. Now suppose that the solution is at depth d. In the worst case, it is the last node generated at that level. Then the total number of nodes generated is

$$b + b^2 + b^3 + \cdots + b^d = O(b^d) .$$

If the algorithm were to apply the goal test to nodes when selected for expansion, rather than when generated, the whole layer of nodes at depth d would be expanded before the goal was detected and the time complexity would be $O(b^{d+1})$.

**Advantage**

Finds the path of minimal length to the goal.

**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 (DFS)

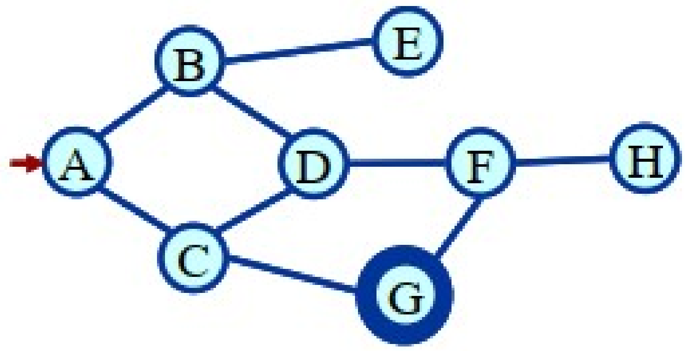
It is implemented in recursion with LIFO stack data structure. It creates the same set of nodes as Breadth-First method, only in the different order. As the nodes on the single path are stored in each iteration from root to leaf node, the space requirement to store nodes is linear. With branching factor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

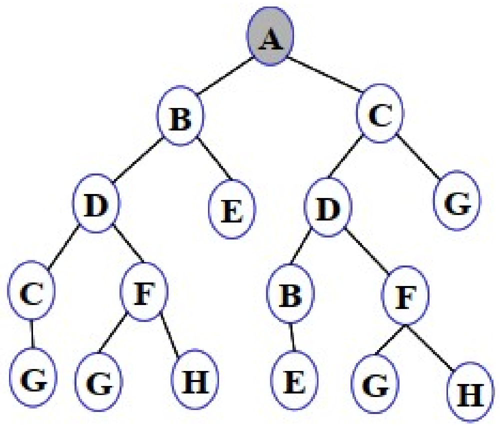
**Algorithm**

```
Depth First Search
Let fringe be a list containing the initial state
Loop
        if fringe is empty return failure
                Node ← remove-first (fringe)
        if Node is a goal
                then return the path from initial state to Node
        else generate all successors of Node, and
                merge the newly generated nodes into fringe
                add generated nodes to the front of fringe
End Loop
```

The depth first search algorithm puts newly generated nodes in the front of OPEN. This results in expanding the deepest node first. Thus the nodes in OPEN follow a LIFO order (Last In First Out). OPEN is thus implemented using a stack data structure.

**DFS illustrated:**

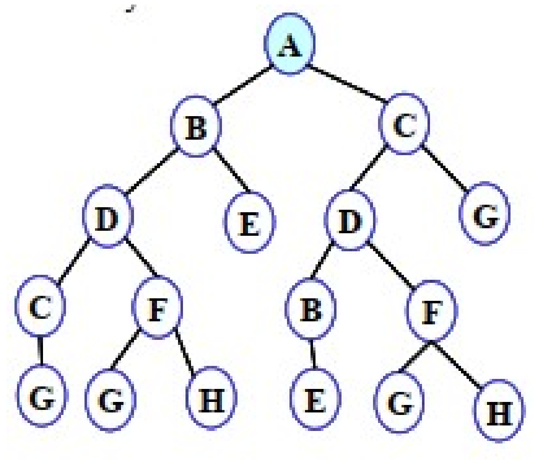




(viii)

Let us now run Depth First Search on the search space given in Figure 34, and trace its progress.

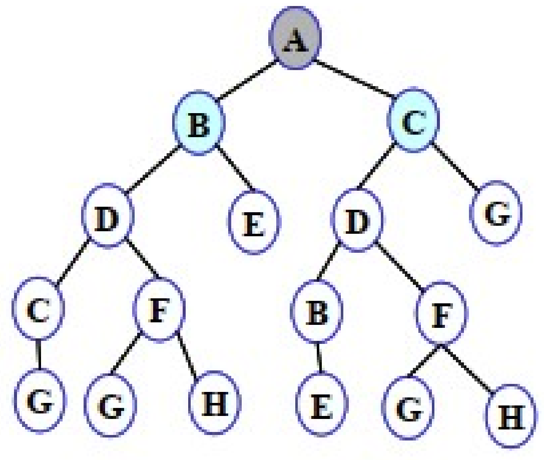
**Step 1:** Initially fringe contains only the node for A.



(ixi)

FRINGE: A

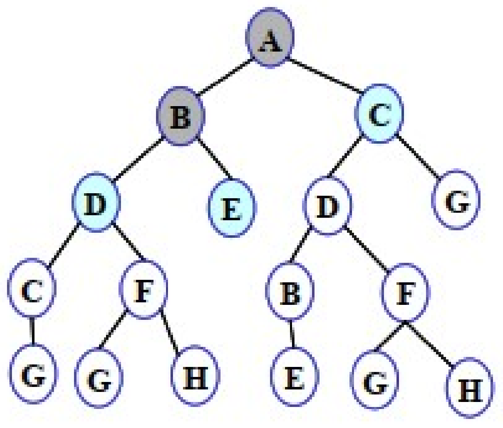
**Step 2:** A is removed from fringe. A is expanded and its children B and C are put in front of fringe.



(x)

FRINGE: B C

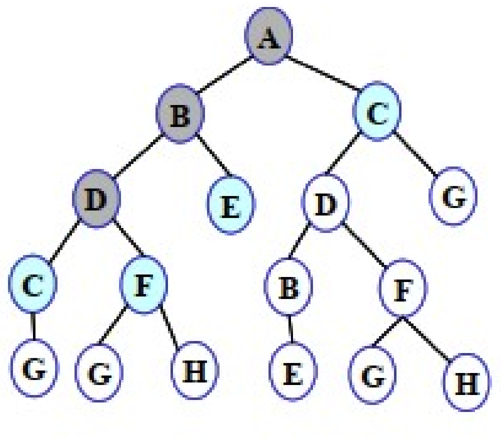
**Step 3:** Node B is removed from fringe, and its children D and E are pushed in front of fringe.



(xi)

FRINGE: D E C

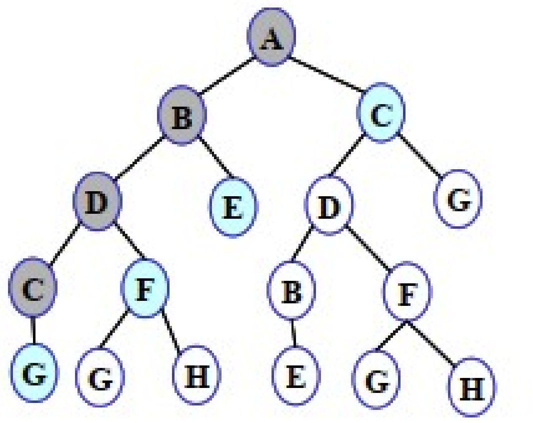
**Step 4:** Node D is removed from fringe. C and F are pushed in front of fringe.



(xii)

FRINGE: C F E C

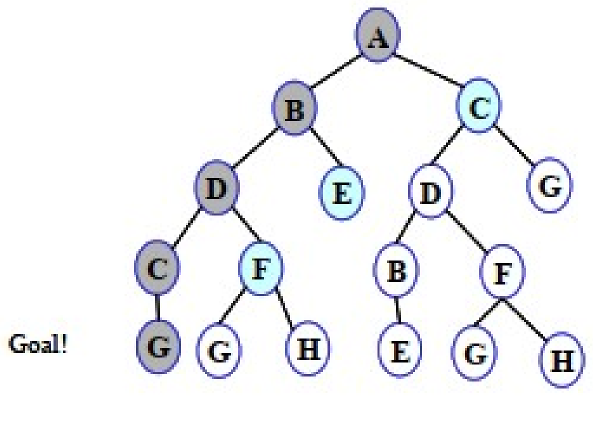
**Step 5:** Node C is removed from fringe. Its child G is pushed in front of fringe.



(xiii)

FRINGE: G F E C

**Step 6:** Node G is expanded and found to be a goal node. The solution path A-B-D-C-G is returned and the algorithm terminates.



(xiv)

FRINGE: ***G*** F E C

This algorithm takes exponential time. If N is the maximum depth of a node in the search space, in the worst case the algorithm will take time $O(b^d)$. However the space taken is linear in the depth of the search tree, $O(b^N)$.

Note that the time taken by the algorithm is related to the maximum depth of the search tree. If the search tree has infinite depth, the algorithm may not terminate. This can happen if the search space is infinite. It can also happen if the search space contains cycles. The latter case can be handled by checking for cycles in the algorithm. Thus Depth First Search is not complete.

**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 cutoff 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.

DEPTH LIMITED SEARCH

The embarrassing failure of depth-first search in infinite state spaces can be alleviated by supplying depth-first search with a predetermined depth limit. That is, nodes at depth  are treated as if they have no successors. This approach is called **depth-limited search**. The depth limit solves the infinite-path problem. Unfortunately, it also introduces an additional source of incompleteness if we choose $l < d$, that is, the shallowest goal is beyond the depth limit. (This is likely when d is unknown.) Depth-limited search will also be nonoptimal if we choose $l > d$. Its time complexity is $O(b^l)$ and its space complexity is $O(bl)$. Depth-first search can be viewed as a special case of depth-limited search with $l = \infty$.

```
Depth limited search (limit)
Let fringe be a list containing the initial state
Loop
        if fringe is empty return failure
                Node ← remove-first (fringe)
        if Node is a goal
                then return the path from initial state to Node
        else if depth of Node = limit return cutoff
        else add generated nodes to the front of fringe
End Loop
```

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.

The idea behind bidirectional search is to run two simultaneous searches—one forward from the initial state and the other ackward from the goal—hoping that the two searches meet in the middle (Figure 3.20). The motivation is that $b^{d/2} + b^{d/2}$ is much less than $b^d$, or in the figure, the area of the two small circles is less than the area of one big circle centered on the start and reaching to the goal.

Bidirectional search is implemented by replacing the goal test with a check to see whether the frontiers of the two searches intersect; if they do, a solution has been found. It is important to realize that the first such solution found may not be optimal, even if the two searches are both breadth-first; some additional search is required to make sure there isn't another short-cut across the gap.

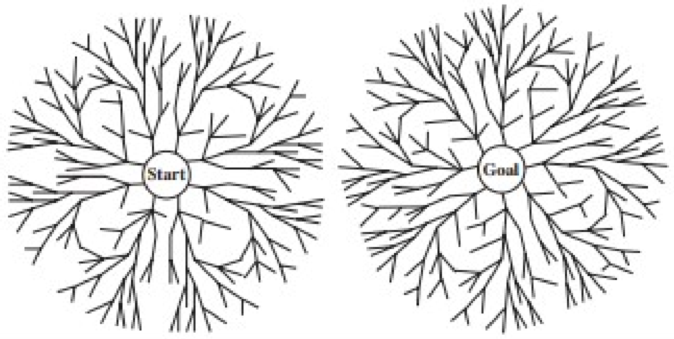
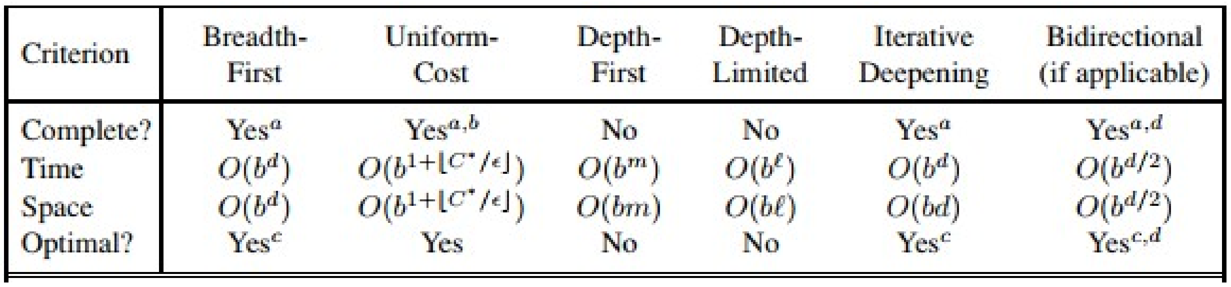


**Fig. 4.2:** A schematic view of a bidirectional search that is about to succeed when a branch from the start node meets a branch from the goal node.

## COMPARING UNIFORM SEARCH STRATEGIES

Figure 3.21 compares search strategies in terms of the four evaluation criteria set forth in Section 3.3.2. This comparison is for tree-search versions. For graph searches, the main differences are that depth-first search is complete for finite state spaces and that the space and time complexities are bounded by the size of the state space.



| Criterion | Breadth-First | Uniform-Cost | Depth-First | Depth-Limited | Iterative Deepening | Bidirectional (if applicable) |
|---|---|---|---|---|---|---|
| Complete? | Yes$^{a}$ | Yes$^{a,b}$ | No | No | Yes$^{a}$ | Yes$^{a,d}$ |
| Time | $O(b^{d})$ | $O(b^{1+\lfloor C^{*}/\epsilon \rfloor})$ | $O(b^{m})$ | $O(b^{\ell})$ | $O(b^{d})$ | $O(b^{d/2})$ |
| Space | $O(b^{d})$ | $O(b^{1+\lfloor C^{*}/\epsilon \rfloor})$ | $O(bm)$ | $O(b\ell)$ | $O(bd)$ | $O(b^{d/2})$ |
| Optimal? | Yes$^{c}$ | Yes | No | No | Yes$^{c}$ | Yes$^{c,d}$ |

**Fig. 4.2:** Evaluation of tree-search strategies. *b* is the branching factor; *d* is the depth of the shallowest solution; *m* is the maximum depth of the search tree; *l* is the depth limit. Superscript caveats are as follows: $^{a}$ complete if *b* is finite; $^{b}$ complete if step costs ≥ ϵ for positive ϵ; $^{c}$ optimal if step costs are all identical; $^{d}$ if both directions use breadth-first search.

# HEURISTIC SEARCH STRATEGIES

**Heuristic Search**

From the Greek word “Heuriskein” the word heuristic comes. It means “to discover”. Heuristic is a technique that improves the efficiency of a search process by guiding the user towards the goal. Heuristics are like tour guide. Heuristic search uses an evaluation function or heuristic function which gives an estimate of each solution i.e. evaluate the path to know whether it leads to goal or not. In the next example, a heuristic function is designed for 8 puzzle problem. At each step, the heuristic function is applied and it estimates the state to discover that whether the state is in the desired path, i.e. in the path which leads to goal or not. The cost of evaluating the heuristic function is an extra overhead to the searching process, but a well designed heuristic function saves a lot of time and effort of the user to search a specific goal state.

One example of a good general purpose heuristic for traveling salesman problem discussed in the previous chapter is nearest neighbor heuristic. The problem is like that:

i. Randomly select a starting city.

ii. To select the next city, choose a city closest to the current city.

iii. Repeat step ii until all cities are visited. It takes time proportional to $N^2$ (N=number of city).

Significant improvement over N!.

Example 5.1: 8 puzzle problem is given below.

| | | |
|---|---|---|
| | 8 | 3 |
| 1 | 6 | 4 |
| | 7 | 5 |

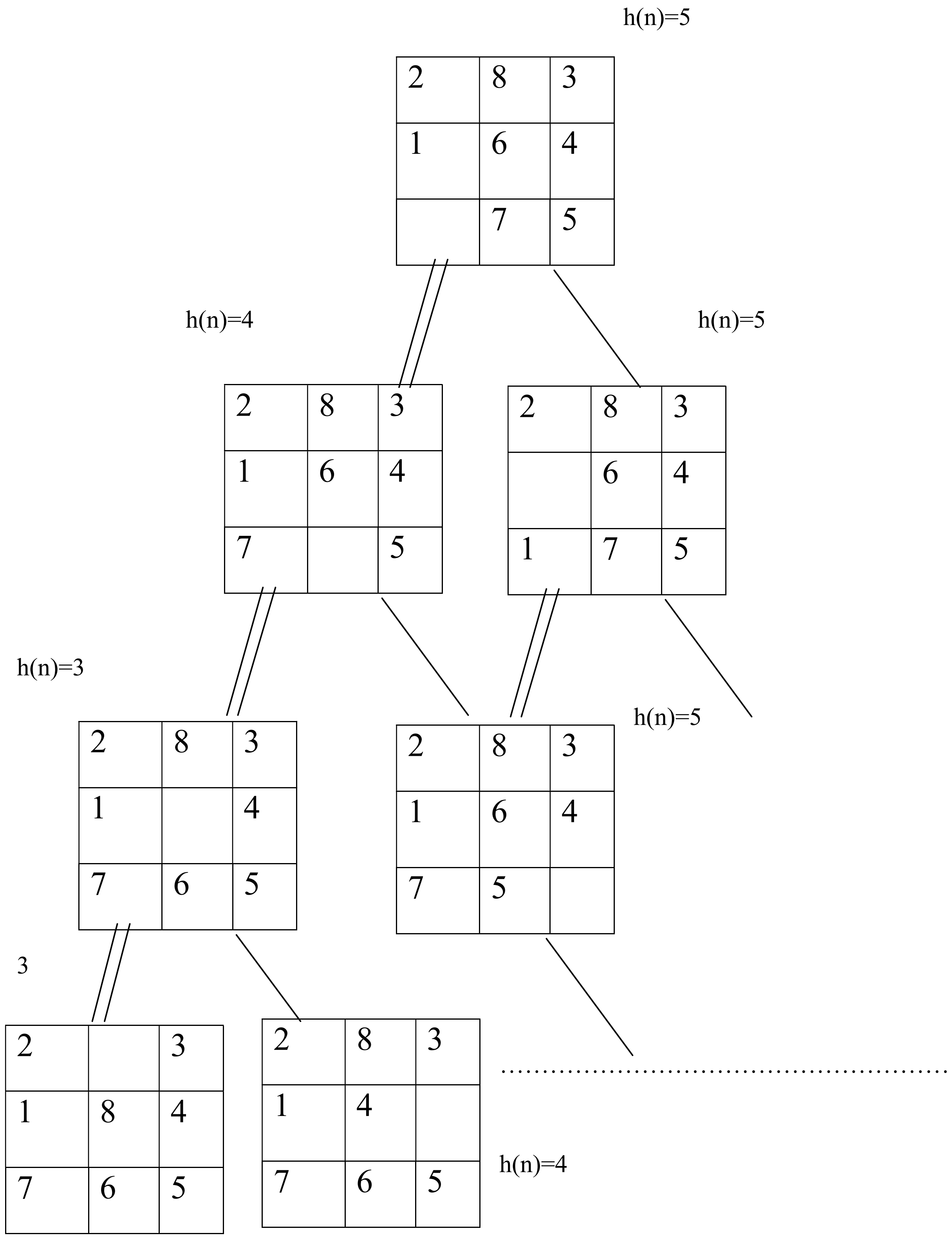
Fig. 5.1: Start state

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

Fig. 5.2: Goal state

We have to design a heuristic function for the above 8 puzzle problem. We design our heuristic function as h(n)=Minimization number of tiles out of place (with respect to goal).

