Artificial Intelligence

for

UGC-NET/JRF

TIFR | BARC | DRDO | ISRO | SLET | ONGC | JNU | Ph.D Entrance

First Edition : August 2016

CAREER ENDEAVOUR PUBLICATIONS has taken due care in collecting the data and providing the solutions, before publishing this book. Inspite of this, if any inaccuracy or printing errors are there, **CAREER ENDEAVOUR PUBLICATIONS** owes no responsibility. **CAREER ENDEAVOUR PUBLICATIONS** will be grateful if you could point out any such error. Your suggestions will be highly appreciated.

CAREER ENDEAVOUR Publications

Regd. Office: 28-A/11, Jia Sarai, New Delhi-16, **Ph:** 011-26851008, 26861009
E: info@careerendeavour.com, **W:** www.careerendeavour.com

CONTENTS

CHAPTER

1

Artificial Intelligence : Introduction

Intelligence: It is an ability to learn OR understand from the experience. It is the ability to learn and retain the knowledge, the ability to respond quickly to a new situation, ability of reason (apply the logic), etc.

Artificial Intelligence (AI) is the study of how to make computers do things which, at the moment, people do better.

In other words, Artificial Intelligence is the branch of computer science in which intelligence is incorporated in an artificial device. Purpose of AI is to create Human clone.

The central problem or goal of AI research include reasoning knowledge, planning, learning, natural language processing, perception and ability to manipulate objects. The AI field is interdisciplinary in which a number of sciences and professional coverage including computer science, mathematics psychology, linguistics, philosopy as well as specialized fields such as artificial psychology.

At the heart of research in artificial intelligence lies what Newell and Simon [1976] call the physical symbol system hypothesis. They define a physical symbol system as follows:

A physical symbol system consists of a set of entities, called symbols, which are physical patterns that can occur as components of another type of entity called an expression (or symbol structure). Thus, a symbol structure is composed of a number of instances (or (tokens) of symbols related in some physical way (such as one token being next to another). At any instant of time the system will contain a collection of these symbol structures. Besides these structures, the system also contains a collection of processes that operate on expressions to produce other expressions: processes through time an envolving collection of symbol structures. Such a system exists in a world of objects wider than just these symbolic expressions themselves.

They then state the hypothesis as

The physical symbol system hypothesis : A physical symbol system has the necessary and sufficient means for general intelligent action.

AI Technique :

One of the result about AI research states that “intelligence requires knowledge”.

The knowledge possesses some less desirable properties as given below:

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

Thus, we conclude that on AI technique is a method that exploits knowledge that should be represented in such a way that

- The knowledge captures generalization.
- It can be understood by people who must provide it.
- It can easily be modified to correct erroro and to reflect changes in the world and its view.
- It can be used in a great many situations even if it is not totally accurate or complete.
- It can be used to overcome its own sheer bulk by helping to narrow the range of possibilities that must usually be considered.

It is possible to solve AI problems without using AI techniques and it is also possible to apply AI techniques to the solution of non AI problems.

Types of Artificial intelligence

1. **Strong AI :** It aims to build a machine which can truely reason and solve the problem.
   - It is self aware
   - Its overall intelligence ability is indistinguishable from the human being
2. **Weak AI :** To create a machine that can not truely reason and solve the problem but its behaviour in such a way that it is very intelligent.
3. **Applied AI :** To build commercially smart system.
4. **Cognitive AI :** It perform tests on the way the human mind work simulation.

Tic-Tac-Toe game Playing:

- There is one human player and other player is computer. The objectives is to write a computer program which wins most of the time. We will present three approaches to play this game which increase in

• Complexity
• Use of generalization
• Clarity of their knowledge
• Extensibility of their appraoch.

- These approaches will move towards being representations of what we will call AI techniques.

Tic Tac Toe

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

Positions

Program 1:

Data Structures:

- Consider a Board having nine elements vector.
- Each element will contain

  • 0 for blank
  • 1 indicating X player move
  • 2 indicating O player move.
- Computer may play as X or O player.
- First player who so ever is always plays X.

Move table:

- It is a vector of 3^9 elements, each element of which is a nine element vector representing board position.
- Total of 3^9 (19683) elements in move table.

Index	Current Board Position	New Board Position
0	000000000	000010000
1	000000001	020000001
2	000000002	000100002
3	000000010	002000010
:		
:	:	
:		

Algorithm:

To make a move, do the following:

1. View the vector (board) as a ternary number and convert it to its corresponding decimal number.
2. Use the computer number as index into the move table and access the vector stored there.
3. The vector selected in step 2 represents the way the board will look after the move that should be made. So set board equal to that vector.

Comments:

- Very efficient in terms of time but has several disadvantages.
- Lot of space to store the move table.
- Error prone because of the data is voluminous.
- Poor extensibility: 3D tic-tac-toe = 3^{27} board position to be stored.
- Not intelligent.

Program 2: Data Structure: A nine-element vector representing the board: B[1..9].

2	–	Indicates blank
3	–	X
5	–	0

Turn : An integer

1	–	First move
2	–	Last move

Algorithm:

1. Make_2:

- Tries to make valid 2
- It first tries to play in the centre if free and returns 5 (square number).
- If not possible, then it tries the various suitable non corner square and returns square number.

2. PossWin (P):

- Returns 0
- It returns 0, if player P cannot win in its next move, otherwise returns the number of square that constitutes a winning move for P.
- If PossWin (P) = 0 {P can not win}, then find whether opponent can win. If so, then block it.

3. Go(n):

- Makes a move in square n {which is blank represented by 2}.

Strategy used by PosWin:

- It operates by checking, one at a time, each of rows/columns and diagonals. It can test for each row/column/diagonal the following:

    If 3*3*2 = 18 then X can win

    Else if 5*5*2 = 50 then O can win.

These three procedures are used in the following algorithm.

Algorithm:

Assumption: The first player uses symbol X.

Computer (C) gets chance to play first (C plays X, H plays O) – **Odd moves**

Human (H) gets change to play first. (Computer plays O, H plays X) – **Even moves.**

1 move: go (5)

2 move: If B[5] is blank, then Go(5) else Go(1).

3 move: If B[9] is blank, then Go(9) else Go(3) **{make 2}.**

4 move: {By now human (playing X) has played 2 chances} If PossWin (X) then **{block H}** Go (PossWin(X)) else Go (Make_2).

5 move: {By now computer has played 2 chances} If PossWin(X) then **{won}** Go (PossWin(X)) else **{block H}** if PossWin (O) then Go (PossWin(O)) else if B[7] is blank then Go(7) else Go(3).

6 move: {By now both have played 2 chances} If PossWin(O) then **{won}** Go(PossWin(O)) else **{block H}** If PossWin(X) then Go(PossWin(X)) else Go (Make_2)

7 & 9 movse: {By now human (Playing O) has played 3 chances} If PossWin(X) then **{won}** Go (PossWin(X)) else **{block H}** if PossWin(O) then Go (PossWin(O)) else Go (Anyhwere).

8 move: {By now computer has played 3 chances} If PossWin (O) then **{won}** Go (PossWin(O)) else **{block H}** If PossWin(X) then Go (PossWin(X)) else Go(Anywhere).

Comments:

- Not as efficient as first one in terms of time. Several conditions are checked before each move.
- It is memory efficient.
- Easier to understand and complete strategy has been determined in advance.
- Still can not generalize to 3-D.

Program 3:

- Same as program 2 except for one change in the representation of the board which makes process of checking for a possible win more simple.
- Board is considered to be a magic square of size 3×3 with 9 blocks numbered by numbers indicated by magic square.

Board

8	3	4
1	5	9
6	7	2

- Keep list of each player's blocks in which he has played.

Strategy for possible win for one player:

- Consider each pair of blocks that player owns.
- Computer difference D between 15 and the sum of the two blocks.
- If D < 0 or D > 9 then these two blocks are not collinear and so can be ignored otherwise if the block representing difference is blank (i.e., not in either list) then a move in that block will produce a win.
- Consider the magic block shown above and the lists of both the players, say after 7th move as:

Player X (Human)

8 3 6 2

Player O (Computer)

5 4 1 7

- First check if computer can win. If not then check if opponent can win. If so, then block it and proceed further.

- Use H for human and C for computer.
- Steps involved in the play are:

• First chance, H plays in block numbered as 8

• Next C plays in block numbered as 5

• H plays in block numbered as 3.

• Now C checks, if H can win or not.

(a) Compute sum of blocks played by H

S = 8 + 3 = 11

(b) Compute D = 15–11 = 4

(c) Block 4 is a winning block for H. So block it and play in block numbered 4. So, list of C gets block numbered as 4 recorded in its list.

• H plays in block numbered as 6.

• Now C checks, if C can win as follows:

(a) Compute sum of blocks played by C

S = 5 + 4 = 9

(b) Compute D = 15–9 = 6

(c) Block 6 is not free, so C can not win at this point in time. Now check if H can win.

• Compute sum of pair of square from list of H which has not been used earlier.

S1 = 8 + 6 =14

S2 = 3 + 6 = 9

• Compute D = 15– 14 = 1

• Block 1 is free, so C plays in block numbered as 1.

• H plays in block numbered as 2.

• Further C checks, if C can win as follows:

(a) Compute sum of blocks played by C which has not been used earlier.

S1 = 5 + 1 = 6

S2 = 4 + 1 = 5

(b) Compute D = 15–6 = 9

(c) Block numbered as 9 is free, as C plyas in 9 and win the game.

Comments:

• This program will require more time than two others since it must search a tree representing all possible move sequence before making each move.

• It could be extended to handle 3-dimensional tic-tac-toe.

• It could also be extended to handle games more complicated than tic-tac-toe.

PRACTICE SET

1. What is Artificial intelligence?
(a) Putting your intelligence into Computer (b) Programming with your own intelligence
(c) Making a Machine intelligent (d) Putting more memory into Computer

2. Which instruments are required for perceiving and acting upon the environment?
(a) Sensors and Actuators (b) Sensors
(c) Perceiver (d) None of the above

3. Strong Artificial Intelligence is
(a) the embodiment of human intellectual capabilities within a computer.
(b) a set of computer programs that produce output that would be considered to reflect intelligence if it were generated by humans.
(c) the study of mental faculties through the use of mental models implemented on a computer.
(d) All of the mentioned

4. Weak Artificial intelligence is?
(a) The embodiment of human intellectual capabilities within a computer
(b) A set of computer programs that produce output that would be consider to reflect intelligence if it
(c) The study of mental faculties using mental models implemented on a computer.
(d) All of the mentioned

5. Input segments of AI programming contain?
(a) Sound and smell (b) Touch
(c) Sight and taste (d) All of the mentioned

ANSWER KEY

1. (c) 2. (a) 3. (a) 4. (c) 5. (d)

CHAPTER

2

Problems, Problem Space & Search

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

1. **Define the problem precisely**

   The 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 immerse impact on the appropriateness of various possible techniques for solving the problem.

3. Isolate and represent the task knowledge that is necessarily to solve the problem.

4. Choose the best problem solving technique(s) and apply it to the particular problem.

Define the problem as a state space search

The state space representations forms the basis of most of the AI methods. Its structure corresponds to the structure of problem solving in two important ways.

- It allows for a formal definition of a problem as the need to convert some given situation into some desired situation using a set of permissible operations.
- It permits us to define the process of solving a particular problem as a combination of known techniques (each represented as a rule defining a single step in the space) and search, the general technique of exploring the space to try to find some path from the current state to a goal state. Search is a very important process in the solution of hard problems for which no more direct techniques are available.

Example : A water jug problem : You are given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4-gallon jug ?

The state space for this problem can be described as the set of ordered pairs of integers (x, y), such that $x = 0, 1, 3, 4$ and $y = 0, 1, 2, 3$. x represents the number of gallons of water in the 4-gallon jug and y represents the quantity of water in 3 gallon jug.

The start state is (0, 0) and the goal state is $(2, n)$ where n may be 0, 1, 2 or 3.

The operator to solve the problem is given in figure 2.1. There are several sequence of operators that solve the problem. One sequence is shown in figure 2.2.

1.	(x, y) if $x < 4$	$\rightarrow (4, y)$	Fill the 4-gallon jug
2.	(x, y) if $y < 3$	$\rightarrow (x, 3)$	Fill the 3-gallon jug

3.	(x, y) if $x > 0$	$\rightarrow (x-d, y)$	Pour some water out of the 4-gallon jug
4.	(x, y) if $y > 0$	$\rightarrow (x, y-d)$	Pour some water out of the 3-gallon jug
5.	(x, y) if $x > 0$	$\rightarrow (0, y)$	Empty the 4-gallon jug on the ground
6.	(x, y) if $y > 0$	$\rightarrow (x, 0)$	Empty the 3-gallon jug on the ground
7.	(x, y) if $x + y \geq 4$ and $y > 0$	$\rightarrow (4, y-(4-x))$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full
8.	(x, y) if $x + y \geq 3$ and $x > 0$	$\rightarrow (x-(3-y), 3)$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
9.	(x, y) if $x + y \leq 4$ and $y > 0$	$\rightarrow (x+y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10.	(x, y) if $x + y \leq 3$ and $x > 0$	$\rightarrow (0, x+y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11.	$(0, 2)$	$\rightarrow (2, 0)$	Pour the 2-gallons from the 3-gallon jug into the 4-gallon jug
12.	$(2, y)$	$\rightarrow (0, y)$	Empty the 2-gallons in the 4-gallon jug on the ground

Fig. 2.1 Production Rules for the Water jug Problem

Gallon in the 4-gallon jug	Gallon in the 3-Gallon jug	Rule Applied
0	0	
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 or 12
2	0	9 or 11

Fig. 2.2 One Solution to the Water jug Problem

Example : Missionaries and Cannibals:

- **Problem Statement:** Three missionaries and three cannibals want to cross a river. There is a boat on their side of the river that can be used by either one or two persons.

  • How should they use this boat to cross the river in such a way that cannibals never outnumber missionries on either side of the river? If the cannibals ever outnumber the missionaries (on either bank) then the missionaries will be eaten. How can they all cross over without anyone being eatent?
- State space for this problem can be described as the set of ordered pairs of left and right bank of the river as (L, R) where each bank is represented as a list [nM, mC, B].

  • n is the number of missionaries M, m is the number of cannibals C, and B represents boat.
- **Start State:** ([3M, 3C, 1B], [0M, 0C, 0B]),

  • 1B means that boat is present and 0B means it is not there on the bank of river.

- **Any State:** ([n_1M, m_1C, 1B], [n_2M, m_2C, 0B]), with constraints/conditions as $n_1(\neq 0) \geq m_1; n_2(\neq 0) \geq m_2; n_1 + n_2 = 3, m_1 + m_2 = 3$

  Goal State: ([0N, 0C, 0B], [3M, 3C, 1B])
- By no means, this representation is unique.
- In fact one may have number of different represenatations for the same problem.
- The table on the next side consists of production rules based on the chosen representation.

Set of Production Rules

Applied keeping constraints in mind

RN	**Left side of rule**	→	**Right side of rule**
Rules for boat going from left bank to right bank of the river			
L1	([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])	→	([(n_1-2)M, m_1C, 0B], [(n_2+2)M, m_2C, 1B])
L2	([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])	→	([(n_1-1)M, (m_1-1)C, 0B], [(n_2+1)M, (m_2+1C), 1B])
L3	([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])	→	([n_1M, (m_1-2)C, 0B], [n_2M, (m_2+2C), 1B])
L4	([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])	→	([(n_1-1)M, m_1C, 0B], [(n_2+1)M, m_2C, 1B])
L5	([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])	→	([n_1M, (m_1-1)C, 0B], [n_2M, (m_2+1C), 1B])
Rules for boat coming from right bank to left bank of the river			
R1	([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])	→	([(n_1+2M)M, m_1C, 0B], [(n_2-2)M, m_2C, 0B])
R2	([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])	→	([(n_1+1M)M, (m_1+1)C, 1B], [(n_2+1)M, (m_2-1)C, 0B])
R3	([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])	→	([n_1M, (m_1+2)C, 1B], [n_2M, (m_2-2)C, 0B])
R4	([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])	→	([(n_1+1)M, m_1C, 0B], [(n_2-1)M, m_2C, 0B])
R5	([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])	→	([n_1M, (m_1+1)C, 0B], [n_2M, (m_2+1)C, 0B])

One of the possible paths

Start →	**([3M, 3C, 1B], [0M, 0C, 0B]]**	
L2:	([2M, 2C, 0B], [1M, 1C, 1B])	1M, 1C →
R4:	([3M, 2C, 1B], [0M, 1C, 0B])	1M ←
L3:	([3M, 2C, 1B], [0M, 3C, 1B])	2C →
R4:	([3M,1C, 1B], [0M, 2C, 1B])	1C ←
L1:	([1M, 1C, 0B], [2M,2C, 1B])	2M →
R2:	([2M, 2C, 1B], [1M,1C, 0B])	1M,1C ←
L1:	([0M, 2C, 0B], [3M,1C, 1B])	2M →
R5:	([0M, 3C, 1B], [3M,0C, 0B])	1C ←
L3:	([0M, 1C, 0B], [3M,2C, 0B])	2C ←
R5:	([0M, 2C, 1B], [3M,1C, 0B])	1C ←
L3:	**([0M, 0C, 0B], [3M,3C, 1B])**	**2C → Goal state**

Example : The 8–Puzzle problem

Problem Statement

- The eight puzzle problem consists of 3 x 3 grid with 8 consecutively numbered tiles arranged on it.
- Any tile adjacent to the space can be moced on it.
- Solving this problem invoIes arranging tiles in the goal state from the start state.

Start State

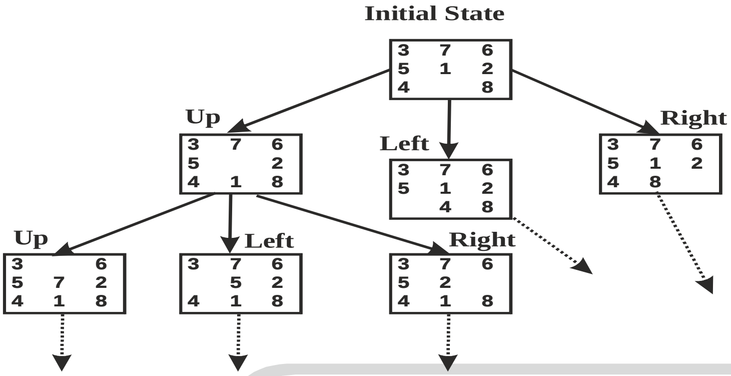
3	7	6
5	1	2
4	□	8

Goal State

5	3	6
7	□	2
4	1	8

Solution by State Space Method:

- The start State could be represented as :
  [[3,7,2], [5, 1, 2], [4, 0, 6]]
- The goal State could be represented as :
  [[5, 3, 6], [7, 0, 2], [4, 1, 8]]
- The operators can be thought of moving {up, down, left, right}, the direction in which blank space effectively moves.
  [[3,7,2], [5, 1, 2], [4, 0, 6]]



In order to provide a formal description of a problem, the following steps has been performed:

1. Define the state space that contains all the possible configurations of the relevant objects (and some impossible ones). It is, of course, possible to define this space without explicitly enumerating all of the states it contains.
2. Specify one or more states within that space that describes possible situations from which the problem solving process may start. These states are called the initial states.
3. Specify one or more states that would be acceptable as solutions to the problem. These states are called *goal states*.
4. Specify a set of rules that describe the actions (operators) available. Doing this will require giving thought to the following issues:
   - What unstated assumptions are present in the informal problem description ?
   - How general should the rules be ?
   - How much of the work required to solve the problem should be precomputed and represented in the rules ?

The problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from initial state to a goal state is found. Thus the process of search is fundamental to the problem-solving process.

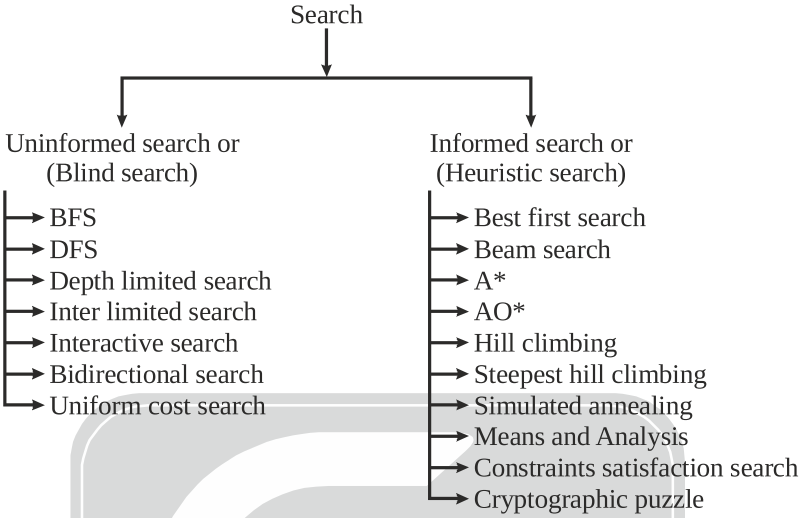
A production system consists of :

- 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 whatever information is appropriate 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 any appropriate way.
- A control strategy that specifies the order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.
- A rule applier.

It also encompasses a family of general production system interpreters, including:

- Basic production system languages, such as OPS5 and ACT*.
- More complex, often hybrid systems called expert system shells, which provide complete (relatively speaking) environments for the contruction of knowledge based expert system.
- General problem solving architectures like SOAR [Laired *et al*., 1987], a system based on a specific set of cognitively motivated hypotheses about the nature of problem solving.

Problem solving by search:



Problem solving by search:

The problem (search space) can defined as four components (S, &, O, G)

Where, S $\rightarrow$ set of states represented as node

$\& \in S \rightarrow$ initial/starting states from where the agent initiates its search process.

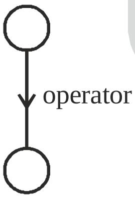
O $\rightarrow$ the operator/action that is used to move the agent from one states to next state (successer state)

$O : S \rightarrow S$ $\quad$ $O(P) = q$

where, P is a state on which operator action is applied.

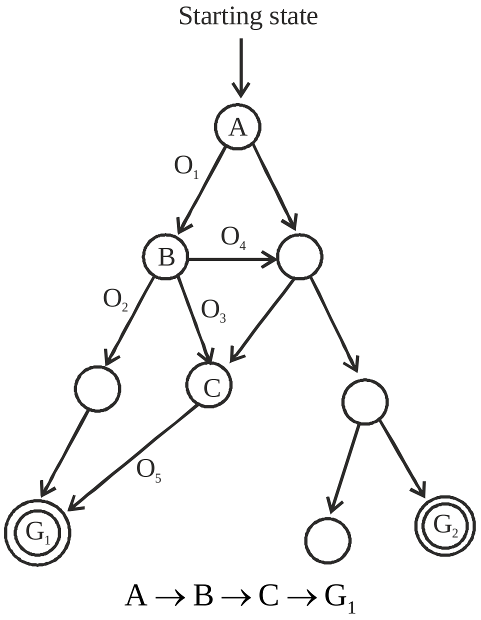
O is a function where takes S (i.e. state) to the initial node.

The operators are represented by an edge in the state space graph.



$G \subseteq S$: The set of goal states|derived states.

Agent is an object who searches the goal node.



Plan: The sequence of action i.e. (O_1, O_3, O_5)

Solution plan: The sequence of action which leads you from initial to goal state i.e. (O_1, O_2, O_5).

Cost of plan: Sum of cost of all the operators in the plan.

$$\text{Cost}(O_1, O_3, O_5) = \text{cost}(O_1) + \text{cost}(O_3) + \text{cost}(O_5)$$

General Frame Work for search:

(1) Initialize : OPEN = {&}   CLOSED = {}

(2) Fail : If OPEN is empty terminate with failure

(3) Select : Pick a state from OPEN and place to the CLOSED.

(4) Terminate : Terminate with success.

(5) Expand : Find the successor on of n and if $m \notin$ OPEN U CLOSED then place it into OPEN.

(6) Loop: goto to step 2.

