CS8691 Artificial Intelligence

UNIT -I INTRODUCTION

Introduction - Definition - Future of Artificial Intelligence - characteristics of Intelligent Agents - Problem Solving Approach to typical AI problems

1. Definition

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

Definitions of AI - organized into 4 categories

(i) Systems that think like humans
ii) Systems that act like humans
iii) Systems that think rationally
iv) Systems that act rationally

2. Future of Artificial Intelligence

* AI is impacting the future of virtually every industry and every human being.
* AI has acted as the main driver of emerging technologies like big data, robotics and IOT and it will continue to act as a technological innovator for the foreseeable future
* In future it is expected of AI to beat a human in all cognitive tasks

* AI has many applications in almost every field

* AI is now introduced to our day to day lives
  - we use Alexa, Google home
  - In email writing, we can see the recommendation to complete sentences
* AI technologies are assisting in Military Action, weapons technology
* AI has enhanced cybersecurity and so now the security systems are able to recognise a pattern of cyber threats and can create a counter-attack security tool

Major fields of AI in future

1. Health care Industries

* AI has the ability to provide the facility to detect disease based on symptoms.
* AI read data from the fitness band / medical history of individual to analyse the pattern & suggest proper medication through cell phone
* Eg. Google's deep mind - detect fatal diseases like breast cancer

2. AI in Education

* AI is going to transform the classical way of education in future

Personalization - With the use of AI, Students now have a personalized approach to learning programs based on their unique experiences & preferences

Answering Questions - AI helps the students in finding answers to the most commonly asked Questions through support automation & conversational intelligence.

Universal 24/7 access to learning - AI powered tools make learning accessible for all students, anytime and anywhere.

* AI plays a role in helping students and teachers optimize and automate both learning and teaching tasks.

3) AI in transportation and Manufacturing Sectors

* Evolution of self driving cars (or) Autonomous Vehicle (AV) through AI make the transportation easier one
* eg. 2002, a self driving car company Amazon bought in June 2021.
  Google's driverless cars and Tesla's Autopilot features are the introduction of AI into automatic sector.
* They have the ability to predict the destination through patterns via AI.

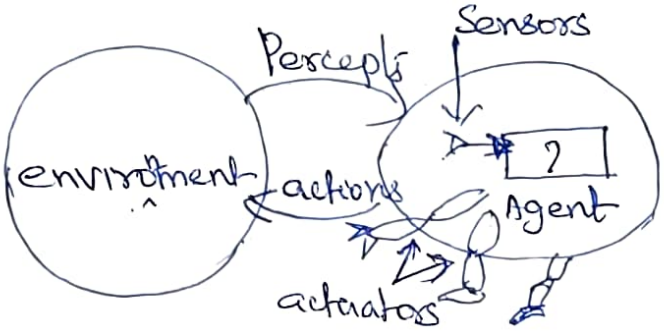
4) E-commerce

* Amazon Go Store attempts to bridge the gap between ecommerce and brick and morter stores
* Advanced supply chain management systems & delivery drones are examples of AI

1.3 Characteristics of Intelligent Agents:

Agents:

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through actuators

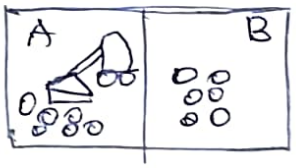


Examples

* Human Agent
* Robotic Agent
* Software Agent

Agent = Architecture + program

Eg: Vacuum - cleaner world with two locations



Percepts: location & status
eg [A, Dirty]

Actions: Left, Right, Suck, Noop

Percept Sequence	Action
[A, clean]	Right
[A, Dirty]	Suck
[B, clean]	Left
[B, Dirty]	Suck

Agent program

```
function Vacuum-Agent
(Location, Status)
if status = Dirty then
  return Suck
else if location = A then
  return Right
else if location = B then
  return Left
```

Intelligent Agent

This agent has some level of autonomy that allows it to perform predictable and repetitive tasks for users or applications "Intelligent" means ability to learn during the process of performing tasks

Characteristics of Intelligent Agents

1) They have some level of autonomy that allows them to perform certain tasks on their own
2) Exhibition of goal oriented behaviour
3) Transportable over networks Mobility
4) Dedication to a single repetitive work
5) Ability to interact with humans, systems and other Agents
6) New rules can be accommodated incrementally
7) They are knowledge based. They use knowledge regarding communications, Processes and entities.

1.4 Types of Agents

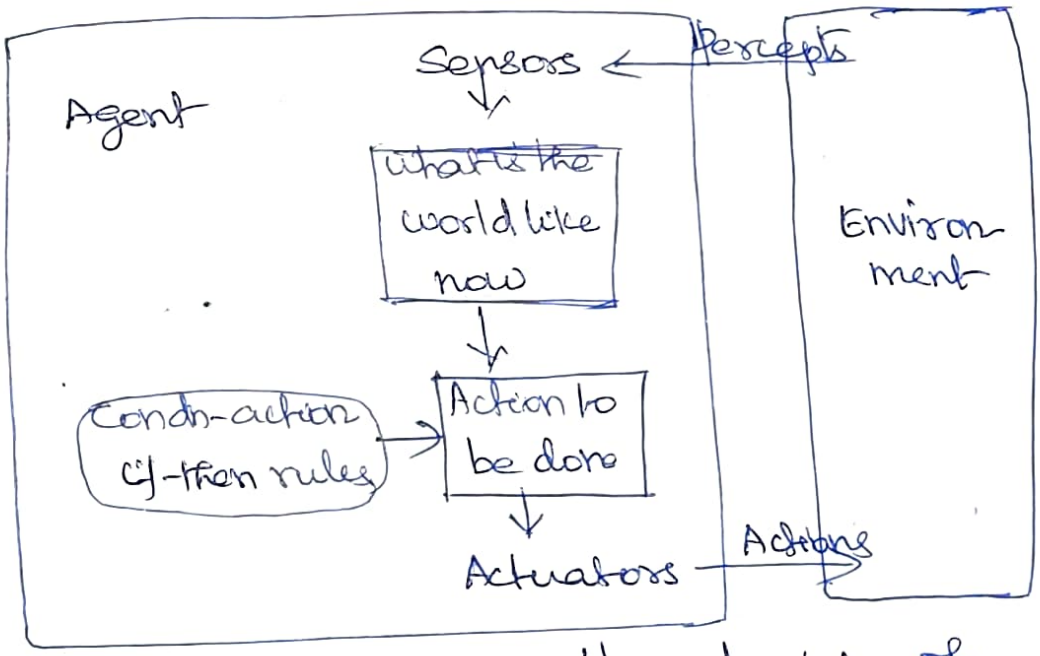
Agents can be grouped into 5 classes based on their degree of perceived intelligence and capability

1) Simple Reflex Agents
2) Model Based Reflex Agents
3) Goal Based Agents
4) Utility Based Agents
5) Learning Agent

1) Simple reflex Agents

- The simplest kind of Agent.
- These agents select actions on the basis of current percept, ignoring the rest of percept history.
- This agent describes about how the Condition — action rules allow the agent to make the connection from percept to action

Eg: If car in front is braking then initiate break



rectangle ▭ — denote current internal state of agents' decision process

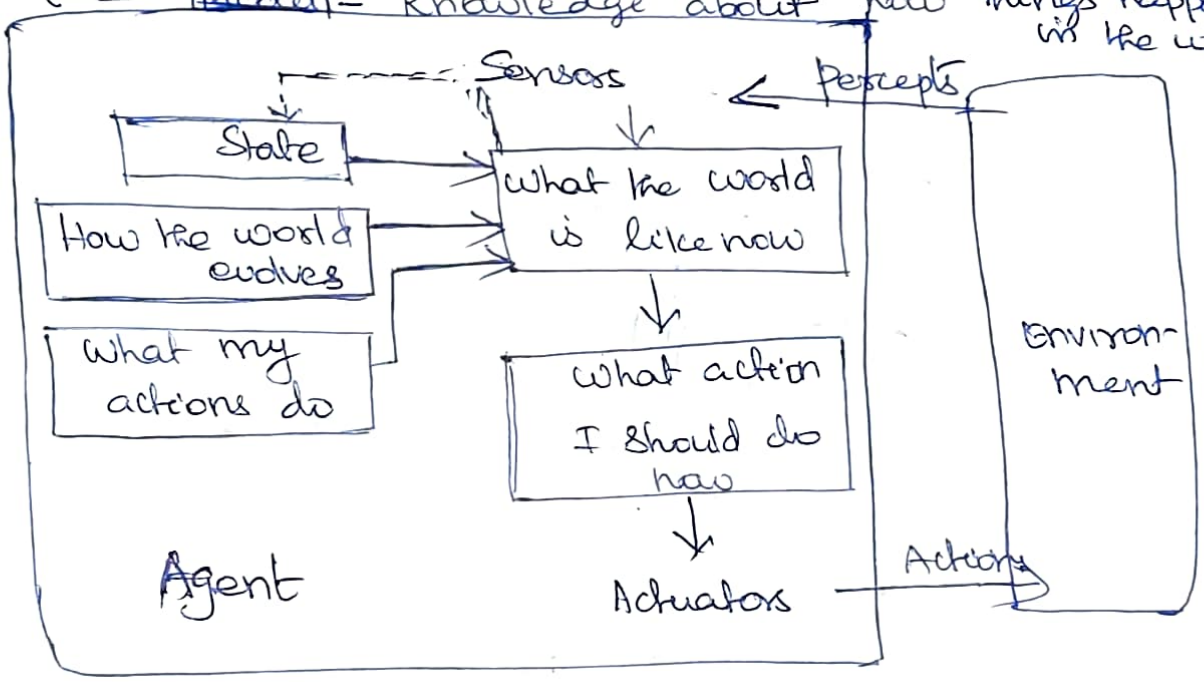
oval ⬭ — background information used in the process

2) Model based reflex Agents

An agent which combines the current percept with the old internal state to generate updated description of the current state

- It works by finding a rule whose condition matches the current situation.

2 Factors
- The agent has to keep track of internal state
- Model - Knowledge about "how things happen in the world"

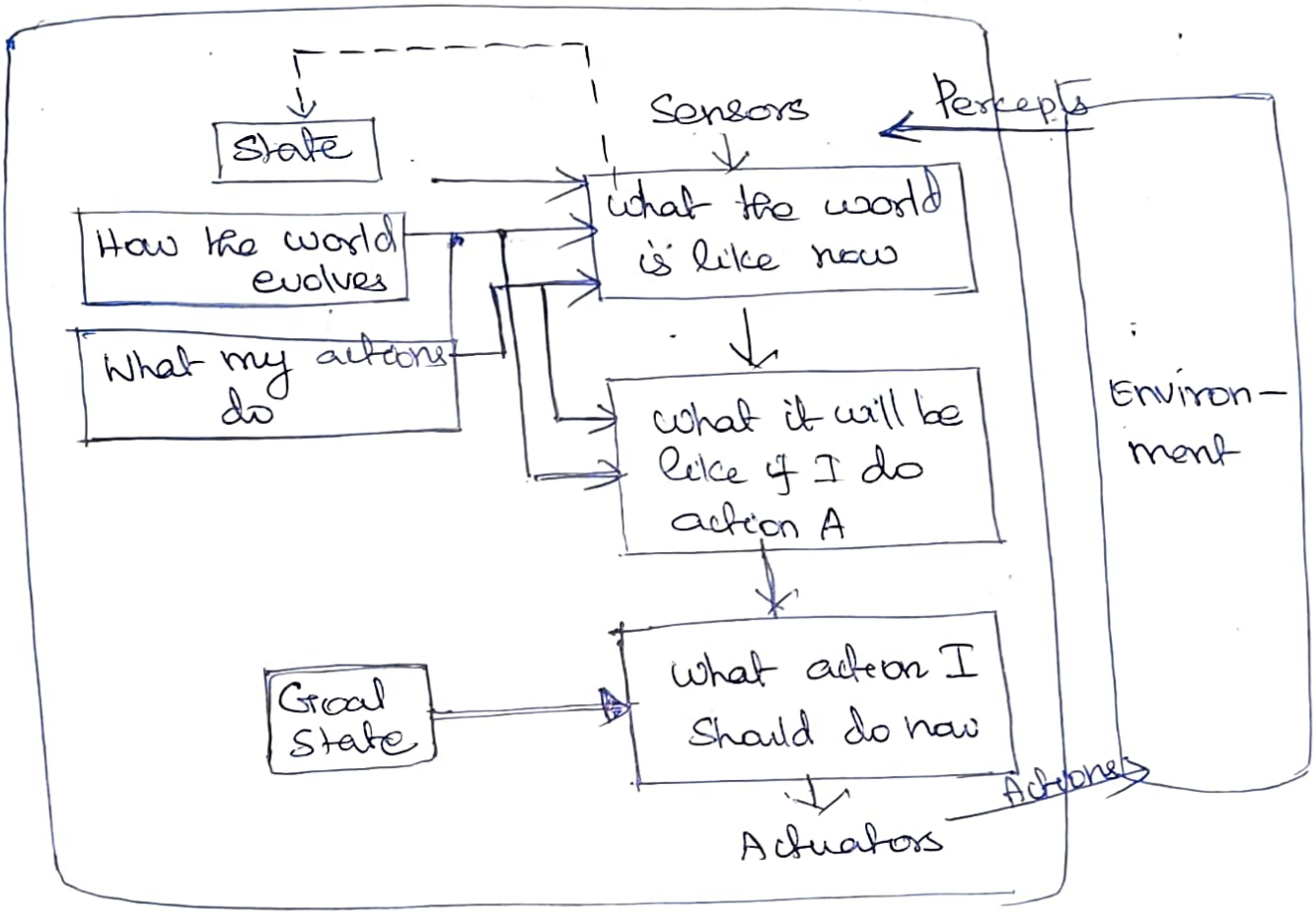


* updating an agent state requires information about
a) How the world evolves
b) How the agent's action affects the world

* The model based agent can work in a partially observable environment, and track the situation.

3. Goal based Agents

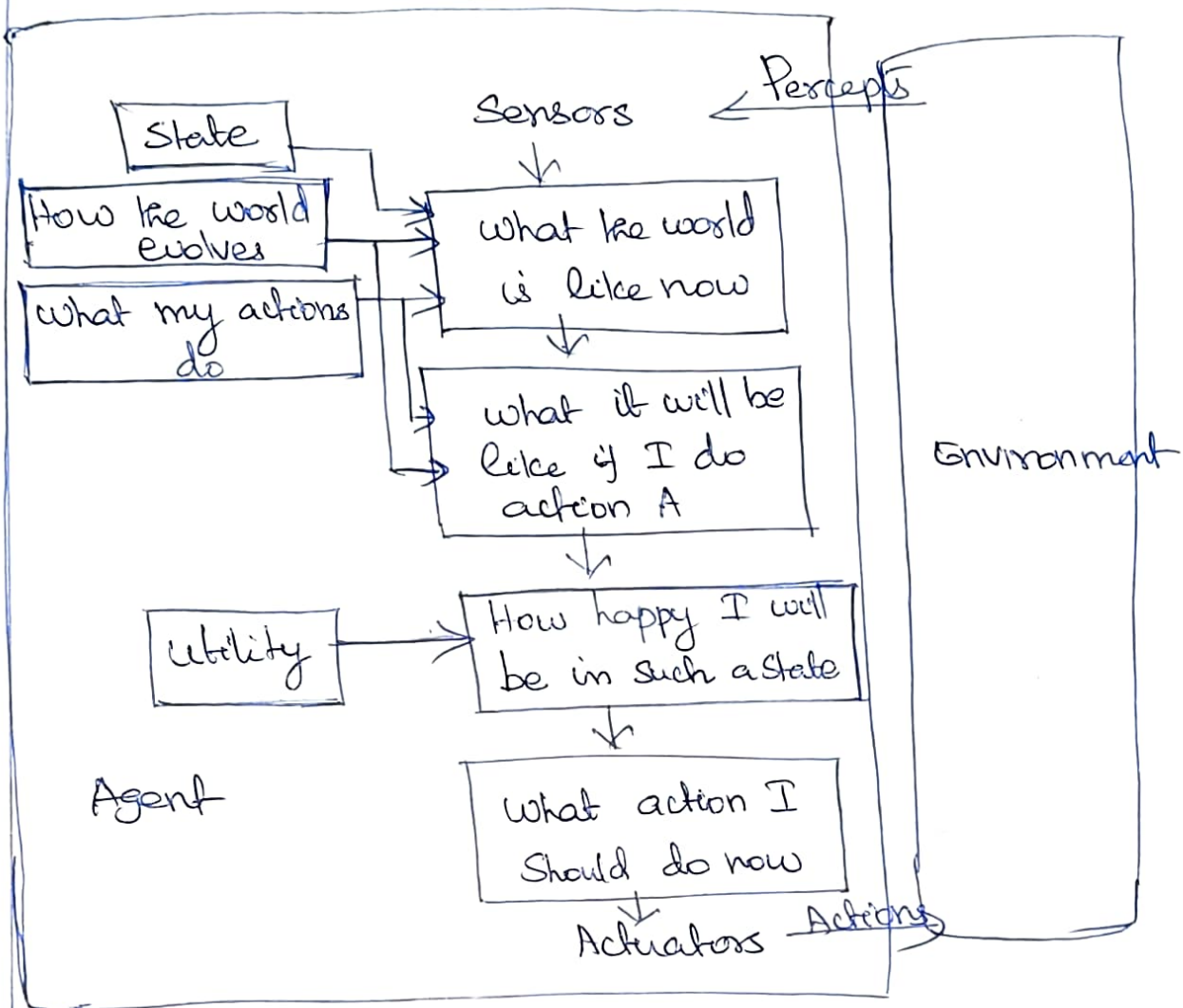
* An agent knows the description of current state as well as goal state. The action matches with the current state is selected depends on the goal state
* This agent select the one among multiple choices which reaches a goal state
* This agent requires search & planning
* Goal based agent's behavior can be changed



4. Utility based Agents

* This agent generates a goal state with high-quality behavior.
* It is useful when there are multiple possible alternatives and an agent has to choose in order to perform the best action.

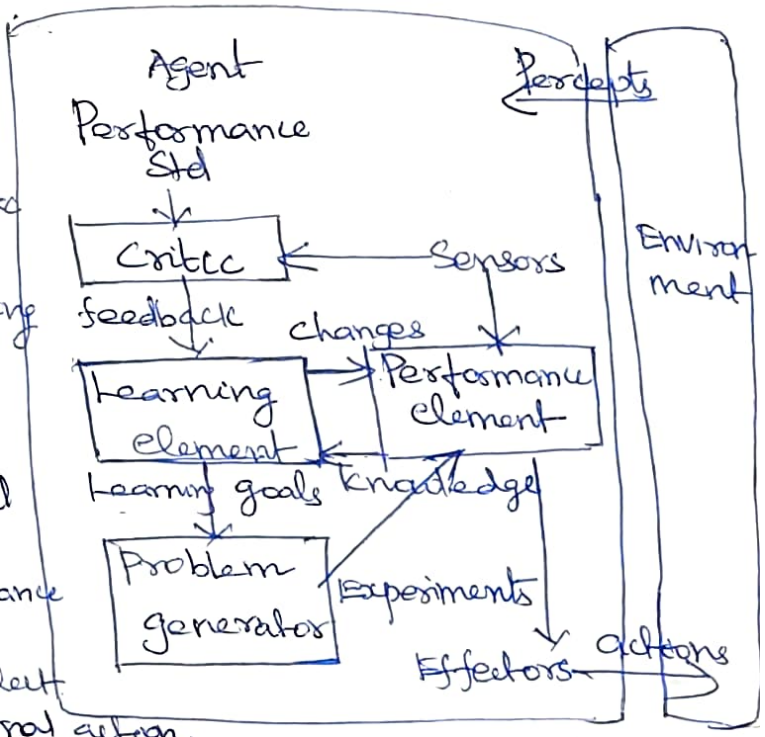
* The utility function maps each state to a real number ~~to check~~ which describes the associated degree of happiness.



5) Learning Agent

* This agent can learn from its past experiences
* It starts to act with basic knowledge and adapt automatically through learning
* It has 4 conceptual components

* Learning element
* Critic - describes how well the agent is doing with respect to a fixed performance std
* Performance element - select external action
* Problem generator - responsible for suggesting actions that will lead to new and (9) informative experiences



Applications of AI

* Autonomous planning and scheduling
  - NASA's Remote Agent program
* Game playing
  - IBM's Deep Blue - first computer program to defeat world champion Gary Kasparov
* Autonomous control
  - ALVINN computer Vision System was trained to steer a car to keep it following a lane
* Diagnosis - Medical diagnosis program
* Logistics planning - DART - for Transportation
* Robotics - Unimate (1954) - Universal Automation
* Language understanding & problem Solving -
  PROVERB - computer program to solve crossword puzzles

5 Problem solving Approach to typical AI problems

Problem solving Agents

* Rational Agents (or) Problem solving agents use Search Strategies (or) algorithms to solve a specific problem and provide the best result.
* Problem solving agents are goal based agents and use atomic representation

5 Important problems solved using AI are

1. Chess
2. Travelling Salesman problem
3. Towers of Hanoi problem
4. Water Jug problem
5. N-Queen problem
6. 8-puzzle problem

Steps to Solve problems using AI

There are 5 steps

1. Defining the problem
   - It should be stated clearly & precisely
   - It should contain the possible initial and final situation of acceptable solution
2. Analyzing the problem
3. - Analyse the problems and its requirements must be done
3. Identification of Solution
   - Generate reasonable amount of solutions to the given problem
4. Choosing a solution
   - From all the identified solutions, the best solution is choosen
5. Implementation -- Best solution is implemented

Well defined problems

A problem can be defined formally by 4 components

1. Initial State - State in which agent starts
2. Successor function - S(x) returns the set of states reachable from x by any single action
3. Goal State - Determines whether the given state is goal state or not
4. Path cost - Function which assign a numeric cost to each path
   - Solution
   - ↳ it reflects agent performance measure

Example problems

A Toy problem is intended to illustrate various problem solving methods

A real world problem is one whose solutions people actually care about

Toy problems

1) Vacuum World

States: The state is determined by both the agent location and dirt location

* The agent is in one of the 2 locations, each of which might or might not contain dirt. So there are $2 * 2^2 = 8$ possible world states

Initial State : Any state can be designated as the initial state

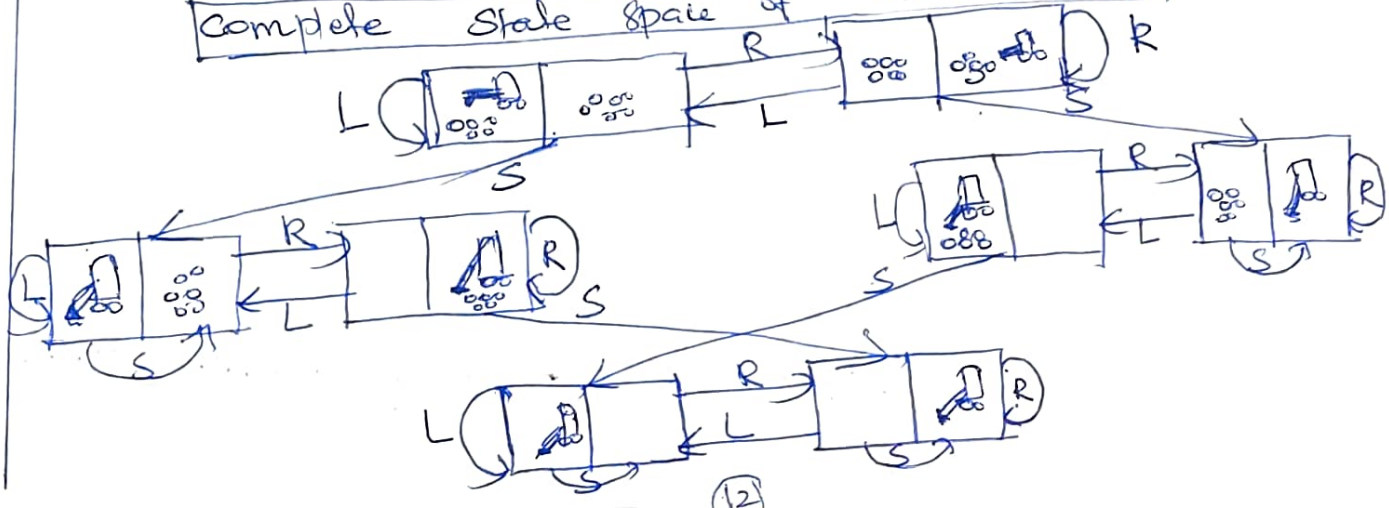
Successor function : Each state has 3 actions
(or)
Action

Left, Right and Suck

Goal test : It checks whether all the squares are clean

Path cost : Each step cost 1, so the path cost is the no. of steps in the path

Complete State Space of Vacuum world



ii) 8-puzzle problem

8 puzzle problem consists of 3×3 board with 8 numbered tiles and a blank space

A tile adjacent to the blank space can slide into the space

States: State description specifies the location of each of the eight tiles and the blank is one of the nine squares.

Initial state - Any state can be designated as the initial state.

Successor function - generates legal states from four actions such as blank moves left, Right, up and down

Goal test: Check whether the state matches the goal configuration

Path cost: Each step cost 1, so the path cost is the no. of steps in the path

Initial State

2	8	3
1	6	4
7		5

Final State

	1	2
3	4	5
6	7	8

(iii) 8 queens problem

The goal of the 8 queens problem is to place 8 queens on a chessboard such that no queen attacks each other.

ie no queen should be placed in the same row or column or diagonal

States: Any arrangement of 1 to 8 Queens on the board

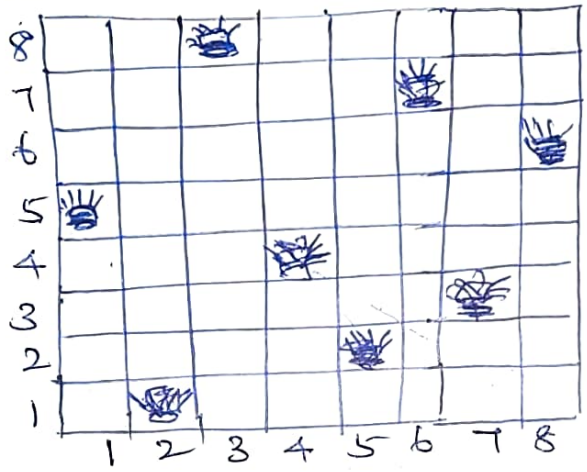
Initial State: No Queens on the board

Successor function: Add a queen to any empty square

Goal test: 8 queens on the board & none attacked
check

Path cost: ~~Zero~~ (search cost only exists)
Each step costs 1, so the path cost is the no. of steps in the path.

8 Queen Solution



(IV) Water Jug problem

- Given two jugs: 4 gallon and 3 gallon one.
- Neither has any measuring marker on it.
- There is a pump which is used to fill the jug with water

Question is: How can you get exactly 2 gallons of water from the 4 gallon jug.

Explicit Assumptions: A jug can be filled from the pump, water can be poured from one jug to another & that there are no other measuring devices available.

Initial State (0,0)

Goal State (2, n) for any value of n

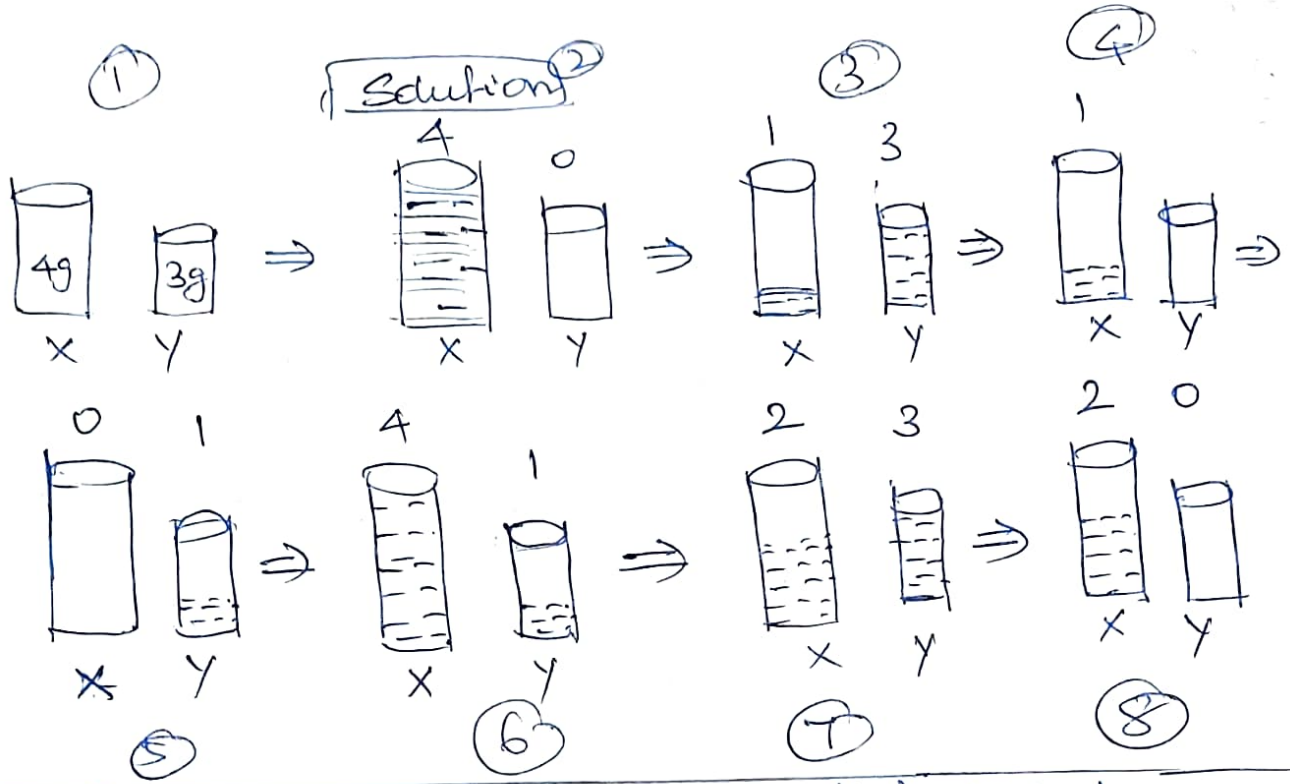
State : State of this problem is a tuple (x,y)

- where x represents the amount of water in the 4 gallon jug
- Y represents the amount of water in the 3 gallon jug.

i $0 \le x \le 4, 0 \le y \le 3$

operators - defined here are tabulated below

Sl no	Current State	Next State	Descriptions
1	(x,y) if $x<4$	(4,y)	Fill the 4 gallon jug
2	(x,y) if $y<3$	(x,3)	Fill the 3 gallon jug
3	(x,y) if $x>0$	(x-d, y)	pour some water out of 4 gallon jug
4	(x,y) if $y>0$	(x, y-d)	Pour some water out of 3 gallon jug
5	(x,y) if $y>0$	(0,y)	Empty the 4 gallon jug
6	(x,y) if $x>0$	(x,0)	Empty the 3 gallon jug
7	(x,y) if $x+y>=4$ & $y>0$	(4, y-(4-x))	Pour water from 3 gallon jug into 4 gallon jug until 4 gallon jug is full
8	(x,y) if $x+y>=3$ & $x>0$	(x-(3-y), 3)	Pour water from 4 gallon jug into 3 gallon jug until it is full
9	(x,y) if $x+y<=4$ & $y>0$	(x+y, 0)	Pour all water from 3 gallon jug to 4 gallon jug
10	(x,y) if $x+y<=3$ & $x>0$	(0, x+y)	pour all water from 4 gallon jug into 3 gallon jug
11	(0,2)	(2,0)	pour 2 gallon from 3 gallon jug into 4 gallon jug
12	(2,y)	(0,y)	Empty the 2 gallon in the 4 gallon jug on the ground



S.No	4 Gallon jug	3 Gallon jug	Rule Applied
1	0	0	Initial State
2	4	0	Fill 4
3	1	3	Pour 4 into 3 to full
4	1	0	Empty 3
5	0	1	Pour all of 4 into 3
6	4	1	Fill 4
7	2	3	Pour 4 into 3

(v) Crypt arithmetic problem

In crypt arithmetic problem letters stand for digits and the aim is to find the substitution of digits for letters such that the resulting sum is arithmetically correct and each letter stands for a different digit

- Rules

1) There should be no more than 10 distinct characters
2) The summation can not be too long
3) There must be a one to one mapping between letters and digits
4) The left most letter can't be zero in any word

States: A crypt arithmetic puzzle with some letters replaced by digits

Initial State: No digit is assigned to letters

Successor function: Replace all occurences of a letter with the digit not already appearing in the puzzle

Goal test: Puzzle contains only digits and represents a correct sum

Path cost: ~~zero~~ Each step costs 1, so the path cost is the number of steps in the path.

Example 1

```
  S E N D
+ M O R E
---------
M O N E Y
```

Solution: S=9, E=5, N=6, D=7, M=1, O=0 R=8, Y=2

Example 2:

```
  F O R T Y
+   T E N
+   T E N
---------
S I X T Y
```

Solution: F=2, O=9, R=7, T=8, Y=6, E=5, N=0

Real world problems

1) Airline travel problem:

State: Each is represented by a location (ie an airport) and the current time

Successor fn: Returns the state resulting from any scheduled flight

Goal test: check whether the destination is reached within the pre specified time

Path cost: Depends on various factors such as monetary cost, wait time, flight time, seat quality, type of airplane & so on.

2) Travelling Salesperson problem (TSP):

It is a touring problem in which each city must be visited exactly once. The aim is to find the shortest tour

3) Robot navigation

It means the robot's ability to determine its own position in its frame of reference and then to plan a path towards some goal location

It involves 1) Self localization, 2) Path planning
3) Map building and map interpretation

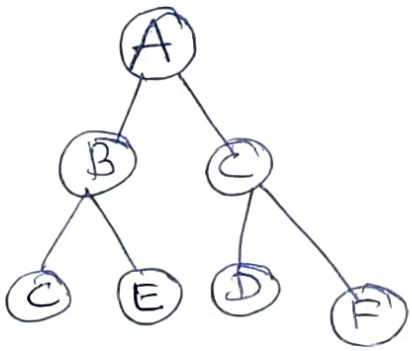
4) Automatic assembly sequencing of complex objects by a robot. The aim is to find an order in which to assemble the parts of some object.

UNIT 2

Problem Solving methods - Search Strategies - Uninformed - Informed - Heuristics - Local Search Algorithms and optimization problems - Searching with partial observations - Constraint Satisfaction problems - Constraint Propagation - Backtracking Search - Game playing - Optimal Decisions in Games - Alpha-Beta Pruning - Stochastic games

2.1 Problem Solving by Search

Search techniques use an explicit search tree which is generated by the initial state and the successor function together called as state space. State space form a graph or tree in which nodes are states and the arcs between them are actions



Search Tree

State - is a representation of problem element at a given moment

State Space - set of all states reachable from the initial state.

