Ch-1 Artificial Intelligence

What is Intelligence?

Intelligence taken as a whole consists of the following tasks/skills:

1. The ability to reason
2. The ability to acquire and apply knowledge
3. The ability to manipulate and communicate ideas

Artificial Intelligence:

1. Artificial intelligence is the study of computations that makes it possible to perceive, reason and act.
2. AI is the study of making computers do things at which at present human beings are better.
3. Intelligence based
4. AI is the area of computer science concerned with designing intelligent cs systems i.e systems which exhibit the characteristics which we usually associate with human behaviour.
5. Symbolic, non algorithmic
6. AI is the branch of computer science dealing with symbolic, non algorithmic methods of problem solving
7. Knowledge representation
8. AI is the branch of computer science that deals with the ways of representing knowledge using symbols rather than numbers and with rule of thumb for representing the information.

Some Arguments.....

- Most people agree that AI concerns with 2 basic ideas.
  - Studying the thought processes of human bei (how our mind works???)
  - Studying how to represent these processes in computers or machines.
- AI's def: Artificial intelligence is the Stud of how to make computer do things at whi at the moment, people are better
- "AI is behaviour by a machine that, it performed by a human being, would be called intelligent!!!"

Intelligent Behaviour

- Learn or understant from experience
- Make sense out of ambigous or contradictory message.
- Respond quickly and successfully to a new situation.
- use reason in solving problems and directing conduct effectively.
- Deal with perplexing situations
- Understand and infer in ordinary, rational ways.
- Apply knowledge to manipulate the environ
- Acquire and apply knowledge
- Think and reason

Introduction to AI: Turing's Test

- In 1950 Alan Turing published his new famous Paper "Computing Machinery and Intelligence". In that paper he describes a method for humans to test AI Programs
- In its most basic form, a human judge sits at a computer terminal and interacts with the subject by written communication only. The judge must then decide if the subject on other end of Computer link is a human or an AI program imitating a human
- http://www.turing.org.uk/

Intelligent vs Natural Intelligence

Artificial Intelligence	Natural Intelligence
• AI is more permanent	• Human can have Creative thought
• AI offers ease of duplication	• Human can use Sensory experience rather than Symbol representations like AI
• AI can be less expensive	• Human can use wide context of experience when solving a problem while AI work well only with narrow domain
• AI allows consistency	
• AI can be documented	
• AI can do a certain task must faster than human	

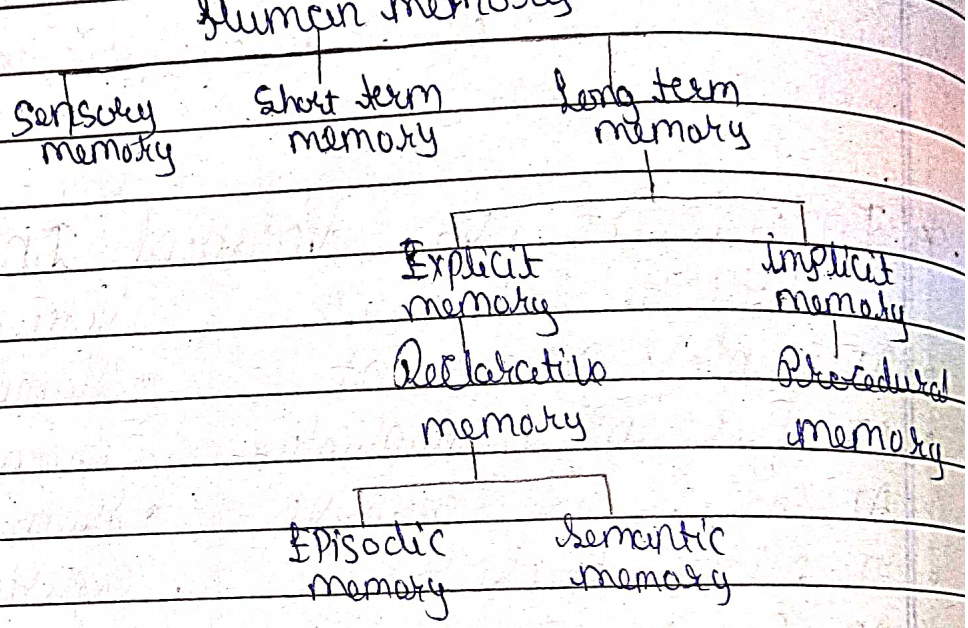
AI vs Conventional Computing

Dimension	AI	Conventional
Processing	Mainly Symbolic	Mainly algorithmic
Input	Incomplete	Complete
Search	Heuristic	Algorithms
Explanation	Provided	Usually not provided

• Output	Can be incomplete	Must be correct
• Reasoning Capability	Limited but can improve	None

Types of Memories

- Memories is defined as encoding, storage and retrieval of an experience
- In a simpler meaning, it is a recollection of the past



Short Term Memory

- A short term memory has the time span ranging from seconds to a few minutes
- They can stored temporarily and then either get to long term memory storage or they are discarded
- E.g - Someone gives us the mobile number

Long Term Memory

- These are the memories stored in brain

over a more extended period
- Memories are sometimes organized to form a long memory that is stored permanently.
- Depending upon the importance and no of details it can fade or remembered forever.
- Eg what you did on your 16th birthday.

Applications of AI
- Game playing
- Natural Language processing
- Expert Systems
- Robotics / Intelligent Agents
- Neural Networks

1. Game playing
a) Properties which make games as an ideal subject to research
i) Use pre defined set of rules
ii) Easy to play → Easy to develop and test
iii) No Expert knowledge is required
iv) No financial or ethical burden
v) problems are very structured

2. Expert Systems
- More specialized AI systems which are expert in their domain
- Expert knowledge is required to develop expert systems
- Expert knowledge is a combination of theoretical understanding of the problem and collection effective problem solving rules
- Eg Medical diagnostic System

Automated Reasoning and Theorem Proving

- Mathematical problems need some level of logic to solve the problems
- Capable to solve real world complex problems
- Complex problems are divided into sub-problems to address each sub problem and then combining the solutions to reach the final solution
- Concerned with decision making

Artificial Neural Networks

- Emerged from biological Neural networks
- Simulation of biological Neural networks.
- Tries to impose the capability of architecture of human brain in AI systems to make systems as intelligent as human beings
- Distributed systems comprising huge no of units connected with each other to behave like human brain Connectionism
- These are called Connectionist models

History of AI

- 1614: Scot mathematician → John Napier
  Basic calculator for addition and subtraction with overflows
- 1620: Francis Bacon
  Search technique for determining an entity from its features
- 1642: French philosopher and Mathematician
- Blaise Pascal → Pascaline
- Calculating machine for addition and subtraction

with barrows and Lathes.

- 1694: Mathematician Leibnitz:
  Leibnitz machine -> Lib wheel inspired by pascaline
  Complex mathematical representation multiplication and division.
- 1642 to 1936: Modern technology
  Charles Babbage: Analytical engine
  Anal Turing: NLP machine for test
- 1840: Joseph Faber
  Invented first talking machine Euphonia
  David Lindsay: Wrote a papers on early "talking devices" Talking Head Invention and Technology Magazine".
- 1950: Claude Shanon
  Published a paper predicting that a day will come when computers will play chess
  Year 1950 is also remembered as year of Turing test.
- 1957: Newell and Simon
  Predicted that chess playing computers will beat human beings.
- 1997: Deep Blue
  Chess playing Computer -> IBM
- 1996: Garry Kasparov vs Deep Blue (6 game match) Computer 4-2
- 1997: Deep Blue vs Garry Kasparov (6 game match) DB 3-2
  Superior intelligence

Goals of AI

- Engineering goal consists of
  - Using AI as armature of ideas
  - To Solve real world problems
  - Adress problems in domain
    Design, Fault finding, Planning, Manufacturing, Marketing, Customer Satisfa

What are AI Problems

- Aid in thinking, planning, decision making
  - Qualitative Solutions instead of Quantitative
  - Exhibit intelligence, patience, incomplete ingu
  - Learn from examples
    - Work effectively and efficiently
    - Solve real world complex problems

Characteristics of AI problems

- Complex
  - Result oriented
  - Requires huge amount of knowledge
  - Voluminous in nature
  - Hard to characterize accurately
  - Constantly changing

AI Technique

Methods which exploit knowledge with above characteristics

Data Structures: Tic Tac Toe Board

A nine elements vector representing the board, where the elements of vector correspond to the board positions as follows:

1 2 3
4 5 6
7 8 9

An element contains the value of 0 if the corresponding square is blank, 1 if it is filled with an X or 2 if it is filled with an O. A large vector of 19,683 elements (3^9) each element of which is a nine element vector. The contents of this vector are chosen specifically to allow algorithm to work.

The Algorithm:

To make a move, do the following:

(i) View the vector board as ternary (base three) number. Convert it to a decimal number.

(ii) Use the number computed in step 1 as an index into a moveTable and access the vector stored there.

(iii) The vector selected in step 2 represents the way the board will look after the move that should be made. So set board equal to vector.

Conversion from ternary to decimal

Ternary 2 0 0 1 0 0

No positions 4 3 2 1 0

3^4 3^3 3^2 3^1 3^0

$\therefore 2\times3^4 + 0\times3^3 + 1\times3^2 + 1\times3^1 + 0\times3^0 = 174$

Q. What is Artificial Intelligence

- It takes a lot of space to store the table that specifies the correct move to make from each board position
- Someone will have to do a lot of work specifying all entries in movetable
- It is very unlikely that all the required movetable entries can be determined and entered without errors
- If we wanted to extend the game, say to 3 dimensions we would have to start from scratch and in fact this technique would no longer work at all, since 3^{27} positions would have to be stored.

The technique embeded in this program does not appear to meet any of our requirements for a good AI technique. Let's see if can we do better

Program 2

This program is identical to Program 2 except for 1 change in representation of board. We again represent the board as a 9 element vector, but this time we assign board positions to vector elements as follows

8	3	4
1	5	9
6	7	2

Notice that this numbering of board produces

a magic square all the rows, columns and diagonals sum upto 15. This means that we can simplify the process of checking for possible win. In + to marking the board as moves are made, we keep a list for each player of the squares in which he or she has played. To check for a possible win for 1 player we consider each pair of squares owned by player and compute the diff b/t 15 and sum of 2 squares were not collinear and so can be ignored. otherwise if square representing diff is blank a move there will produce a win since no player can have more than squares at a time, there will be many fewer squares examined using this scheme.

Data Structures

Board - A nine element vector representing the board as described in prog 1. But instead of using no 0, 1, or 2 in each element, we store 2, 3 or 5.

Turn - An integer indicating which move of the game is about to be played. 1 indicates 1st move, 9 the last.

The Algorithm

The main algorithm uses 3 subprocedures

- Make 2 - Returns 5 if the center square of board is blank. that is if Board[5] = 2. Otherwise this function returns any blank Noncorner square (2, 4, 6 or 8)
- Posswin (p)

Returns 0 if player p constitutes a winning move. Otherwise it returns the no of square that constitutes a winning move. This function will enable the program

both to win and to block the opponents win. possWin operates by checking one a time, each of the rows and columns, diagonals BOZ of the way values are numbered, it can test an entire row to see if it is a possible win by multiplying the values of its squares together. If the product is 18 (3x3x2) then X can win. If the product is 50 (5x5x2) then O can win. If we find a winning row, we determine which square is blank and return the no of that square.

Make 2

If Board[5] = 2
    Return 5
Else
    Return 2, 4, 6, 8

Test entire row, column and diagonal by
3x3x2 = 18 → X can win 5x5x2 = 50 → O can win
Return 2, 4, 6, 8

Technique 2:

To find ¶
Test entire row, column and diagonal by
Find occurance of X or O
Mark the board which show occurance of X or O
Compute is - (pair of tiles occupied with X or O)
[Board[8] + Board[3] + Board[4] = 15]
possWin P1 → X
P2 → O
If diff is -ve or diff > 9
Then the tiles are non collinear and

Can be ignored
Else
Return 2(9 is diff^ representing title index which is blank)

* Technique 3
Minimax Procedure
Minimizing - possibility of my opponents loose
Maximizing - possibility of my opponents win

* Data Structures
Board position - A structure containing a nine element vector representing the board, a list of board position that could result from the next move, and a no representing an estimate of how likely the board positions is to lead an ultimate win for player to move.

• The Algorithm
To decide on next move, look ahead at a board positions that result from each possible move. Decide which position is best, make the move that leads to that position and assign the rating of that best move to current position.
To decide which of a set of board position is best, do following each of them.
1 See if it is a win. If so call it the best do the following for each other.
2 Otherwise consider all the moves the opponent could make next. See which of them is worst for us. Assume the opponent will make that move whatever rating that move has, assign it to the node we are considering

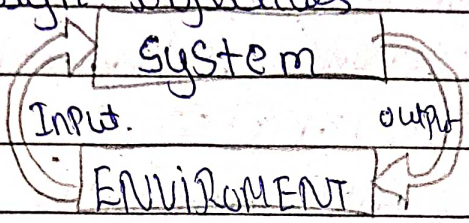
3 The best node is then the one with highest rating.

This algorithm will look ahead at various sequences of moves in order to find a sequence that leads to a win. It attempts to maximize the likehood of winning while assuming the opponent will try to minimize that likelihood. This algorithm is called the minimax procedure and it is discuss in detail.

Ch-2 Intelligent Agents

What is an Agent?

- Agents are autonomous: capable of acting independently, exhibiting control over their internal state.
- Thus an agent is a computer system capable of an autonomous action in some environment in order to meet its design objectives.



Structure of Intelligent Agents

- An intelligent agent
  - perceives its environment
  - through its sensors.
  - then achieves its goals
  - by acting on its environment via actuators

Examples of Agent :

- Agent - mail sorting robot
- Environment: conveyer belt of letters
- Goals: route letter into correct bin
- Percepts: array of pixels intensities
- Actions: route letter into bin

2.
- Agent: Intelligent house
- Environment: occupants enter and leave house
  - occupants enter and leave rooms.

- daily variation in outside light and temperature.
- Goals: occupants warms, rooms light when occupied, house energy efficient
- Percepts/Sensors: Signals from temperature sensor, movement sensor, clock, sound sensor
- Actuators: room heaters on/off, lights on/off

3
- Agent: automatic car
- Environment: Streets, other vehicles, pedestrians, traffic signals/lights/signs
- Goals: Safe, fast, legal trip
- Percepts - Camera, Gps signals, Speedometer, Sonar
- Actions: steer, accelerate, brake

What is an Agent?

- An intelligent agent is a computer system capable of flexible autonomous actions in some environment to achieve the goals
- By flexible, we mean:
- reactive • pro-active • social

Reactivity

- If a program's environment is guaranteed to be fixed, the program need never worry about its own success or failure.
  Program just executes blindly
- The real world is not like that, things that change information is incomplete. Most environme

are dynamic
- Software is hard to build for dynamic domains
- A reactive system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it.

Proactiveness
- Reacting to an environment is easy
- But we generally want agents to do things for us
- Hence goal directed behaviour
- Pro activeness = generating and attempting to achieve goals; not driven solely by events taking initi
- Recognizing opportunities

Balancing Reactive and Goal Oriented Behaviour
- Two expectations from agents can be at odds with one another:
- To be reactive, responding to changing conditions in an appropriate fashion
- To be systematic, working towards long term goals
- Designing an agent that can balance 2 remains an open research problem

Social Ability
- The real world is a multi agent environment
- We cannot go around attempting to achieve goals without taking others into account
- Some goals can only be achieved with the co operation of others

- Social ability in agents is the ability to interact with other agents via some kind of agent communication language, and perhaps cooperate with others.

Other Properties

- Mobility - the ability of an agent to move around an electronic network
- Veracity - an agent will not knowingly communicate false information
- Benevolence - agents do not have conflicting goals and that every agent will therefore always try to do what is asked of it
- Rationality - agent will act in order to achieve its goals and will not act in such a way as to prevent its goals being achieved at least insofar as its beliefs permit
- Learning / adaption - agents improve performance over time

Agents and objects

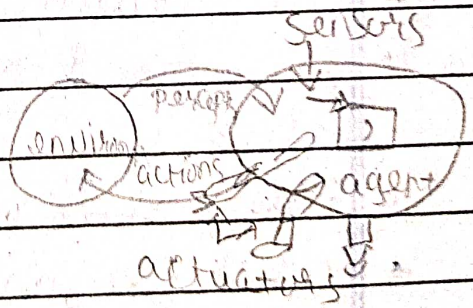
- Are agents just objects by another name?
- Object:
- encapsulates some state
- communicates via message passing
- has methods, corresponding to operations that may be performed on this state

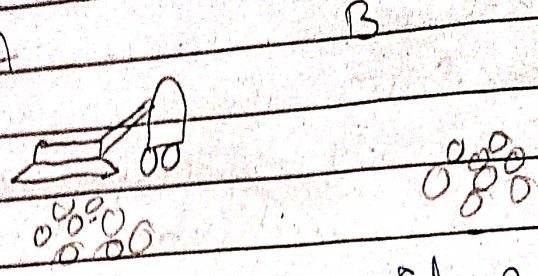
Main Difference

- **Agents are autonomous**
  agents embody stronger notion of autonomy than objects, and in particular, they decide for themselves whether or not to perform an action on request from another agent.
- **Agents are smart**
  Capable of flexible (reactive, proactive, social) behaviour, and the standard object model has nothing to say about such type of behaviour.
- **Agents are active**
  A multi agent system is inherently multi threaded in that each agent is assumed to have at least one thread of active control.