BLIND SEARCH/UNINFORMED SEARCH

When there is no information available regarding the search space, such type of search is known as blind/ uninformed search.

1. Breadth First Search (BFS):

- We can implement it by using two lists called OPEN and CLOSED.
- The OPEN list contains those states that are to be expanded and CLOSED list keeps track of states already expanded.
- Here OPEN list is used as a **queue.**

Algorithm (BFS):

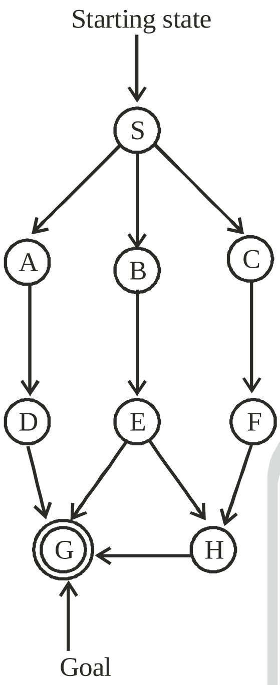
- **Input :** Two states, START and GOAL
- **Local Variables:** OPEN, CLOSED, STATE–X, SUCCESSORS
- **Output :** Yes or NO

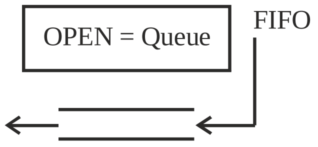
Method :

```
• Initially OPEN list contains a START node and CLOSED list is empty,  Found = false;
• While (OPEN $\neq$ empty and found = false)
Do {
        Remove the first state from OPEN and call it STATE-X;
        Put STATE-X in the front of CLOSED list;
If STATE-X=GOAL then Found = true else
        {-perform EXPAND operation on STATE-X, producing a list of SUCCESSORS;
        - Remove from SUCCESSORS those staes, if any, that are in the CLOSED list;
        - Append SUCCESSORS at the end of the OPEN list/*queue*/
        } } /* end while */
```

- If found = true then return **YES** else return **No** and Stop

e.g.





OPEN ← S   Closed { }

OPEN ← A B C   Closed {S}

OPEN ← B C D   Closed {S,A}

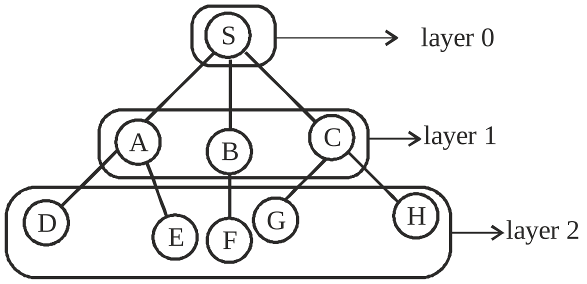
OPEN ← C D E   Closed {S, A, B}

OPEN ← D E F   Closed {S, A, B, C}

OPEN ← E F G H   Closed {S, A, B, C, D, E, F, G}

It is BFS (Breadth First Search)

Graph in a $\boxed{\text{layer wise manner}}$

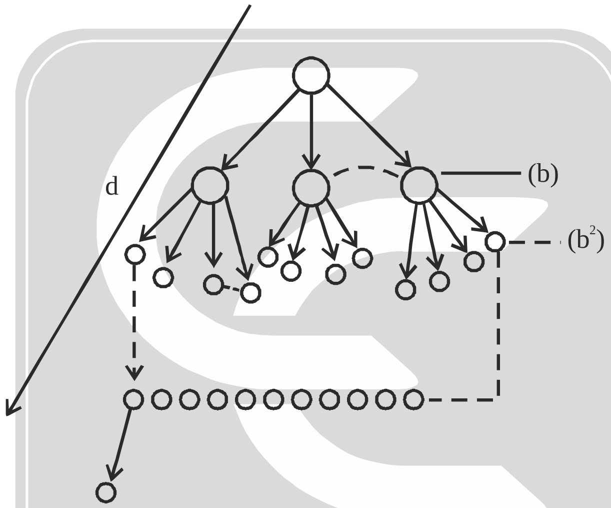


If the branching factor = b and depth of the goal = d

$\boxed{\text{Branching factor}(b) \text{ is a max number of child}}$

Time complexity = # of node accessed during the traversal = $\boxed{O\left(b^d\right)}$

Space complexity: Max size of the OPEN list.



Time complexcity:

$$1+b+b^2+......+b^{d-1}+b^d=\frac{b^{d+1}-1}{b-1}=O\left(b^d\right)$$

Space complexcity:

$$\boxed{O\left(b^d\right)}$$

It increases exponentially

2. Depth First Search

- In depth-first search we go as far down as possible into the search tree/graph before backing up and trying alternatives.
- It works by generating a descendent of the most recently expanded node until some depth cut off is reached and then backtracks to next most recently expanded node and generates one of its descendants.
- It avoids memory limitation as it only stores a singles path from the root to leaf node along with the remaining unexpndable siblings for each node on the path.
- Again use two lists called OPEN and CLOSED with the same conventions explained earlier.
- Here OPEN list is used as a **stack.**
- If we discover that first element of OPEN is the **Goal** state, then search terminates successfully.

- We could get track of our path through state space as we traversed but in those situations where many nodes after expansion are in the closed list, then we fail to keep track of our path.
- This information can be obtained by modifying CLOSED list by putting pointer back to its parent in the search tree.

Algorithm (DFS)

Input : Two states, START and GOAL

Local Variables: OPEN, CLOSED, RECORD–X, SUCCESSORS

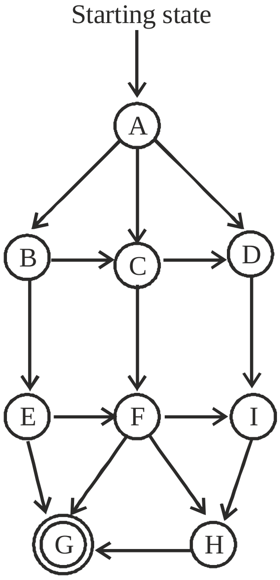
Output: A path sequence, if one exists, otherwise return No.

Method: OPEN, CLOSED, RECORD–X, SUCCESSORS

```
Form a stack consisting of (START, nil) and call it OPEN list.
Initially set CLOSED list as empty;
Found = false;
While (OPEN ≠ empty and found = false) DO
{
Remove the first state from OPEN and call it RECORD-X
Put RECORD-X in the front of CLOSED list;
If the state variable of RECORD-X=GOAL,
then Found- true
Else
{
Perform EXPAND opertation on STATE-X, a state variable of RECORD-X, producing a list of action records called SUCCESSORS;
create each action record by associating with each state its parent.
 Remove from SUCCESSORS any record whose state variables are in the record already in the CLOSED list.
Insert SUCCESSORS in the front of the OPEN list /*Stack*/
}
} /*end while*/
 If Found = true
then return the plan used /* find it by tracing through the pointers on the CLOSED list */else retum No
Stop
```

DFS:

OPEN = STACK



Here, we can also follow the path from A→B→C→D→I........ then the cost may be difference it we follow difference path.

OPEN | A | → Closed { }

OPEN | D | C | B | → Closed {A}

OPEN | D | C | E | → Closed {A, B}

OPEN | D | C | F | G | → Closed {A, B, E}

OPEN | D | C | F | → Closed {A, B, E, G}

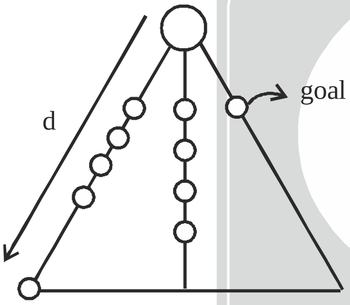
G is the goal node

This is the depth first search (DFS).

In DFS we go in on direction deeper untill we hit to a dead end. After hitting to the dead end we need to back track.

If the branching factor is b and depth of the tree is m

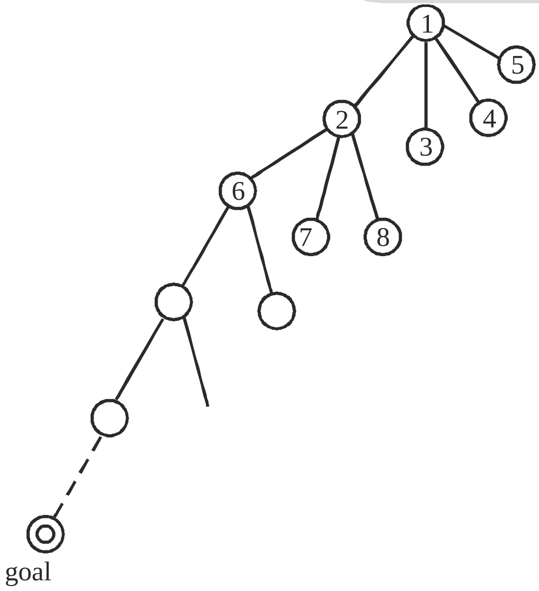
Time complexity: $O\left(b^m\right)$

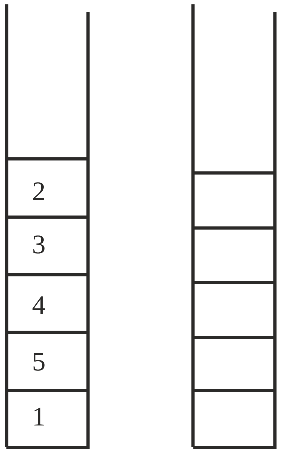


Space complexity :

$\boxed{O\left(bm\right)}$

+ b + b + b + ... m times.





Comparisons : (These are unguided/blind searches, we can not say much about them)

- DFS is effective when there are few sub trees in the search tree that have only one connection point to rest of the states.
- DFS is best when the GOAL exist in the lower portion of the search tree
- DFS can be dangerous when the path closer to the START and further from the GOAL has been chosen.
- DFS may no find solution, even if it exists
- DFS may go to infinite loop if rules applied are left recursive.
- BFS is effective when the search tree has a low branching factor.
- BFS required a lot memory as number of nodes in level of the tree increases exponentially as compared to DFS.
- BFS is superior when the GOAL exists in the upper right portion of a search tree.

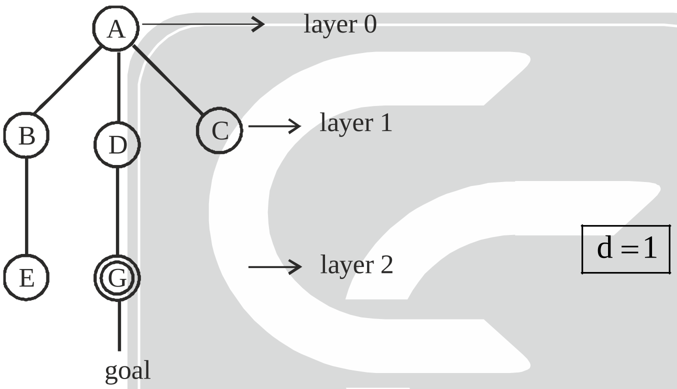
3. Depth Limited Search:

Apply the search only to the specified depth = d. It may follow BFS and DFS.

Successful: If the good is within the depth

Unsuccessful: If the good is at depth $> d$.

DFS :



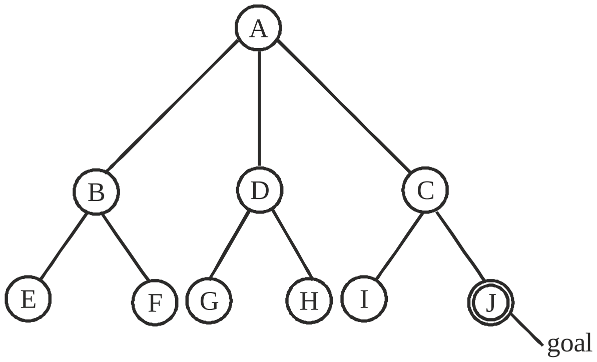
It can only search at the level 1 because depth (d) = 1. It goes $A \to B$ then backtrack A and then $A \to D$ then $A \to C$.

Problem with this search is that if the goal is not with in specified search space then it does not find the goal. Solution for this search is that we increase the depth (d) by 1 i.e. search is called iterative deepening search.

Iterative Deepering Search:

We increase the depth in each iteration if the goal is not achieved.

DFS follows:



$d = 0 \Rightarrow A \rightarrow$ unsuccessful

$d=1 \Rightarrow$ A,B,D,C $\rightarrow$ unsuccessful

$d = 2 \Rightarrow A, B, E, F, D, G, H, C, I(J) \rightarrow$ unsuccessful

Here, we visit the root node d + 1 times and next level is d times.
If the branching factor = b and goal is at depth d.

$$\text{Time complexity} = 1.b^{d} + 2.b^{d-1} + 3.b^{d-2}.....+\left(d+1\right)\times 1$$

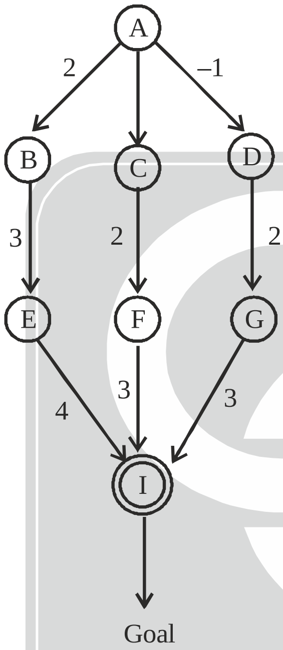
$$\boxed{\text{Time compl.} = O\left(b^{d}\right)}$$

Solution Plan:

(1) $A \xrightarrow{2} B \xrightarrow{3} E \xrightarrow{4} I = 9$

(2) $A \xrightarrow{1} C \xrightarrow{2} F \xrightarrow{3} I = 6$

(3) $A \xrightarrow{-1} D \xrightarrow{2} G \xrightarrow{3} I = 4$



Path with cheapest Cost is the shorted path.

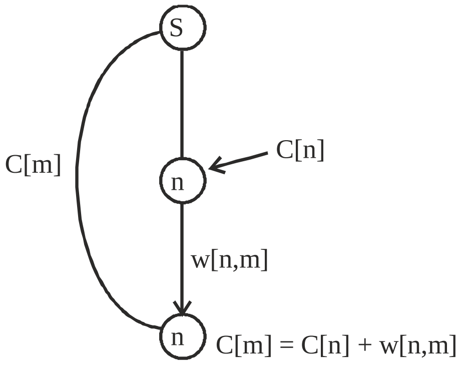
4. Uniform Cost search: It uses only historical value.
(1) Initiative: OPEN = { } CLOSED = { } C[&] = 0
(2) Fail: If open is empty the terminate the search with failure.
(3) Select : Select a state n with minimum C value and put it into closed.
(4) Terminate : If n is a goal then terminate with success.
(5) Expand : For all the successor of n for a succesor m of n if $m \notin \text{OPEN} \cup$ CLOSED C[m] = C[n] + w[n,m] and put n inito open.

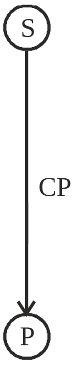
If $m \in \text{OPEN} \cup \text{CLOSED}$ then set

$$C[m] \leftarrow \min\left\{C[m], C[n] + w[n,m]\right\}$$

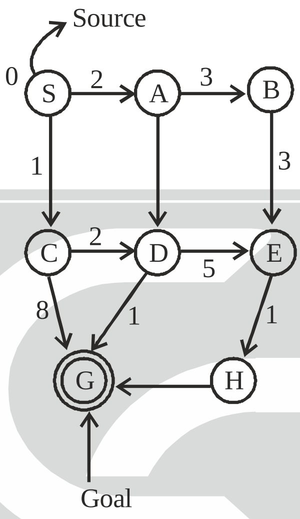
If C[m] has declared and $m \in$ closed then remove m from closed and put it into the OPEN.

Loop goto step # 2:





C(P) : the cost of the vertices from source vertex path to vertices P



$\text{Open} = \{S^0\}, \qquad \text{closed} = \{\ \}$

$\text{Open} = \{A^2, C^1\}, \qquad \text{closed} = \{S^0\}$

$\text{Open} = \{A^2, D^3, G^9\} \qquad \text{closed} = \{S^0, C^9\}$

Out the node which has min cost value i.e.

$\text{Open} = \{D^3, G^9, B^5\}, \quad \text{closed} = \{S^0, C^1, A^2\}$

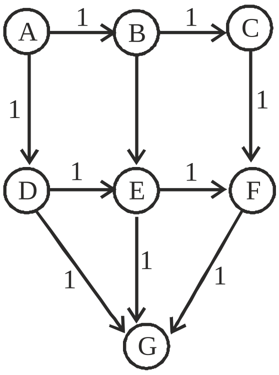
$\text{Open} = \{G^7, B^5, E^8\}, \quad \text{closed} = \{S^0, C^1, A^2, D^3\}$

$\text{Open} = \{G^4, B^5, E^8\}, \quad \text{closed} = \{S^0, C^1, A^2, D^3\}$

$\text{Open} = \{B^5, E^4\}, \qquad \text{closed} = \{S^0, C^1, A^2, D^3, G\}$

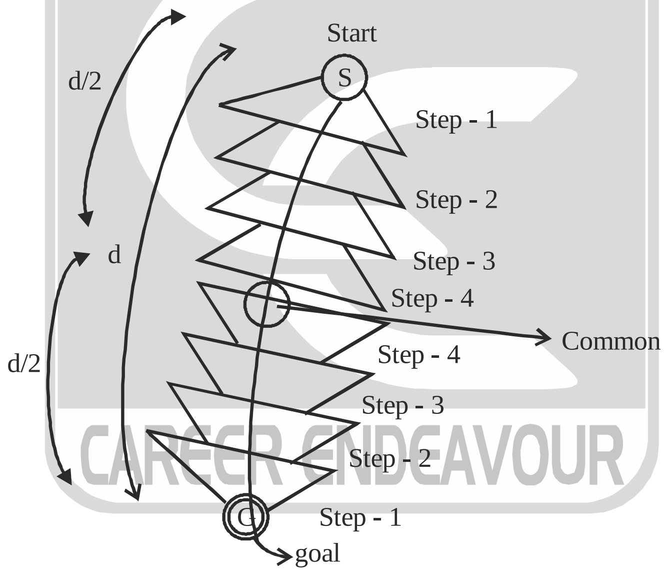
$\boxed{G^4 = \text{cost of the path} = 4}$

Note: If all the cost of edges in a graph are equal then UNIFORM cost search behaves like BFS or also in case if cost are absent.



Bi-Directional Search

- For those problems having a single goal state and single start state, bi-directional search can be used.
- It starts searching forward from initial state and backward from the goal state simultaneously starting the states generated until a common state is found on both search frontiers.
- DFID can be applied to bi-direction search for k = 1, 2, ... as follows :
- Kth iteration consists of a DFS from direction to depth k storing all states at depth k, and DFS from other direction : one to depth k and other to depth k+1 not storing from forward direction. /*The search to depth k+1 is necessary to find odd-length solutions */.



It is useful for solving 8 Puzzle.

1	2	3
8		4
7	6	5

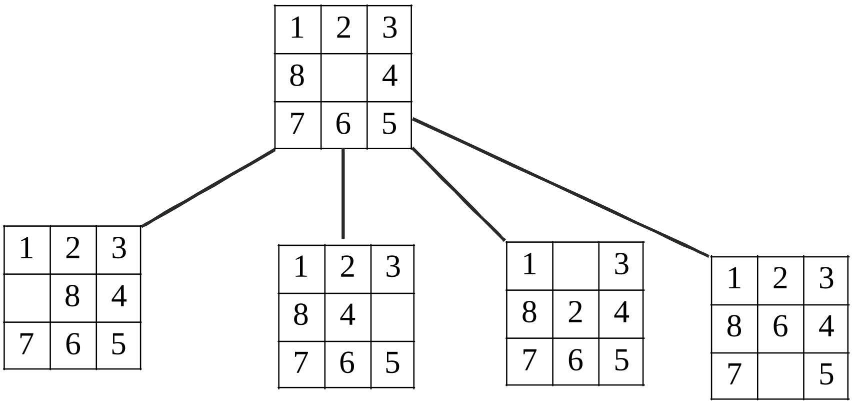
⟶

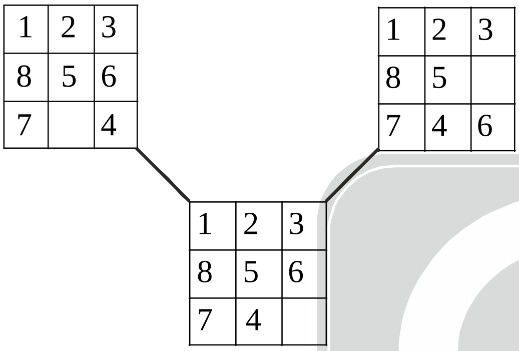
1	2	3
8	5	6
7	4	

Goal state

Operator :

(1) Left slide (2) Right slide (3)Up and (4) Down





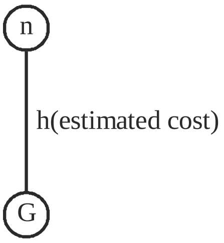
INFORMATION/HEURISTIC SEARCH

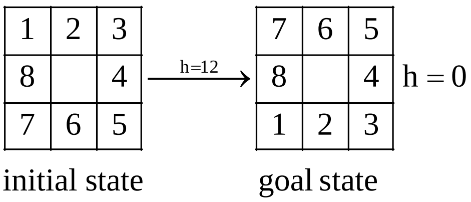
A heuristic function is a function that maps from problem state description to measures of desirability, usually represented as numbers.

Well defined heuristic functions can play on important role in efficiently guiding a search process toward a solution. The purpose of heuristic function is to guide the search process in the most profitable direction by suggesting which part to follow first when more than one is available.
It uses the domain specific information to estimate the quality or the potential of the partial solution.

- Search specific means estimated cost from current node to goal node.
- Every node is assocaited with a Heuristic value.

Heuristic Value: The estimated cost of moving from the current node to the goal node.





(1) h = # of digits or tiles which are out of place = 6

(2) Manhatton distance:

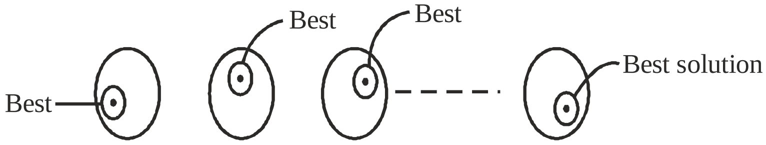
$$\begin{array}{cccccccccccccccc} 2 & + & 2 & + & 2 & + & 0 & + & 2 & + & 2 & + & 2 & + & 0 & = 12 \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\ 1 & + & 2 & + & 3 & + & 4 & + & 5 & + & 6 & + & 7 & + & 8 & \end{array}$$

Two types of heuristic:

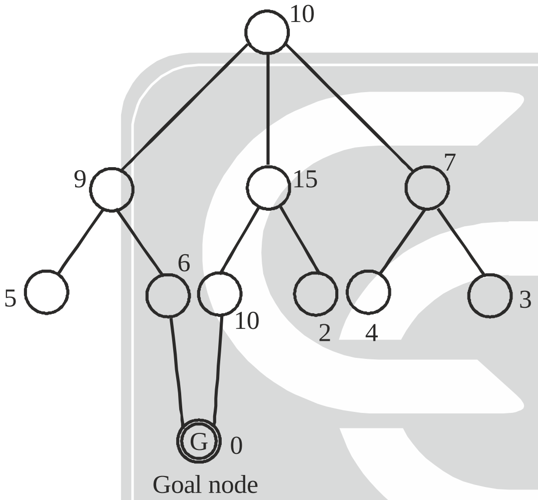
(1) Accurate: Estimated cost is right

(2) Inaccurate: Estimated cost is not correct.

1. Best first search: The next successor is choosen as a node having Best Heuristic value. It considers only the future (Greedy approach)



In the greedy aproach we consider locally best solution.