Terminology

Search Node: The root of search tree that is the initial state of the problem

Search Strategy:

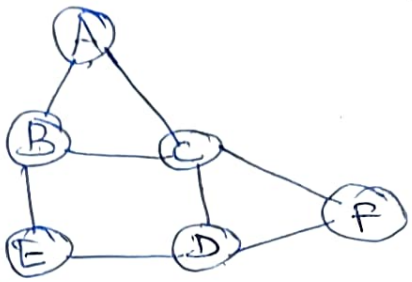
The choice of which state to expand first is determine by search strategy

Search tree: The tree which is constructed for the search process over the search space

Path - Any sequence of actions leading from one state to another

Eg: Tree Search Algorithm

A-Chennai
B-Covai
E-Kollam
C-Trychy
D-Madurai
F-Tirunelveli



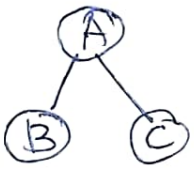
Task: Find a path from A to reach F

1. Start the sequence with the initial state and check whether it is a goal state or not
2) a) If it is a goal state return success
   b) otherwise perform the following steps
3) Generate and expand the new set of states from the initial state

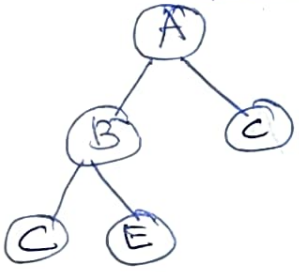
[The collection of nodes that have been generated but not expanded is called Fringe]

Initial State

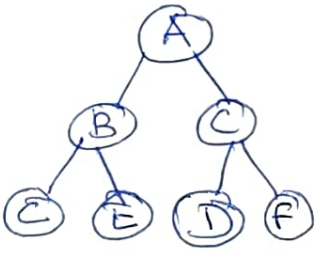
Expanding A



Expanding B



Expanding C



So the sequence of steps to reach the goal state F from A is: **A-C-F**

4) Repeat generation, testing of goal state & expanding until a solution is found (or) until there are no more states to be expanded

Measuring problem Solving performance

The search strategy algorithms are evaluated based on the following 4 criteria

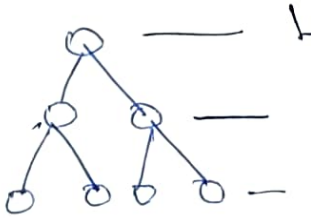
(i) Completeness: Whether the Strategy guaranteed to find the solution or not

(ii) Time complexity: Time taken to run a solution

(iii) Space complexity: Memory needed to perform the Search

(iv) Optimality: Best solution. Selection method

Branching factor (b): The no. of nodes which is connected to each of the node in the search tree. It is used to find time & space complexity of Search Strategy.



2.2 Problem Solving ~~Search~~ Strategy By Searching

Two categories

1. Uninformed Search Algorithms (Blind Search)
2. Informed Search Algorithms (Heuristic Search)

Uninformed Search Algorithms

* 6 uninformed Search Algorithm
1. Breadth First Search
2. Uniform cost Search
3. Depth First Search
4. Depth limited search
5. Iterative deeping DFS
6. Bidirectional search

Informed Search Algorithms

- 3 uninformed Search algorithms
1. Best First Search
2. Greedy Search
3. A* Search.

Informed Vs Uninformed Search

Informed Search	Uninformed Search
1. It uses knowledge for the searching process	It doesn't use knowledge for searching process
2. It contains information on goal State	It donot have additional information
3. It finds solution more quickly	It finds solution slow as compared to informed Search
4. It may or may not be complete	It is always complete
5. It provides direction regarding the solution	No suggestion is given regarding the solution
6. Eg. Greedy search, A* Search, Graph Search	Eg: Depth First Search; Breadth First Search

Uninformed (or) blind search means they have no information about the no. of steps or path cost from the current state to goal state

Informed (or) heuristic Search means they have additional information about the path cost from current state to goal State

2.3 Uninformed Search Algorithms

Breadth First Search (BFS)

* It is the most common search strategy for traversing a tree (or) graph. This algorithm searches breadthwise so it is called breadth-first search

* It starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level

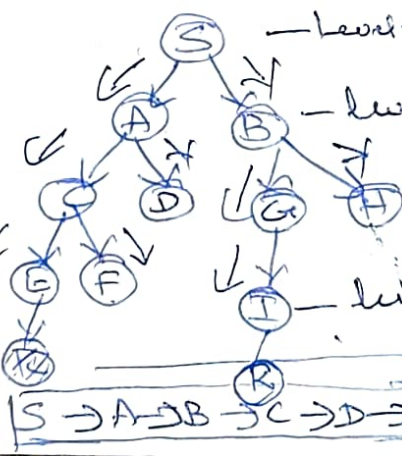
* It is implemented using FIFO-queue data structure

Advantages

1) BFS will provide a solution if any solution exists
2) If more than one solution exist, BFS will provide the minimal solution. ie solution which requires least no. of steps

Disadvantages

1) It requires lot of memory space
2) It needs a lot of time if the solution is far away from the root node



S → A → B → C → D → G → H → E → F → I → K

b - branching factor

① Time complexity : It is obtained by no of nodes traversed in BFS until the shallowest node, [d = depth]

$T(b) = 1 + b + b^2 + b^3 + \cdots + b^d = O(b^d)$

② Space complexity : $O(b^d)$ b - node at every state

③ Completeness : BFS is complete

④ Optimality : optimal soln is possible

2. Depth First Search (DFS)

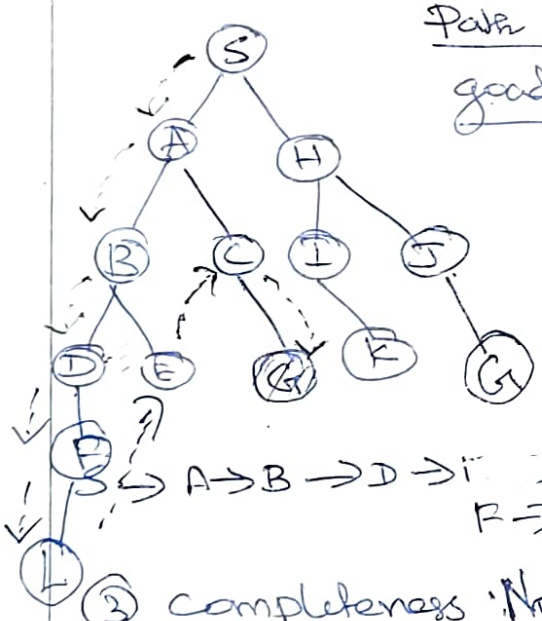
* It is a recursive algorithm for traversing a tree(or) graph data structure
* It starts from the root node and follows each path to its greatest depth node before moving to the next path
* It uses stack data structure

Advantages:

1. DFS requires very less memory
2. It takes less time to reach the goal node than BFS

Disadvantages:

1. There is no guarantee of finding the solution
2. While goes for deep down searching, sometimes it may go to infinite loop



Path to reach goal node G1

S → A → B → D → F → L → E → C → G1

① Time complexity

$T(b) = 1 + b + b^2 + b^3 + \cdots + b^m = O(b^m)$

m – maximum depth of any node and this can be larger than d

② Space complexity: $O(bm)$

③ Completeness: Not complete within finite state space [In some cases]

④ Optimal: DFS is non-optimal [In some cases], it generates large no of steps (or) high cost to reach the goal state.

3. Depth Limited Search Algorithm:

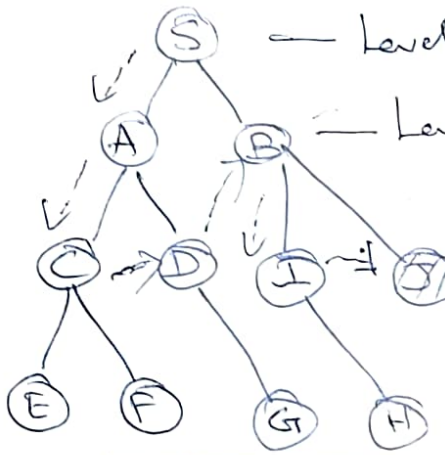
* It is similar to depth first search with a predetermined limit.
* It can solve the drawback of infinite path in the DFS

Advantages:

1) It is memory efficient

Disadvantages:

1) Early prediction of good depth limit
2) It may not be optimal



① Time complexity: $O(b^l)$

② Space complexity: $O(b \times l)$

③ Completeness: It is complete if the solution is above the depth limit (if $l > d$)

④ Optimal: It is not optimal even if $l > d$

$S \rightarrow A \rightarrow C \rightarrow D \rightarrow B \rightarrow I \rightarrow J$

4. Uniform Cost Search Algorithm

* It is used for traversing a weighted tree (or) graph
* The primary goal of the uniform cost search is to find a path to the goal node with lowest total cost.
* It is implemented by the priority queue

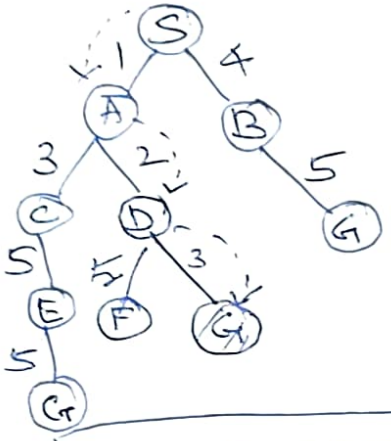
Advantages

1) It is optimal because path with least cost is chosen

Disadvantages:

1) As the algorithm concerned with path cost, this algorithm may be stuck in an infinite loop

Eg:



$S \to A \to D \to G$
$1 + 2 + 3 = 6$

① Time complexity

Let C^* is cost of optimal solution

ε - each step to get closure to the goal node

No. of Steps = $C^*/\varepsilon + 1$

$= C^*/\varepsilon$

[start from state 0]

Worst case Time complexity

$= O(b^{1+(C^*/\varepsilon)})$

② Space complexity:

Worst case Space complexity is $O(b^{1+[C^*/\varepsilon]})$

③ Completeness: Yes, complete

④ Optimal: Optimal, It selects a path with lowest path cost

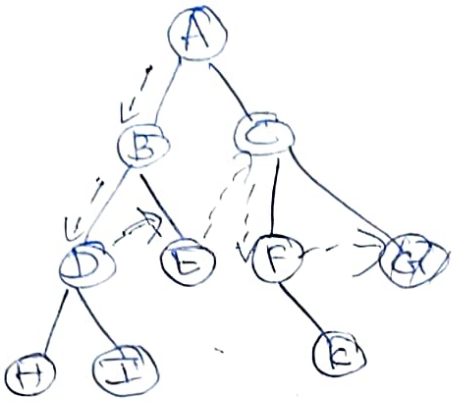
⑤ Iterative deepening DFS

* It is a combination of DFS and BFS
* It finds out the best depth limit and gradually increasing the limit until a Goal is found.

Advantages: ① It combines the benefits of both BFS & DFS

Disadvantages: It repeats all the work of the previous phase

Example: Find Goal State G from the following search tree



Limit 1 → A

Limit 2 → A → B → C

Limit 3 → A → B → D → E → C → F → G

Limit 4 → A → B → D → H → I → E

→ C → F → K → G

Goal State G is reached in Limit=3

① Time complexity: $O(b^d)$ - b - branching factor
d - depth

② Space complexity: $O(bd)$

③ Completeness: complete if the branching factor is finite

④ optimal: It is optimal

6. Bidirectional Search Algorithm

* It runs two simultaneous searches, one from initial state called forward search, and other from goal node called as backward search to find the Goal node.

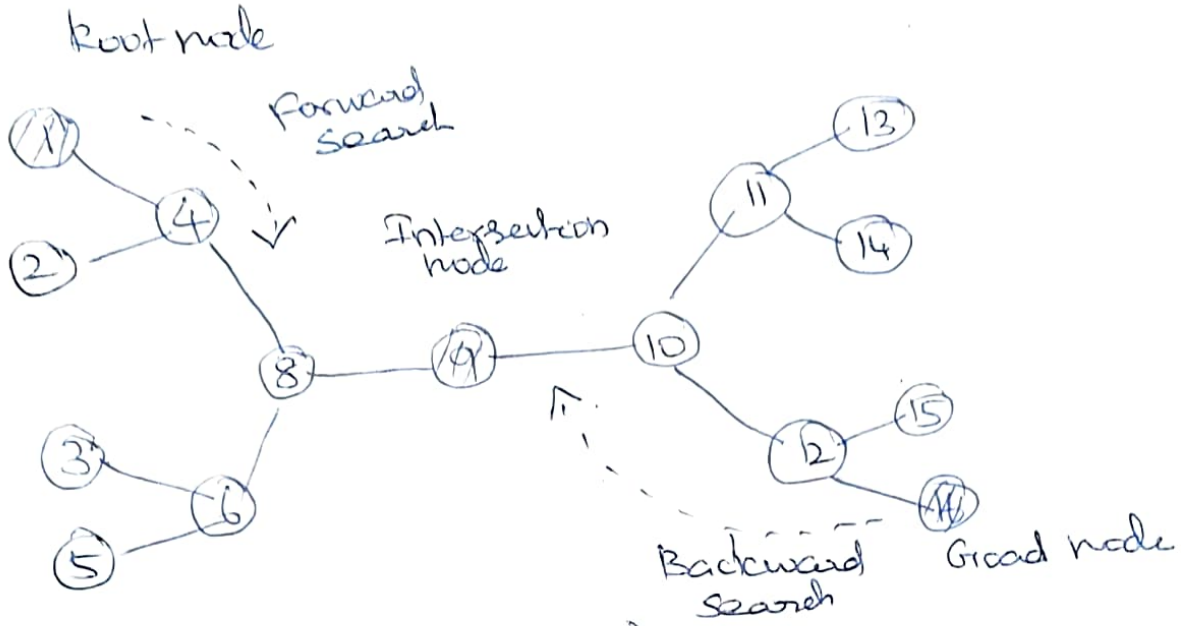
* The search stops when these two graphs intersect each other.

Advantages

1. It is fast
2. It requires less memory

Disadvantages

1. Implementation is difficult
2. One should know the goal state in advance.



① Time complexity: $O(b^d)$

② Space complexity: $O(b^d)$

③ Completeness : It is complete if we use BFS in both searches

④ optimal : It is optimal

2.4 Informed Search Algorithms

Informed Search Strategy uses problem specific knowledge to find solutions more efficiently

1. Best first search (Greedy Search)
2. A* Search

Best First Search

* In this search, a node is selected for expansion based on evaluation function $f(n)$

where $f(n)$ is cost estimate value

$f(n) = h(n)$

$h(n)$ = estimated cost of the cheapest path from the current node n to the goal node

* Best first search is known as Greedy search because it tries to expand node which is nearest to the goal node. Thus it evaluates node with the help of heuristic function h(n)

Best first search Algorithm

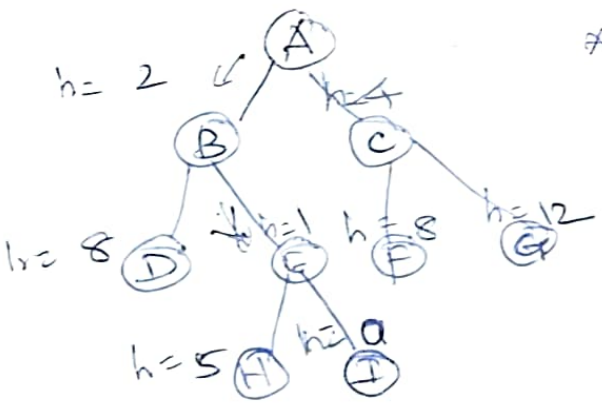
1) Set an OPEN list and an CLOSE list where OPEN list - contains visited but unexpanded nodes & CLOSE list - contains visited & expanded nodes.

2) Initially traverse the root node and visit the next successor node & place them in OPEN list in ascending order of their heuristic values

3) Select the first successor node from the OPEN list with minimum heuristic value & expand further.

4) Rearrange all the remaining unexpanded nodes in the OPEN list and repeat above 2 steps

5) If the goal node is reached, terminate the search, else expand further.



Path from A to I is A → B → E → I

* In this figure, root node is A and its next successor nodes are B and C with h(B)=2 & h(C)=4
* Choose node with lowest heuristic to expand further
* select node B & expand D & E. Then explore node with lowest heuristic i.e. node E.

① Time & Space complexity: $O(b^m)$, m - maximum depth of search tree

② completeness: It is incomplete

③ Optimality: It does not provide an optimal solution

Disadvantages

① It does not guarantees to reach the goal state

② It does not give an optimal solution

③ It may cover a long distance in some cases

A* Search Algorithm

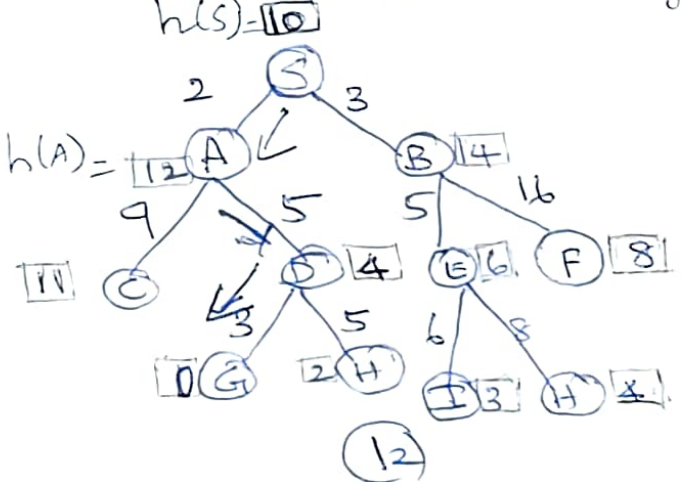
* A* search is the most widely used informed search algorithm
* A node is evaluated by combining the values of the functions g(n) and h(n)

g(n) is path cost from the start node to node n

h(n) is the estimated cost of the cheapest path from node n to the goal node

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

f(n) is the estimated cost of cheapest solution through n



In this example

S is the root node

G is the goal node

f(n) of node S is

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

$= 0 + 10 = 10$

$f(A) = g(A) + h(A)$, $f(B) = 3+14 = 17$

$= 2 + 12 = 14$

Therefore, node A has the lowest $f(n)$ & node A will be explored to its next level nodes C and D

$$f(C) = g(C) + h(C)$$
$$= 11 + 11 = 22$$

$$f(D) = g(D) + h(D)$$
$$= 7 + 4 = 11$$

So $f(D)$ is explored next as it has lowest $f(n)$ value

$$f(G) = g(g) + h(g)$$
$$= 10 + 0$$

$$f(H) = g(H) + h(H)$$
$$= 12 + 2 = 14$$

So the node G is selected with $f(n) = 10$

The sequence we follow is S → A → D → G

A* Search Algorithm — Steps

1. Place the starting node in the OPEN list
2. Check if the OPEN list is empty or not, if the list is empty return failure and stop
3. Select the node from the OPEN list which has the smallest value of evaluation function $(g+h)$, if node n is goal node then return success and stop otherwise
4. Expand node n and generate all of its successor and put n into the closed list. For each successor n' check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into OPEN list

Step
5. Else if node n' is already in OPEN and CLOSED then it should be attached to the back pointer which reflects the lowest g(n') value

6. Return to Step 2

Advantages:

1. It is the best algorithm than other search algorithm
2. A* search algorithm is optimal and complete
3. It can solve very complex problems

Disadvantages

1. It does not produce shortest path always
2. It has some complexity issues
3. More memory is needed

① Time complexity : $O(b^d)$

② Space complexity : $O(b^d)$

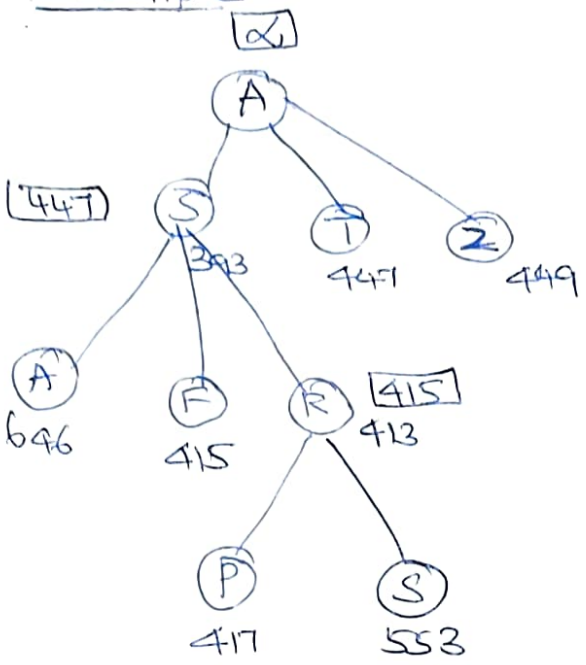
③ Completeness : It is complete as long as
   a) branching factor is finite
   b) cost at every action is fixed

④ Optimal : optimal if it follows the two condition
   a) Admissible & b) Consistency

Iterative deepening A* search (IDA*)

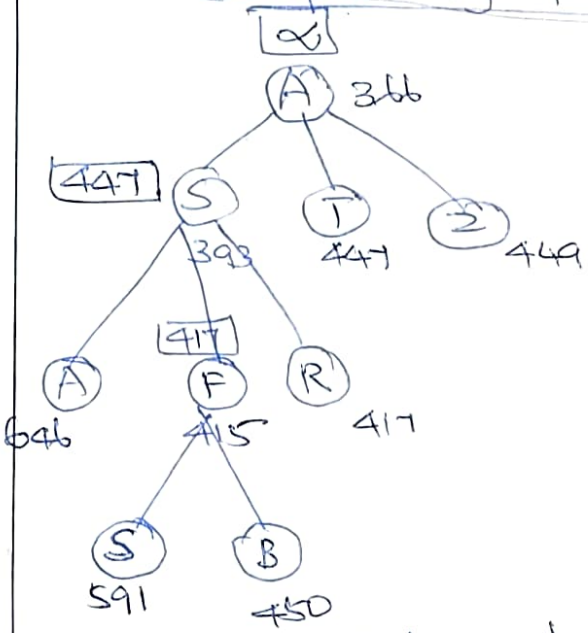
In this algorithm, memory requirements of A* is reduced by combining heuristic function with iterative deepening algorithm resulting in IDA* algorithm

Example



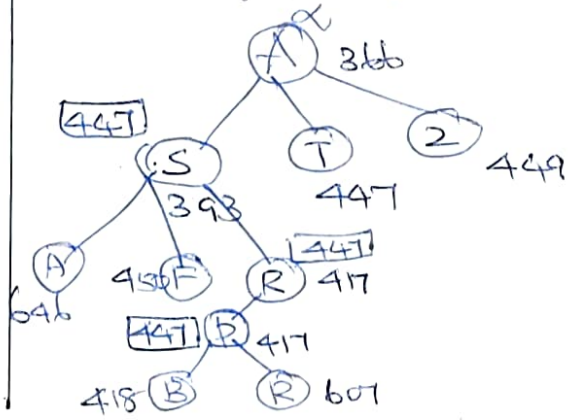
* After expanding A, S, R the current best leaf (P) has a value which is worse than the best alternative path (F)
* f_limit value is shown on top of each current node
* After expanding R, the condn f(best) > f_limit (417 > 415) is true and returns f[best] to that node

b) After unwinding back to & expanding F



* Here f[best] is 450 & which is greater than the f-limit of 417. Therefore it returns & unwinds with f[best] value to that node

c) After switching back to R & expanding P



* The best alternative path through T costs at least 447, therefore the path through R & P is considered as the best one.

* $f(n)$ limit is used rather than depth limit.

① Time complexity: Depends on the no of different values the heuristic function take

② Space complexity: It is proportional to the longest path of exploration i bd

③ Optimality: Yes

④ Completeness: yes

Disadvantage: It requires more storage space in complex domain

Memory bounded Algorithms are

1) Recursive Best First Search
2) Memory bounded A* Search

Recursive Best First Search

* It is a simple recursive algorithm similar to best first search but uses linear space
* It keeps track of f-values of best alternative path available from any ancestor of the current node

Advantage

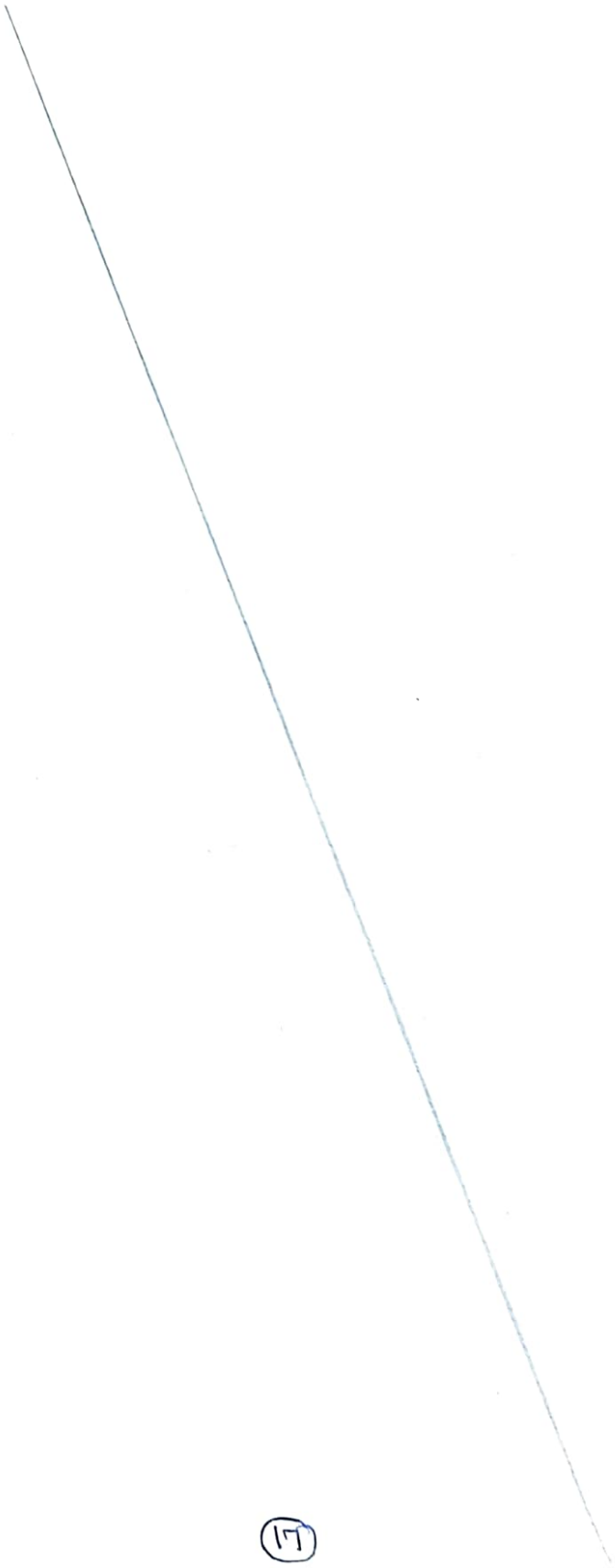
1) More effecient than IDA*
2) It is optimal if $h(n)$ is admissible
3) Space complexity is $O(bd)$

Disadvantages

1) It suffers from excessive nod Generation

Completeness: It is complete, provided the available memory is sufficient to store the shallowest solution path

Optimal : optimal



Simplified Memory bounded A* Search (SMA*)

SMA* algorithm can make use of all available memory to carry out the search.

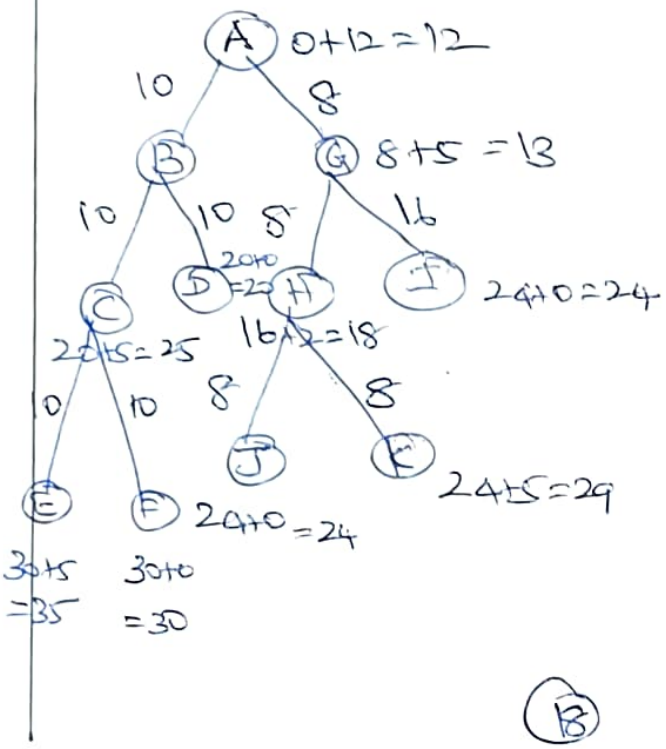
Adv! SMA* uses only the available memory

Disadv! If enough memory is not available it leads to unoptimal solution

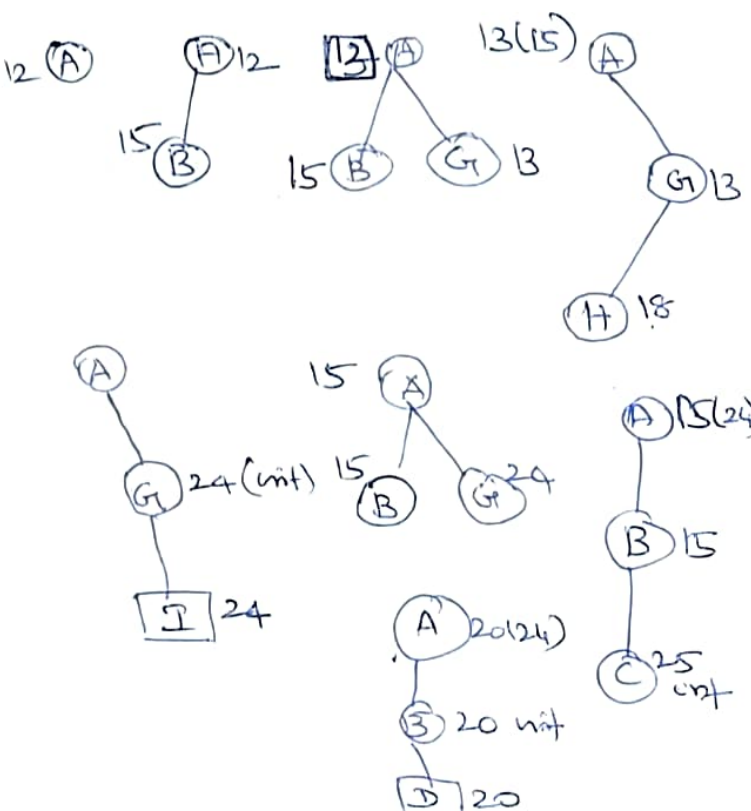
Algorithm

1) Expand deepest lowest f-cost leaf node
   - Best first search on f-cost
2) Update f-cost of nodes whose successors have higher f-cost
3) Drop shallowest & highest f-cost leaf node
   - Remember best forgotten descendant
4) Paths longer than node limit get ∞ cost

Example



SMA* Example - 3 node limit



2.5 Heuristic functions: - Information required to solve a given problem more efficiently

It is a function which ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow

* Selection of a good heuristic function is very important

* A good heuristic function is determined by its efficiency

* Simple toy problems such as 8 puzzle, 8-queen, tic-tac-toe etc can be solved more efficiently with the help of a heuristic function

8-puzzle problem

Consider the following 8-puzzle problem with a start state and a Goal state

1	2	3
8	6	
7	5	4

Start State

1	2	3
8		4
7	6	5

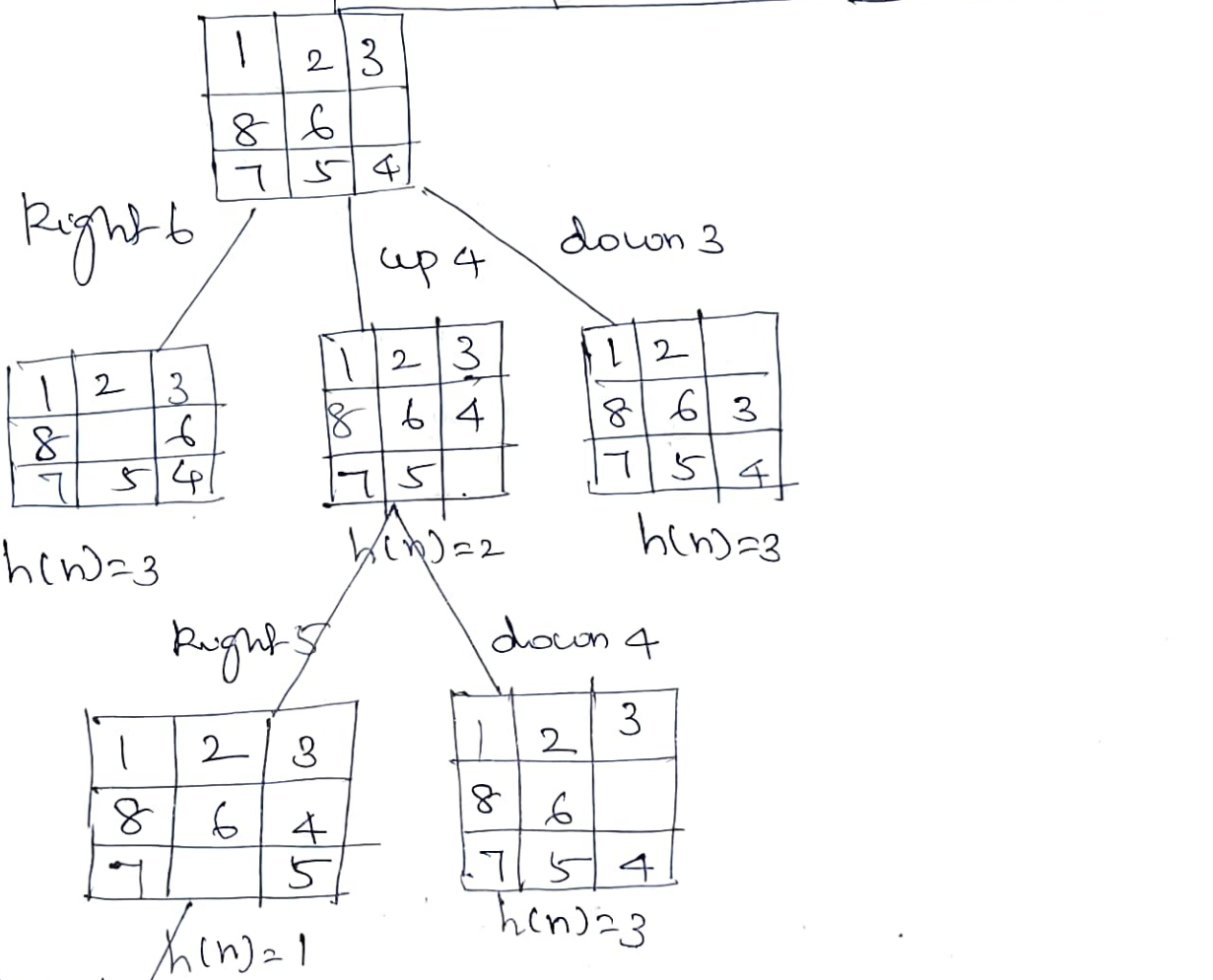
Goal State

* Our task is to slide the tiles of the current/start state & to reach the Goal state
* There can be 4 moves - Left, Right, up & down
* Heuristic function $h(n)$ is used to solve the problem efficiently

$h(n)$ = Number of tiles out of position

* Totally 3 tiles out of position ie 6, 5, 4
  So $h(n) = 3$
* But we require to minimize the value of $h(n) = 0$ to reach the goal state

State space tree of 8-puzzle



* It is seen from the State space tree that goal state is minimized from $h(n) = 3$ to $h(n) = 0$

Properties of Heuristic Search Algorithm

1. Admissible condition
2. Completeness
3. Dominance property
4. Optimality property

2.6 Local Search Algorithms and optimization problem

Local Search Algorithms

* Local Search algorithms focus only on solution state needed to reach the goal node and does not consider the path cost.

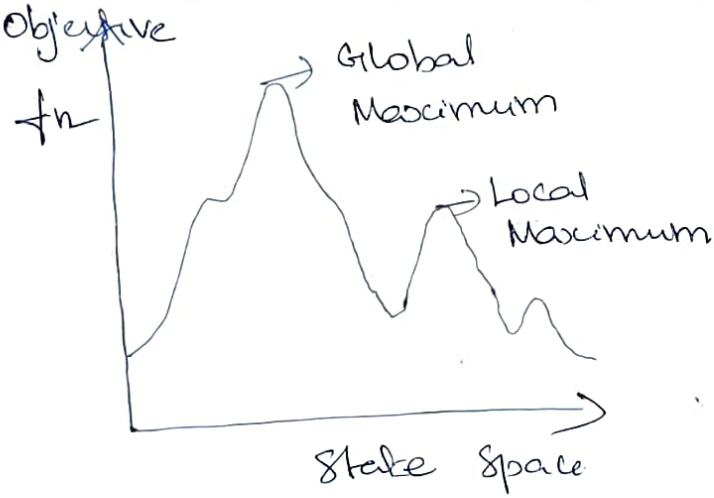
* They operate only on a single current state and move only to neighbors of that state

[Paths followed by the search are not retained]

Advantages

1) They use very little memory
2) They can often find reasonable solutions in large or infinite state spaces for which systematic algorithms are unsuitable.