We try to minimize h(n). The search space of the problem looks like



h(n)=5
h(n)=4
h(n)=5
h(n)=3
h(n)=5
3
h(n)=4

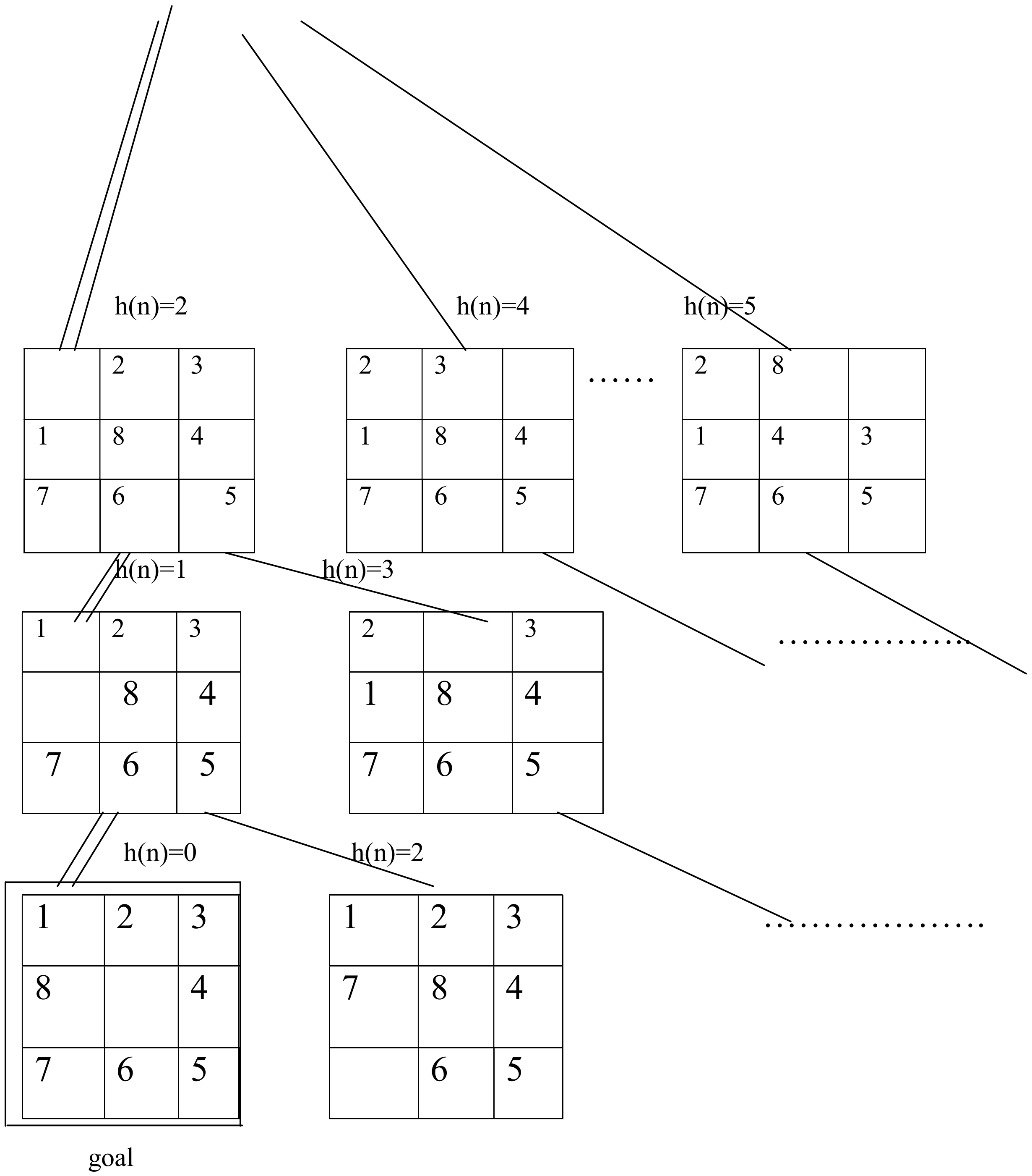


Fig. 5.3: 8 Puzzle problem

Thus our heuristic leads us to goal. Whenever in any step, there is a tie, we can choose any one of the state. For convenience, when a tie occurs, states which leads to goal are chosen but, students may try to choose a different one and check what happens. Design of heuristic function may vary from user to user. Heuristic function may be a maximization function or may be a minimizing one. It depends on the choice of user. Heuristic value of goal=0 thus, heuristic function measures the distance from goal node.

**Manhattan Distance Heuristic**-- Another heuristic for 8 puzzle problem is Manhattan distance heuristic. In this heuristic, the distance of a tile is measured by the sum of the difference of a tile in the X position and Y position. So, manhattan distance heuristic for start node of the above problem is h(n)=1+1+0+0+0+1+1+2=6. Only tile 8 has manhattan distance 2, because the difference from goal in X position is 1 and in Y position is 1. So sum is 2. All the rest of the tiles have manhattan distance 0 or 1.

### 5.3.1 Heuristic Search for OR graphs

In forward reasoning problem, we reach towards the goal state from a starting state. This class of algorithm when implemented with heuristic function is called heuristic search for OR graphs or the Best First Search algorithms.

In best first search algorithm, we start with a promising node (which has maximum or minimum fitness value according to the problem designed) and generate all its children. The fitness of each of the children is then examined and the most promising node among all the unexpanded nodes is selected for expansion. The most promising node is then expanded and the fitness of its offsprings are measured. Now, among all the unexpanded nodes, the most promising node is selected for expansion. This process continues until we reach to goal. The best first search algorithm is stated below:

**Procedure of best-First search**

```
Begin

Step1: Identify possible starting states. Evaluate them using heuristic function (f) and put them in a list L.

Step2: While L is not empty do

        Begin

            a) Chose node n from L that has minimum of f value. If there is a tie, then select a
               node randomly;
            b) If n is goal state
               Then return node n with its path from root node and
               Exit;
               Else
                   Remove n from list L and generate all children of n.    Add them in list L.
               End while;
               END.
```

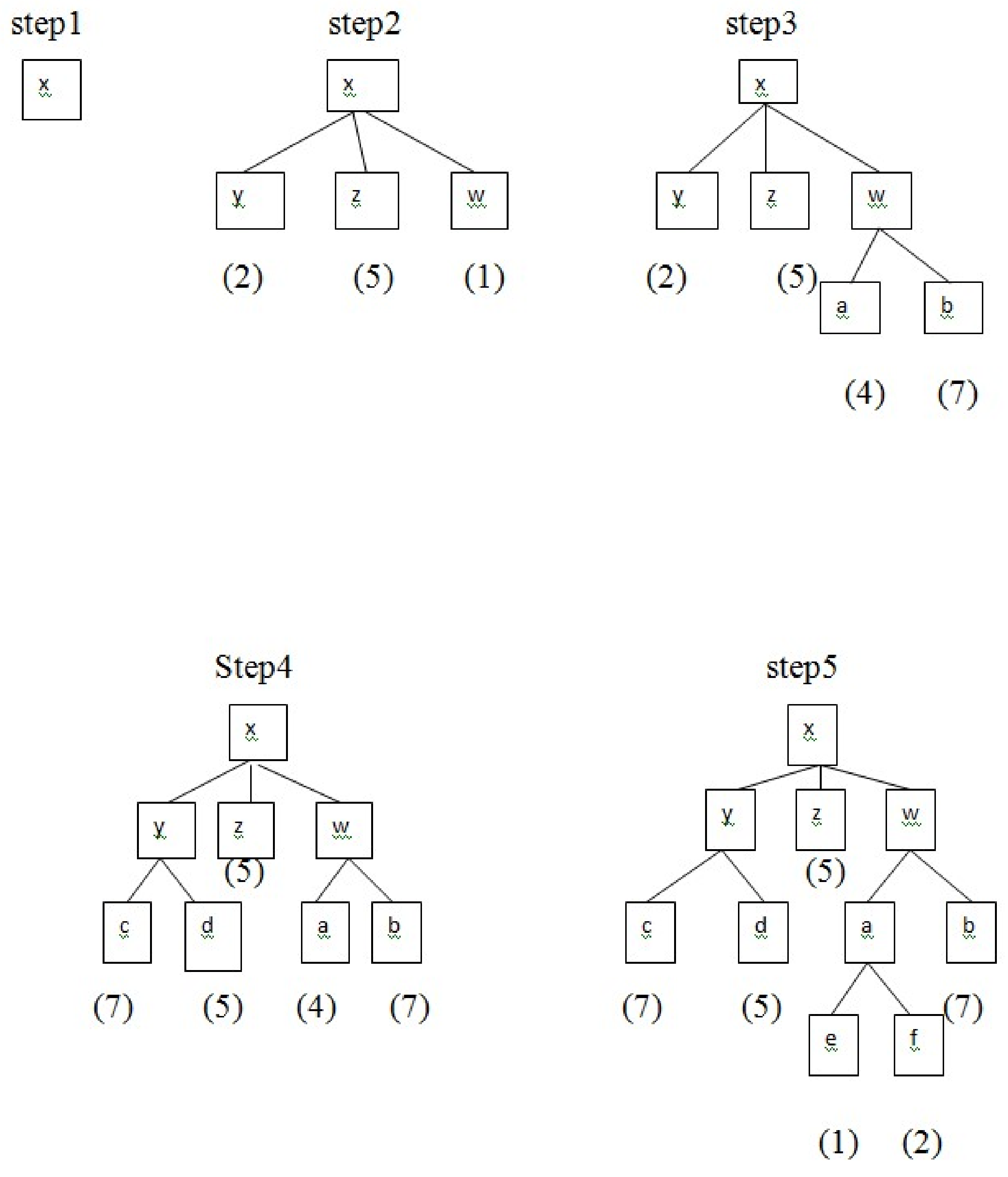


**Fig. 5.4**: Best first search

**Advantage of Best First search compared to hill climbing**

In best first search technique, all the unexpanded nodes are taken into consideration for finding the most promising node. In hill climbing, once a node is selected for expansion then there is no scope of backtracking if that path is not suitable. But, best first search, assures always a scope of choosing an alternative path if the current path is not leading towards goal.

Best first search is a class of algorithms and one member of this class is A* algorithm.

Two new definition are added for discussing A* algorithm.

**Definition 5.1**: A node is called **open** if the node has been generated and the heuristic function has been applied over it but the node has not been expanded yet.

**Definition 5.2:** A node is called **closed** if it has been expanded for generating offspring.

In A* algorithm, for evaluation of a node, two cost function are used. One is heuristic cost and another is generation cost.

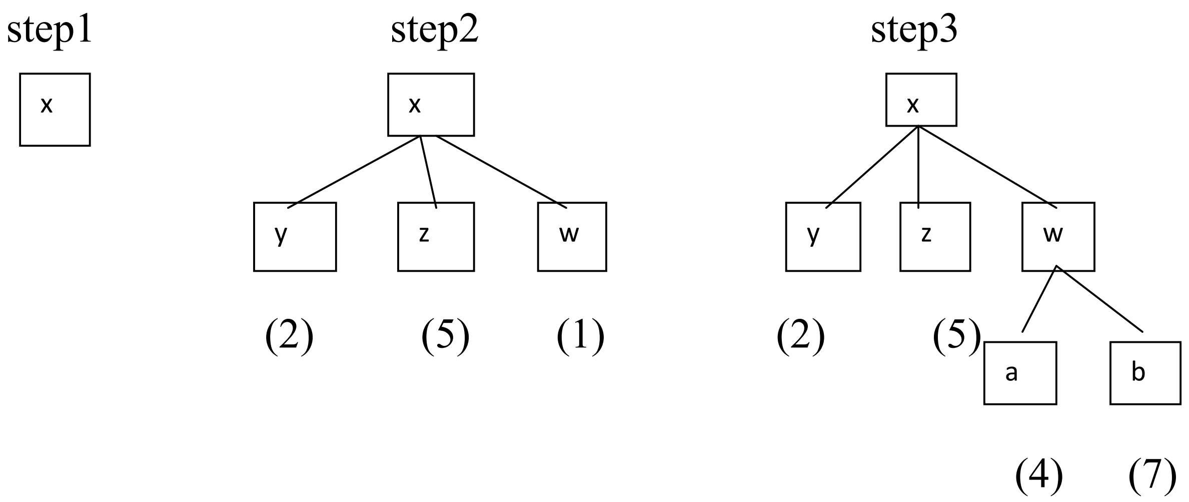
Heuristic cost: It measures the distance of the current node (x) with respect to the goal node and denoted by h(x).

Generation cost: it measures the distance of the current node (x) with respect to starting node and denoted by g(x).

Total cost of a node (x) is denoted as f(x)=g(x)+h(x).

Now g(x) can be easily measured as it is the distance from starting node but, distance from goal h(x) can be measured through prediction and is denoted as $h^{/}(x)$. Consequently, the predicted total cost is denoted by $f^{/}(x)$, where

$$f^{/}(x)=g(x)+ h^{/}(x).$$



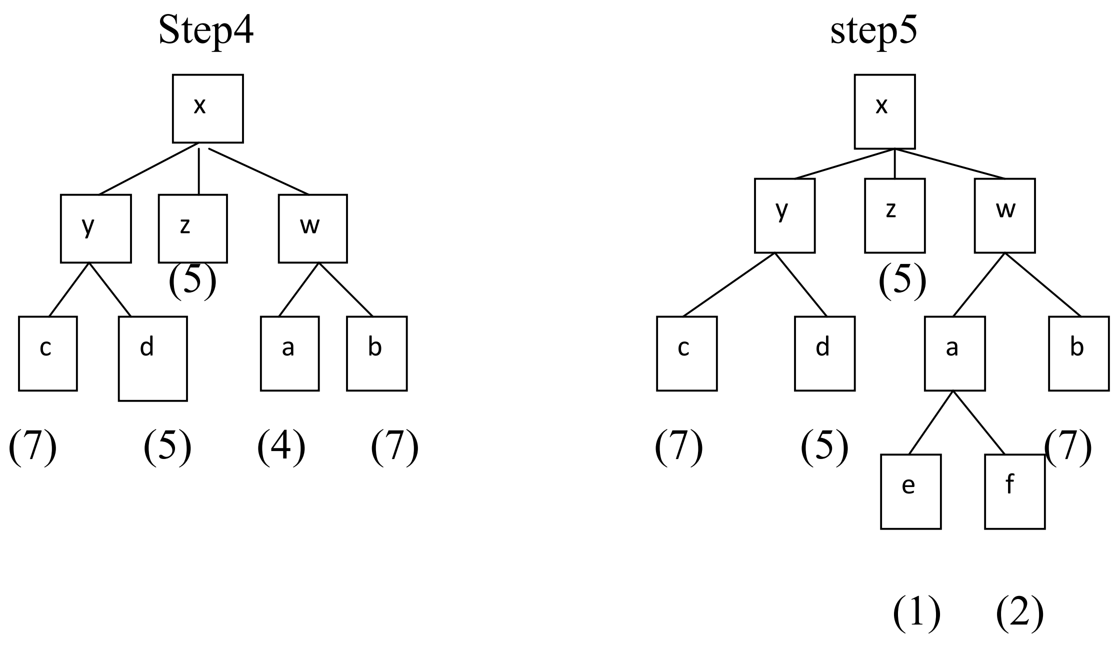


**Fig. 5.5:** Best First Search

Fig. 5.5 shows the procedure of best first search. Initially there is a single node x. so it is expanded. It generates 3 nodes y, z, and w. heuristic function is applied on each node and the values are 2, 5, and 1 respectively. Since, node w looks promising it is expanded next, generating two solutions a and b. Heuristic function is applied to a and b. Now, another path, node y looks promising. So, y is expanded next, generating c and d. they are evaluated. Now, c and d look less promising to node a, which lies in another path. So, a is expanded, generating e and f. This process continues till goal is found.

```
Procedure  A*
Begin
Step I:   Place a node n to open and measure its $f'(n)=g(n)+h'(n)$;
Step II:  Repeat (Until open is not empty )
          Begin
          If n = goal then stop and return n along with the path of n from starting node;
          Else do
          Begin
a) Remove n from open and place it under closed;
b) Generate the children of n;
c) If all children are new then add them to open and calculate their $f'$ and the path from root node
through back pointers;
```

d) If one or more children of n already exist as open nodes in the graph before their generation then those children must have multiple parents. Then compare their $f'$ values through current path and old paths and connect them through the shortest among them. Label the back pointers from the children of n to their parent.
e) If one or more children of n already exist as closed nodes before their generation, then they also have multiple parents. Calculate the shortest path by which $f'$ value of n is minimum. If the current path is selected, then the nodes in the sub tree rooted at the corresponding child of n should have revised $f'$. Label the back pointer from the children of n to their parent.

```
End
    End while;
  End
```

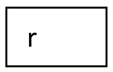
Example 5.2 There are to jugs containing 3 gallon and 5 gallon of water respectively. How do you measure 4 gallons of water in the 5 gallons jug? Solve using A* algorithm.