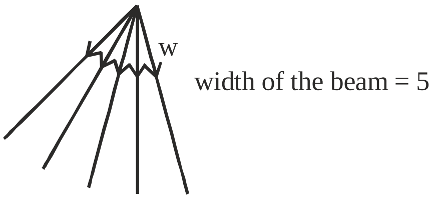


Algorithm: Best-First Search:

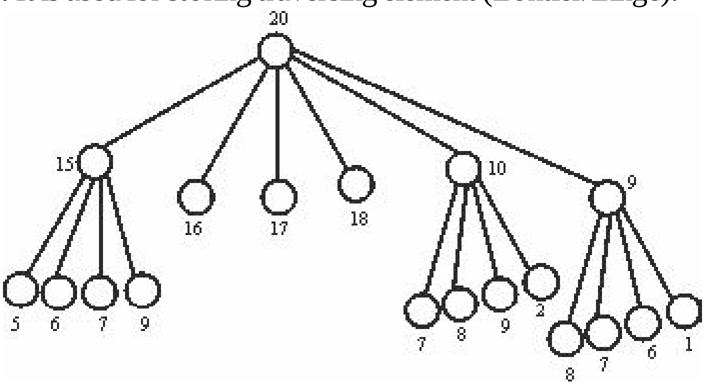
```
1. Start with OPEN containing just the initial state.
2. Until a goal is found or there are no nodes left as OPEN do:
        Pick the best node on OPEN
        Generate its successors.
        For each succesor do:
                If it has not been generated before, evaluate it, add to OPEN, and record its parent.
                If it has been generated before, change the parent if this new path is better than the previous
                one. In that case, update the cost of getting to this node and to any successors that this node
                may already have.
```

Note: Best-First Search is Hill Climbing where we compare Heuristic value not only among same level node but also with lower level nodes.

2. Beam Search: Here each possibility is checked for the desired solution. It keeps track of k states rather than just 1 state.



OPEN : It is used for storing traversing element (frontier/fringe).



1	2	6
3		7
5	8	4

$\xrightarrow{h=10}$

1	8	7
2		6
3	4	6

8 puzzle problem   goal state

0 + 2 + 1 + 1 + 2 + 1 + 1 + 2 = 10

↓ Left

1	2	6
	3	7
5	8	4

h = 11

↓ Right

1	2	6
3	7	
5	8	4

h = 11

↓ Up

1		6
3	2	7
5	8	4

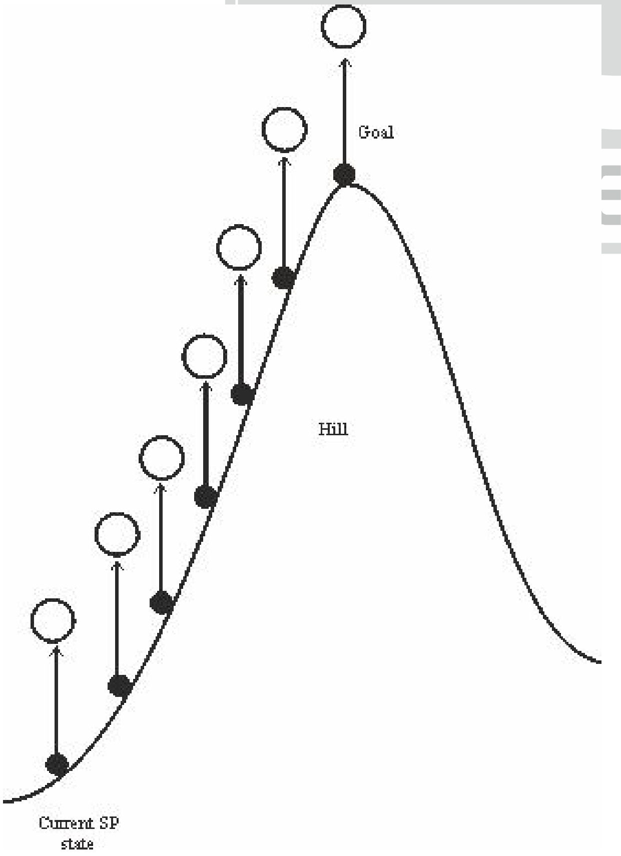
h = 9

↓ Down

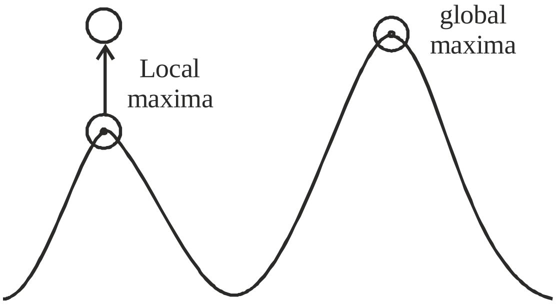
1	2	6
3	8	7
5		4

h = 9

3. Hill Climbing:



If successor is better than current state then it may becomes current state. This is perform till we reach the goal.



Hill Climbing

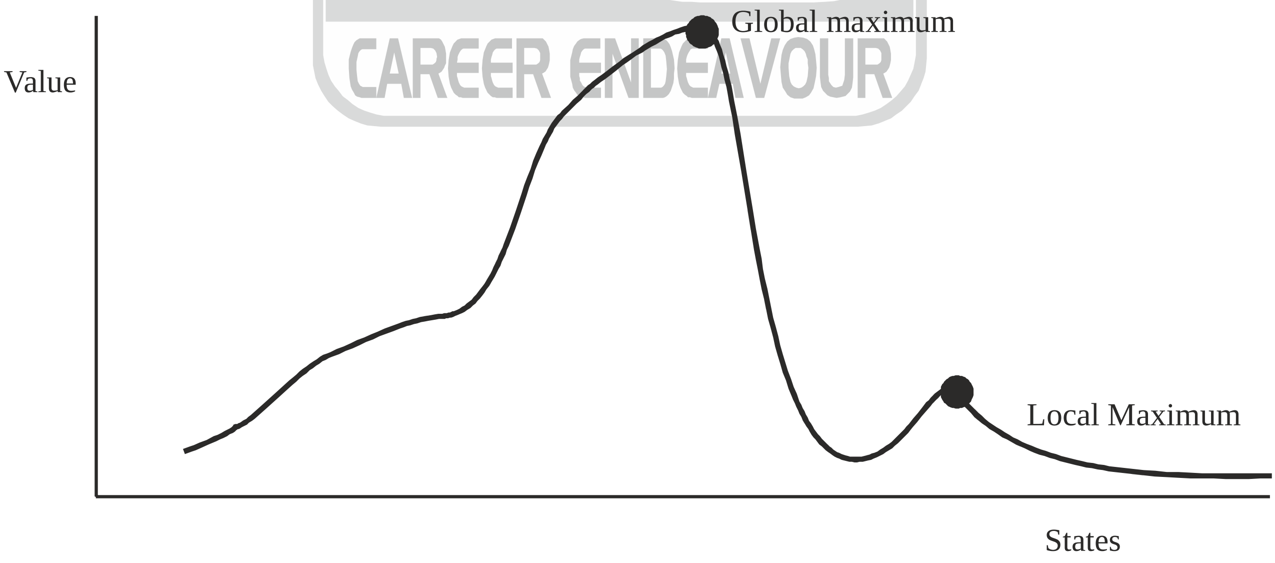
Algorithm

```
Function Hill-Climbing(problem)
    Returns a local maximum
    Inputs: problem
    Local variables: current (a node)
                  neighbour (a node)
    Current <- make-node(initial-state[problem])
    Loop do
                Neighbour <- highest valued successor of current
                If value(neighbor) <- value(current) return state(current)
                   current <- neighbour
                   End.
```

Note: So Hill Climbing is Heuristic DFS.

Disadvantages of Hill Climbing:

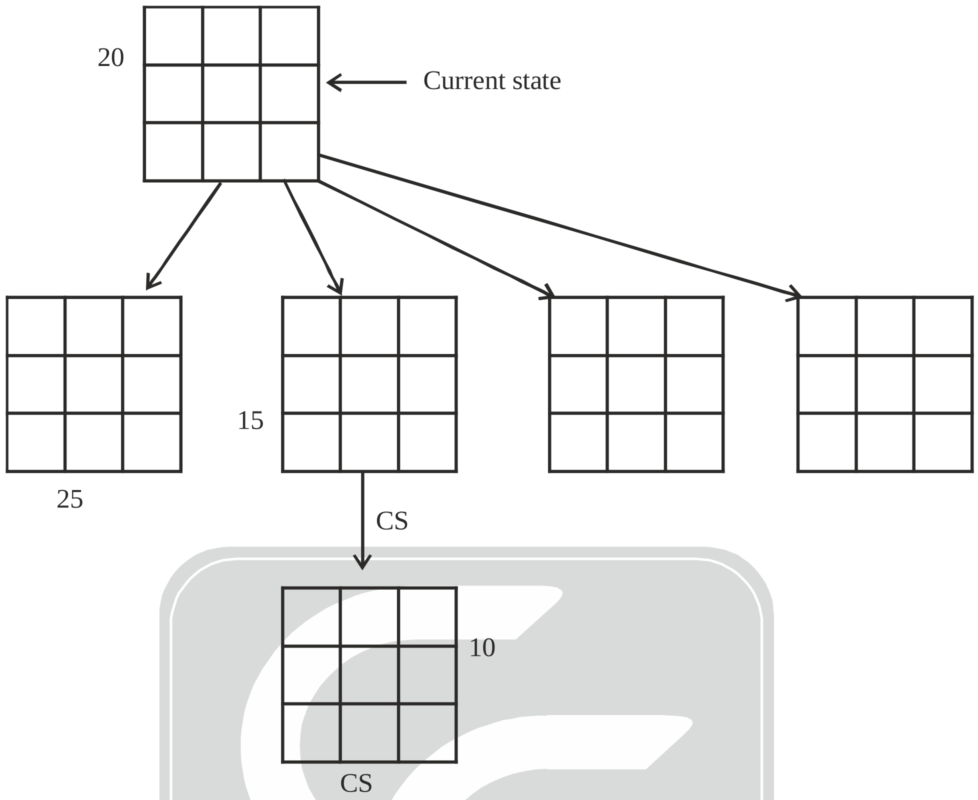
- Local maxima: A local maximum is a peak that is higher than each of its neighboring states, but lower than the global maximum
- Ridges: a sequence of local maxima
- Plateaux: an area of the state space landscape where the evaluation function is flat.



Solution for local maxima problem:

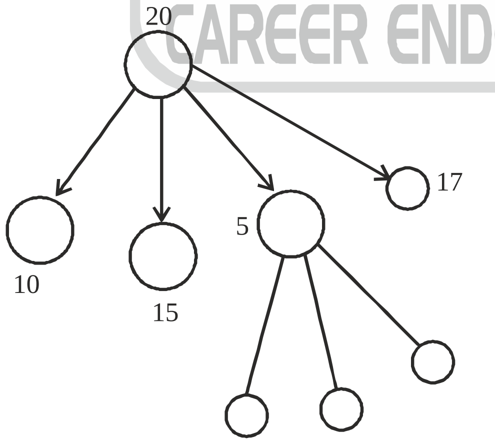
Backtrack and start from another suitable position.

Simplest Hill Climbing: We move that state which having lower h-function from current state and make that new current state.



In the simplest hill climbing, we generate the first node and compare with this current state if its h-function is lower, then choose it and move that state. We cannot traverse all node.

Steepest ascent hill climbing:

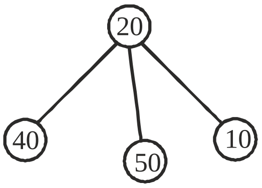


In the steepest ascent hill clombing use first, generate all the successor and then chooses best one and then after generate succesor of that node.

Compartively in steepest ascent hill climbing there is low chance of local maxima problem.

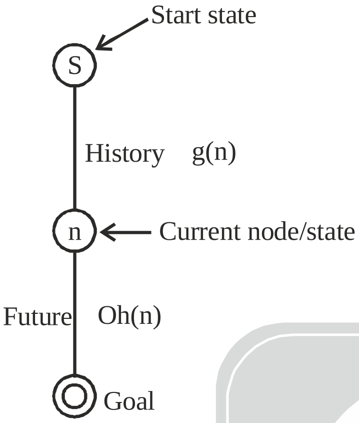
4. Simulated annealing : Generate a random move by $p = e^{-\Delta E/T}$ (Boltzmann formula). Accept if improvement. Otherwise accept with continually decreasing probability.

If initialy temprature is very high then the molecule moves in any direction. So, it is also picks bad direction.



Here in this e.g. choose al the direction 40, 50, 10.

5. A* algorithm : It use both history as well as future for performing action.



A* = Uniform cost search + best first search

$$f(n) = g(n) + h(n)$$

where, g(n) is cost of initial state to node and h(n) is heuristic function i.e. cost from node to goal.
In A* algorithm if h = 0 for all nodes A* becomes uniform cost search.

In A* algorithm if n = 0 and the cost of the operators are equal then A* becomes. Breath First Search (BFS).

if g = 0 in A* then A* behaves likes : Best First Search.

A* algorithm:

(1) Initialise : OPEN = {&} CLOSED = { }
f(S) = h(&) ; g(s) = 0
(2) Fail : If open is empty then return with failure.
(3) Select : Select a state n from OPEN having minimum value and put it into CLOSED.
(4) Terminate : If n is a goal node then terminate with success.
(5) Expand : Generate all the successor m of n for a successor m of n.
if $m \notin$ OPEN $\cup$ CLOSED

$$g(m) = g(n) + C(n,m)$$

$$f(m) = cg(m) + h(m)$$

put m into OPEN
Step-2: If $m \in$ OPEN $\cup$ CLOSED then

$$g(m) = \min\left(g(m),\, g(n) + c(n,m)\right)$$

$$f(m) = g(m) + h(m)$$

If f(n) is decreased and $m \in$ CLOSED then move m from CLOSED to OPEN.

Loop: goto step 2

$h(s) \leftarrow \& \rightarrow$ starting vertex

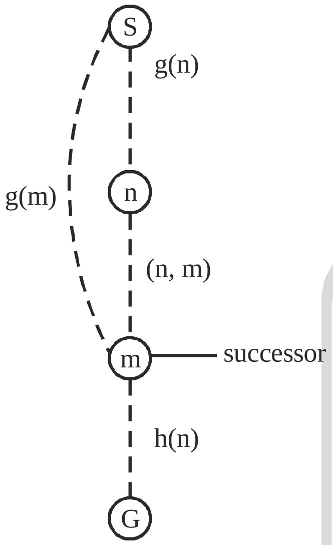
$g(s) = 0$

$f = g + h$

$f(s) = g(s) + h(s)$

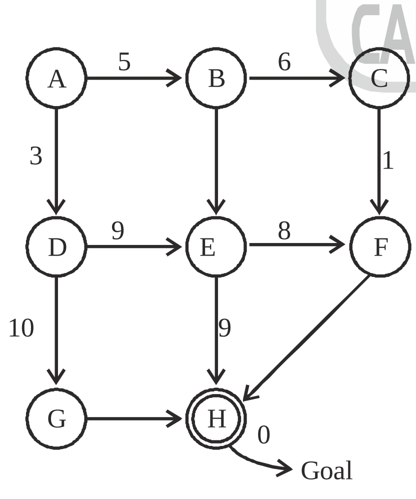
$f(s) = 0 + h(s)$

$f(s) = h(s)$ at initial



$$g(m) = g(n) + c(n,m)$$

Example:



OPEN	CLOSED
A^{18}	
B^{11} O^{17}	A^{18}
B^{11} O^{17}	A^{18}
O^{17} C^{17} E^{9}	A^{18} B^{18}
O^{17} C^{17} F^{16} H^{17}	A^{11} B^{11} E H^{12}

Cost = time complexitiy = $O(b^a)$

Space complexity = $O(b^a)$:

A* algorithm always gives a solution if heuristic always understimate.

$$\text{if } h(n) < f^*(n) \ \forall \text{ all nodes } n$$

Proof of the optimality of A*.

Assume: h admissible, f non-decreasing along any path from the root.
Let G be an optimal goal state, with path cost f*.

Let G_2 be a suboptimal goal state, with path cost $g(G_2) > f^*$.

n is a leaf node on an optimal path to G.

Becasue h is admissible, we must have

$$f^* \geq f(n)$$

Also, if n is not chosen over G_2, we must have

$$f(n) \geq f(G_2)$$

Gives us $f^* \geq f(G_2) = g(G_2)$. (Then G_2 is not suboptimal!)

AND-OR GRAPHS

Useful for certain problems where

- The solution involves decomposing the problem into smaller problems.
- We then solve these smaller problems.
- Here the alternatives often involve branches where some or all must be satisfied before we can progress.

AND OR graph has 2 different kind of nodes AND node and OR mode.

AND node : It represent a decomposition of a problem.

OR Node : It represents a possible decomposition to multiply the matrix ABC in such a way the cost is minimum.

$$A = 5 \times 6$$

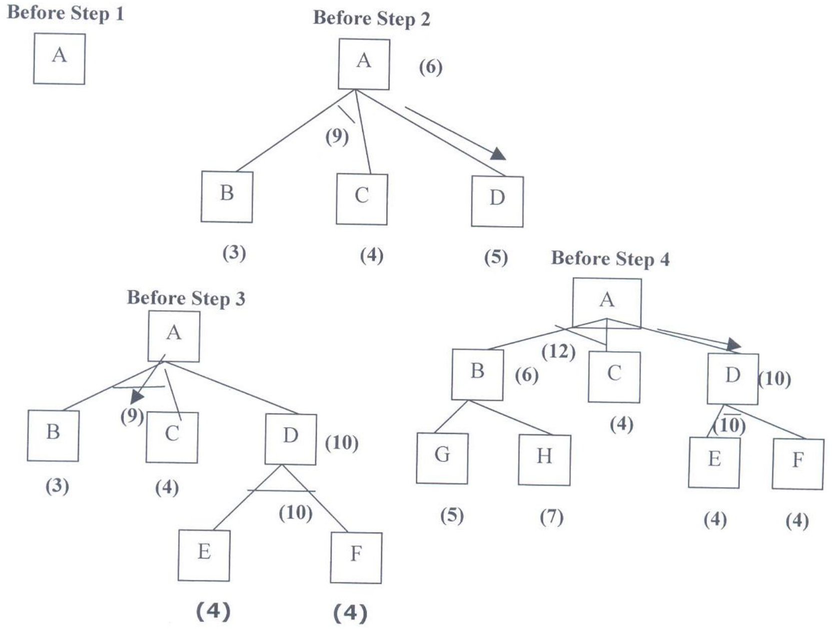
$$B = 6 \times 7$$

$$C = 7 \times 8$$

$$\left.\begin{array}{l} A \to m \times n \\ B \to n \times p \end{array}\right\}$$

Number of multiplication for AB = mnp

Example of Searching in an AND-OR Graph.



6. AO* algo:

In AND node all successor are marked and in OR node only best successor is marked.

(1) Inilialise: G = <S>; f(s) = h(s)

If S is terminal label it as solved.

(2) Terminate : If s is labelled as solved then terminate.

(3) Select : Select a non leaf node n from the marked subtree of G

(4) Expand: Generate all the successor m of n and set f(m) = h(m)

If m is a terminal node lebel it as solved.

(5) Cost revision : call cost revision (n).

Cost revision (n):

(i) Let $z = \langle n \rangle$

(ii) If z is ϕ, terminate.

(iii) Select a node n from z whose descendent is not in z.

(iv) If n is a AND node with successor r_1, r_2, r_3, ...r_p.

$$f(n) = \sum_{i=1}^{P} f(r_1) + c((n, r_i))$$

Mark the edges corresponding to all successor of n. If all the successors of n are labelled as solved label n as solved.

(v) If n is an OR node with successor r_1, r_2, r_3, ...r_p.

$$f(n) = \min\{f(r_1) + c(n, r_1)\}$$

Mark the best successor of n and if best successor of n is labelled as solved then label n also a solved.

(vi) If f(n) decreases then put those parent of n into z for which it is marked as best successor.

7. Costrained Satisfaction search:

A search in which lots of constraints are defined and these constraints will guide the search move in a particular direction.

Crypt- Arithmetic Puzzels.

Man design tasks can also viewed as constrained satisfaction problems.

Such problems do not require a new search methods but they can be solved using any of the search strategies which can be augmented with the list of constraints that change as parts of the problem are solved.

General form of the constrained satisfaction procedure is as follows

Algorithm:

```
Until a complete solution is found or all paths have lead to dead ends.
{
Select an unexpanded node of the search graph.
Apply the constraint inference rules to the selected node to generate all possible new constraints.
If the set of constraints contain a contradiction, then report that this path is dead end.
If the set of constraint describes a complete solution, then report success.
If neither a contrdiction nor a compete solution has been found, then
        Apply the problem space rules to generate new partial solutions that are consistent with the current
            set of constraints.
        Insert these partial solutions into the search graph.
}
```

Problem: Crypt-Arithmetic Puzzle: Solve the following puzzle by assigning unique digit which satisy the following addition.

Constraints : No two letters have the same value. (The constraints of arithmetic)

Intitial Problem State

$$\begin{array}{ccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \\ \hline \end{array}$$

S=?; E=?; N=?; D=?;, M=?; O=?;, R=?;, Y=?

1. Apply constraint inference rules to generate the relevant new constraints.
2. Apply the letter assignment rules to perform all assignments required by the current set of constraints.
3. Then choose another rules to generate an additional assignment, which will, in turn, generate new constrains at the next cycle.

At each cycle, there may be several choices of rules to apply

A useful heuristics can help to select the best rule to apply first.

For example, if a letter that has only two possible values, then there is a better chance of guessing right on the first than on the second.

This procedure can be implemented as a DF search.

Carries:

C_4=?; C_3=?; C_2=? C_1=?

$C_4 \quad C_3 \quad C_2 \quad C_1 \quad \leftarrow \quad$ *Carry*

$$\begin{array}{ccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \\ \hline \end{array}$$

Constraint Equations:

$Y = D + E \rightarrow C_1$

$E = N + R + C_1 \rightarrow C_2$

$N = E + O + C_2 \rightarrow C_3$

$O = S + M + C_3 \rightarrow C_4$

$M = C_4$

- We can easily see that M has to be non zero digit, so the value of C4 = 1

1. M = C4 $\Rightarrow$ **M = 1**
2. O = S + M + C3 $\rightarrow$ C4

For C4 = 1, S + M + C3 > 9 ⇒

S + 1 + C3 > 9 ⇒ S + C3 > 8.

If C3 = 0, then S = 9 else if C3 = 1,

then S = 8 or 9

- We see that for S = 9

C3 = 0 or 1

It can be easily seen that C3 = 1 is not possible as O = S + M + C3 ⇒ O= 11 ⇒ O has to be assigned to M, so not possible

Therefore, O=0

$\boxed{M=1, O=o}$

Y=D+E → C1
E=N + R + C1 → C2
N=E + O + C2 → C3
O = S + M + C3 → C4
M= C4

3. Since C3 = 0; N = E + O + C2 proceduces no carry.

• As O = 0, N = E+C2.

•Since N ≠ E, therefore, C2 = 1

Hence N = E + 1

• Now E can take value from 2 to 8 {0,1,9 already assigned so far}

• If E = 2, then N=3.

Since C2 = 1, from E = N + R + C1, we get 12 = N + R +C1, we get 12 = N + R + C1

If C1 = 1, then R = 9, which is not possible as we are on the path with S = 9

If C1 = 1, then R = 8, then

From Y = D + E, we get 10 + Y= D + 2

For no value of D, we can get Y.

Try similarly for E = 3, 4. We fail in each case.

• If E =5, then N =6

Since C2 = 1, from E = N + R + C1, we get 15 = N + R + C1,

If C1 = 0, then R = 9, which is not possible as we are on the path with S = 9

If C1 = 1, then R = 8, then

From Y = D + E, we get 10 + Y= D + 5 ile., 5 + Y = D

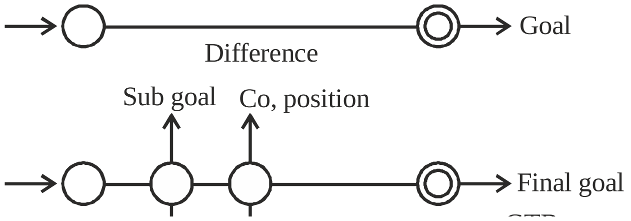
If Y = 2 then D = 7. These values are possible.