Agents and Environment

- Agents include human, robots softbots, thermostats etc
- The agent function maps percept sequence to actions
  $f : P^* \rightarrow A$
- An agent can perceive its own actions, but not always its effects
- The agent function will internally be represented by agent program
- The agent program runs on physical architecture to produce f.





- Goal: To keep the room clean
- Enviroment: Square A and B
- Percepts: [location and content] eg [A, Dirty]
- Actions: left, right, suck and no op

function REFLEX-VACUUM-AGENT return an action

if status == Dirty then return Suck
else return no op
if location == A then return right
else return left

what is right function? Can be implemented in a small agent program?

The Concept of Rationality

- A rational agent is one that does the right thing.
- Every ~~thing~~ entry in the table is filled out correctly
- what is right thing?
- Approximation: the most successful agent
- Measure of Sucess?
  - Performance measure should be objective
  - E.g the amount of dirt cleaned with a certain time
- E.g - how clean the floor is
  - Performance measure according to what is

wanted in the environment instead of how the agents should behave

Rationality

- What is rational at a given time depends on four things
- performance measure
- prior environment knowledge
- Actions
- percept sequence to date (sensors)
- A rational agent chooses whichever action maximize the expected value of the performance measure given the percept sequence to date and prior environment knowledge
- Rationality ≠ omniscience
- An omniscient agent knows the actual outcome of its actions
- Rationality ≠ perfection
- Rationality maximizes expected performance, while perfection maximizes actual performance
- The Propose Refination Requires
- Information gathering / exploration
- To maximize future rewards
- Learn from percepts
- Extending prior knowledge
- Agent autonomy
- Compensate for incorrect prior knowledge

Environments

- To design a rational agent we must spec its task environment
- PEAS description of the environment
- Performance · Environment · Acutators · Sensors

PEAS Description

- Eg: Fully Automated Taxi
- Performance: Safety, destination, cost effic legality, comfort.
- Environment: Street / freeways, other traffic, pedestrians, animals, weather.
- Acutators: Steering, accelator, brake, horn speaker / display
- Sensors: Video, speedometer, engine sensors, mic / keyboard, GPS ---

Environment Types

	Solitare	Backgammon	Internet shopping	Taxi
Observable??	FULL	FULL	PARTIAL	PARTIAL
Determinable??	yes	No	yes	No
Episodic?	No	No.	No	No
Static??	yes	yes	yes	No
Discrete??				
Single agent??				

- Episodic vs Sequential: In an episodic envir the agents experience can be divided into atomic

Steps where the agents perceives and then performs A single action. The choice of action depends only on the episode itself.

- Static vs dynamic: If the environment can change a while the agent is choosing an action the environment is dynamic. Semi dynamic if the agent's performance changes even when the environ remains the same.
- The Singlest Enviroment is
- Fully observable, deterministic, episodic, Static discrete and Single agent.
- Most real Situations are
- partially observable, Stochastic, Sequential, dynamic, Continous and multi agent

Agent Types

- How does the inside of the agent Work?
- Agent = architecture + program
- All agents have the same Skeleton:
- Input = Current percepts
- Output = action
- Program = manipulates input to produce output

The different in agent function differs the agents from one other.

Function: TABLE_DRIVEN_AGENT (percept)
returns an action
Static: percepts, a sequence, initially empty
table, a table of actions, indexed by percept

Sequence
Append percept to the end of percepts
actions ← LOOKUP (percepts, table)
Return action.

- Four basic kind of agent programs will be discussed:
- Simple reflex agents
- Model based reflex agents
- Goal based agents
- Utility based agents
- All these can be turned into learning agents

PEAS

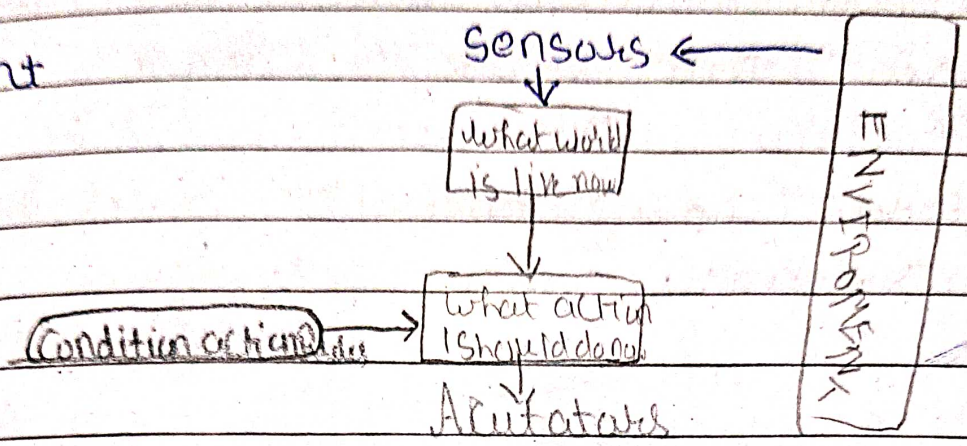
1. E.g Mail Sorting Agent
P = Accuracy, Precision, time, efficient, work
E = Confidental documents, heavy parcels
A = Containers, Actuators, Robotics, Conveyer Belt
S = Camera, Scanner, Sowing Machine

2. E.g Vacumm Cleaner
P = Cleanliness, efficiency, distance travel to clean, mobility
E = Room, table, floor, carpet, furniture, electronic
A = wheels, diff brushes, vacumm extractors
S = Camera, dist detect sensor, bump sensors

Simple Reflex Agents

- Action depends only on immediate percepts.
- Implement by condition action Rules
Example.

- Agent: Mail Sorting robot
- Environment: Conveyer belt of letters
- Rule: eg - Area = Navrangpura → put in green bag.



- Select action on basis of current percept.
  E.g the Vacuum agent.
- Large reduction in possible percept/action situations
- Implemented through condition action rules
- If dirty then suck

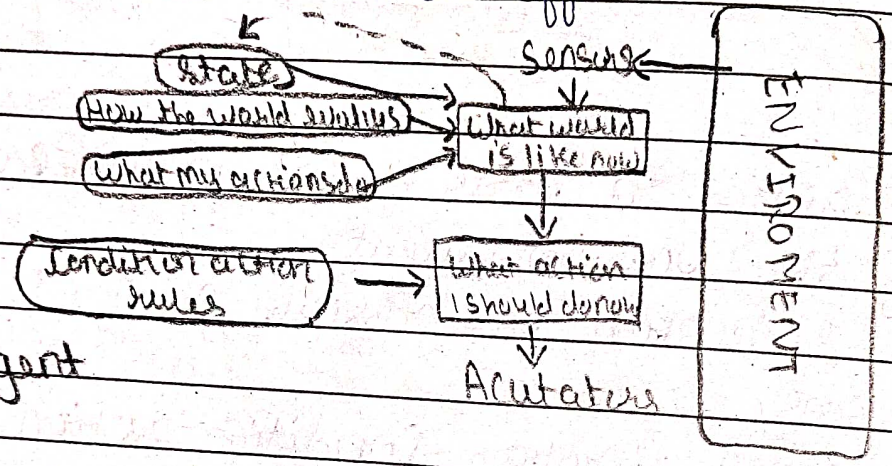
Reflex Agents

function SIMPLE-REFLEX-AGENT returns an action
static: rules, a set of condition action rules
state ← INTERPRET-INPUT(percept)
rule ← RULE-MATCH(state, rule)
action ← RULE-ACTION(rule)
return action

Will only work if the environment is fully observable otherwise infinite loops may occur

Model-Based Reflex Agents

- Action may depend on history or unperceived aspects of the world
- Need to maintain internal world model. E.g.
- Agent: Robot vacumm cleaner
- Environment: dirty room furniture
- Model: map of room which areas already cleaned
- Sensor/model trade off.



function REFLEX AGENT-WITH-STATE returns an acti
Static: rules a set of condition action rules
state, a description of current world state
action, the most recent action
State ← UPDATE-STATE (state, action, percept,
Rule ← RULE-MATCH (state, Rule)
action ← RULE-ACTION [rule]
return action

Goal Based Agents

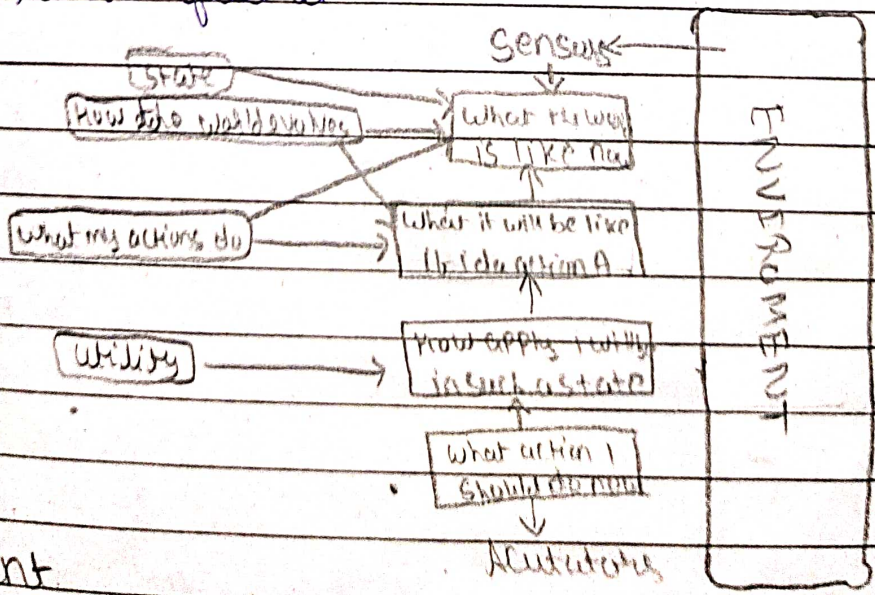
- Agents so far have fixed, implict goals
- We want agent with variable goals
- Forming plans to achieve goals is later topic

Example:
- Agent: Robot maid
- Environment: house and people
- Goals: clean clothes, tidy room, table laid etc

Fig. Same as Model Based Reflex agents

Utility Based Agents

- Agents so far have had a single goal
- Agents may have to juggle conflicting goals
- Need to optimise utility over a range of goals
- Utility: measure of goodness (a real no)
- Combine with probability of sucess to get a expected utility
- Example
- Agent: automatic car
- Environment: roads, vehicles, signs etc
- Goals: stay safe, reach destination, be quick, obey law, save fuel etc.



Agent

- Certain goals can be reached in different ways. Some are better, have a higher utility

- Utility function maps a (sequence of) states onto a real number.
- Improve on goals
- Selecting between conflicting goals
- Select appropriately between several goals based on likelihood of success

Ch-3 Problems, problem Spaces And Search

Defining The Problem As a State Space Research

Suppose we start with the problem statement "play Chess". Although there are a lot of people to whom we could say that and reasonably expect that they will do as we intended, as our request now stands it is a very incomplete statement for the problem we want solved. To build a program that could "play Chess" we would first have to specify the starting position of the chess board, the rules that define the legal moves, and the board positions that represent a win for one side of the other. In addition we must make explicit the previously implicit goal of not only playing a legal game of chess but also winning the game, if possible.

For the problem "play chess", it is fairly easy to provide a formal and complete problem description. The starting position can be described as an 8x8 array where each position can contains a symbol standing for the appropriate piece in the official chess opening position. We can define as our goal any board position in which the opponent does not have a legal move and his or her king is under attack. The legal moves provides the way of getting from initial state to a goal state. They can be described as a set of rules consisting of two parts a left side that serves as a pattern to be matched against the current board position and a right side that describes

the change to be made to the board position to reflect the move.

However if we write rules like the one above we have to write a very large no of them since there has to be separate rule for each of the roughly 10^{120} possible board positions.

- No person could ever supply a complete set of such rules. It could too long and could certainly not be done without mistakes
- No program could easily handle all those rules. Although a hashing scheme could be used to find the relevant rules for each move fairly quickly, just storing that many rules poses serious difficulties.

In order to minimize such problems, we should look for a way to write the rules describing the legal moves in a general way as possible. To do this it is useful to introduce some convenient situation for describing patterns and substitutions for e.g the rule described in as well as many like it, could be written In general, the more succinctly we can describe the rules we need, the less work we will have to do provide them and more efficient the program that uses them can be.

White pawn at
Square (file e, rank 2)
Square AND
Square (file e, rank 3) → move pawn from Square (file e, rank 2) to Square (file e, rank 4)
is empty AND
Square (file e, rank 4) is empty

We have just define the problem of playing chess as a problem of moving around in a state space, where each state corresponds to a legal position of board. We can play the chess by starting at an initial state using a set of rules to move from one state to another and attempting to end up in one of a set of final states. This state space representation seems natural for chess bcz the set of states, which corresponds in the set of board positions, is artificial and well organized. This same kind of representation is also useful for naturerality occuring less well structured problems. Although it may be necessary to use more complex structures than to be complete by accurate, this rule should include a check for pinned pieces, which have been ignored here.

* State Space Representation:

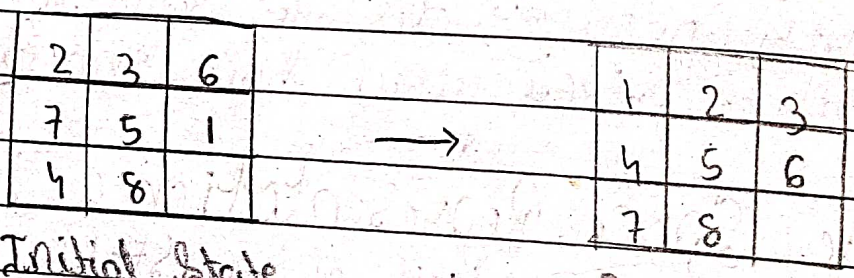
A representation of a states of a problem where each state corresponds a legal position on the board.

In state space representation of a problem each node of the tree corresponds to a partial problem solution state and the arcs correspond to steps in problem solving process.
The tree also represents

- Initial state corresponding to given information about the problem which forms root node of tree
- Goal state(s) which are possible acceptable solutions of the problem which forms leaf node of the tree

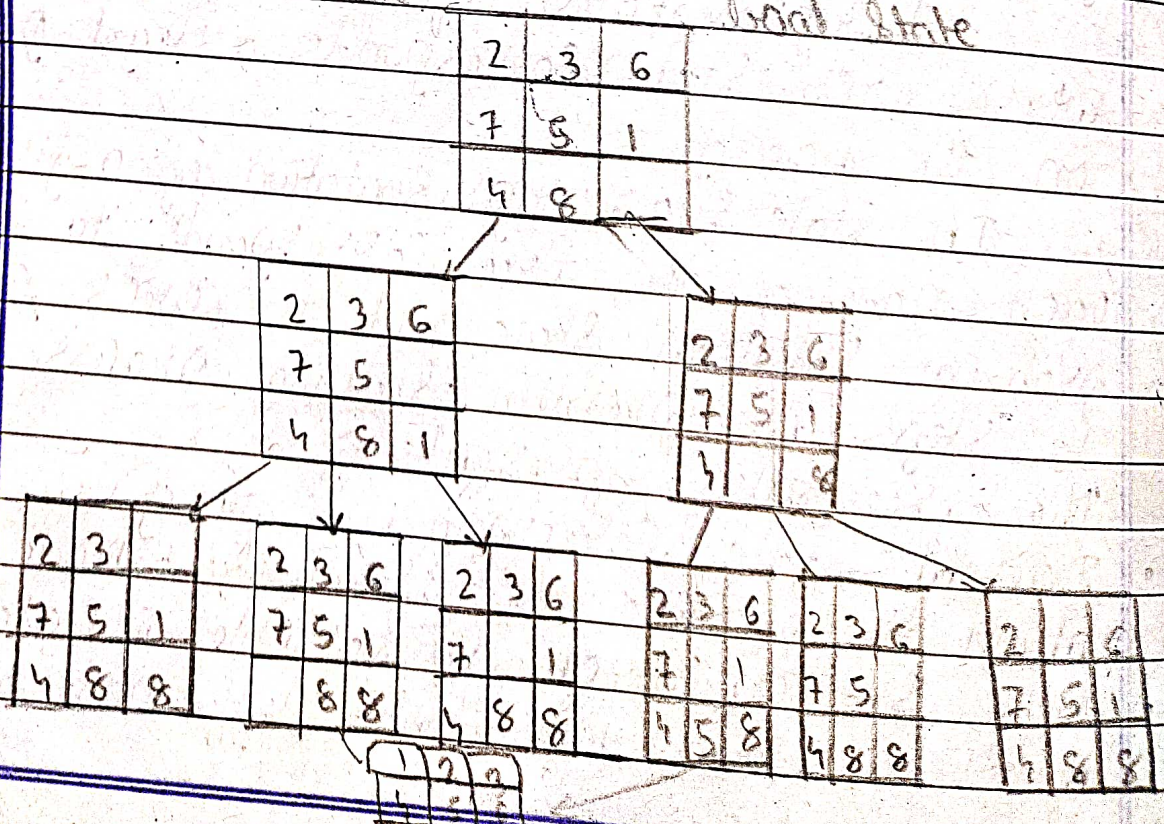
State Space Search

- A State Space is represented by four tuples [N, A, S, G, O] where N is the set of nodes or states of a graph which correspond to all the state in problem solving process.
- A is the set of arcs between the nodes which correspond to steps taken in problem solving process
- S is a non empty subset of N which contains state space of the problem.
- G is a non empty subset of N which contains acceptable goal states.