Let us consider
X= Amount of water in 3 gallon jug
Y= Amount of water in 5 gallon jug
P is an arbitrary node in the search space
We design heuristic like that

$h'(p) = 2$, when $0<X<3$ AND $0<Y<5$
$= 4$, when $0<X<3$ OR $0<Y<5$
$= 10$, when i) $X=0$ AND $Y=0$
or ii) $X=3$ AND $Y=5$
$= 8$, when i) $X=0$ AND $Y=5$
or ii) $X=3$ AND $Y=0$

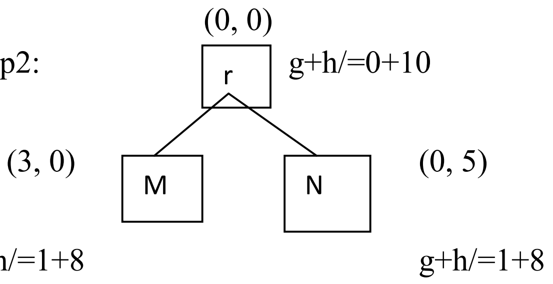
We assume $g(p)=0$ for root node and $g(p)=n$, if the number of parent nodes=n for pth node, starting from root node. Discovery of the search space using A* Algorithm is illustrated below

Step1:



(0, 0) $g+h'=0+10$

Step2:



Step3:

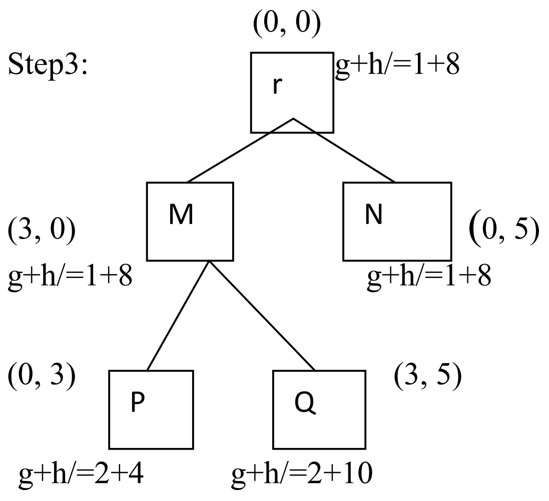


**Fig: 5.6:** A* algorithm

In Step I, we have node r whose g+h$^{/}$=0+10=10. In the next step, we have two terminal nodes M and N, where M and N have same g+h$^{/}$ value. We choose randomly node M for expansion and generate nodes P and Q with g+h$^{/}$=6 and g+h$^{/}$=12 respectively. Now out of the three nodes N, P and Q, P has the minimum value of f$^{/}$. So, we select node P for expansion. The process thus continues till goal is found.

**Behaviour of A* Algorithm**

**Underestimation**

If we can guarantee that h, never over estimates actual value from current to goal, then A* algorithm is able to find an optimal path to a goal, if it exists.

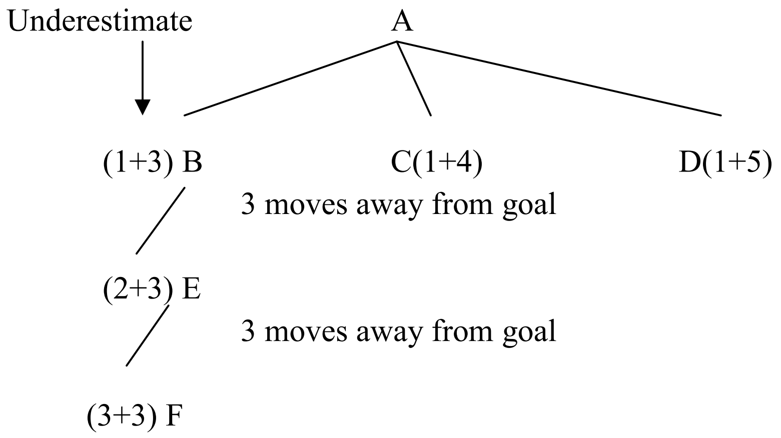


**Fig. 5.7:** Underestimation of A*

In the above example, node B has the smallest f$^{/}$(g+h$^{/}$) value. So, it is expanded first. Suppose, it has only one successor E whose f$^{/}$(E) is same as f$^{/}$(c) and equal to 5. If we FOLlow the current path, then break the tie by choosing node E. we expand E and generate node F. but,

$f'(F)=6$ which is greater than $f'(c)$. So, we choose c next. Thus, we see that by underestimating h(B) we have wasted some effort.

**Overestimation**

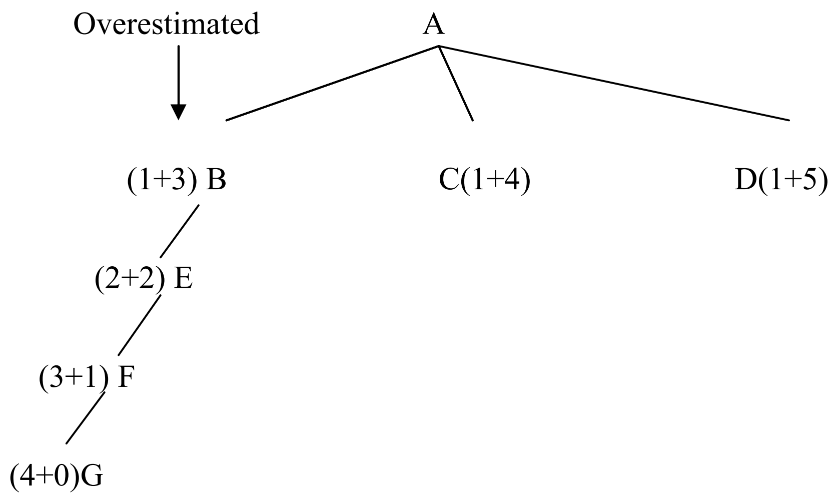


**Fig. 5.8:** Overestimation

We expand B which have minimum $f'(g+h')$ value. Next, we expand node E and then node F and get node G. The length of the solution path is 4. But, if there is a direct path from node D to node G, we will never find it.
We have overestimated $h'(D)$ which make D so bad that we may explore the worst path by ignoring D.

**Properties of Best first search Algorithms**

a) Completeness: An algorithm is complete, if it is able to find a solution if it exists.
b) Admissibility: An algorithm is admissible, if it is able to find an optimal solution, if it exists.
c) Dominance: An algorithm A1 is said to dominate another algorithm A2, if every node expanded by A1 is also expanded by A2.
d) Optimality: An algorithm is optimal over a class of algorithms, if the algorithm dominates all members of the class.

**A* Search: properties**

. A* is admissible under the FOLlowing conditions:

In the state space graph

- Each node has finite number of successors
- Every arc in the graph has a cost greater than some £>0
- Heuristic function, for every node n,
  $h(n)<=h^*(n)$

A* is also complete under the above conditions.

A* is optimally efficient for a given heuristic----------It means that no other optimal algorithm will expand fewer nodes to find a solution than A*.

A heuristic is consistent if:
$h(n)<=cost(n, n^{/})+h(n^{/})$. This comes under properties of heuristic in the next page.
If a heuristic h is consistent, the f values along a path will be non decreasing.

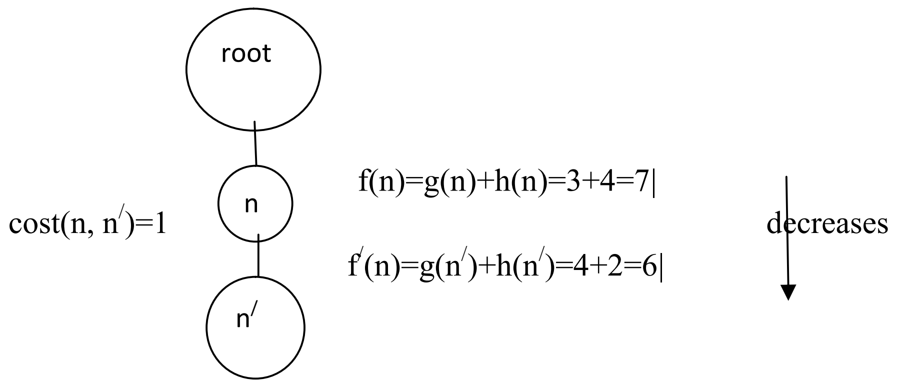


**Fig. 5.9:** f values decreases from n to $n^{/}$

In the fig. 5.9 $h(n)=4$, but $cost(n, n^{/})+h(n^{/})=1+2=3$

So, $h(n)<cost(n, n^{/})+h(n^{/})$. So this heuristic is consistent.
f(n’)= estimated distance from root node to goal through $n^{/}$
= actual distance from root node to n+ cost of n to $n^{/}$+ estimated distance from n’ to goal.
So, $f(n^{/})=g(n)+ cost(n, n^{/})+h(n^{/})$
$>=g(n)+h(n)$ [$cost(n, n^{/})+h(n^{/})>=h(n)$ for consistency of heuristic]
$=f(n)$
So, $f(n^{/})>=f(n)$. f should not decrease along a path.

**Proof of admissibility of A***

A* is admissible if it uses a monotone heuristic. A monotone heuristic is such that along any path f-cost never decreases. By making use of some trick, we can make the f value monotonic.
$f(m)=max(f(n),g(m)+h(m))$
Where G is the optimal goal state
C* is the optimal path cost
G1 is the suboptimal goal state: $g(G1)>C^*$
Let, A* has selected G1from OPEN for expansion. Let, n is a node on OPEN on an optimal path to G. Thus, $C^*>=f(n)$
As, n is not chosen for expansion over G1, $f(n)>=f(G1)$
G1 is a goal state $f(G1)=g(G1)$
Hence, $C^*>=g(G1)$
This is a contradiction. Thus A* could not select G1 for expansion before reaching the goal by an optimal path.

**Proof of completeness of A***

Let G= goal state
A* cannot reach a goal state only if there are infinitely many nodes where f(n)<=C*
Lemma: A* expands nodes in increasing order of their f values.
A* is **complete** and **optimal,** assuming an admissible and consistent heuristic function.

**Properties of Heuristics**

- **Dominance:**
  h2 is said to dominate h1 if h2 (n)>=h'(n)
  For any node n.
  A* will expand fewer nodes on average using h2 than h1.

Proof: Every node for which f(n)<C* will be expanded. This n is expanded whenever h(n)<f*----g(n)
Since, h2(n)>=h'(n), any node expanded using h2 will be expanded using h1.

- **Admissible:**
  An heuristic function h is said to be admissible if
  h(n)<=h*(n)
- **Monotonic:**
  A heuristic function is said to be monotonic if it satisfies
  h(n)<=cost(n, n/)+h(n') for all n, n/
  Where n' is successor of n.
- **Every consistent heuristic is also admissible**

Proof: We have
h(n)<=k(n, n/)+h(n/) [h is consistent]
Replace ɣ against n', we have
h(n')<=k(n, ɣ)+h(ɣ)
=>h(n)<=h*(n)
This is the condition for admissibility.

**Iterative Deepening A* Algorithm**
Iterative Deepening A* (or IDA*) algorithm combines the partial features of iterative deepening and A* algorithms together. The algorithm is described below,

```
            Procedure IDA*
        Begin
         Step I: Initialize current depth cut-off=1;
         Step II: Push the starting node at stack .initialize cut-off at next iteration c'=α;
         Step III: While (Stack is not empty) do
                              Begin
                               n= Stack[top];
if n= goal then report goal found and
return with path from start node;
                          Else do
                            Begin
for each child n' of n
                              If f(n/)<=C
                               Push n/ into Stack;
                             Else
```

```
                    $C^{/}=\min(c^{/}+f(n^{/}))$;
                 End for;
              End;
           End while;
Step IV: If Stack is empty AND $C^{/}=\alpha$, then stop and exit;
  Step V: If Stack is empty AND $C^{/}\neq\alpha$
                    C=C' and return Step II
           End.
```

Advantage of IDA* over A*

A* requires exponential amount of memory because of no restriction on depth cut-off.

IDA* expands a node n only when all of its children n' have $f(n^{/})$ value less than the cut off value C.

Thus IDA* saves a considerable amount of memory.

### 5.3.2 Heuristic Search on AND-OR Graphs

Backward reasoning problem is implemented through AND-OR graphs. Consider the FOLlowing problem which is represented through an AND-OR graph.

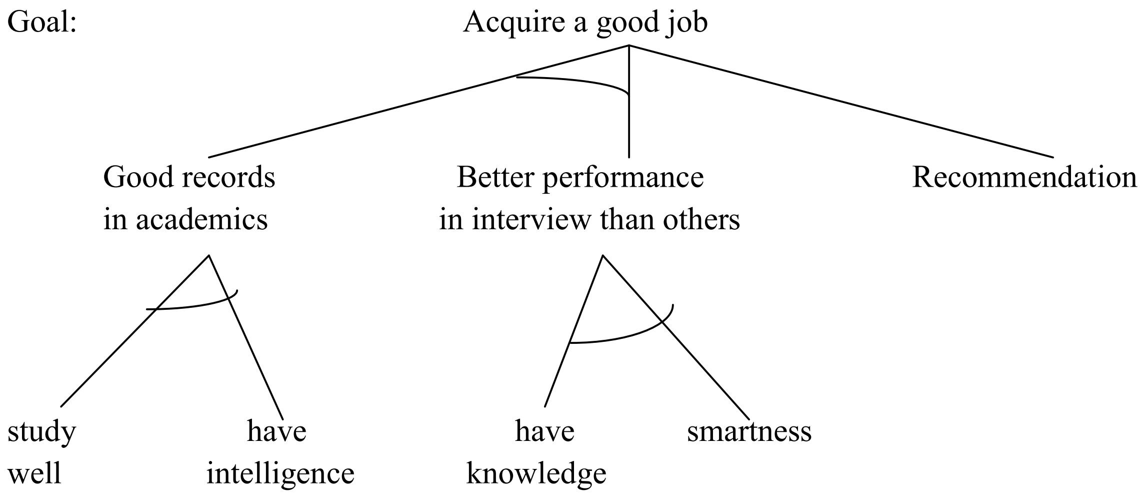


**Fig. 5.10:** AND-OR graph

Fig. 5.10 describes an AND-OR graph. Here the goal is "to get a good job" and the terminals of the graph describe the possible means to achieve the goal.

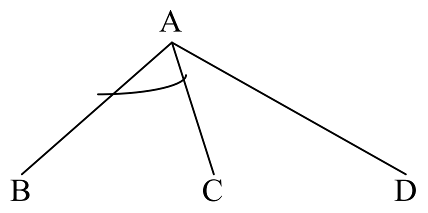
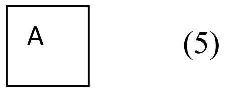


**Fig. 5.11:** Symbols of AND-OR graph

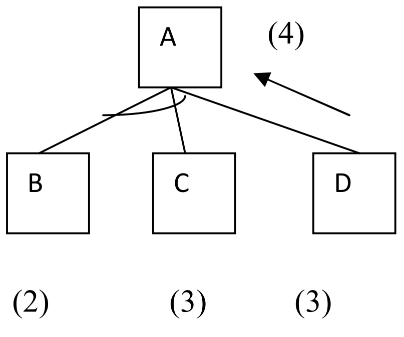
Fig. 5.11 represents symbols of AND-OR graph representing IF(B AND C) OR D Then A
AO* algorithm is an algorithm which is a heuristic search on AND-OR graph. AO* algorithm represented below:

Step I: A goal node is given. Find the possible means by which goal can be achieved;
Step II: Estimate h' values at the leaves and find the leaf with minimum h';
Cost of the Parent of the leaf=minimum (((Cost of OR Clause$_1$, Cost of OR Clause$_2$,…………, Cost of OR Clause$_n$)+1 ), ( (Cost of AND Clause+, Cost of AND Clause+,………… ................,+Cost of AND Clause$_n$)+n));
Children (n) with minimum $h^{/}(n)$ are calculated and pointer is attached to point from the Parent to its promising children;
Step III: One of the unexpanded OR clauses/ the set of unexpanded AND clauses, where the pointer points from its parent is now chosen for expansion and the h' value of the newly generated children are calculated. Recalculate the f1 of the parent or the parent of the parent of new children by propagating h' value up to the root through a least cost path. Thus, the pointers may be modified depending on the changed cost.