• Hence we get the final solution as given below and on backtracking, we may find more solutions.

$$\boxed{\begin{array}{l} S=9;\ E=5;\ N=6;\ D=7; \\ M=1;\ O=0;\ R=8;\ Y=2 \end{array}}$$

8. Means and Analysis:

(1) Find the difference between the goal and current position.



(2) Apply the operator to reduce the difference.

The process of determining the subgoal by applying the operator in a sequenceis called operator subgoaling.

9. MIN-MAX Procedure

Convention :

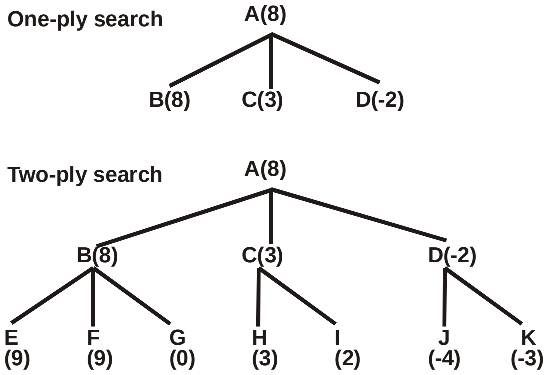
Positive number indicates favor to one player

Negative number indicates favor to other 0, an even match

It operates on game tree and is recursive procedure where a player tries to maximize its own and minimize its opponent's advantage.

The player hoping for positive number is called the maximizing player. His opponent is the minimizing player.

If the player to move is the maximizing player, he is looking for a path leading to a large positive number and his opponent will try to force the play toward situations with strongly negative evaluations.



Values are backed up to the starting position.

The procedure by which the scoring information passes up the game tree is called MIN MAX procedure

The score at each node is either minimum or maximum of the scores at the nodes immediately below.

It is a depth-first, depth limited search procedure.

ALGORITHMIC STEPS

If the limit of search has reached, compute its static value of the current position relative to the appropriate player as given below (Maximizing or Minimizing player). Report the result (value and path).

```
If the level is a maximizing level then
    Generate the successors of current position
    Apply MINIMAX to each of these successors
    Return the maximum of the results.
If the level is a minimizing level then
    Generate the successors of the current position.
    Apply MINIMAX to each of the successors
    Return the minimum of the result
```

MINIMAX (Pos, Depth, Player) Function returns the best path along with best value. It will use the following function.

GEN (Pos) : Generates a list of SUCCs of 'Pos'

EVAL (Pos, Player) : It returns a number respresnting the goodness of 'Pos' for player from the current position.

DEPTH (Pos, Depth) : It is a Booloan function that returns true if the search has reached to maximum depth from the current position otherwise false.

MIN MAX (Pos, Depth, Player):

```
{       If DEPTH(Pos, Depth) then (val = EVAL(Pos, Player), Path = Nil)
        Else
        {       SUCC_List = Nil then return (val = EVAL(Pos,Player), Path = Nil)
                Best_Val = Some minimum value returned by EVAL function;
                For each SUCC ∈ SUCC_List DO
                {       SUCC_Result = MINIMAX(SUCC, Depth + 1, ~Player);
                        NEW_Value = -Val of SUCC_Result;
                        If NEW Value > Best_Val then
                        {       Best_Val = NEW_Value;
```

```
                    Best_Path = Add(SUCC, Path of SUCC_Result);
               }
          }
          Return (Val = Best_Val, Path = Best_Path);
     }  }
```

Example: Evaluation function for Tic-Tac-Toe game

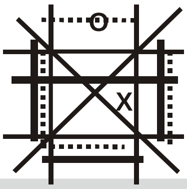
Static evaluation function (f) to position P is defined as:

– If P is a win for MAX, then f(P) = n, (a very large +ve number)

– If P is a win for MIN, then f(P) = -n

– If P is not a winning position for either player, then f(P) = (Total number of rows, columns and diagonals that are still open for MAX) - (total number of rows, columns and diagonals that are still open for MIN)

Consider X for MAX and O for MIN and the following board position P. Now is the turn for MAX.



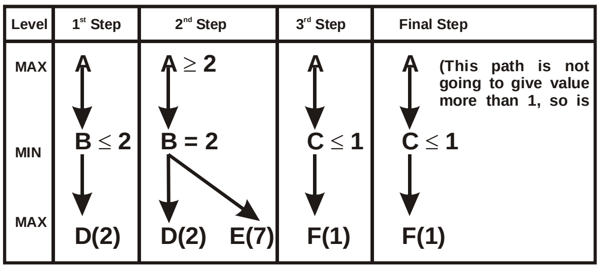
- Total number of rows, columns and diagonals that are still open for MAX (thick line) = 6
- Total number of rows, columns and diagonals that are still open MIN (dotted lines) = 4

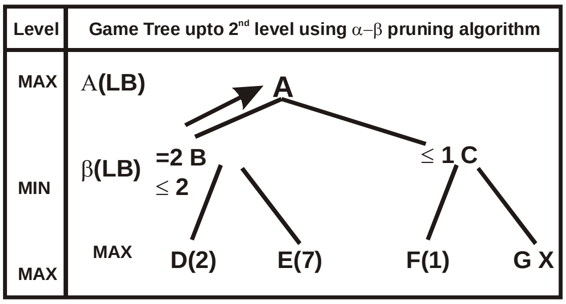
  f(P) = (Total number of rows, columns and diagonals that are still open for MAX) - (total number of rows, columns adn diagonals that are still open for MIN) = 2.

  Remarks:
- Since the MIN MAX procedure is a depth-first process, the efficiency can often be improved by using dynamic branch-and-bound technique in which partial solutions that are clearly worse than known solutions can be abandoned.
- Further, there is another procedure that reduces
  - the number of tree branches to be explored and
  - the number of static evaluation to be applied
- This strategy is called **Alpha-Beta pruning.**

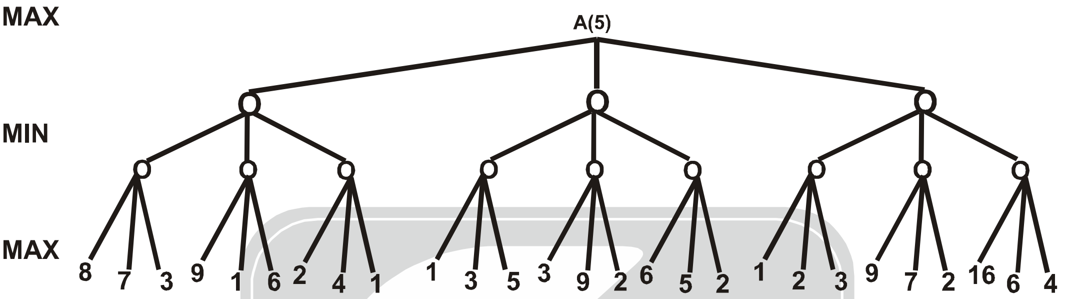
Alpha-Beta Pruning

- It requires the maintenance of two threshold values.
- One representing a **lower bound** (α) on the value that a maximizing node may ultimately be assigned (we call this **alpha*)*** and
- Another representing **upper bound** (β) on the value that a minimizing node may be asigned (we call it **beta**).





- There is no need to explore right side of the tree fully as that result is not going to alter the move decision.
- Given below is a game tree of depth 3 and branching factor 3.
- Find which path should be chosen.



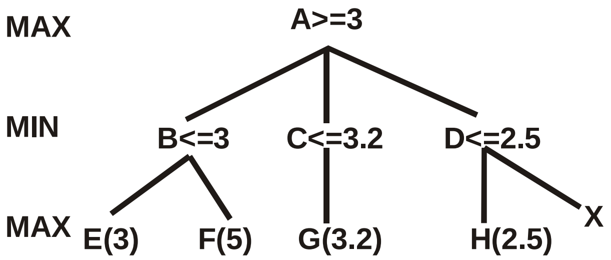
- Given above is a game tree of depth 3 and branching factor 3
- Note that only 16 static evaluations are made instead of 27 required without alpha-beta pruning.

```
Function MINIMAX_αβ(Pos, Depth, Player, Alpha, Beta)
{ If DEPTH (Pos, Depth) then return ({Val = EVAL(Pos, Depth), Path = Nil})
Else
{       SUCC_List = GEN (Pos);
                If SUCC_List = Nil then return ({Val = EVAL(Pos, Depth), Path = Nil})
Else
{ For each SUCC ∈ SUCC_List DO
{ SUCC_Result = MINIMAX_αβ(SUCC, Depth + 1, ~Player, -Beta, - Alpha);
        NEW_Value = -Val of SUCC_Result;
        If  NEW_Value > Beta then
        { Beta = New_Value;
        Best_Path = Add (SUCC, Path of SUCC_Result);
        };
        If Beta ≥ Alpha then Return ({Val = Beta, Path = Best_Path});
        };
Return ({Val = Beta, Path = Best_Path});
}
}
}
```

Remarks:

- The effectiveness of α–β pruning procedure depends greatly on the order in which paths are examined.
- If the worst paths are examined first, then no cut offs at all will occur.
- If posible best paths are known in advance, then they can be examined first.
- It is possible to prove that if the nodes are perfectly ordered then the number of terminals nodes considered by search to depth **d** using α–β pruning is approximately equal to 2* number of nodes at depth d/2 without α–β pruning.
- So, doubling of depth by some search procedure is a significant gain.

- Further, the idea behind α–β pruning procedure can be extended by cutting-off additional paths that appear to be slight improvements over paths already beeen explored.

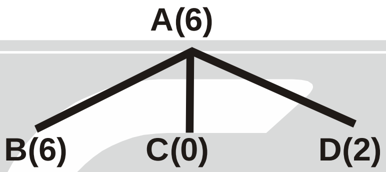


- We see that 3.2 is slightly better than 3, we may even terminate one exploration of C further
- Terminating exploration of Sub-tree that offers little possibility for improvement over known paths is called futility cut off.

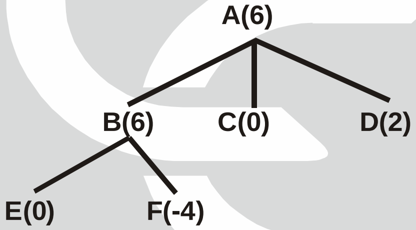
ADDITIONAL REFINEMENTS

- In addition to α–β pruning, there are variety of other modifications to MINMAX procedure, which can improve its performance. One of the factors is that when to stop going deeper in the search tree.

Waiting for Quiescence:

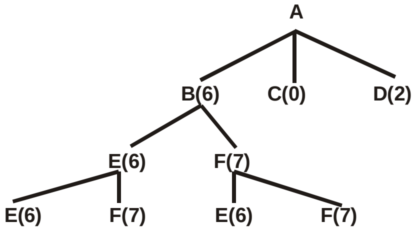


- Suppose node B is expanded one more level and the result is as



- Our estimate of worth of B has changed, this may happen if opponent has significantly improved
- If we stop exploring the tree at this level and assign –4 to B and therefore decide that B is not a good move.
- To make sure that such short term measures don't influence our choice of move, we should continue the search until no such drastic change occurs from one level to the next or till the condition is stable. This is called waiting for **quiescence** (Go deeper till the condition is stable before deciding the move.)

- Node B again looks like a reasonable move



PRACTICE SET

1. What is state space?
(a) Your Definition to a problem
(b) Problem you design
(c) Representing your problem with variable and parameter
(d) A space where you know the solution

2. A problem in a search space is defined by,
A. Initial state B. Goal test C. Intermediate states
(a) A only (b) B only (c) A and B (d) A, B and C

3. The Set of actions for a problem in a state space is formulated by a ___________.
(a) Initial state
(b) Path Cost
(c) Successor function, which takes current action and returns next immediate state
(d) None of the above

4. The major component(s) for measuring the performance of problem solving
A. Completeness B. Optimality C. Time and Space complexity D. Correctness
(a) C only (b) B and C only (c) B, C and D only (d) A, B, C and D

5. Which search method takes less memory?
(a) Depth-First Search (b) Breadth-First search
(c) Linear Search (d) Optimal search

6. Which is the best way to go for Game playing problem?
(a) Linear approach (b) Heuristic approach
(c) Random approach (d) Stratified approach

7. Which search is implemented with an empty first-in-first-out queue?
(a) Depth-first search (b) Breadth-first search
(c) Bidirectional search (d) None of the mentioned

8. When is breadth-first search is optimal?
(a) When all step costs are equal (b) When all step costs are unequal
(c) Both a & c (d) When there is less number of nodes

9. What is the space complexity of Depth-first search?
(a) O(b) (b) O(bl) (c) O(m) (d) O(bm)

10. Which search algorithm imposes a fixed depth limit on nodes?
(a) Depth-limited search (b) Depth-first search
(c) Iterative deepening search (d) Bidirectional search

11. Which of the following is/are Uninformed Search technique/techniques
A. Bidirectional Search B. Best First Search
C. Depth First Search D. Depth Limited Search
(a) A and C only (b) A, B and C only
(c) A, C and D only (d) A, B, C and D

12. Uniform-cost search expands the node n with the__________.
(a) Lowest path cost (b) Heuristic cost
(c) Highest path cost (d) Average path cost

13. Which is used to improve the performance of heuristic search?
(a) Quality of nodes (b) Quality of heuristic function
(c) Simple form of nodes (d) None of the mentioned

14. A heuristic is a way of trying
(a) To discover something or an idea embedded in a program
(b) To search and measure how far a node in a search tree seems to be from a goal
(c) To compare two nodes in a search tree to see if one is better than another
(d) All of the above

15. Best-First search is a type of informed search, which uses ________________ to choose the best next node for expansion.
(a) Evaluation function returning lowest evaluation
(b) Evaluation function returning highest evaluation
(c) Both a & b can be used
(d) None of them is applicable

16. Heuristic function h(n) is,
(a) Lowest path cost
(b) Cheapest path from root to goal node
(c) Estimated cost of cheapest path from root to goal node
(d) Average path cost

17. In A* approach evaluation function is
(a) Heuristic function
(b) Path cost from start node to current node
(c) Path cost from start node to current node + Heuristic cost
(d) Average of Path cost from start node to current node and Heuristic cost

18. Though local search algorithms are not systematic, key advantages would include
A. Less memory B. More time
C. Finds a solution in large infinite space D. No optimum solution
(a) A and B (b) A and C (c) A, B and C (d)All of the above

19. Hill-Climbing algorithm terminates when,
(a) Stopping criterion met (b) Global Min/Max is achieved
(c) Some neighbor has higher values (d) Local Min/Max is achieved

20. Which method is effective for escaping from local minima?
(a) Updating heuristic estimate (b) Reducing heuristic estimate
(c) Eliminating heuristic estimate (d) None of the mentioned

21. ________________ are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations.
(a) Constraints Satisfaction Problems (b) Uninformed Search Problems
(c) Local Search Problems (d) Simulated Annealing

22. Consider a problem of preparing a schedule for a class of student. This problem is a type of
(a) Search Problem
(b) Backtrack Problem
(c) Constraint Satisfaction Problem
(d) Planning Problem

23. Solving a constraint satisfaction problem on a finite domain is an/a ___________ problem with respect to the domain size.
(a) P complete (b) NP complete (c) NP hard (d) Domain dependent

24. Which value is assigned to alpha and beta in the alpha-beta pruning?
(a) Alpha = max (b) Beta = min (c) Beta = max (d) Both a & b

25. Where does the values of alpha-beta search get updated?
(a) Along the path of search (b) Initial state itself
(c) At the end (d) None of the mentioned

ANSWER KEY

1. (c) **2. (c)** **3. (c)** **4. (d)** **5. (a)** **6. (b)** **7. (b)**

8. (a) **9. (d)** **10. (a)** **11. (c)** **12. (a)** **13. (b)** **14. (d)**

15. (a) **16. (c)** **17. (c)** **18. (b)** **19. (d)** **20. (a)** **21. (a)**

22. (c) **23. (b)** **24. (d)** **25. (a)**

CHAPTER

3

Structured Knowledge Representation

Proposional Logic:

- Logic is a study of methods and principles used to distinguish correct from incorrect reasoning.
- Formally it deals with the notion of truth in an abstract sense and is concerned with the principles of valid inferencing.
- A proposition in Logic is a declarative statements such as "Jack is a male", "Jack loves Mary" etc. Which are either true or false (not both) in a given context.
- For example, if we are given a set of propositions such as "It is hot today" and "If it is hot it will rain", then we can infer that "It will rain today".

Compound Proposition:

A proposition that is a combination of two or more simple proposition is called as compound proposition and in order to have a compound proposition. We use different connectives to connect them.

(1) Negation $(\neg)$**:**

$\neg P$ OR $\sim P$

P	$\neg$P
T	F
F	T

(2) Disjunction $(V/+)$

P	q	$P \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

(3) Conjunction $(\wedge / \cdot)$:

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

(4) Implication:

$$\underset{\substack{\text{premise OR}\\ \text{hypothesis}}}{P} \longrightarrow \underset{\substack{\text{conclusion or}\\ \text{anticedent}}}{q}$$

$$\boxed{p \rightarrow q \equiv \neg p \vee q}$$

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

(5) Bi-implication | Bi-conditional $(\leftrightarrow)$:

$$\boxed{p \leftrightarrow q}$$

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

$$\boxed{p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)}$$

Derived connectives:

(1) NAND$(\uparrow)$ $\neg(p \wedge q) \equiv \neg p \vee \neg q$ (De-morgan's law)

p	q	$p \uparrow q$
T	T	F
T	F	T
F	T	T
F	F	T

(2) NOR $(\downarrow)$: $\neg(p \vee q) \equiv (\neg p \wedge \neg q)$

p	q	$p \downarrow q$
T	T	F
T	F	F
F	T	F
F	F	T

(3) XOR $\rightarrow p \oplus q$ **:**

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

(4) X-NOR $(p \odot q)$

p	q	$p \odot q$
T	T	T
T	F	F
F	T	F
F	F	T

Implication : $\boxed{p \rightarrow q}$

Converse : $q \rightarrow p$

Inverse : $\neg p \rightarrow \neg q$

Contrapositive : $\neg q \rightarrow \neg p$

$p \rightarrow q$ is equivalent to the contrapositive.

Two proposition P and Q with propositinoal variable p, q, r .... are said to be equivalent if they have same truth values.

$$\boxed{p \rightarrow q \equiv \neg p \vee q}$$

p	q	$\neg p$	$p \rightarrow q$	$\neg p \vee q$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

$$\boxed{q \rightarrow p \equiv \neg p \rightarrow \neg q}$$

$p \rightarrow q$ $\boxed{\neg q \rightarrow \neg p}$

Problem: $p \to q$ is equivalent to which of the following proposition

(a) $\neg p \vee q$ (b) $\neg q \to \neg p$

(c) $\neg(p \wedge \neg q)$ (d) all of these

Soln.

p	q	$\neg p$	$\neg q$	$\neg p \vee q$	$\neg q \to \neg p$	$(p \wedge \neg q)$	$\neg(p \wedge \neg q)$
T	T	F	F	T	T	F	T
T	F	F	T	F	F	T	F
F	T	T	F	T	T	F	T
F	F	T	T	T	T	F	T

Correct option is (d).

Problem:

p	q	$(p \vee q) \to (p \wedge q)$	$p \leftrightarrow q$
T	T	T	T
T	F	F	F
F	T	F	F
F	F	T	T

$$\boxed{(p \vee q) \to (p \wedge q) \equiv p \leftrightarrow q}$$

$\boxed{p \equiv p \to p \wedge q}$ → False

p	q	$p \wedge q$	$p \to p \wedge q$
T	T	T	T
T	F	F	F
F	T	F	T
F	F	F	T

Tautology:

A proposition /compound proposition known to be a tautology if it produces true corresponding to all proposition literal.

e.g. $\boxed{p \vee \neg p}$

p	$\neg p$	$p \vee \neg p$
T	F	T
F	T	T

e.g. $\boxed{p \wedge \neg p}$

p	$\neg p$	$p \wedge \neg p$
T	F	F
F	T	F

Not a tautology

e.g. $((p \to q) \wedge q) \to q$

p	q	$p \to q$	$(p \to q) \wedge p$	$((p \to q) \wedge q) \to q$
T	T	T	T	T
T	F	F	F	T
F	T	T	T	T
F	F	T	F	T

is a tautology

e.g. $\boxed{(p \wedge q) \to q}$

$$\equiv\sim (p \wedge q) \vee q \equiv\sim (p \vee \sim q) \vee q \equiv\sim pV(\sim q \vee q) \equiv\sim pVT \equiv T$$

Contradiction: All the conditions are false

Contengency: For same value of proposition variable it is true and for some value of propositional variable it false.

OR

If a proposition is neither a tautology nor a contradiction.

- $p \wedge \neg p \equiv F(\text{contradition})$
- $p \to q \to$ contengency

$$\neg((p \to q) \wedge (pq \to r) \to p \to r)$$

p	q	r	$p \to q$	$q \to r$	$(p \to q) \wedge (q \to r)$	$p \to r$		
T	T	T	T	T	T	T	T	F
T	T	F	T	F	F	F	F	F
T	F	T	F	F	F	T	T	F
T	F	F	F	T	F	F	T	F
F	T	T	T	T	T	T	T	F
F	T	F	F	F	F	T	T	F
F	F	T	T	T	T	T	T	F
F	F	F	T	F	F	T	T	F

Satisfiable:

If atleast one value of the proposition is true for some value of the proposition variable.

Tautology:

Satisfeable but vice versa is not true

Satisfiable $\neq$ tautology (not always equal)

$\boxed{\text{Contingency} = \text{satisfiable}}$

$\boxed{\text{unsatisfiable} \sim \text{contradiction}}$ means all false.

(1) Negation $(\neg)$:

$$\neg(\neg P) = P$$

(2) Negation of conjunction:

$$\boxed{\neg(p\wedge q)\equiv\neg p\vee\neg q}$$

(3) Negation of disjunction:

$$\boxed{\neg(p\vee q)\equiv\neg p\vee\neg q}$$

(4) Negation of implication:

$$\boxed{\neg(p\rightarrow q)\equiv\neg(\neg p\vee q)\equiv p\wedge\neg q}$$

(5) Negation of bidirectional:

$$\neg(p\leftrightarrow q)\equiv\neg(p\rightarrow q)\wedge(q\rightarrow p)$$

$$\equiv\neg(p\rightarrow q)\vee(q\rightarrow p)\equiv\neg(\neg p\vee q)\vee(\neg q\vee p)$$

$$\equiv(p\wedge\neg q)\vee(q\wedge\neg p)$$

$$\equiv((p\wedge\neg q)\vee q)\wedge((p\wedge\neg q)\vee\neg p)$$

$$\equiv((p\vee q)\wedge(\neg q\vee q))\wedge(p\vee\neg p)\wedge(\neg q\vee\neg p)=(p\vee q)\wedge(\neg q\vee\neg p)$$

$$\equiv\boxed{(p+q)(\neg q+\neg p)}$$ it is XOR.

Dual of a proposition:

(1) Conjunction ↔ disjunction

(2) $T\leftrightarrow F$

$T\vee P=T$ dual of a proposition is

$F\wedge P=F$

$\boxed{\neg(p\wedge q)=\neg p\vee\neg q}$ → De-morgan's rule.