Initial State

Goal State



2) Production Rules / Systems:

- A production system is a model of computation used in AI for implementing such algorithms and for modelling human problem solving process it provides control over problem solving process and consist of the following:

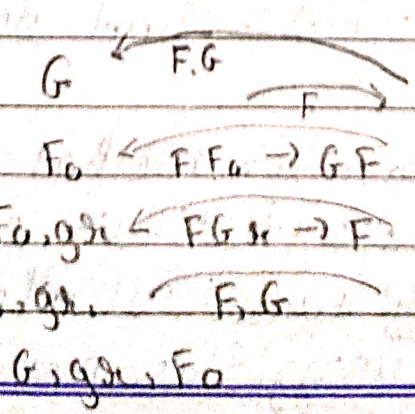
i) a set of rules called production rules consisting of condition action pair.

ii) one or more knowledge data base which contains description of current state of problem. This description is matched against the condition part of production rules to determine the least action.

- A control strategy which is a mechanism which help to decide which rule should be applied next.
- A Rule Applier.

3) Farmer, Frog, Grass, Goat problem

Farmer will take first goat on other side and return back alone. Now he will take wolf along and drop wolf on other side and return with goat. Now he takes grass along returns alone. Now finally he crosses the river with the goat and they all crossed the river.

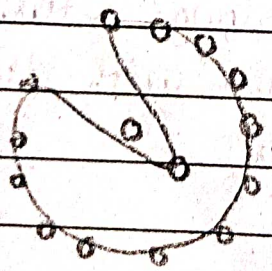


Three Missionaries And Three Cannibals

First One Missionary and one cannibal went on other side and M come alone. Now take 2C on other side of their river and one will return back. Now 2C are on other side and we get 1M and 1C on other side. And 1M and 1C will return back. Now we send 2M on other side and 1C back alone. The 3M are on other side. Now we send 2C and one will be on other side and one return back and then the other 2 reaches other side. This way all have crossed the river.

	1M 1C →	3 3
1C	← 1C	M C
	1M →	
2C	← 2C	3M 2C
	1C →	
1C 1M	← 2M	1M 1C
	1C →	
1C 3M	← 2M	2C
	1C →	
2C 3M	← 2C	1C
	1C →	
3C 3M	← 2C	

Travelling Salesman Problem

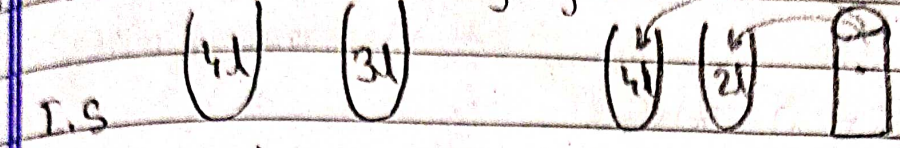


Cover all the cities once
Minimum distance
Source and destination same

→ This is key to this method is to always visit the nearest destination and then go back to the first city when all other cities are visited. To solve TSP using this method, choose a random

City and then look for closest unvisited city and go there.

Q. Two water Jug Problem:



I.S = empty (0,0) G.S = (2,0)

Steps: [(0,0) (4,0) (1,3) (4,0) (0,1) (4,1) (2,3) (2,0)]

Q. 8 puzzle Problem As production System:

(i) Production Rules

	Condition	Action
a	Goal State in Knowledge database	Halt
b	Blank is not on the right	Move Blank to the right
c	Blank is not on the left	Move Blank to the left
d	Blank is not on the top	Move Blank to top
e	Blank is not on the bottom	Move Blank to bottom

I.S

7	5	3
2		1
4	8	6

G.S

1	2	3
4	5	6
7	8	

(ii) Knowledge Database:

It is the description of current board position.

(iii) Control Strategy:

(a) Stop when goal is found.

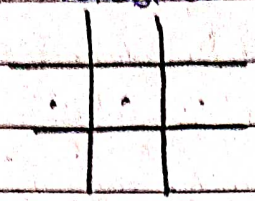
(b) Do not follow production rules in order (sequentially)

(c) Do not allow loops.

Date
Page

Production System for Tic Tac Toe:

Initial State



Goal State
- As obtained from State Space representation
- All leaf nodes

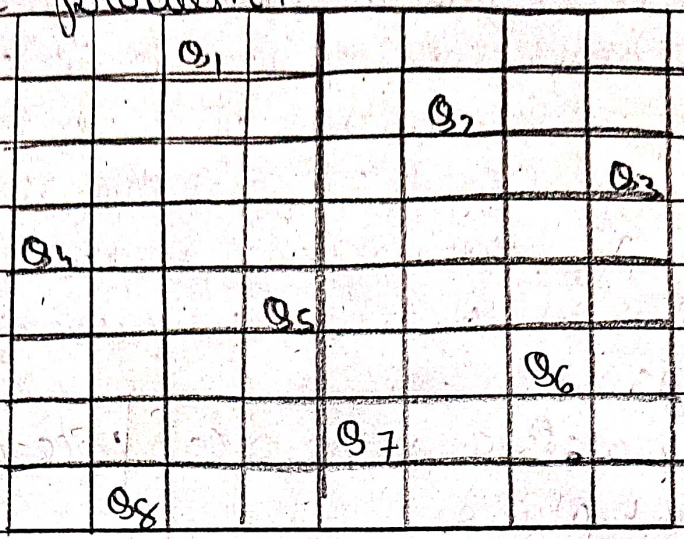
(i) Production Rules

	Condition	Action
a	Goal State in knowledge database	Halt
b	If top right block is blank	put 'x' in top right blo
c	If top left block is blank	put 'x' in top left block
d	If top middle block is blank	put 'x' in top middle blo
e	If 4th blok is blank	put 'x' in 4th block
f	The 5th block is blank	put 'x' in 5th block
g	If 6th block is blank	put 'x' in 6th block
h	If bottom right is blank	put 'x' in bottom R blo
i	If bottom middle is blank	put 'x' in middle is blank

(ii) Knowledge Database
- It is the description of current board position

(iii) Control Strategy
a) Stop when goal is found
b) Do not follow production rule in order
c) Do not allow loops
d) Block the way when opponent is winning
e) Try to form a vertical, horizontal and diagonal triplet to win the game.

8 Queen Problem:



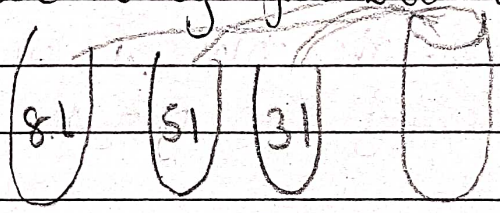
Crypt Arithmetic Problem

$S_4 E_5 N_6 D_7$ 0 1 2 3 4 5 6 7 8 9

$M_1 O_0 R_8 E_5$ O M Y E N D R S

$M_1 O_0 N_6 E_5 Y_2$

3 water Jug Problem: 8L, 5L, 3L



- First water is poured from 8l Jug to 5l, leaving 3ls in original 8l Jug.
- Next water is poured from 5l Jug into 3l So now we have 3l of water in 8l, 2l of water in 5l and 3l of water in 3l
- The 3l Jug is emptied into 8l Jug so 8l Jug now contains 6l of water.
- The 2l of water in 5l Jug are now poured into empty 3l Jug
- Water is poured from 8l Jug in 5l Jug. We now have 5l of water in 5l, 2l in 3l and 1l in 8l Jug

- Water is poured from 5l Jug to fill 3l which already contains at this stage 2l of water
- We are left with 4l of water in 5l Jug which is given to 1 friend and 3l in 3l Jug that is pour back to 5l Jug that already contains 1l of water. This gives 4l of water which are given to other friend.

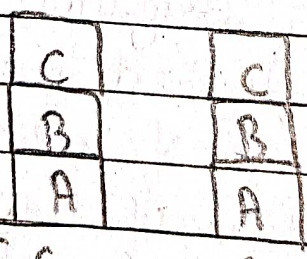
Steps

I.S (8,0,0), (3,5,0) (3,2,3) (6,2,0) (6,0,2) (1,5,2) (1,4,3) (4,4,0) G.S

Monkey And Banana Problem

If monkey and block both are on the floor and block is at the center then monkey can climb on the block so the vertical position of monkey will be changed. When the monkey is on block, the block is on the center, then the monkey can get the bananas

Blocks word problem



I.S G.S

Conditions

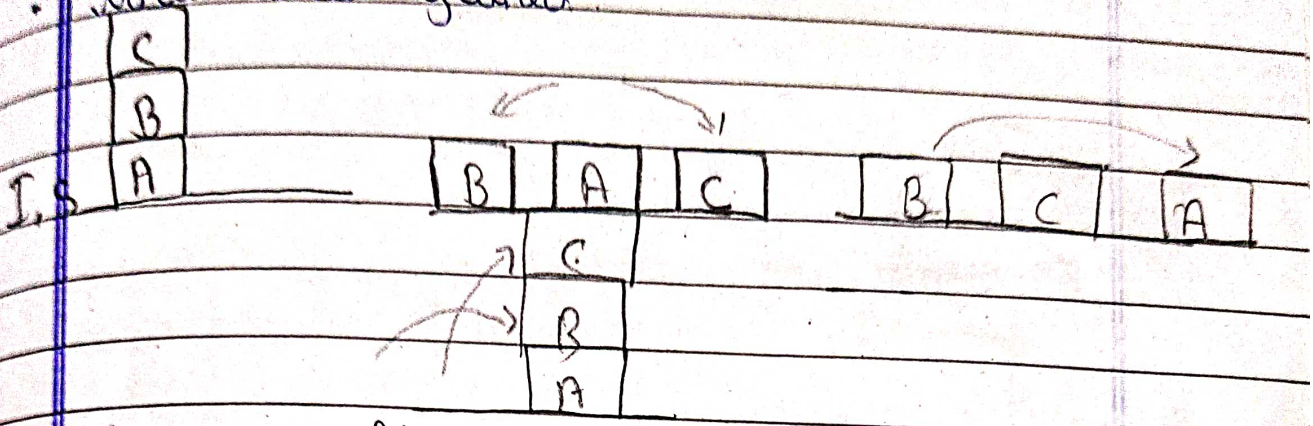
(1) Cant be reverse

(2) At a time only two blocks

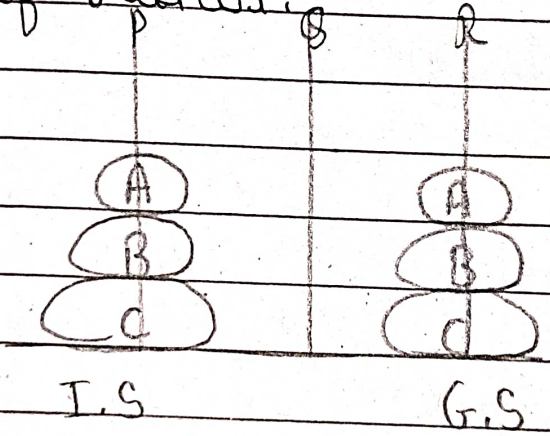
Explanation

Here the blocks can't be

- We will keep C block and B block aside on the table.
- Now will keep A block on other side
- And then will arrange A block on it and put C blocks on it.
- Goal State gained



Tower of Hanoi:

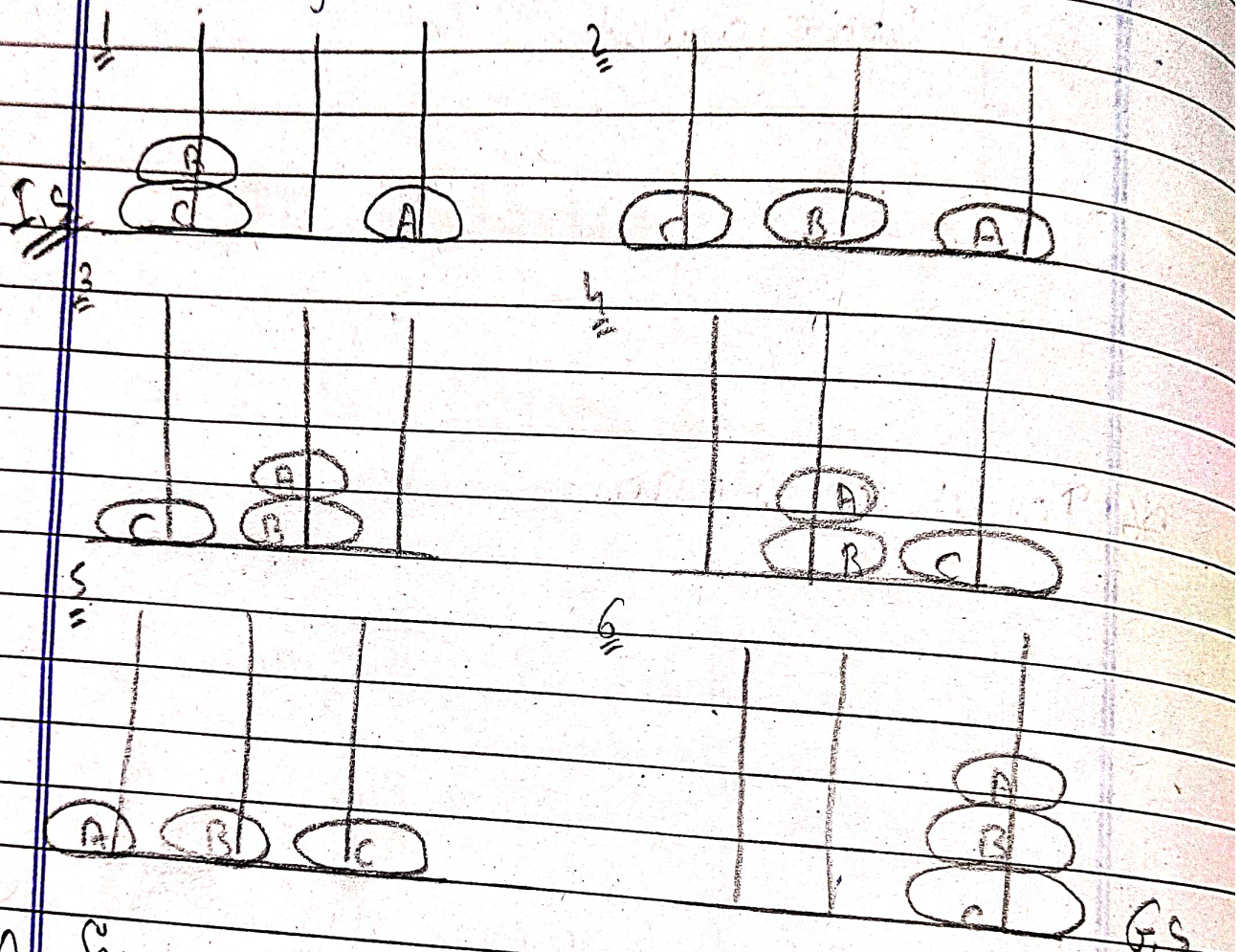


- We have 3 rods here and then on the last rod we want the same result as we have first one
- Let the rod be named P, Q and R

Step 1: First we will have take A, B keep it on R
Step 2: Then will take out B and keep it in Q rod
Step 3: Now from rod R we will take A, B will keep it on B on rod Q.
Step 4: Now from rod P take C and keep that on R
Step 5: Now taking A from the rod Q we will keep it on rod P

Step 6: And then from rod B we will keep B on C on rod

Step 7: Now from rod P we will take A and will keep on C → B on rod R → Goal state



* Given problem characteristics:

(1) Is the problem decomposible?

E.g $\int \cos x + x^2 + 3x^3 dx$

$\int \cos x dx + \int x^2 dx + \int 3x^3 dx$

$0 + 0 + 0 = 0$

A program is said to be a decomposible if it is devided into two parts and then combine it in the one answer.

(2) Can the Solution steps be ignored or undone?

Solution → Irrecoverable

Ignorable Recoverable.

(3) Is the universe predictable?

Yes No

Plain Plan and revise

(4) Is the good solution a state or a path?

state path

(5) Is the solution absolute or relative?

absolute Relative

(6) What is amount of knowledge required?

Less Intermediate more

(7) Does the problem require interaction with a person

Solitary (single player) Interactive

Search Techniques:

1 Depth first Search (DFS).

(i) Place the starting node s in the queue.