Example: Landscape having both location & Elevation



Location – It is defined by the state

Elevation – It is defined by the value of the objective function

* The local search algorithm explores the above landscape by finding either
Global Minimum (elevation means cost) or
Global Maximum (finds the highest peak)

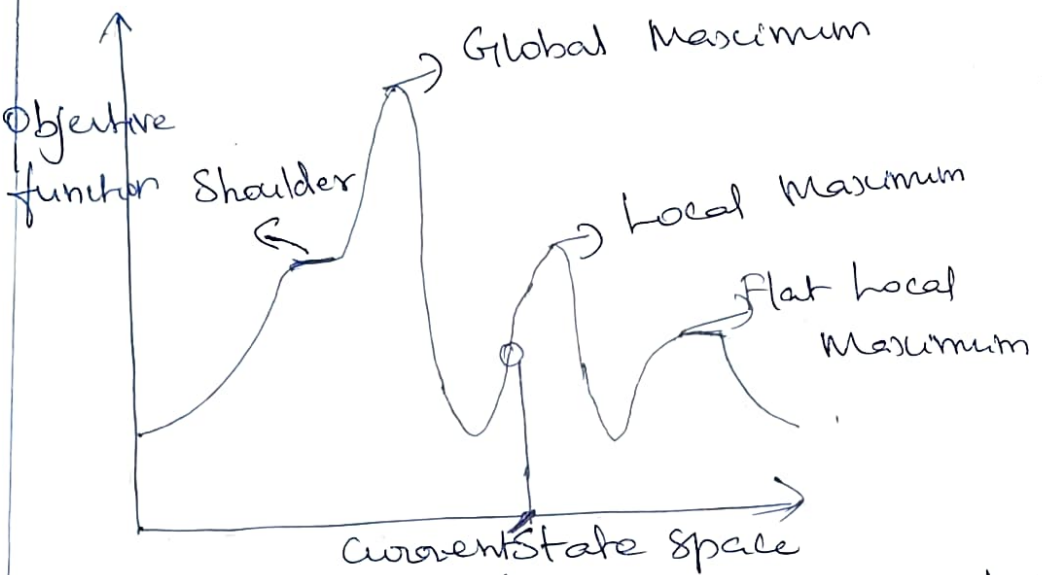
Different types of local search Algorithms

1) Hill climbing search
2) Simulated Annealing
3) Local Beam Search

Hill climbing Algorithm

* It is a local search problem
* The purpose of the hill climbing search is to climb a hill and reach the topmost point of that hill
* It is based on heuristic search technique to estimate the direction which lead to the highest peak

State Space Landscape of Hill climbing algorithm



Shoulder - It is also a flat area where the summit is possible

Global Maximum - Highest point on the hill which is the Goal State

Local maximum - It is the peak higher than all other peaks but lower than Global Maximum

Flat Local Maximum - It is the flat area over the hill where it has no uphill or downhill

Types of Hill climbing Search Algorithms

1) Simple Hill climbing
2) Steepest Ascent hill climbing
3) Stochastic hill climbing
4) Random restart hill climbing

1) Simple hill climbing

* It is the simplest technique to climb a hill
* The task is to reach the highest peak of the mountain
* This search focus only on the previous and next step (or) move

Algorithm

1. Create a CURRENT node, NEIGHBOUR node and a GOAL node
2. If the CURRENT node = GOAL node, return GOAL & terminate the search
3) Else CURRENT node <= NEIGHBOUR node, move ahead
4. Loop until the goal is not reached (or) a point is not found

2) Steepest Ascent Hill climbing

* It considers all the successive nodes, compares them and choose the node which is closest to the solution.
* It is similar to best first search because it focuses on each node instead of one.

Algorithm

1. Create a CURRENT node and a GOAL node.
2. If the CURRENT node = GOAL node, return GOAL and terminate the search
3. Loop until a better node is not found to reach the solution
4. If there is any better successor node present, expand it
5. When the GOAL is reached, return GOAL & terminate

3) Stochastic Hill climbing

It does not focus on all the nodes. It selects one node at random and decides whether it should be expanded (or) search for better one

4) Random-restart hill climbing

It iteratively searches the node and selects the best one at each step until the goal is not found.

* It conducts a series of hill climbing searches from randomly generated initial state, stopping when a goal is found

Hill climbing drawbacks

① Local Maximum: It is a peak which is higher than each of its neighboring state, but lower than global maximum

*To avoid this state, random node is selected using backtracking

② Plateau(or) Shoulder: It is an area of the state Space landscape where the evaluation function is flat

* To avoid this state, random node is selected or skip the level ie select the node in the next level

③ Ridges: Ridges result in a sequence of local maxima ie very difficult for greedy algorithms to navigate

Simulated Annealing Search

It is an algorithm which combines hill climbing with random walk to yield both efficiency and completeness.

In metallurgy, annealing is the process used to temper (or) harden metals & glass by heating them to a high temperature and then gradually cooling them

* When the search stops at the local maxima, allow "bad" moves to escape the local maximaa

* The node is selected randomly & it checks whether it is a best move or not

* ΔE – variable to calculate the probability of worsened

T– To determine the probability

function Simulated Annealing (problem, schedule)

returns a solution state

inputs: problem, schedule

local variables: current, a node next, a node T

current ← Make-node (Initial State [Problem])

for t ← 1 to ∞ do

T ← Schedule[t]

if T=0 then return current

next ← a randomly selected successor of curren

ΔE ← Value[Next] – Value[current]

if $\Delta E > 0$ then current ← Next

else current ← next only with probability $e^{\Delta E/T}$

Applications

1. VLSI Layout
2. Airline Scheduling

Local beam Search

* Local beam Search is a variation of beam search & is a path based algorithm
* It uses k states and generates successors for k states in parallel instead of one state.
* The useful information is passed among the k parallel threads

The sequence of Steps to perform local beam Search

1) Keep track of k states rather than just one
2) Start with k randomly generated states
3) At each iteration, all the successors of all k states are generated
4) If anyone is a goal state Stop; else select the k best successors from the complete list & repeat

Optimization problems

In addition to finding goals, local search algorithms are useful for solving pure optimization problems, in which the aim is to find the best state according to the objective function.

An optimal algorithm always finds a global minimum / Maximum

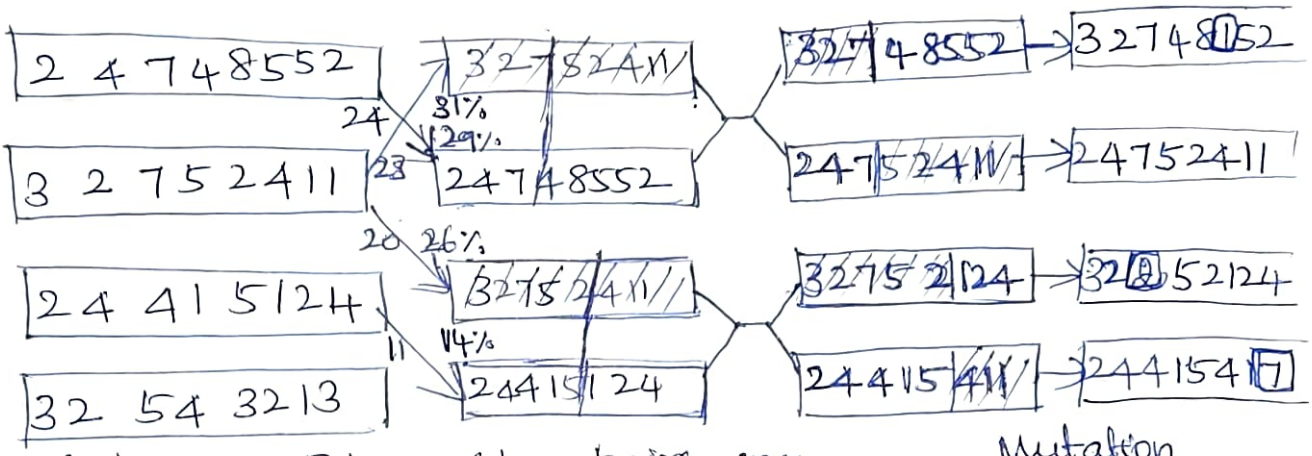
Genetic Algorithms (GA)

It is a varient of stochastic beam search in which successor states are generated by combining two parent states, rather than by modifying a single state.

* GA begin with a set of k randomly generated states called **population**

* Each state is represented as a string over a finite alphabet - ie a strn of 0s & 1s.

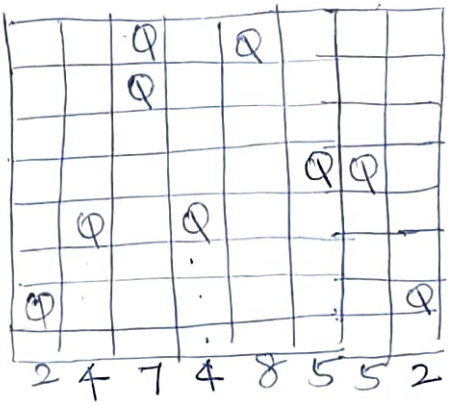
For eg an 8×8 Queen state must specify the position of 8 Queen, each in a column of 8 squares, so requires $8 \times \log_2 8 = 24$ bits



(a) Initial Selection — Fitness Selection — pairs — cross over — Mutation

Initial population

Eg



Initial population

2 4 7 4 8 5 5 2

3 2 7 5 2 4 1 1

2 4 4 1 5 1 2 4

3 2 5 4 3 2 1 3

Evaluation function (or) fitness function:

A function that returns higher values for better state

Eg: For 8 Queens problem - no of non attacking pairs of Queens is defined as fitness function

Fitness values of 4 states are: 24, 23, 20 and 11

The probability of being chosen for reproduction is directly proportional to the fitness score

24/(24+23+20+11) = 31% 20/(24+23+20+11) = 26%

23/(24+23+20+11) = 29% 11/(24+23+20+11) = 14%

Selection: A random choice of two pair is selected for reproduction by considering the probability of fitness function of each state.

Cross over: Each pair to be mated, a cross over point is randomly chosen. For the first pair, the crossover point is chosen after 3 digits and after 5 digits for the second pair

First pair

327	52411
247	48552

Second pair

32752	411
24415	124

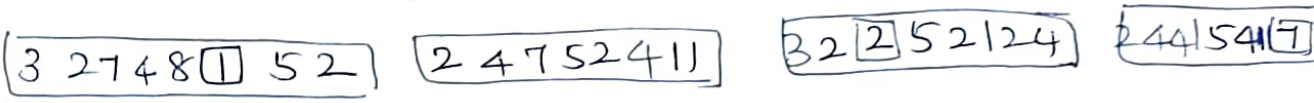
Offspring: Offspring is created by crossover the parent strings in the crossover point

327	52411
247	48552

327 48552

247 52411

Mutation: Each location is subject to random mutation with a small independent probability. One digit was mutated in the first, third & 4th offspring.



The Sequence of steps to perform GA

1) Start with k randomly generated states population
2) Each state is represented as a string over a finite alphabet
3) Evaluation function is applied to find better states with higher values
4) Produce the next generation of states by selection, crossover and mutation.

2.7 Searching with partial Observations

When the knowledge of the states is incomplete about the environment, only partial information is known to the agent. This incompleteness lead to 3 distinct problem types. They are

(i) Sensorless problems: If the agent has no sensors at all, then it could be in one of several possible initial states, and each action might lead to one of possible successor states.

* A Constraint Satisfaction problem is defined by a set of variables $X_1, X_2, \ldots X_n$ and a set of constraints $C_1, C_2, \ldots C_m$.

* Each Variable X_i has a nonempty domain D of possible values

* A State of the problem is defined by an assignment of values to some (or) all of the variables $\{X_i = V_i, X_j = V_j \ldots\}$

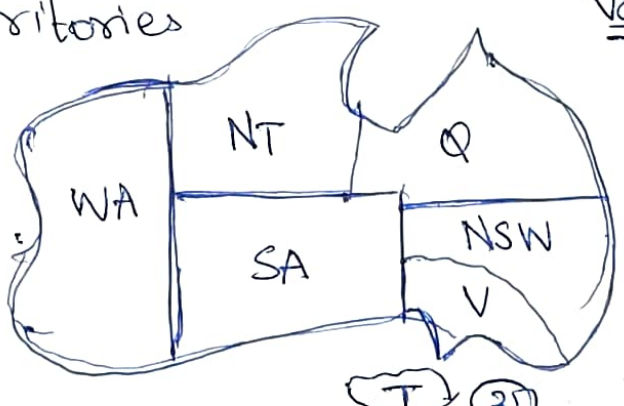
* An assignment that does not violate any constraints is called consistant (or) legal assignment

* A solution to a CSP is a complete assignment that satisfies all the constraints

* Some CSPs require a solution which maximizes an objective function

Example for CSP - Coloring of Australia Map

Consider Australia Map - Showing its states & territories



Variables - WA, NT, Q, NSW, V, SA, T

Domain $D_i = \{Red, Green, Blue\}$

Constraint: Adjacent region must have different colors

ii) Contingency problem

If the environment is partially observable or if actions are uncertain, then the agents percepts provide new information after each action. To handle the situation of unknown circumstances the agent needs a contigency plan

iii) Exploration problem:

It is an extreme case of contigency problem where the states and actions of the environment are unknown and the agent must act to discover them

2.8 Constraint Satisfaction problems (CSP)

Constraint satisfaction problems are mathematical problems where one must find states (or) objects that satisfy a no of constraints or criteria.

A constraint is a restriction of the feasible solution in an optimization problem.

Examples

1) N-Queens problem
2) A crossword puzzle
3) A map coloring problem
4) Boolean satisfiability problem
5) Cryptarithmetic problem

Given task is to color each region either red, green or blue in such a way that the neighboring regions have the same color

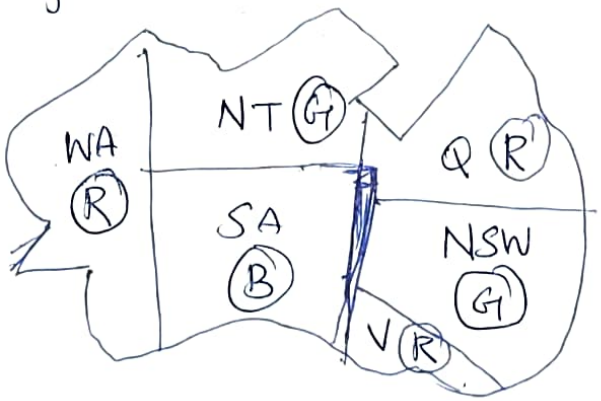
Problem formulation

* Defining variables to represent the regions
  - Western Australia - WA
  - Northern Territory - NT
  - Queensland - Q
  - New South Wales - NSW
  - Victoria - V
  - South Australia - SA
* Domain of each variable is the set {red, green, blue}
* Constraints require neighboring regions to have distinct colors.

Allowable combinations of WA & NT are: {(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)}

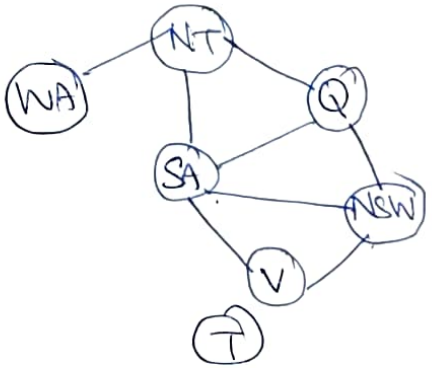
Map coloring problem

According to the above said constraints Australia map is colored as follows



R - Red
G - Green
B - Blue

Constraint Graph
- nodes are variables
- arcs are constraint



CSP can be viewed as a standard search problem

Initial State: the empty assignment {} in which all variables are unassigned

Successor function: A value can be assigned to any unassigned variable provided that it does not conflict with previously assigned variable

Goal test: The current assignment is complete

Path cost: A constant cost for every step

Real world CSPs: Assignment problem, Timetable problem, Floor planning, Transportation Scheduling, Factory scheduling

Crypt arithmetic problem

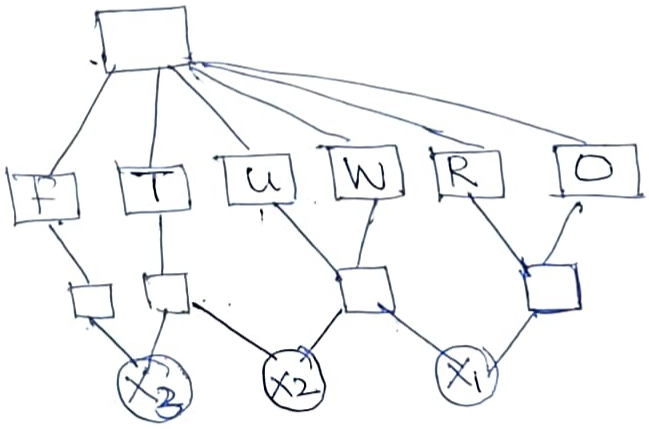
It is a type of CSP where the game is about digits and its unique replacement either with alphabets. The task is to substitute each digit with an alphabet to get the result arithmetically correct, with the added restriction that no leading zeroes are allowed

Eg.
```
  T W O
+ T W O
-------
F O U R
-------
```

Variables: F T U W R O
X_1 X_2 X_3

Domains: {0,1,2,3,4,5,6,7,8,9}

Constraints: Alldiff (F T U W R O)



Constraint hyper graph for crypt arithmetic problem

Where X_1, X_2 & X_3 are auxiliary variables representing the digit 0 or 1 carried over into the next column.

- $O+O = R + 10 \cdot X_1$
- $X_1+W+W = U+10 \cdot X_2$
- $X_2+T+T = O+10 \cdot X_3$
- $X_3 = F$, $T \neq 0$ $F \neq 0$

```
  T W O
+ T W O
-------
F O U R
```

- This problem has 7 solutions

1) 734 F=1 O=4 R=8 U=6 W=3
```
  734
  734
-----
 1468
```

2) 765 F=1 O=5 R=0 T=7 U=3 W=6
```
  765
  765
-----
 1530
```

3) Varieties of CSPs

1) Discrete variables can have
   Finite Domain - eg Map coloring problem
   Infinite Domain - eg set of integers (or) set of strings
2) continous domain - Linear programming problem

Varieties of constraints

1) Unary constraints - restricts a single variable (SA ≠ Greek)
2) Binary constraints - relates pairs of variable
3) Higher order constraints - 3 or more variables (SA ≠ WA)
   - cryptarithmetic puzzle

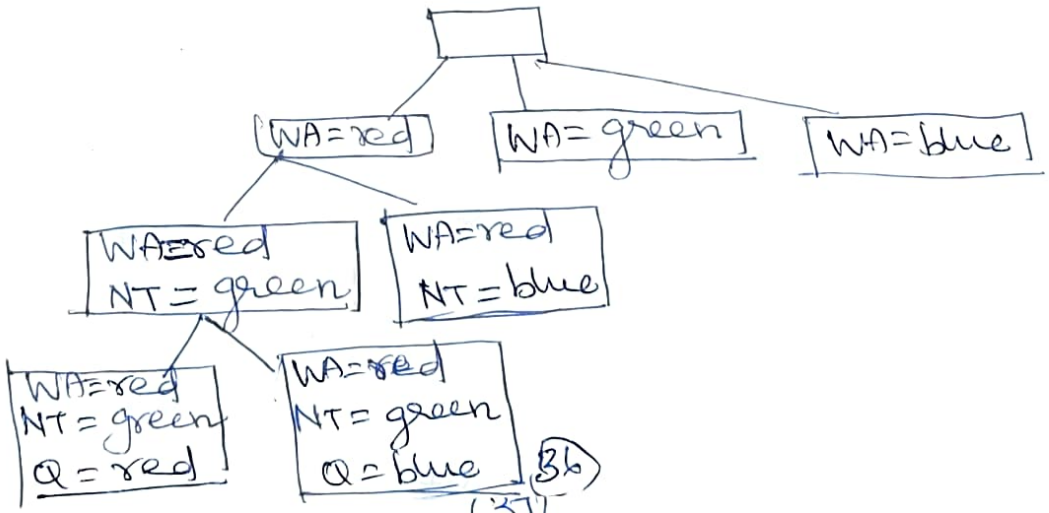
2.9 Backtracking Search for CSPs

* Depth first search for CSPs with single variable assignment is called backtracking search

* Backtracking search is the basic uninformed algorithm for CSP

```
function BACKTRACKING-SEARCH (csp) returns solution/failure
    return RECURSIVE-BACKTRACKING ({}, csp)

function RECURSIVE-BACKTRACKING (assignment, csp)
                                    returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE (VARIABLES[csp],
                                    assignment, csp)
    for each value in ORDER-DOMAIN-VALUES (
                    var, assignment, csp) do
        if value is consistent with assignment given
                                    CONSTRAINTS[csp] then
            add {var=value} to assignment
            if result ≠ failure then return result
            remove {var=value} from assignment
    return failure
```



Improving Backtracking Efficiency

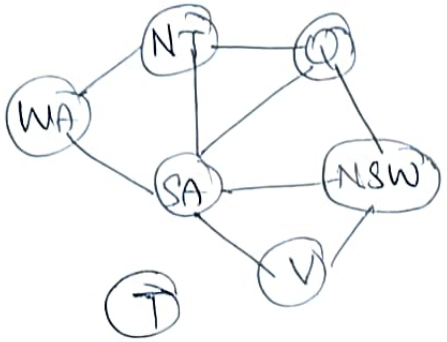
1) Variable & value ordering
   - Choose the most constrained variable
   - Select most constraining variable
     (Variable which has largest no of constraints on other unassigned variables)
   - Given a variable, choose the least constraining value

Constraint propagation

Although forward checking detects many inconsistencies, it does not detect all of them

Constraint propagation means propagating the implications of a constraint on one variable onto other variables

1) Arc consistency (two consistency)
   * One method of constraint propagation
   * Stronger than forward checking
   * Arc refers to a directed arc in the constraint graph



Consider two nodes SA & NSW
   * An arc is consistent if for every value x of SA there is some value y of NSW ie consistent with x
   * Examine arcs for consistency in both directions

Node consistency

- Simplest consistency technique
- The node representing a variable V in constraint graph is node consistent if for every value x in the current domain of V, each unary constraint on V is satisfied

K Consistency (path consistency)

A CSP is K-consistent if for any set of K-1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable.

2.10 Game playing: (Adversarial Search problems - Agents have conflicting goals - games)

It is an important domain of AI.

Games need knowledge in the form of rules, legal moves and the conditions of winning (or) losing the game. Both players try to win the game. So both try to make the best move possible at each turn. So these problems are known as adversarial Search problems

* The most common search technique in game playing is MiniMax Search

* It is depth first & depth limited search procedure

* It is used for games like chess and tic-tac-toe

Optimal Decisions in Games

Game problem formulation

A game with 2 players (MAX and MIN, MAX moves first, turn-taking) can be defined as a search problem with

Initial State: Board position

player: player to move either Max or Min

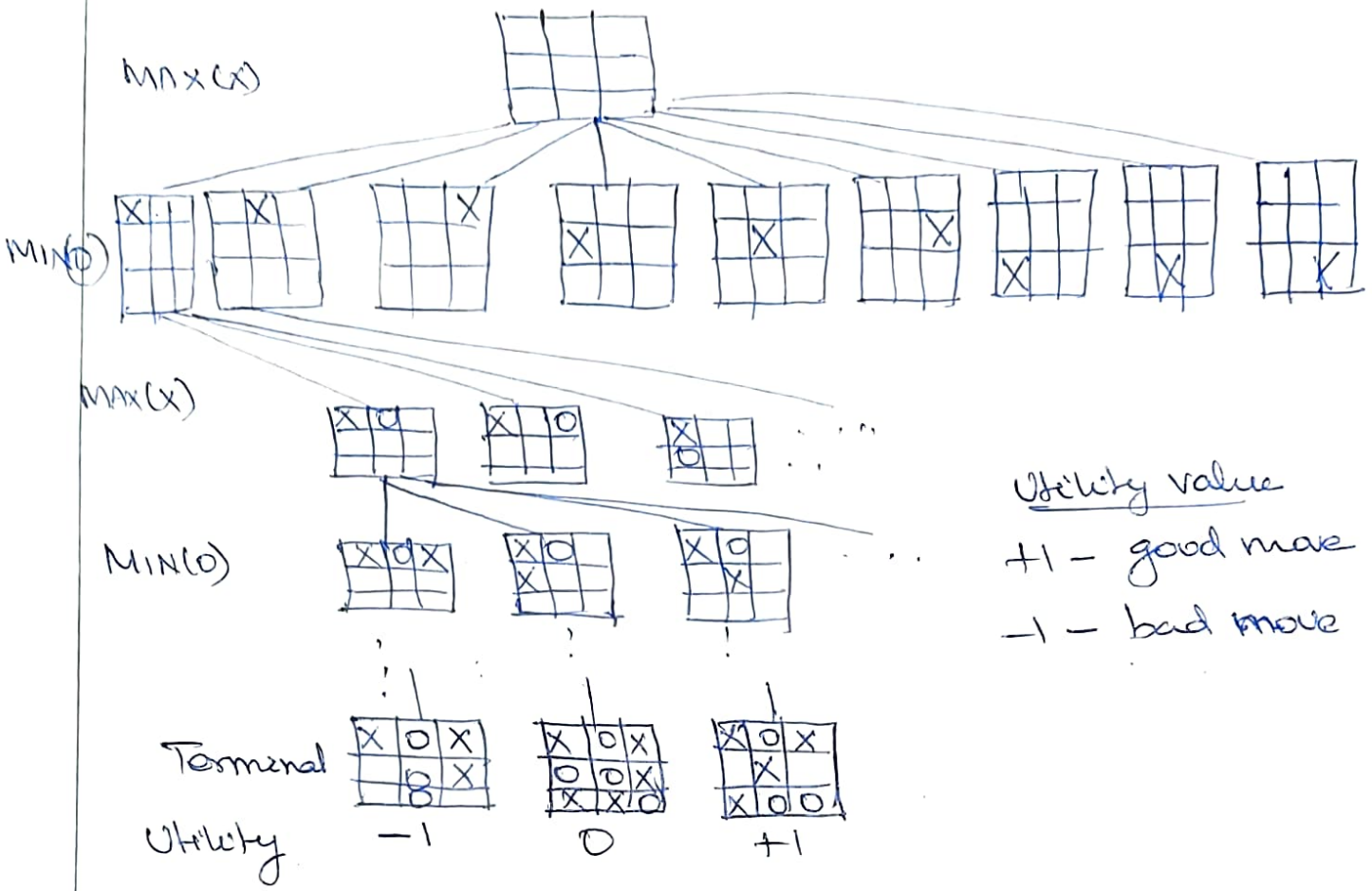
Successor function: a list of legal (move, state) pairs

Goal test: whether the game is over

Utility function: Gives a numeric value for the terminal states (win, loss, draw) +1 −1 0

Game tree = Initial State + legal moves.

Part of Game Tree for Tic-Tac-Toe (noughts and crosses)



In this game, player MAX place an X and MIN place a O until we reach leaf nodes corresponding to the terminal states

Then the terminal states are numbered with Utility value from MAX player point of view

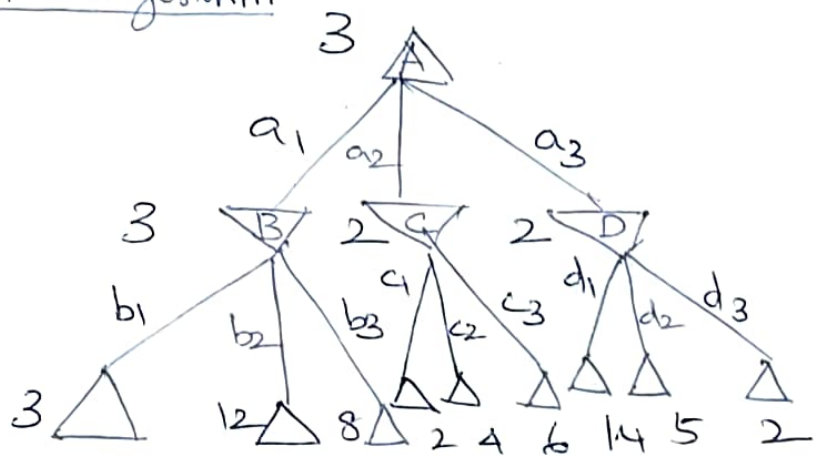
Optimal Decisions in Games

In game, MAX must find a contingent strategy which specifies, MAX's move in the initial state.. An optimal strategy leads to outcome atleast as good as any other strategy when one is playing an infallible opponent..

Given a game tree, the optimal strategy can be determined by examining the minimax value of each node MINIMAX VALUE(n)

$$\text{MINIMAX VALUE}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal node.} \\ \max_{s \in \text{Successors}(n)} \text{MAX-VALUES} & \text{if } n \text{ is MAX node} \\ \min_{s \in \text{Successors}(n)} \text{MIN-VALUES} & \text{if } n \text{ is a MIN node} \end{cases}$$

Minimax algorithm



* The Minimax algorithm computes the minimax decision from the current state.
* It uses a simple recursive computation of minimax values of each successor state.
* The recursion proceeds all the way down to the leaves of the tree, and then the minimax values are backed up through the tree as the recursion unwinds.

function MINIMAX-DECISION returns an action

inputs: state, a current state in game

$V \leftarrow$ MAX-VALUE (state)

return the action in SUCCESSOR (state) with value V

MAX-NODE

function MAX-VALUE (state) returns a utility value

if TERMINAL-TEST(state) then return UTILITY(state)

$V \leftarrow -\infty$

for a, s in SUCCESSORS(state) do

$V \leftarrow$ MAX(V, MIN-VALUE(s))

return V

MIN-NODE

function MIN-VALUE(state) returns a utility value

if TERMINAL-TEST(state) then returns UTILITY(state)

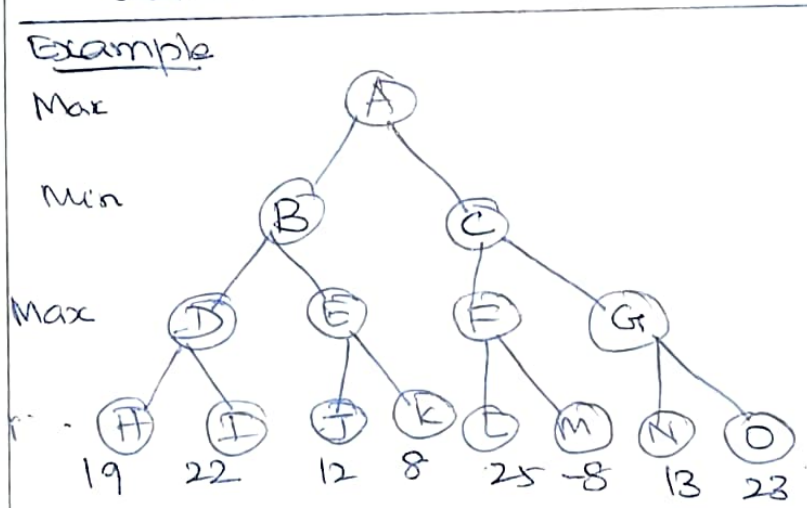
$V \leftarrow \infty$

for a, s in SUCCESSORS(state) do

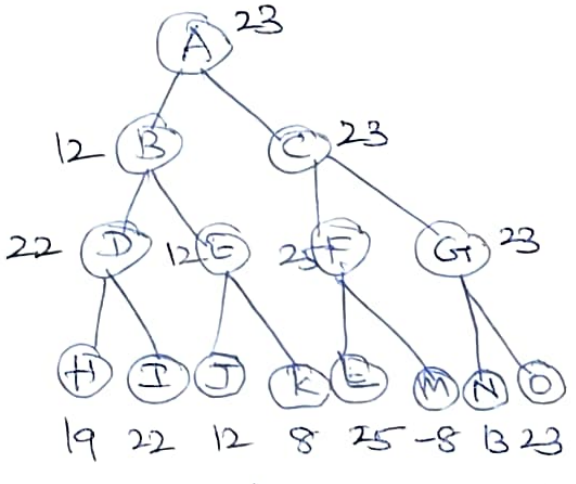
$V \leftarrow$ MIN(V, MAX-VALUES(s))

return V

Example



Before Backing up of values



After backing up of values

The minimax algorithm performs a complete depth first exploration of the game tree

Time complexity : $O(b^m)$

Space complexity : $O(bm)$

m - maximum depth of the tree, b - legal moves at each point

2.11) Alpha - Beta pruning

The problem with minimax search is that the number of game states has to examine is exponential. But we can reduce it with halt by performing Pruning.

Pruning - The process of eliminating a branch of the search tree from consideration without examining is called pruning

Two parameters of pruning are

Alpha (α) : Best choice for the value of MAX along the path (or) lower bound

Beta (β) : Best choice for the value of MIN along the path (or) upper bound

Alpha Beta search updates the values of α and β as it goes along and prunes the remaining branches at a node as soon as the value of current node is known to be worse than the current α and β value for MAX and MIN.

function ALPHA-BETA-SEARCH (State) returns an action

inputs: state, current state in game

$v \leftarrow$ MAX-VALUE(state, $-\infty$, $+\infty$)

return the action in SUCCESSORS(State) with value v

---

function MAX-VALUE (State, α, β) returns a utility value

inputs: State, current state in game

α, the value of best alternative for MAX

β, the value of best alternative for MIN

if TERMINAL-TEST(State) then return UTILITY(...)