Duality is

$$\boxed{\neg(p\vee q)=\neg p\wedge\neg q}$$

$\boxed{(p^{D})=p}$ dual of dual p = p

Identies:

(1) Idempotent law:

$$p\wedge p=p$$

$$p\vee p=p$$

(2) Commutation law:

$$p\vee q=q\vee p$$

$$p\wedge q=q\wedge p$$

(3) Associative law:

$$p\wedge(q\wedge r)=(p\wedge q)\wedge r$$

$$p\vee(q\vee r)=(p\vee q)\vee r$$

(4) Distributive law:

$$(p\wedge q)\vee r=(p\vee r)\wedge(q\vee r)$$

$$(p\vee q)\wedge r=(p\wedge r)\vee(q\wedge r)$$

$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$$

$$p \wedge (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$

(5) Identity law:

$P \vee T = T$

$P \vee F = P$

$P \wedge T = P$

$P \wedge F = F$

(6) Involution law:

$\sim (\sim P) = P$

(7) Complement law:

$P \wedge \neg P = F$

$P \vee \neg P = T$

$\sim T = F$

$\sim F = T$

(8) Absorption law: (Here, q is absorbed by P. So, called absorption law)

$p \wedge (p \vee q) = P$

$p \vee (p \wedge q) = P$

Proof: $p \wedge (p \vee q) = P$

$= (p \wedge F) \vee (p \wedge q)$

$= p \vee (F \wedge q) = P \vee F = P = RHS$

Proof: $p \vee (p \wedge q) = P$

$(p \wedge T) \vee (p \wedge q)$

$p \wedge (T \vee q) = p \wedge T = P$

NORMAL FORM

1. Disjunctive Normal form (DNF):

The disjunction of clause and the clause are the conjunction of the all proposition variable in true form or its negated form. It is also known as Disjunction of Conjunction.

$\Rightarrow$ $p \wedge q \rightarrow$ single clause

$\Rightarrow$ $p \vee q = p + q$

$p \rightarrow q = \neg p \vee q$

(1) $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

$$\equiv (\neg p \vee q) \wedge (\neg q \vee p)$$

$$\equiv (((\neg p \vee q) \wedge \neg q) \vee (\neg p \vee q) \wedge p)$$

$$\equiv (\neg p \wedge \vee q) \vee (q \wedge \neg q) \vee (\neg p \wedge p) \vee (q \wedge p)$$

$$\equiv ((\neg p \wedge \neg q) \vee F) \vee (F \vee (q \vee p))$$

$$\equiv (\neg p \wedge \neg q) \vee (p \wedge q)$$

(2) $(p \rightarrow q) \rightarrow r$

$(\neg p \vee q) \rightarrow r$

$$\neg(\neg p \vee q) \vee r$$

$$(p \wedge q) \vee r$$

Principle Disjunctive normal form (PDNF):

It is also called canonical DNF i.e. CDNF.

(1) $\underset{C_1}{p}\vee \underset{C_2}{q}$: This is not PDNF because it does not contain all variable. C$_1$ contain only p but not q and C$_2$ contain only q but not p.

$$p \equiv p \wedge T \equiv p \wedge (q \vee \neg q) \text{ (distributive law)}$$

$$\equiv (p \wedge q) \vee (p \wedge \neg q)$$

$$q \equiv T \wedge q \equiv (p \vee \neg p) \wedge q \equiv (p \wedge q) \vee (\neg p \wedge q)$$

$$\boxed{p \vee q = (p \vee q) \vee (p \wedge \neg q) \vee (\neg p \wedge q)}$$

$p \wedge q$: It is single clause, it is in both PDNF and DNF

$$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p) \equiv (\neg p \vee \wedge q) \vee (p \wedge q)$$

$$p \to q \equiv \neg \underset{C_1}{p} \vee \underset{C_2}{q}$$

$$\neg p : (\neg p \vee q) \vee (\neg p \wedge \neg q)$$

$$q : (p \vee q) \vee (\neg p \wedge q)$$

$$\neg p \vee q : (\neg \underset{C_1}{p} \vee q) \vee (\neg p \underset{C_2}{\wedge} \neg q) \vee (\underset{C_2}{p \wedge q})$$

If the two clauses are same then write it one time because of principle of idempotent.

$$p \vee p \vee p \text{..........} \vee p = p$$

$$p \wedge p \wedge p \text{..........} \wedge p = p$$

e.g. $(p \to q) \to r$

$$(p \underset{C_1}{\wedge} \neg q) \vee \underset{C_2}{r} \quad DNF$$

For, PDNF:

$$= (p \wedge \neg q \wedge r) \vee (p \wedge \neg q \wedge \neg r) \vee (p \wedge \neg q \wedge r)$$

$$p \wedge r \equiv (p \wedge q \wedge r) \vee (p \vee \neg q \wedge r)$$

$$(\neg p \wedge r) \equiv (\neg p \wedge q \wedge r) \vee (\neg p \wedge \neg q \wedge r)$$

Second method by truth table

$p \vee q$:

p	q	$p \vee q$	$p \leftrightarrow q$
T	T	T	T
T	F	T	F
F	T	T	F
F	F	F	T

$$p \vee q : (p \wedge q) \vee (p \wedge \neg q) \vee (\neg p \wedge \neg q)$$

$$p \leftrightarrow q : (p \wedge q) \wedge (\neg p \wedge \neg q)$$

2. Conjuctive normal form (CNF):

It is the conjunction of clauses and a clause is the disjunction of proposition variable

$p \vee q \rightarrow$ in CNF form $\quad (p+q) \rightarrow$ single clause

$p \wedge q \rightarrow$ in CNF form $\quad pq \rightarrow$ single clause

(1) $p \rightarrow q$: convert to CNF

$\neg p \vee q$ (CNF)

(2) $p \leftrightarrow q$

$$(p \rightarrow q) \wedge (q \rightarrow p) = (\neg p \vee q) \wedge (\neg q \vee p)$$

(3) $(p \wedge q) \rightarrow (r \wedge s)$

$$\equiv (\neg(p \wedge q)) \vee (r \wedge s)$$

$$\equiv (\neg p \vee \neg q) \vee (r \wedge s)$$

$$\equiv \underset{C_1}{(\neg p \vee \neg q \vee r)} \wedge \underset{C_2}{(\neg p \vee \neg q \vee s)}$$

Principle/canonical conjunction normal form:

It is the conjunction of clause and a clause is the disjunction of all proposition variable in their true or negated form.

$(p \wedge q) \rightarrow (r \wedge s)$ to convert PCNF

$$\equiv (\neg p \vee \neg q \vee r) \wedge (\neg p \vee q \vee s)$$

Rules of inferences:

(1) Modes Pones:

$$\begin{array}{l} p \rightarrow q \\ p \end{array} \quad \Rightarrow \quad \begin{array}{l} \neg p \wedge q \\ \underline{\ p \qquad} \\ q \end{array}$$

Conclusion, q is true.

If $p \rightarrow q$ is true and p is also true then q is also true.

We write it : $(p \vee (p \rightarrow q)) \rightarrow q$ (tautology)

(2) Modus Tolens (Rules opf detachment):

$$\begin{array}{l} p \rightarrow q \\ \neg q \end{array} \quad \Rightarrow \quad \begin{array}{l} \neg q \rightarrow \neg p \\ \underline{\ \neg q \qquad\quad} \\ \neg p \end{array}$$

If $\neg q \rightarrow \neg p$ is true and $\neg q$ is also true the $\neg p$ is true.

(3) Hypothetical syllogism:

$$\begin{array}{l} p \rightarrow q \\ \underline{q \rightarrow r} \\ p \rightarrow r \end{array}$$

(4) Disjunctive syllogism:

$$\begin{array}{l} p \vee q \\ \underline{\neg p \quad} \\ \ q \end{array}$$

(5) Simplification:

$$\frac{p \wedge q}{\therefore p} \text{ and } \frac{p \wedge q}{q}$$

(6) Conjunction:

$$\begin{array}{c} p \\ \underline{\quad q \quad} \\ p \wedge q \end{array}$$

(7) Addition:

$$\frac{p}{p \vee q}$$

(8) Constructive Dilmma:

$$(p \to q) \wedge (r \to s)$$

$$\frac{p \vee r}{p \vee s}$$

$$\begin{array}{lcl} p \to q & & r \to s \\ p & \vee & r \\ q & \vee & s \end{array}$$

(9) Destructive Dilmma:

$$(p \to q) \wedge (r \to s)$$

$$\frac{\neg p \vee \neg s}{\neg p \vee \neg r}$$

From MT,

$$\begin{array}{lcl} p \to q & & r \to s \\ \dfrac{\neg q}{\neg p} & or & \dfrac{\neg s}{\neg r} \end{array}$$

(10) Absorption law:

$$\frac{p \to q}{p \to p \wedge q}$$

Valid argument:

An argument $p_1, p_2, p_3, \ldots\ldots\ldots\ldots p_n$ is called valid, if we are able to derive C from the set of permises $p_1, p_2, p_3, \ldots\ldots\ldots\ldots p_n$ by applying the rule of inference.

Example:

(1) $p,\ p \to q \vdash q$ valid argument

(2) $p,\ p \to q,\ q \wedge r,\ r \to s,\ \neg s \vee t \vdash t$ conclusion

(3) $p \to q, r \to q, s \to (p \vee r), s \vdash q$

(4) $p \to \sim q, \sim \dfrac{r \vee q}{q}, \dfrac{r}{r} \vdash \sim p$ valid argument

$= q \to \sim p$

$$\frac{q}{\therefore\ \sim p} \text{ from M.P.}$$

(5) $p \to \neg q, \neg r \to q, p \vdash r$

(6) $p \to \neg q, \dfrac{r \to q}{q}, \dfrac{r}{r} \vdash \neg p$

(7) $(p \wedge q) \to r, r \to q \vdash (p \wedge q) \to (q \wedge r)$

$$\frac{r \to q}{r \to (q \wedge r)} \text{ by absorption law}$$

$(p \wedge q) \to (q \wedge r)$ by hypothetical rule

This is true.

Resolution:

We can apply the resolution in two clause (Disjunction of variables in true or negated form), if they have complementary variable.

Horn Clause: A clause (Disjunction of literal) which contain Atmost one positive literal proposational variable.

(i) $\neg A \vee \neg B \vee \neg C$, this is horn clause i.e. 0 positive

(ii) $\neg A \vee B \vee \neg C$, this also horn clause i.e. literal

(iii) $\neg A \vee B \vee C \vee \neg D$, this is not horn caluse because there is two positive literal.

A horn clause which has no negative literals is called fact. The fact will always have one literal. When we resolve two clause then we get a resolvent and the complementary literals are ommitted.

e.g. (1) $$\underbrace{\underset{C_1}{A \vee B} \qquad\qquad \underset{C_2}{\neg A \vee C}}_{B \vee C \to \text{Resolvent}}$$

(2) $$\underbrace{(A \vee B) \wedge (\neg A \vee C)}_{B \wedge C}$$

(3) $$\underbrace{\underset{C_1}{A \vee \neg B \vee \neg C} \qquad\qquad \underset{C_2}{B \vee C \vee \neg D}}_{B \vee \neg D}$$

Deduction Theorem:

To prove a formula $\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n \to \beta$, it is sufficient to prove a theorem **from α_1, α_2,...α_n infer β.**

Example : Prove the following theorem:

infer ((Q→P) ∧ (Q→R)) → (Q→(P∧R))

Solution : In order to prove **infer** ((Q→P)∧(Q→R))→ (Q→(P∧R)), we prove a theorem **from {Q→ P, Q →R{ infer** Q→ (P∧R). further for proving Q→(P∧R), we have to prove a sub theorem from Q infer P∧R **{Theorem} from Q→P, Q→R infer Q→(P∧R)**

{Premise 1}	$Q \rightarrow P$	(1)
{Premise 2}	$Q \rightarrow R$	(2)
{sub theorem}	from Q infer $P \wedge R$	(3)
{premise}	Q	(3.1)
{E-$\rightarrow$, (1,3.1)}	P	(3.2)
{E-$\rightarrow$, (2,3.1)}	R	(3.3)
{I-$\wedge$, (3.2, 3.3)}	$P \wedge R$	(3.4)

Limitation of Propositional Logic:

- The facts: "peter is a man", "paul is a man", "John is a man" can be symbolized by P, Q and R respectively, in PL. But we would not be able to draw any conclusions about similarities P. Q and R.
- It would be much better to represent these facts as MAN(peter), MAN (paul) and MAN(john).
- Further, we are even in more difficulty, if we try to represent sentences like "All men are mortal" in Propositional Logic.
- In Predicate Logic, such sentences can be easily represented and the limitations of propositional logic are removed to some extent.

Predicate Logic:

- The predicate logic is logical extension of propositional logic. The first order predicate logic is one where the quantification is over simple variables.

- It has three more logical notions as compared to propositional calculus.
    - Terms, Predicates and
    - Quantifiers (universal or existential quantifiers i.e. "for all" and "there exists")

A term is a constant (single individual or concept i.e. 5, john, etc.), a variable that stands for different individuals and n-place function $f(t_1,...,t_n)$ where $t_1,...,t_n$ are terms.

- A *function* is a mapping that maps n terms to a term.
- A **predicate** is a relation that maps n terms to a truth value true (T) or false (F).

Examples:

- A statement *x* is *greater than y* is represented in predicate calculus as Greater (x,y). It is defined as follows:

GREATER (x,y)	=	T, if x>y
	=	F, otherwise

- A statement *x* loves *y* is represented as LOVE (x,y) which maps it to true or false when x and y get instantiated to actual values.
- A statement *john's father loves john* is represented as LOVE (father(john), john). Here, *father* is a function that maps *john* to his father.
- The predicate names GREAT and LOVE take two terms and map *T* or *F* depending upon the values of their terms.

Quantifiers: Variables are used in conjunction with quantifiers. Ther are two types of quantifiers viz.., "there exist" ($\exists$) and "for all" ($\forall$) known as Existential and Universal quantifiers, respectively.

Example : Translate the text *"Every man is mortal. John is a man. Therefore, John is mortal"* into predicate Calculus formula.

Soln. Let MAN (x), MORTAL (x) represent 'x is a man' and 'x is mortal' respectively.

- Every man is mortal : $(\forall x)$ (MAN(x) $\rightarrow$ MORTAL (x))
- John is a man : MAN(john)
- John is mortal : MORTAL (john)

The whole text can be represented by the following formula.

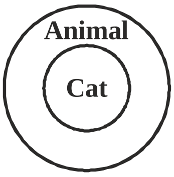
$(\forall x)$ ((MAN(x) $\rightarrow$ MORTAL (x)) $\wedge$ MAN (john)) $\rightarrow$ MORTAL(john)

Converting into Predicate logic:

e.g. all cats are animal

cat (x) : x is a cat

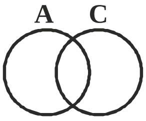
Animal (x) : x is a animal



- Turbo is a cat : cat (turbo)
- Cow is a animal : animal (cow)
- All cats are animal $\forall x\left[\text{cat}(x) \rightarrow \text{animal}(x)\right]$

$\boxed{\text{with } \forall \text{ we always use implication} (\rightarrow)}$

Some cat are animal.



$\exists x\left(\text{cat}(x) \wedge \text{animal}(x)\right)$

$\boxed{\text{for } \exists \text{ quantifier we use } \wedge \text{ always}}$

Some cats are not animal

$\exists x\left(\text{cat}(x) \wedge \neg\,\text{animal}(x)\right)$

Brothers are siblings:

Brother (x, y) $\equiv$ x is a brother of y

Sibling (x, y) $\equiv$ x is a sibling of y

$\forall x \forall y\left[\text{Brother }(x, y) \rightarrow \text{sibling }(x, y)\right]$

Sibling is commutative

$\forall x \forall y\left[\text{sibling }(x, y) \rightarrow \text{sibling }(x, y)\right]$

Everyone loves his/her mother

Mother (x, y): x is mother of y

Love (x, y): x loves y

Example:

(1) $P(x, y)$: x divides y

UOD = integer number or UOD = $\langle 5,10,15,20 \rangle$

- $\forall x \forall y\ P(x, y) = F$
- $\forall y \forall x\ P(x, y) = F$
- $\exists x \forall y\ P(x, y) = T$

Some int divides all int.

$\forall x \exists y\ P(x, y) \quad T$

(2) $P(x, y): x + y < 0$

$\forall x \forall y \quad P(x, y) \quad F$

$\forall x \exists y \quad P(x, y) \quad T$

$\forall x \exists ! y \quad P(x, y) \quad F$

$\exists ! x\ \forall y\ P(x, y) \quad F$

(3) $P(x,y,z): x+y=z$

$\forall x \forall y \forall z \ \ P(x,y,z): F$

$\forall x \forall y \exists z \qquad P(x,y,z): T$

$\forall x \forall y \exists ! z \ \ P(x,y,z): F$

(4) *x* is an even number

$\exists y(y+y=x)$ OR $\exists ! y(y+y=x)$

(5) *x* is an odd number

$\exists y(y+y+1=x)$

(6) *x* divides *y*

$\exists z(x \cdot z=y)$

Substitution:

Putting a value | variable | function in place of another variable.

Unification:

It is process which takes two sentences P on Q with same predicate and apply some substitution to make both P and Q equal.

know(Ram, y) know (x, shyam)

know (x, y) : x knows y

(x|Ram, y|Shyam) and replace x with Ram & y with Shyam to satisfy unification property.

- A formula α is said to be **consistent** (satisfiable) if and only if there exists an interpretation I such that I[α] = T. Alternatively we say that I is a *model* of α or I satisfies α.
- A formula α is said to be **inconsistent** (unsatifiable) if and only if ∃ nointerpretation that satisfies α or there exists no model for α.
- A formula α is **valid** if and only if for every interpretation I, I[α] = T.
- A formula α is a **logical consequence** of a set of formulae {α_1, α_2, ..., α_n}if and only if for every interpretation I, if I[$a_1 \cup ... \cup a_n$] = T, then [α] = T.

Prenex Normal Form

- In FOL, there are infinite number of domains and consquently infinite number of interpretations of a formula.
- Therefore, unlike PL, it is not possible to verify a validity and inconsistency of a formula by evaluating it under all possible interpretations.
- We will discuss the formalism for verifying inconsistency and validity in FOL.
- In FOL, ther is also a third type of normal form called Prenex Normal Form.
- A closed formula α of FOL is said to be in **Prenex Normal Form** (PNF) if and only if α is represented as $(q_1x_1)(q_2x_2)...(q_nx_n)(M)$

where, q_k, $(1 \le k \le n)$ quant ($\forall$ or ∃) and M is a formula free from quantifiers.

- A list of quantifiers $[(q_1x_1)...(q_nx_n)]$ is called **prefix** and M is called the **matrix** of a formula α. Here, M is represented in CNF.

Transformation of Formula in PNF

- A formula can be easily transformed into PNF using various equivalence laws. We use the following conventions:

$\alpha[x]$	–	a formula α, which contains a variable x.
α	–	a formula without a variable x.
q	–	Quantifier ($\forall$ or $\exists$)

Equivalence Laws: There are following pairs of logically equivalent formulae called equivalence laws in addition to the equivalence laws given for PL.

1. $(q\ x)\ \alpha\ [x]\ V\ \beta \quad \cong \quad (q\ x)\ (\alpha\ [X] \vee \beta)$
2. $\alpha \vee (q\ x)\ \beta\ [x] \quad \cong \quad (q\ x)\ (\alpha \vee \beta\ [X])$
3. $(q\ x)\ \alpha\ [x] \wedge \beta \quad \cong \quad (q\ x)\ (\alpha[X] \wedge \beta)$
4. $\alpha \wedge (q\ x)\ \beta[x] \quad \cong \quad (q\ x)\ (\alpha \wedge \beta[X])$
5. $\sim((\forall\ x)\ \alpha\ [x]) \quad \cong \quad (\exists\ x)\ (\sim\alpha[X])$
6. $\sim((\exists x)\ \alpha\ [x]) \quad \cong \quad (\forall\ x)\ (\sim\alpha[X])$

Skolemisation:

The process of removing the existential quantifier from the given expression is called as Skolemisation.

$\exists x\ P(x)$

- $\forall x\ P(x, \text{ice cream}) \equiv P(a, \text{icecream})$

$\exists x\ P(x, \text{ice cream}) \equiv P(c, \text{ice cream})$ Here c is skolen constant

$\forall x \exists y\ P(x, y)$ Remove quantifer

$\forall x\ P(x, f(x))$ Here $f(x)$ is skolen function.

$\exists u\ \forall x\ \forall y\ \exists z\ \ P(x, y, z, u)$

$\equiv \forall x \forall y\ \exists z\ \ P(x, y, z, C_1)$

$\equiv \forall x\ \forall y\ \ P(x, y, f(x, y), C_1)$

$\equiv P(C_2, C_3, f(C_3, C_2) C_1)$ Here, C_i is skolen variable and $f(\ )$ is skolen function.

Convertion of PNF to its Standard Form

- Scan prefix from left to right, q_1 is the first existential quantifier then choose a new constant C $\in$ matrix. Replace all occurance of x_1 appearing in matrix by c and delete $(q_1 x_1)$ from the prefix to obtain a new prefix and matrix.
- If q_r is an existential quantifier and $q_1 \ldots q_{r-1}$ are universal quantifiers appearing before q_r, then choose a new (r-1) place function symbol 'f' $\notin$ Matrix. Replace all occurence of x_r in matrix by $f(x_1, \ldots, x_{r-1})$ and remove $(q_r\ x_r)$.
- Repeat the process till all existential quantifiers are removed from matrix.
- Any formula α in FOL can be transformed into its standard form.
- Matrix of standard formula is in CNF and prefix is free from existential quantifiers.
- Formula in standard form is expressed as
  $(\forall\ x_1) \ldots (\forall\ x_n)\ (C_1 \wedge \ldots C_m)$,
  where C_k $(1 \le k \le m)$ is formula in disjunctive normal form.
- Since all the variables in prefix are universally quantified, we omit prefix part from the standard form for the sake of convenience and write standard form as $(C_1 \wedge \ldots \wedge C_m)$.
- A *clause* is a closed formula of the form $(\forall\ x_1) \ldots\ (\forall\ x_n)\ (L_1 V \ldots V L_m)$, where each L_k is a literal and $X_1 .. X_n$ are variables occuring in $L_1, \ldots. L_m$.

- Since all the variables in a clause are universally quantified, we omit them and write clause as $(L_1 V..V L_m)$ for the sake of simplicity.
- The standard form of any formula is of the form $(C_1 \wedge .. \wedge C_m)$, where C_i $(1 \le i \le m)$ is a clause.
- We define S= $\{C_1, ... C_M\}$ to be a set of clauses that represents a standard form of a formula α.
- S is said to be *unsatisfiable* (inconsistent) if and only if there $\exists$ no interpretation that satifies all the clauses of S simultaneously.
- A formula α is unsatisfiable (inconsistent) if and only if its corresponding set S is unsatisfiable.
- S is said to be *satisfiable* (consistent) if each clause is consistent i.e., $\exists$ an interpretation that satifies all the clauses of S simultaneously.
- Alternatively, an interpretation I models S if and only if I models each clause of S.

Resolution in Predicate Logic

- The resolution principle basically checks whether empty clause is contained or derived from S.
- Resolution for the clauses containing no variables is very simple and is similar to propositional logic. It becomes complicated when clauses contain variables.
- In such case, two complementary litrals are resolved after proper substitutions so that both the literals have same arguments.

Example: Consider two clauses C_1 and C_2 as follows:

C_1 = P(x) V Q(x)
C_2 = ~P(f(x)) V R(x)

- Substitute ‘f(a)’ for ‘x’ in C_1 and ‘a’ for ‘x’ in C_2, where ‘a’ is a new constant from the domain, then

C_3 = P(f(a)) V Q (f(a))
C_4 = ~P(f(a)) V R (a)

- Resolvent C of C_3 and C_4 is **[Q(f(a)) V R(a)]**
- Here C_3 and C_4 do not have variables. They are called ground instances of C_1 and C_2.
- In general if we substitute ‘f(x)’ for ‘x’ in C_1, then

C_1 = P(f(x)) V Q(f(x))

- Resolvent C’ of $C’_1$ and C_2 is **[Q(f(x)) V R(x)]**
- We notice that C is an instance of C’.

Theorem: L is a logical consequence of S if $\{S \cup \sim L\} = \{C1, ... Cn, \sim L\}$ is unsatisfiable

- A deduction of an empty clause from a set of clauses is called a *resolution refutation* of S

Theorem: (Soundness and completeness of resolution);
There is a *resolution refutation* of S if and only if S is unsatisfiable (inconsistent).