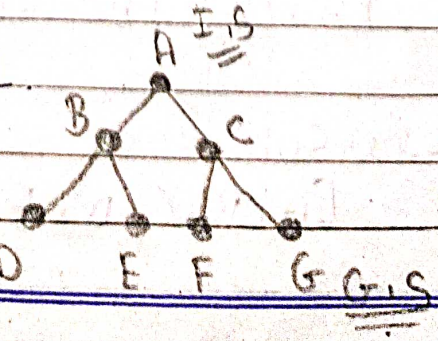
(ii) If the queue is empty report failure and stop.

(iii) If the first element in the queue is goal. Report success and stop.

(iv) Remove first element of the queue expand it and place all the children in front of the queue.

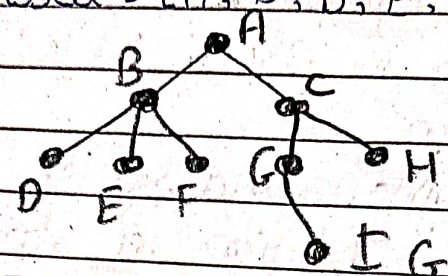
(v) Go to step 2.

E.g



Trace = [A], [B, C], [D, E, C], [E, C], [C], [F, G]
[G] Goal state
Traversal = [A, B, D, E, C, F, G]

(2)

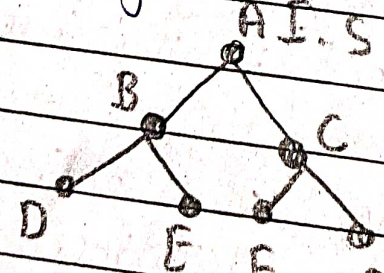


[A] [B, C] [D, E, F, C], [E, F, C] [F, C] [C]
[G, H] [I] Success
Traversal = [A, B, D, E, F, C, G, I]

2 Breadth First Search (BFS)

(i) Place the starting node s in the queue
(ii) If the queue is empty report failure and stop
(iii) If the first element in the queue is goal, report success and stop
(iv) Remove first element of the queue, expand it and place all the children in rear of the queue
(v) Go to Step 2

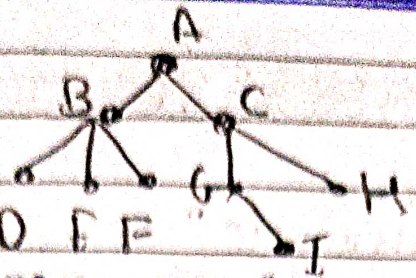
E.g



Trace: [A] [B, C] [C, D, E] [D, E, F, G] [E, F, G] [F, G]
[G] Sucess

Traversal:
[A, B, C, D, E, F, G]

(2.)



Trace: [A] [B, C] [C, D, E, F] [D, E, F, G] [E, F, G, H] [F, G, H] [G, H] [H, I] [I] Sucess

Transversal: [A, B, C, D, E, F, G, H, I]

* Advantages of Depth First Search (DFS)/Disadvantages of Breadth First Search (BFS)

(i) It requires less amount of memory

(ii) It can get to a solution quickly without examining much of search space.

* Advantages of Breadth First Search (BFS) / Disadvantages of Depth First Search (DFS)

(i) It will always find a solution if it exists larger paths are never explored if shorter on exists

(ii) It never gets trapped.

Ch-4 Heuristic Search Techniques

What is Heuristic?

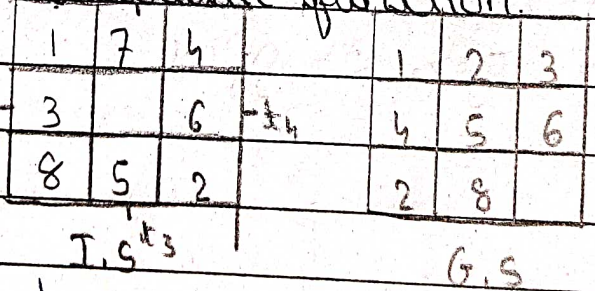
- Heuristic is a technique which improves the efficiency of search process possibly by sacrificing the claims of completeness.
- Heuristics are applied in two situations (i) when a problem may not have an exact solution because of inherent ambiguities in the problem statement (ii) when a problem may have an exact solution but computational cost of finding the solution is prohibitive. e.g. Chess

Heuristic Function:

$f(\text{Current State}) = n$ (goodness)

f is a maps problem state description to a Numeric Value

8 puzzle Heuristic function



(i) Find the distance of each tile from its goal position.

(ii) Sum all the distances and at reject the one big greatest distance

(iii) Choose one possibility which minimizes the distance.

(iv) In order to select which of the above tile

should be used for making next move use the following scheme or strategy.

a) Consider each tile from the current state that can be moved say $t_1, t_2 \ldots t_n$. Make the move and get the resulting states $S_1, S_2 \ldots S_n$.

b) Evaluate each state $S_1, S_2 \ldots S_n$ as follows.

(i) In each state S_i, as if consider each tile one by one

(ii) Find the distance of each tile from current position to destination position in Goal State and all the distances for each tiles.

(iii) Let the resulting distances are $D_1, D_2 \ldots D_n$.

(iv) Find the smallest distance D_i. This indicates the state S_i is selected for next move.

$7 = t_1$
$3 = t_2$
$5 = t_3$
$6 = t_4$

1		4
3	7	6
8	5	2

S_1

1	7	4
	3	6
8	5	2

S_2

1	7	4
3	5	6
8		2

S_3

1	7	4
3	6	
8	5	2

S_4

$f : S_1 \to N$

1	2	3	4	5	6	7	8

D_1 0 + 3 + 3 + 3 + 3 + 0 + 1 + 0 = 13

$f : S_2 \to N$

1	2	3	4	5	6	7	8

D_2 0 + 3 + 2 + 3 + 1 + 0 + 3 + 1 = 13

$f : S_3 \to N$

1	2	3	4	5	6	7	8

D_3 0 + 3 + 3 + 3 + 0 + 0 + 3 + 1 = 13

$f: S_4 \to N$

1	2	3	4	5	6	7	8
D_1 0	3	3	3	1	1	3	1 = 15

Note: If we choose minimum D_1 then S_1 will be used

(1) Generate And Test:

Generator Tester C.S = Current State N.S = Next State

Generator will generate next step for us → If N.S = G.S → yes or no

C.S → If N.S = G.S ?

yes no

Characteristic of Generate And Test:

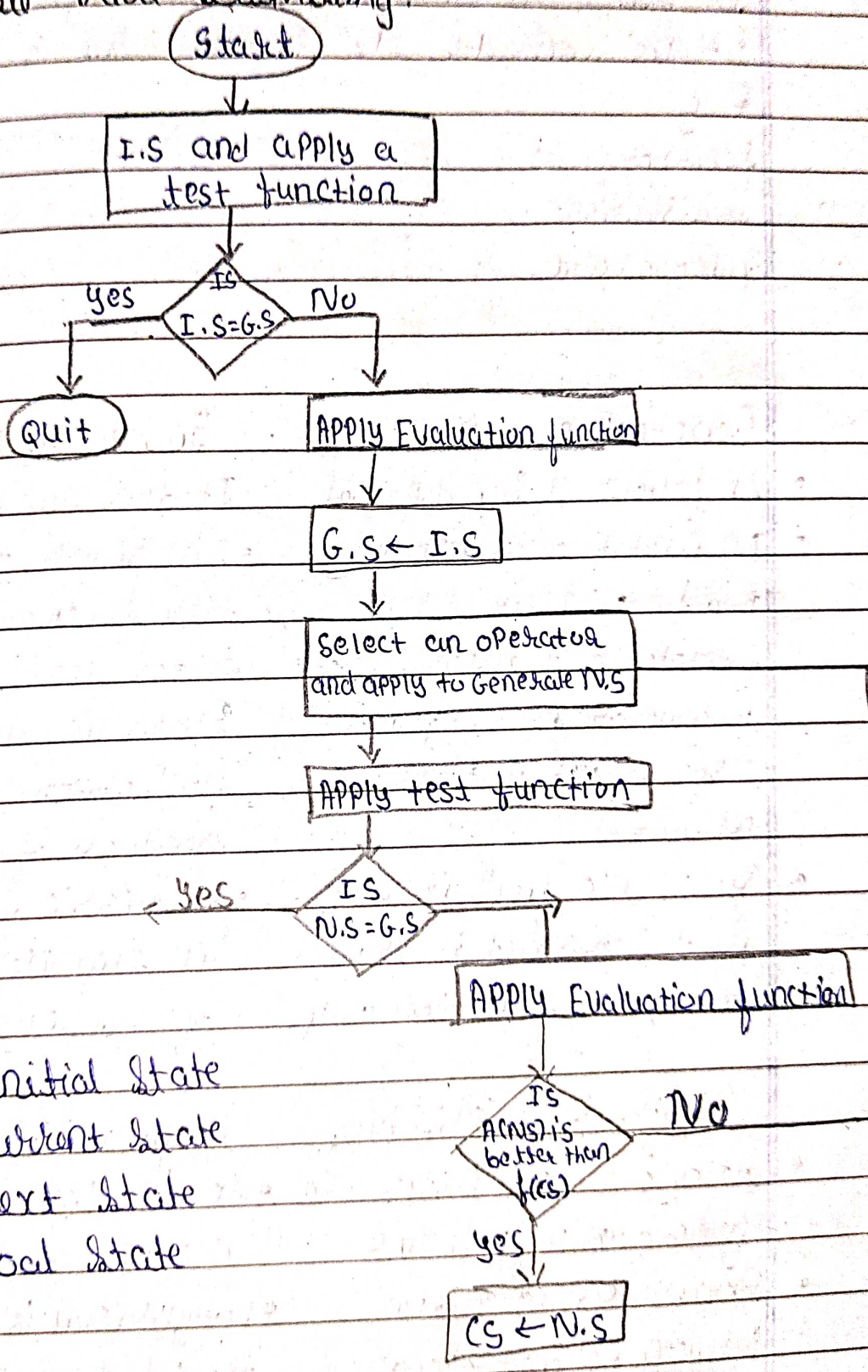
- Generate and test displays DFS to generate a possible before being tested
- If generate and test is performed systematically it leads to exhaustive search in the search space.
- As compare to randomly generating a nodes generate and test takes lesser time to reach a goal state.
- Evaluation of reaching to G.S by tester function is performed using heuristic function.
- It does not guarantee that a G.S will be found.

Or Systematic Generate and test

- DFS with backtracking

or Plan Generate and test

2 Simple Hill Climbing:



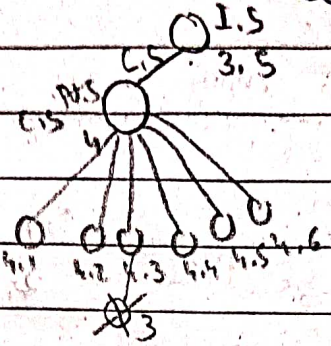
I.S = Initial State
C.S = Current State
N.S = Next State
G.S = Goal State

- Note: If I.S ≠ G.S then I.S = G.S then go to N.S and If f(NS) ≠ G.S then Convert N.S to C.S. If the value of N.S is better than C.S then convert it to C.S and then create their Children paths. And if N.S

is not better than C.S then generate other state
Let us consider the bigger value is better for now

E.g

f(N.S) or f(C.S) is a Heuristic function value



f(N.S) f(C.S)
4 3.5

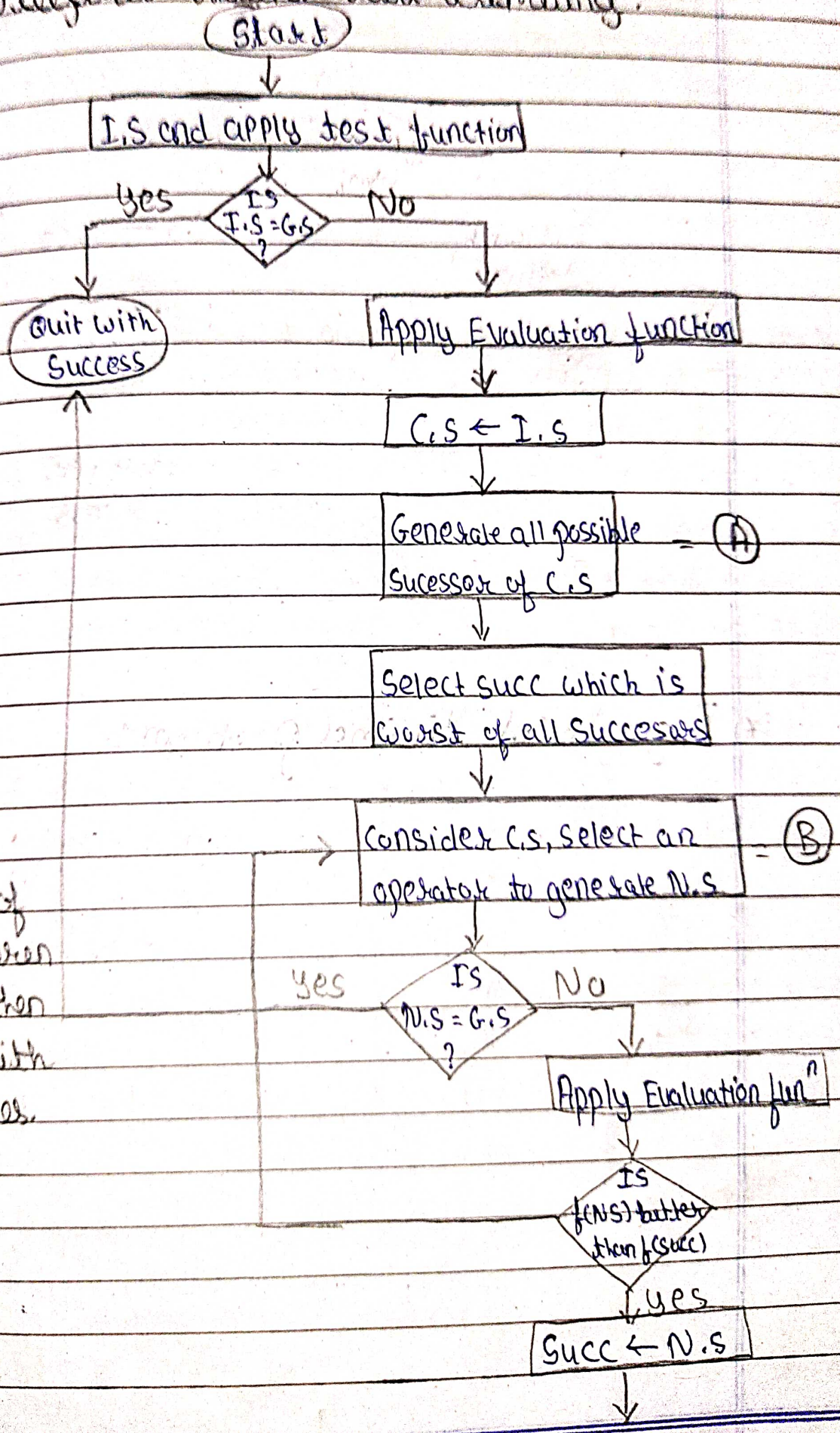
Generate and Test	Simple Hill Climbing
• It follows a blind method • In Generate and test only two function are used generator and testor. Generator to generate next step and testor to test if N.S is G.S or not • Heuristic fun^n is not used seperated it is used in testor function only	• It does not follow any path blindly • In Simple hill Climbing three function are used. Generator testor and Evaluation function. Evaluation fun^n is a Heuristic fun^n which evaluates the goodness of state • Heuristic fun^n in Simple hill Climbing gives an estimate of how close we are to G.S

31/12/21

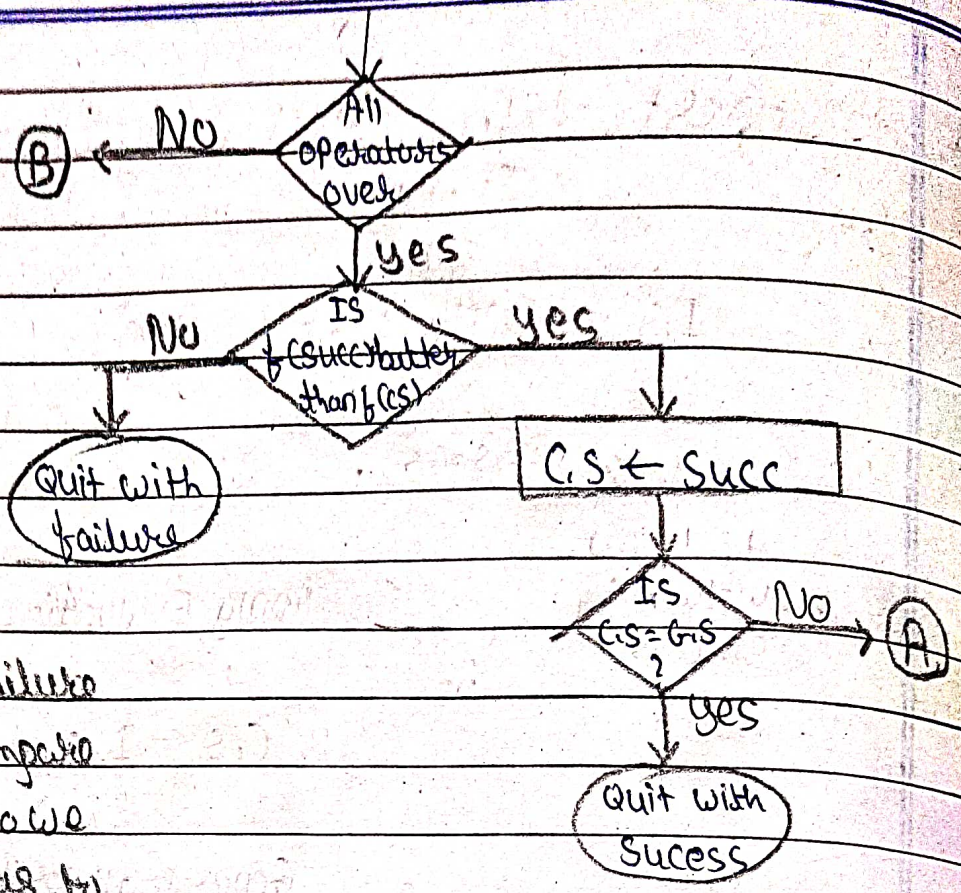
Simple Hill Climbing	Steepest Ascent Hill Climbing
• At one time 2 nodes are generated: C.S & N.S • Comparison is better between parent state and child state • Compare S.A.H.C success will be blow. • Here we couldn't properly	• At one time, 3 nodes are generated C.S, N.S and success state • Comparison is 1st bet^n child state themselves & then between parent state and best child state • There are more chances to find success easily • Here failure is possible bcz we are checking every nodes

ine a situation where would find failure

Steepest Ascent Hill climbing:



Succ is finally selected out of all the children generated then compared with other nodes.