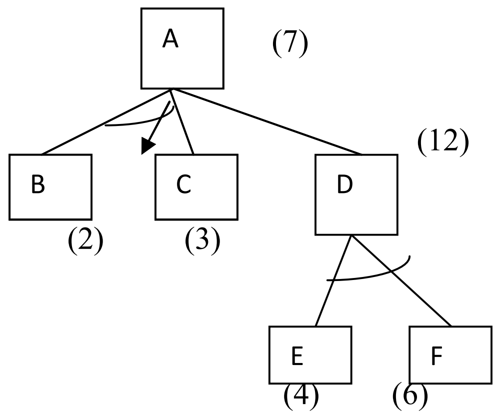
Step1:



Step2:



Step3:



Step4:

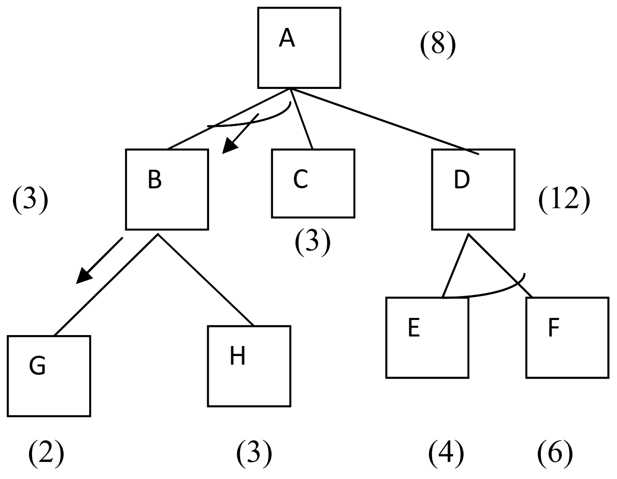


**Fig. 5.12:** AO* illustration

# KNOWLEDGE & REASONING

**Artificial intelligence** is a system that is concerned with the study of understanding, designing and implementing the ways, associated with knowledge representation to computers.
In any intelligent system, representing the knowledge is supposed to be an important technique to encode the knowledge.
The main objective of AI system is to design the programs that provide information to the computer, which can be helpful to interact with humans and solve problems in various fields which require human intelligence.

**Types of knowledge in AI**

**Depending on the type of functionality, the knowledge in AI is categorized as:**

**1. Declarative knowledge**

The knowledge which is based on concepts, facts and objects, is termed as 'Declarative Knowledge'. It provides all the necessary information about the problem in terms of simple statements, either true or false.

**2. Procedural knowledge**

Procedural knowledge derives the information on the basis of rules, strategies, agendas and procedure. It describes how a problem can be solved. Procedural knowledge directs the steps on how to perform something.

**For example:** Computer program.

**3. Heuristic knowledge**

Heuristic knowledge is based on thumb rule. It provides the information based on a thumb rule, which is useful in guiding the reasoning process. In this type, the knowledge representation is based on the strategies to solve the problems through the experience of past problems, compiled by an expert. Hence, it is also known as **Shallow knowledge.**

**4. Meta-knowledge**

This type gives an idea about the other types of knowledge that are suitable for solving problem. Meta-knowledge is helpful in enhancing the efficiency of problem solving through proper reasoning process.

**5. Structural knowledge**

Structural knowledge is associated with the information based on rules, sets, concepts and relationships. It provides the information necessary for developing the knowledge structures and overall **mental model** of the problem.

**Issues in knowledge representation**

The main objective of knowledge representation is to draw the conclusions from the knowledge, but there are many issues associated with the use of knowledge representation techniques.

**Some of them are listed below:**

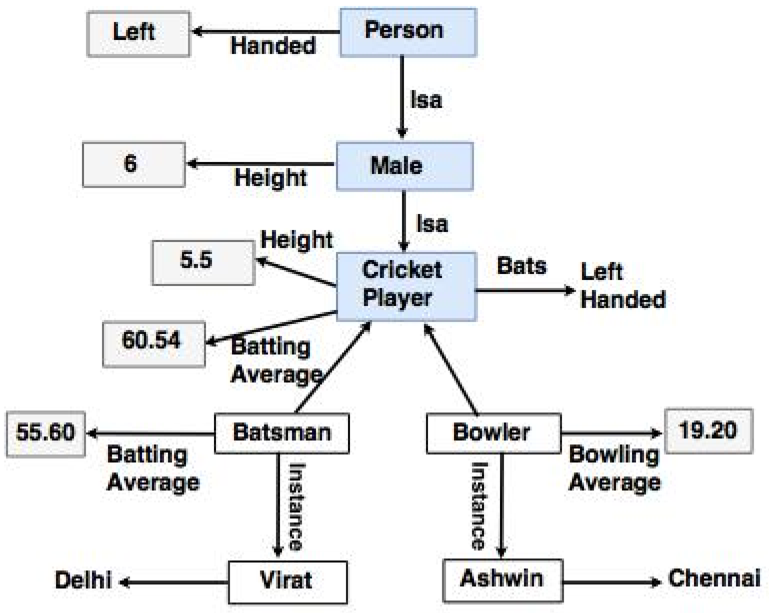


Fig: Inheratable Knowledge Representation

**Refer to the above diagram to refer to the following issues.**

**1. Important attributes**
There are two attributes shown in the diagram, **instance** and **isa.** Since these attributes support property of inheritance, they are of prime importance.

**2. Relationships among attributes**
Basically, the attributes used to describe objects are nothing but the entities. However, the attributes of an object do not depend on the encoded specific knowledge.

**3. Choosing the granularity of representation**
While deciding the granularity of representation, it is necessary to know the following:

**i.** What are the primitives and at what level should the knowledge be represented?

**ii.** What should be the number (small or large) of low-level primitives or high-level facts?

High-level facts may be insufficient to draw the conclusion while Low-level primitives may require a lot of storage.
**For example:** Suppose that we are interested in following facts:
John spotted Alex.

Now, this could be represented as "Spotted (agent(John), object (Alex))"

Such a representation can make it easy to answer questions such as: Who spotted Alex?

Suppose we want to know : "Did John see Sue?"
Given only one fact, user cannot discover that answer.

Hence, the user can add other facts, such as "Spotted (x, y) → saw (x, y)"

**4. Representing sets of objects.**
There are some properties of objects which satisfy the condition of a set together but not as individual;

**Example: Consider the assertion made in the sentences:**
"There are more sheep than people in Australia", and "English speakers can be found all over the world."
These facts can be described by including an assertion to the sets representing people, sheep, and English.

**Finding the right structure as needed**
To describe a particular situation, it is always important to find the access of right structure. This can be done by selecting an initial structure and then revising the choice.

While selecting and reversing the right structure, it is necessary to solve following problem statements. **They include the process on how to:**

- Select an initial appropriate structure.
- Fill the necessary details from the current situations.
- Determine a better structure if the initially selected structure is not appropriate to fulfill other conditions.
- Find the solution if none of the available structures is appropriate.
- Create and remember a new structure for the given condition.
- There is no specific way to solve these problems, but some of the effective knowledge representation techniques have the potential to solve them.

# USING PREDICATE LOGIC

**Logic Representation**
Facts are the general statements that may be either True or False. Thus, logic can be used to represent such simple facts.

**To build a Logic-based representation:**
User has to define a set of primitive symbols along with the required semantics.
The symbols are assigned together to define legal sentences in the language for representing TRUE facts.
New logical statements are formed from the existing ones. The statements which can be either TRUE or false but not both , are called propositions. A declarative sentence expresses a statement with a proposition as content;
Example: The declarative "Cotton is white" expresses that Cotton is white. So, the sentence "Cotton is white" is a true statement.

**What is Propositional Logic (PL)?**
Propositional logic is a study of propositions.
Each proposition has either a true or a false value but not both at a time.
Propositions is represented by variables.
For example: Symbols 'p' and 'q' can be used to represent propositions.
There are two types of propositions:
1. Simple Preposition
2. compound Prepositions.

1. A simple preposition: It does not contain any other preposition.
For example: Rocky is a dog.

2. A compound preposition: It contains more than one prepositions.
For example: Surendra is a boy and he likes chocolate.

**Connectives and the truth tables of compound prepositions are given below:**
Consider 'p' and 'q' are two prepositions then,

**1. Negation** (¬p) indicates the opposite of p.
**Truth table for negation:**

| p | ¬p |
|---|---|
| 0 | 1 |
| 1 | 0 |

**2. Conjunction** (p ∧ q) indicates that p and q both and are enclosed in parenthesis. So, p and q are called conjuncts .

**Truth table for conjunction:**

| p | q | p∧q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**3. Disjunction** (p ∨ q) indicates that either p or q or both are enclosed in parenthesis. Thus, p and q are called disjuncts.

**Truth table for disjunction:**

| p | q | p∨q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**4. Implication** (p ⇒ q) consists of a pair of sentences separated by the ⇒ operator and enclosed in parentheses. The sentence to the left of the operator is called as an antecedent, and the sentence to the right is called as the consequent.

**Truth table for implication:**

| p | q | p →q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**5. Equivalence** (p ⇔ q) is a combination of an implication and a reduction.

**Truth table for Equivalence:**

| p | q | p ↔q |
|---|---|---|
| 0 | 0 | 1 |

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

PROBLEM DISCUSSION:

Q1. “If SRK plays hero’s part, then the movie will be hit, if the plot is not too melodramatic. If SRK plays the hero’s part, the plot will not be too melodramatic.
Therefore, if SRK plays hero’s part, the movie will be a hit.”
Is it a valid argument?
SAMPLE ANSWER:
Sentence 1: SRK plays hero’s part
Sentence 2: The movie will be hit
Sentence 3: the plot will not be too melodramatic

Premise 1: $A\to(C\to B)$
Premise 2: $A\to C$
Conclusion: $A\to B$

Note: *An argument is said to be valid, if the conclusion is true, whenever the premises are true.*

| A | B | C | $C\to B$ | $A\to(C\to B)$ | $A\to C$ | $A\to B$ | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | √ |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | √ |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | √ |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | √ |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | √ |

Through the above table we can see that, whenever premises are true, conclusions are true. So the given argument is VALID.

Sample questions to be practiced:

1. Verify if the following wff’s are tautologies:
   a) $(A\to B)\to((A\to\neg B)\to\neg A)$

    b) (A→)→((A∨C)→(B∨C))
2. Find whether the following wff's are equivalent:
    a) A→(B∨C), and ¬A∨B∨C
    b) A∨(¬B→C), and A∨¬(B→C)

**First Order Predicate Logic**
The prepositional logic only deals with the facts, that may be true or false.
The first order logic assumes that the world contains objects, relations and functions.

**Syntax for first order logic:**
In prepositional logic, every expression is a sentence that represents a fact.
First order logic includes the sentences along with terms which can represent the objects.
Constant symbols, variables and function symbols are used to build terms, while quantifiers and predicate symbols are used to build the sentences.
**Syntax:**

| | |
|---|---|
| Constants | A, B, C..... |
| Functions | Size, Color |
| Variable | x, a |
| Terms | Constant, variable or function(Term..) |
| Predicates | True, False |
| Quantifiers | ∀, ∃ |
| Atomic sentences | Predicate, Predicate(Term,…), Term=Term |
| Sentences | ¬ Sentence, Sentence ∨ Sentence, Sentence ∧ Sentence, Sentence ⇒ Sentence, Sentence ⇔ Sentence, Quantifier Variable,… Sentence |

**Semantics**:

Lets understand with an example,
Consider the sentence "Elephants are big".There are many ways to represent this sentence.
HasSize(Elephant, Big)
SizeOF(Elephant)= Big

Lets introduce a new syntax,
IsEqual(SizeOf(Elephant, Big), this states that a object Elephant is big, which is a useless fact in any reasoning process about the Elephants in general. So let us represent that all Elephants are big.
So, we can find FOPL statement as,
All things that are Elephants are big.
For all things x, for which x is an Elephant, x is big.

For all things x, if x is a Elephant, then x is big.
Finally the FOPL will be written as.
∀x: Elephant (x) → Big(x)

SAMPLE EXAMPLES: Conversion of sentences into Predicate Logic

1. Marcus was a man.
   Man(Marcus)
2. Marcus was a Pompeian.
   P(Marcus)
3. All Pompeians were Romans.
   ∀x: P(x) → Rom(x)
4. Caesar was a Ruler.
   Rul(Caesar)
5. All Romans were either loyal to Romans or hated him.
   ∀x: Rom (x) → Loy(x, Caesar) ∨ H(x, Caesar)
6. Everyone is loyal to someone.
   ∀x, ∃y: Loy(x,y)
7. People only try to assassinate rulers they are not loyal to.
   ∀x, ∀y: Man(x) ∧ Rul(y) ∧ Try(x,y) → ¬Loy(x,y)
8. Marcus tried to assassinate Caesar.
   Try(Marcus, Caesar)

For the above eight sentences, can we prove that ¬Loy(Marcus,Caesar) ?
Yes, we can.

$$\begin{tikzcd}
\text{Man(Marcus)} \arrow[d, "(8)"] \\
\text{Man(Marcus)} \wedge \text{Try(Marcus, Caesar)} \arrow[d, "(4)"] \\
\text{Man(Marcus)} \wedge \text{Try(Marcus, Caesar)} \wedge \text{Rul(Caesar)} \arrow[d, "\text{(7, substitution)}"] \\
\neg\text{Loy(Marcus,Caesar)}
\end{tikzcd}$$

Hence it is proved.
Now, if it is told that prove the same thing through Resolution Refutation method, then the technique is different and we must know what is Resolution.

In mathematical logic and automated theorem proving, resolution is a rule of inference leading to a refutation theorem-proving technique for sentences in propositional logic and first-order logic.

In other words, iteratively applying the resolution rule in a suitable way allows for telling whether a propositional formula is satisfiable and for proving that a first-order formula is unsatisfiable. Attempting to prove a satisfiable first-order formula as unsatisfiable may result in a nonterminating computation; this problem doesn't occur in propositional logic.

**Resolution in propositional logic**

Resolution rule

The resolution rule in propositional logic is a single valid inference rule that produces a new clause implied by two clauses containing complementary literals. A literal is a propositional variable or the negation of a propositional variable. Two literals are said to be complements if one is the negation of the other (in the following ¬c is taken to be the complement to *c*. The resulting clause contains all the literals that do not have complements. Formally:

$$\frac{\square 1 \cup \square 2 \cup \ldots . \square, \square 1 \cup \square 2 \cup \ldots \neg c}{\square 1 \cup \square 2 \cup \ldots . . \cup \square 1 \cup \square 2 \cup \ldots}$$

where

all $a_i$, $b_i$ and c are literals,

the dividing line stands for "entails".

The above may also be written as:

$$\frac{(\neg \square 1 \cap \neg \square 2 \cap \ldots .) \rightarrow \square, \square \rightarrow (\square 1 \cap \square 2 \cap \ldots)}{(\neg \square 1 \cap \neg \square 2 \cap \ldots .) \rightarrow (\square 1 \cap \square 2 \cap \ldots)}$$

The clause produced by the resolution rule is called the resolvent of the two input clauses. It is the principle of consensus applied to clauses rather than terms.

When the two clauses contain more than one pair of complementary literals, the resolution rule can be applied (independently) for each such pair; however, the result is always a tautology.

Modus ponens can be seen as a special case of resolution (of a one-literal clause and a two-literal clause).

$$\frac{\square \rightarrow \square, \square}{\square}$$

is equivalent to

$$\frac{\neg \square \cup \square, \ \square}{\square}$$

**A resolution technique**

When coupled with a complete search algorithm, the resolution rule yields a sound and complete algorithm for deciding the satisfiability of a propositional formula, and, by extension, the validity of a sentence under a set of axioms.

This resolution technique uses proof by contradiction and is based on the fact that any sentence in propositional logic can be transformed into an equivalent sentence in conjunctive normal form.[4] The steps are as follows.

- All sentences in the knowledge base and the negation of the sentence to be proved (the conjecture) are conjunctively connected.
- The resulting sentence is transformed into a conjunctive normal form with the conjuncts viewed as elements in a set, S, of clauses.[4]
- For example $(A_1 \lor A_2) \land (B_1 \lor B_2 \lor B_3) \land (C_1)$ gives rise to the set $S=(A_1 \lor A_2),(B_1 \lor B_2 \lor B_3),(C_1)$ .
- The resolution rule is applied to all possible pairs of clauses that contain complementary literals. After each application of the resolution rule, the resulting sentence is simplified by removing repeated literals. If the sentence contains complementary literals, it is discarded (as a tautology). If not, and if it is not yet present in the clause set S, it is added to S, and is considered for further resolution inferences.
- If after applying a resolution rule the empty clause is derived, the original formula is unsatisfiable (or contradictory), and hence it can be concluded that the initial conjecture follows from the axioms.
- If, on the other hand, the empty clause cannot be derived, and the resolution rule cannot be applied to derive any more new clauses, the conjecture is not a theorem of the original knowledge base.

One instance of this algorithm is the original Davis–Putnam algorithm that was later refined into the DPLL algorithm that removed the need for explicit representation of the resolvents.

This description of the resolution technique uses a set S as the underlying data-structure to represent resolution derivations. Lists, Trees and Directed Acyclic Graphs are other possible and common alternatives. Tree representations are more faithful to the fact that the resolution rule is binary. Together with a sequent notation for clauses, a tree representation also makes it clear to see how the resolution rule is related to a special case of the cut-rule, restricted to atomic cut-formulas. However, tree representations are not as compact as set or list representations, because they explicitly show redundant sub derivations of clauses that are used more than once in the derivation of the empty clause. Graph representations can be as compact in the number of clauses as list representations and they also store structural information regarding which clauses were resolved to derive each resolvent.

So, in a nutshell, a Resolution process is a simple iterative process, at each step, two clauses called the parent clauses compared (i.e. resolved), yielding a new clause that has been inferred from them.