Theorem: L is a logical consequence of S if and only if there is a resolution refutation of $S \cup \{\sim L\}$.

- We can summarize that in order to show L to be a logical consequence the set of formula $\{\alpha_1, ... \alpha_n\}$, use the following procedure.

Procedure:

- Obtain a set S of all the clauses by converting each α_k to its corresponding standard form and then to the clauses.
- Show that a set $S \cup \{\sim L\}$ is unsatisfiable i.e., the set $S \cup \{\sim L\}$ contains either empty clause or empty can be derived in finite steps using resoultion method.
- If so, then report *‘yes’* and conclude that L is a logical consquence of S and subsquently of formula α_1, ... α_n otherwise report *‘No’*.
- If the choice of clauses to resolve at each step is made in a systematic ways, then resolution algorithm will find a contradiction if one exists.
- There exist various strategies for making the right choice that can speed up the process considerably.

Useful Tips:

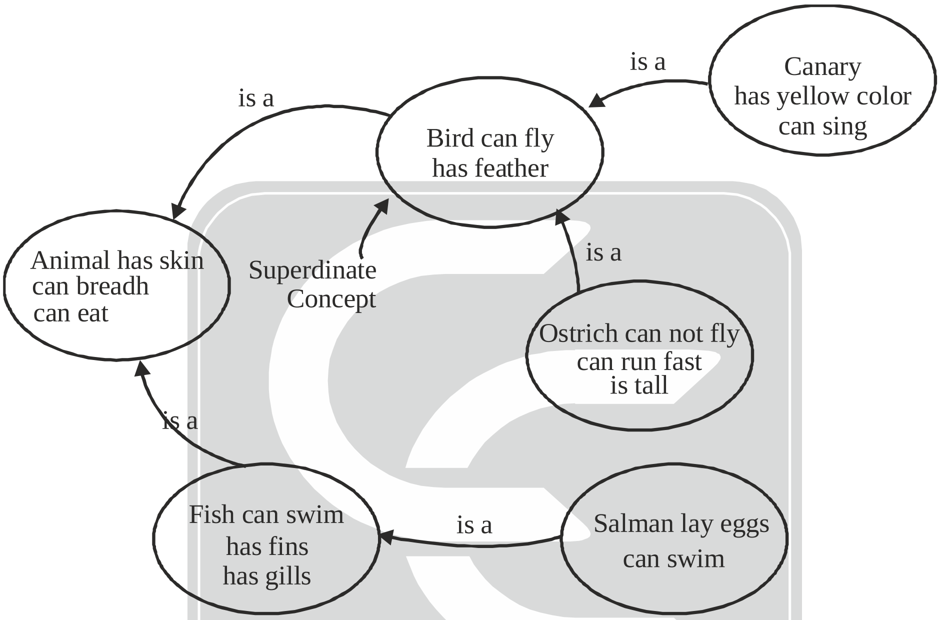
- Initially choose a clause from the negated goal clauses as one of the parents to be resolved (This corresponds to intution that the contradiction we are looking for must be because of the formula we are trying to prove).
- Choose a resolvent and some existing clause if both contin complementary literals.
- Whenever possible, resolve with the clauses with single literal. Such resolution generate new clauses with fewer literals than the larger of their parent clauses and thus probably algorithm terminates faster.

Semantic Network :

Ross Auillian has disigned a computational model in which the processes/concepts are represented in heirarchical structures.

Semantic = Syntatically + semantically

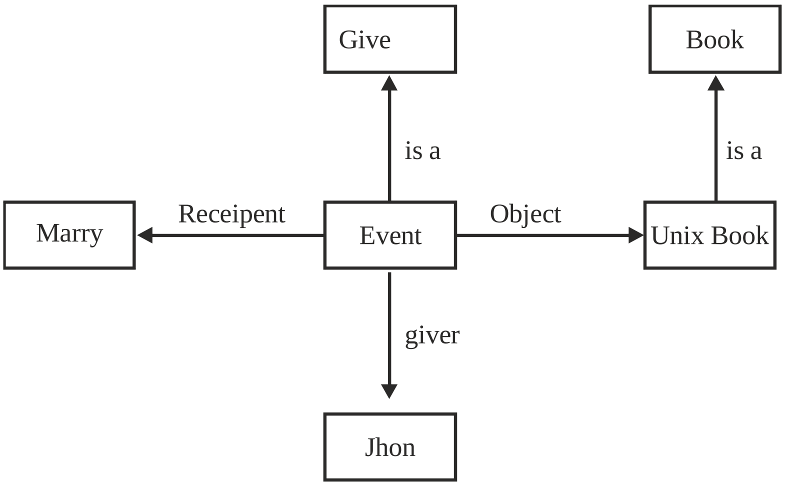
Correct grammar part of sentence is meaning for view.



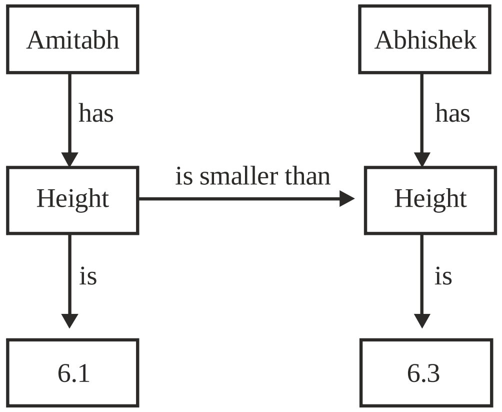
Subordinate inherit all the attributes of superordinate.

Representing Events by Semantic network:

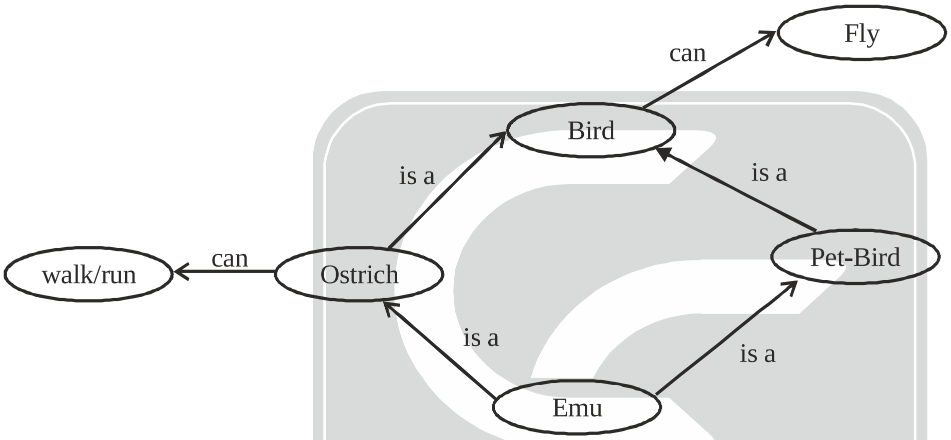
Jhon gave marry a book



Amitabh has smaller height than Abhishek



Multiple Inheritance:



"What is locomotion of EMu" = walk/runs which is more close to concept i.e locomotion.

PRACTICE SET

1. How many logical connectives are there in artificial intelligence?
   (a) 2 (b) 3 (c) 4 (d) 5

2. (A) Knowledge base (KB) is consists of set of statements.
   (B) Inference is deriving a new sentence from the KB.
   Choose the correct option.
   (a) A is true, B is true (b) A is false, B is false
   (c) A is true, B is false (d) A is false, B is true

3. Which is used to compute the truth of any sentence?
   (a) Semantics of propositional logic (b) Alpha-beta pruning
   (c) First-order logic (d) Both a & b

4. Which form is called as conjunction of disjunction of literals?
   (a) Conjunctive normal form (b) Disjunctive normal form
   (c) Normal form (d) All of the mentioned

5. Which is not Familiar Connectives in First Order Logic?
   (a) and (b) iff (c) or (d) not

6. Which are needed to compute the logical inference algorithm?
   (a) Logical equivalence (b) Validity
   (c) Satisfiability (d) All of the above

7. Translate the following statement into FOL.
   "For every a, if a is a philosopher, then a is a scholar"
   (a) $\forall$ a philosopher(a) $\rightarrow$ scholar(a) (b) $\exists$ a philosopher(a) $\rightarrow$ scholar(a)
   (c) Both A and B are true (d) None of the above

8. ___________ trees can be used to infer in Horn clause systems
   (a) Min/Max Tree (b) And/Or Trees
   (c) Minimum Spanning Trees (d) Binary Search Trees

9. What kind of clauses are available in Conjuctive Normal Form?
   (a) Disjunction of literals (b) Disjunction of variables
   (c) Conjuction of literals (d) Conjunction of variables

10. What is meant by factoring?
    (a) Removal of redundant variable (b) Removal of redundant literal
    (c) Addition of redundant literal (d) Addition of redundant variable

11. What is the logical translation of the following statement?
    "Noen of my friends are perfect."
    (a) $\exists x\left(F(x)\wedge\neg P(x)\right)$ (b) $\exists x\left(-F(x)\wedge P(x)\right)$
    (c) $\exists x\left(-F(x)\wedge\neg P(x)\right)$ (d) $\neg\exists x\left(F(x)\wedge P(x)\right)$

12. What is the correct translation of the following statement into mathematical logic? "Some real numbers are rational"
    (a) $\exists x\left(\text{real}(x)\vee\text{rational}(x)\right)$ (b) $\exists x\left(\text{real}(x)\rightarrow\text{rational}(x)\right)$
    (c) $\exists x\left(\text{real}(x)\wedge\text{rational}(x)\right)$ (d) $\exists x\left(\text{rational}(x)\rightarrow\text{real}(x)\right)$

13. Which one of the following options is CORRECT given three positive integers x, y and z, and a predicate?

$$P(x) = \neg(x=1) \wedge \forall y\left(\exists z\left(x = y * z\right) \Rightarrow \left(y = x\right) \vee \left(y = 1\right)\right)$$

(a) P(x) being true means that x is a prime number
(b) P(x) being true means that x is a number other than 1
(c) P(x) is always true irrespective of the value of x
(d) P(x) being true means that x has exactly two factors other than 1 and x

14. Suppose the predicate F(x, y, t) is used to represent the statement that person x can fool person y at time t. which one of the statements below expresses best the meaning of the formula $\forall x \exists y \exists t\left(\neg F\left(x, y, t\right)\right)$?

(a) Everyone can fool some person at some time
(b) No one can fool everyone all the time
(c) Everyone cannot fool some person all the time
(d) No one can fool some person at some time

15. Which of the following are equivalent.

(I) $\neg\forall x\left(P\left(x\right)\right)$ (II) $\neg\exists x\left(P\left(x\right)\right)$ (III) $\neg\exists x\left(\neg P\left(x\right)\right)$ (IV) $\exists x\left(\neg P\left(x\right)\right)$

(a) I and III (b) I and IV (c) II and III (d) II and IV

16. P and Q are two propositions. Which of the following logical expressions are equivalent?

(I) $P \vee \sim Q$ (II) $\sim\left(\sim P \wedge Q\right)$

(III) $\left(P \wedge Q\right) \vee \left(P \wedge \sim Q\right) \vee \left(\sim P \wedge \sim Q\right)$ (IV) $\left(P \wedge Q\right) \vee \left(P \wedge \sim Q\right) \vee \left(\sim P \wedge Q\right)$

(a) I and II (b) I, II and III (c) I, II and IV (d) I, II, III and IV

17. Let Graph(x) be a predicate which denotes that x is a graph. Let Connected(x) be a predicate which denotes that x is connected. Which of the following first order logic sentences DOES NOT represent the statement: "Not every graph is connected"?

(a) $\neg\forall x\left(\text{Graph}\left(x\right) \Rightarrow \text{Connected}\left(x\right)\right)$ (b) $\neg\exists x\left(\text{Graph}\left(x\right) \wedge \neg\text{Connected}\left(x\right)\right)$

(c) $\neg\forall x\left(\text{Graph}\left(x\right) \vee \text{Connected}\left(x\right)\right)$ (d) $\neg\forall x\left(\text{Graph}\left(x\right) \Rightarrow \neg\text{Connected}\left(x\right)\right)$

18. Which one of the following propositional logic formulas is TRUE when exactly two of p, q, and r are true?

(a) $\left(\left(P \leftrightarrow q\right) \wedge r\right) \vee \left(p \wedge q \wedge \sim r\right)$ (b) $\left(\sim\left(P \leftrightarrow q\right) \wedge r\right) \vee \left(p \wedge q \wedge \sim r\right)$

(c) $\left(\left(P \rightarrow q\right) \wedge r\right) \vee \left(p \wedge q \wedge \sim r\right)$ (d) $\left(\sim\left(P \leftrightarrow q\right) \wedge r\right) \wedge \left(p \wedge q \wedge \sim r\right)$

19. The CORRECT formula for the sentence, "not all rainy days are cold" is

(a) $\forall d\left(\text{Rainy}\left(d\right) \wedge \sim \text{Cold}\left(d\right)\right)$ (b) $\forall d\left(\sim\text{Rainy}\left(d\right) \rightarrow \text{Cold}\left(d\right)\right)$

(c) $\exists d\left(\sim\text{Rainy}\left(d\right) \rightarrow \text{Cold}\left(d\right)\right)$ (d) $\exists d\left(\text{Rainy}\left(d\right) \wedge \sim \text{Cold}\left(d\right)\right)$

20. Which one of the first order predicate calculus statements given below correctly express the following English statement?
Tigers and lions attack if they are hungry or threatened.

(a) $\forall x\left[\left(\text{tiger}\left(x\right) \wedge \text{lion}\left(x\right)\right) \rightarrow \left\{\left(\text{hungry}\left(x\right) \vee \text{threatened}\left(x\right)\right) \rightarrow \text{attacks}\left(x\right)\right\}\right]$

(b) $\forall x\left[\left(\text{tiger}\left(x\right) \vee \text{lion}\left(x\right)\right) \rightarrow \left\{\left(\text{hungry}\left(x\right) \wedge \text{threatened}\left(x\right)\right) \wedge \text{attacks}\left(x\right)\right\}\right]$

(c) $\forall x\left[\left(\text{tiger}\left(x\right) \vee \text{lion}\left(x\right)\right) \rightarrow \left\{\left(\text{attack}\left(x\right) \rightarrow \text{hungry}\left(x\right)\right) \vee \text{threatened}\left(x\right)\right\}\right]$

(d) $\forall x\left[\left(\text{tiger}\left(x\right) \vee \text{lion}\left(x\right)\right) \rightarrow \left\{\left(\text{hungry}\left(x\right) \vee \text{threatened}\left(x\right)\right) \rightarrow \text{attacks}\left(x\right)\right\}\right]$

21. **.** Consider the following propositional statements:

$P_1 : ((A \wedge B) \to C)) \equiv ((A \to C) \wedge (B \to C))$

$P_2 : ((A \wedge B) \to C)) \equiv ((A \to C) \vee (B \to C))$ Which one of the following is true?

(a) P_1 is a tautology, but not P_2 (b) P_2 is a tautology, but not P_1
(c) P_1 and P_2 are both tautologies (d) Both P_1 and P_2 are not tautologies

ANSWER KEY

1. (d)	**2. (a)**	**3. (a)**	**4. (a)**	**5. (d)**	**6. (d)**	**7. (a)**
8. (b)	**9. (a)**	**10. (b)**	**11. (d)**	**12. (c)**	**13. (a)**	**14. (b)**
15. (b)	**16. (b)**	**17. (d)**	**18. (b)**	**19. (d)**	**20. (d)**	**21. (d)**

CHAPTER 4

Logic Programming

Procedural versus declarative knowledge:

A declarative representation is one in which knowledge is specified, but the use to which that knowledge is to be put is not given. To use a declarative representation we must argument it with a program that specifies what is to done to the knowledge and how.

A procedural representation is one in which the control information that is necessary to use the knowledge is considered to be embedded in the knowledge itself. To use a procedural representation, we need to argument it with an interpreter that follows the instructions given in the knowledge.

Logic programming is a programming language paradigm in which logical assertions are viewed as programs. Prolog is one of the most popular logic programming given by Clocksin and Mellish, 1984 and Bratko, 1986. Logic programming is based on first order predicate logic. The program here is a collection of clauses.

A **clause** in logic programming is represented in a clausal notaion as $A_1,\ldots A_k \leftarrow B_1,\ldots B_t$, where A_j are positive literals and B_k are negative literals.

Conversion of a clause into Clausal Notation

- A clause in FOL is a closed formula of the form, $(\forall x_1)\ldots(\forall x_n)\,(L_1 V\ldots\ldots V L_m)$, where each L_k, $(1 \le k \le n)$ is a literal and x_k, $(1 \le k \le n)$ are all the variables occuring in $L_1\ldots\ldots L_{ms}$.
- Remove the prefix $(\forall x_1)\ldots(\forall x_n)$ for the sake of simplicity because all the variables appearing in a clause are universally quantified.
- Now a clause is represented as $L_1 V\ldots VL_m$, where L_k, $(1 \le k \le n)$ are literals free from quant.
- Separate positive and negative literals in the clause as follows:

  $(L_1 VL\backslash V\ldots VL_m)$

  $\cong \quad (A_1 V..VA_k\ V{\sim}B_1 V\ldots V{\sim}B_t)$

  where $m = k + t$, A_j, $(1 \le j \le k)$ are positive literals and B_j, $(1 \le j \le k)$ are negative literals.

  $\cong \quad (A_1 V..VA_k)\ V{\sim}(B_1 V\ldots V{\sim}B_t)$

  $\cong \quad (B_1 \wedge..\wedge B_t) \rightarrow (A_1 V\ldots VA_k)$

  $\{\text{since } P \rightarrow Q \cong {\sim} P\ V\ Q\}$
- Clausal notation is written in the form :

  $(A_1 V\ldots VA_k) \leftarrow (B1\wedge\ldots.\wedge Bt)$ or $A_1, \ldots A_k \leftarrow B_1\ldots B_t$
- Here A_j, $(1 \le j \le k)$ are positive literal and B_i, $(1 \le i \le t)$ are negative literals
- It must be noted that interpretation of $A \leftarrow B$ is same as $B \rightarrow A$
- In clausal notation, all variables are assumed to be universally quantified.
- B_i, $(1 \le i \le t)$ (negative literals) are called *antecedents* and A_j, $(1 \le j \le k)$ (positive literals) are called *consequents.*

- Commas in antacedent and consequent denote *conjunction* and *disjunction* respectively.
- Applying the result of FOL to the logic programs, a goal G with respect to a program P (finite set of clauses) is solved by showing that the set of clauses P ∪ {~G} is unsatisfiable or there is a resolution refutation of P ∪ {~G}.
- If so, then G is logical consequence of a program P. The basic constructs of logic programming are inherited from FOL.
- There are three basic statements: fact, rules and queries. these are special forms of clauses.

Example: consider the following logic program.
GRANDFATHER (x,y) ← FATHER (x, z), Parent (z, y)
PARENT (x,y) ← FATHER (x, y)
PARENT (x,y) ← MOTHER (x, y)
FATHER (abraham, robert) ←
FATHER (robert, mike) ←

• In FOL above program is represented as a set of clauses as

S={GRANDFATHER (X, Y) V ~FATHER (X ,Z) V~PARENT (z, y),
PARENT(X,Y) V ~ FATHER(x, y), PARENT (x, y) V~MOTHER (x,y),
FATHER(abraham, robert), FATHER (robert, mike)}

• Let us number the clauses of S as follows :

i. GRANDFATHER (x, y) V ~FATHER (x, z) v~PARENT (z, y)

ii. PARENT (x,y) V ~FATHER (x, y)

iii. PARENT (x,y) V ~MOTHER (x, y)

iv. FATHER (abraham, robert)

v. FATHER (robert, mike)

• *Simple Queries :*

(a) Ground Query
Query : "Is abraham a grandfather of mike?"
←GRANFATHER (abraham, mike)

• In FOL, ~ GRANDFATHER (abraham, mike) is negation of goal {GRANDFATHER (abraham, mike).

• Include {~goal} in the set S and show using resolution refutation that S ∪ {~goal} is unsatisfiable in order to conclude the goal.

• Let ~goal is numbered as (vi) in continuation of first five clauses of S listed above.
vi. ~GRANDFATHER (abraham, mike)

• Resolution tree is given as follows.

(b) Non ground queries
Query : "Does there exist x such that x is a father of robert ?" {who is father of robert?}
← FATHER (abraham, x).
Query : "Do there exist x and y such that x is a father of y?" {who is father of whom?}
← FATHER (x, y).

PROLOG:
A prolog program is described as a series of logical assertions each of which is a *Horn clause*. A Horn clause is a clause that has at most one positive literal.
It is a declarative language not procedural. A key difference between logic and prolog representation is that prolog interpreter has a fixed control strategy and so the assertions in the prolog program define a particular search path to an answer to any question. In contrast, the logical assertions define only the set of answer that they justify, they themselves say nothing about how to choose among those answers if there are more than one.

Prolog is all about:

(i) Fact **(ii) Rule** **(iii) Queries**

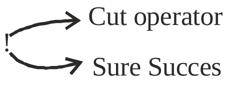
Woman(Sita).
Woman(Gita).
Woman(Mita).
Woman(Rita).
Woman(Sarita).

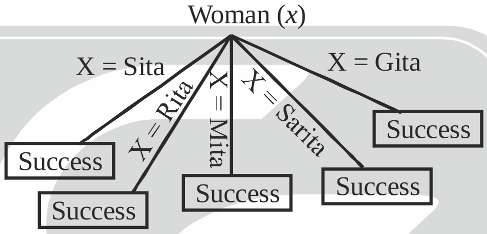
Full stop is the termination

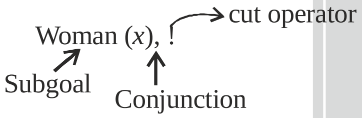
K.B. (Knowledge Base)
Woman (*x*) : *x* is a woman.

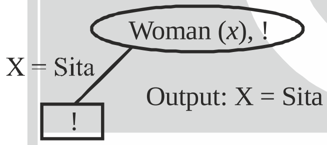
If we want to check only one time then query is written in this form.

? Woman (*x*)



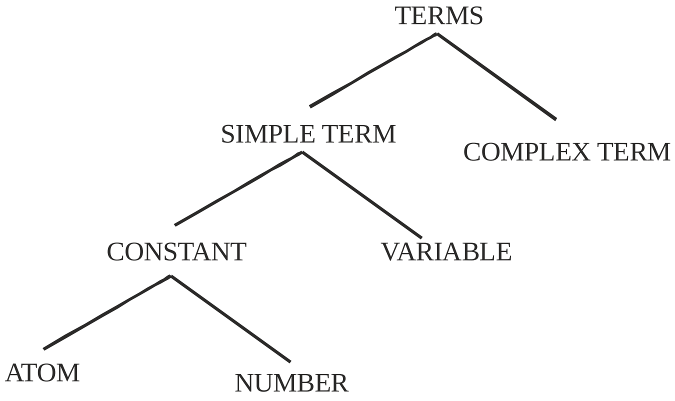






In cut operator there is no need of back tracking. If first time it fails (woman (*x*)) then it backtrack and check for second time.

"Prolog uses Backward Chaining"
Prolog is a combination of rules, facts and Queries i.e. called **Terms.**



ATOM : An atom is a sequence of capital letters, small case letters, digits and underscore, but it always starts with small case letters.

e.g. mita, sita, happy, sing-123-ABC

• Any sequence of character with in single quote.

- '12-BABA' 12-ABC not a Alam
- 'Shri-420$
- Special symbol

•, :, –, ;

(2) Number:

Integer number : 123, 10, 50, 60 etc.

Real number : 10.42, 123.45

(3) Variable: A variable is a sequence of capital case letters, small case letter, digits and underscore but it always starts with capital case letters or underscore.