Here we have possibility of failure so here we compare all the nodes so we get an idea as to where we will (in which situation) find failure.

E.g - Local Minima problem:

-1	A
+1	H
+1	G
+1	F
+1	E
+1	D
+1	C
-1	B

f(I.S) = 1

I.S

+1	H
+1	G
+1	F
+1	E
+1	D
+1	C
+1	B
+1	A

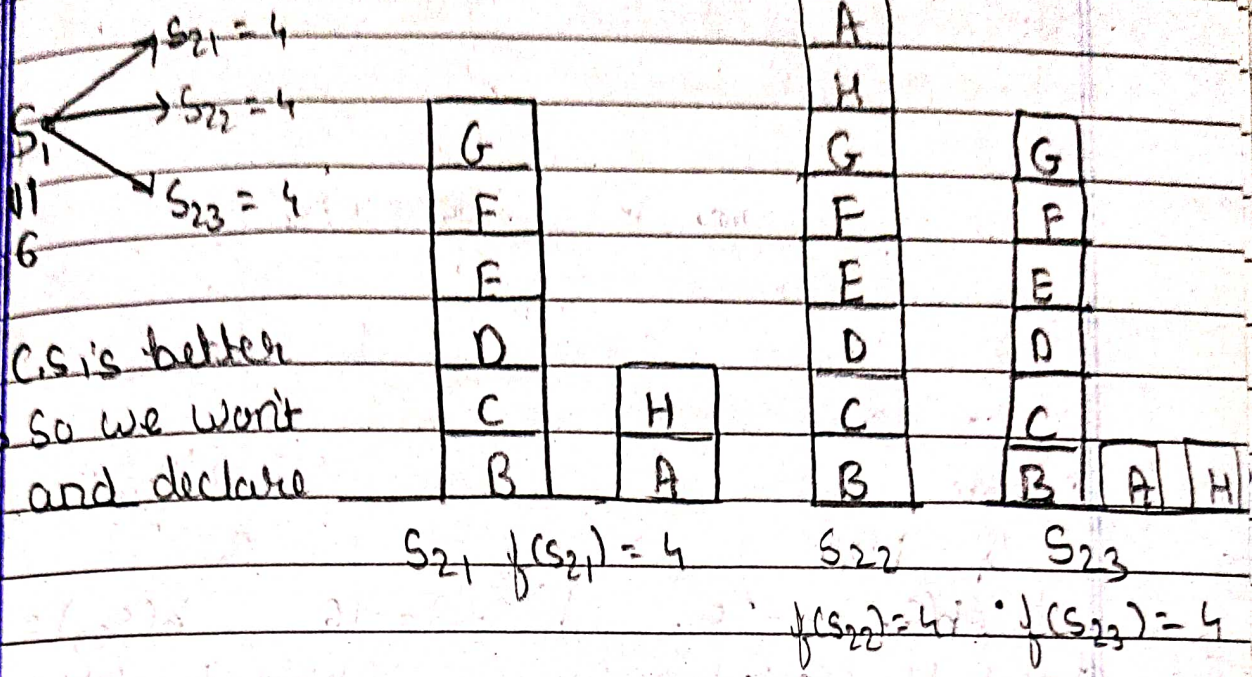
G.S

f(G.S) = 8

$S_1 \rightarrow$ f(S1) = 6

+1	H
+1	G
+1	F
+1	E
+1	D
+1	C
-1	B

A	+1



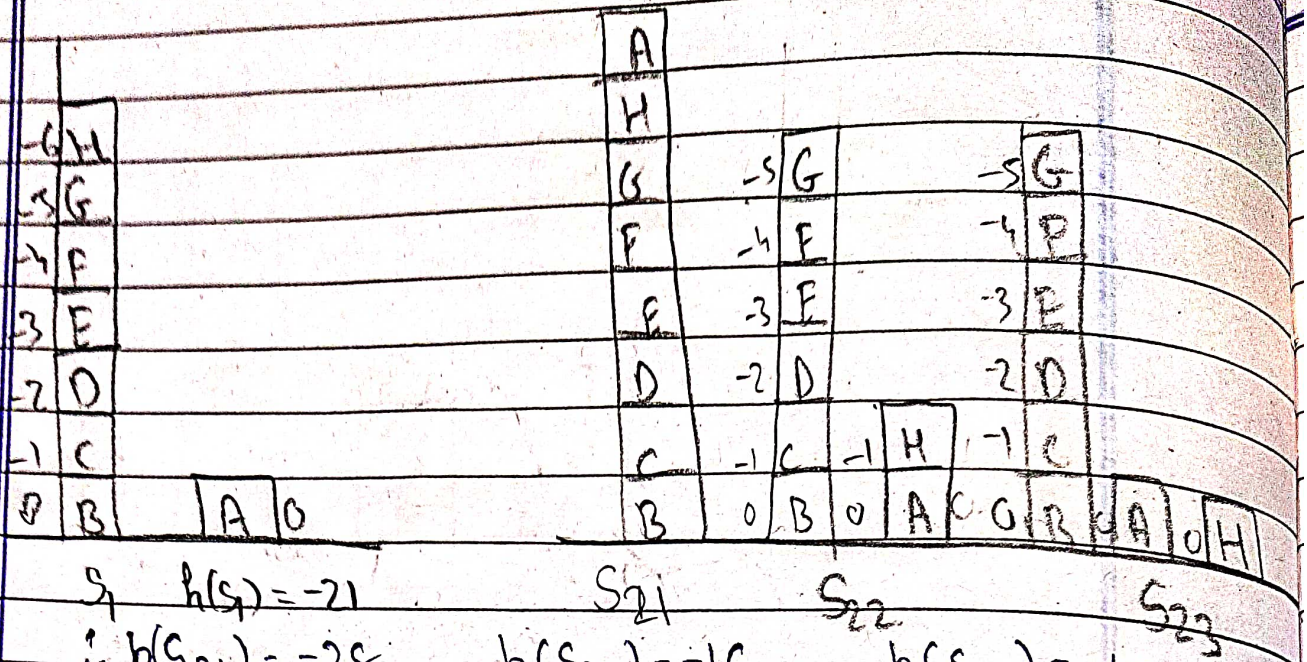
So here C.S is better than N.S so we won't find ans and declare failure

Global Heuristic function:

For each block that has the correct support structure (i.e the complete structure underneath it is exactly as it should be) add one point of every block in the support structure. For each block that has an incorrect support structure, subtract one point for every block in the existing support structure.

+7	A	H	+7	$h(G,S) = 28 = B+C+D+E+F+G+H+A$
+6	H	G	+6	$h(I,S) = -28 = A+B+C+D+E+F+G+H$
+5	G	F	+5	
+4	F	E	+4	
+3	E	D	+3	
+2	D	C	+2	
-1	C	B	+1	
0	B	A	0	

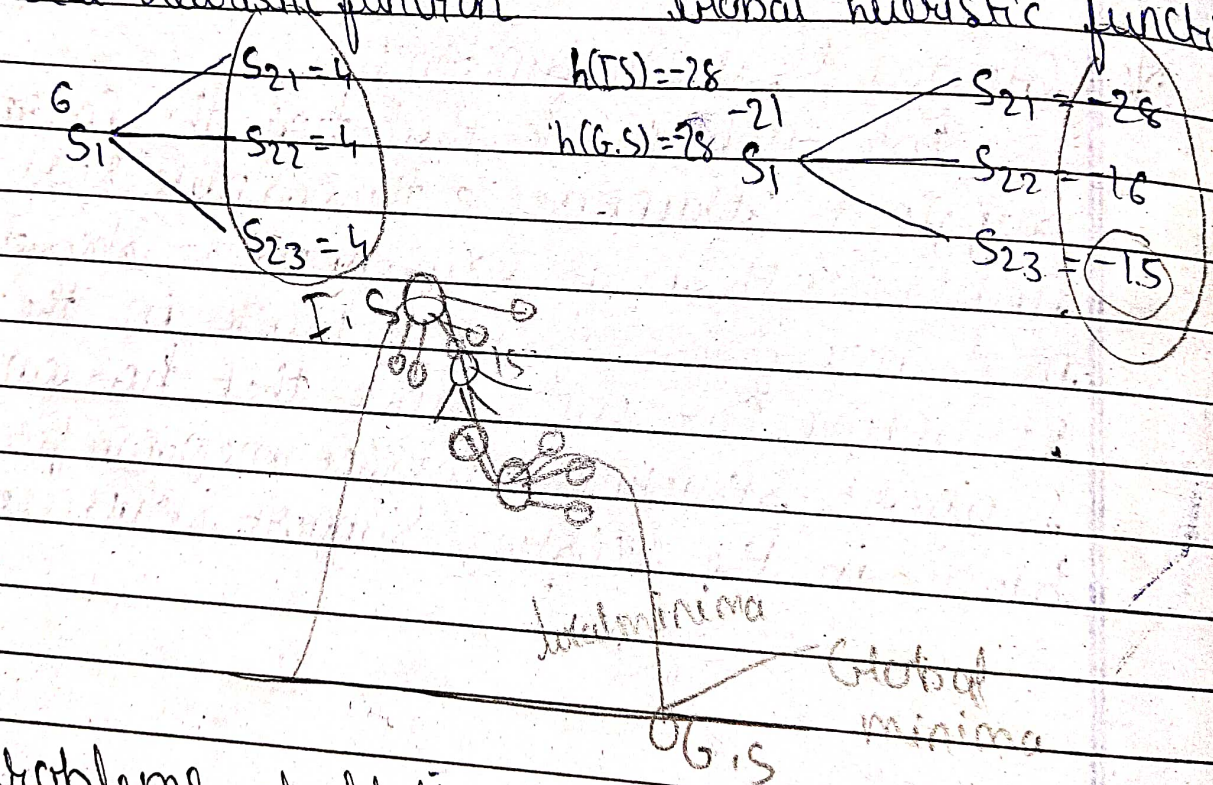
IS GS



S_1 $h(S_1) = -21$ S_{21} S_{22} S_{23}

$\therefore h(S_{21}) = -28$ $h(S_{22}) = -16$ $h(S_{23}) = -15$

Local Heuristic function Global heuristic function



Problems of Hill climbing:

1. Local Minima

It is a state which is better than all its neighbouring states but not better than some other state which is far away.

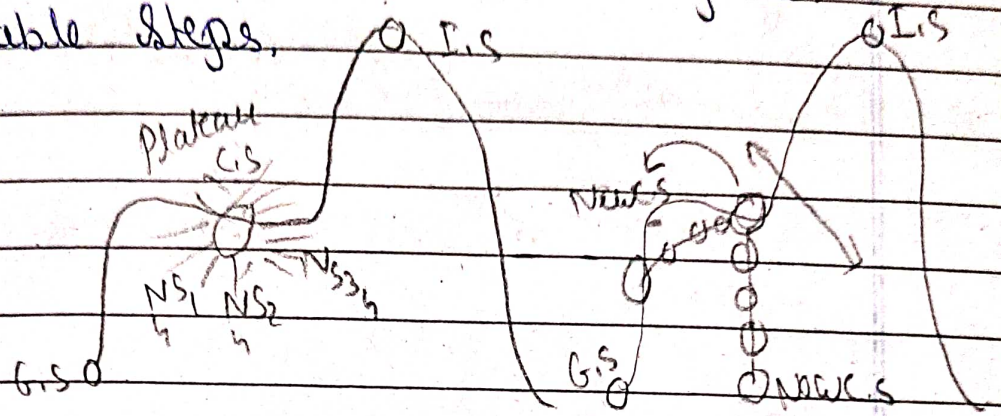
Solution: Backtrack to some earlier state and try getting to some other direction.

2. Plateau:

A flat area of search space in which all neighbouring states have the same value

- Solution: Take a big jump to try to get to a new section in the search space

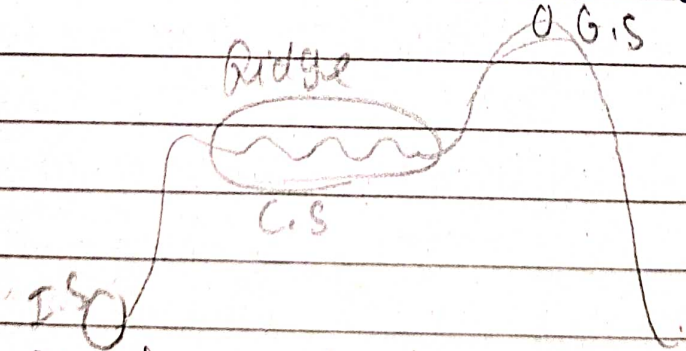
Ignorable steps.



3. Ridge:

It is an area where the orientation of the high region, compared to the set of a variable moves makes it impossible to climb up.

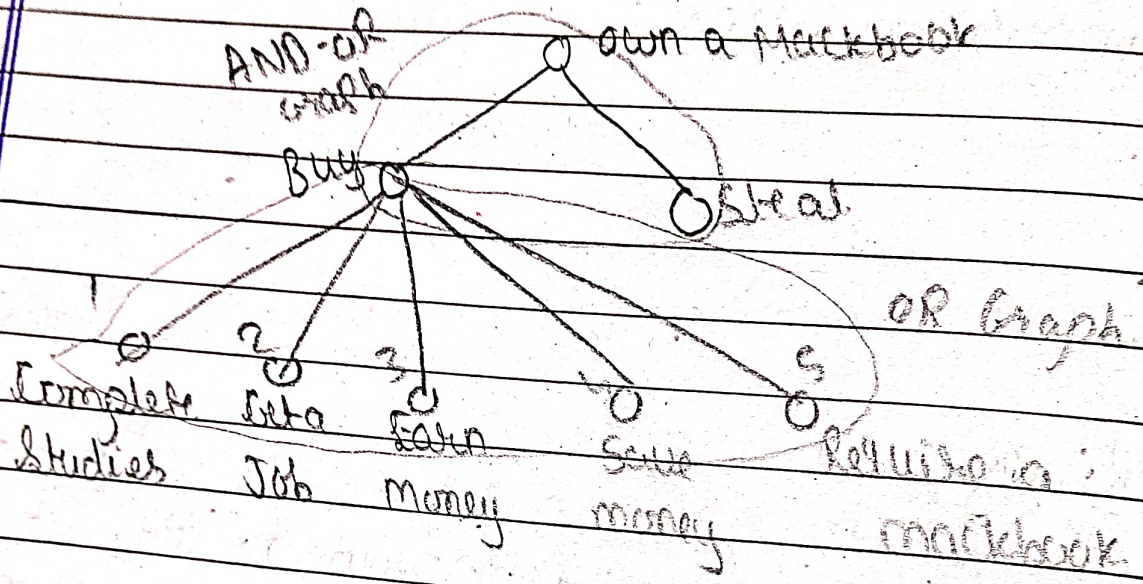
- Solution: Move in several directions at once this to see which is better next state



OR Graph and AND-OR Graph

OR Graph	And OR Graph
• Several arcs may emerge from a single node indicating variety of ways in which it can be solved.	• Several arcs may emerge from a single node but some of the alternatives contain sub branches which should be solved together.