So, we must know how to convert a predicate logic into its equivalent clauses. There are few thumb rules to be followed sequentially while converting a predicate logic into a clause. They are as follows:

1) Eliminate the implication sign ($\rightarrow$) with the following identity:

   $a \rightarrow b$ is equivalent to $\neg a \lor b$.

2) Reduce the scope of each negation symbol (¬) with the help of the following identities:

   $\neg(P \lor Q) = \neg P \land \neg Q$

   $\neg(P \land Q) = \neg P \lor \neg Q$

   $\neg(\forall x)\ w(x) = (\exists x) \neg\ w(x)$

   $\neg(\exists x)\ w(x) = (\forall x) \neg\ w(x)$

3) Standarize variables

4) i) Eliminate the existential quantifier with the help of SKOLEM function.

   e.g. ∀x, ∃y: Fatherof(y, x) can be replaced as follows:

   ∀x: Fatherof(**f(x)**, x) //where f(x) is SKOLEM function

   ii) If the existential quantifier variable doesn't appear within the scope of any universally quantifier variable, then the SKOLEM function will be a constant.

   e.g. ∃y: P(y) can be converted into P(A), where A is a constant.

5) Put the wff in prenex form.

6) Put the matrix in Conunctive Normal Form using the Distributive laws:

   e.g. $P \lor (Q \land R)$ is equivalent to $(P \lor Q) \land (P \lor R)$

7) Drop the universal qquantification.

8) Eliminate the conjunctions and break the entire form into a set of clauses.

9) Rename the variable so that no variable name may appear in more than one clause.

**Example study:**

**Question:**

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a Ruler.
5. All Romans were either loyal to Romans or hated him.
6. Everyone is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.

Prove using Resolution that ¬Loy(Marcus, Caesar).

**Answer:**

**Step 1: Convert all the sentences into predicate logic.**

**Step 2: Convert all the predicate logics into its equivalent clauses.**

1. Man(Marcus)
2. P(Marcus)
3. ¬P(x) ∨ Rom(x)

4. Rul(Caesar)
5. ¬Rom $(x_2)$ ∨ Loy($x_2$, Caesar) ∨ H($x_2$, Caesar)
6. Loy($x_3$,$y_3$)
7. ¬Man(x4) ∨¬ Rul(y1) ∨¬ Try(x4,y1) ∨ ¬Loy(x4,y1)
8. Try(Marcus, Caesar)

**Step 3:** Whatever to proof, make negation of the whole fact and assume it as a true value and try to prove it.
**Step 4:** Resolve accordingly.

# REPRESENTING KNOWLEDGE USING RULES

Knowledge is understanding of a subject area. Intelligence is the way of applying knowledge. So, at first we must understand the basic differences of data, information and knowledge.

**Data:** Primitive verifiable facts. Example: name of novels available in a library.

**Information:** Analyzed data. Example: The novel that is frequently asked by the members of library is "Harry Potter and the Chamber of Secrets".

**Knowledge:** Analyzed information that is often used for further information deduction. Example: Since the librarian knows the name of the novel that is frequently asked by members, s/he will ask for more copies of the novel the next time s/he places an order.

Unlike human mind, computers cannot acquire and represent knowledge by themselves. Therefore we need to represent knowledge properly.

There are different types of knowledge. Some of them are:

- Procedural Knowledge
- Declarative Knowledge
- Meta – Knowledge
- Heuristic Knowledge
- Structural Knowledge

**Procedural knowledge** is the knowledge of how to perform, or how to operate. Names such as know-how are also given. It is said that one becomes more skilled in problem solving when he relies more on procedural knowledge than declarative knowledge.

Example: How to cook vegetable or how to prepare a particular dish is procedural knowledge.

**Declarative knowledge** is defined as the factual information stored in memory and known to be static in nature. Other names, e.g. descriptive knowledge, propositional knowledge, etc. are also given. It is the part of knowledge which describes how things are. Things/events/processes, their attributes, and the relations between these things/events/processes and their attributes define the domain of declarative knowledge.

Example: The first step in cooking a vegetable is chopping it.

So, the difference between Procedural and Declarative knowledge can be defined as follows:

| **Procedural knowledge** | **Declarative knowledge** |
|---|---|
| high efficiency | higher level of abstraction |
| low modifiability | good modifiability |
| low cognitive adequacy (better for knowledge engineers) | good cognitive matching (better for domain experts and end-users) |

|  |  |
|---|---|
| Ex: Process of planting herbs | Ex: Knowing something about herbs |

**Meta - Knowledge**: knowledge about other types of knowledge.

Example: bibliographic data, catalogue of books

May be used to reveal patterns in research, relationship between researchers and identify contradictory results

**Heuristic Knowledge**: rules of thumb based on previous experience, awareness of approaches that are likely to work but which are not guaranteed.

Example: Knowledge about the web navigation habits of an individual

An educated guess for example, about the search needs of a person

**Forward vs Backward Reasoning:**

In Artificial intelligence, the purpose of the search is to find the path through a problem space. There are two ways to pursue such a search that are forward and backward reasoning.

The solution of a problem generally includes the initial data and facts in order to arrive at the solution. These unknown facts and information is used to deduce the result. For example, while diagnosing a patient the doctor first check the symptoms and medical condition of the body such as temperature, blood pressure, pulse, eye colour, blood, etcetera. After that, the patient symptoms are analyzed and compared against the predetermined symptoms. Then the doctor is able to provide the medicines according to the symptoms of the patient. So, when a solution employs this manner of reasoning, it is known as **forward reasoning**.

The **backward reasoning** is reverse of forward reasoning in which goal is analyzed in order to deduce the rules, initial facts and data. We can understand the concept by the similar example given in the above definition, where the doctor is trying to diagnose the patient with the help of the inceptive data such as symptoms. However, in this case, the patient is experiencing a problem in his body, on the basis of which the doctor is going to prove the symptoms. This kind of reasoning comes under backward reasoning.

The significant difference between both of them is that forward reasoning starts with the initial data towards the goal. Conversely, backward reasoning works in opposite fashion where the purpose is to determine the initial facts and information with the help of the given results.

| **Basis For Comparison** | **Forward Reasoning** | **Backward Reasoning** |
|---|---|---|
| Basic | Data-driven | Goal driven |
| Begins with | New Data | Uncertain conclusion |
| Objective is to find the | Conclusion that must follow | Facts to support the conclusions |
| Type of approach | Opportunistic | Conservative |
| Flow | Incipient to consequence | Consequence to incipient |

# PROBABILISTIC REASONING

**Probabilistic Reasoning**

 Probability theory is used to discuss events, categories, and hypotheses about which there is not 100% certainty.

 We might write A→B, which means that if A is true, then B is true. If we are unsure whether A is true, then we cannot make use of this expression.

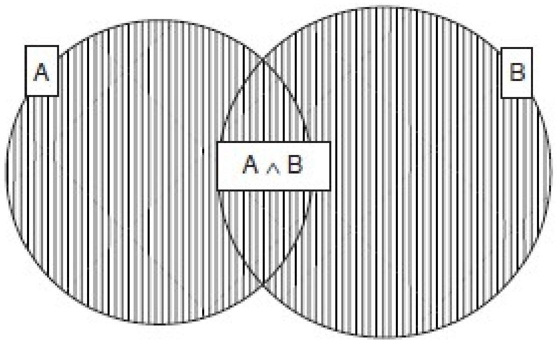
 In many real-world situations, it is very useful to be able to talk about things that lack certainty. For example, what will the weather be like tomorrow? We might formulate a very simple hypothesis based on general observation, such as "it is sunny only 10% of the time, and rainy 70% of the time". We can use a notation similar to that used for predicate calculus to express such statements:

$P(S) = 0.1$

$P(R) = 0.7$

The first of these statements says that the probability of S ("it is sunny") is 0.1. The second says that the probability of R is 0.7. Probabilities are always expressed as real numbers between 0 and 1. A probability of 0 means "definitely not" and a probability of 1 means "definitely so." Hence, $P(S) = 1$ means that it is always sunny.

Many of the operators and notations that are used in prepositional logic can also be used in probabilistic notation. For example, $P(\neg S)$ means "the probability that it is not sunny"; $P(S \wedge R)$ means "the probability that it is both sunny and rainy." $P(A \vee B)$, which means "the probability that either A is true or B is true," is defined by the following rule: $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$



The notation P(B|A) can be read as "the probability of B, given A." This is known as conditional probability—it is conditional on A. In other words, it states the probability that B is true, given that we already know that A is true. P(B|A) is defined by the following rule: Of course, this rule cannot be used in cases where $P(A) = 0$.

For example, let us suppose that the likelihood that it is both sunny and rainy at the same time is 0.01. Then we can calculate the probability that it is rainy, given that it is sunny as follows:

$$P(R|S) = \frac{P(R \wedge S)}{P(S)}$$

$$= \frac{0.01}{0.1}$$

$$= 0.1$$

The basic approach statistical methods adopt to deal with uncertainty is via the axioms of probability:
 Probabilities are (real) numbers in the range 0 to 1.
 A probability of *P*(*A*) = 0 indicates total uncertainty in *A*, *P*(*A*) = 1 total certainty and values in between some degree of (un)certainty.
 Probabilities can be calculated in a number of ways.
 Probability = (number of desired outcomes) / (total number of outcomes)
So given a pack of playing cards the probability of being dealt an ace from a full normal deck is 4 (the number of aces) / 52 (number of cards in deck) which is 1/13.
Similarly the probability of being dealt a spade suit is 13 / 52 = 1/4.

Conditional probability, *P*(*A*|*B*), indicates the probability of of event *A* given that we know event *B* has occurred.
 *A* Bayesian Network *is a* directed acyclic graph*:*
 A graph where the directions are links which indicate dependencies that exist between nodes.
 Nodes represent propositions about events or events themselves.
 Conditional probabilities quantify the strength of dependencies.
 Consider the following example:
 The probability, that my car won't start.
 If my car won't start then it is likely that
o The battery is flat or
o The staring motor is broken.
 In order to decide whether to fix the car myself or send it to the garage I make the following decision:
 If the headlights do not work then the battery is likely to be flat so i fix it myself.
 If the starting motor is defective then send car to garage.
 If battery and starting motor both gone send car to garage.
 The network to represent this is as follows:

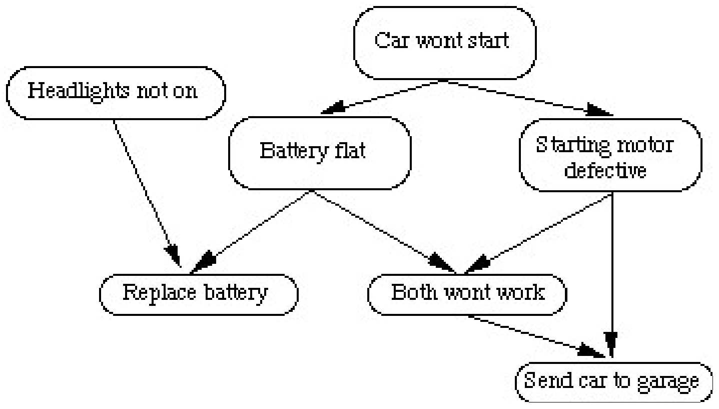


Fig. A simple Bayesian network

**Bayesian probabilistic inference**

 Bayes' theorem can be used to calculate the probability that a certain event will occur or that a certain proposition is true

 The theorem is stated as follows:

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

P(B) is called the prior probability of B. P(B|A), as well as being called the conditional probability, is also known as the posterior probability of B.

 $P(A \wedge B) = P(A|B)P(B)$

 Note that due to the commutativity of $\wedge$, we can also write

 $P(A \wedge B) = P(B|A)P(A)$

 Hence, we can deduce: $P(B|A)P(A) = P(A|B)P(B)$

 This can then be rearranged to give Bayes' theorem:

 Bayes Theorem states:

 This reads that given some evidence *E* then probability that hypothesis is true is equal to the ratio of the probability that *E* will be true given times the *a priori* evidence on the probability of and the sum of the probability of *E* over the set of all hypotheses times the probability of these hypotheses.

 The set of all hypotheses must be mutually exclusive and exhaustive.

 Thus to find if we examine medical evidence to diagnose an illness. We must know all the prior probabilities of find symptom and also the probability of having an illness based on certain symptoms being observed.

Bayesian networks are also called Belief Networks or Probabilistic Inference Networks.

- **Bayes' Theorem**

Let **S** be a sample space.

Let **A1, A2, ... , An** be a set of mutually exclusive events from **S**.

Let **B** be any event from the same **S**, such that **P(B) > 0**.

Then Bayes' Theorem describes following two probabilities :

$$P(A_k|B) = \frac{P(A_k \cap B)}{P(A_1 \cap B) + P(A_2 \cap B) + ----+ P(A_n \cap B)}$$ and

by invoking the fact $P(A_k \cap B) = P(A_k).P(B|A_k)$ the probability

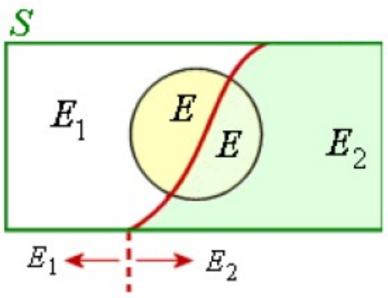
$$P(A_k|B) = \frac{P(A_k).P(B|A_k)}{P(A_1).P(B|A_1) + P(A_2).P(B|A_2) + ----+ P(A_n).P(B|A_n)}$$

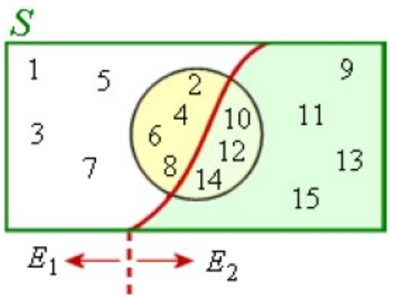
**Application Of Bayes Therom:**

‡ Let **S** be a sample space.

‡ Let **E1** and **E2** be two mutually exclusive events forming a partition of the sample space **S**

‡ Let **E** be any event of the sample space such that **P(E) ≠ 0**.





**Recall from Conditional Probability**

The notation **$P(E_1 | E)$** means "the probability of the event **$E_1$** given that **E** has already occurred".

**Recall from Conditional Probability**

The notation **$P(E_1 \mid E)$** means "the probability of the event **$E_1$** given that **E** has already occurred".

- ‡ The sample space **S** is described as "the integers 1 to 15" and is partitioned into :

  **$E_1$ = "the integers 1 to 8"** and

  **$E_2$ = "the integers 9 to 15".**

- ‡ If **E** is the event "**even number**" then the probabilities for the situation described by Baye's Theorem can be calculated in two ways, both giving same results.

$$P(E_1|E) = \frac{P(E_1 \cap E)}{P(E_1 \cap E) + P(E_2 \cap E)} = \frac{4/15}{(4/15) + (3/15)} = 4/7$$

$$P(E1|E) = \frac{P(E_1).P(E|E_1)}{P(E_1).P(E|E_1) + P(E_2).P(E|E_2)} = \frac{8/15 \times 4/8}{(8/15 \times 4/8) + (7/15 \times 3/15)} = 4/7$$

Thus Bayes' Theorem can be extended for Mutually Exclusive Events as :

$$P(Ei \mid E) = \frac{P(Ei \cap E)}{P(E1 \cap E) + P(E2 \cap E) + \ldots\ldots + P(Ek \cap E)}$$

**Clinical Example:**

In a clinic, the probability of the patients having HIV virus is **0.15**.

A blood test done on patients :

If patient has virus, then the test is **+ve** with probability **0.95**.

If the patient does not have the virus, then the test is **+ve** with probability **0.02.**

Assign labels to events : **H** = patient has virus; **P** = test +ve

Given : **$P(H) = 0.15$ ; $P(P|H) = 0.95$ ; $P(P|\neg H) = 0.02$**

Find :

If the test is **+ve** what are the probabilities that the patient

i) has the virus ie **$P(H|P)$** ; ii) does not have virus ie **$P(\neg H|P)$** ;

If the test is **-ve** what are the probabilities that the patient

iii) has the virus ie **$P(H|\neg P)$** ; iv) does not have virus ie **$P(\neg H|\neg P)$** ;

**Calculations :**

**i)** For **$P(H|P)$** we can write down Bayes Theorem as

$P(H|P) = [\, P(P|H)\, P(H)\, ] / P(P)$

We know $P(P|H)$ and $P(H)$ but not $P(P)$ which is probability of a +ve result.

There are two cases, that a patient could have a +ve result, stated below :

ie $P(P) = P(H \cap P) + P(\neg H \cap P)$.

But from the second axiom of probability we have :

$P(H \cap P) = P(P|H)\, P(H)$ and $P(\neg H \cap P) = P(P|\neg H)\, P(\neg H)$.