$v \leftarrow -\infty$

for a, s in SUCCESSORS (State) do

$v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))

if $v \geq \beta$ then return v

$\alpha \leftarrow$ MAX(α, v)

return v

---

function MIN-VALUE (State, α, β) returns a utility value

inputs: State, current state in game

α, the value of the best alternative for MAX

β, the value of the best alternative for MIN

if TERMINAL-TEST (State) then return UTILITY (State)

$v \leftarrow +\infty$

for a, s in SUCCESSORS (State) do

$v \leftarrow$ MIN (v, MAX-VALUE(s, α, β))

if $v \leq \alpha$ then return v

$\beta \leftarrow$ MIN (β, v)

return v

Key points

1) Alpha: Initial value of alpha is $-\infty$

2) Beta: Initial value of Beta is $+\infty$

3) The condition for Alpha Beta pruning is $\alpha \geq \beta$

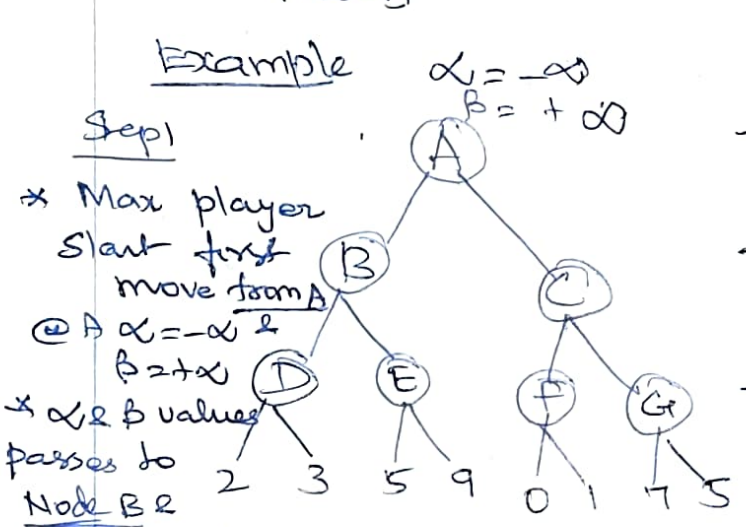
4) Each node has to keep track of its alpha and beta values. Alpha can be updated only when its MAX turn, beta can be updated only when its MIN's turn

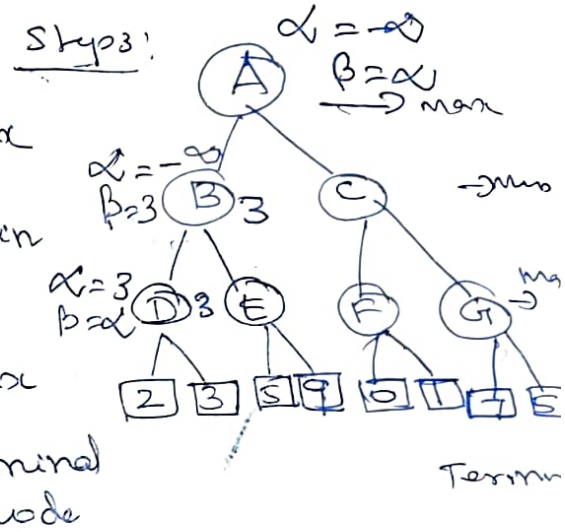
5) MAX will only update Alpha values and MIN " " " Beta values

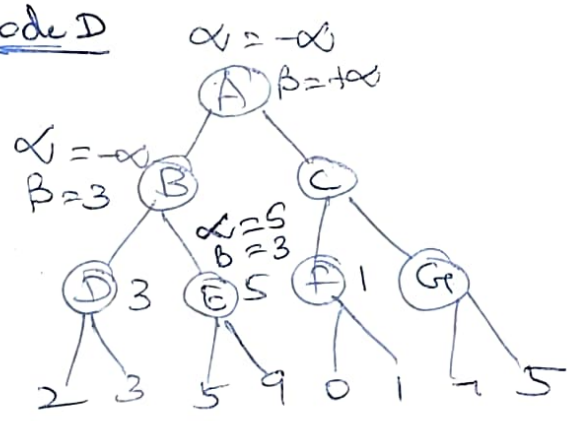
6) The node values will be passed to upper nodes

7) Alpha & Beta values only be passed to child nodes

Example







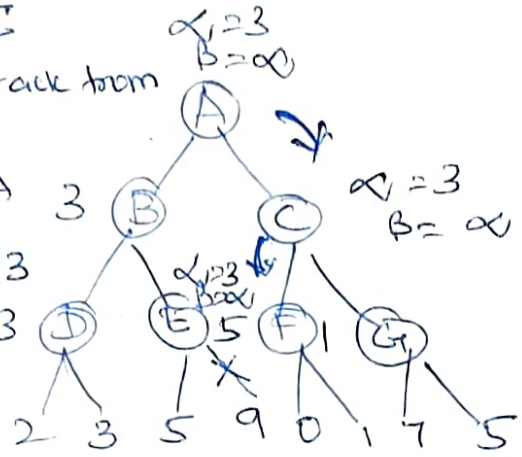
Now $\beta = +\infty$, will compare with the available subsequent nodes value i.e $\min(\infty, 3) = 3$, hence at node B $\alpha = -\infty$, $\beta = 3$

Step2 At node D, value of α will be calculated

* value of α is compared with 2 & then 3 & then $\max(2,3) = 3$ will be the value of α @ node D

Step3: Algorithm backtrack to node B, where the value of β will change as this is a turn of Min.

Step 5



* Backtrack from B to A.

At node A

α becomes 3 i.e Max(-∞,3)3

α=3

β=+∞

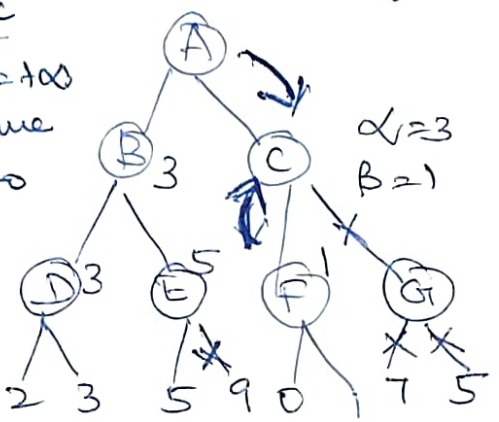
* These 2 values passes to right successor of A which is Node C

α=3, β=∞

At node C

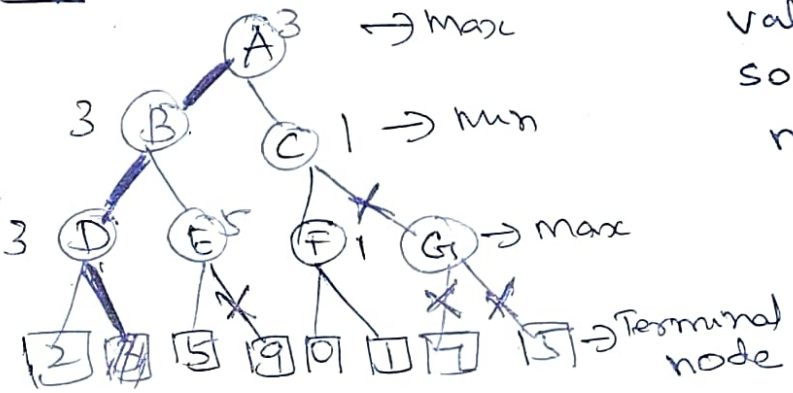
α=3 β=+∞

Same value passes to node F



Again it satisfies the condn α≥β, so the next child of C which is G will be pruned

Step 8



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α = -∞ & β = 3 will also be passed.

Step 4: At node E, Max turn. α will be updated

max(-∞,5) = 5

At node E α=5 & β=3 where α>=β

So right successor of E will be pruned, & value at E is 5

Step 6: At node F, value of α will be compared with left child which is 0, max(3,0)=3 & then compared with right child which is 1 & max(3,1)=3, Still α remains 3, but the node value of F will become 1

Step 7: Node F returns the node value 1 to node C.

at C α=3, β=+∞

β will be updated with 1 min(1,∞)=1

Now at C=α=3, β=1

C Now returns the value of 1 to A.

So best value for A is

max(3,1)=3

Hence the optimal value for the maximizer is

[3] for this example.

UNIT III Knowledge Representation

First order predicate Logic - Prolog Programming - Unification - Forward chaining - Backward chaining - Resolution - Knowledge representation - Ontological Engineering - categories and objects - Events - Mental events and mental objects - Reasoning systems for categories - Reasoning with Default Information

3.1 First order Logic

It is a logic which is sufficiently expressive to represent a good deal of our common sense knowledge. It is also called as First order predicate calculus FOL or FOPC

Representation of FOL.

FOL assumes the world contains

Objects: people, houses, colors, numbers etc

Relations: brother of, part of, bigger than

Functions: father of, best friend, plus...

[* are relations in which there is only one value for the given input]

Eg "one plus two equals three"

FOL: plus(one, two) = three

Objects: one, two, three

Relation: equals

Function: plus

Syntax and Semantics of FOL

FOL consists of Sentences and terms

Terms : represent objects

Constant symbols, Variables and functions are used to build terms

Sentences : Quantifiers and predicate symbols are used to build sentences

Model for FOL

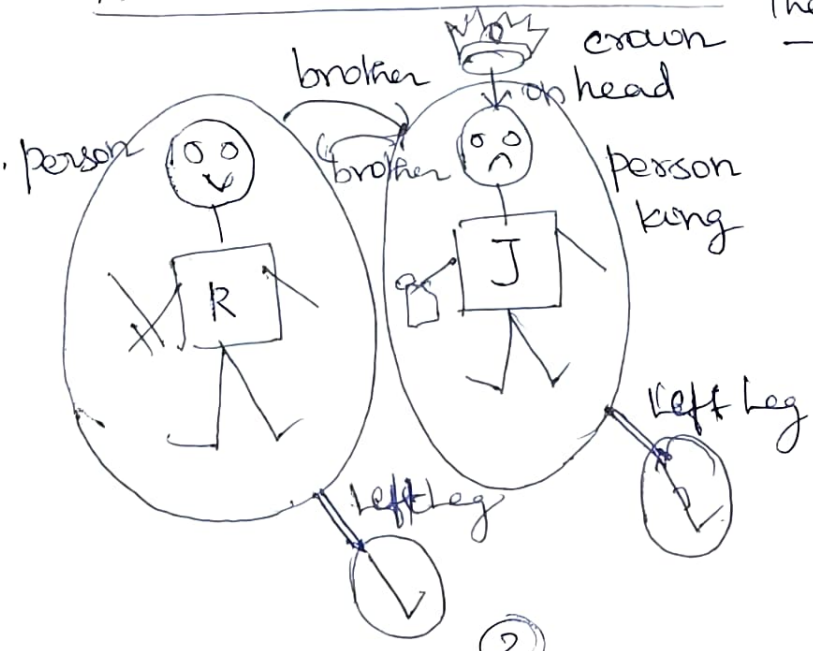
Model of FOL covers

* Domain : Set of objects (or) Domain elements
* Relation : Set of tuples of related objects
* Tuple : Collection of objects arranged in a fixed order & written using < >

Eg. Brotherhood relation in the given model

{<Richard the lionheart, king John>,
<king John, Richard the Lionheart>}

A model with 5 objects



The 5 objects are

1) Richard the Lionheart
2) The evil king John
3) The left legs of Richard & John
4) A crown

* Brotherhood relation in this model is

(<Richard the Lionheart, King John>, <King John, Richard the Lionheart>)

* On head relation – binary relation

onhead (the crown, King John)

* Each person has one left leg – unary relation

Left leg(King John)

Left leg(Richard the lionheart)

Symbols and Interpretation

The basic elements of FOL are the symbols that stand for objects, relations and functions

kinds of symbols

① Constant symbols represent objects eg. Richard

② Predicate symbols " Relations eg King

③ function symbols " functions eg (Left leg)

Syntax of first order Logic – in BNF

Sentences → Atomic sentence | complex sentence

Atomic Sentence → Predicate | Predicate(Term) | Term = Term

Complex Sentence → (Sentence) | ¬Sentence | Sentence ∧ Sentence | Sentence ∨ Sentence | Sentence ⇒ Sentence | Sentence ⇔ Sentence | Quantifier Variable ... Sentence

Term → Function (Term ...) | Constant | Variable

Quantifier → ∀ | ∃

Constant → A | X1 | John | ...

Variable → a | x | s | ...

predicate → True | False | After | Loves | raining

Function → Mother | LeftLeg | ...

operator precedence : ¬, =, ∧, ∨, ⇒, ⇔

Atomic Sentences: It is a combination of objects and predicate symbols

Eg. Brother (Richard, John)

Predicate ← → term

* Predicate symbol followed by a term is a atomic sentence

Complex Sentences : It uses logical connectives to form a complex sentences

Eg. King (Richard) ∧ king (John)

Quantifiers

* It is used to enumerate the objects by name

Eg. All kings are persons

∀x king (x) ⇒ person (x)

Two types of Quantifiers

1) Universal Quantifier : ∀x king (x) ⇒ person (x)

2) Existential Quantifier: ∃x,y Brother (x, Richard) ∧ Brother (y, Richard) ∧ ¬(x=y)

Equality (=)

This symbol is used to equate two terms if they refer the same object

eg. Father(John) = Henry

Both refer to same object

Nested Qualifiers

1. Brothers are Siblings

$\forall x \forall y$ Brother $(x,y) \Rightarrow$ Sibling (x,y)

2) Everybody loves somebody

$\forall x \exists y$ loves (x,y)

order of Quantification is important

Domain: A problem specific area

For eg. Map coloring problem,
cryptarithmetic problem

Assertions and Queries in FOL

Assertions : These are sentences that are added to knowledge base using TELL operation

Eg. TELL (KB, $\forall x$ king(x)) $\Rightarrow$ Person(x)

Queries

- Questions are asked using ASK operations

Eg. ASK (KB, Person(John))

Kinship Domain: Family relationship in FOL

* The objects in this domain are people
* Two unary predicates — Male and Female
* kinship relations — Parenthood, brotherhood are represented by binary predicates

Parent, Sibling, Brother, Sister, Child, Daughter.

* The functions are : Mother and Father

① one's mother is one's female parent

$\forall m, c \; Mother(c) = m \Leftrightarrow Female(m) \wedge Parent(m, c)$

$\forall m \forall c \; mother(m, c) \Leftrightarrow Female(m) \wedge$

② One's husband is one's male spouse

$\forall w, h \; Husband(h, w) \Leftrightarrow Male(h) \wedge Spouse(h, w)$

③ Male and female are disjoint categories

$\forall x \; Male(x) \Leftrightarrow \neg Female(x)$

④ Parent and child are inverse relations

$\forall p, c \; Parent(p, c) \Leftrightarrow Child(c, p)$

⑤ A grandparent is parent of one's parent

$\forall g, c \; Grandparent(g, c) \Leftrightarrow \exists p \; parent(g, p) \wedge parent(p, c)$

⑥ A sibling is another child of one's parent

$\forall x, y \; Sibling(x, y) \Leftrightarrow x \neq y \wedge \exists p \; parent(p, x) \wedge parent(p, y)$

Knowledge engineering in FOL

General process of knowledge base construction is called knowledge engineering

The Knowledge engineering process

1. Identify the task

The knowledge engineer must describe the range of questions & kinds of facts for each specific problem instance.

2. Assemble the relevant knowledge

The knowledge engineer need to work with real experts to extract relevant knowledge & to understand how the domain actually works
This process is called as Knowledge Acquisition

3. Decide on vocabulary of predicates, functions and constants

* This translates the important domain level concepts into logic-level names.
* Result of this process is a vocabulary of that domain

4. Encode general knowledge about the domain

The knowledge engineer writes down the axioms for all the vocabulary terms. & also finds the gaps in the vocabulary

5. Encode a description of the specific problem instance

* This involves writing simple atomic sentences about instances of concepts.

6 Pose queries to the inference procedure & get answers.

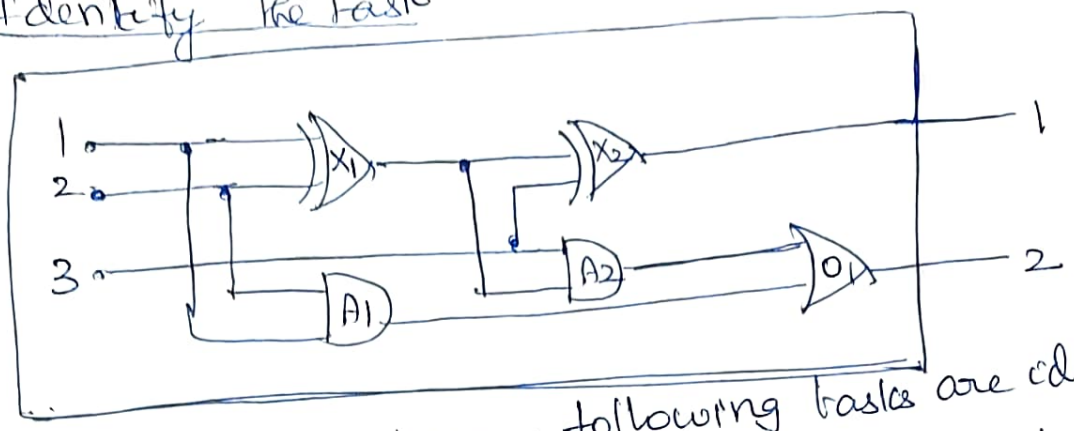
- The inference procedure operates on the axioms & problem specific facts to derive the facts

7. Debug the knowledge base

Missing or weak axioms are identified
Incorrect axioms can be identified

The Electronic circuit domain

1) Identify the task



During this phase, following tasks are identified
* does the circuit actually add properly
* what are the gates connected to the first input terminal?
* Does the circuit contains feedback loops?

2. Assemble the relevant knowledge

Knowledge about the electronic circuit such as signal flow from i/p terminal to output terminal, function of each gate & so on

3. Decide on Vocabulary

Knowledge engineer decide the functions, Predicates and constants to represent the fact

Eg. Representing the Gate: Gate(X_1), Type(X_1) = XOR

Predicate: circuit(C1), Terminal(x), connected(out(1,X_1), In(1,X_2))

Function: In(1,X_1) - First i/p terminal for Gate X_1

4) Encode General knowledge of the domain

* Rules / Axioms can be stated clearly and concisely.

Eg. The signal at every terminal is either 1 or 0
$\forall t$ Terminal(t) $\Rightarrow$ Signal(t) = 1 V Signal(t) = 0

Connected is Communicative:

$\forall t_1, t_2$ Connected$(t_1, t_2) \Leftrightarrow$ Connected(t_2, t_1)

5. Encode the specific problem issue

Encoding of the circuit C_1 shown in the figure.

(1) Categorize the circuit and its component gates

Circuit$(C_1) \wedge$ Arity$(C_1, 3, 2)$

Gate$(X_1) \wedge$ Type$(X_1) =$ XOR

Gate$(X_2) \wedge$ Type$(X_2) =$ XOR

Gate$(A_1) \wedge$ Type$(A_1) =$ AND

Gate$(A_2) \wedge$ Type$(A_2) =$ AND

Gate$(O_1) \wedge$ Type$(O_1) =$ OR

(2) Specify (or) Encode the connections

Connected$(Out(1, X_1), In(1, X_2))$

Connected$(Out(1, X_1), In(2, A_2))$

(6) Pose queries to the Inference procedure

Eg. What are the possible sets of values of all the terminals for the adder circuit?

$\exists i_1, i_2, i_3, O_1, O_2$ Signal$(In(1, C_1)) = i_1 \wedge$ Signal$(In(2, C_1)) = i_2 \wedge$ Signal$(In(3, C_1)) = i_3 \wedge$ Signal$(Out(1, C_1)) = O_1 \wedge$ Signal$(Out(2, C_1)) = O_2$

This final query will return a complete input-output table for the device it is for circuit verification

(7) Debug the knowledge base:

Final step of the knowledge Engg. This step find out the issues of knowledge Base.

3.2 PROLOG Programming

PROLOG - PROgramming in LOGIC

- Prolog is a logic programming language
- In Prolog, logic is expressed as Relations called as Facts & Rules
- Computation carried out by running a Query over the relations
- In Prolog facts are expressed in Definite patterns
- Facts contains objects and Relations

Eg

Facts

```
friends (raju, Mahesh)
singer (priya)
odd_number(5
```

Running Queries

1) Query 1 : ?- singer(priya). - our KB contains the fact
   output : yes So it returns 'yes'
2) Query 2 : ?- odd number(7). - our KB does not contains this fact
   output : No So it returns 'NO'

Key features

* Unification
* Backtracking
* Recursion

Advantages

1) Easy to build database. Doesn't need a lot of programming effort
2) Pattern matching is easy
3) It has built in list handling function

Disadvantages

1) LISP dominates over Prolog with respect to I/O features
2) Sometimes input and output is not easy

Application

1) Prolog is used in AI to build a database
2) It is used for pattern matching over Natural language parse tree

Sample program & Query window

facts → likes(john, jane). ← arguments / objects
likes(jane, john).
likes(jack, jane). → predicate name

Rule → friends(X,Y) :- likes(X,Y), likes(Y,X). ← variables
head body

Query window

```
2 - likes(john, jane).
true.
2 - friends(X, Y).
X = john,
Y = jane;
X = jane,
Y = john.
```

3.3 Unification

Generalized Modus Ponens rule (GMP)

For atomic sentences P_i, P_i' and q where there is a substitution θ such that

$$\text{SUBST}(\theta, P_i') \equiv \text{SUBST}(\theta, P_i)$$

$$\frac{P_1', P_2', \ldots P_n', (P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

GMP is a sound inference rule

$$\therefore P \models \text{SUBST}(\theta, P)$$

Any Sentence — Substitution

Ex:

$P_1' = \text{king}(\text{John})$ $\quad P_1 = \text{king}(x)$

$P_2' = \text{Greedy}(y)$ $\quad P_2 = \text{Greedy}(x)$

$\theta = \{x/\text{John}, y/\text{John}\}$ $\quad q : \text{Evil}(x)$

$\text{SUBST}(\theta, q) : \text{Evil}(\text{John})$

Unification: This algorithm find substitutions that make different logical sentences to look identical

- UNIFY algorithm takes 2 sentences and returns a unifier (substitution) for them exists!

(1) UNIFY(Knows(John, x), knows(John, Jane)) = $\{x/\text{Jane}\}$

(2) UNIFY(knows(John, x), knows(y, Bill)) = $\{x/\text{Bill}, y/x\}$

(3) UNIFY(knows(John, x), knows(y, mother(y))) = $\{y/\text{John}, x/\text{mother}\}$

(4) UNIFY(knows(John, x), knows(z, Elizabeth)) = Fail

Because 'x' cannot take two values at the same time.

Unification Algorithm

function UNIFY (x, y, θ) returns a substitution to make x and y identical

Inputs: x, a variable, constant, list, or compound expn
y, "
θ, the substitution build up

If θ = failure then return failure
else if $x = y$ then return θ
else if VARIABLE?(x) then return UNIFY_VAR(x, y, θ)
else if VARIABLE?(y) then return UNIFY_VAR(y, x, θ)
else if COMPOUND?(x) and COMPOUND?(y) then
returns UNIFY$(x.\text{ARGS}, y.\text{ARGS}, \text{UNIFY}(x.\text{OP}, y.\text{OP}, \theta))$
else if LIST?(x) and LIST?(y) then
return UNIFY$(x.\text{REST}, y.\text{REST}, \text{UNIFY}(x.\text{FIRST}, y.\text{FIRST}, \theta))$
else return failure

---

function UNIFY_VAR (var, x, θ) returns a substitution
if $\{var/val\} \in \theta$ then return UNIFY(val, x, θ)
else if $\{x/val\} \in \theta$ then return UNIFY(var, val, θ)
else if OCCUR_CHECK?(var, x) then return failure
else return add $\{var/x\}$ to θ

Standardizing apart

UNIFY(knows(John, x), knows(x, Elizabeth)) = FAIL

This unification fails because x cannot take 2 values at the same time. This problem can be avoided by standardizing apart. ie Rename the variables to avoid name clashes

Eg. Rename x in Knows(x, Elizabeth) to

x_{17} - new variable name

Now Unification works

UNIFY (Knows(John, x), knows(x_{17}, Elizabeth))

$= \{x / \text{Elizabeth}, x_{17} / \text{John}\}$

Occur check

When matching a variable against a complex term check whether the variable itself occur inside the term.

If it does, then the match fails

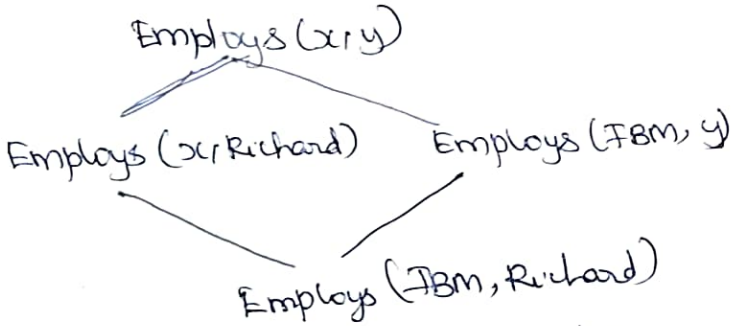
eg S(x) cannot unify with S(S(x))

Storage and Retrieval in KB

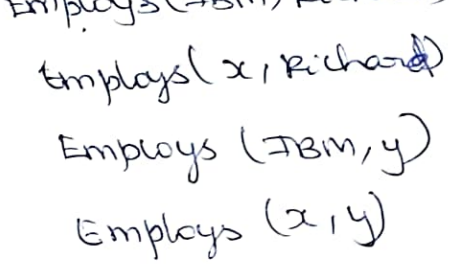
- TELL and ASK functions used to inform and interrogate a KB
- STORE(S): Stores a sentence 'S' into KB
- FETCH(q): returns all the unifiers such that the query 'q' unifies with some sentences in the KB

Predicate Indexing - means putting all facts in "buckets". These buckets can be stored in a hash table for efficient access

Subsumption Lattice



Queries forms a subsumption lattice



Properties of Subsumption lattice

* child node is obtained from the parent by a single substitution
* The highest common descendant of any two nodes is the result of applying their most general unifier
* For a predicate with n arguments, lattice contains $O(2^{n^2})$ nodes

3. Forward chaining

First order Definite clauses

A definite clause is either:

(1) an atomic literal (or)

(2) an implication whose antecedent is a conjunction of positive literals and who consequent is a positive literal

Eg:

1) like (John, Apple) ∧ eating (John, Apple) ⇒ happy (John)

2) Happy (John)

Forwaad chaining Algorithm

Idea:

1) Algorithm steals from the known facts

2) Apply Modus Ponens Inference rules in forward direction to extract more knowledge until goal is reached

function FOL-FC-ASK(KB, α) returns a substitution or false

inputs: KB, the knowledge base, a set of first order definite clauses

α, the query, an atomic sentence

local variables: new, the new sentence inferred on each iteration

repeat until new is empty

new ← { }

for each rule in KB do

$(P_1 \wedge \dots \wedge P_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES(Rule)

for each θ such that

SUBST(θ, $P_1 \wedge \dots \wedge P_n$) = SUBST(θ, $P_1' \wedge \dots \wedge P_n'$)

for some $p_1', \ldots, p_n'$ in KB

$q' \leftarrow$ SUBST(θ, q)

if q' does not unify with some sentence already in KB or new then

add q' to new

$\phi \leftarrow$ UNIFY(q', α)

if ϕ is not fail then return ϕ

add new to KB

return false

Properties of forward chaining

1) It is a down-up approach, move from bottom to top
2) It is a process of making a conclusion from known facts
3) It is called data driven approach
4) It is commonly used in Expert Systems

Example problem

The law says that it is a crime for an American to sell weapon to Hostile nations. All the missiles of the country Nano which is enemy of America were sold to it by Colonel West who is American

Task:

Rewrite the above paragraph as a definite clause & prove that

Colonelwest is Criminal

a) It is a crime for an American to sell weapon to Hostile nations.

① American(x) $\wedge$ weapon(y) $\wedge$ Sells(x, y, z) $\wedge$ Hostile(z) $\Rightarrow$ Criminal(x)

(b) Nano has some missiles

$\exists x \, owns(Nano, x) \wedge Missile(x)$

Apply existential elimination & Introduce New constant

(2) $owns(Nano, M_1)$ (3) $Missile(M_1)$

(c) All of its Missiles were sold to it by colonel West

(4) $Missile(x) \wedge owns(Nano, x) \Rightarrow Sells(West, x, Nano)$

(d) Missiles are weapons

(5) $Missile(x) \Rightarrow Weapon(x)$

(e) An enemy of America counts as "hostile"

(6) $Enemy(x, America) \Rightarrow Hostile(x)$

(f) West is an American

(7) $American(West)$

(g) The country Nano, an enemy of America

(8) $Enemy(Nano, America)$

Proof

Step 1: Start with the known facts.

$American(West)$ $Missile(M_1)$ $owns(Nano, M_1)$

$Enemy(Nano, America)$

Step 2:

Find those facts inferred from available facts and with satisfied premises.

Rule(1) - does not satisfy premises

Rule(2) and (3) are already added

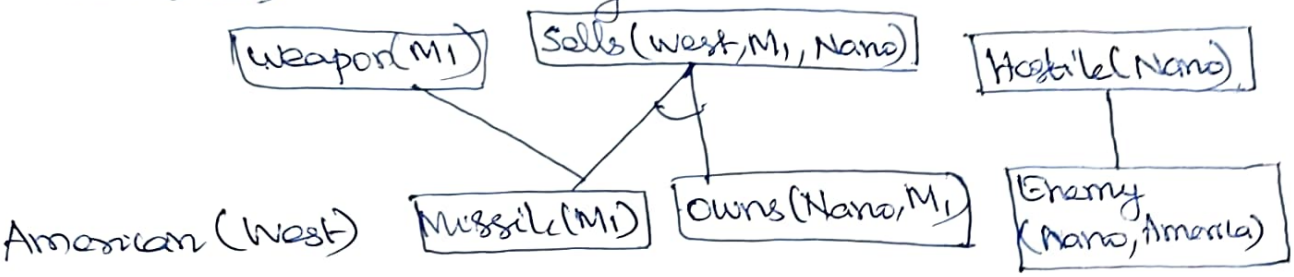
Rule(4) satisfy with the substitution $\{x/M_1\}$

$\therefore Sells(West, M_1, Nano)$ will be added

Rule (5) Satisfied with the substitution $\{x/M_1\}$ so Weapon(M_1) is added

Rule (6) Satisfied with the substitution $\{x/Nano\}$ and the conclusion Hostile(Nano) will be added

Rule (7) & (8) are already in known facts

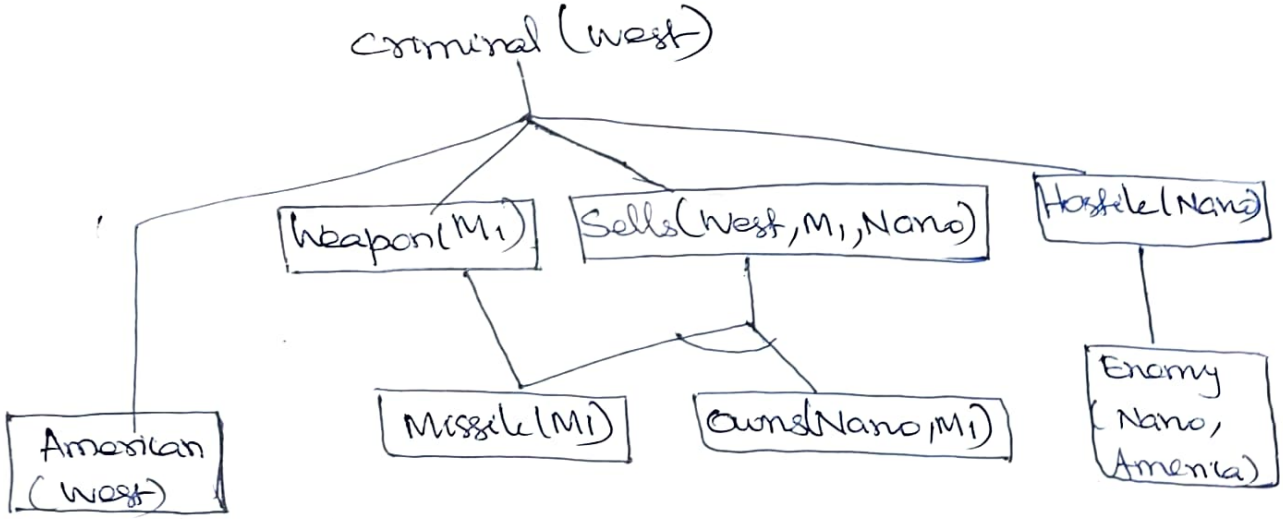


Step 3: Rule(1) is satisfied with substitution $\{x/west, y/M_1, z/Nano\}$ so that add

criminal (West)

Hence we reached our Goal Statement.

Proof tree generated by forward chaining



The initial facts appear at the bottom level, facts inferred on the first iteration in the middle level, facts inferred on the second iteration at the top level

Hence. It is proved that "West colonel is a criminal" using criminal (West)

3.5 Backward chaining

Backward chaining is also known as backward deduction method.

- It starts with the goal and work backward, chaining through rules to find known facts that support the goal.

Properties:

* It is a top down approach
* It is based on Modus Ponens Inference rule
* Goal is broken into subgoals to prove the facts true
* It is a goal driven approach, as a list of goals decide which rules are selected & used.
* It is used in Game-theory, Inference engine, Automated theorem proving etc.
* It is mostly used a depth first search strategy for proof

function FOL-BC-ASK(KB, @uery) returns a generator of Substitutions

return FOL-BC-OR(KB, @uery, {})

Generator FOL-BC-OR(KB, goal, θ) yields a Substitution

for each rule (lhs ⇒ rhs) in FETCH-RULES-FOR-GOAL (KB, goal) do

(lhs, rhs) ← STANDARDIZE-VARIABLES(lhs, rhs)

for each θ' in FOL-BC-AND(KB, lhs, UNIFY(rhs, goal, θ)) do

yield θ'

generator FOL-BC-AND (KB, goals, θ) yields a Substitution

if θ = failure then return

else if LENGTH (goals) = 0 then yield θ

else do

first, rest ← FIRST (goals), REST (goals)

for each θ' in FOL-BC-OR (KB, SUBST (θ, first), θ) do

for each θ'' in FOL-BC-AND (KB, rest, θ') do

yield θ''

Example - Use the same example & prove that colonelwest is criminal

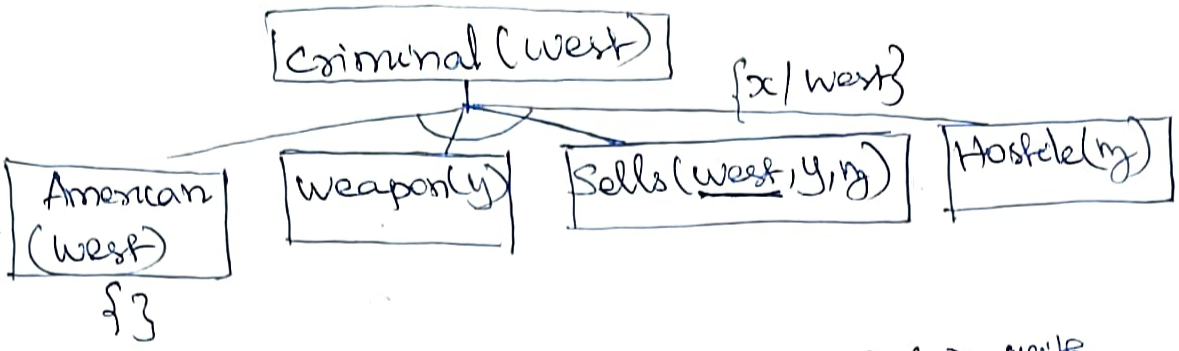
Rules:

1. American(x) ∧ weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) ⇒ Criminal(x)
2) Owns(Nano, M1)
3) Missile(M1)
4) Missile(x) ∧ Owns(Nano, x) ⇒ Sells(West, x, Nano)
5) Missile(x) ⇒ Weapon(x)
6). Enemy(x, America) ⇒ Hostile(x)
7). American(West)
8) Enemy(Nano, America)

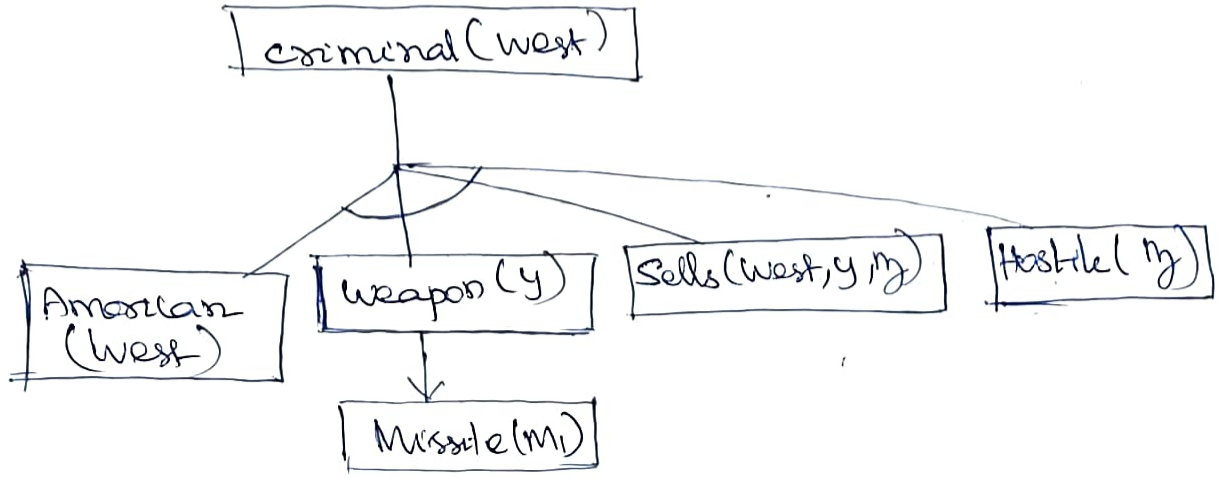
Step1: Start with goal fact & Prove all facts / inferred facts are true.