• To Solve problems in OR graphs BFS is used • Nodes and arcs can be considered independently to find the solution	• To Solve problems in And-O graphs DFS BFS is used • Nodes and arcs are to be chosen in combination with other nodes



Ch-5 Expert Systems

Intelligent Behaviour

- learn from experience.
- Apply knowledge acquired from experience
- Handle complex situations.
- Solve problems when some information is missing
- React to a new situation
- understand visual images.
- process and manipulate symbols.
- Be creative and imaginative.

Human beings have it all.

Overview of Artificial Intelligence

- Artificial Intelligent Computers.
- Computers with the ability to mimic or duplicate
  - The intelligence of human beings.
  - The functions of the human beings.

- Artificial Intelligent Systems
- The systems and machines that demonstrate the characteristics of intelligence

- Artificial Intelligence (AI)
- Area of Computer Science that deals with the Artificial Intelligent Systems.

Date
Page

Application Area of Artificial Intelligence.

Artificial Intelligence

- Robotics
- Natural Language Processing
- Neural Networks
- Expert Systems

What is an Expert System?

- Expert
  The people who are very familiar with solving specific types of problems
- Knowledge Based Systems
- The fundamental function of the expert system depends upon its knowledge.
- The expert system is sometimes called knowledge based system.

In short an ES is an intelligent computer program that can perform special and difficult task(s) in some field(s) of at the level of human experts.

Overview of Expert Systems

- Expert system can....
  - Display intelligent behaivour
  - Explain their reasoning or suggested decisions
  - Draw conclusions from complex relationships.
- Expert System Shell
  - A collection of software packages and

Tools to used to develop expert systems

* Components of an Expert System
  - Knowledge Base
    - Stores all relevant information, data, rules, cases and relationships used by the expert system.

  - Inference Engine
    - Seeks information and relationships from the knowledge base and provides answers, predictions, and suggestions in the way a human expert would

  - Rule
    - A conditional statement that links given conditions to actions or outcomes.

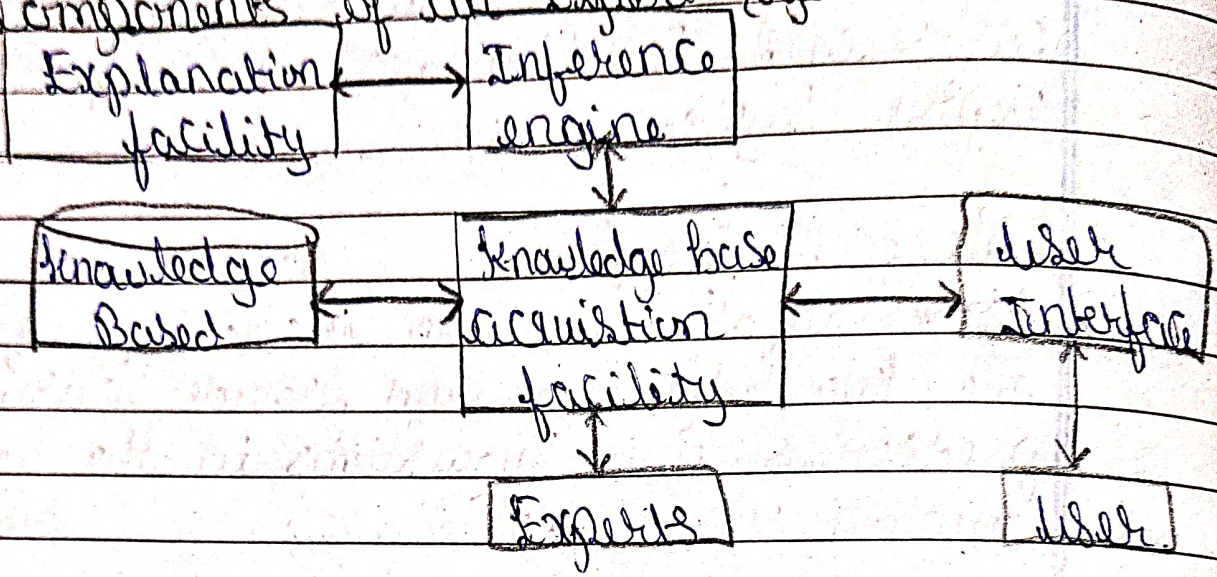
Eg If patient has fever and has headache and has sore throat then
- It is flue.

→ If patient has fever and has headache and has nausea, then
- It has viral fever.

  - Backward Chaining
    - A method of reasoning that starts with a conclusions and works backward to the supporting facts.

- Forward Chaining:
  - A method of reasoning that starts with the facts and works forward to the conclusions.

Components of an Expert System



Rules for a Credit Application

→ Application for a loan for Rs 1,00,000 to Rs 2,00
- If there are no previous credit problems and
- If month net income is greater than 4 x monthly loan payment and
- If down payment is 15% of total value of Proper
- If net income of borrower is > Rs 25,000 and
- If employement is > 3yrs at same company.

Then accept the application
Else check other credit rules

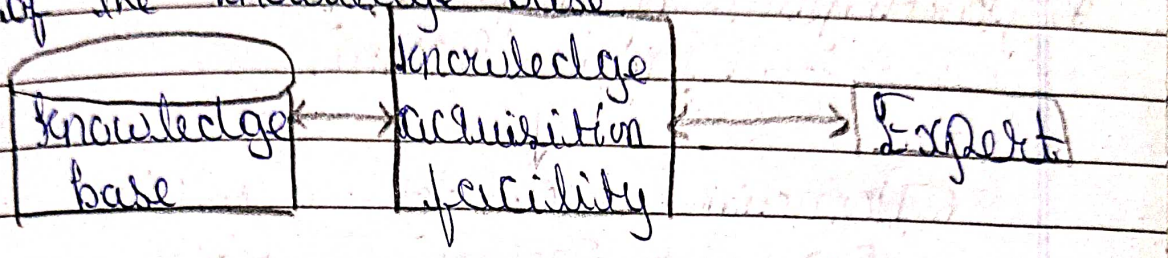
Explanation Facility

- A part of the expert system that allows a user or decision maker to understand

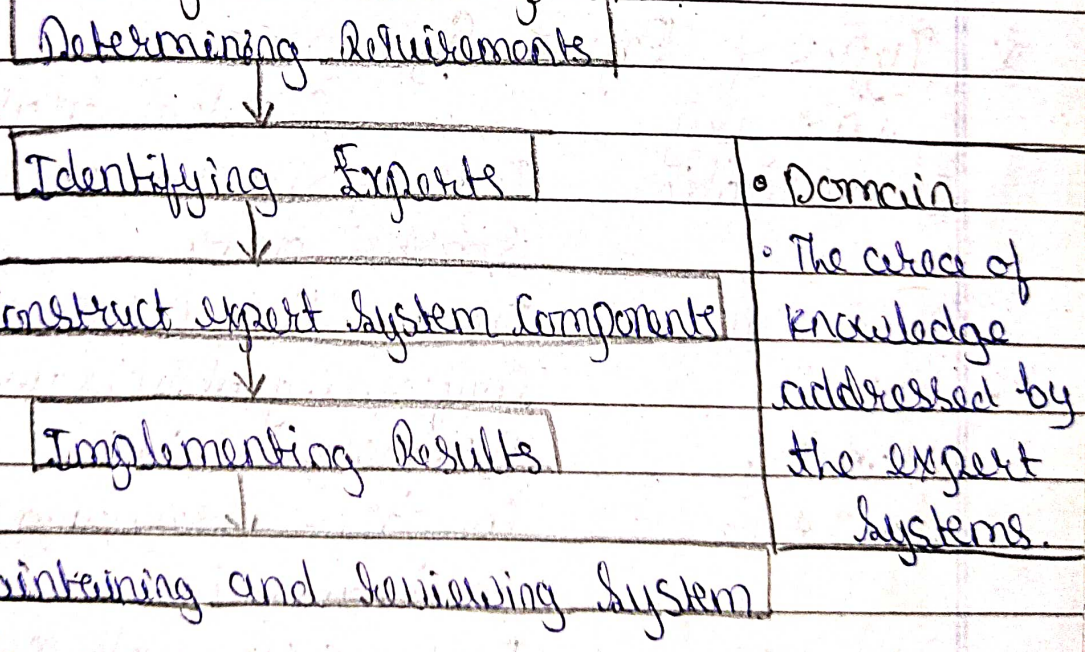
how the expert system arrived at a certain conclusions or results.

Knowledge Acquisition Facility

- provides a convenient and efficient means of capturing and storing all components of the knowledge base



Expert Systems Development



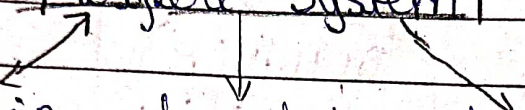
Participants of Expert Systems

- Domain Expert
  - The individual or group whose expertise and knowledge is captured for used in an expert system.
- Knowledge user.
  - The individual or group whose users and

Get benefits from the expert system.

- Knowledge Engineer.
- The individual or group who are trained or have experience in design, development, implement and maintanance of an expert system.

Participants of Expert Systems



Types of Expert System

Category	Problem Addressed
• Interpretation	Inferring situation description from sensor data
• Prediction	Inferring likely consequences of given situa
• Design	Configuring objects under constrains
• Planning	Designing actions
• Monitoring	Comparing observation to plan a vulnerabilities
• Diagnosis	Inferring system malfunction from observat
• Debugging	Prescribing remedies for malfunctio
• Instruction	Diagnosing, debugging and a repairing student behaviour
• Control	Interpreting, Predicting, Repairing repairing and monitoring system behaviour

Limitations of Expert Systems

- Not widely used or tested.
- Limited to relatively narrow problems
- Cannot readily deal with "mixed" knowledge.
- Possibility of error.
- Cannot define own knowledge base.
- Difficult to maintain.
- May have high development costs

Expert System - MYCIN

- An early expert system developed in early 1970s at Standford university.
- It provides advice through a consultive dialogue

MYCIN: The Problem

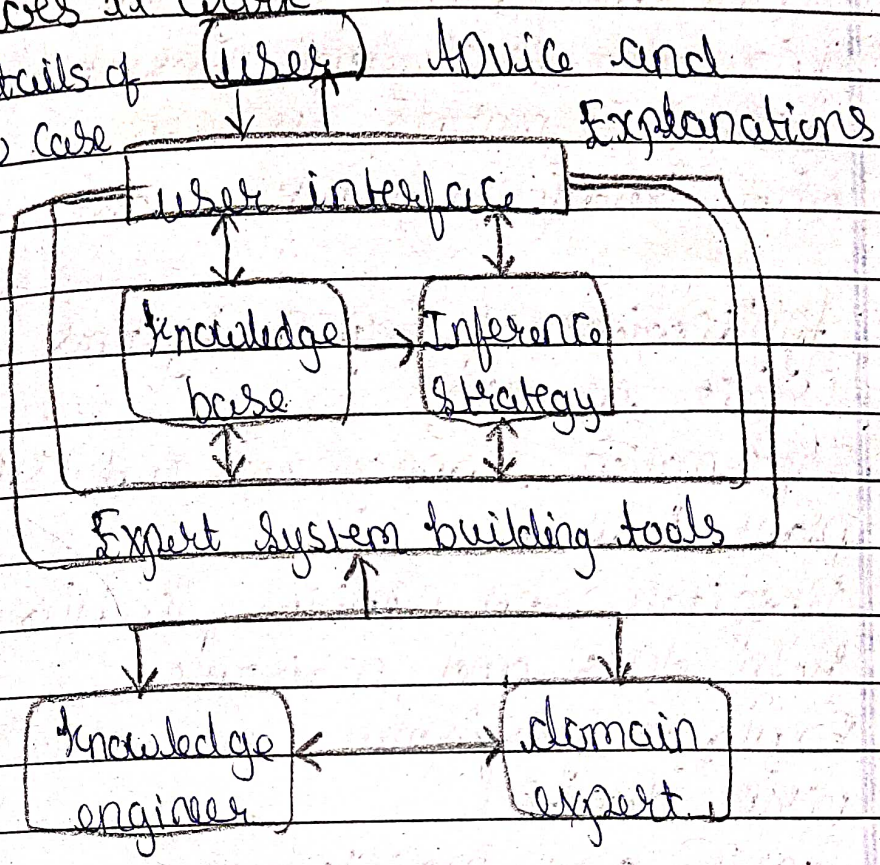
- Roberts and Misconti (1972):
- only 13% of patients are treated rationaly.
- 66% are being given irrational treatement
- 21% are being given questionable treatement.
- Reason
- Lack of time for proper diagonsis
- Lack of complete knowledge

Design Parameters:

- Expected design parameters: MYCIN
- Program must be competent and easy to use
- Must handle a large, changing body of knowledge
- Interact with human users.
- Must take time into account

- Work with incomplete or uncertain information.

* How does it work?



* Reasoning And Problem Solving Strategy
- MYCIN could use backward chaining to find out whether a possible bacteria was to blame.
- performs assessment of the likehood of a bacteria
- The procedure is carried out in following style of dialougue:
- MYCIN: Was the patient had neurosurgery?
- USER: No
- MYCIN: Is the patient a burn patient?
- USER: No
- ...
- MYCIN: It could be Diplococcus

Diagnostic Reasoning: Internist

- Internist is a medical expert system for General disease diagnosis
- Knowledge in system consists of disease profiles giving symptoms associated with disease and relationship between them

Applications of Expert Systems

- Credit granting
- Information management and retrival
- Plant layout
- Hospitals and medical facilities
- Help desks and assistance
- Employee performance evaluation
- virus detection
- Repair and maintenance
- Shipping
- Marketing
- warehouse optimization

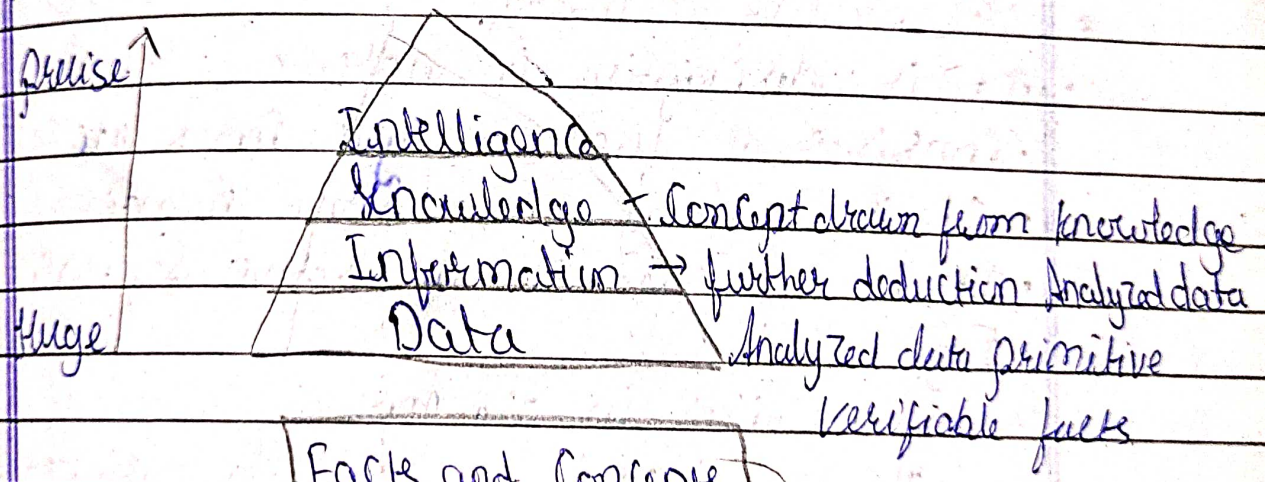
Examples of Expert System

- MYCIN: It was based on backward chaining and could identify various that could cause acute infections. It could also recommend drugs base on patient's weight
- Dendral: Expert system used for chemical analysis to predict molecular structure
- PXDES: An example of Expert system used to predict degree and type of lung cancer
- CaDet: one of best e.g that can identify cancer at early stages

Ch-6 Knowledge Representation

* Knowledge
  It is the understanding of a particular subject area.

* Knowledge Representation
  The methods or techniques used for representing the knowledge in a best way to be stored and retrieved by knowledge based systems in knowledge base.



* Knowledge Manipulation involves:
  - Knowledge acquisition
  - Gather knowledge
  - Structure knowledge

- Organizing knowledge
- Knowledge Storage
  - Put the knowledge in knowledge base
- Knowledge Retrieval
  - fetch the knowledge whenever required
- Reasoning
  - Draw inference
  - Explain the inference
  - Reason.

Types of knowledge:

- Declarative knowledge:
  - It is descriptive in nature
  - Consists of facts, concepts, cases etc.
  - It is knowing about some domain.
  - It is expressed in declarative sentences.
  - Eg.
    - Sun rises in the east
    - The first step in cooking is to wash vegetables
- Procedural knowledge:
  - It is imperative in nature
  - Consists of rules, procedural strategies etc.
  - It is defining how to do something
  - It is expressed in form of some steps to be followed.
  - E.g
    - If it rains, I will carry an umbrella.
    - The complete recipe to cook a vegetable.