Therefore putting these we get :

$P(P) = P(P|H)\, P(H) + P(P|\neg H)\, P(\neg H) = 0.95 \times 0.15 + 0.02 \times 0.85 = 0.1595$

Now substitute this into Bayes Theorem and obtain **$P(H|P)$**

$$P(H|P) = \frac{P(P|H)\, P(H)}{P(P|H)\, P(H) + P(P|\neg H)\, P(\neg H)} = 0.95 \times 0.15 / 0.1595 = 0.8934$$

**ii)** Next is to work out **$P(\neg H|P)$**

$P(\neg H|P) = 1 - P(H|P) = 1 - 0.8934 = 0.1066$

**iii)** Next is to work out $P(H|\neg P)$ ; again we write down Bayes Theorem

$$P(H|\neg P) = \frac{P(\neg P|H)\, P(H)}{P(\neg P)}$$ here we need $P(\neg P)$ which is $1 - P(P)$

$$= (0.05 \times 0.15)/(1-0.1595) = 0.008923$$

**iv)** Finally, work out $P(\neg H|\neg P)$

It is just $1 - P(H|\neg P) = 1 - 0.008923 = 0.99107$

**Definition and importance of knowledge**

 Knowledge can be defined as the body of facts and principles accumulated by humankind or the act, fact, or state of knowing

 Knowledge is having familiarity with language, concepts, procedures, rules, ideas, abstractions, places, customs, facts, and associations, coupled with an ability to use theses notions effectively in modeling different aspects of the world
 The meaning of knowledge is closely related to the meaning of intelligence
 Intelligent requires the possession of and access to knowledge
 A common way to represent knowledge external to a computer or a human is in the form of written language
 Example:
 Ramu is tall – This expresses a simple fact, an attribute possessed by a person
 Ramu loves his mother – This expresses a complex binary relation between two persons
 Knowledge may be declarative or procedural
 Procedural knowledge is compiled knowledge related to the performance of some task. For example, the steps used to solve an algebraic equation
 Declarative knowledge is passive knowledge expressed as statements of facts about the world. For example, personnel data in a database, such data are explicit pieces of independent knowledge
 Knowledge includes and requires the use of data and information
 Knowledge combines relationships, correlations, dependencies, and the notion of gestalt with data and information
 Belief is a meaningful and coherent expression. Thus belief may be true or false
 Hypothesis is defined as a belief which is backed up with some supporting evidence, but it may still be false
 Knowledge is true justified belief
 Epistemology is the study of the nature of knowledge
 Metaknowledge is knowledge about knowledge, that is, knowledge about what we Know

**DEMPSTER- SHAFER THEORY**

 The Dempster-Shafer theory, also known as the theory of belief functions, is a generalization of the Bayesian theory of subjective probability.
 Whereas the Bayesian theory requires probabilities for each question of interest, belief functions allow us to base degrees of belief for one question on probabilities for a related question. These degrees of belief may or may not have the mathematical properties of probabilities;
 The Dempster-Shafer theory owes its name to work by A. P. Dempster (1968) andGlenn Shafer (1976), but the theory came to the attention of AI researchers in the early 1980s, when they were trying to adapt probability theory to expert systems.
 Dempster-Shafer degrees of belief resemble the certainty factors in MYCIN, and this resemblance suggested that they might combine the rigor of probability theory with the flexibility of rule-based systems.
 The Dempster-Shafer theory remains attractive because of its relative flexibility.
 The Dempster-Shafer theory is based on two ideas:
 the idea of obtaining degrees of belief for one question from subjective probabilities for a related question,
 Dempster's rule for combining such degrees of belief when they are based on independent items of evidence.
 To illustrate the idea of obtaining degrees of belief for one question from subjective

probabilities for another, suppose I have subjective probabilities for the reliability of my friend Betty. My probability that she is reliable is 0.9, and my probability that she is unreliable is 0.1. Suppose she tells me a limb fell on my car. This statement, which must true if she is reliable, is not necessarily false if she is unreliable. So her testimony alone justifies a 0.9 degree of belief that a limb fell on my car, but only a zero degree of belief (not a 0.1 degree of belief) that no limb fell on my car. This zero does not mean that I am sure that no limb fell on my car, as a zero probability would;
it merely means that Betty's testimony gives me no reason to believe that no limb fell on my car. The 0.9 and the zero together constitute a belief function.
 To illustrate Dempster's rule for combining degrees of belief, suppose I also have a 0.9 subjective probability for the reliability of Sally, and suppose she too testifies, independently of Betty, that a limb fell on my car. The event that Betty is reliable is independent of the event that Sally is reliable, and we may multiply the probabilities of these events; the probability that both are reliable is 0.9x0.9 = 0.81, the probability that neither is reliable is 0.1x0.1 = 0.01, and the probability that at least one is reliable is 1 - 0.01 = 0.99. Since they both said that a limb fell on my car, at least of them being reliable implies that a limb did fall on my car, and hence I may assign this event a degree of belief of 0.99. Suppose, on the other hand, that Betty and Sally contradict each other—Betty says that a limb fell on my car, and Sally says no limb fell on my
car. In this case, they cannot both be right and hence cannot both be reliable—only one is reliable, or neither is reliable. The prior probabilities that only Betty is reliable, only Sally is reliable, and that neither is reliable are 0.09, 0.09, and 0.01, respectively, and the posterior probabilities (given that not both are reliable) are 9 19 , 9 19 , and 1 19 , respectively. Hence we have a 9 19 degree of belief that a limb did fall on my car (because Betty is reliable) and a 9 19 degree of belief that no limb fell on my car (because Sally is reliable).
 In summary, we obtain degrees of belief for one question (Did a limb fall on my car?) from probabilities for another question (Is the witness reliable?). Dempster's rule begins with the assumption that the questions for which we have probabilities are independent with respect to our subjective probability judgments, but this independence is only a priori; it disappears when conflict is discerned between the different items of evidence.
 Implementing the Dempster-Shafer theory in a specific problem generally involves solving two related problems.
 First, we must sort the uncertainties in the problem into a priori independent items of evidence.
 Second, we must carry out Dempster's rule computationally. These two problems and their solutions are closely related.
 Sorting the uncertainties into independent items leads to a structure involving items of evidence that bear on different but related questions, and this structure can be used to make computations
 This can be regarded as a more general approach to representing uncertainty than the Bayesian approach.
 The basic idea in representing uncertainty in this model is:
 Set up a confidence interval -- an interval of probabilities within which the true probability lies with a certain confidence -- based on the Belief $B$ and plausibility $PL$ provided by some evidence $E$ for a proposition $P$.
 The belief brings together all the evidence that would lead us to believe in $P$ with some certainty.

 The plausibility brings together the evidence that is compatible with *P* and is not inconsistent with it.
 This method allows for further additions to the set of knowledge and does not assume disjoint outcomes.

**Benefits of Dempster-Shafer Theory:**

- Allows proper distinction between reasoning and decision taking
- No modeling restrictions (e.g. DAGs)
- It represents properly partial and total ignorance
- Ignorance is quantified:

o low degree of ignorance means
- high confidence in results
- enough information available for taking decisions

o high degree of ignorance means
- low confidence in results
- gather more information (if possible) before taking decisions

- Conflict is quantified:

o low conflict indicates the presence of confirming information sources
o high conflict indicates the presence of contradicting sources

- Simplicity: Dempster's rule of combination covers

o combination of evidence
o Bayes' rule
o Bayesian updating (conditioning)
o belief revision (results from non-monotonicity)
 DS-Theory is not very successful because:
 Inference is less efficient than Bayesian inference
 Pearl is the better speaker than Dempster (and Shafer, Kohlas, etc.)
 Microsoft supports Bayesian Networks
 The UAI community does not like „outsiders"

**Fuzzy Set Theory**

**What is Fuzzy Set ?**

• The word "fuzzy" means "vagueness". Fuzziness occurs when the boundary of a piece of information is not clear-cut.
• Fuzzy sets have been introduced by Lotfi A. Zadeh (1965) as an extension of the classical notion of set.
• Classical set theory allows the membership of the elements in the set in binary terms, a bivalent condition - an element either belongs or does not belong to the set.
Fuzzy set theory permits the gradual assessment of the membership of elements in a set, described with the aid of a membership function valued in the real unit interval [0, 1].

**• Example:**

Words like young, tall, good, or high are fuzzy.
−There is no single quantitative value which defines the term young.

−For some people, age 25 is young, and for others, age 35 is young.
−The concept young has no clean boundary.
−Age 1 is definitely young and age 100 is definitely not young;
−Age 35 has some possibility of being young and usually depends on the context in which it is being considered.

**Introduction**

In real world, there exists much fuzzy knowledge;
Knowledge that is vague, imprecise, uncertain, ambiguous, inexact, or probabilistic in nature.
Human thinking and reasoning frequently involve fuzzy information, originating from inherently inexact human concepts. Humans, can give satisfactory answers, which are probably true.
However, our systems are unable to answer many questions. The reason is, most systems are designed based upon classical set theory and two-valued logic which is unable to cope with unreliable and incomplete information and give expert opinions.

**• Classical Set Theory**

A Set is any well defined collection of objects. An object in a set is called an element or member of that set.
−Sets are defined by a simple statement describing whether a particular element having a certain property belongs to that particular set.
−Classical set theory enumerates all its elements using **A = { a1 , a2 , a3 , a4 , . . . . an }**
If the elements **ai (i = 1, 2, 3, . . . n)** of a set **A** are subset of universal set **X**, then set **A** can be represented for all elements **x** **X** by its **characteristic function**

−A set **A** is well described by a function called characteristic function.
This function, defined on the universal space **X**, assumes :
a value of **1** for those elements **x** that belong to set **A**, and a value of **0** for those elements **x** that do not belong to set **A**.
The notations used to express these mathematically are
**A : X →[0, 1]**
**A(x) = 1 , x is a member of A Eq.(1)**
**A(x) = 0 , x is not a member of A**
Alternatively, the set **A** can be represented for all elements **x** **X**
by its characteristic function μ**A (x)** defined as

$$\mu A(x) = \begin{cases} 1 & \text{if } x \; X \\ 0 & \text{otherwise} \end{cases}$$

−Thus in classical set theory **A (x)** has only the values **0** ('false') and **1** ('true''). Such sets are called **crisp sets.**

**• Fuzzy Set Theory**

Fuzzy set theory is an extension of classical set theory where elements have varying degrees of membership. A logic based on the two truth values, *True* and *False*, is sometimes inadequate when describing human reasoning. Fuzzy logic uses the whole interval between
**0** (false) and **1** (true) to describe human reasoning.

−A **Fuzzy Set** is any set that allows its members to have different degree of membership, called **membership function**, in the interval **[0 , 1]**.
−The **degree of membership** or truth is not same as probability;
fuzzy truth is not likelihood of some event or condition.
fuzzy truth represents membership in vaguely defined sets;
−**Fuzzy logic** is derived from fuzzy set theory dealing with reasoning that is approximate rather than precisely deduced from classical predicate logic.
−Fuzzy logic is capable of handling inherently imprecise concepts.
−Fuzzy logic allows in linguistic form the set membership values to imprecise concepts like **"slightly", "quite" and "very".**
−Fuzzy set theory defines Fuzzy Operators on Fuzzy Sets.

**• Crisp and Non-Crisp Set**

−As said before, in classical set theory, the **characteristic function μA (x)** of Eq.(2) has only values **0** ('false') and **1** ('true").
Such sets are **crisp sets.**
−For Non-crisp sets the characteristic function **A(x)** can be defined.
The characteristic function **A(x)** of Eq. (2) for the crisp set is generalized for the Non-crisp sets.
This generalized characteristic function **A(x)** of Eq.(2) is called **membership function**. Such Non-crisp sets are called **Fuzzy Sets**.
−Crisp set theory is not capable of representing descriptions and classifications in many cases; In fact, Crisp set does not provide adequate representation for most cases.

**• Representation of Crisp and Non-Crisp Set**

Example : Classify students for a basketball team
This example explains the grade of truth value.
- **tall students** qualify and **not tall students** do not qualify
- if students 1.8 m tall are to be qualified, then should we exclude a student who is 1/10" less? or should we exclude a student who is 1" shorter?

■ Non-Crisp Representation to represent the notion of a tall person.

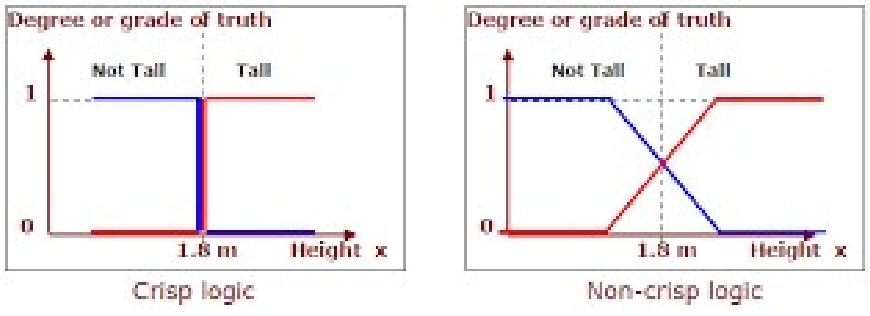


**Fig. 1 Set Representation – Degree or grade of truth**

A student of height 1.79m would belong to both tall and not tall sets with a particular degree of membership.
As the height increases the membership grade within the tall set would increase whilst the membership grade within the not-tall set would decrease.

# NATURAL LANGUAGE PROCESSING

## INTRODUCTION

Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language. Natural language processing is a subfield of computer science and in artificial intelligence that is concerned with computational processing of natural languages, emulating cognitive capabilities without being committed to a true simulation of cognitive processes. It is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human like language processing for a range of tasks or applications. It is a computerized approach to analyzing text that is based on both a set of theories and a set of technologies. NLP is a very active area of research and development. Naturally occurring texts can be of any language, mode and genre etc. The text can be oral or written. The only requirement is that they be in a language used by humans to communicate to one another. Also, the text being analyzed should not be specifically constructed for the purpose of analysis, but rather that the text is gathered from actual usage.

The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable. Most NLP techniques rely on machine learning to derive meaning from human languages.

Generally NLP is the means for accomplishing a particular task. It is a combination of computational linguistics and artificial intelligence. The natural language processing uses the tools of AI such as: algorithms, data structures, formal models for representing knowledge, models or reasoning processes etc. There are two ways through which the natural languages are being processed. First parsing technique and the second is the transition network. The architecture of NLP is given in the following figure.

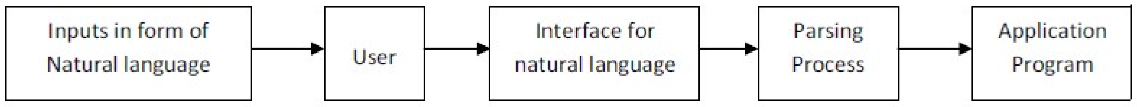

Inputs in form of Natural language
User
Interface for natural language
Parsing Process
Application Program

**Fig. 12.1: Architecture of NLP**

The goal of natural language processing is to specify a language comprehension and production theory to such a level of detail that a person is able to write a computer program which can understand and produce natural language. The basic goal of NLP is to accomplish human like language processing. The choice of word “processing” is very deliberate and should not be replaced with “understanding”. For although the field of NLP was originally referred to as Natural Language Understanding (NLU), that goal has not yet been accomplished. A full NLU system would be able to:

Paraphrase an input text.
Translate the text into another language.
Answer questions about the contents of the text.
Draw inferences from the text.

NLP lie in a number of disciplines like computer and information sciences, linguistics, mathematics, electrical and electronic engineering, artificial intelligence and robotics, psychology etc. Applications of NLP include a number of fields of studies such as machine translation, natural language text processing, summarization, user interfaces multilingual and Gross language information retrieval (CLIR), speech recognition, artificial intelligence and expert system.

As natural language processing technology matures, it is increasingly being used to support other computer applications. Such use naturally falls into two areas, one in which linguistic analysis merely serves as an interface to the primary program and the second one in which natural language considerations are central to the application. Natural language interfaces into a request in a formal database query language, and the program then proceeds as it would without the use of natural language processing techniques. Also some more application areas include information and text categorization. In both applications, natural language processing imposes a linguistic representation on each document being considered.

Natural Language Processing is the driving force behind the following common applications:

- Language translation applications such as Google Translate
- Word Processors such as Microsoft Word and Grammarly that employ NLP to check grammatical accuracy of texts.
- Interactive Voice Response (IVR) applications used in call centers to respond to certain users’ requests.
- Personal assistant applications such as OK Google, Siri, Cortana, and Alexa.

Syntactic analysis and semantic analysis are the main techniques used to complete Natural Language Processing tasks. Broadly construed, natural language processing is considered to involve at least the following steps

- Lexical analysis
- Syntactic analysis
- Semantic analysis
- Discourse Integration
- Pragmatic Analysis

## LEXICAL ANALYSIS

It involves identifying and analyzing the structure of words. Lexicon of a language means the collection of words and phrases in a language. Lexical analysis is dividing the whole chunk of txt into paragraphs, sentences, and words.

## SYNTACTIC PROCESSING

Processing a sentence syntactically involves determining the subject and predicate and the place of nouns, verbs, pronouns, etc. Given the variety of ways to construct sentences in a natural language, it's obvious that word order alone will not tell you much about these issues, and depending on word order alone would be frustrated anyway by the fact that sentences vary in length and can contain multiple clauses.

In NLP, syntactic analysis is used to assess how the natural language aligns with the grammatical rules. Computer algorithms are used to apply grammatical rules to a group of words and derive meaning from them.

Here are some syntax techniques that can be used:

**Lemmatization**: It entails reducing the various inflected forms of a word into a single form for easy analysis.

**Morphological segmentation**: It involves dividing words into individual units called morphemes.

**Word segmentation**: It involves dividing a large piece of continuous text into distinct units.

**Part-of-speech tagging**: It involves identifying the part of speech for every word.

**Parsing**: It involves undertaking grammatical analysis for the provided sentence.

**Sentence breaking**: It involves placing sentence boundaries on a large piece of text.

**Stemming**: It involves cutting the inflected words to their root form.

There are a number of algorithms researchers have developed for syntactic analysis, such as −

Context-Free Grammar

Top-Down Parser

## SEMANTIC ANALYSIS

Semantics refers to the meaning that is conveyed by a text. Semantic analysis is one of the difficult aspects of Natural Language Processing that has not been fully resolved yet. It involves applying computer algorithms to understand the meaning and interpretation of words and how sentences are structured.