Criminal(West)

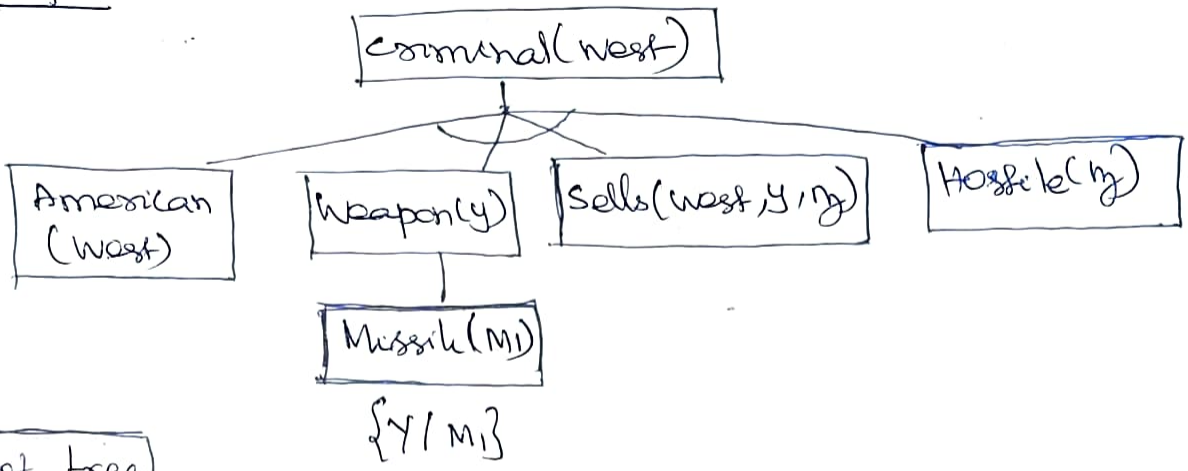
Step2: Infer facts from goal facts which satisfies the rules.



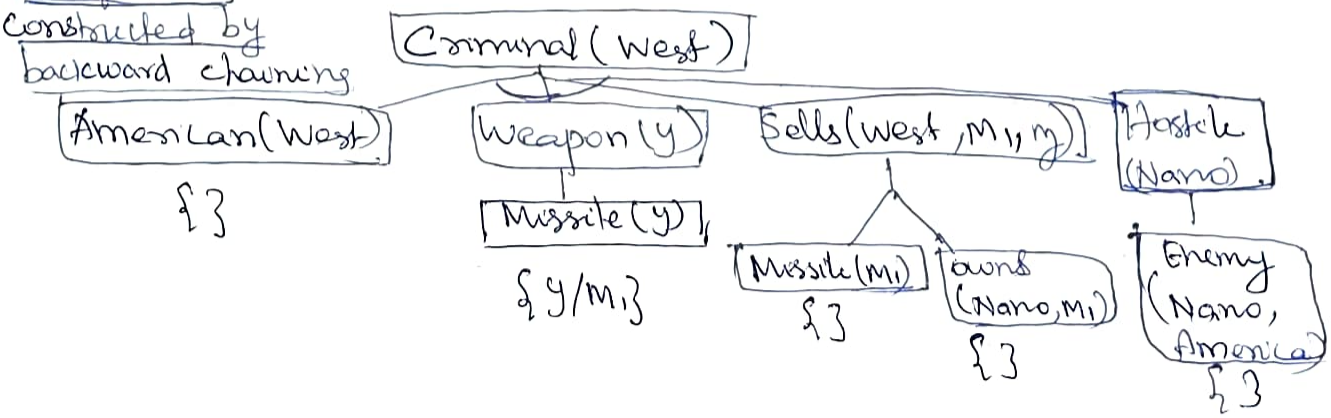
Step 3: Infer fact Missile(x) from Rule(5) with Substitution {x/M1}



Step 4:



Proof tree constructed by backward chaining



3.6 Resolution

Resolution is a theorem proving technique that proceeds by building refutation proofs. ie Proof by contradiction

- Resolution is used to prove a conclusion of various statements
- Resolution is a single-Inference rule which efficiently operate on conjuctive Normal Form (CNF) (or) clausal form

Clause: is a disjunction of literals

Conjunctive Normal Form

Resolution requires every sentences to be in a CNF

* Conjunction of clauses
* Each clause is a disjunction of literals
* Literals contain Universally Quantified variables

Conversion to CNF

(1) Eliminate all Implication ($\Rightarrow$) using the fact

$$A \Rightarrow B \equiv \neg A \vee B$$

(2) Move $\neg$ inwards

$\neg \forall x P \equiv \exists x \neg P$ $\quad \neg(A \wedge B) \equiv \neg A \vee \neg B$ $\quad \neg(\neg A) \equiv A$

$\neg \exists x P \equiv \forall x \neg P$ $\quad \neg(A \vee B) \equiv \neg A \wedge \neg B$

(3) Standardizing variables

Rename variables

$(\forall x\ P(x)) \vee (\exists x\ Q(x))$ - Change the name of anyone variable

$\forall x P(x) \vee \exists y Q(y)$

(4) Skolemize: It is the process of removing existential quantifiers by elimination.

$\exists x\ P(x)$ to $P(A)$

(5) Drop Universal Quantifiers

(6) Distribute $\wedge$ over $\vee$

$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

Eg: Convert "Everyone who loves all animals is loved by someone" to CNF

(1) FOL: $\forall x [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$

Eliminate Implications

$\forall x [\neg \forall y (\neg Animal(y) \vee loves(x,y) \vee \exists y\ loves(y,x))$

Move $\neg$ inwards

$\forall x [\exists y\ \neg(\neg Animal(y) \vee loves(x,y)] \vee \exists y\ loves(y,x)$

$\forall x [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee \exists y\ loves(y,x)$

Standardize Apart Rename variables

$\forall x [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee \exists z\ loves(z,x)$

Skolemize (Remove Extential Quantifiers)

$\forall x [Animal(A) \wedge \neg Loves(x,A)] \vee loves(B,x)$

$\forall x [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee loves(G(x), x)$

F, G $\Rightarrow$ Skolem functions

Drop Universal Quantifiers

$[Animal\ F(x) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$

Distribute $\wedge$ over $\vee$

$[Animal(F(x)) \vee Loves(G(x), x)] \wedge \neg Loves(x, F(x)) \vee Loves(G(x), x)$

Steps for Resolution

1. Conversion of facts into First order logic (FOL)
2. Convert FOL statements into CNF
3. Negate the statement which need to prove (Proof by contradiction)
4. Draw resolution graph (Unification)

Algorithm: Resolution proof

1. Negate the theorem to be proved and add to the KB
2. Convert KB into conjunctive Normal form (CNF)
3. Until there is no resolvable pair of clauses
   a) Find resolvable clauses and resolve them
   b) Add the results of resolution to the KB
   c) If empty clause is produced, stop and report that the original theorem is true
4) Report that the original theorem is false

Prove that "West is Criminal" using resolution.

The Sentences in CNF are

1) ¬American(x) V ¬Weapon(y) V ¬Sells(x, y, my) V ¬Hostile(my) V Criminal(x)
2) ¬Missile(x) V ¬Owns(Nono, x) V Sells(West, x, Nono)
3) ¬Enemy(x, America) V Hostile(x)
4) ¬Missile(x) V Weapon(x)
5) Owns(Nono, M1)
6) Missile(M1)
7) American(West)
8) Enemy(Nono, America)

Resolution Proof that West is a Criminal

¬American(x) V ¬Weapon(y) V ¬Sells(x, y, my) V ¬Hostile(my) V Criminal(x) ∨ ¬Criminal(West)

American(West) ∨ ¬American(x) V ¬Weapon(y) V ¬Sells(West, y, my) V ¬Hostile(my)

¬Missile(x) V Weapon(x) ∨ ¬Weapon(y) V ¬Sells(West, y, my) V ¬Hostile(my)

Missile(M1) ∨ ¬Missile(y) V ¬Sells(West, y, my) V ¬Hostile(my)

¬Missile(x) V ¬Owns(Nono, x) V Sells(West, x, Nono) ∨ ¬Sells(West, M1, my) V ¬Hostile(my)

Missile(M1) ∨ ¬Missile(M1) V ¬Owns(Nono, M1) V ¬Hostile(Nono)

Owns(Nono, M1) ∨ ¬Owns(Nono, M1) V ¬Hostile(Nono)

¬Enemy(America) V Hostile(x) ∨ ¬Hostile(Nono)

Enemy(Nono, America) ∨ ¬Enemy(Nono, America) ☐

Knowledge representation

8.7 Ontological Engineering

Representation of events, Time, physical objects and beliefs in many domains is called ontological engineering

Upper Ontology

* It is a general framework of a given concept
* It is represented using graph
  - General concepts are at the top of Graph
  - More specific concepts lies below

Example : Upper ontology of the world

General pupose ontologies

2 Major characteristics

1) It should be applicable in any special purpose domain
2) Different areas of knowledge can be unified

3.8 Categories and objects

The organization of objects into categories is a vital part of knowledge representation

Use of category

① Reasoning is easy ② Prediction is simple

2 choices for representing categories in FOL

(1) Predicates: apple(x)

(2) Reification of categories into objects : Apple

(3) Subcategory : Subclass, Subset

Subset(apples, fruits)

Inheritance

Categories use Inheritance concept to simplify the KB

All instances of food are edible

we have Food ↓ Fruit ↓ Apple - So we infer, Apples inherits the property of edibility from food

Part of relation is Transitive & Reflexive

- Part of(x,y) ∧ Part of(y,z) ⇒ part of(x,z)
- Part of(x,x)

Eg Part of (India, Earth)

two (or) More categories are

a) Disjoint — No members in common

eg Disjoint{Animals, Vegetables}

2) Exhaustive decomposition - {Americans, Canadians, Mexicans, NorthAmericans}
3) Partition e.g. partition({Males, Females}, Animals)

Composite objects:

Eg. Biped has two legs attached to a body

$$\text{Biped}(a) \Rightarrow \exists l_1, l_2, b \ \text{Leg}(l_1) \wedge \text{Leg}(l_2) \wedge \text{Body}(b) \wedge \text{PartOf}(l_1, a) \wedge \text{PartOf}(l_2, a) \wedge \text{PartOf}(b, b) \wedge \text{Attached}(l_1, b) \wedge \text{Attached}(l_2, b) \wedge l_1 \neq l_2 \wedge [\forall l_3 \ \text{Leg}(l_3) \wedge \text{PartOf}(l_3, a) \Rightarrow (l_3 = l_1 \vee l_3 = l_2)]$$

Bunch

Eg. BunchOf (Apple1, Apple2, Apple3)

- It denotes the composite object with 3 apple as parts

Measurements

The values assigned for the object properties (height, mass, cost) are called Measures

- Measures are written using unit functions

Length (L_1) = Inches(1.5) = Centimeters(3.8)

Measures and objects

- Measures can be used to describe objects

$d \in \text{days} = \text{Duration}(d) = \text{Hours}(24)$

+ Measures can be ordered using '>' symbol

eg. Norvig's exercises are tougher than Russell's exercises

$$e_1 \in \text{Exercises} \wedge e_2 \in \text{Exercises} \wedge \text{Wrote}(\text{Norvig}, e_1) \wedge \text{Wrote}(\text{Russell}, e_2) \Rightarrow \text{Difficulty}(e_1) > \text{Difficulty}(e_2)$$

Substances & objects

Realworld contains primitive objects (or) particles

Composite objects can be built from primitive objects.

Individuation

It is the process of dividing the objects into distinct objects. The divided portion is called Stuff

Count nouns - Cat, Dog, theorems

Mass nouns - Butter, water, Energy

2 kinds of properties

Intrinsic - Properties are retained under subdivision

eg density, flavour, color

Extrinsic - Properties are not retained under subdivision

eg- weight, length, Shape etc

3.9 Events - Representing change with events

Events are based on time rather than situation

Event calculus - is a formalism for reasoning about events, times and actions

Event calculus reifies fluents and objects

Eg- fluent

At(Shankar, Berkeley)

Fluent is an object that refers to the fact of "Shankar being in Barkeley"

A fluent is true at some point in time

T (At (Shankar, Berkeley), t)

↑ Predicate for time

Eg- for Event - E1- Shankar flying from Madurai to chennai.

$E1 \in Flying \wedge Flyer(E1, Shankar) \wedge origin(E1, Mdu) \wedge Destination(E1, Chennai)$

Complete set of predicates for Event calculus

T(f, t)	Fluent 'f' is true at time 't'
Happens (e, i)	Event 'e' happens over the time interval
Initiates (e, f, t)	Event 'e' causes fluent 'f' to start at time 't'
Terminates (e, f, t)	Event 'e' causes fluent 'f' to end at time 't'
Clipped (f, i)	Fluent f ceases to be true at some point during time interval
Restored (f, i)	Fluent f becomes 'true' sometime during time interval

Liquid Event

Categories of events with discrete property are called process or liquid events

- Liquid event is analogous to temporal substance

Time Intervals

2 kinds of time intervals

(1) Moments (2) Extended Intervals

- Moments have zero duration

$i \in moments \iff Duration(i) = Seconds(0)$

Duration - It gives the difference between the end time and start time

It is a fn

$Interval(i) \Rightarrow Duration(i) = Time(End(i) - Time(Begin(i))$

Date: It is a function that takes 6 arguments & returns a time point

Date (0, 20, 21, 24, 1, 1995) = Seconds (3000000000)

hour min sec day Month year

Meet: Two intervals meet if end time of first equals the start time of the second

Complete set of Interval relation

$Meet(i,j) \iff End(i) = Begin(j)$

$Before(i,j) \iff End(i) < Begin(j)$

$After(j,i) \iff Before(i,j)$

$During(i,j) \iff Begin(j) < Begin(i) < End(i) < End(j)$

$Overlap(i,j) \iff Begin(i) < Begin(j) < End(i) < End(j)$

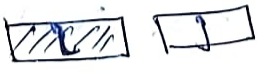
$Finishes(i,j) \iff End(i) = End(j)$

$Equals(i,j) \iff Begin(i) = Begin(j) \wedge End(i) = End(j)$

Graphical representation

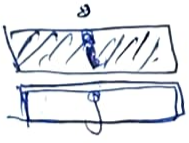
Meet (i, j)

Before (i, j)



During (i, j)

Equals (i, j)



3.10 Mental Events and Mental objects

Mental objects and mental processes are needed to manipulate objects

Propositional attitudes of an agent towards mental objects

- Believes
- Knows
- Wants
- Intends
- Informs

– these attitudes do not behave like 'normal' predicates

Eg. Suppose we try to assert that
Lois knows that Superman can fly

knows (Lois, canfly(superman))

- one minor issue with this is we normally think of canfly(Superman) as a sentence, but here it appears as a term.

– This issue can be patched up making canfly(superman) as a fluent

– Another issue is, if the Superman is Clark Kent is true, we must conclude that
Lois knows that Clark can fly

(Superman = clark) ∧ knows (Lois, canfly(superman))
⊨ knows (Lois, canfly(clark))

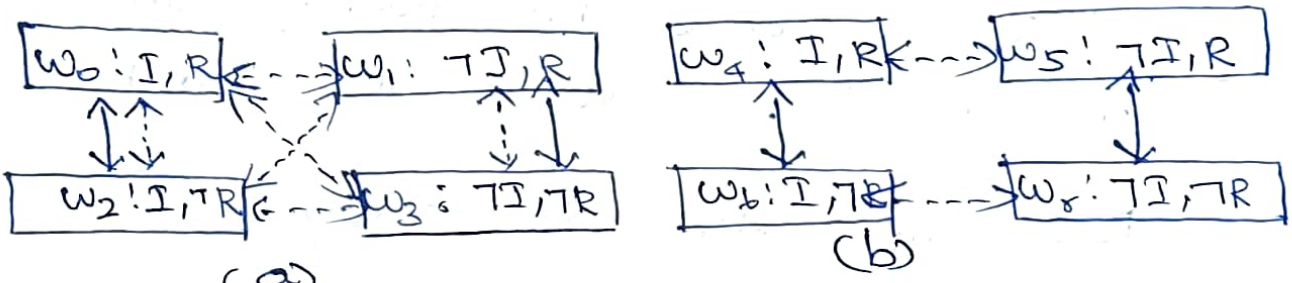
Modal logic is designed to address this problem
↳ includes special modal operators that takes sentences (rather than terms) as arguments

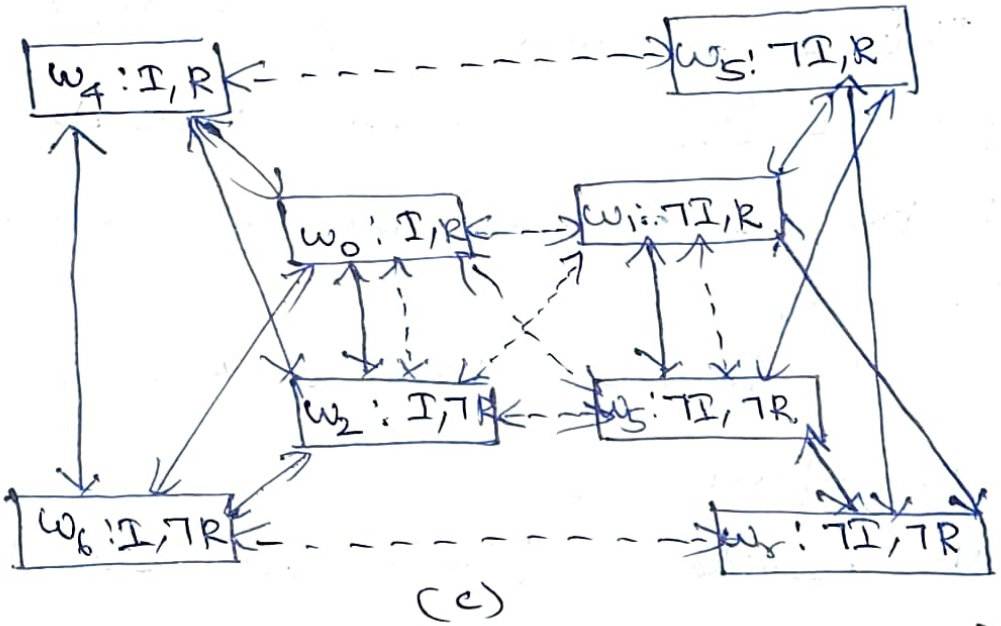
~~Figure 8~~

For mental event and mental objects need more complicated model that consists of all possible worlds

The worlds are connected by Accessibility relations in a graph.

World w_1 is accessible from world w_0 with respect to modal operator K_A if everything w_1 is consistent with what A knows in w_0. This is written as $Acc(K_A, w_0, w_1)$





In (a) the common knowledge is "Superman knows his own identity and neither he nor Lois has seen the weather report"

So in w_0, the worlds w_0 and w_2 are accessible to superman.

For Lois, all 4 worlds are accessible

In (b) the common knowledge is

" Lois has seen the weather report. Superman doesnot know the weather report"

So in w_4, Lois knows rain is predicted. In w_0 she knows rain is not predicted.

In (c), 2 top diagrams are combined. & the common knowledge is

" Superman knows his identity and Lois might or might not seen the weather report"

3.11 Reasoning Systems for categories

This can be done using 2 logics

(1) Semantic Logic : Semantic Networks

* It provides a graphical representation of a knowledge base
* Also it provides Inference Algorithm

(2) Description Logic

* It provides a formal language to construct category definition
* Also provide efficient algorithm for deciding subset and superset relationship between categories

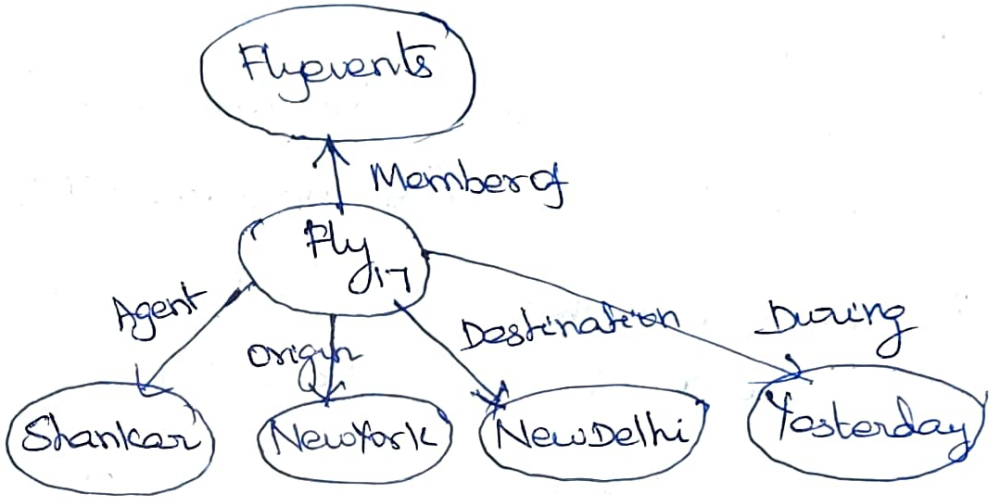
Semantic Networks

It represents individual objects, categories of objects and relations among objects

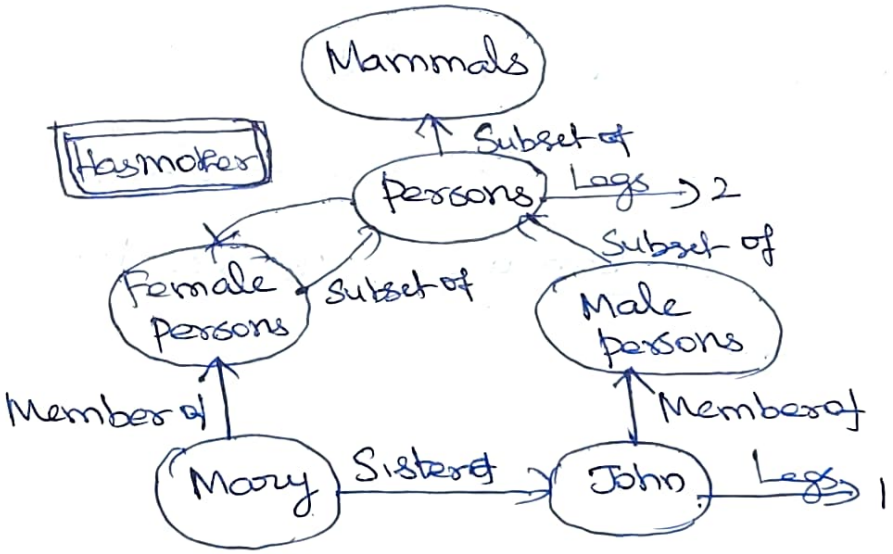
- Objects / category name - ovals/boxes
- links are labeled

Semantic Network - for the logical assertion

Fly (Shankar, NewYork, NewDelhi, Yesterday)



A Semantic Network with 4 objects (John, Mary, 1 and 2) and 4 categories. Relations are denoted by labeled links



In the above figure

Links :- member of link
- Sisteral link
- Subset of link – connect categories

Assertions

(1) Mary ∈ femalepersons

(2) Sister of (Mary John)

Double – boxed link:

Assertion

$\forall x \; x \in \text{Persons} \Rightarrow \forall y$ Has mother (x, y)

(i) $\forall x \; x \in \text{Persons} \Rightarrow [\forall y \; \text{Hasmother}(x, y) \Rightarrow y \in \text{Female persons}]$

(ii) To assert that person has two legs

$\forall x \; x \in \text{Person} \Rightarrow \text{Legs}(x, 2)$

Drawbacks : Links represent only binary relations

(ii) Description Logic

- It make use of Notations to describe definitions and properties of categories

Principal Inference Tasks

(1) Subsumption (2) classification

The classic language - It is a typical description language.

Syntax of descriptions in a subset of CLASSIC language

Concept → Thing | ConceptName
| AND (Concept, ...)
| ALL (Rolename, Concept)
| AtLeast (Integer, RoleName)
| AtMost (Integer, RoleName)
| Fills (RoleName, IndividualName, ...)
| SameAs (Path, Path)
| OneOf (IndividualName, ...)

Path → [Rolename, ...]

Examples

① Bachelors are unmarried adult males

Bachelor = And (Unmarried, Adult, Male)

② Describe the set of men with atleast three sons who are all unemployed and married to doctors and atmost two daughters who are all professors in physics or maths department

Description Logic!

And (Man, AtLeast (3, Son), Atmost (2, Daughter),
All (Son, And (Unemployed, Married, All (Spouse, Doctor)),
All (Daughter, And (Professor, Fills (Department, Physics, Maths))))

3.2 Reasoning with Default Information

1. Circumscription and Default logic
2. Truth Maintenance Systems

(i) Circumscription and Default Logic:

Idea: Specify particular predicates that are assumed to be false except those for which they known to be true

Eg. To assert the default rule

"Birds fly"

To specify a predicate, Abnormal(x) & we write

$Bird(x) \wedge \neg Abnormal_1(x) \Rightarrow Flies(x)$

The conclusion Flies(Tweety) can be drawn from Bird(Tweety). But the conclusion becomes false if $Abnormal_1$(Tweety) is asserted.

Default Logic & Default Rules

It generates contigent, nonmonotonic conclusions

Default rule : Bird(x) : Flies(x) / Flies(x)

Form of default rule

$$p : J_1, \ldots, J_n / C$$

where

- p → Prerequisite
- C → Conclusion
- J_i → Justifications

* If any one of the Justification is proven false then the conclusion cannot be drawn

ii) Truth Maintenance System

Sometimes, Inferred facts can be wrong and need to be retracted. This is said to be belief revision

Truth Maintenance systems are used to handle such kind of complications

For example, Suppose the KB contains a Sentence P which is an incorrect assertion

To avoid contradiction before executing TELL(KB, ¬P), one must execute RETRACT(KB, P)

Simple Approach to Truth Maintenance System

→ keep track of the order in which the sentences are added to the KB by numbering them from P_1 to P_n

→ When RETRACT(KB, P_i) is made, the system reverts to the state just before P_i was added by removing P_i and all the inferences derived from P_i. Then the sentences P_{i+1} to P_n can be added again

Justification based Truth Maintenance System (JTMS)

- More efficient approach
- Each sentence in the KB is annotated with a set of sentences from which it was inferred
- Justification makes retraction efficient

If we issue RETRACT(P), the JTMS will delete those sentences for which P is a member of every justification

Assumption based Truth Maintenance system (ATMS)

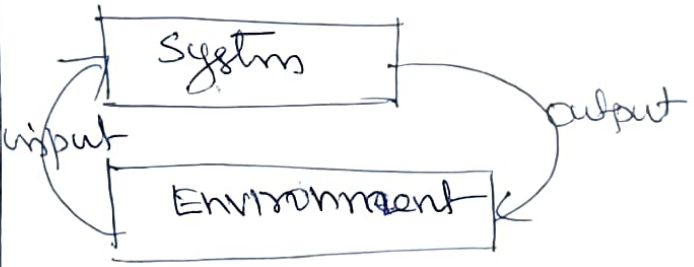
- It is efficient for context switching between hypothetical worlds
- ATMS keeps track of assumptions that cause the sentence to be true.
- TMS provides a mechanism for generating explanations
- An explanation of a sentence P is a set of sentences E such that E entails P.

UNIT IV -SOFTWARE AGENTS

Architecture for Intelligent Agents - Agent Communication - Negotiation and Bargaining - Argumentation among Agents - Trust and Reputation in Multi Agent Systems.

4.1 Definition of Agent:

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors



Reactivity

An agent has to be able to react in an appropriate way to the dynamic changes in its environment

Kinds of enviroments

1) Accessible Vs inaccessible
   ↳ is one in which agent can obtain complete information about the environment
2) Deterministic Vs non-deterministic
   ↳ is one in which any action has a single guaranteed effect.
3) Episodic Vs Non-episodic
   ↳ the performance of an agent is dependent on a no of discrete episodes.

4) Static Vs dynamic
↳ is one which can be assumed to remain unchanged.

5) Discrete Vs continuous
↳ if there are a fixed finite no. of actions and percepts in it

Agent Architectures

An architecture proposes a particular methodology for building an autonomous agent

Main kinds of agent architectures

1) Reactive Architectures
2) Deliberative Architectures
3) Hybrid architectures
4) Belief - Desire - Intention (BDI) Architecture
5) Blackboard Architecture
6) Mobile Architecture

1) Reactive Architectures

→ focused on fast reactions/responses to changes detected in the environment

→ Reactive agents have
* atmost a very simple internal representation of the world
* But provide tight coupling of perception and action

Main characteristics

→ Emergent functionality
* simple agents & simple interaction
* complex behaviour patterns appear as a

result of the dynamic interactions

→ Task decomposition

* Agents composed of autonomous modules
* Each module manages a given task
* Minimal, low level communication between modules

→ Raw data

* Basic data from sensors
* No complex symbolic management of data

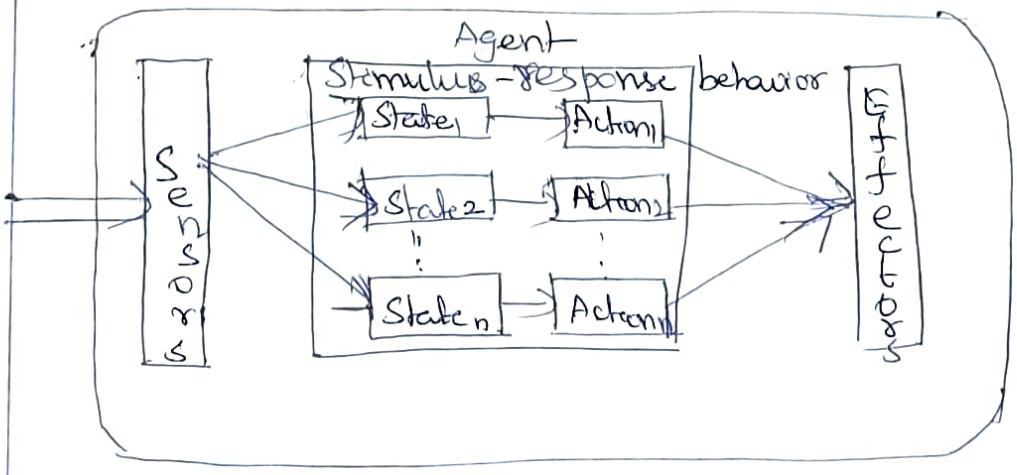
Basic concept

* Each behaviour maps perceptual input to action output
* Reactive behaviour: action rule : S → A

S - denote the States of the environment

A → primitive actions

Eg: $action(S) = \begin{cases} \text{Heater off}, & \text{if temp is OK in state } S \\ \text{Heater on}, & \text{otherwise} \end{cases}$



Basic schema of reactive architecture

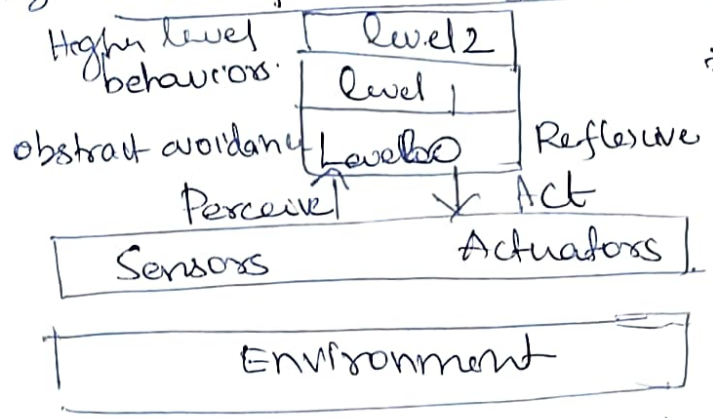
Advantages:

1. Simplicity
2. Flexibility & adaptability
3. Computational tractability
4. Robustness against failure

Disadvantages

1. They apply only to simple environments
2. Sequence of actions require the presence of state, which is not encoded into the mapping function

Eg. Subsumption Agent Architecture



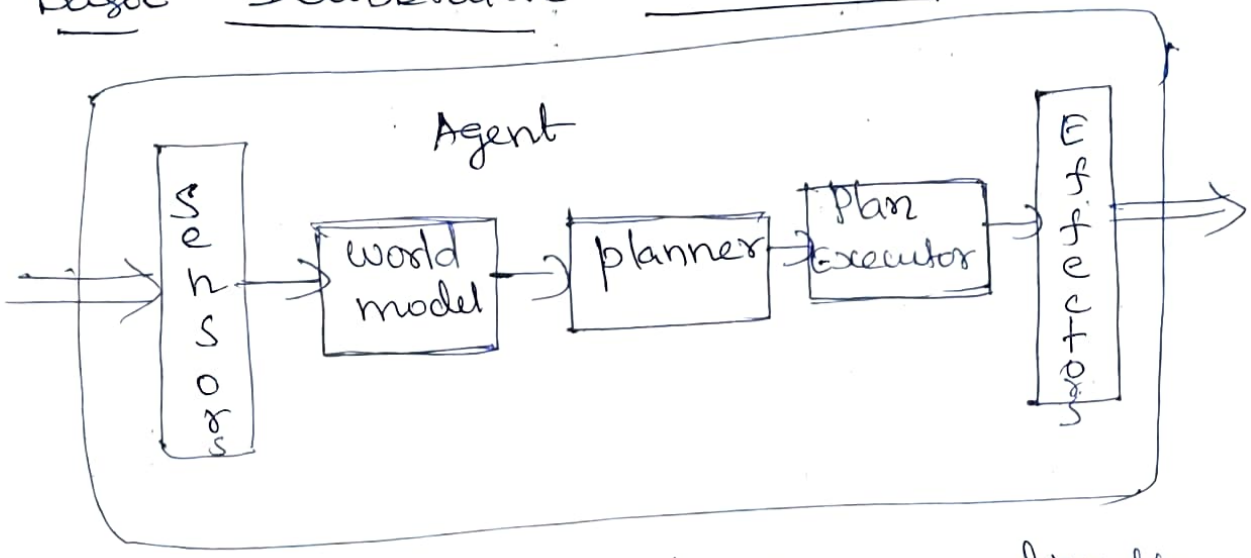
* Subsumption is a parallel and distributed architecture for managing sensors and actuators.

- It was created by Rodney Brooks in the year 1980 to do research in behavior-based robotics
- Intelligent behavior can be created through a collection of simple behavior modules
- Behavior modules are collected into layers.
  - Lower layers are reflexive in nature & top level behaviors are complex
- It Begins with a simple set of behaviors, once they succeeded, additional higher level behaviors are added.