Knowledge Representation Techniques

1. Propositional logic

Propositions

- Marcus was a man
  manmarcus
- Marcus was a pompeian
  Marcuspompeain , Pompeianmarcus
- My house is red
  redhouse
- She is a beautiful girl
  beautiful girl
- Some girls are beautiful
  beautiful girls
- All the girls in MSC-I are beautiful
  beautifulgirls

Predicate Logic:

Symbols used

- ∀ : universal quantifier
  - For all
- ∃ : Existential quantifier
  - For some
- ¬ : Negation
  - Not
- ∧ : Conjuction
  - And relationship
- ∨ : Disjunction
  - Or relationship
- → : Implication

- If then Rules.

Examples

- Apples are food
  food(apple)
- Apples is red in color
  Red(apple) , Color(apple, Red)
- Marcus was a pompeian
  pompeian(marcus) Religion(marcus, pompeian
- Pizza is a food.
  food(pizza)
- Pizza is a mexican food.
  food(mexican, pizza)
- Ram likes all kinds of food.
  - $\forall x$: food(x) ^ likes(ram, x)
    For all x, such that x is food, ram likes
- Lagaan is a movie.
  movie(lagaan)
- Dashrath was a king
  king(dashrath)
- Ram was brother of Laxman
  brother_of(ram, laxman) brother(ram, laxman)
- Shikhar dhawan is an all rounder cricketer
  all_rounder cricketer(shikhar_dhawan)
  cricketer(shikhar_dhawan, all rounder)
- Rohit Sharma is a batsman
  Batsman(Rohit_Sharma)
  cricketer(Rohit_Sharma, batsman)
- Ram and Laxman are brothers
  brother(Ram, laxman) or brother(laxman, Ram

- Ankit is not freind of Dilip.
  ¬ friend(ankit, dilip)
- Archit is not punctual
  ¬ punctual(archit)
- Mohan eats peanuts and is alive
  eats(Mohan, peanuts) ∧ alive(mohan)
- Venus is a heavenly body which is not a star
  heavenly_body(venus) ∧ ¬ star(venus)
- Every heavenly body is a star, a planet or a comet
  ∀x: heavenly body(x) → star(x) ∨ planet(x) ∨ comet(x)
- all comets near sun have tail
  ∀x: (comet(x) ∧ near(x, sun)
- Venus is near sun but does not have tail
  ~~have_tail(venus, near sun)~~
  near(venus, sun) ∧ ¬ has_tail(venus)
- All yellow mushrooms are poisonous
  ∀x: mushroom(x) ∧ yellow(x) → poisonous(x)
- When vacation started, all students went home
  ∀x: student(x) ∧ started(vacation) → went home(x)
- Mary likes everybody who likes to play chess
  ∀x: person(x) ∧ like to play(x, chess) → like(mary, x)
- Mohan likes to watch all kinds of movies
  ∀x: movie(x) ∧ like(mohan, x)
- I like to watch animated movies
  ∀x: movie_type(x, animated) ∧ like(I, x)
- When it will rain, Shreya will carry an umbrella
  weather(rainy) → carry(shreya, umbrella)
  rain → carry(shreya, umbrella)
- John plays football in Sardar patel stadium

Stadium (Sardar Patel) ^ Play-football (John, Sardar Patel)

- Rohit Sharma is my favourite Cricketer
  Cricketer (Rohit Sharma) ^ Play-for (Rohit Sharma, India)
  ^ like (I, Rohit Sharma)
  Indian-Cricketer (Rohit Sharma) ^ like (~~Nidhi~~ I, Rohit Sharma)
- All the Students were happy to go to College
  ∀x: student (x) ^ studies (x, college) → happy-to-go (x, college)
- All MSC-I Students like to Study AI
  ∀x: student (x, msc-I) ^ like-to-study (x, I)
- All yellow mushrooms are poisonous
  ∀x: mushroom (x) ^ yellow (x) → Poisonous (x)
- One who knows driving does not know cooking
  ∀x: person (x) ^ know (x, driving) → ¬ know (x, cooking)
- Laxman eats everything that Ram eats
  ∀x: food (x) ^ eats (Ram, x) → eats (Laxman, x)
- Everyone likes Chocolates
  ∀x: person (x) ^ likes (x, Chocolates)
- Nobody like taxes
  ∀x: person (x) ^ ¬ likes (x, taxes)
- John teaches Python to Sally
  teacher (John) ^ student (Sally) ^ subject (Python) ^
  teach (John, Sally)
- Peter and Kerry play Chess
  person (Peter) ^ person (Kerry) ^ game (Chess) ^ play-with
- Brothers are Siblings
  ∀x: ∀y: male (x) ^ male (y) ^ brother (x, y) → Sibling (x, y)
- All those who did not complete the homework did not attend the class today
  ∀x: student (x) ^ ¬ complete (x, hw) → ¬ attend (x, class)

- All purple mushroom are not necessarily poisonous
  $\forall x: mushroom(x) \wedge color(x, purple) \rightarrow \neg poisonous(x)$
- Peter did not fail in all subjects
  $\exists x: subject(x) \wedge \neg fail\_in(peter, x)$
- All that glitters is not gold
  $\exists x: thing(x) \wedge glitter(x) \rightarrow \neg gold(x)$
- Anything anyone watches in multiplex is a movie
  $\forall x: \forall y: person(x) \wedge thing(y) \wedge watch\_in\_multiplex(x, y) \rightarrow movies(x, y)$
- Rohit Sharma can't score 100 in every match
  $\exists x: cricket\ match(x) \wedge \neg score\_100(rohit\_sharma, x)$
- Ravan was a Brahmin and was killed by Lord Ram
  $brahmin(Ravan) \wedge lord(ram) \wedge killed\_by(Ravan, Ram)$
- Rohan eats carrot and is still alive
  $Eats(Rohan, Carrot) \wedge Still\_alive(Rohan)$
- Anything Anyone eats is not killed by is food
  $\forall x: \forall y: person(x) \wedge thing(y) \wedge eats(x, y) \wedge \neg killed\_by(x, y) \rightarrow food(y)$
- John was killed by someone in his appartment
  $\exists x: person(x) \wedge killed\_in\_appartment(x, John)$
- Everyone can't swim in floods
  $\exists x: person(x) \wedge \neg swim(x, floods)$
- Shinchan is not everyone's favourite cartoon
  $\exists x: person(x) \wedge cartoon(shinchan) \wedge \neg favourite(x, cartoon)$
  or $\exists x: person(x) \wedge \neg favourite\_cartoon(x, Shinchan)$
- Everyone likes street food at Gujarat University
  $\forall x: person(x) \wedge place(Gujarat\ University) \wedge likes(x, street\_food, gujarat\_university)$
- Everyone travelled thousand kms on bike, but

Aman is excited to drive more.
$\forall x$: Person(x) $\wedge$ travel_on_bike(x, thousand_km) $\wedge$ excited_to_drive_more(aman)

- Dilip was absent in yesterday's lecture
Student(Dilip) $\wedge$ day(yesterday) $\wedge$ absent(dilip, yesterday lecture)
- If people will create huge crowd, then corona will spread.
$\forall x$: Person(x) $\wedge$ create(x, huge_crowd) $\rightarrow$ spread(corona)
- Ronaldo is plays football in Stadium
play_in_stadium(Ronaldo, football)
or Player(Ronaldo) $\wedge$ play_in_stadium(Ronaldo, football)
- Everyone can fly if they have wings
$\forall x$: person(x) $\wedge$ have_wings(x) $\rightarrow$ fly(x)
- Everybody believes in Stories
$\forall x$: person(x) $\wedge$ believe_in(x, stories)
- Nobody likes Stories believes in Stories
$\neg\forall x$: person(x) $\wedge$ believe_in(x, stories)
- All new sentences begin with Capital letters
$\forall x$: new_sentence(x) $\wedge$ begin_with(x, Capital_letters)
- When mam takes attendance some Students don't respond
$\exists x$: student(x) $\wedge$ take(mam, attendance) $\rightarrow \neg$(respond(x))
- Anything anyone can get to read from library
$\forall x$: $\forall y$: thing(x) $\wedge$ person(y) $\wedge$ read_in(y, x) $\rightarrow$ library(x)
- Peter's AI book is good for learning
$\exists x$: ai_book(x) $\wedge$ author(peter) $\wedge$ written_by(x, peter) $\wedge$ good_for_learning(x)
- Everyone can't understand python programming

Language
$\exists x: Person(x) \wedge Programming\_language(Python) \wedge \neg Understand(x, Python)$

- Story
- Ram was a mighty king
  mighty_king(Ram)
- Ram ~~was~~ has a crown on his head
  wear_crown(Ram_head) or
  wear_on_head(Ram, crown)
- The crown has beautiful multicolored gems
  $\forall x: gem(x) \wedge beautiful(x) \wedge multicolored(x) \wedge crown(x)$
- All courtiers are attached towards Ram's crown
  $\forall x: courtier(x) \wedge attracted\_towards(x, Ram\ crown)$
- Ram slept in night and next day morning
  found his crown missing
  $sleep\_in(Ram, night) \wedge day(next\ day) \wedge time(morning) \wedge \neg found(Ram, crown)$
- Ram instructs his minister to investigate
  $instruct(Ram, minister) \wedge investigator(minister)$
- The minister suspects three courtiers Mohan, Rohan and Sohan.
  $courtier(Mohan) \wedge courtier(Rohan) \wedge courtier(Sohan) \wedge suspect(minister, Rohan) \wedge suspect(minister, Mohan) \wedge suspect(minister, Sohan)$
- Rohan is a thief
  thief(Rohan)
- Rohan is a thief of a crown.
  ~~Has~~ Stolen_by(crown, Rohan)
- Adarsh and Yash are friends

Person(adarsh) ^ Person(yash) ^ friend(adarsh, yash) ^ friend(yash, adarsh)

Backward Chaining - Reduction - inference making

Story 2

1. If it croaks and eat flies, it is frog
   ∀x: animal(x) ^ croaks(x) ^ eats(x, flies) → frog(x)
2. If it chirps and sings, it is canary
   ∀x: bird(x) ^ chirps(x) ^ sings(x) → canary(x)
3. A frog is green in color
   ∀x: frog(x) → color(x, green)
4. A canary is yellow in color
   ∀x: canary(x) → color(x, yellow)
5. Ritz croaks and eats flies
   croaks(ritz) ^ eat(ritz, flies)

What is color of Ritz ?

→ green. 5, 1, 3

Adding new facts

Ritz is an animal

animal(ritz)

color(ritz, green)
↑ (3 substitution)
frog(ritz) (x = ritz)
↑ (1 substitution)
animal(ritz) croaks(ritz) ^ eats(ritz, flies) x = ritz
↑ 6
croaks(ritz) ^ eats(ritz, flies)
↑ 5
NIL

Rule

Quantifier
∀x, ∃x

• Story 3

1. Steve likes easy courses.
   $\forall x$: course(x) ^ easy(x) → likes(steve, x)
2. Science courses are hard.
   $\forall x$: course(x) ^ dept(x, Scien(e) → hard(x)
3. All courses in basket weaving department are easy
   $\forall x$: course(x) ^ dept(x, basket_weaving) → easy(x)
4. BK301 is a basket weaving course
   course(BK301) ^ dept(BK301, basket-weaving)

Which course Steve likes

BK301 {3, 1} likes(Steve, BK301)

↑ (1, substitution) BK301

course(x) ^ easy(BK301)

↑ {4} a.

4a = course(BK301) · easy(BK301)

↑ (3, substitution) x = BK301

4b = dept(BK301, basket weaving) course(BK301) ^ dept(BK301, basket_weaving)

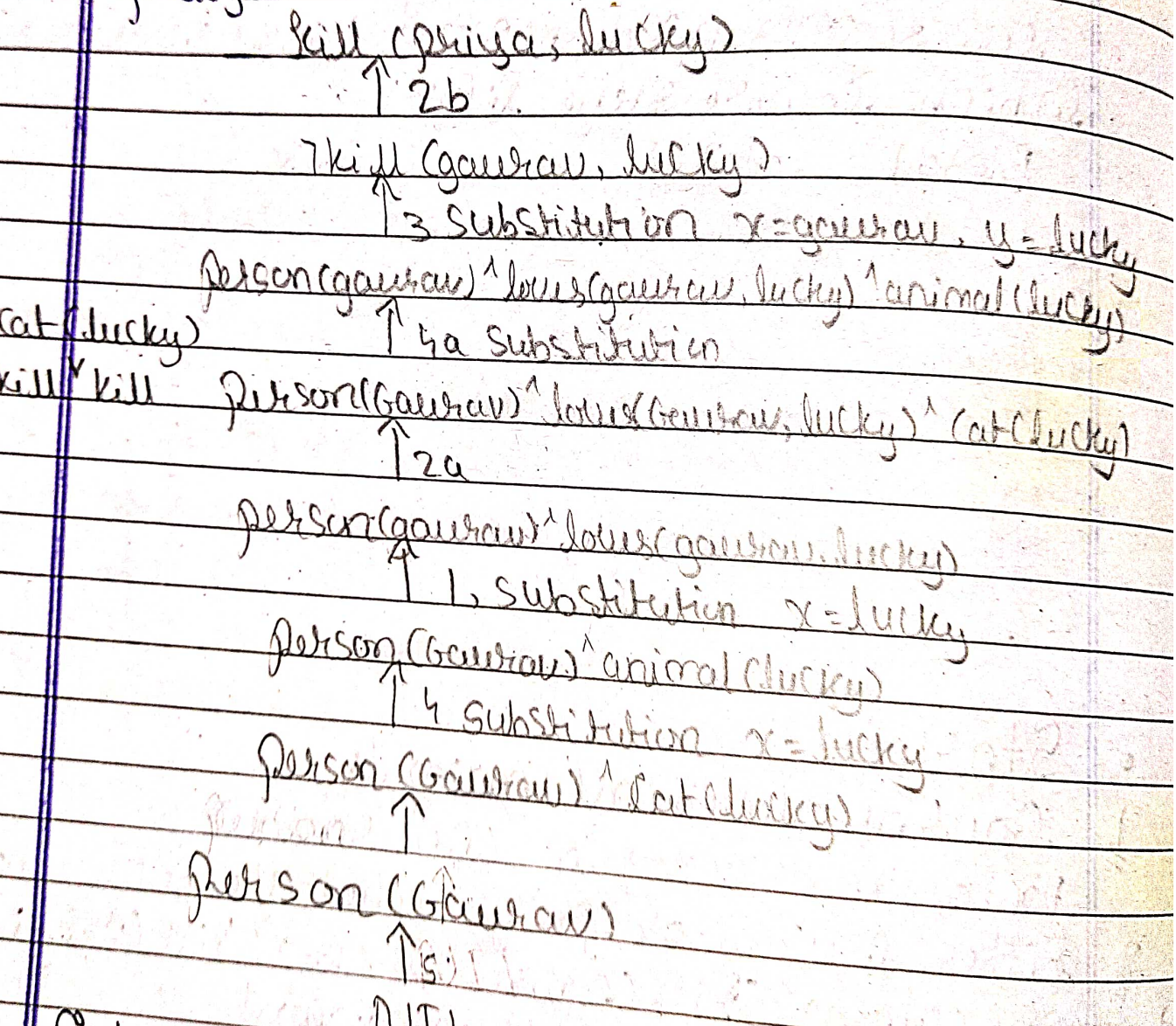
↑ {4}

NIL

• Story 4

1. Gaurav loves all animals
   $\forall x$: animals(x) → love(Gaurav, x)
2. Either Gaurav or Priya killed the cat name lucky
   Cat(lucky) ^ [kill(Gaurav, lucky) ∨ kill(Priya, lucky)]
3. No animal lover kills an animal
4. If there is an animal lover, he will not kill any animal
   $\forall x$: $\forall y$: person(x) ^ animal(y) ^ loves(x, y) → ¬kill (x, y)

4. All cats is an animals
$\forall x: Cat(x) \rightarrow Animal(x)$
5. Gaurav is a Person.
Person(Gaurav)
6. Priya is a person.
Person(Priya)
8. Who killed Lucky?
Priya



- Q Today 5

1. John likes all kinds of food.
$\forall x: food(x) \rightarrow likes(John, x)$
2. Apples are food

food(apple)

3. Anything Anyone eats and is not killed by is food
$\forall x : \forall y : person(x) \wedge thing(y) \wedge eats(x,y) \wedge \neg killed\_by(x,y) \rightarrow food(x,y)$

4. Bill eats peanuts and is still alive
$eats(Bill, peanuts) \wedge alive(Bill)$

5. Sue eats everything that Bill eats
$\forall x : food(x) \wedge eats(Bill, x) \rightarrow eats(Sue, x)$

6. Peanut is a thing
$thing(peanut)$

7. All who are alive are not killed by anything
$\forall x : \forall y : person(x) \wedge thing(y) \wedge alive(x) \rightarrow not\_kill\_by(x,y)$

8. Bill is a person.
$person(Bill)$

9. Prove that John likes peanuts
$4a = eats(bill, peanuts)$ 8, 6, 4b, 7, 4a, 3, 6, 8, 1
$4b = alive(bill)$

Examples

- Everyone is loyal to someone
$\forall x : \exists y : person(y) \wedge person(y) \wedge loyal\_to(x,y)$
- Everyone loves someone
$\forall x : \exists y : person(x) \wedge person(y) \wedge loves(x,y)$
- There is someone who is loved by everyone
$\exists y : \forall x : person(x) \wedge person(y) \wedge loves(x,y)$ or
$\exists y : \forall x : person(x) \wedge person(y) \wedge loved\ by(y,x)$
- Someone is loyal to everyone
$\exists y : \forall x : person(x) \wedge person(y) \rightarrow loyal\_to(x,y)$

likes(John, peanut)

↑ 3

food(peanuts)

↑ 4a

person(bill) ^ thing(peanut) ^ eats(bill, peanut) ^ ¬killed_by(B, f

person(bill) ^ thing(peanut) ^ ¬killed_by(bill, peanut)

↑ 6, 8 (add that one by one)

killed_by(bill, peanut)

↑ 7 (x = bill, x = peanut)

person(bill) ^ thing(peanuts) ^ alive(bill)

↑ 4b

person(bill) ^ thing(peanuts)

↑

person(bill)

↑ 8

NIL

Story 6:

1. Marcus was a man
   man(marcus)
2. Marcus was a pompeian
   region(marcus, pompeian)
3. All pompeians were romans
   ∀x: people(x) ^ region(x, pompeian) → live_in(x, roman)
4. Caesar was a ruler
   ruler(caesar)
5. All romans were either loyal to caesar or hated him.