Cat, –abc, –124bc

Cat (X)

Complex Term: function (Argument)

Number of argument = Arity of the function

$P(x, y) \rightarrow \text{arity} = 2$

$\boxed{\text{happy}(Jhon) \rightarrow \text{complex term}}$

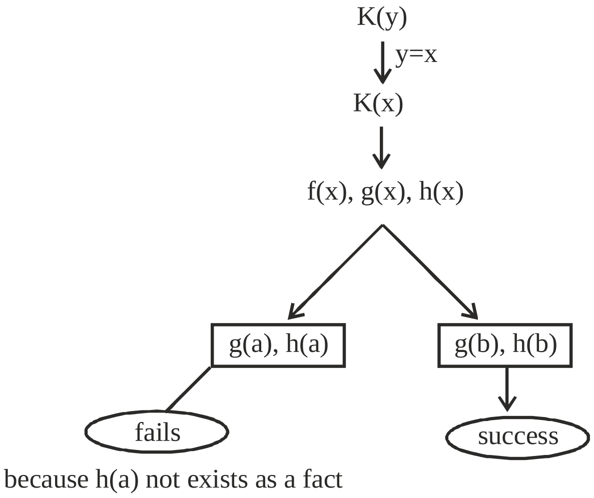
Back tracking:

e.g.

$f(a)$
$f(b)$
$g(a)$
$g(b)$
$h(b)$
$k(x): f(x), g(x), h(x)$

K.B.

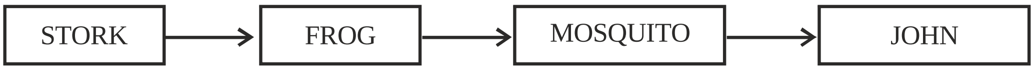
? K(y)



Recusion:

F1.justate (Mosquito, Jhon's blood)
F2.justate (frog, mosquito)
F3.justate (frog, mosquito)
R1 is digesting (X, Y): jutate (X, Y)
R2 is digesting (X, Y) : justate (X, Z); is digesting (Z, Y)

Recursive case:



OUTPUT:

IS-Digesting (X, Y)

	X	**Y**
R1 :	Mosquito	Jhons and blood
	Frog	Mosquito
	Stork	Frog
R2 :	Frog	John's blood
	Stork	Mosquito

Ancester:

Ancestor (X, Y) = X is Ancestor of Y
Child (X, Y) : Y is child of X

Child (a, b)
Child(b, c)
Child(c, d)

Ancestor(X, Y) : Child(X, Y)
Ancestor(X, Y) : Child(X, Z), Ancetor(Z, Y)

K.B.

$$X \downarrow Y \downarrow Z$$

$$X \downarrow Z \vdots \vdots Y$$

$$a \downarrow b \downarrow c \downarrow d$$

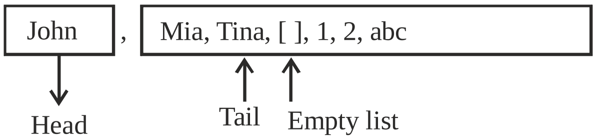
? Ancestor (X, Y)

X	Y
a	b
a	c
a	d
b	c
b	d
c	d

List:

Prolog supports only the list data structure.

List is a sequence of terms.



"Tail is always a list"

- [John] head = John
  tail = [ ] (empty)
- [(1, 2, 3)], [John, mia], [ ], abc 123]
  head = [1, 2, 3]
  tail [[John, mia], [ ], abc123]
- Built in operator (|)
  [head | tail]
  ? [X | Y]
- Concatenation ([X], [Y], [Z])
  Return true if the concatenation of X and Y is Z.

Negation as failure:

not (G) : G, if G is true then not (G) is false
not (G) : G, fail → Predicate is used with negation
not (G): G, !, fail; not(G)
not (G) : (G, !, fail); not (G)
if G is true then do not back track

Frames (Minskey):

The knowledge is catpured in a structured form. A frame is a static data structure that is used to represent well understood and stereotype situation.

Frames represent the knowledge in a structured form having slot, name and their values.

Frame – Name. e.g.

Slot name	Slot filter

Car	
Slot Name	Slot filter
Model Reg. No. Producer Owner	

Frames contains the following information:

- Frame identification name
- The attribute (slot name) and the attributed values (slot filter)
- Relation with other frames
- By default values of the slot names.

Attribute can be

(i) Static (ii) Dynamic

(i) Static Attribute values does not change, e.g. number of tyer in the Car.

(ii) Dynamic Attribute value changes from time to time. e.g. Owner of Car.

Slot value are obtained by:

(i) User (ii) Initialize (iii) From the other frame (iv) Inheritance

Different types of frame:

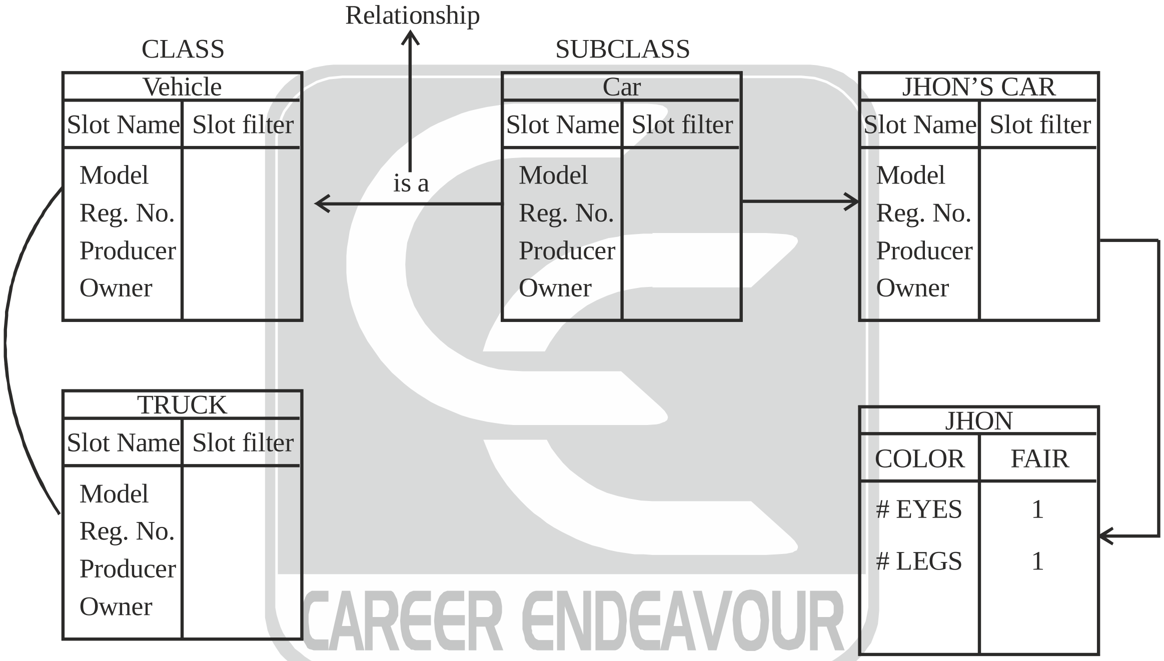
(i) Class Frame (ii) Subclass frame (iii) Instance frame

Class Frame:

Vehicle Frame:

Vehicle	
Slot Name	Slot filter
Model Reg. No. Producer Owner	

Example:



SCRIPTS

A script is a structure that describes a stereotyped sequence of events in a particular context. A script consists of a set of slots. Associated with each slot may be some information about what kinds of values it may contain as well as a default value to be used if no other information is available. Difference between script and frame is that in a script we can make more prescise statement about its structures. [Schank and Abelson Scripts, Plans, Goals and Understanding, Erlbaum, 1977].

A script is a data structure used to represent a sequence of events. Scripts are used for interpreting stories. Popular examples have been script driven systems that can interpret and extract facts from Newspaper Stories.

Scripts have been used to

(1) Interpret, understand and reason about stories,

(2) Understand and reason about observed events

(3) Reason about observed actions

(4) Plan actions to accomplish tasks.

A script is composed of

(1) A scene

(2) Props (objects manipulated in the script)

(3) The actors (agents that can change the state of the world).

(4) Events

(5) Acts: A set of actions by the actors.

In each scene, one or more actors perform actions. The actors act with the props. The script can be represented as a tree or network of states, driven by events.

As with Frames, scripts drive interpretation by telling the system what to look for and where to look next. The script can predict events.

Example of a script:

The classic example is the restaurant script:

Scene: A restaurant with an entrance and tables.

Actors: The diners, servers, chef and Maitre d'Hotel.

Props: The table setting, menu, table, chair.

Acts: Entry, Seating, Ordering a meal, Serving a meal, Eating the meal, requesting the check, paying, leaving.

Schema systems capture Frames, Scripts, and semantic network as networks of schema. Typically, schema represent relations between entities playing roles.

PRACTICE SET

1. Which of the following language is a declarative language?
(a) Algol (b) Java (c) Prolog (d) C#

2. Which one of the following is not a type of polymorphism
(a) coercion (b) overloading (c) overriding (d) generics

3. What formal system provides the semantic foundation for Prolog?
(a) Predicate calculus; (b) Lambda calculus; (c) Hoare logic; (d) Propositional logic.

4. Which of the following is the advantage of declarative languages over imperative languages?
(a) Can use abstract data type;
(b) Easy to verify the properties of the program;
(c) Is more efficient;
(d) Can be implemented by an interpreter or compiler;

5. Given the expression $((\lambda x \cdot x)\lambda x \cdot x)a$ in lambda calculus. Derive the expression as far as possible using β reduction. The final result will be:
(a) $(\lambda x \cdot x)a$ (b) $\lambda x \cdot x$ (c) xa (d) a

6. Which of the following captures all the pointcuts of calling private methods that take int as its only argument?
(a) call(private int *()) (b) call(private * *(int))
(c) call (private *(int)) (d) call(* private *(*))

7. In Scheme language, which of the following is not a higher-order function?
(a) map (b) member (c) apply (d) compose

8. What is the value of (map (lambda (x) (* 2 x)) ’(1 2 3)) ?
(a) a run-time error (b) ( ) (the empty list)
(c) 12 (d) (2 4 6)

9. Semantic Networks is
(a) A way of representing knowledge (b) Data Structure
(c) Data Type (d) None of the mentioned

10. Which of the following statement(s) is/are true:
A. Frames in artificial intelligence is derived from semantic nets.
B. frame is a data structure.
(a) A only (b) B only (c) Both A and B (d) None of the above

ANSWER KEY

1. (c) **2. (c)** **3. (a)** **4. (b)** **5. (d)** **6. (b)** **7. (b)**

8. (d) **9. (a)** **10. (c)**

CHAPTER

5

Expert System

An expert system is a computer program conceived to simulate some forms of human reasoning by the intermediary of an inference engine and capable to manage an important quantity of specialized knowledge.

A system that uses human knowledge captured in a computer to solve problems that ordinarily require human expertise (Turban & Aronson, 2001).

A computer program designed to model the problem solving ability of a human expert (Durkin, 1994).

An intelligent computer program that uses knowledge and inference procedures to solve problems that was difficult enough to acquire significant human expertise for their solutions (Feigenbaum).

An expert system is a computer application that solves complicated problems that would otherwise require extensive human expertise. To do so, it simulates the human reasoning process by applying specific knowledge and interfaces.

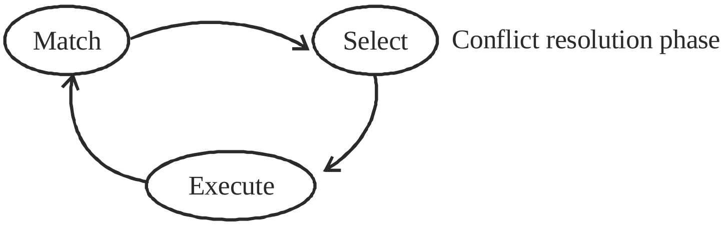
There are many well known advantages to using computerised tools and expert systems:

- reduction of missing data,
- better collection of data,
- no omission of questions,
- no data transcription,
- broader coverage of diagnosis

The most widely used way of representing domain knowledge in expert systems is as a set of production rules, which are often coupled with a frame system that defines the objects that occur in the rules.

Working of an Expert System:

First phase:



fact
a
b
c
$a \wedge b \wedge c \rightarrow p$
$p \rightarrow s$
$a \wedge b, c \rightarrow q, q \rightarrow p$

Knowledge Base

If a, b, c are true then $a \wedge b \wedge c$ are also true always.

$$\begin{array}{l} p \\ \underline{q} \\ p \wedge q \end{array} \text{ conjunction rule}$$

$$\begin{array}{ll} & a \wedge b \wedge c \\ & \underline{a \wedge b \wedge c \rightarrow p} \\ \therefore & p \\ & \underline{p \rightarrow s} \\ \therefore & s \end{array}$$

(i) Match the rule
(ii) Select any one of the rule
(iii) Execute the rule.
This process or cycle is repeated.

Backward Chaining:

$a \wedge b \wedge c \rightarrow p$
$p \rightarrow s$
$a \wedge b \wedge c \rightarrow q$
$q \rightarrow p$

Note : Facts are always considered to be true

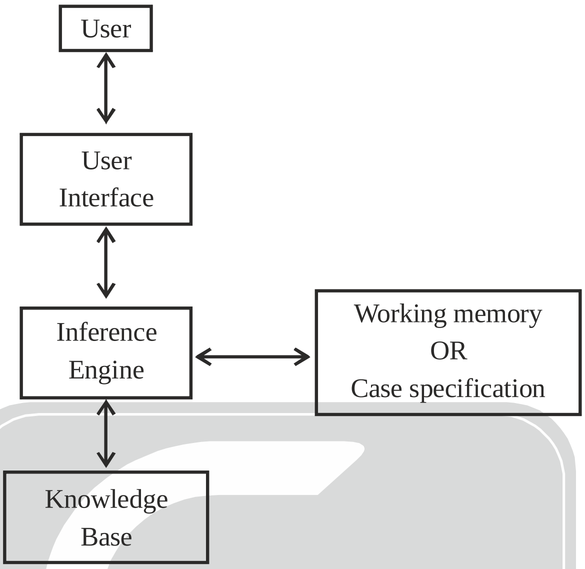
Proof :
s

$p \rightarrow (s)$ $\quad\quad\quad$ $(s) \rightarrow p \rightarrow a \wedge b \wedge c$

Expect system (Properties):

It is highly domain specific i.e. the domain of application is narrow. The chance of success is very high.
It must deliver the expert quality services. It is very close to Human expert.
It must have the facility of explanation.

Structure of expert system:



Knowledge Base : Here KB store only the rules
Working Memory: Contain the facts

Inference Engine: It asks questions from the user.

Expert system shell:
Mycin: Bacterial and Blood related disease expert system
Dendral : Molecular bond decrease expert system.
PXDES : Lung Disease diagnosis expert system
CADET : Early cancer detection expert system
ARGEX : Argriculture related expert system
GERMWATCHER : Bacterial infection controlling expert system.
An expert system without knowledge base is known as expert system shell.
Knowledge Engineer: Which collect the knowledge regarding the expert sytsem.
In expert system shell, knowledge base is absent because of the reuse of the expert system.

PRACTICE SET

1. Which of the following, is a component of an expert system?
(a) inference engine (b) knowledge base
(c) user interface (d) All of the mentioned

2. Following are the applications of Expert systems
(a) Disease Diagnosis (b) Planning and Scheduling
(c) Decision making (d) All of the above

3. Which of the followong statement(s) is/are true:
A. MYCIN and CADUCEUS are the well known Expert Systems for medical diagnosis systems
B. DARPA, the agency that has funded a great deal of American Artificial Intelligence research, is part of the Department of Defense
(a) A only (b) B only (c) Both A and B (d) None of the above

4. Which of the following is being investigated as a means of automating the creation of a knowledge base?
(a) automatic knowledge acquisition
(b) simpler tools
(c) discovery of new concepts
(d) All of the mentioned

ANSWER KEY

1. (d) **2. (d)** **3. (c)** **4. (d)**

CHAPTER

6

Advanced Topics

Recursive Transition Network (RTN) :

A recursive transition network (RTN) is a transition network which permits arc labels to refer to other networks including the network's own name, and in turn may refer back to the referring network rather than just permitting word categories used previously.

Recursive Transition Networks (RTNs) resemble Finite-State and Push-Down Automata (FSAs and PDAs) in several respects. In the first place, like FSAs and PDAs, RTNs are recognisers (acceptors) of sentences generated by phrase-structure grammars. In fact, RTNs are equivalent to PDAs in that they are capable of accepting sentences generated by Chomsky Type 2 (context-free) grammars. RTNs also resemble FSAs and PDAs in that their workings may be represented employing transition networks. RTNs actually developed out of the concept of the transition network associated with an FSA (and hence, their name).

Beyond their pedigree, however, RTNs share little else with FSAs and differ substantively from FSA transition networks. The most obvious of the differences between the two classes of machine lies in the labelling of the states (nodes) and arcs of their transition networks. While the states of the transition network of an FSA are labelled by nonterminal symbols of the grammar to which the automaton is equivalent, and the arcs are labelled by the terminals, the arcs of an RTN are labelled by the nonterminals of the equivalent grammar, and the nodes (states) are labelled arbitrarily, with the labelling employed solely to distinguish among them. Thus, there is normally no connection between the nodes of an RTN and the grammar to which it corresponds (is equivalent).

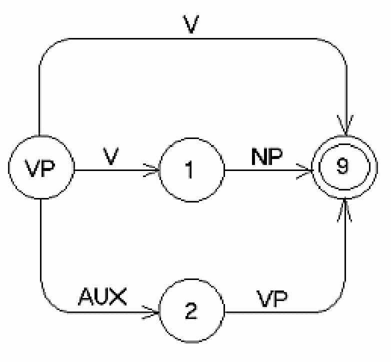
Note that, although the foregoing observations cite FSA transition networks, these statements apply equally to PDA networks, with the exception that in the case of the later machines, arcs corresponding to lambda/pop transitions are not labelled by terminal symbols of the equivalent grammar.

Also note that, not with standing the statements above regarding the labelling of RTN arcs only with nonterminal symbols, some special arcs can be treated as if they were labelled with terminals. In some treatments of RTNs, those arcs involved directly in the processing (reading and accepting) of terminal symbols can be labelled with the terminal symbols. This issue will be discussed below.

Another significant difference between RTNs and the transition networks of FSAs and PDAs is that, while the networks of the later machines consist of one complete network, the network of the former can be treated as a collection of subnetworks. There is one subnetwork for each distinct left- hand side symbol in the rules of the production set of the grammar to which the RTN is equivalent. For example, if the grammar includes the rewrite rules

VP $\rightarrow$ V,

VP → V NP, and
VP → AUX VP,



VP Subnetwork

AUGMENTED TRANSITION NETWORK

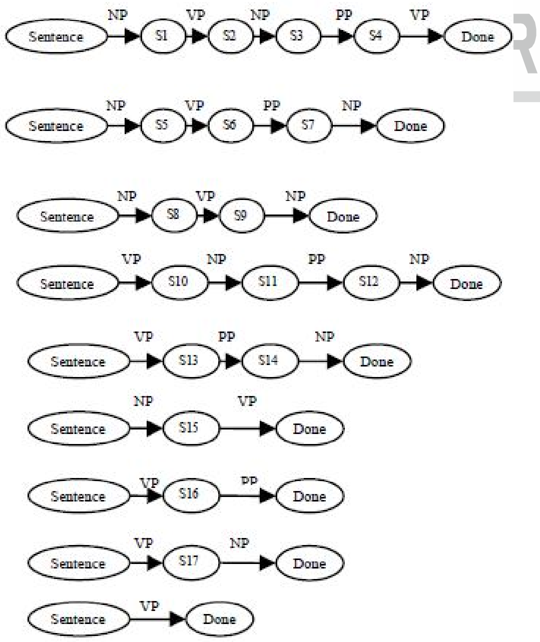
The transition networks and recursive transition networks are not very useful for language understanding. They have only been capable of accepting or rejecting a sentence based on the grammar and syntax of the sentence. To be more useful, an interpreter must be able to build structures which will ultimately be used to create the required knowledge entities for an AI system. The additional capabilities required can be achieved by augmenting an RTN with the ability to perform additional tests and store immediate results as a sentence is being parsed. When an RTN is given these additional features, it is called an augmented transition network (ATN). An ATN is like a finite state transition network, but is augmented in three ways:

(a) Arbitrary tests can be added to the arcs. A test must be satisfied for the arc to be traversed.

(b) Structure-building actions can be added to the arcs. These actions may save information in registers to be used later by the parser, or to build the representation of the meaning of the sentence.

(c) Phrase names, as well as part-of-speech names, may appear on arcs. This allows a grammar to be called as a subroutine.

The combination of these features gives the ATN the power of a turing machine, i.e., it can do anything a computer program can do. In other words, an ATN can recognize any language that a general purpose computer can recognize. This versatility also makes it possible to build deep sentence structures rather than just structures with surface features only.


Sentence NP S1 VP S2 NP S3 PP S4 VP Done
Sentence NP S5 VP S6 PP S7 NP Done
Sentence NP S8 VP S9 NP Done
Sentence VP S10 NP S11 PP S12 NP Done
Sentence VP S13 PP S14 NP Done
Sentence NP S15 VP Done
Sentence VP S16 PP Done
Sentence VP S17 NP Done
Sentence VP Done

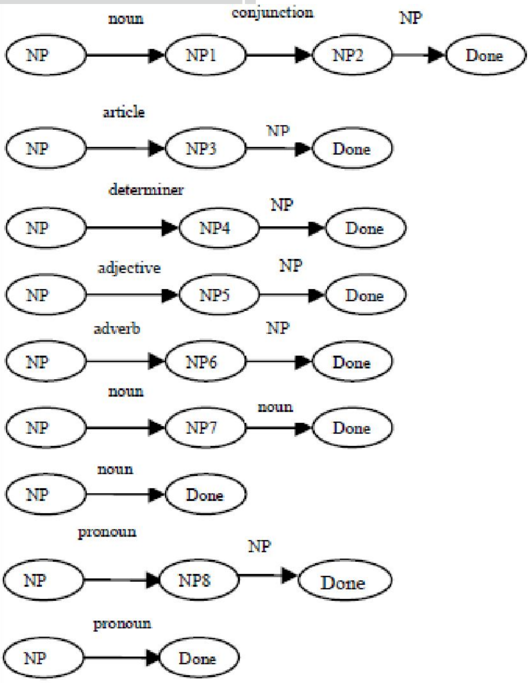


Figure: Top Level ATNs for sentence

Figure: ATNs for NP (Noun Phrase)

The shortcomings of CFG:

(a) Difficulty in dealing with different sentence structures that has the same meanings. Typically, the grammar has to be expanded to handle many special cases.

(b) Handling number agreement between subjects and verbs. To handle the agreement phenomenon with context free grammar (CFG) parsers the size of the grammar needs to be doubled.

(c) Determining the "deep structure" of input texts. With CFG parsers it is very difficult to determine the deep structure of a sentence. By using ATN parser one can easily build the deep structure of a sentence.

PARSING WITH ATN PARSER

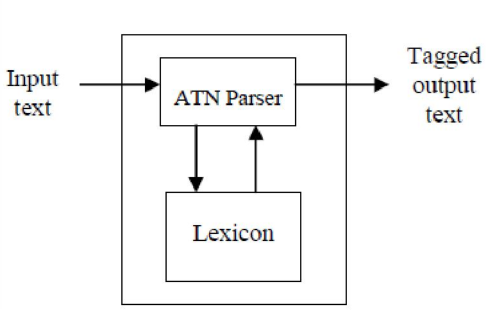
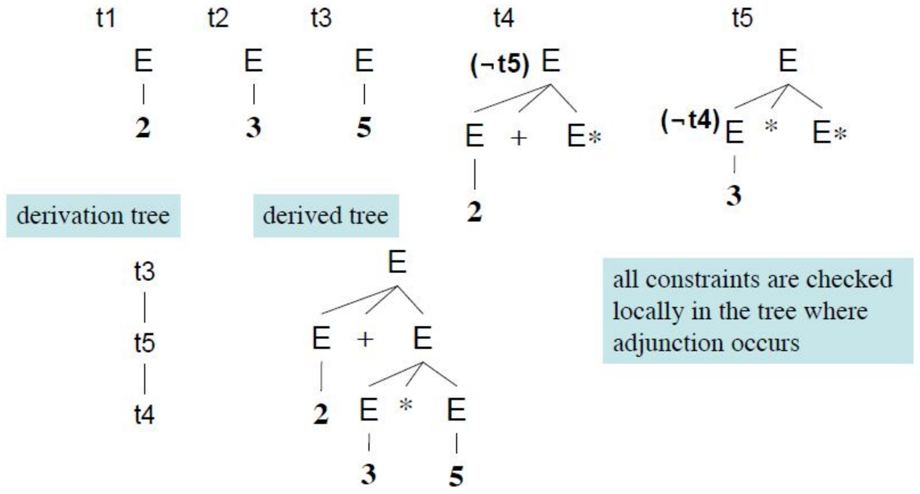


Figure: Design Aspects of ATN Tagger

PARSING WITH TREE-ADJOINING GRAMMARS

- Construct a tree set out of tree fragments
- Each fragment contains only the structure needed to express the locality of various CSG predicates
- Each tree fragment is called an elementary tree
- In general we need to expand those nonterminals that are not leaf nodes, leads to the notion of adjunction



TAG: Tag is capable of parsing ambiguous grammar (CFG). So, it helps in Natural Language Processing (NLP).