(2) Deliberative agent Architecture

- Explicit Symbolic model of the world
- Decisions are made via logical reasoning, based on pattern matching and symbolic manipulation
- Sence – plan – act problem solving paradigm of classical AI planning systems

Basic Deliberative Architecture



- Instead of mapping the sensors directly to the actuators, it considers the sensors, State, prior results of given action & other information in order to select the best action to perform

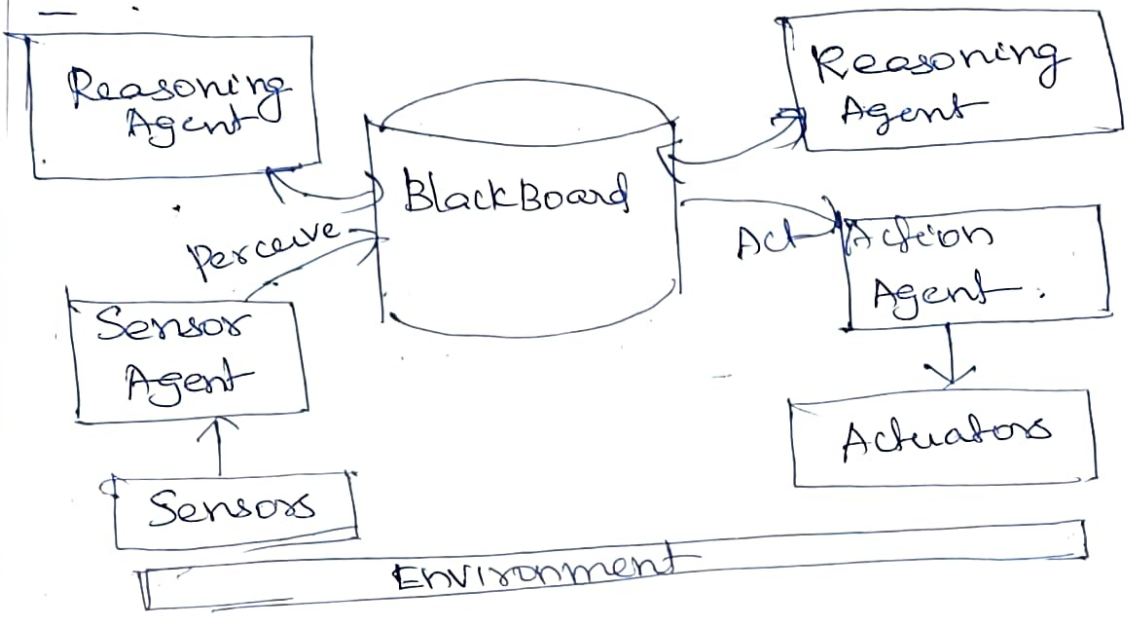
Advantages 1) It can be used to solve much more complex problems than reactive architecture

2. It can perform planning, sequences of actions to achieve a goal

Disadvantage: It is slower than reactive architecture due to the deliberation for the action to select

5) Blackboard Architecture

- It is a very common architecture
- First blackboard architecture was HEARSAY-II which was Speech understanding system
- The blackboard is a common work area for a number of agents that work cooperatively to solve a given problem



* Two separate agents are used to sample the environment through the available sensors
  - Sensor agent & action agent
* The control of the agent system is provided by one or more reasoning agents
* First reasoning agent - implement the goal definition behaviors, second reasoning agent implement - translate goals into sequence of actions
* This architecture along with its globally available work area is implemented with a multi-threading system

4. Belief - Desire - Intention (BDI) Architecture

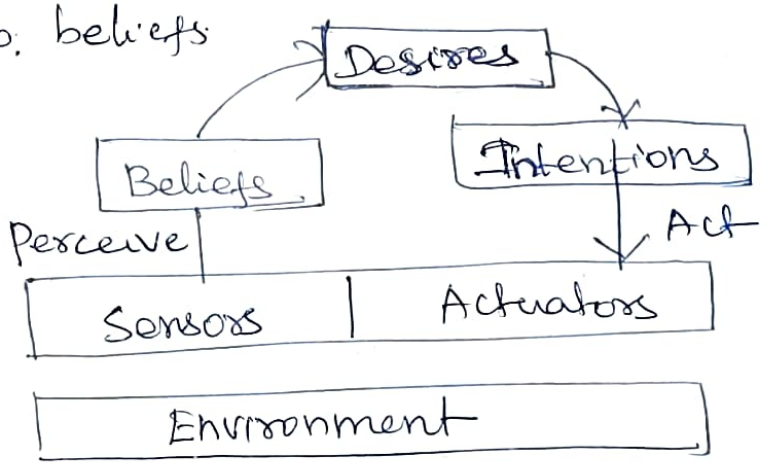
- It is an architecture that follows the theory of human reasoning

Belief - represents the view of the world by the agent.

Desires - are the goals that define the motivation of the agent

Intentions - is Choice with commitment. Specify that the agent uses Beliefs and Desires in order to choose one or more actions in order to meet the desires

- It stores a representation of the state of the environment (beliefs), maintains a set of goals (desires), and an intentional element which maps desires to beliefs



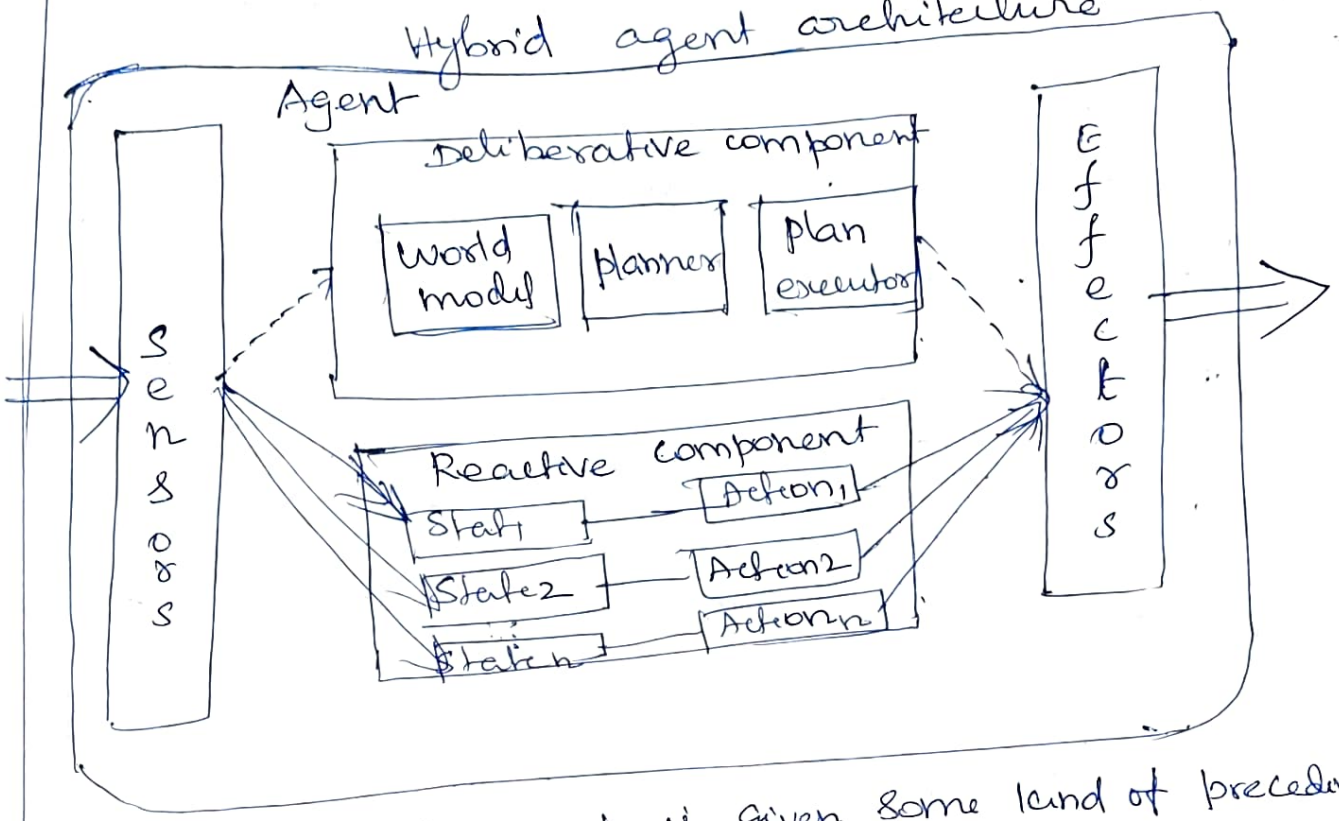
5) Hybrid Architecture

This approach is to build an agent out of two subsystems such as deliberative and reactive one.

Deliberative one - Containing a symbolic world model which develops plans & makes decision in the way proposed by symbolic AI.

A reactive one - which is capable of reacting quickly to the events

Hybrid agent architecture

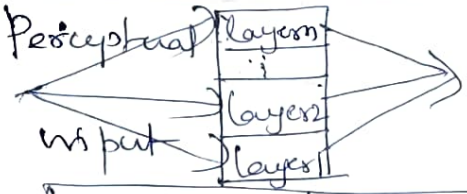


* Reactive component is given some kind of precedence over deliberative one
* This kind of structuring leads to the idea of layered architectures

Eg. TOURINGMACHINES, INTERRAP

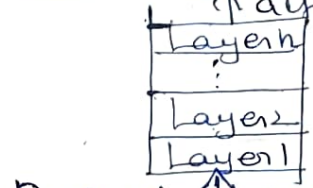
* An agent's control subsystems are arranged into a hierarchy with higher layers dealing with information at increasing levels of abstraction

Horizontal layering



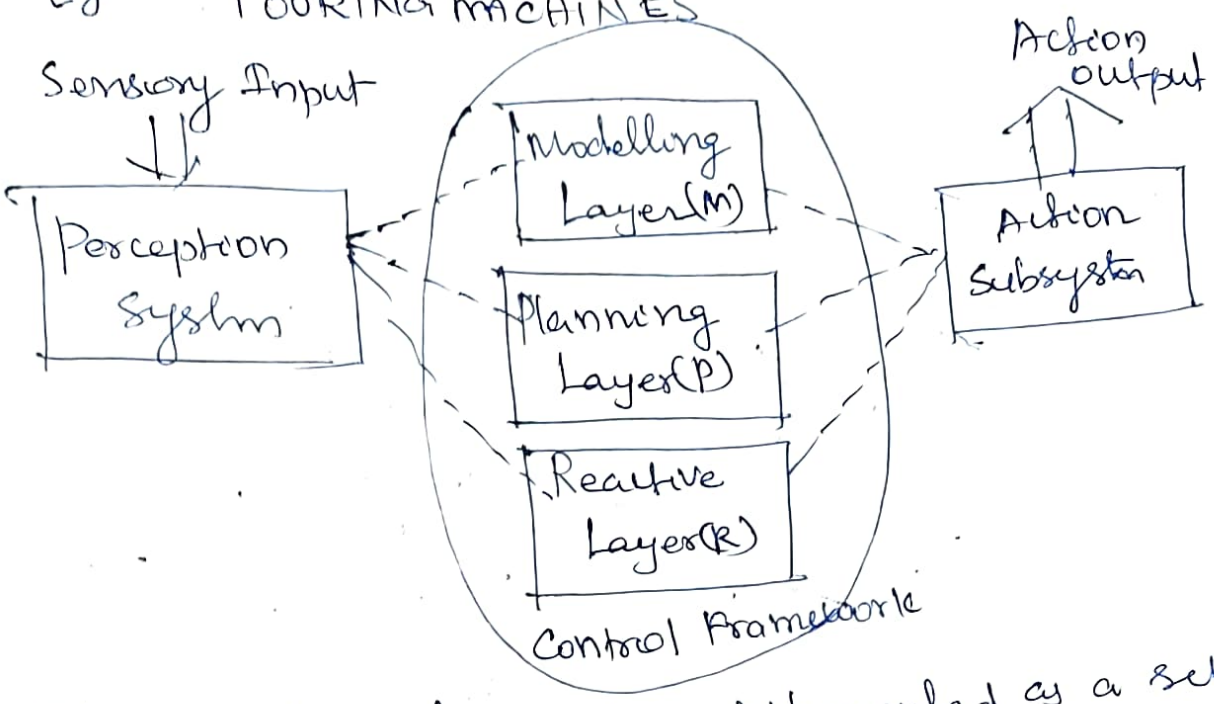
* Each layer is directly connected to the sensory input & action output

Vertical layering



* Sensory input and action output are dealt with by atmost one layer each.

Eg. TOURING MACHINES



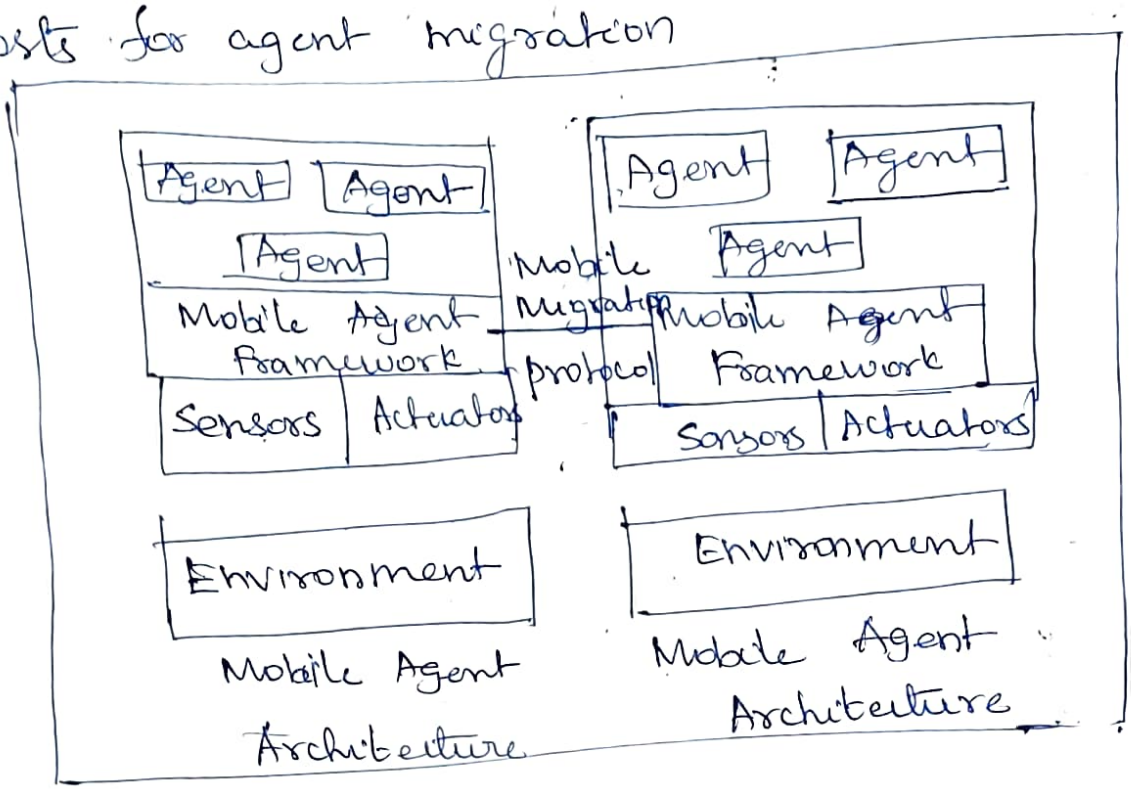
* The reactive layer is implemented as a set of situation–action rules,
* The planning layer constructs plans and select action to execute in order to achieve the agents goals
* The modelling layer contains symbolic representations of the cognitive state of other entities in the agent Environment
* The three layers communicate with each other & are embedded in a control framework

6. Mobile Architecture

– This architecture pattern introduces the ability for agents to migrate themselves between hosts.

– This architecture includes the ~~mobility~~ mobility element, which allows an agent to migrate from one host to another.

* The mobile agent framework provides a protocol that permits communication between hosts for agent migration



Architecture Descriptions

1. Subsumption Architecture (Reactive Architecture)
2. Behavior Networks (")
3. ATLANTIS (Deliberative Architecture)
4. Homer (")
5. BB1 (Blackboard)
6. Open Agent Architecture (Blackboard)
7. Procedural Reasoning System (Belief-Desire-Intention)
8) Aglets (Mobile Architecture)
9) Messengers (")
10) Soar (Hybrid Architecture)

4.2 Agent Communication

In multi agent systems, communication is an important characteristic to support both coordination and the transfer of information.

Need for Agent communication

* Multi agent system allow distributed problem solving. In order to do so, individual agents to interact
  - allows cooperation
  - allows information Sharing

Communication categories

- Type of Sendee - addressee link
- Nature of the medium

Sendee - Addressee Link

Communication can be

(i) point to point — An agent talks directly to another agent

(ii) Broadcast — An agent sends same information to a group of agents

(iii) Mediated — Communication between two agents is mediated by a third party

eg. - facilitators

Nature of the medium

(i) Direct routing — Messages sent directly to other agents, no interception in strength

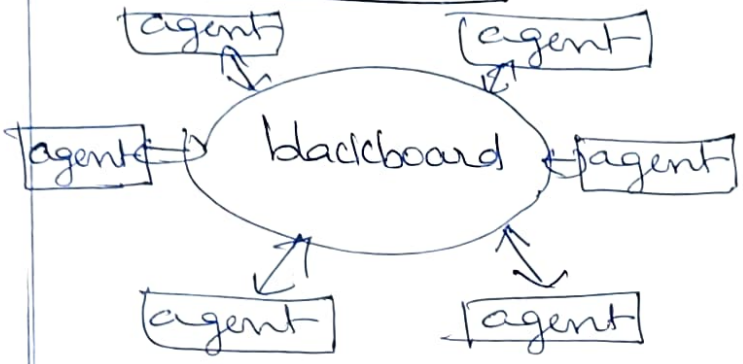
(ii) Signal propagation routing — Intensity decreases according to distance.

(iii) public notice routing — Blackboard systems

Basic options used in Multiagent Systems

1) Blackboard systems 2) Direct message passing

Blackboard System



* Each agent can put information/ data on the common information space
* No direct communication between agents

Message passing

Information is passed from one agent to another. Nature of this information can be varied. Speech act provide one way to describe this variety

Speech Act:

* A speech act is an act of communication
* By using the various types of speech act, agents can interact effectively.

Types of Speech Act

1) inform other agents about some data
2) Query others about their current scituation
3) Answer Questions
4) request others to act
5) promise do do something
6) offer, deals
7) acknowledge offers & requests

Speech Act components

Two components 1) A performative verb
(eg. request, inform, promise..)
2) propositional content (: "the door is closed")

Communication Standards!

Standards
* Allow different groups to write cooperating agents
* Hold abstract out communication, by defining high level general languages & protocols.

FIPA – ACL — Foundation for Intelligent physical system – Agent FIAP, ACL Communication lanage

Components of FIPA – ACL message

Begin message → ACL message

Structure

(inform
:Sender agent1
Communicative act type :receiver hpl-auction-Server
:Content
(price (bid good02) 150) ← Message content Expression
:inreply-to round 4
:language sl
:ontology hpl-auction ← parameter expression
)

Parameters in a FIPA ACL message

:Sender - who sends the message

!receiver - who is the recipient of the message

!content - content of the message

:reply-with - identifier of the message

:reply-by - deadline for replying the message

:in-reply-to - identifier of the message being replied

:language - language in which the content is written

:ontology - ontology used to represent the domain concepts

:Protocol - communication protocol to be followed

:Conversion-id - identifier of conversation

Communication protocols

1) FIPA - request protocol

2) FIPA - Query protocol

3) FIPA - contract Net protocol

KQML - Knowledge Query and Manipulation Language

Communication requires

- the ability to locate and engage a peer in a conversation - communication layer
- A method for packing the messages (messaging layer)

Content Layer
- an internal format that represents the messages, requests, responses & plans

* In a network of KQML-speaking agents there exists
  - programs to support communication
  - facilitators which serve as name servers to KQML components
  - KQML router supports the routing of messages

* A KQML message can be transferred to any particular socket & has a format that consists of performative & a set of arguments

* The performative defines the speech act which defines the purpose of the message
(assertion, command, request etc)

* Performative name – defines the particular message type to be communicated
(evaluate, ask-if, Stream-about, reply, tell)

* Sender & receiver – define the unique names of agents in the dialogue

* Content – is information specific to the performative being performed

The Structure of KQML message

```
(performative-name
 :sender X
 :receiver Y
 :content Z
```

: language L
: ontology Y
: reply-with R
: in-reply to Q

Eg: Conversation between two KQML agents regarding - An agent requests the current value of a temperature sensor in a system

(ask one
: sender thermal-control-appl
: receiver temp-server
: language prolog
: ontology CELSIUS-DEGREES
: content "temperature(TEMP-SENSOR-1 A) temperature)"
: reply-with request-102
)

KQML PERFORMATIVES

Evaluate - Evaluate the content of the message

ask one - Request for the answer to the question

reply - communicate a reply to the question

Stream about - provide multiple response to a question

Sorry - return an error

tell - inform an agent of sentence

* KQML provides a rich set of capabilities that cover basic speech acts, more complex acts including data streaming and control of information transfer

ACL (FIPA- Agent communication Language)

- FIPA-ACL is a consortium based language for agent communication
  - ACL is a speech act language defined by a set of performatives
  - ACL is similar to KQML - adopt the views and uses content layering for message construction
  - ACL also uses Semantic Language (SL) which provides the means to support BDI themes. (beliefs, desires, intentions)

4.3 Negotiation and Bargaining

Negotiation - It is a form of interaction in which a group of agents with conflicting interests try to come to a mutually acceptable agreement over some outcome

- outcome is represented in terms of allocation of resources
- Agents interests are conflicting if they cannot be simultaneously satisfied,

Bilateral negotiation

- Negotiations involving two agents

Multi party Negotiations - involving many of agents

Main ingredients of Negotiation

1. Negotiation object - which defines the set of possible outcomes
2. The agents conducting the negotiation
3. The protocol - Based on which agents search for a specific agreement
4. The individual Strategies - which determine the agents behavior based on their preferances over the outcomes

Negotiation outcomes - also called as agreements (or) deals

↳ characteristics → continuous or discrete
→ Single issue or multiple issues

Eg: Ltr milk between X & Y

- Dividing milk - is - single issue & continuous
- possible outcome can be represented as [0, 1.0]
- one possible outcome is 0.21 for X & 0.79 for Y

Utility function

- One way to define preference relation for agent i is to define a utility function

$u_i : O \to R^+$, assign real no. to each possible outcome

- In case of multiple issues, the utility function is a mapping of the form

$u^i : A_1 \times \cdots \times A_m \to k$

A_k - is set of possible values for the issue k

Cumulative utility - is the weighted sum of utilities from the individual issues and is of the form

$$u^i = \sum_{k=1}^{m} w_k^i u_k^i(a_k) \quad \text{where } u_k^i : A_k \to *$$

* The utility function $u_i(*)$ represents the relation $\geq_i$ when we have $u_i(O_1) > u_i(O_2)$ if $O_1 \geq_i O_2$.

- A rational agent attempts to reach a deal that maximizes his/her utility

Protocols - It is the rules of Interaction for enabling agents to search for an agreement

Strategy

- The Strategy specify what offer to make next (or) what information to reveal
- A rational agents strategy aim to achieve the best possible outcome for her/him
- Game theory is analyzing agents' Strategic behavior

Attributes of Ideal Negotiation Mechanism

1) Efficiency
2) Stability
3) Simplicity
4) Distribution
5) Symmetry

Three types of environments

1) Worth oriented domains
2) State oriented domains
3) Task oriented domains

A task oriented domain is one where agents have a set of tasks to achieve, all resources needed to achieve the tasks are available, and the agents can achieve the tasks without ~~interference~~ interference from each other.

Eg. Internet downloading domain

- where each agent is given a list of documents that it must access over the Internet
- There is a cost associated with downloading, each agent would like to minimize

The environment provide the following negotiation mechanism & constraints

1) Each agent declares the documents it wants
2) Documents common to two or more agents are assigned to agents based on the toss of a coin
3) Agents pay for the documents they download
4) Agents are granted access to the documents they download as well as any in their common set.

In formal term, a task oriented domain is represented as a tuple $\langle T, A, C \rangle$

Where

T – is the set of tasks

A – is the set of Agents

C(X) – function for the cost of executing the tasks X.

A deal is - a redistribution of tasks

The utility of deal d for agent k is

$$U_k(d) = c(T_k) - c(d_k)$$

The conflict deal D occurs when the agents cannot reach a deal.

4.3 (b) Bargaining

In a Bargaining settings, agents can make a mutually beneficial agreement, but have a conflict of interest about which agreement to make.

2 major subfields of bargaining theory

1) Axiomatic 2) Strategic

Axiomatic Bargaining Theory

– Desirable properties for a solution called axioms of the bargaining solution are postulated and then the solution concept that satisfies those axioms is sought.

The Nash bargaining solution

Nash analyzed a 2 agent setting where the agents have to decide on an outcome $O \in \mathcal{O}$, and the fallback outcome $O_{fallback}$ occurs if no agreement is reached. There is a utility function $u_i : \mathcal{O} \to \mathbb{R}$ for each agent i. It is assumed that the set of feasible utility vectors

$\{(u_1(O), u_2(O)) \; O \in \mathcal{O}\}$ is convex.

This occurs, if outcomes include all possible lotteries over actual alternatives

The axioms of the Nash bargaining solution $u^* = (u_1(O^*), u_2(O^*))$ are

Invariance - The agents numeric utility functions really represent ordinal preferences among outcomes. The actual cardinalities of the utilities do not matter.

Anonymity (symmetry) - Switching labels on the players does not affect the outcome.

Independent of irrelevant alternatives

Pareto efficiency - It is not feasible to give both players higher utility than under $u^* = (u_1(O^*), u_2(O^*))$

The unique solution that satisfies these four axioms is

$$O^* = \arg\max_O [u_1(O) - u_1(O_{fallback})][u_2(O) - u_2(O_{fallback})]$$

Strategic Bargaining Theory

In this theory, the bargaining solution is modeled as a game and the solution concept is based on an analysis of which of the players' strategies are in equilibrium.

- This theory explains the behaviour of rational utility maximizing agent better than axiomatic approaches
- This theory analyses sequential bargaining where agents alternate in making offers to each other in a prescribed order

Eg: One can think of deciding how to split a dollar

In a protocol with a finite no. offers and no time discount, the unique payoffs of the subgame perfect Nash equilibria are such that the last offerer will get the whole dollar (minus ϵ), because the other agent is better off accepting ϵ than by rejecting and receiving nothing.

Rubinstein bargaining solution

In a discounted infinite round setting, the subgame perfect Nash equilibrium outcome is unique. Agent 1 gets $(1-\delta_2)/(1-\delta_1\delta_2)$ where δ_1 is 1's discount factor, and δ_2 is 2's. Agent 2 gets one minus this. Agreement is reached in the first round.

Let $\overline{\Pi_1}$ - the maximum undiscounted share that I can get in any subgame perfect Nash equilibrium on his turn to offer.

Setting them equal gives:

$$\overline{\Pi_1} = 1 - \delta_2(1 - \delta_1\overline{\Pi_1}) \Leftrightarrow \overline{\Pi_1} = \frac{1-\delta_2}{1-\delta_1\delta_2}$$

Computation in Bargaining

There are actually two searches occurring in bargaining.

In the intra-agent deliberative search, an agent locally generates ~~the~~ alternatives, evaluates them lookahead in the negotiation process.

In the inter-agent committal search, the agents make agreements with each other regarding the solution.

- The two search model is ~~for~~ similar to the Real time A* search where an agent has to trade off thorough deliberation against more real-world actions.

In modeling bargaining settings, each agent's strategy should incorporate both negotiation actions and deliberation actions. The agents' strategies should be in equilibrium.

4.4 Argumentation among Agents

Argumentation - can be defined as an activity aimed at convincing of the acceptability of a standpoint by putting forward propositions justifying (or) refuting the standpoint

Argument - Reasons / Justifications supporting a conclusion

Process of Argumentation

(1) Constructing arguments in favor of / against a statement from available information.

A : Tweety is a bird, so it flies

B : Tweety is just a cartoon!

(2) Determining the different conflicts among the arguments

"Since Tweety is a cartoon, it cannot fly!"
(B attacks A)

(3) Evaluating the acceptability of different arguments.

We'll assume Tweety is a cartoon (accept B)

This means despite being a bird he cannot fly
(reject A)

(4) Concluding (or) defining the justified conclusions

"We conclude that Tweety cannot fly!"

Argument

- Arguments as chained Inference Rules
- Arguments as Instances of Schemes
- Arguments as Graphs

Arguments as chained Inference Rules

A tuple $(L, -, R_s, R_d, \leq)$

L - logical language

R_s - Strict Rules

R_d - defeasible rules

$\leq$ - Partial order over R_d

$- : L \to 2^L$ captures conflict between formulas

Contrariness function

Knowledge Base

- A particular knowledge base $(K, \leq')$ with:

$K \subseteq L$ divided into:

* K_n are necessary axioms (cannot be attacked)
* K_p are ordinary premises
* K_a are assumptions
* K_i are issues

- $\leq'$ is a partial order on $K \setminus K_n$

Inference rules:

* defeasible rule $\varphi_1, \ldots \varphi_n \Rightarrow \varphi$ means conclusion φ follows from the premises $\varphi_1, \ldots \varphi_n$
* Strict rule $\varphi_1, \ldots \varphi_n \to \varphi$ stands for classical implication

- Functions Prem(A), Conc(A) and Sub(A) returns premises, conclusion and sub-arguments of argument A respectively

An argument is any one of the following.

- $\varphi \in K$, where $Prem(A) = \{\varphi\}$, $Conc(A) = \varphi$ and $Sub(A) = \{\varphi\}$
- $A_1, \ldots A_n \rightarrow \psi$, where $A_1, \ldots An$ are arguments, there exists in R_s a strict rule $Conc(A_1), \ldots, Conc(An) \rightarrow \psi$
- $A_1, \ldots An \Rightarrow \psi$ where $A_1, \ldots An$ are arguments, and there exists in R_d a defeasible rule $Conc(A_1), \ldots Conc(An) \Rightarrow \psi$

where

- $Prem(A) = Prem(A_1) \cup \ldots \cup Prem(An)$
- $Sub(A) = Sub(A_1) \cup \ldots \cup Sub(An) \cup \{A\}$

Example: With this knowledge Base

* $R_s = \{p, q \rightarrow s ; u, v \rightarrow w\}$, $R_d = \{p \Rightarrow t ; s, r, t \Rightarrow v\}$,
* $k_n = \{q\}$, $k_p = \{p, u\}$, $k_a = \{r\}$

We can construct the following arguments

A_1: p A_3: r A_5: $A_1 \Rightarrow t$ A_7: $A_3, A_5, A_6 \Rightarrow v$

A_2: q A_4: u A_6: $A_1, A_2 \rightarrow s$ A_8: $A_4, A_7 \rightarrow w$

with

* $Prem(A_8) = \{p, q, r, u\}$, $Conc(A_8) = \{w\}$,
* $Sub(A_8) = \{A_1, A_2, A_3, A_4, A_5, A_6, A_6, A_8\}$

Attack Among Arguments

Undercut – An argument can undercut another argument by showing that a defeasible rule cannot be applied

A_8 can be undercut by an argument with conclusion $\bar{A}_5$ since argument A_5 is constructed with a defeasible rule

Rebut – An argument can rebut another argument by supporting the opposite conclusion

A_8 can be rebutted on A_5 with an argument with conclusion $\bar{t}$ or rebutted on A_7 with an argument with conclusion $\bar{v}$

Undermine (or) Premise Attack)

An argument can undermine another argument by attacking one of its premises

A_8 can be undermined by an argument with conclusion $\bar{p}, \bar{r}$ or u

Defeat among Arguments

Defeat – Argument A defeats argument B if the former attacks the latter, and is also preferred to it according to some preference relation $\prec$

Argumentation Scheme

– They are forms of argument representing stereotypical ways of drawing inferences from particular patterns of premises to conclusions in a particular domain.

For each scheme, we list

① Premises

② Conclusion

A set of critical questions that can be used to scrutinize the argument by questioning explicit or implicit premises

Arguments as Graphs

An argumentation framework is a pair $AF = \langle A, \rightarrow \rangle$ where A is a finite set of arguments and $\rightarrow \subseteq A \times A$ is a defeat relation.

An argument α defeats an argument β if $(\alpha, \beta) \in \rightarrow$

Eg Argument α_1 has two defeaters α_2 and α_4 which are themselves defeated by arguments α_3 and α_5 respectively.

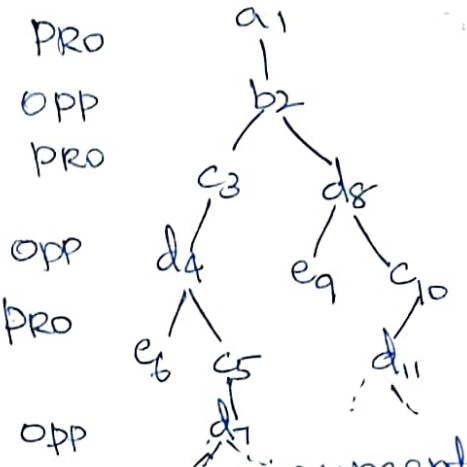
Argument Labeling

A labelling specifies which arguments are:

* Accepted (in) * Rejected (out) * Undecided (undec)

* An argument is in if and only if all of its defeaters are out
* An argument is out if and only if atleast one of its defeators is in
* Otherwise it is undecided

Example
Dispute tree



Argumentation protocols

1) Abstract Argument Games
2) Dialogue systems

① Abstract Argument Game

- PRO (the proponent)
- OPP (the opponent)

* Dialogue begins with PRO asserting an argument.
* Then PRO and OPP take turns in a dispute, where each player makes an argument that attacks his counterpart's last move
* Agent wins a dispute if his counterpart cannot counter attack
* But the counterpart may try a different line of attack, creating a new dispute
* This results in a dispute tree structure

② Dialogue Systems

- Arguments have explicit internal structure
- More permissible than abstract argument games
- uses elements of speech act theory
- The protocol generates a tree structure
- Each utterance is a node, and its possible responses are its children
- A move is in if it is surrendered (or) if all its attacking replies are out
- A move is out if it has a reply ie in
- whether the proponent (or) the opponent wins depends on whether the critical move is in or out

Argument Interchange Format

* common language for annotating argument structures
* ~~Expressive~~ expressive enough & customizable
* Nodes of two main types
  - Information nodes (or I-nodes) N_1 CN: represent a claim, premise, data etc
  - 'Scheme' nodes (or S-nodes) N_S CN: capture applications of patterns of reasoning

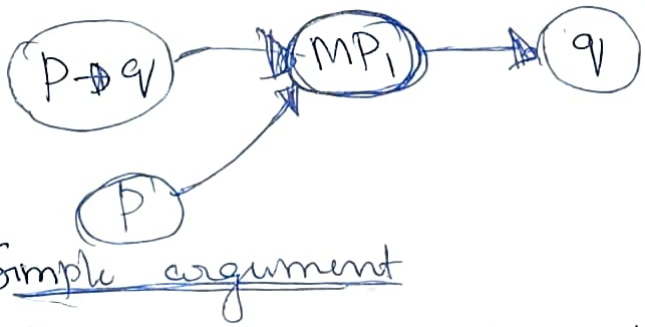
Connecting Information nodes via Scheme nodes

* cannot connect two I-nodes directly
* Must go via S-node, which captures the relationship

Scheme nodes are of 3 types

- rule of inference application nodes (RA-nodes)
- preference application nodes (PA-nodes)
- conflict application nodes (CA-nodes)

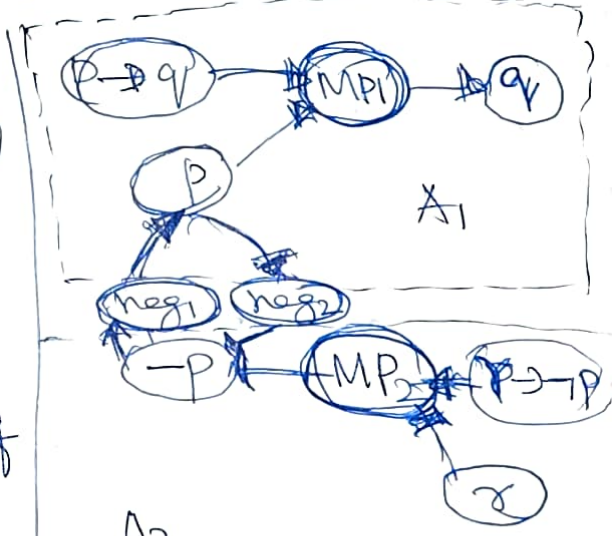
Eg.