Here are some techniques in semantic analysis:

**Named entity recognition (NER):** It involves determining the parts of a text that can be identified and categorized into preset groups. Examples of such groups include names of people and names of places.

**Word sense disambiguation:** It involves giving meaning to a word based on the context.

**Natural language generation**: It involves using databases to derive semantic intentions and convert them into human language.

## DISCOURSE INTEGRATION

The meaning of any sentence depends upon the meaning of the sentence just before it. In addition, it also brings about the meaning of immediately succeeding sentence.

While syntax and semantics work with sentence-length units, the discourse level of NLP works with units of text longer than a sentence i.e. it does not interpret multi-sentence texts as just concatenated sentences, each of which can be interpreted singly. Discourse focuses on the properties of the text as a whole that convey meaning by making connections between component sentences. Several types of discourse processing can occur at this level like anaphora resolution and discourse/text structure recognition. Anaphora resolution is the replacing of words such as pronouns which are semantically vacant with the appropriate entity to which they refer. For example, newspaper articles can be deconstructed into discourse components such as: lead, main story, previous events, evaluation etc. A discourse is a sequence of sentences. Discourse has structure much like sentences do. Understanding discourse structure is extremely important for dialog system.

For example: The dialog may be

When does the bus to Bhubaneswar leave?

There is one at 10 a.m. and one at 1 p.m.

Give me two tickets for the earlier one, please.

The problems with discourse analysis may be non-sentential utterances, cross-sentential anaphora.

**PRAGMATIC PROCESSING**

During this, what was said is re-interpreted on what it actually meant. It involves deriving those aspects of language which require real world knowledge.

This level is concerned with the purposeful use of language in situations and utilizes context over and above the contents of the text for understanding. The goal is to explain how extra meaning is read into texts without actually being encoded in them. This requires much world knowledge including the understanding of intentions, plans and goals. Some NLP applications may utilize knowledge bases and inferencing modules. Pragmatic is the study of how more gets communicated than is said. Speech acts in the pragmatic processing is the illocutionary force, the communicative force of an utterance, resulting from the function associated with it.

For example: Suppose the sentence is I will see you later.

Prediction: I predict that I will see you later.

Promise: I promise that I will see you later.

Warning: I warn you that I will see you later.

# LEARNING

## INTRODUCTION

Learning process is the basis of knowledge acquisition process. Knowledge acquisition is the expanding the capabilities of a system or improving its performance at some specified task. So we can say knowledge acquisition is the goal oriented creation and refinement of knowledge. The acquired knowledge may consist of various facts, rules, concepts, procedures, heuristics, formulas, relationships or any other useful information. Knowledge can be acquired from various sources like, domain of interests, text books, technical papers, databases, reports. The terms of increasing levels of abstraction, knowledge includes data, information and Meta knowledge. Meta knowledge includes the ability to evaluate the knowledge available, the additional knowledge required and the systematic implied by the present rules.

An agent is **learning** if it improves its performance on future tasks after making observations about the world. Learning can range from the trivial, as exhibited by jotting down a phone number, to the profound, as exhibited by Albert Einstein, who inferred a new theory of the universe.

## FORMS OF LEARNING

Factoring its representation of knowledge, AI learning models can be classified in two main types: inductive and deductive.

### INDUCTIVE LEARNING

This type of AI learning model is based on inferring a general rule from datasets of input-output pairs.. Algorithms such as knowledge based inductive learning(KBIL) are a great example of this type of AI learning technique. KBIL focused on finding inductive hypotheses on a dataset with the help of background information.

### DEDUCTIVE LEARNING

This type of AI learning technique starts with te series of rules nad infers new rules that are more efficient in the context of a specific AI algorithm. Explanation-Based Learning(EBL) and Relevance-0Based Learning(RBL) are examples examples o f deductive techniques. EBL extracts general rules from examples by “generalizing” the explanation. RBL focuses on identifying attributes and deductive generalizations from simple example.

Based on the feedback characteristics, AI learning models can be classified as supervised, unsupervised, semi-upervised or reinforced.

### UNSUPERVISED LEARNING

Unsupervised models focus on learning a pattern in the input data without any external feedback. Clustering is a classic example of unsupervised learning models.

In unsupervised learning the agent learns patterns in the input even though no explicit feedback is supplied. The most common unsupervised learning task is clustering: detecting potentially useful clusters of input examples. For example, a taxi agent might gradually develop a concept of "good traffic days" and "bad traffic days" without ever being given labeled examples of each by a teacher.

## SUPERVISED LEARNING

Supervised learning models use external feedback to learning functions that map inputs to output observations. In those models the external environment acts as a "teacher" of the AI algorithms.

In supervised learning the agent observes some example input–output pairs and learns a function that maps from input to output.

## SEMI-SUPERVISED LEARNING:

Semi-Supervised learning uses a set of curated, labeled data and tries to infer new labels/attributes on new data sets. Semi-Supervised learning models are a solid middle ground between supervised and unsupervised models.

In semi-supervised learning we are given a few labeled examples and must make what we can of a large collection of unlabeled examples. Even the labels themselves may not be the oracular truths that we hope for. Imagine that you are trying to build a system to guess a person's age from a photo. You gather some labeled examples by snapping pictures of people and asking their age. That's supervised learning. But in reality some of the people lied about their age. It's not just that there is random noise in the data; rather the inaccuracies are systematic, and to uncover them is an unsupervised learning problem involving images, self-reported ages, and true (unknown) ages. Thus, both noise and lack of labels create a continuum between supervised and unsupervised learning.

## REINFORCEMENT LEARNING

Reinforcement learning models use opposite dynamics such as rewards and punishment to "reinforce" different types of knowledge. This type of learning technique is becoming really popular in modern AI solutions.

In reinforcement learning the agent learns from a series of reinforcements—rewards or punishments. For example, the lack of a tip at the end of the journey gives the taxi agent an indication that it did something wrong. The two points for a win at the end of a chess game tells the agent it did something right. It is up to the agent to decide which of the actions prior to the reinforcement were most responsible for it.

## LEARNING DECISION TREE

Decision tree induction is one of the simplest and yet most successful forms of machine learning. We first describe the representation—the hypothesis space—and then show how to learn a good hypothesis.

## THE DECISION TREE REPRESENTATION

**Decision Tree.** A decision tree represents a function that takes as input a vector of attribute values and returns a "decision"—a single output value. The input and output values can be discrete or continuous. For now we will concentrate on problems where the inputs have discrete values and the output has exactly two possible values; this is Boolean classification, where each example input will be classified as true (a positive example) or false (a negative example).

A decision tree reaches its decision by performing a sequence of tests. Each internal node in the tree corresponds to a test of the value of one of the input attributes, $A_i$, and the branches from the node are labeled with the possible values of the attribute, $A_i = v_{ik}$. Each leaf node in the tree specifies a value to be returned by the function. The decision tree representation is natural for humans; indeed, many "How To" manuals (e.g., for car repair) are written entirely as a single decision tree stretching over hundreds of pages.

## EXPLANATION BASED LEARNING

Explanation based learning has ability to learn from a single training instance. Instead of taking more examples the explanation based learning is emphasized to learn a single, specific example. For example, consider the Ludoo game. In a Ludoo game, there are generally four colors of buttons. For a single color there are four different squares. Suppose the colors are red, green, blue and yellow. So maximum four members are possible for this game. Two members are considered for one side (suppose green and red) and other two are considered for another side (suppose blue and yellow). So for any one opponent the other will play his game. A square sized small box marked by symbols one to six is circulated among the four members. The number one is the lowest number and the number six is the highest for which all the operations are done. Always any one from the 1st side will try to attack any one member in the 2nd side and vice versa. At any instance of play the players of one side can attack towards the players of another side. Likewise, all the buttons may be attacked and rejected one by one and finally one side will win the game. Here at a time the players of one side can attack towards the players of another side. So for a specific player, the whole game may be affected. Hence we can say that always explanation based learning is concentrated on the inputs like a simple learning program, the idea about the goal state, the idea about the usable concepts and a set of rules that describes relationships between the objects and the actions.

Explanation based generalization (EBG) is an algorithm for explanation based learning, described in Mitchell at al. (1986). It has two steps first, explain method and secondly, generalize method. During the first step, the domain theory is used to prune away all the unimportant aspects of training examples with respect to the goal concept. The second step is to generalize the explanation as far as possible while still describing the goal concept. Consider the problem of learning the concept bucket. We want to generalize from a single example of a bucket. At first collect the following information.

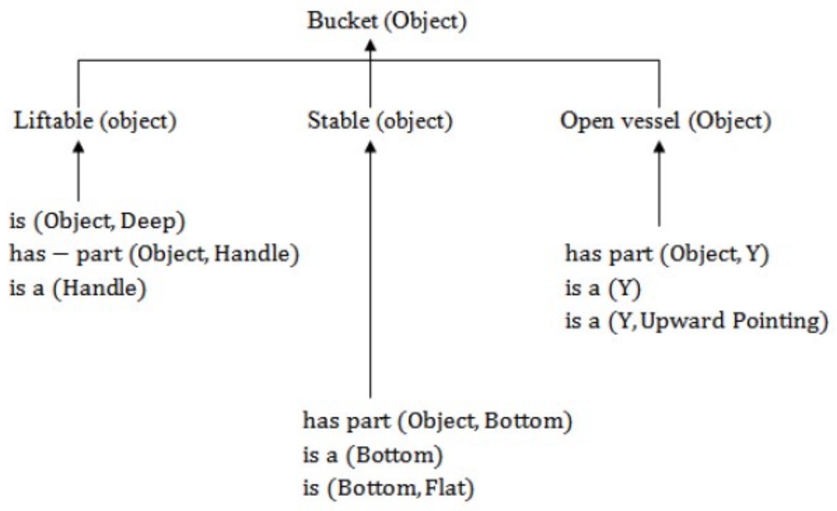


**Fig. 13.1: An explanation of BUCKET Object**

Given a training example and a functional description, we want to build a general structural description of a bucket. In practice, there are two reasons why the explanation based learning is important.

1. **<u>Input Examples:</u>**

Owner (object, X) ∧ has part (object, Y) ∧ is(object, Deep) ∧ Color(Object, Green)

∧ ... ... (Where Y is any thin material)

2. **Domain Knowledge:**
   is (a, Deep) ∧ has part(a, b) ∧ is a(b, handle) → liftable(a)
   has part(a, b) ∧ is a(b, Bottom) ∧ is(b, flat) → Stable(a)
   has part(a, b) ∧ is a(b, Y) ∧ is(b, Upward - pointing) → Open –vessel(a)

3. **Goal:** Bucket
   B is a bucket if B is a liftable, stable and open-vessel

4. **Description of Concept:** These are expressed in purely structural forms like Deep, Flat, Rounded etc.

## LEARNING USING RELEVANCE INFORMATION

The prior knowledge Background concerns the relevance of a set of features to the goal predicate. This knowledge, together with the observations, allows the agent to infer a new, general rule that explains the observations:

*Hypothesis ∧ Descriptions |= Classifications ,*

*Background ∧ Descriptions ∧ Classifications |= Hypothesis*

This kind of generalization is known as relevance-based learning or RBL although the name is not standard). Notice that whereas RBL does make use of the content of the observations, it does not produce hypotheses that go beyond the logical content of the background knowledge and the observations. It is a deductive form of learning and cannot by itself account for the creation of new knowledge starting from scratch.

## NEURAL NET LEARNING

A neural network consists of inter connected processing elements called neurons that work together to produce an output function. The output of a neural network relies on the cooperation of the individual neurons within the network to operate. Well designed neural networks are trainable systems that can often “learn” to solve complex problems from a set of exemplars and generalize the “acquired knowledge” to solve unforeseen problems, i.e. they are self-adaptive systems. A neural network is used to refer to a network of biological neurons. A neural network consists of a set of highly interconnected entities called nodes or units. Each unit accepts a weighted set of inputs and responds with an output.

Mathematically let I = ($I_1$,$I_{2,\dots}$ …$I_n$) represent the set of inputs presented to the unit U. Each input has an associated weight that represents the strength of that particular connection. Let W = ($W_1$,$W_2$,...... $W_n$) represent the weight vector corresponding to the input vector X. By applying to V, these weighted inputs produce a net sum at U given by

$$S = SUM (Wi * I_i)$$

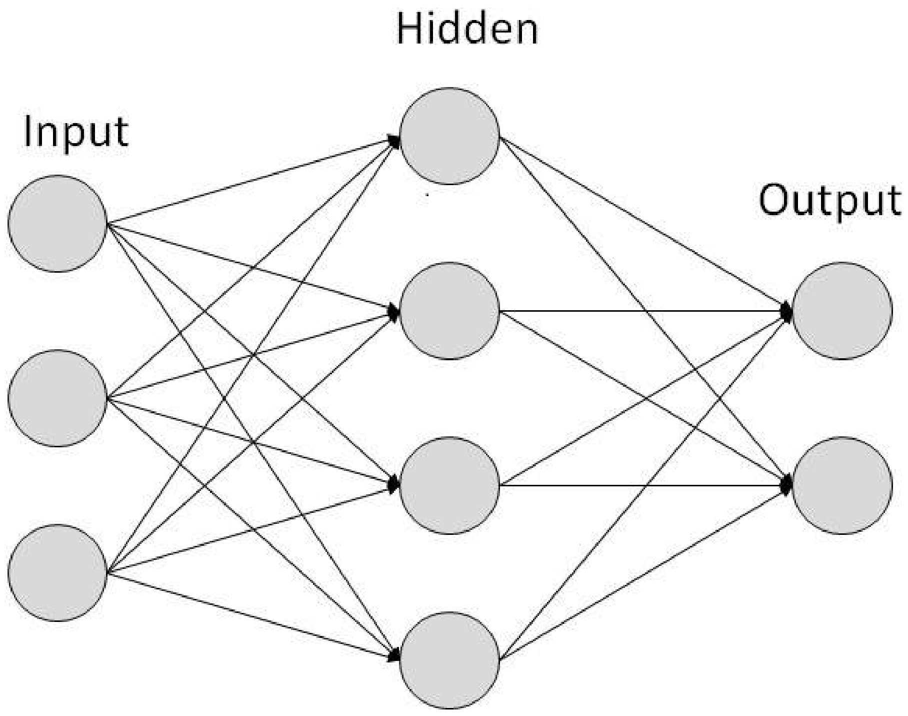


**Fig. 13.2:** Example of a Neural network structure

## FEATURES OF ARTIFICIAL NETWORK (ANN)

Artificial neural networks may by physical devices or simulated on conventional computers. From a practical point of view, an ANN is just a parallel computational system consisting of many simple processing elements connected together in a specific way in order to perform a particular task. There are some important features of artificial networks as follows.

(1) Artificial neural networks are extremely powerful computational devices (Universal computers).

(2) ANNs are modeled on the basis of current brain theories, in which information is represented by weights.
(3) ANNs have massive parallelism which makes them very efficient.
(4) They can learn and generalize from training data so there is no need for enormous feats of programming.
(5) Storage is fault tolerant i.e. some portions of the neural net can be removed and there will be only a small degradation in the quality of stored data.
(6) They are particularly fault tolerant which is equivalent to the "graceful degradation" found in biological systems.
(7) Data are naturally stored in the form of associative memory which contrasts with conventional memory, in which data are recalled by specifying address of that data.
(8) They are very noise tolerant, so they can cope with situations where normal symbolic systems would have difficulty.
(9) In practice, they can do anything a symbolic/ logic system can do and more.
(10) Neural networks can extrapolate and intrapolate from their stored information. The neural networks can also be trained. Special training teaches the net to look for significant features or relationships of data.

## TYPES OF NEURAL NETWORK

There are two Artificial Neural Network topologies − **FeedForward** and **Feedback.** Also according to number of layers it can be classified as **Single Layer Neural Network** and **Multi-layer Neural Network**.

**FeedForward ANN.** In this ANN, the information flow is unidirectional. A unit sends information to other unit from which it does not receive any information. There are no feedback loops. They are used in pattern generation/recognition/classification. They have fixed inputs and outputs.

**FeedBack ANN.** Here, feedback loops are allowed. They are used in content addressable memories.

**Single Layer Neural Network.** A single layer neural network consists of a set of units organized in a layer. Each unit U; receives a weighted input $I_m$ with weight $W_m$;. Figure shows a single layer neural network with j inputs and n outputs.

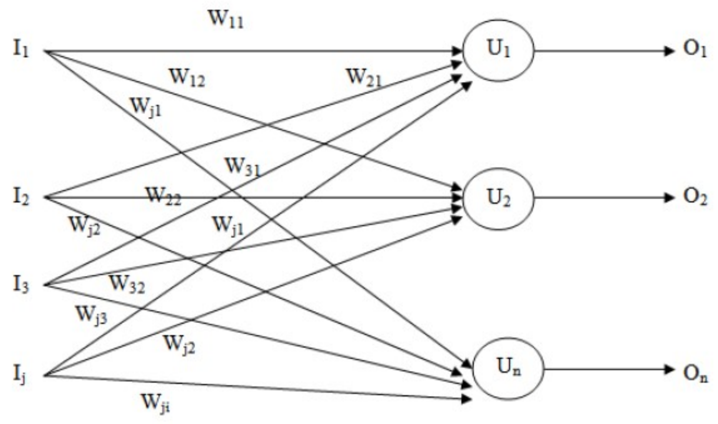


**Fig. 13.3:** Single layer neural network

Let I = ($i_1$, $i_2$ ....... $i_j$) be the input vector and let the activation function f be simply, so that the activation
value is just the net sum to a unit. The **j** *x* **n** weight matrix is calculated as follows.

$$W = \begin{pmatrix} W_{11} & \cdots & W_{1n} \\ \vdots & \ddots & \vdots \\ W_{j1} & \cdots & W_{jn} \end{pmatrix}$$

Thus the output $O_x$ at unit $U_x$ is

$$O_K = (W_{1K}, W_{2K} \cdots\cdots\cdots W_{JI})$$

**Multi-layer Neural Network**. A multilayer network has two or more layers of units, with the output from one layer serving as input to the next. Generally in a multilayer network there are 3 layers present like, input layer, output layer and hidden layer. The layer with no external output connections are referred to as hidden layers. A multilayer neural network structure is given in figure.