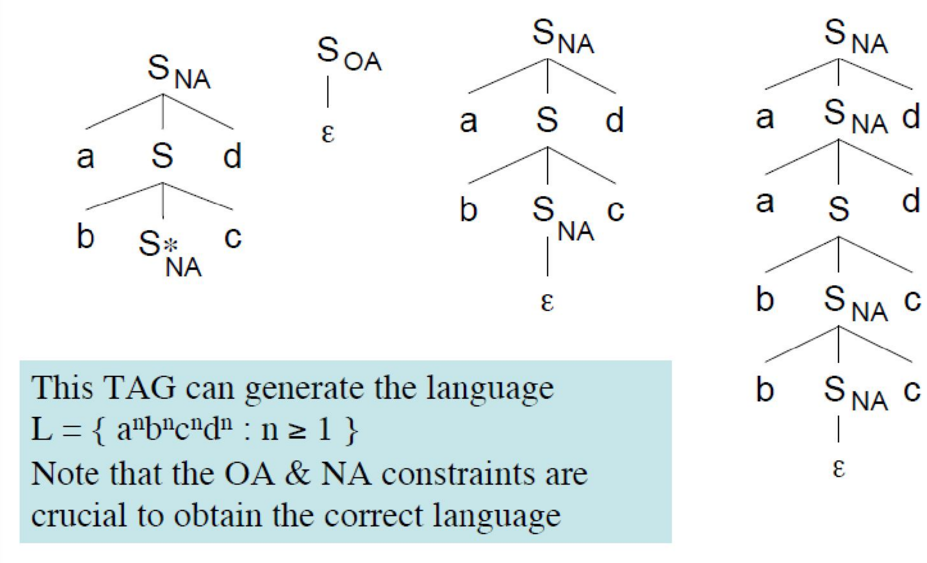
Formal Language Theory	NLP
Language (possibly infinite)	Text Data, Corpus (finite)
Grammar	Grammar (usually inferred from data, produces infinite set)
Automata	Recognition/Generation algorithms

- **Weak Generative Capacity** of a grammar is the set of strings or the language
- **Strong Generative Capacity** of a grammar is the set of structures (usually the set of trees) produced by the grammar

Formaly a TAG G = (N, T, I, A, S) where

- N is the set of non-terminal symbols
- T is the set of terminal symbols
- I is the set of initial or non-recursive trees built from N, T and domination predicates
- A is the set of recursive trees: one leaf node is a nonterminal with same label as the root node
- S is set of start trees (has to be initial)
- I and A together are called *elementary trees*

Adjunction Constraints:



This TAG can generate the language
L = { $a^n b^n c^n d^n$: n ≥ 1 }
Note that the OA & NA constraints are crucial to obtain the correct language

TAG FORMAL PROPERTIES

- Membership is in P: O(n^6)
- Tree-Adjoining Languages (TALs) are closed under *union, concatenation, Kleene closure* (*), *h*, *h*-1, *intersection with regular languages*, and *regular substitution*
- There is also a pumping lemma for TALs
- TALs are a full abstract family of languages (AFL)
- TALs are not closed under intersection, intersection with CFLs, and complementation.

MANAGEMENT INFORMATION SYSTEM (MIS)

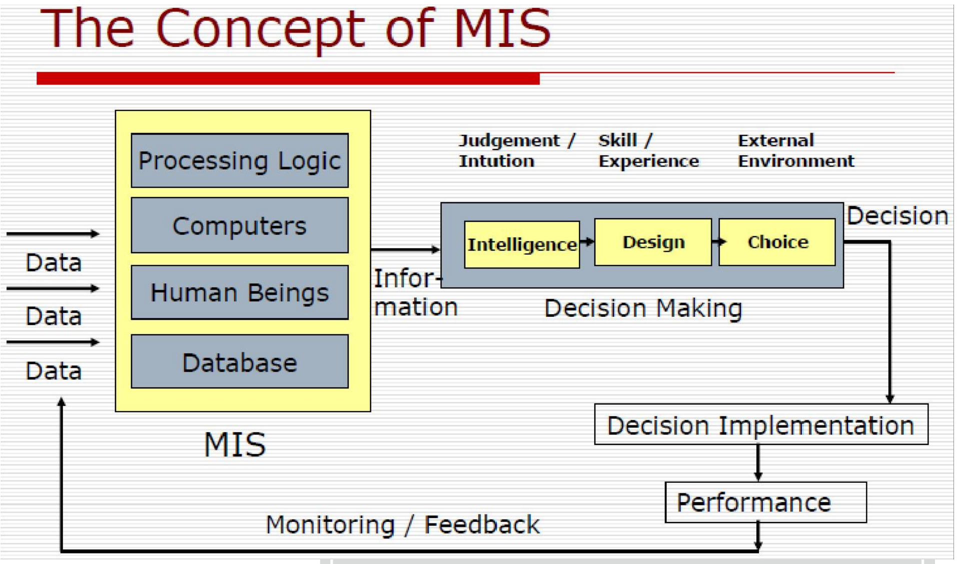
A management information system (MIS) is a system or process that provides the information necessary to manage an organization effectively. MIS and the information it generates are generally considered essential components of prudent and reasonable business decisions.

Management information systems involve three primary resources: technology, information, and people. It's important to recognize that while all three resources are key components when studying management information systems. The most important resource is people. Management information systems are regarded to be a subset of the overall internal controls procedures in a business, which cover the application of people, documents, technologies, and procedures used by management accountants to solve business problems such as costing a product, service or a business-wide strategy. Management information systems are distinct from regular information systems in that they are used to analyze other information systems applied in operational activities in the organization. Academically, the term is commonly used to refer to the group of information management methods tied to the automation or support of human decision making, e.g. Decision Support Systems, Expert systems, and Executive information systems.

There are many types of information management systems in the market that provide a wide range of benefits for companies. Strategic information management system, customer relation management systems and enterprise resource planning systems are some of them.

Advantages of information management systems:

1. The company is able to highlight their strength and weaknesses due to the presence of revenue reports, employee performance records etc. The identification of these aspects can help the company to improve their business processes and operations.
2. The availability of the customer data and feedback can help the company to align their business processes according to the needs of the customers. The effective management of customer data can help the company to perform direct marketing and promotion activities.
3. Information is considered to be an important asset for any company in the modern competitive world. The consumer buying trends and behaviors can be predicted by the analysis of sales and revenue reports from each operating region of the company.



The Concept of MIS

Processing Logic
Computers
Human Beings
Database
MIS

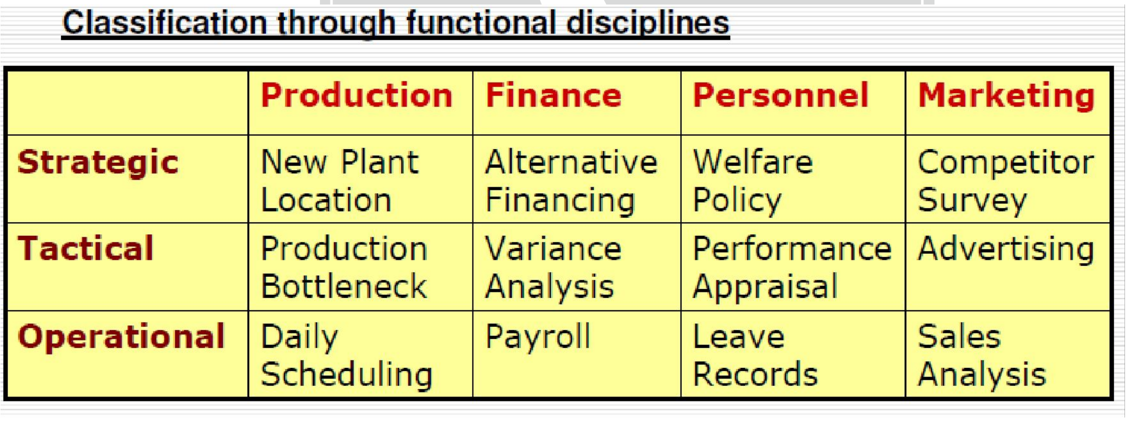
Data
Data
Data

Infor-
mation

Judgement / Intution
Skill / Experience
External Environment

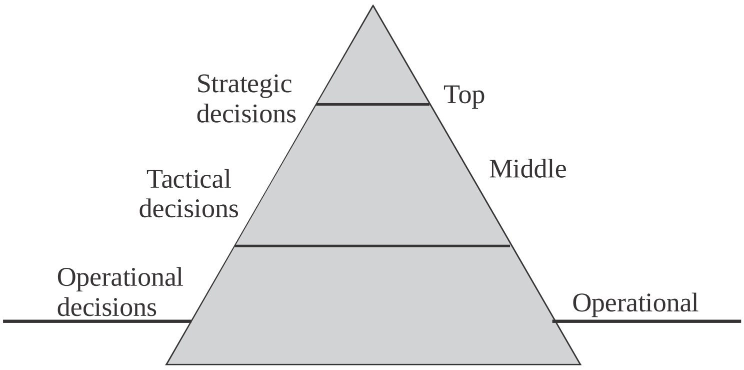
Intelligence
Design
Choice
Decision Making
Decision

Decision Implementation
Performance
Monitoring / Feedback



Classification through functional disciplines

	Production	Finance	Personnel	Marketing
Strategic	New Plant Location	Alternative Financing	Welfare Policy	Competitor Survey
Tactical	Production Bottleneck	Variance Analysis	Performance Appraisal	Advertising
Operational	Daily Scheduling	Payroll	Leave Records	Sales Analysis



Management information system:

Based on research model, the challanger of MIS implementation issues are grouped into five main categories as presented by Beaomaster (1999). By compairing empirical data and theoretical parts, analysis will be broken down into five main issues as follows.

(1) Leadership issue: Leadership issue effect reflect those areas which requires the interaction, commitment and direction from the top management, such as interdepartmental coordination, organization support, individual support and time frames and scheduling. The inter departmental coordination relates to the ability of the organization to coordinates its implementation, process across departments.

The point of view shows that in MIS implementation process requires organization and individual support in order to achieve the goals and objectives

(2) Organisational environment issue:

Organisational environment issue are essential to define the challanges which affect or might be affected by environmental factors. The findings indicate that fenix system was implemented in defence organization. Thus, security and confidentiality are the main issues. According to the finding, before the system goes live, a number of documents descriptions of the system and so forth need to get approval from SAF in order to ensure the security issues.

(3) Management process issue:

One challenge in the management process is referred by the respondents in strategic plannings of the implementation project. At the beginning, one of the problems regarding information technology implementation can be seen as lack of a strategic plan.

(4) Personal issue:

Personal issue is one of the major problematic issue regarding the MIS implementation process. When it comes to personal issue, it states that when the five components of an information system (i.e. hardware, software, data, procedures, and people) are considered the most important component is people for instance even if an organization that has a perfect information system.

(5) Technical system issues:

Technical systems issues are primarily related to the impact information technology on organizations and individuals. In addition, these issues include hardware and software consideration as well as the compatibility and life cycles of various information technologies.

DECISION SUPPORT SYSTEM

A decision support system (DSS) is a computer-based information system that supports business or organizational decision-making activities. DSSs serve the management, operations, and planning levels of an organization and help to make decisions, which may be rapidly changing and not easily specified in advance.

DSSs include knowledge-based systems. A properly designed DSS is an interactive software-based system intended to help decision makers compile useful information from a combination of raw data, documents, personal knowledge, or business models to identify and solve problems and make decisions.

Typical information that a decision support application might gather and present are:

1. Inventories of information assets (including legacy and relational data sources, cubes, data warehouses, and data marts),
2. Comparative sales figures between one period and the next,
3. Projected revenue figures based on product sales assumptions.

Desirable Features of DSS Evaluation Method:

1. DSS development is evolutionary in nature.
2. DSS are designed to support organization objectives
3. Different levels of users are involved in DSS development.
4. Most DSS benefits are qualitative
5. A DSS should be flexible in modeling the related business tasks.

Desirable Features	Operational Features/Effectiveness Criterion
1. Support for DSS Evolution	(a) Integrated support for DSS planning and desing.
2. Align DSS with organizational objectives	(b) Goal orientation.
3. Support for multiple evaluation/users.	(c) Prioritizations/synthesis of viewpoints
4. Assessment of qualitative benefints	(d) • Qualitative benefits measurement • Consistent/comparable measurement • Incorporation of uncertainties.
5. Support DSS flexibility	(e) • Modeling capabiltiy • Adaptability of the approach.

COGNITIVE STYLE

Cognitive style or "thinking style" is a term used in cognitive psychology to describe the way individuals think, perceive and remember information. Cognitive style differs from cognitive ability (or level), the latter being measured by aptitude tests or so-called intelligence tests. Controversy exists over the exact meaning of the term cognitive style and also as to whether it is a single or multiple dimension of human personality. However, it remains a key concept in the areas of education and management. If a people has a cognitive style that is similar to that of his/her teacher, the chances that the people will have a more positive learning experience are improved. Likewise, team members with similar cognitive styles likely feel more positive about their participation with the team. While matching cognitive styles may make participants feel more comfortable when working with one another, this alone cannot guarantee the success of the outcome.

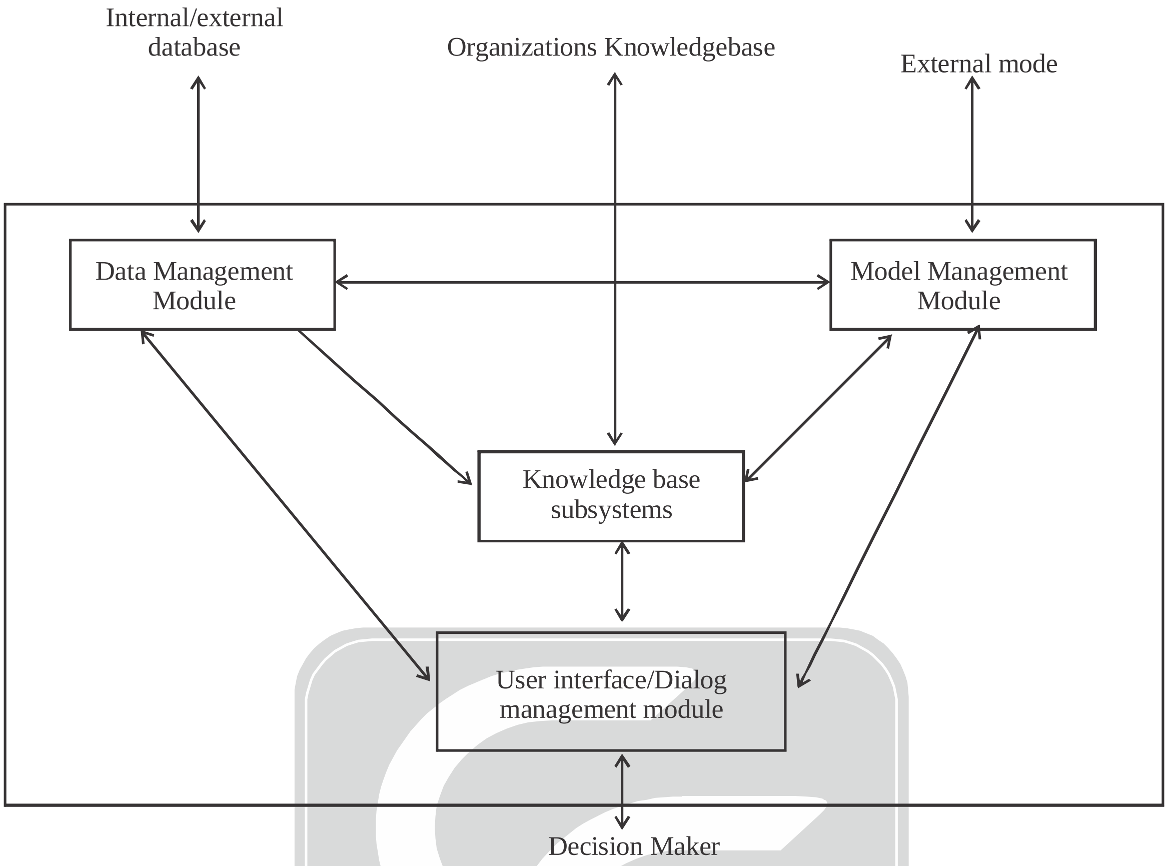
Weakness of cognitive style for designing a DSS:

1. Inadequately developed theory of cognitive styles.
2. Multitude of measuring instruments with inadequately established psychometric properties (e.g.) reliability and validity)
3. Faulty research designs used in impirical investigation of cognitive styles.

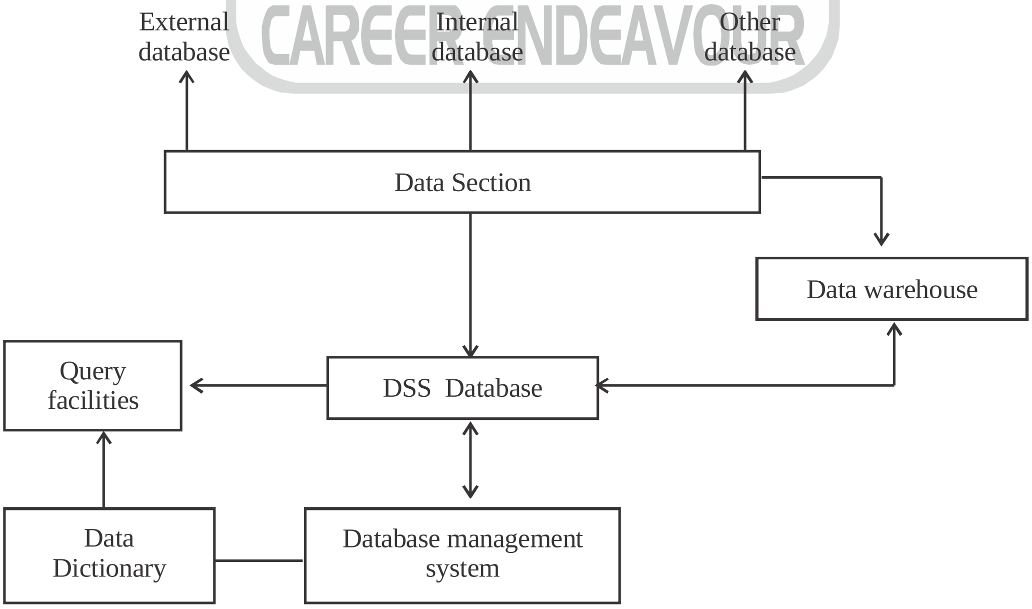
Decision support system:

According to Sprague and Cartson, decision support systems would consist in the following components, data management component, model management component, user interface mangement component and decision support system architecture. Nowadays, the decision support systems are very much like the ones identified by Sprague in 1982. The user interface is a component that provides the communication between the user and the decision supports system. The proper design of this component is really important as it is the only one the user actually deals with. The data management method is a subsystem of the computer based decision support system and has a number of subcomponents of its own.

- The integrated decision support system database, which includes data extracted from internal and external sources data which can be maintained in the database or can be accesed only when is useful.
- The database management system can be relational or multidimensional.


Internal/external database
Organizations Knowledgebase
External mode
Data Management Module
Model Management Module
Knowledge base subsystems
User interface/Dialog management module
Decision Maker

- A data dictionary implying a catalog containing all the definitions of database data. It is used in the decisional process identification and definition phase.
- Query tools assuming the experience of languages for querying database



The model management module consist of following components.

- The model base that contain the quantitative models that offer the system the capacity of analyzing and finding solutions to problems.
- The model base management module that is meant to create new models by using programming languages.
- The model dictionary that contains the models definition and other information related to them.
- The creation execution and integration module of models that will transfer them towards model management system.

One naturally raises questions about the future of DSS will it follow the foot steps of MIS? A field with out definition has the flexibility of expansion and charging direction but also has the danger of falling apart. Consequently it is possible to generate heated debate and disagreement.

Example of DSS:

(1) Non-programmable calculator
(2) Programmable calculator
(3) Financial modelling
(4) Spreadsheet
(5) Statistics package
(6) Expert system.

Characteristics of DSS:

- They trend to be aimed at the less well structured, under specified problem that upper level manages typically face.
- They attempt to combine the user of model or analytics techniques with traditional data access and reterieval functions.
- They specifically focus on features that make them easy to use by non-computer people in an interactive mode.
- They emphasise flexibility and adaptibility to accomodate changes in the environment and decision making approach of the user.

Group Decision Support Systems:

To support a set of decision makers working together as a group, DSS need a typical combination of hardware, software people and procedures, each member of the group have a personal computer linked to the personal computers of the other members of the group and to one or more large public viewing screens so that each member can see the input of other members or let other members to see their work.

Computers based information systems which are meant to support group (multi participant) activities have been known under various names such as group decision support systems, computer supported corporative work group support systems corlaboration support system and electronic meeting systems.

The group DSS is an interactive communication and computer based system utilized to facilitate solution of unstructured problem by a set of co-operating decision markers. It consist of most of the DSS elements plus a specific software to support group decisions. GDSS is a particular subclass of the more general class of computerized collabarative.

A GDSS supports more than one person which works on a shared task. Several people work together to come up with a set of decisions to implement a solution or strategy.

A general architecture of a GDSS includes:

- The hardware element which is the conference facility which include the decision room, computers, internet access and other mean of communcation.
- Software tools which are web based applications, e-questionaires, e-brainstorming tools, policy formation tools and people who include the decision makers themselves and in many cases a trained facilators and possibly the support staft.

PRACTICE SET

1. An ‘agent’ is anything that,
(a) Perceives its environment through sensors and acting upon that environment through actuators
(b) Takes input from the surroundings and uses its intelligence and performs the desired operations
(c) A embedded program controlling line following robot
(d) All of the mentioned

2. Agents behavior can be best described by
(a) Perception sequence
(b) Agent function
(c) Sensors and Actuators
(d) Environment in which agent is performing

3. Given the following definitions:
(define f (lambda (x) (lambda (y) (+ x y)))) (define (g x) ((f x) 3)) (define h (lambda (x) (lambda (y) (y x))))
(define (p f) ((h 2) f))
(a) What is the return value of (g 2) ?
(b) What is the return value of (p +) ?

4. Assuming that the following definitions are executed in this order:
(define b (3 14 27))
(define c (cons (car (cdr b)) (list a b c)))
What is the result of typing the following into the Scheme interpreter:

Answer : (14 a b c) (car (cdr (cdr c)))

5. Write the following two Prolog programs:

(a) findnth that will find the Nth element in a list. For example, findnth([11, 12, 13, 14, 15], 4, X) returns X=14.
(b) subset(A,S), which succeeds if the set A is a subset of the set S. For example, subset([2,5], [1,5,3,2]). returns Yes.
findnth([H|T], 1, H).
findnth([H|T], N, X):- N1 is N-1, findnth(T,N1,X).
subset([ ],Y).
subset([A|X],Y) :- member(A,Y), subset(X,Y).

ANSWER KEY

1. (d) **2. (b)** **3. (a_5, b_2)** **4. (b)**