Simple argument

MP_1 – denotes the application of modus ponens

neg_1 – " application of classical negation



A2

4.5 Trust and Reputation in Multiagent Systems

Trust is an individual measure of confidence that a given agent has over other agents

Reputation is a social measure of confidence that a group of agents or a society has over agents or groups. Reputation is one mechanism to compute trust

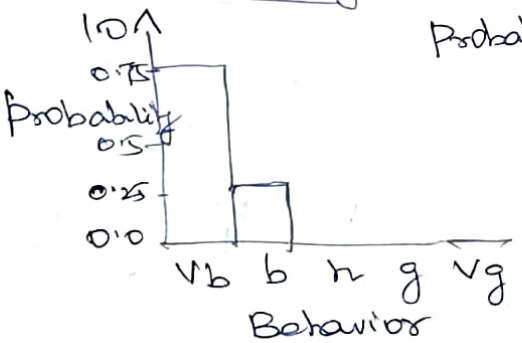
Computational representation of trust and reputation values

Boolean < True – the trustee is trustworthy
False – " untrustworthy

Numerical values – Real(or) integer values in a range
eg [-1.0, 1.0], [0, 3000]

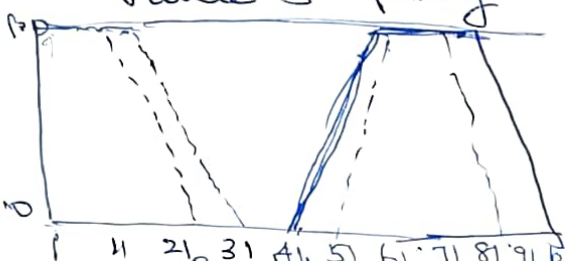
Quantitative labels – finite set of labels in an ordered set
Eg. { very bad, bad, neutral, good, very good}

Probability Distribution of Probability 0.75 - the behavior of the agent will be very bad with a probability of 0.25 it will be bad.

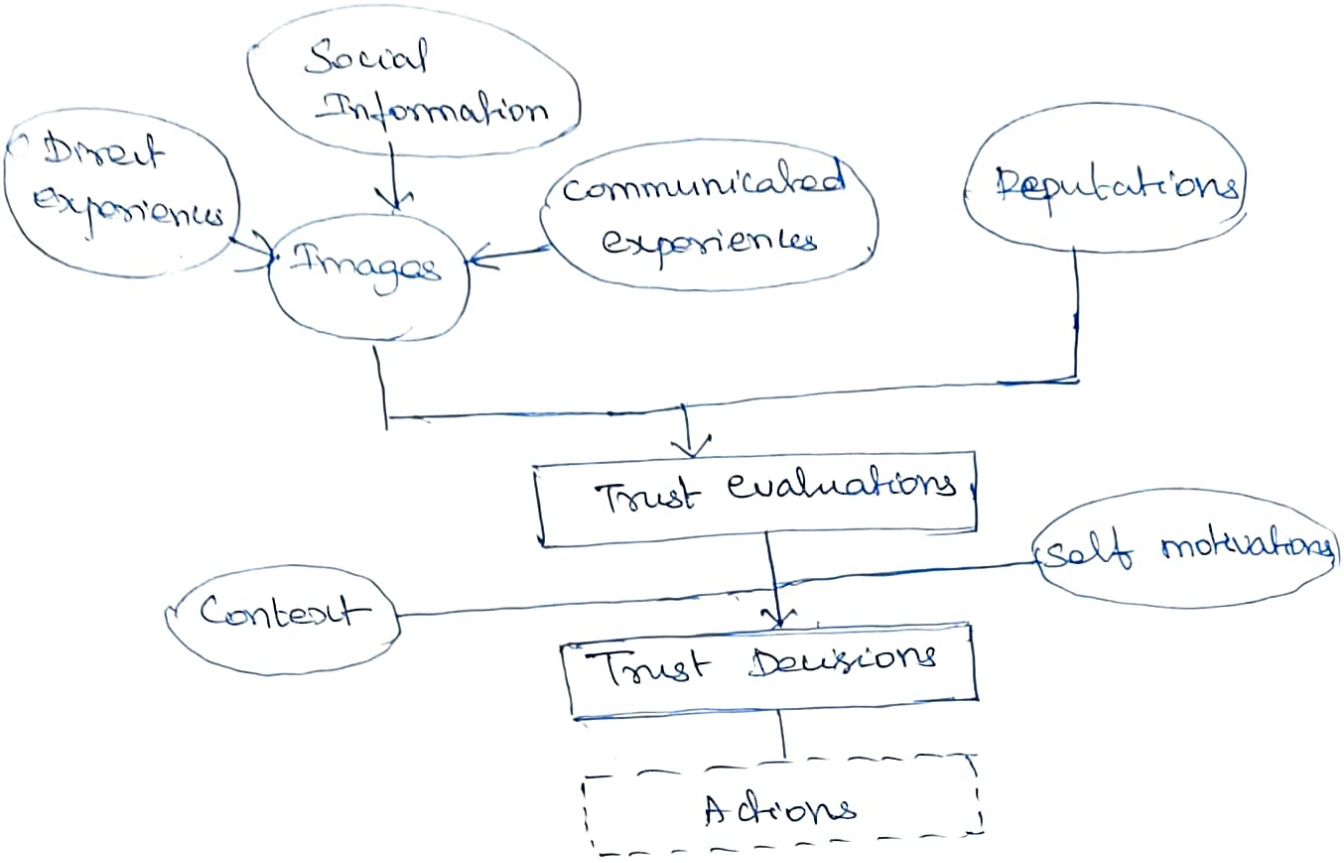


Fuzzy Sets – The reputation value is fuzzy set over a range

The reliability of reputation is represented in the shape of fuzzy set.



Trust processes in multiagent systems



Trust processes

A dual nature of trust:
- Trust as an evaluation
- Trust as an act

* Trust evaluation - A trustor X uses various information sources to decide if a trustee Y is trustworthy.

It consists of a set of social evaluations (either images or reputations)

* Trust Decision

It is a decision process taking into account trust evaluations

Trust evaluations

- input coming from different sources

Direct experiences

- Direct interactions between the trustor and the trustee

Communicated experiences

- Interactions between the trustee and another agent communicated to the trustor

Social Information - Social relations and position of the trustee in the society

Trust evaluation by

① A statistical evaluation - Compute a single value from a set of input

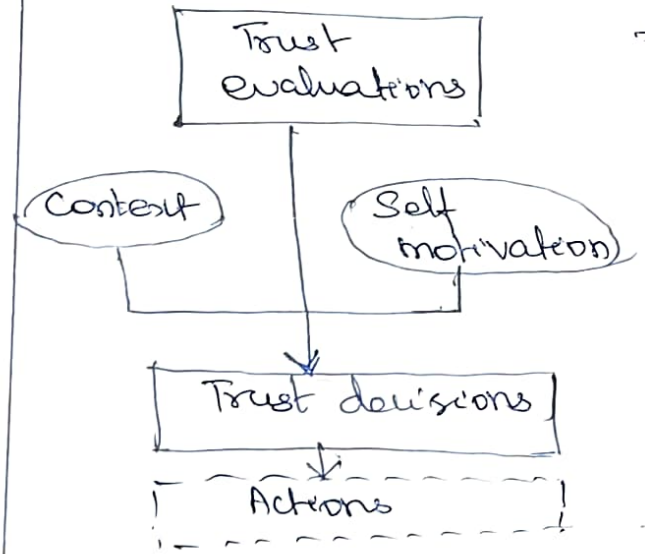
Eg. Aggregation $t_n = T_j^i = p/n$

Trustor i had n experiences with the trustee j in which p were positive

② Logical beliefs generation

Infer a trust evaluation from a set of beliefs

Trust decision - Trust as an act



* Trust decision process takes into account
  - trust evaluations
  - context of the decision
  - motivations of the trustor

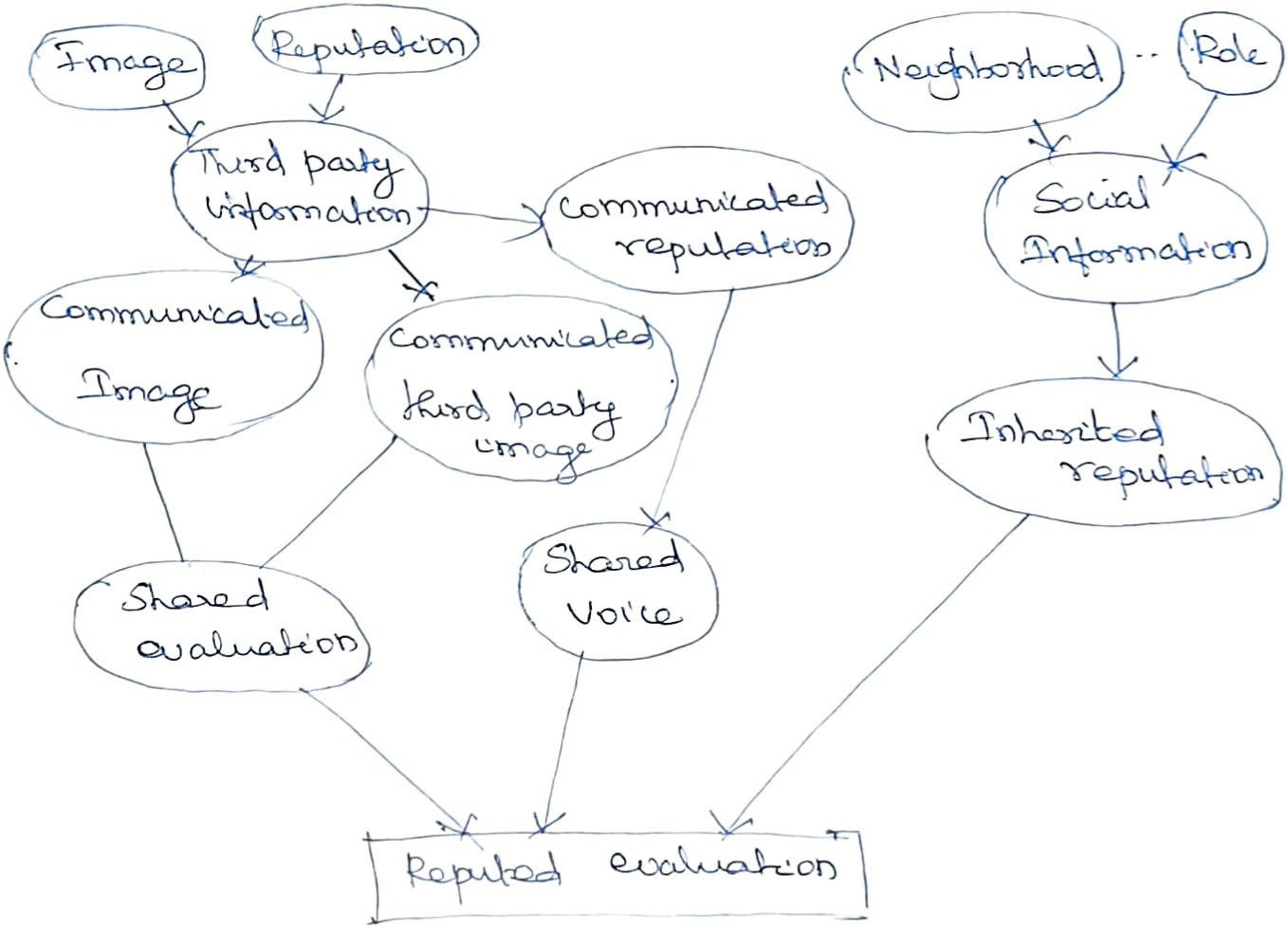
* It depends on the representation formalism of trust evaluations.

* Trust decision relying on
  - trust value threshold
  - trust belief

Reputation in multiagent Societies

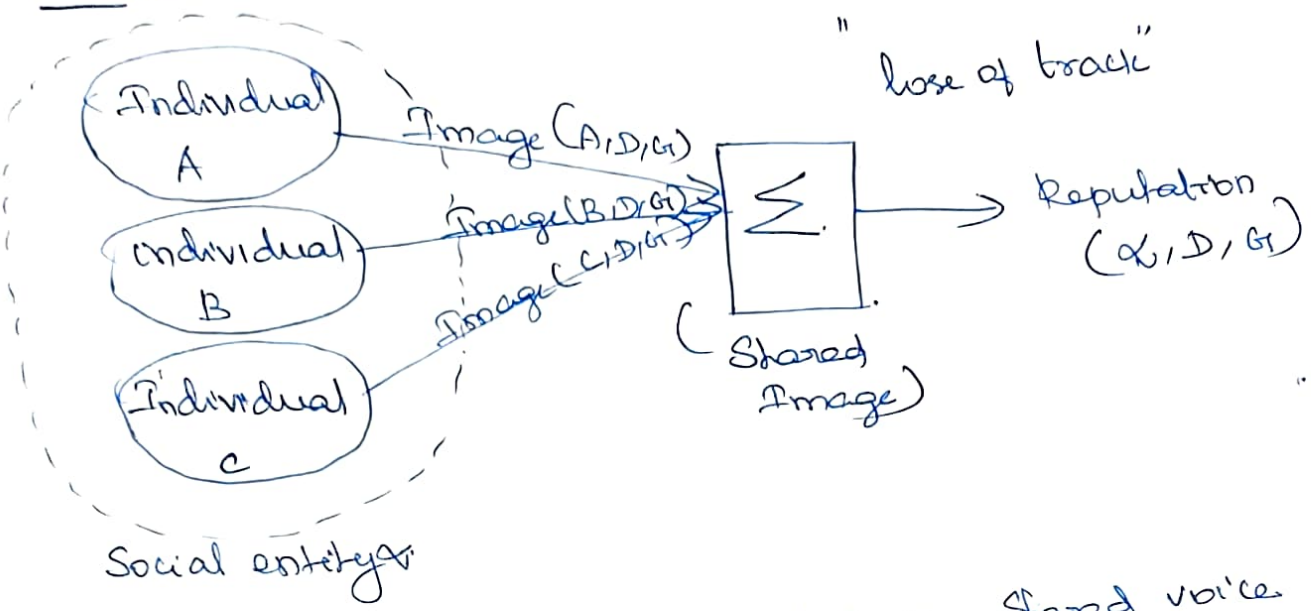
- What a social entity says about a target regarding his/her behavior
  - It is always associated to a specific behavior / property
- Reputation is one of the elements that allow us to build trust
- Reputation has a social dimension.

Sources of Reputation



Communicated Image as a source for reputation

It consists of aggregating the images that other members in the society communicate, as the reputation value

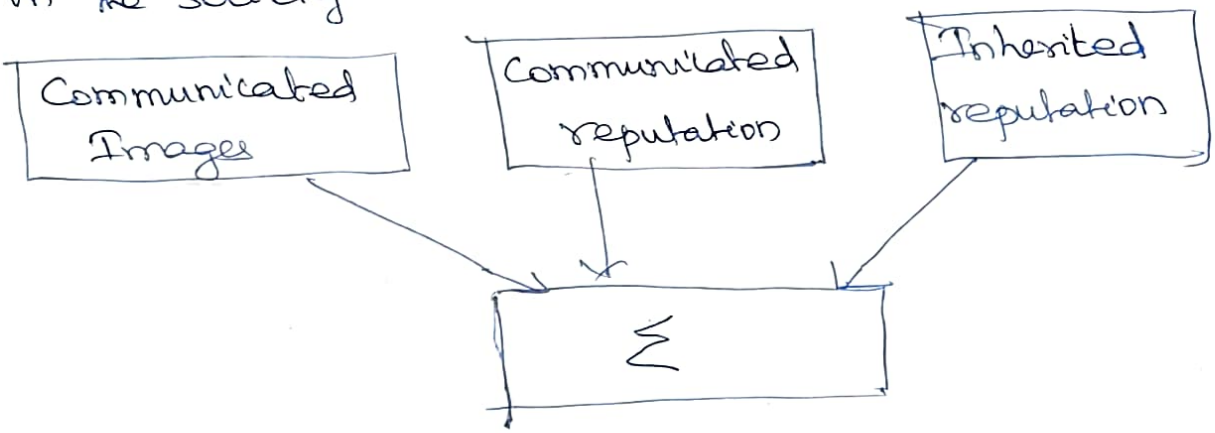


Communicated Reputation - Shared voice

It is based on the aggregation of information about reputation received from third parties

Inherited reputation

- It is directly inherited from third party agents with whom the subject has some kind of social relation
- It is associated to the role the subject is playing in the society



Centralized Vs Decententralized model

Centralized modul

Pros: 1. All the information available in the society can be used
2. Wrong or biased information has a lesser impact on the final value
3. First comers can benefit from the information

Cons: 1. Do not take into account personal preferences and biases
2. The central repository is bottleneck for the system
3. Security problems

Decentralized models

Pros: 1. No trust on external central entity is necessary
2. They donot introduce any bottleneck
3. Each agent can decide the method that wants to use to calculate reputation

Cons: 1. It takes some time for the agent to obtain enough information to calculate a reliable reputation value
2. It demands more complex and intelligent agents.

Reputation is a social evaluation that circulates in a society. It is a source of trust

UNIT V – APPLICATIONS

AI Applications – Language Models – Information Retrieval – Information Extraction – Natural language Processing – Machine Translation – Speech Recognition – Robot – Hardware – Perception – Planning – Moving

5.1 AI applications

* AI aims to create machines and automated technologies that can successfully simulate human intelligence.
* AI is changing the world Socially / Politically and economically
* Google Assistance, SIRI, Alexa, Smart car are prime examples of AI
* AI has revolutionized almost every sector from business, healthcare, military, education gaming, entertainment and much more

Examples of AI Technology

AI is incorporated into a variety of different types of technology

1. Automation – Robotic process Automation (RPA) a type of software that automates repetative, rules-based data processing tasks done by human
2. Machine Learning – This is the field of study that gives computers the capability to learn without being explicitly programmed.

3. Machine vision - This technology gives a machine the ability to see - captures and analyzes visual information using camera. (computer vision)

4. Natural language processing (NLP) This is the processing of human language by a computer program. NLP tasks includes text translation, Sentiment analysis & Speech recognition

5. Robotics - Focuses on design & manufacturing of Robots [Robots are used to perform tasks that are difficult for human to perform consistently]

6. Self driving cars - Autonomous vehicles uses a combination of computer vision, image recognition & deep learning

Applications of AI

① AI in healthcare - Administration, Telemedicine, Assisted Diagnosis, Robot-assisted surgery, online health monitoring.

② AI in business - Better recommendations, chatbots, Filtering Spams Fake reviews, Supply-chain Management.

③ AI in education - Automate grading, Personalised learning

④ AI in Finance - Predicting future patterns of market Stock trading

⑤ AI in Gaming - Deep minds. AI based AlphaGo Implementation of Augmented reality & Virtual Reality

5.2 Language Models

Language modeling is the use of various statistical and probabilistic techniques to determine the probability of a given sequence of words occuring in a sentence

* Language models analyze bodies of text data to provide a basis for their word predictions

Classification of language Models

Statistical Language Models:

These models use traditional statistical techniques like N-grams, Hidden Markov Models (HMM) and certain linguistic rules to learn the probability distribution of words.

Neural Language Models

- They use different kinds of Neural Networks to model language

Unigram language Model

Unigram: A combination of one-state finite automata. It evaluates each word (or) term independently

This model calculates the probability of each word in the language model as distributed over the document is

The probability distribution over the model will sum to 1

$$\sum_{\text{term in doc}} P(\text{term}) = 1$$

The probability of a specific query is

$$p(\text{query}) = \prod_{\text{term in query}} p(\text{term})$$

N-gram Language model.

It create a probability distribution for a sequence of n words

For eg if n=5, a gram look like

" can you please call me"

Types of n-grams are unigrams, bigrams, trigrams and so on.

$$p(w_1, \ldots w_m) = \prod_{i=1}^{m} p(w_i | w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} p(w_i | w_{i-(n-1)}, \ldots w_{i-1})$$

The n-gram model assumes that the probability of encountering the ith word, w_i in the context history of the proceding i-1 words can be predicted by the probability of the shortened context history.

Eg In the bigram (n=2) language model the sentence " I saw the red house" is approximated as

$$p(I, saw, the, red, house) \approx p(I|<s>)\, p(saw|I)\, p(the|saw)\, p(red|the)\, p(house|red)\, p(</s>|house)$$

<s> - marker denoting the beginning & end of the sentence

In a trigram (n=3) language model

$$p(I, saw, the, red, house) \approx p(I|<s><s>)\, p(saw|<s>, I)\, p(the|I, saw)\, p(red|saw, the)\, p(house|the, red)\, p(</s>|red, house)$$

Exponential language models

Also known as maximum entropy models.

It encode the relationship between a word and the n-gram history using feature functions.

$$P(w_m | w_1, \cdots, w_{m-1}) = \frac{1}{Z(w_1, \cdots, w_{m-1})} \exp(a^T f(w_1, \cdots, w_m))$$

where

$Z(w_1, \cdots, w_{m-1})$ - is the partition function

a - is the parameter vector

$f(w_1, \cdots, w_m)$ - is the feature function

This model is based on the principle of entropy, which states that the probability distribution with the most entropy is the best choice

Neural networks language models

* This type of model represents words as a non-linear combination of weights in a neural network
* It is useful as data sets get increasingly large and include unique words
* With increasing words, the possible word sequences increase, and so the patterns predicting the next word become weaker.
* This model can be trained using standard neural algorithms such as stochastic gradient descent with back propagation

$$P(w_t | w_{t-k}, \cdots, w_{t-1})$$

Network predicts from a feature vector that represents the previous k words.

One could also use future words as well as past words as features and the estimated probability

$$p(w_t | w_{t-k}, \cdots, w_{t-1}, w_{t+1}, \cdots, w_{t+k})$$

This is referred as "bag of words" model

Uses and examples of language modeling

Language models are backbone of NLP. Some applications are:

① Speech recognition – used by voice assistants like Alexa & SIRI
② Machine translation – Google Translate and Microsoft Translate
③ Parsing – Spell checking application
④ Sentiment Analysis – Repustate & Hubspot's servicehub Google NLP tool – BERT
⑤ OCR – use of machine to convert images of text into machine encoded text
⑥ Information retrieval – web browsers

5.3 Information Retrieval

Information retrieval is the task of finding documents that are relevant to a user's need.

Eg: Search engines on WWW.

An IR is Characterized by

① A corpus of documents

- Document may be a paragraph, a page or a

② Queries posted in a Query language

Queries can be list of words, phrase of words can contain a boolean operator (AI and Book)

③ A result set - Subset of documents relevant to the Query

④ Presentation of result set - It is the ranked list of document titles

Information Retrieval model

IR model is basically a pattern that defines retrieval procedure and consists of the following

1) A model for documents
2) A model for Queries
3) A matching function that compares queries to documents

Mathematically it is

$R(q, d_i)$ - A similarity function which orders the documents with respect to the query.

R - Representation of Queries

D - Representation of documents

F - The modeling framework for D, Q along with relationship between them

Types of IR model

1) Classical IR model -
   - This model is based on mathematical knowledge that was easily recognized & understood

Eg:- Boolean, Vector & Probabilistic models

2) Non-classical IR model
   - Not based on principles such as similarity probability and boolean operations

Eg - Information logic model, situation theory model & interaction model

3) Alternative IR model - enhancement of classical IR model

Eg. cluster model, fuzzy model & latent Semantic indexing

Boolean Model

- It is the oldest IR model. It is based on set theory & the boolean algebra.

Documents are sets of terms and queries are boolean expression on terms.

Boolean model can be defined as

D - A set of words i.e the indexing terms present in document

Q - A boolean expression, where terms are index terms and operators are AND, OR & NOT

F- Boolean algebra over sets of terms as well as sets of documents

K- A document is predicted as relevant to the query expression

Eg: Query with terms "social" and "economic" will produce the documents that are indexed with both terms

Advantages

1. It only retrieves exact matches
2. Simplest model based on sets

Disadvantages

1. Query language is expressive & it is complicated
2. No ranking for retrieved documents

Vector Space Model

* The index representations (documents) and the queries are considered as vectors embedded in a high dimensional Euclidean space
* The similarity measure of a document vector to a query vector is usually the cosine of the angle between them

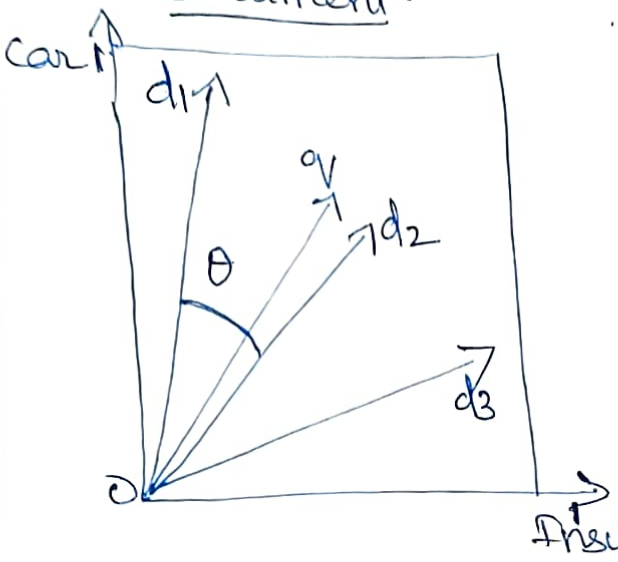
Cosine similarity

$$Score(\vec{d}\vec{q}) = \frac{\sum_{k=1}^{m} d_k \cdot q_k}{\sqrt{\sum_{k=1}^{m} (d_k)^2} \sqrt{\sum_{k=1}^{m} m(q_k)^2}}$$

Score $(\vec{d}\vec{q}) = 1$ when $d = q$

Score $(\vec{d}\vec{q}) = 0$ when d and q share no item

Eg. Vector space representation of Query & Document



* The query & documents are represented by a two dimensional vector space
* Terms are "car" & "Insurance"
* One Query & 3 documents
* The top ranked document in response to terms "car" & "insurance" is d_2

Because the angle between q and d_2 is the smallest

IR Refinements

— Refining the information retrieval

Casefolding

- More IR Systems do case-folding.

Eg. Convert "COUGH" to "cough"

ie If the query contains "COUGH" it includes those documents contain both COUGH as well as cough

Stemming Algorithm

It is the heuristic process of extracting the base form of words by chopping off ends of words

Eg. the words laughing, laughs, laughed would be stemmed to root word laugh

Synonyms related words - can be considered.

Eg. [New Sofa] [new couch] update it to [new sofa or new couch]

Meta data - could be considered
Eg- Human supplied keywords
Links - must be considered.
Hypertext between documents are important

Page Rank Algorithm

The pageRank algorithm outputs a probability distribution which is used to represent the likelihood that a person randomly clicking on the links will arrive at any particular page.

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

ie PageRank value for a page 'u' is dependent on the PageRank values for each page 'v' contained in the set B_u (the set containing all pages linking to page u), divided by the amount $L(v)$ of links from page v.

HITS Algorithm

-HITS uses Link Analysis for analysing the relevance of pages but only works on small sets of subgraph and it is query dependent.
The subgraphs are ranked according to weights in hubs and authorities where pages that ranks highest is fetched & displayed.

5. Information Extraction

It is the task of automatically extracting structured information from unstructured and/or semi structured machine readable documents & other electronically represented sources

Eg: Automatic annotation and content extraction out of images / audio / video / documents

→ Extracting addresses from web pages with fields for street, city, state and pin

Finite State Automata for Information Extraction

- Simplest type of Information extraction
- Attribute based extraction system
- It assumes the entire text refers to a single object and the task is to extract attributes of that object

Eg. Consider the following text

"IBM Think Book 970. Our Price: $300"

After Extraction

{Manufacturer = IBM, Model = Think Book 970
Price = $300}

- Can extract attributes by defining a template pattern for each attribute
- This template is defined by finite state Automaton, Regular expression.

Ex: Template for prices in dollar

[0-9] matches any digit from 0 to 9

[0-9]+ matches one or more digits

[$][0-9]+[(.)[0-9][0-9]] matches $249.99 or $300.49 ...

Relational extraction systems

- Deal with multiple objects & relations among them

Eg: FASTUS system — A system for extraction of information from Natural language text.

Relational extraction system can be built as a series of cascaded Finite State transducers

FASTUS consists of 5 Stages

1. Tokenization
2. Complex word handling
3. Basic group handling
4. Complex phrase handling
5. Structure Merging

① Tokenization : It segments the stream of chars into tokens (words, numbers, punctuation)

② Complex words handling:

It handles complex words such as "set up", "joint venture"

It uses finite state grammar rules

CapitalizedWord + ("company" | "co" | "Inc" | "Ltd")

A company name might be recognized by the above rule.

③ Basic groups

It creates different groups such as verb group (VG), Noun group (NG), prepositions / conjuction

VG : Said PR : to

NG : Friday CJ : and

④ Complex phrases

It combines basic groups into complex phrases.

⑤ Structure Merging

It merges the structures built up in the previous step.

Probabilistic models for Information Extraction

- Simplest model for Information extraction is Hidden Markov Model (HMM)
- Hidden Markov Model progress through a sequence of hidden states with observation at each step

2 types of (HMM)

① Building HMM for all the Attributes

② Building a HMM for each Attribute

Eg:

Consider the HMM

① To recognize the speaker in a talk announcement

② To recognize dates

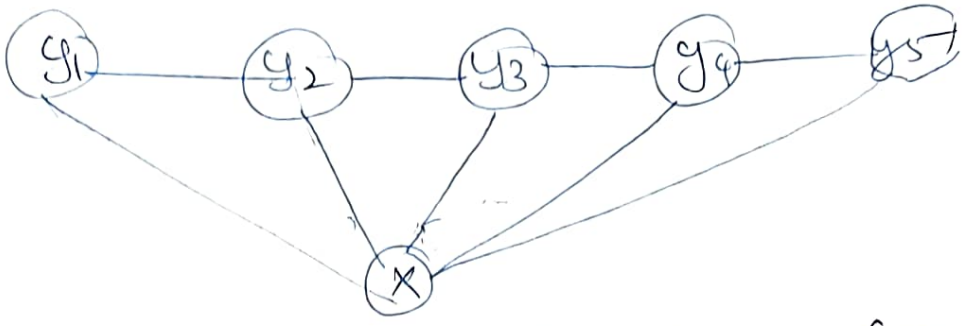
Text:	There	will	be	a	Seminar	by	Dr	Andrew	on	Friday
MMM, Speaker	–	–	–	–	PRE	PRE	TARGET	TARGET	POST	–
Hmm2 Date	–	–	–	–	–	–	–	–	PRE	TARGET

* The '_' indicates a background state
* observation : "words of text"
* Hidden states : target, prefix, postfix

- Prefix covers expressions such as "Speaker", "seminar by"
- Target covers the titles & first names
  ↳ covers Initials & Last name

Conditional Random fields for Information extraction

- CRFs are a type of discrimination undirected probabilistic graphical model.
- CRF is used to encode the relationship between observation and construct consistent interpretation



Ontology extraction from Large Corpora

* Generally Information Extraction finds a specific set of relations in a specific text.

Eg: Find speaker, time & location in talk Announcement

* Different Applications of Information Extraction

* Building a large KB or ontology of facts from a corpus

This can be done in 3 ways:

i) open ended: Acquire knowledge about all types of domains not a specific domain

ii) With a Large Corpus: With Question answering on the web corpus

iii) Results / facts can be collected from multiple sources

Automated Template construction

Information are extracted from a large corpus using templates

Given a good set of templates system can collect a good set of examples. Given a good set of examples the system can build a good set of templates

Consider the following examples.

The words in these examples were used in a search over a web corpus and that results in 100 matches. Each match is defined as a tuple of 7 strings.

Eg: ("Norvig", Artificial Intelligence", "Russell" "AIMA")

Template is

(Author, Title, Order, Prefix, Middle, Postfix, URL)

Machine Reading

Machine reading systems are more like a human reader. They learn from the text itself.

Machine that could read a document and answer as well as human

Eg: TEXTRUNNER

From the parse of the sentence "Einstein received the Nobel prize in 1921". TEXTRUNNER is able to extract the relation

("Einstein", "received", "Nobel prize")

Generally TEXTRUNNER has extracted hundreds of million of facts from a corpus of half billion web pages.

It has extracted 2000 Answers to the Query [What kills bacteria] with no predefined medical knowledge.

Some of the answers are

* antibiotics
* ozone
* chlorine etc

Template used in TEXTRUNNER

Type	Template	Example
Verb	NP_1 Verb NP_2	X Established Y
Noun-Prep	NP_1 NP prep NP_2	X Settlement with Y
Verb-Prep	NP_1 Verb Prep NP_2	X moved to Y
Infinitive	NP_1 to Verb NP_2	X plans to acquire Y
Modifier	NP_1 Verb NP_2 Noun	X is Y winner
Noun-coordinate	NP_1(, / and / -) NP_2 NP	X-Y deal
Verb-coordinate	NP_1(, / and) NP_2 Verb	X, Y merge

5.5 Natural language processing

Definition Natural language processing (NLP) is the ability of a computer program to understand human language as it is spoken and written ie in natural language.

It is a component of Artificial Intelligence

Phases of Natural language processing

1) Data preprocessing (2) Algorithm development

(1) Data preprocessing

It involves preparing and cleaning text data for machines to be able to analyze it

Some of the preprocessing tasks are

Tokenization : Text is broken down into smaller units

Stop word removal : Common words are removed from text so unique words that offer the most information about the text remain

Lemmatization and Stemming : words are reduced to their root forms to process

Parts-of-Speech tagging : words are marked based on the part-of-speech they are nouns, verbs and ~~etc~~ adjectives

NLP algorithms

Rule based [illegible] : This system uses carefully designed linguistic rules.

Machine learning based system:

Machine learning algorithms use Statistical methods

↓

(Combination of machine learning, deep learning, Neural networks & NLP algorithms)

Techniques of NLP

1. Syntax based
2. Semantic analysis based

1. Syntax based

'Syntax' is the arrangement of words in a sentence to make grammatical sense.

Syntax techniques include

Parsing – Grammatical analysis of a sentence

Eg: "The dog barked"

Parsing involves breaking this sentence into parts of speech i.e. dog = noun, barked = Verb

Word Segmentation – Act of taking a string of text and deriving words from it

Eg – The algorithm recognise that the words are divided by white spaces

Sentence breaking – This places sentence boundaries in large text through recognition of period

Morphological Segmentation – This divides words into smaller parts called morphemes

Eg. The word "untestably" would be broken into "un", "test", "able", "ly" & recognise them as morphemes

Stemming – Recognise the root of the word

Eg "barked" → "bark"

"Semantics" involves the use of meaning behind words. NLP applies algorithms to understand the meaning & structure of sentences

*Semantic techniques includes

Word Sense disambiguation - Derive the meaning of a word based on context.

Eg. 'The pig is in the pen"

The word "pen" has different meanings. In this context, 'Pen' here refers to a fenced in area, not a writing item

Named entity recognition - This determines words that can be categorized into groups

Eg. "Daniel McDonald's son went to McDonald's & ordered a Happy meal"

This algorithm could recognize the two instances of "McDonald's" as two separate entities one a restaurant and one a person

Natural language generation - This uses a database to determine semantics behind words and generate new text.

Eg- An algorithm automatically write a summary of findings from a business intelligence platform

Functions of NLP algorithms are

① Text classification: This involves assigning tags to text to put them in categories. This can be useful for sentiment analysis.

② Text extraction: This involves automatically summarizing text & finding important pieces of data eg. keyword extraction.

③ Machine Translation – computer program which translates text from one language (English) to another language such as French without human intervention

Real-world Applications of NLP

1) customer feedback analysis – AI analyzes social media reviews
2) customer service automation – use of voice assistants
3) Automatic translation – Google Translate, Bing Translator
4) Analysis & categorization of medical records
5) Word processors used for plagiarism & proofreading
6) Stock forecasting
7) Talent recruitment in human resources

5.6 Machine Translation

Definition – Machine Translation is the automatic translation of text from one natural language (source) to another (target)

Eg. Google Translate.

Machine Translation System

* All translation systems must model the source and target languages.