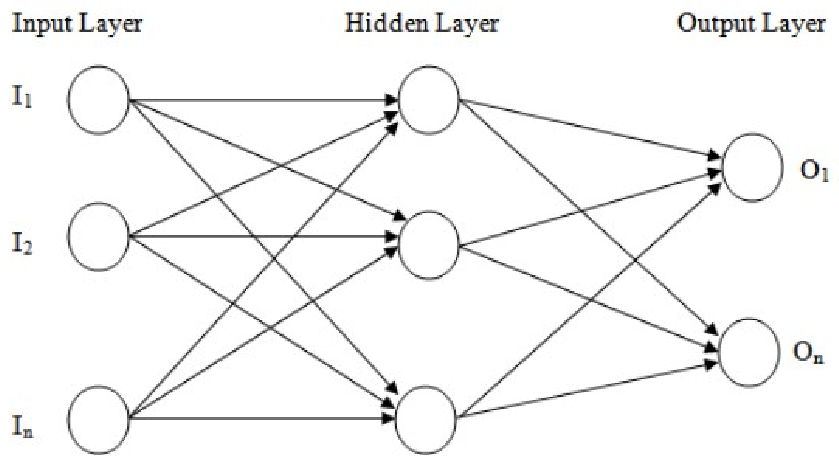


**Fig. 13.4:** Multi-layer neural network

Any multilayer system with fixed weights that has a linear activation function is equivalent to a single layer linear system, for example, the case of a two layer system. The input vector to the first layer is $I_r$ the output $O = W_1 * I$ and the second layer produces output $O_2 = W_2 * O$. Hence $O_2 = W_2 * (W_1 * I) = (W_2 * W_1) * I$

So a linear system with any number n of layers is equivalent to a single layer linear system whose weight matrix is the product of the n intermediate weight matrices. A multilayer system that is not linear can provide more computational capability than a single layer system. Generally multilayer networks have proven to be very powerful than single layer neural network. Any type of Boolean function can be implemented by such a network. At the output layer of a multilayer neural network the output vector is compared to the expected output. If the difference is zero, no changes are made to the weights of connections. If the difference is not zero, the error is calculated and is propagated back through the network.

## BACK PROPAGATION NEURAL NETWORK

The main objective in neural model development is to find an optimal set ofweight parametersw, such that y=y(x,w) closely represents (approximates)the original problem behavior. This is achieved through a process called training(that is, optimization inw-space). A set of training data is presented to theneural network. The training data are pairs of $(x_k,d_k)$, k=1,2,...,P, where $d_k$ is the desired outputs of the neural model for inputs $x_k$, and P is the total number of training samples.

During training, the neural network performance is evaluated by computing the difference between actual neural network outputs and desired outputs for all the training samples. The difference, also known as the error, is quantified by

$$E = \frac{1}{2} \sum_{k \in T_r} \sum_{j=1}^{m} (y_j(x_k, w) - d_{jk})^2$$

Where $d_{jk}$ is the jth element of$d_k$,$y_j(x_k,w)$ is the jth neural network outputfor input $x_k$, and $T_r$ is an index set of training data. The weight parametersware adjusted during training, such that this error is minimized. In 1986,Rumelhart, Hinton, and Williams[1] proposed a systematic neural networktraining approach. One of the significant contributions of their work is theerror back propagation (BP) algorithm.

**GENETIC LEARNING**

Genetic algorithms are based on the theory of natural selection and work on generating a set of random solutions and making them compete in an area where only the fittest survive. Each solution in the set is equivalent to a chromosome. Genetic algorithm learning methods are based on models of natural adaption and evolution. These learning methods improve their performance through processes which model population genetics and survival of the fittest. In the field of genetics, a population is subjected to an environment which places demands on the members. The members which adapt well are selected formatting and reproduction. Generally genetic algorithm uses three basic genetic operators like reproduction, crossover and mutation. These are combined together to evolve a new population. Starting from a random set of solutions the algorithm uses these operators and the fitness function to guide its search for the optimal solution. The fitness function guesses how good the solution in question is and provides a measure to its capability. The genetic operators copy the mechanisms based on the principles of human evolution. The main advantage of the genetic algorithm formulation is that fairly accurate results may be obtained using a very simple algorithm. The genetic algorithm is a method of finding a good answer to a problem, based on the feedback received from its repeated attempts at a solution.

Genetic algorithms are inspired by nature and evolution, which is seriously cool to me. It's no surprise, either, that artificial neural networks ("NN") are also modeled from biology: evolution is the best general-purpose learning algorithm we've experienced, and the brain is the best general-purpose problem solver we know. These are two very important pieces of our biological existence, and also two rapidly growing fields of artificial intelligence and machine learning study.

**Genetic Algorithm Pseudocode**

```
START
Generate the initial population
Compute fitness
REPEAT
   Selection
   Crossover
   Mutation
   Compute fitness
UNTIL population has converged
STOP
```

# EXPERT SYSTEM

## INTRODUCTION

The most important applied area of AI is the field of expert systems. An ***expert system*** (ES) is a knowledge-based system that employs knowledge about its application domain and uses an inferencing (reason) procedure to solve problems that would otherwise require human competence or expertise. The power of expert systems stems primarily from the specific knowledge about a narrow domain stored in the expert system's ***knowledge base***.

Expert systems are assistants to decision makers and not substitutes for them. Expert systems do not have human capabilities. They use a knowledge base of a particular domain and bring that knowledge to bear on the facts of the particular situation at hand. The knowledge base of an ES also contains ***heuristic knowledge* -** rules of thumb used by human experts who work in the domain.

In other terms, **Expert system**, a computer program that uses artificial-intelligence methods to solve problems within a specialized domain that ordinarily requires human expertise. The first expert system was developed in 1965 by Edward Feigenbaum and Joshua Lederberg of Stanford University in California, U.S. Dendral, as their expert system was later known, was designed to analyze chemical compounds. Expert systems now have commercial applications in fields as diverse as medical diagnosis, petroleum engineering, and financial investing.

## REPREENTATING AND USING DOMAIN KNOWLEDGE

The knowledge base of an ES contains both factual and heuristic knowledge. ***Knowledge representation*** is the method used to organize the knowledge in the knowledge base. Knowledge bases must represent notions as actions to be taken under circumstances, causality, time, dependencies, goals, and other higher-level concepts.

Several methods of knowledge representation can be drawn upon. Two of these methods include:

**1. Frame-based systems** - are employed for building very powerful ESs. A frame specifies the attributes of a complex object and frames for various object types have specified relationships.

**2. Production rules** - are the most common method of knowledge representation used in business. ***Rule-based expert systems*** are expert systems in which the knowledge is represented by production rules.

A production rule, or simply a rule, consists of an IF part (a condition or premise) and a THEN part (an action or conclusion). IF condition THEN action (conclusion). The ***explanation facility*** explains how the system arrived at the recommendation. Depending on the tool used to implement the expert system, the explanation may be either in a natural language or simply a listing of rule numbers.

**The inference engine:** 1. Combines the facts of a specific case with the knowledge contained in the knowledge base to come up with a recommendation. In a rule-based expert system, the inference engine controls the order in which production rules are applied and resolves conflicts if more than one rule is applicable at a given time.

2. Directs the user interface to query the user for any information it needs for further inferencing.

The facts of the given case are entered into the ***working memory***, which acts as a blackboard, accumulating the knowledge about the case at hand. The inference engine repeatedly applies the rules to the working memory, adding new information (obtained from the rules conclusions) to it, until a goal state is produced or confirmed.

One of several strategies can be employed by an inference engine to reach a conclusion. Inferencing engines for rule-based systems generally work by either forward or backward chaining of rules.

Forward-chaining systems are commonly used to solve more open-ended problems of a design or planning nature, such as, for example, establishing the configuration of a complex product. Backward chaining is best suited for applications in which the possible conclusions are limited in number and well defined. Classification or diagnosis type systems, in which each of several possible conclusions can be checked to see if it is supported by the data, are typical applications.

## RULE BASED ARCHITECTURE OF AN EXPERT SYSTEM

The most common form of architecture used in expert and other types of knowledge based systems is the production system or it is called rule based systems. This type of system uses knowledge encoded in the form of production rules i.e. if-then rules. The rule has a conditional part on the left hand side and a conclusion or action part on the right hand side. For example if: condition1 and condition2 and condition3
Then: Take action4
Each rule represents a small chunk of knowledge to the given domain of expertise. When the known facts support the conditions in the rule's left side, the conclusion or action part of the rule is then accepted as known. The rule based architecture of an expert system consists of the domain expert, knowledge engineer, inference engine, working memory, knowledge base, external interfaces, user interface, explanation module, database spreadsheets executable programs s mentioned in figure .

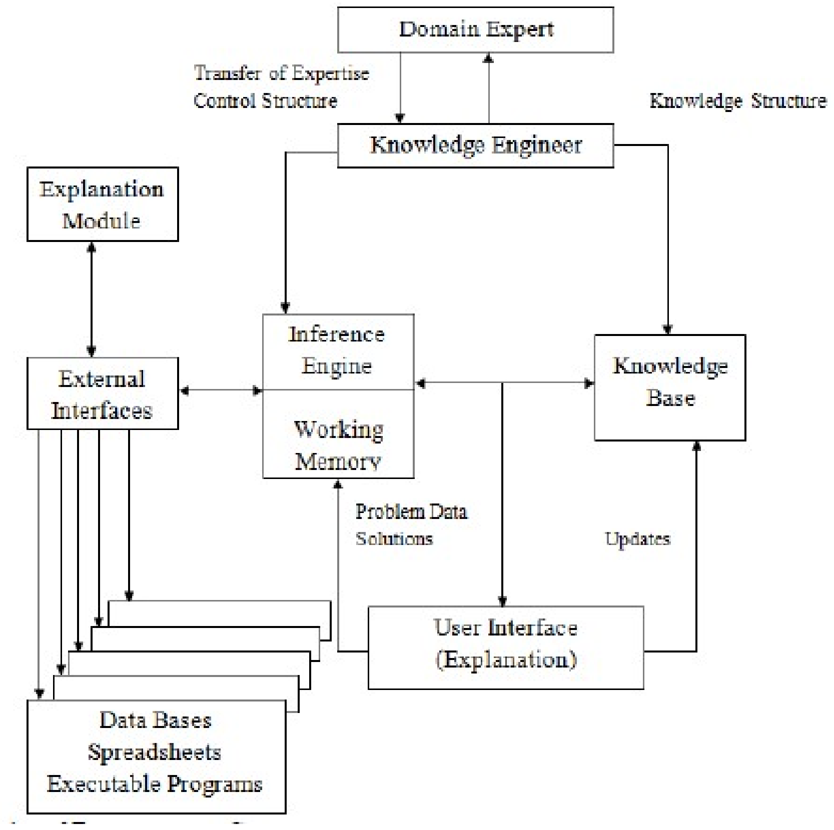


**Fig. 14.1:** Integration of Expert systems Components

The components of the rule based architecture are as follows.

1. **User Interface:** It is the mechanism by which the user and the expert system communicate with each other i.e. the use interacts with the system through a user interface. It acts as a bridge between user and expert system. This module accepts the user queries and submits those to the expert system. The user normally consults the expert system for following reasons.

a) To get answer of his/her queries.

b) To get explanation about the solution for psychological satisfaction.

The user interface module is designed in such a way that at user level it accepts the query in a language understandable by expert system. To make the expert system user friendly, the user interface interacts with the user in natural language. The user interface provides as much facilities as possible such as menus, graphical interfaces etc. to make the dialog user friendly and more attractive.

2. **Explanation Module:** The explanation module explains the reasoning of the system to a user. It provides the user with an explanation of the reasoning process when requested. The credibility of expert system will be established only when it is able to explain "how and why" a particular conclusion is drawn. This explanation increases the belief of user in the expert system.

a) **Explanation(How):** To respond to a how query, the explanation module traces the chain of rules fired during a consolation with the user. This explanation mode can be activated once the process is over. It explains how a fact was deduced by the system and similarly how a rule was/wasn't used. The simplest way to specify this is to explain the rule which allows the

deduction. For e.g. If the system (S) will give information about the parent-child relationship to the user (U) then the followings can be possible.
*S: My diagnosis is "A is the father of B"*
*U: How?*
*S: The result was obtained by the addition of following facts and rules.*
Fact no 11: A is the parent of Hari.
Fact no 15: A is a male.
Fact no 110: X is father of Y:
X is parent of Y, X is male.
So A is the father of B.
b) **Explanation (Why)?** To respond to a why query, the explanation module must be able to explain why certain information is needed by the inference engine to complete a step in the reasoning process. This mode of explanation is dynamic and is used when the system functions in backward chaining. If the user doesn't understand the reason behind the question. The system is capable of explaining why the question was asked.
For example S: Is the following true?
A is the father of B.
U: Why?
S: I need the fact:
A is the father of B to establish the following fact "B is the son of A".
By using the rule no. 4:
A is the father of B:
B is the son of A.
3. **Working Memory:** It is a global database of facts used by the rules.
4. **Knowledge Engineering:** The primary people involved in building an expert system are the knowledge engineer, the domain expert and the end user. Once the knowledge engineer has obtained a general overview of the problem domain and gone through several problem solving sessions with the domain expert, he/she is ready to begin actually designing the system, selecting a way to represent the knowledge, determining the search strategy (backward or forward) and designing the user interface. After making complete designs, the knowledge engineer builds a prototype. The prototype should be able to solve problems in a small area of the domain. Once the prototype has been implemented, the knowledge engineer and domain expert test and refine its knowledge by giving it problems to solve and correcting its disadvantages.
5. **Knowledge Base:** In rule based architecture of an expert system, the knowledge base is the set of production rules. The expertise concerning the problem area is represented by productions. In rule based architecture, the condition actions pairs are represented as rules, with the premises of the rules (if part) corresponding to the condition and the conclusion (then part) corresponding to the action. Case-specific data are kept in the working memory. The core part of an expert system is the knowledge base and for this reason an expert system is also called a knowledge based system. Expert system knowledge is usually structured in the form of a tree that consists of a root frame and a
number of sub frames. A simple knowledge base can have only one frame, i.e. the root frame whereas a large and complex knowledge base may be structured on the basis of multiple frames.
**Inference Engine:** The inference engine accepts user input queries and responses to questions through the I/O interface. It uses the dynamic information together with the static knowledge stored in the knowledge base. The knowledge in the knowledge base is used to derive

conclusions about the current case as presented by the user’s input. Inference engine is the module which finds an answer from the knowledge base. It applies the knowledge to find the solution of the problem. In general, inference engine makes inferences by deciding which rules are satisfied by facts, decides the priorities of the satisfied rules and executes the rule with the highest priority. Generally inferring process is carried out recursively in 3 stages like match, select and execute. During the match stage, the contents of working memory are compared to facts and rules contained in the knowledge base. When proper and consistent matches are found, the corresponding rules are placed in a conflict set.

## EXPERT SYSTEM SHELLS

The ES shell simplifies the process of creating a knowledge base. It is the shell that actually processes the information entered by a user relates it to the concepts contained in the knowledge base and provides an assessment or solution for a particular problem. Thus ES shell provides a layer between the user interface and the computer O.S to manage the input and output of the data. It also manipulates the information provided by the user in conjunction with the knowledge base to arrive at a particular conclusion.

**Expert system shells** are the most common vehicle for the development of specific ESs. A shell is an expert system without a knowledge base. A shell furnishes the ES developer with the inference engine, user interface, and the explanation and knowledge acquisition facilities.

***Domain-specific shells*** are actually incomplete specific expert systems, which require much less effort in order to field an actual system.

## KNOWLEDGE ACQUISITION

Knowledge acquisition is the process of adding new knowledge to a knowledge base and refining or otherwise improving knowledge that was previously acquired. Acquisition is usually associated with some purpose such as expanding the capabilities of a system or improving its performance at some specified task. It is goal oriented creation and refinement of knowledge . It may consist of facts, rules , concepts, procedures, heuristics, formulas, relationships, statistics or other useful information.

The knowledge acquisition component allows the expert to enter their knowledge orexpertise into the expert system, and to refine it later as and whenrequired.Historically, the knowledge engineer played a major role in this process, but automatedsystems that allow the expert to interact directly with the system arebecomingincreasingly common.The knowledge acquisition process is usually comprised of three principal stages:

1. Knowledge elicitation is the interaction between the expert and the knowledge engineer/program to elicit the expert knowledge in some systematic way.

2. The knowledge thus obtained is usually stored in some form of human friendlyintermediate representation.

3. The intermediate representation of the knowledge is then compiled into anexecutable form(e.g. production rules) that the inference engine can process.

**STAGES OF KNOWLWDGE ACQUISITION**

The iterative nature of the knowledge acquisition process can be represented in the following diagram (five stages):

- Identification: -break problem into parts.
- Conceptualisation: identify concepts.
- Formalisation: representing knowledge.
- Implementation: programming.
- Testing: validity of knowledge.