* Some systems analyse the source language and then generate sentences in the target language using interlingua knowledge representation

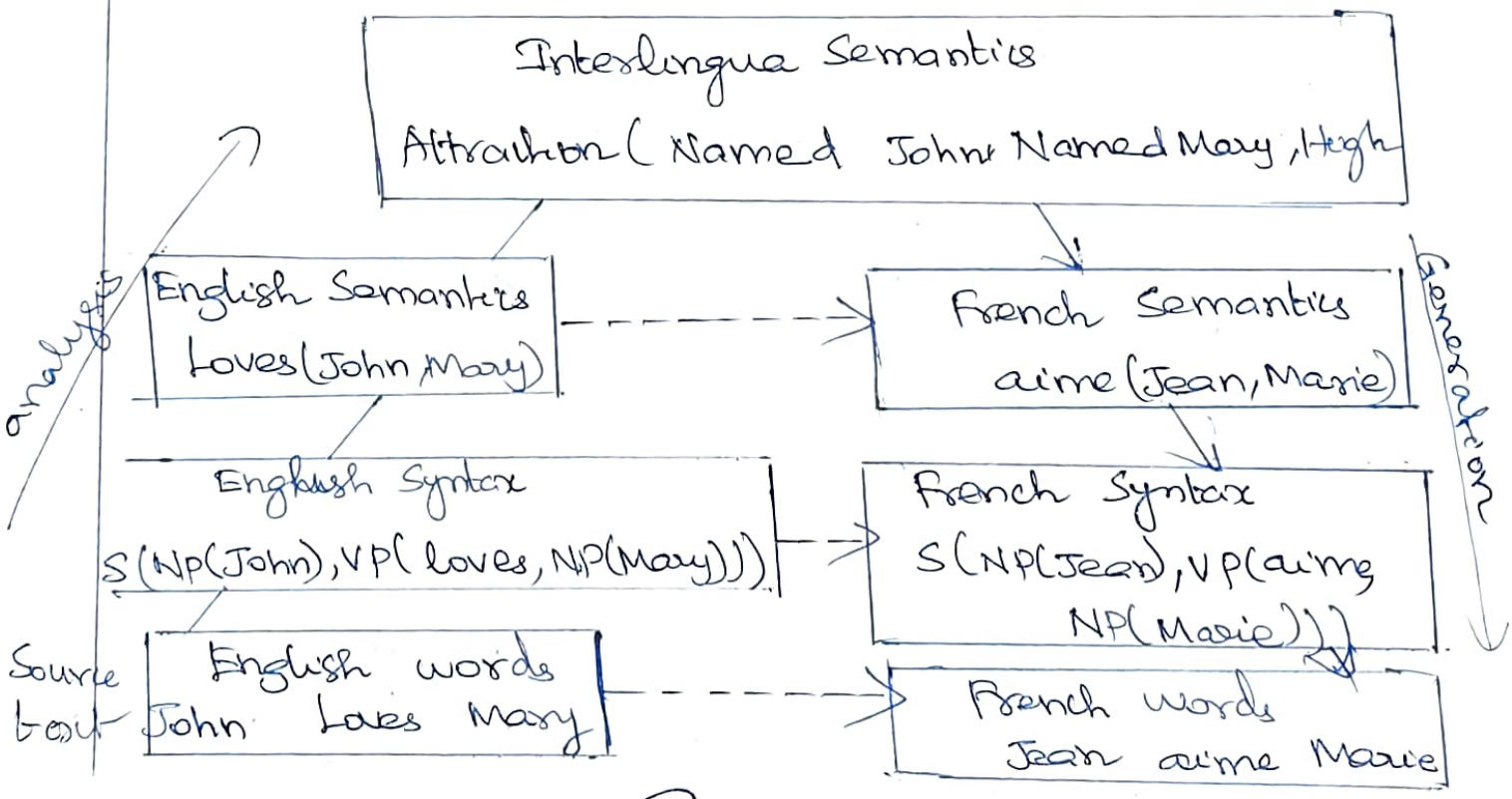
* Some systems are based on a transfer model. They use translation rules

* Translation occur at Lexical, Syntactic(or) Semantic level

Eg. English [Adjective noun] to French [Adjective Noun] mapping using Syntactic rules

French [S_1 "et puis" S_2] to English [S_1 and then S_2]

Vacquois' Pyramid



It starts with English text & end with French text.

Solid line ——— denote Interlingua based system

Dashed line ----- denote transfer based system

Interlingua System

1) Parses English into syntax → Semantics → Interlingu Representation

2) Generate Semantic → syntactic → lexical form in French

Machine Translation Approaches

Rule based:

* This approach includes transfer based machine translation, interlingual machine translation and dictionary based machine translation paradigms
  * It involves more information about the
    - linguistics of source & target languages
    - morphological, syntactic rules and Semantic analysis of both languages
* It involves linking the structure of i/p sentence with the structure of o/p sentence using a parser and an analyzer for the source language, a generator for the target target language
* To translate between closely related languages, rule based machine translation is used.

Transfer based MT

- It is similar to interlingual Machine Translation
- It creates a translation from an intermediate representation which simulates the meaning of the original sentence.

Interlingual

- It is one instance of rule-based Machine Translation
- Source language is transformed into an interlingual language, ie "language neutral" representation which is independant of any language
- Target language is then generated out of the interlingua.

Dictionary based

- It ~~uses~~ is based on dictionary entries

Statistical MT

- Generate translations using stastical methods based on bilingual text corpora such as Canadian Hansard corpos, English-French corpos

Example Based

In this approach, the corpus that is used is one that contains text that have already been translated

Hybrid MT - use the strengths of statistical & rule based translation approaches

Neural MT

- A deep learning based approach
- Google has announced its translation services using this technology

Statistical Machine Translation

Most successful machine translation systems are built by training a probabilistic model using statistics from large corpus of text.

It need sample translations from which a translation model can be learned.

To translate a sentence in English (e) into French (f), we find the string of words f^*

$$f^* = \text{argmax}\ p(f|e) = \text{argmax}\ p(e|f) \cdot p(f)$$

$p(f)$ – target language model for French

$p(e|f)$ – translation model from french to English

$p(f|e)$ – translation model from English to French

Given a source English sentence, e, finding a French translation f involves 3 steps:

1) Break the English sentence into phrases

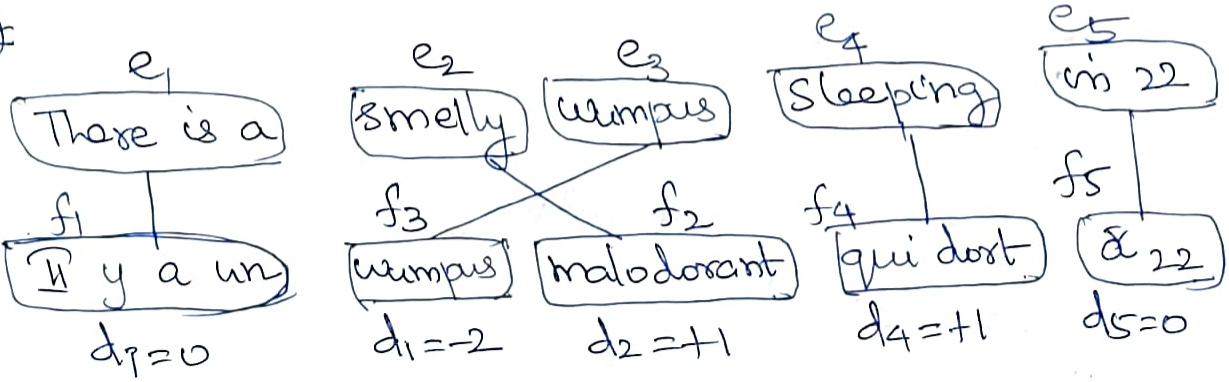
$e_1, e_2, \ldots, e_n$

2) Translate each phrase e_i into French phrase f_i. The phrasal probability of translation of e_i to f_i is denoted by

$p(f_i|e_i)$

3) Choose the permutation of phrase $f_1, f_2, \ldots f_n$
for each f_i choose a distortion
$d_i \rightarrow$ no of words that phrase f_i moved with respect to f_{i-1}
(+)ve for moving right
(−)ve for moving left
0 if f_i immediately follow f_{i-1}

Eg



The English sentence "Ther is a Smelly wumpus Sleeping in 22" is broken into 5 phrases e_1, e_2, e_3, e_4, e_5. Each of them is translated into corresponding f_i - $f_1, f_3, f_2, f_4 f_5$

Distortion d_i = Start(f_i) − End(f_{i-1}) − 1

Translation procedures

1) Find parallel text
2) Segment into Sentences
3) Align Sentences
4) Align phrases
5) Extract Distortions
6) Improve ~~esti~~ estimates with EM algorithm

5.7 Speech Recognition

It is the task of identifying a sequence words uttered by a speaker give the acoustic signal.

* It is the mainstream application of AI.
* Voice Assistant
* Voice Mail

- Sounds made by speaker are ambiguous and noisy

3 kinds of difficulties

1) Segmentation

In fast speech there are no pause in "wreck a nice" . So this will be confused with "~~Recognize~~" "Recognize"

2) Coarticulation

"Wreck ~~a~~ nice beach"

When speaking quickly the "s" sound at the end of "nice" merges with "b" sound at the beginning of "beach" . This yields "sp" .

3) Homophones

Words like 'to', 'too', 'two' that sound the same but differ in meaning

Defining speech recognition using Baye's rule

$$\text{argmax}\ P(\text{word}_{1:t} | \text{Sound}_{1:t}) = \text{argmax}\left(P(\text{Sound}_{1:t} | \text{word}_{1:t}) \, P(\text{word}_{1:t})\right)$$

Here $P(\text{Sound}_{1:t} | \text{word}_{1:t})$ — is the Acoustic model

$P(\text{word}_{1:t})$ - Language model

Noisy channel model:

An original message is transmitted over a noisy channel such that the corrupted message are received at the other end.

Acoustic Model

Phoneme:

A phoneme is the smallest unit of sound that has distinct meaning to speakers of a particular language.

Eg: 't' in 'stick' & 'tick' has the same phoneme

Frame: Represent the sound signal at a particular time

Features: Features from sound signal

Eg "French horns are playing loudly & violins playing softly"

The transition model defined by HMM

$$P(X_t | X_{t-1})$$

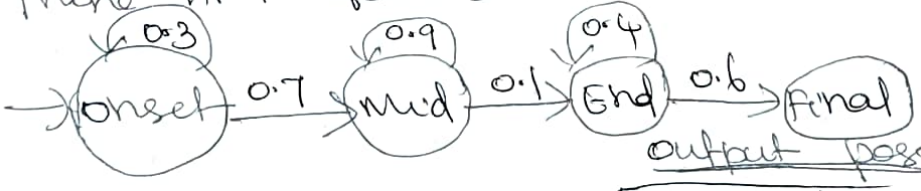
Transition model is broken into 2 levels

word & phone

The phone has 3 states

1) onset 2) middle 3) end

Phone Hmm for [m]:



output possibilities for the phone HMM

	onset	mid	End
c_1	0.5	c_3 0.2	c_4: 0.1
c_2	0.2	c_4 0.7	c_6: 0.5
c_3	0.3	c_5 0.1	c_7: 0.4

Pronunciation model for a word:

Consider the word "tomato" It has 2 pronunciation

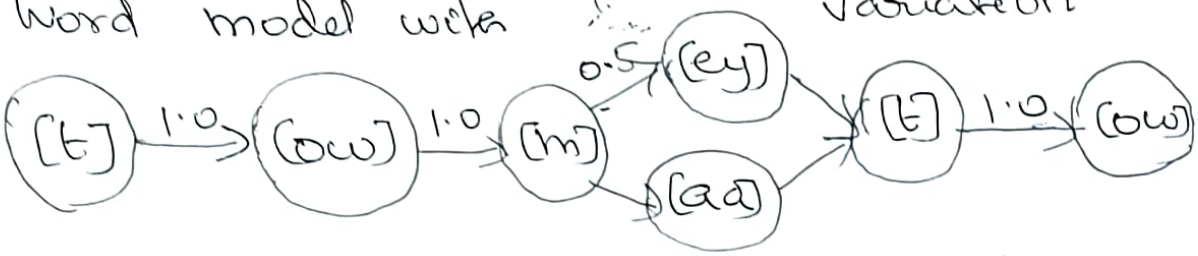
British [t ow m ey t ow]

American [t ow m aa t ow]

Each O denote the phone model

Transition model for dialect variation

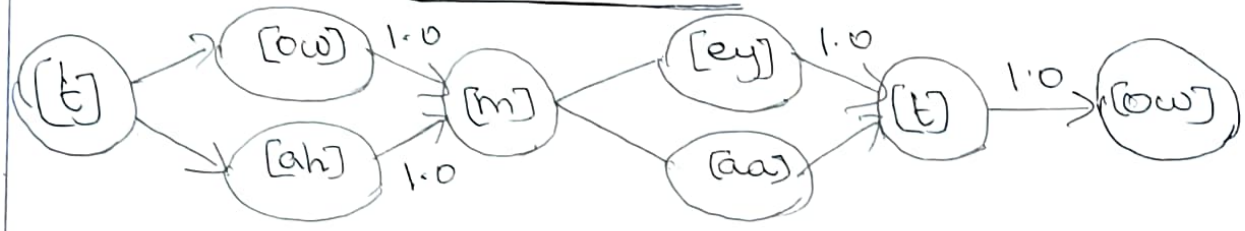
Word model with variation



Pronunciation Model 1 for "tomato"

Transition model for coarticulation variation

When speaking quickly, the tounge spells [t ah] rather than [t ow]. This variation is said to be a "coarticulation" variation



Two pronunciation models of the word "tomato"

Building a Speech Recognizer

The quality of speech recognition system depends on the quality of all its components.

1) Language model
2) Word - Pronunciation model
3) Phone models

Language Model - For speech recognition, the language can be an n-gram model of text learned from a corpus. Better use a corpus of transcripts of spoken language.

* Signal processing algorithms used to extract spectral features from the acoustic model
* Pronunciation models are developed by hand
* We use corpus of speech for speech recognition

5.8 Robotics - is a domain in artificial intelligence that deals with the study of creating intelligent robots.

Robots - Robots are active, artificial agents acting in real world environment. They are equipped with sensors and effectors to interact with physical world.

Sensors - Camera, IR sensors to perceive their environment

Effectors - to assert physical forces to the environment.

[Legs, wheels, joints]

Primary categories of Robots:

1. Manipulators (Robot arms)
2. Mobile Robots
3. Mobile Manipulators

① Manipulators (Robot arms)
- physically designed to their workplace

Eg. Factory Assembly line
- common type of Industrial Robots used within the workplace

② Mobile Robots
- Move about their environment using wheels, legs
- used for delivering foods in hospitals

Popular Mobile Robots
a) Unmanned Ground Vehicles (UGVs)
b) planetary Rover: (NASA Robots)
c) Unmanned Air Vehicles (UAVs)
d) Autonomous Underwater Vehicles (AUVs)

③ Mobile Manipulator
It combines mobility with manipulation.
Humanoid Robots - mimic the human torso
Eg SOFIA

Sq Robot Hardware
The success of real robots depends on the design of sensors and effectors

Sensors: Sensors are the perceptual Interface between robot and environment.

Passive Sensors:

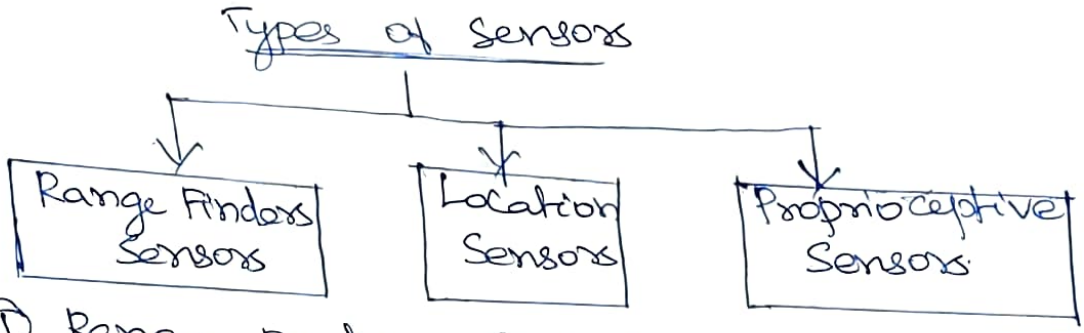
- Are the true observers of the environment

Eg - Camera

Active Sensors

- Send energy into environment & observe the feedback from the environment

Eg: SONAR



① Range Finders Sensors

It measures the distance to nearby objects

Sonars - Emit directional sound waves which are reflected by objects.

- used in Autonomous Underwater Vehicles

Stereo Vision - Relies on multiple cameras to image the environment from different viewpoints & then analyze the resulting image to compute the range of surrounding objects

Other range sensors

- time of flight camera
- Scanning Lidars
- Radar

② Location Sensors

* Global positioning system (GPS)
- It measures the distance to satellites which emit pulsed signals
* Differential GPS - GPS with differential value

③ Proprioceptive Sensors

Inform robot about its own motion.

* Shaft decoders - It counts the revolution of motors. It can be used for odometry ie measures the distance travelled
* Force & Torque sensors
- Force sensor sense how hard it is gripping the bulb
- Torque sensor sense how hard it is turning

Effectors - Effector is a device at the end of Robot arm, designed to interact with the environment

Degree of Freedom - a single motion an effector determine

⊙ Kinematic state of robot

A rigid mobile robot has 6 degrees of freedom 3 for x, y, z locations & 3 for angular orientation

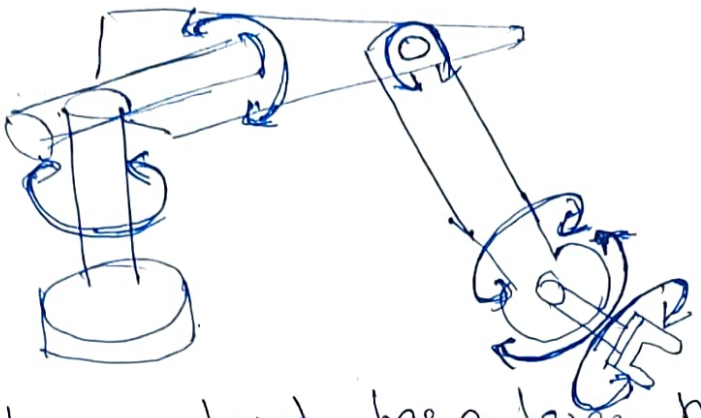
⊙ Dynamic state of Robot

⊙ Revolute joints - Generate rotational motion

⊙ Prismatic Joints - Generate Sliding Motion

Electric Motor - It is the most popular mechanism to drive the effectors

A PUMA Robot -Showing 6 rotary joints



A human hand has a large number of degrees of freedom.

S.10 Robot - Perception

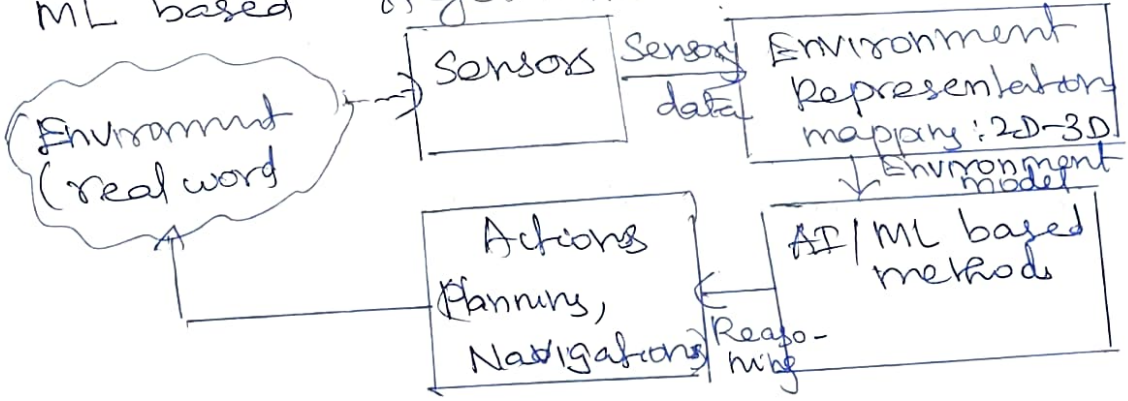
Perception - It is the process by which robots map sensor into internal representation of the environment.

Perception is difficult because

* Sensors are noisy &
* Environment is
  - Partially observable
  - Unpredictable
  - Dynamic

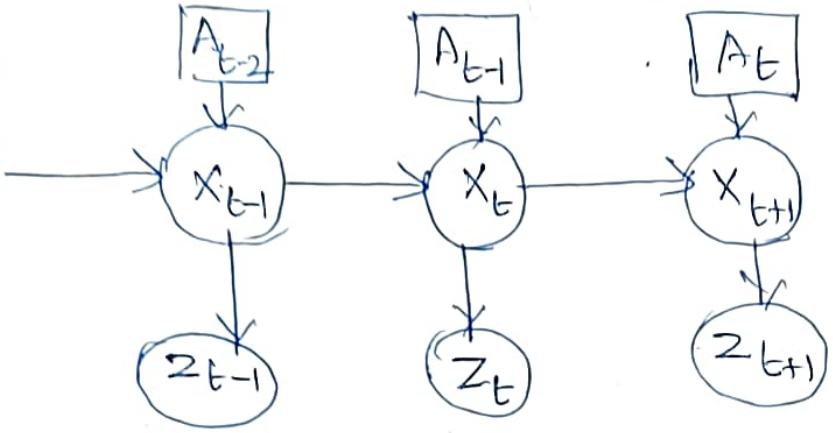
The key components of perception systems are

1) Sensory data processing
2) Data representation (Environment modeling)
3) ML based Algorithms



Robotic perception can be illustrated using a Bayesian Belief Network.

It can be defined as a temporal inference from sequences of actions and measurements



X_t – State of Environment at time t

Z_t – Observation received at time t

A_t – Action taken after the observation

Motion model (or) Transition model is

$$P(X_{t+1} | X_t, A_t)$$

Sensor model is

$$P(Z_{t+1} | X_{t+1})$$

Localization and Mapping

Localization – It is the problem of finding the things / objects.

* Robot manipulators must know the location of objects to manipulate them

A simplified kinematic model of a Mobile Robot

* The pose of the mobile robot is defined by its 2 cartesian coordinates with values x and y and the orientation θ

* Any state is defined using these 3 values as a vector

$$X_t = (x_t, y_t, \theta_t)^T$$

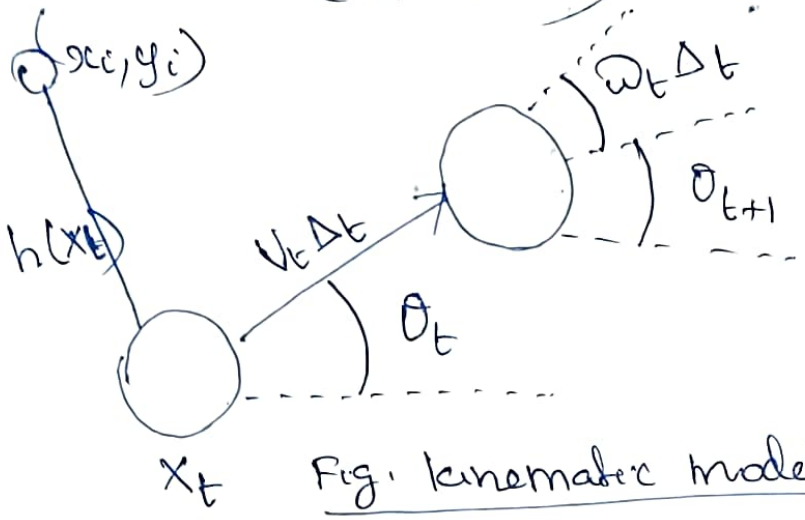


Fig. kinematic model of Mobile Robot

* current state X_t consists of (X_t, Y_t) & orientation θ_t
* New State X_{t+1} is obtained by the velocity $V_t \Delta t$ with orientation $\omega_t \Delta t$
* Each action consists of 2 velocities
  (i) Translational velocity V_t
  (ii) Rotational velocity ω_t

Sensor Model

Two types of sensor model for Mobile Robots

(i) Landmark model - Sensors that detect stable & recognizable features of the environment called Landmarks

Range Scan Model

- No need to identify the landmark
- Scan the object within a range

Eg: Four beam range scan & 2 possible Robot poses

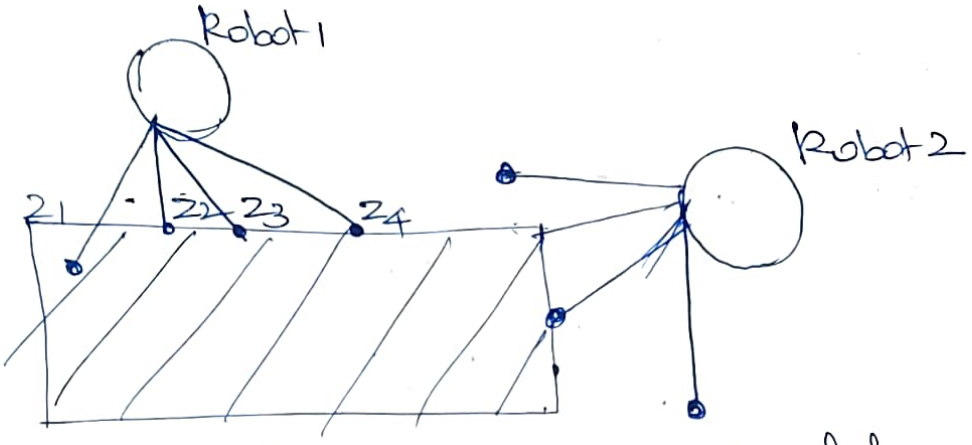


Fig. Range Scan Sensor model

- Range Scan (z_1, z_2, z_3, z_4)
- Robot 1 is much more likely than Robot 2

* Different ways to perform localization

(i) Localization using particle filtering is called Monte Carlo Localization

(ii) Kalman filter method

Other types of Perception

Robots also perceive the following

→ temperature
→ odours
→ Acoustic Signals etc

These are estimated using Dynamic Bayes Networks. They make use of conditional probability distribution

Machine learning in Robot perception

* ML plays an important role in robot perception
* **Low dimensional embedding**
  - It maps high dimensional sensor streams into low dimensional spaces using unsupervised Machine learning methods
* **Adaptive Perception Techniques** – enable robots to adjust to changes in the environment
* **Self Supervised** – These methods make robots to collect their own training data

5.11 Planning to Move

- All the robots plan "how to move effectors"

Types of Motion

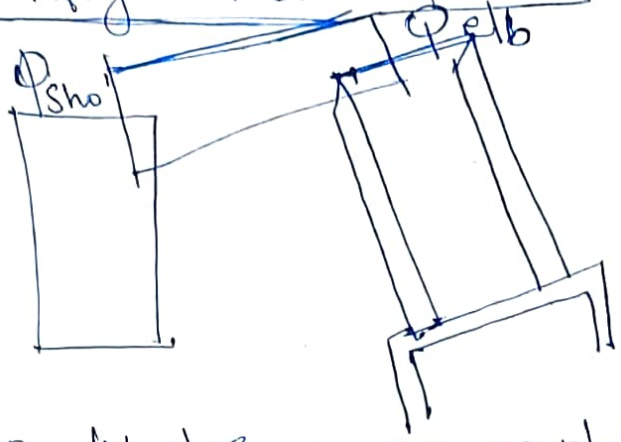
a. **Point to point**
   - Deliver robot to target location

b. **Compliant motion**
   Move while in contact to an obstacle
   [robot pushing a box]

Path planning problem

To find a path from one configuration to another in configuration space

Configuration space



- Robot arm has 2 joints that move independently
- Robot configuration is described by a 4 dimensional coordinate

Fig. Workspace representation of Robot Arm

(x_e, y_e) - represent the location of elbow
(x_g, y_g) - represent the location of gripper

Workspace Representation

The four coordinates (x_e, y_e) & (x_g, y_g) characterize the full state of the robot. Thus the 4 coordinates constitute the workspace representation

Configuration space representation

Robot state is represented by a configuration of robots joints

Generally robot has 2 points

(i) ϕ_s - Shoulder joint (ii) ϕ_e - Elbow joint

Kinematics

* Transformation of configuration space into workspace coordinates involves a series of coordinate transformations This chain of coordinate transformation is known as kinematics

Configuration space can be decomposed into 2 subspaces

(i) free space - space that a robot may attain

(ii) occupaid space - space of unattainable configuration

Cell Decomposition Methods:

* It is useful for path planning
* It decomposes the free space into finite no. of regions called cells

Adv: Simple to Implement

Limitation:- It works only for low-dimensional configuration spaces

Cell decomposition method is improved in two ways

(i) Subdivision of mixed cells until a path is found in the free space

(ii) Exact cell decomposition of the free space

Modified cost function:

potential field: It is a function defined over a statespace. Potential field value grows with the distance to the closest obstacle

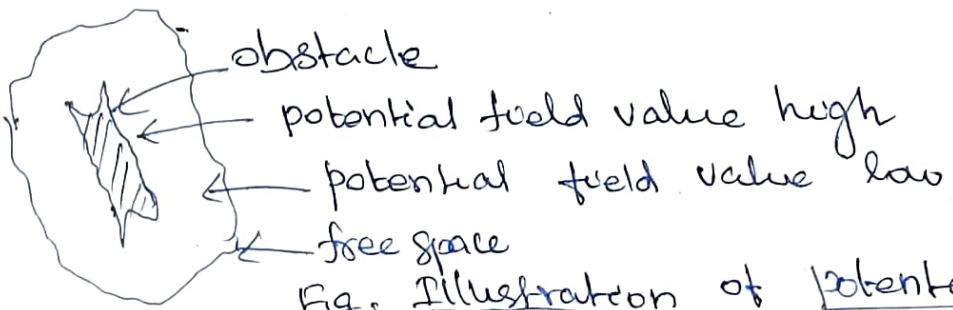


Fig. Illustration of potential field

Therefore, potential field can be used as an additional cost in the shortest-path calculation

Two objectives for finding the shortest path:

1) The robot seeks to minimize path length to the goal
2) The robot tries to stay away from obstacles by minimizing the potential function

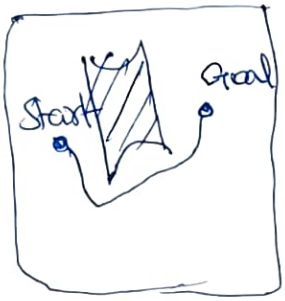


Fig. Shortest path by minimizing path length of the potential

Skeletonization Methods

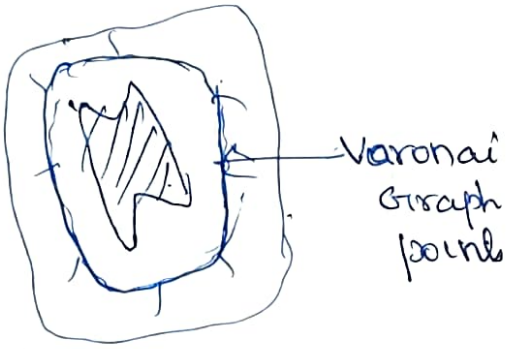
* It is useful for path planning algorithm
* These algorithms reduce the robots freespace to 1D representation. This makes the planning problem more easier
* This lower dimensional representation is called skeleton of the configuration space

Example of Skeletonization

* It generates a Voronoi graph of the free space

Voronoi Graph- It is a set of all points that are equidistant to two or more obstacles

* Path planning can be done using Voronoi graph. Robot changes its present configuration to a point on the voronai graph and it follows the graph until it reaches the point nearest to the target. Finally robot leaves the voronai graph & moves to the Target



Voronai Graph

Fig- Path planning using voronai graph

5.12 Moving

In the real world, Robots have inertia and cannot execute arbitrary paths except at arbitrarily slow speeds. In most cases, robots get to exert forces.

This Moving concept focuses on methods for calculating those forces

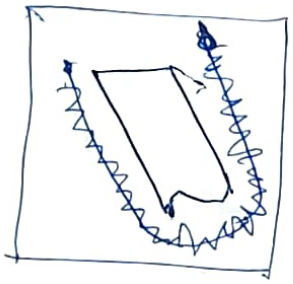
(i) Dynamics and control

A common technique is to use a controller mechanism for keeping the robot on track

If the objective is to keep the robot on a preplanned path, it is referred as "reference controller" & the path is called a "reference path". The controllers that optimize a global cost function are known as optimal controllers

Consider a problem of keeping a robot on a prespecified path. Now the controller causes the robot to vibrate rather violently. The vibration is the result of a natural inertia of robot arm

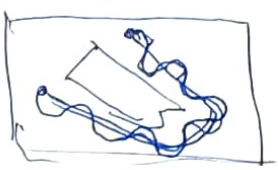
Robot arm control!



Proportional control with gain factor 1.0.
-This vibrates violently

* P controllers: Controllers that provide force in negative proportion to the observed error are known as P controllers

* A reference controller can be
(i) Stable: if small perturbations lead to a bounded error between robot and reference signal
ii) Strictly Stable: if it is able to return to and stay on its reference path

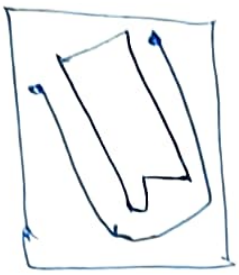


Proportional control with gain factor of

PD controller

P → Proportional D → Derivative

* This controller achieves Strict Stability



– PD controller with gain factor 0.3 for P & 0.8 for D

PID controller

* PID – Proportional Integral Derivative
* PID - controllers are widely used in industry for a variety of control problems

Potential field control

* It is used for generating robot motion directly
* Two things are defined to achieve a goal
  (1) Attractive force: Pulls the Robot towards its goal configuration
  (2) Repellant potential field: That pushes the robot away from obstacles

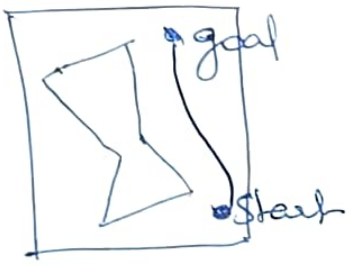


Fig- Successful path to goal

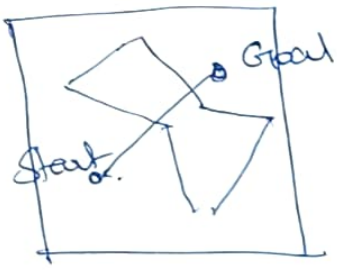


Fig- Local optimum to goal

* Potential field control is suitable for local robot motion

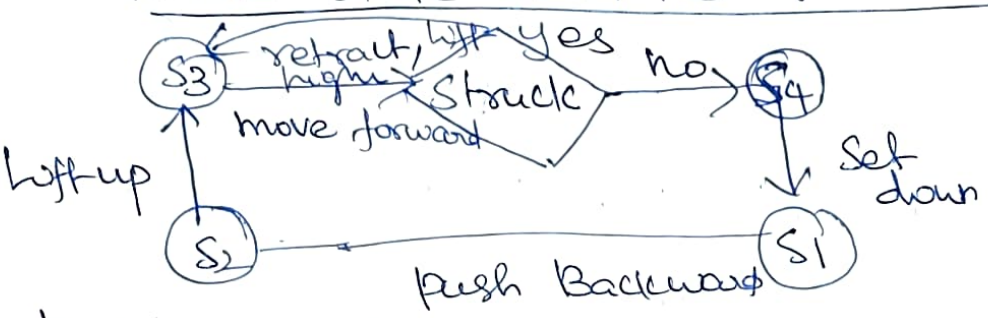
Dis adv: Potential field control fails if the robot is moving quickly

Reactive Control: For constructing a reference path a reflex agent architecture using reactive control is used

Eg: Consider a six-legged robot that tries to lift a leg over an obstacle

Rule: "Lift the leg a small height h and move it forward, and if the leg perceives an obstacle move it back & start again at a higher height"

Finite State machine for a Reflex Agent



Reinforcement learning control

It is based on policy search. It is a form of Reinforcement learning. It has solved challenging Robotics problems

Eg. Acrobatic Autonomous Helicopter flight

A policy search method with a policy can safely execute a flip everytime.