$\forall x: people(x) \wedge live\_in(x, name) \rightarrow loyal\_to(x, caesar) \vee hate(x, caesar)$

6. Everyone is loyal to someone
$\forall x: \exists y: People(x) \wedge people(y) \rightarrow loyal\_to(x,y)$

7. people try to assasinate rulers they are not loyal to
$\forall y: \forall x: ruler(y) \wedge people(x) \wedge try\_to\_assacinate(x,y) \rightarrow loyal\_to(x,y)$

8. Marcus try to assacinate Caesar
try_to_assicinate(marcus, caesar)

* Adding facts

9. Man are people
$\forall x: man(x) \rightarrow People(man)$

10. Caesar was Ruler of rome
ruler(Caesar, rome)

Q Was marcus loyal to Caesar?

Conclusion: Marcus was not loyal to Caesar.

loyal_to(marcus, caesar)
↑ Note: this is L.H.S of Sentence 7
ruler(caesar) ^ people(marcus) ^ try_to_assacinate(M, C)
↑ (Add 4)
people(marcus) ^ try_to_assacinate(marcus, caesar)
↑ (Add 8)
people(marcus)
↑ (9, Substitution, x = marcus)
fact 1: man(marcus)

NIL

Ch - 7 Uncertainity

Types of Uncertainity

- Uncertainity in prior knowledge
E.g Some causes of a disease are unknown and are not represented in the knowledge base of a medical agent

- Uncertainity in actions
E.g actions are represented with relatively short lists of preconditions, while these lists are in a fact arbitrary long.

- For E.g. You are driving towards college in the morning.
- It must not have flat tyres
- There must be fuel in tank
- The battery must not be dead
- The ignition must work.
- You must not have lost the car keys.
- No truck should obstruct the driveaway.
- It must not have been stolen during the night
- You must have suddenly become blind or paralytic etc.

It is not possible to list all of them

- Uncertainity in Perception
E.g Sensors do not return exact & complete information about the world; a robot

never knows exactly in absolute position.
- What we call uncertainity is a summary of all that which is not explicity taken into account in the agent's knowledge base.

- Sources of uncertainity
1. Ignorance
2. Inefficiency

Knowledge and Inexact Reasoning
- inexact knowledge (truth of α not clear)
- incomplete knowledge (lack of knowledge about α)
- defaults, beliefs (assumption about truth of α)
- contradictory knowledge (α true or false)
- vague knowledge (truth of α not old)

Inexact Reasoning
- CF theory - uncertainity
- Fuzzy - vagueness
- Truth Maintenance - belief, defaults
- Probability Theory - likehood of events

Inexact knowledge - Example
- One day way back home, person A walk towards the bus stop. A few hundred yards away, A sees someone and is quite sure that it's his next door neighbour B, who usually goes by car to university. A screams B's name

Q. Which forms of inexact knowledge and reasoning are involved here?

→ default - A wants to take a bus
belief, (un)certainity - it's the neighbour of B
probability, default, uncertainity - the neighbour goes by car
default - A wants to get a lift
default - A wants to go home

Examples of Inexact knowledge:

- Fuzzy - a few hundred yards
define a mapping from "#hundreds" to 'few', many
not uncertain or incomplete but graded, vague
- Probabilistic: the neighbour usually goes by car
probability based on measure of how often he takes car; calculates always p(F) = 1 - P(¬F)
- Belief - Its his next door neighbour B
"reasoned assumption", assumed to be true
- Default - A wants to get a lift assumption based on common sense knowledge

Truth Maintenance System

- Truth Maintenance Systems (TMS) also called Reason Maintenance Systems, are used with Problem solving systems, in conjuction with Inference Engines (IE) such that rule-based

inference systems to manage as a Dependency Network the inference engine's beliefs in given sentences.

Problem Solving System

Truth Maintenance

- A Truth Maintenance System tries to adjust the knowledge base or fact base upon changes to keep it consistent and correct.

- A TMS uses dependencies among facts to keep track of conclusions and allow revision/retraction of facts and conclusions.

- Why TMS?
The need of TMS arises when changes in the knowledge base lead to inconsistency/incorrectness among the facts → non monotic reasoning

Monotonic Reasoning

- A clause logic is monotonic if:
- anything that could be concluded before a clause is added can still be concluded after it is added.

- adding knowledge does not reduce the set of propositions that can be derived.

- there is no need to check for in consistency between new statements and the old knowledge
- Monotonic systems are not good in real problem domains where the information is incomplete situations change and new assumptions are generated while solving new problems

Non Monotonic Reasoning

- A clause logic is non monotonic if
- Adding a new fact might lead to an inconsistency which requires the removal of one of the contradictory facts
- In first order predicate logic this does not happen once a fact is asserted, it's forever true.
- Non monotonic reasoning is based on default reasoning or "most probabilistic choice"
- For example when we visit a freind's home we buy chocolates for children because we believe that most children like chocolates.

Certainity Factor Theory

- Certainity Factor CF of Hypothesis H
- Ranges between -1 (denial of H) and +1 (confirmation of H)

- Based on measures of belief MB and disbelief MD
- MB is belief that H is true
- MD is belief that H is not true
- MB is not 1-MD - It's not like probabilities
- Experts determine values for MB, MD of H based on given evidence E → Subjective

Characteristics of Certainity Factors

Aspect	(Believed) Probability	MB	MD	CF
Certainly true	P(H|E) = 1	1	0	1
Certainty false	P(¬H|E) = 1	0	1	-1
No evidence	P(H|E) = P(H)	0	0	0

Ranges

measure of belief $0 \leq MB \leq 1$

measure of disbelief $0 \leq MD \leq 1$

Certainity factor $-1 \leq CF \leq +1$

Note:

$P(H|E) = 1 \Rightarrow MB = 1$, $MD = 1$ $\rightarrow CF = -1$

$CF = H$

Evidence

$CF = 1$ | $CF = 1$

$P(H|E) = P(H)$

5 5

$P(\neg H|E) = 1$

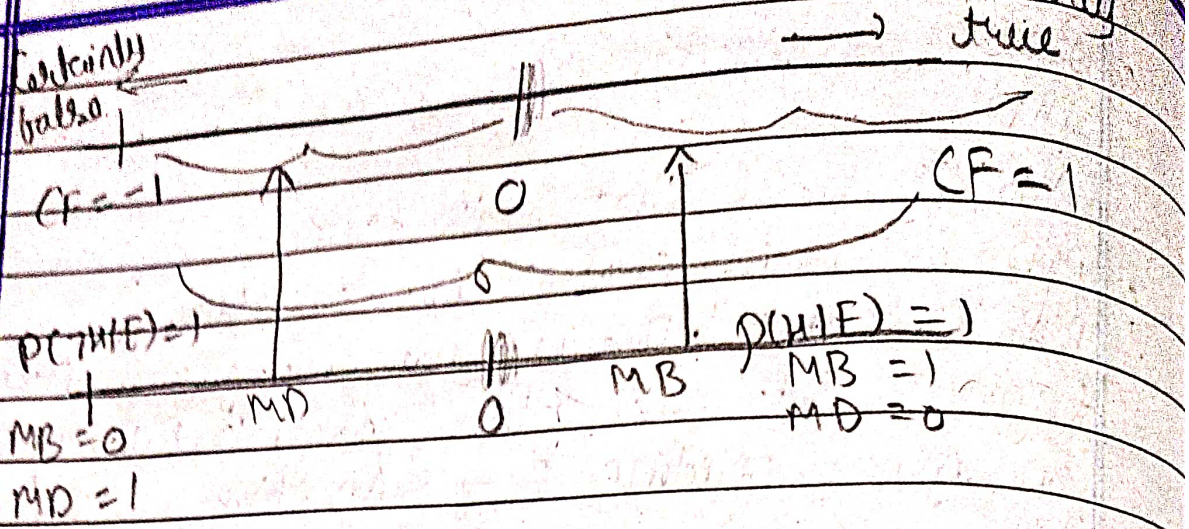
Certainly false

Certainly true

$CF = -1$ 0 $CF = 1$

$P(\neg H|E) = 1$ $P(H|E) = 1$

$MB = 0$ $MD = 1$ $MB = 1$ $MD = 0$



Basis of Probability Theory

- Mathematical approach to process uncertain information
- Sample Space (event) set: $S = (x_1, x_2, \dots x_n)$
  - Collection of all possible events
- Probability $p(x)$ is likehood that the event $x_i \in S$ occurs
  - Values are in $[0, 1]$
- total probability of sample space is 1

$$\Sigma p(x_i) = 1 \quad \text{for all } x_i \in S$$

E.g $S = \{\underset{x_1}{\text{Heads}}, \underset{x_2}{\text{Tails}}\}$

$x_i \in S$

$P(x_i)$ $p(\text{heads}) = [0,1]$ $p(\text{tails}) = [0,1]$

$p(S) = 1$ $(x \in S)$ $\Sigma p(x_i) = 1$

Compound Probabilities

- For mutually exclusive events
  - Cannot occur together at same time
  - Eg one dice and events "1" and "6" one coin and event "heads" and "tail"
- Joint Probability of two different events A

$P(A \cap B) = 0$

Eg: Throw a dice and show both "1" and "6" cannot happen.

- Union probability of two events A and B
  $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
  E.g: Throw coin and show either "heads" or "tail"

Basis of Probability Theory

- Mathematical approach to process uncertain information
- Sample Space (event) Set: $S = \{x_1, x_2 \dots x_n\}$
  - Collection of all possible events
- Probability $p(x_i)$ is likehood that the event $x_i \in S$ occurs.
- Values are in $[0, 1]$
- total probability of Sample Space is 1
  $\Sigma p(x_i) = 1$ for all $x_i \in S$

Fuzzy Set Theory

- Aimed to model and formalize "vague" Natural Language terms and expressions
  E.g: Peter is relatively tall.

- Define a set of Fuzzy sets (predicates or categories) like tall, small.

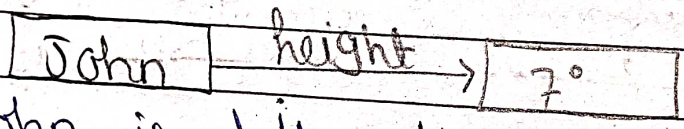
Semantic Networks

Semantic Networks

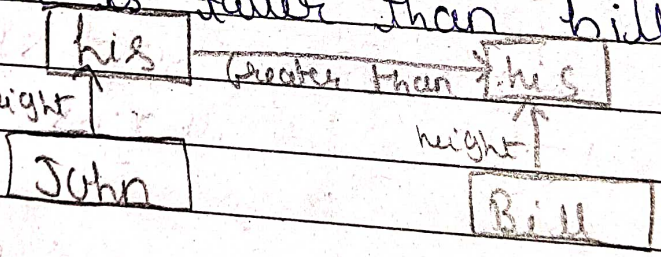
Semantic networks consist of compone
of logic systems that specify the meanin
of well performed expressions
- In semantic networks the information is
represented as a set of nodes connected
to each other by a set of labelled arcs whic
represents the relationship between the node

E.g

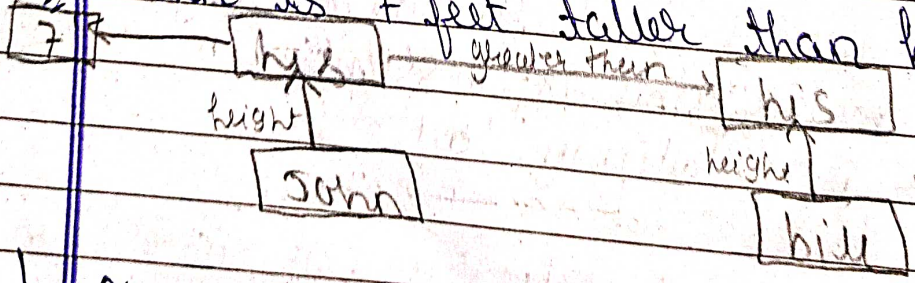
1. John is 7 feet tall



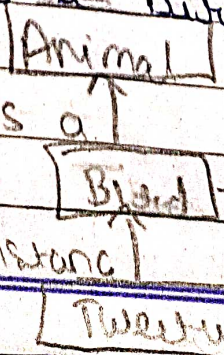
2. John is taller than bill

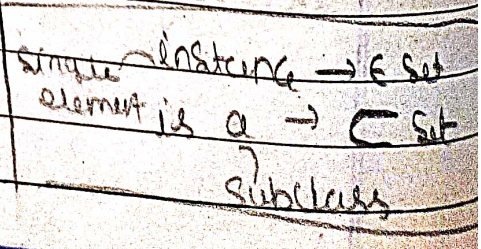


3. John is 7 feet taller than bill

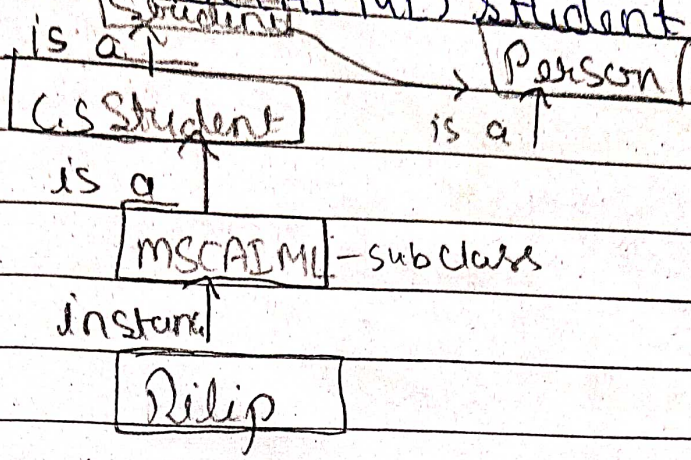


4. Tweety is a bird



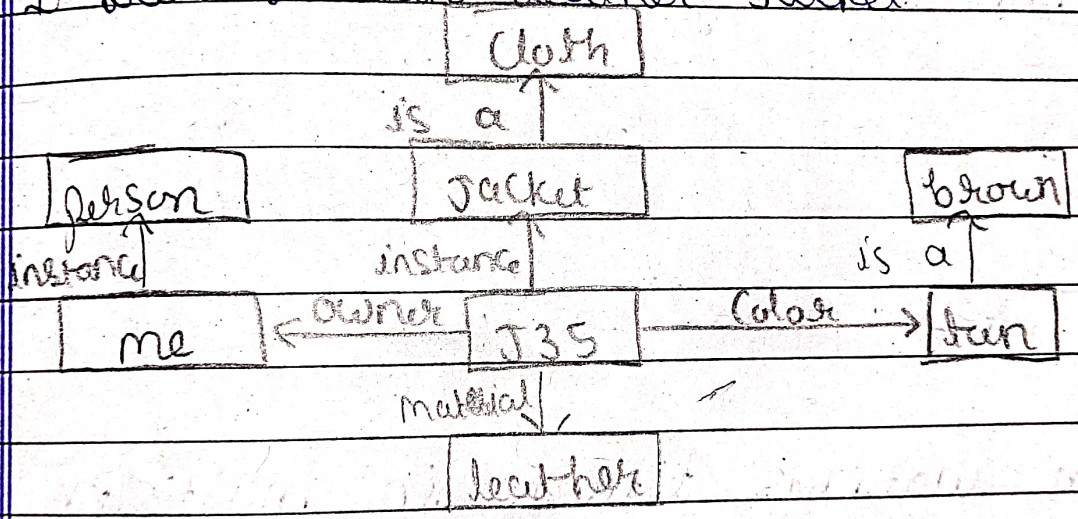


4. Dilip is a MSC(AI ML) Student

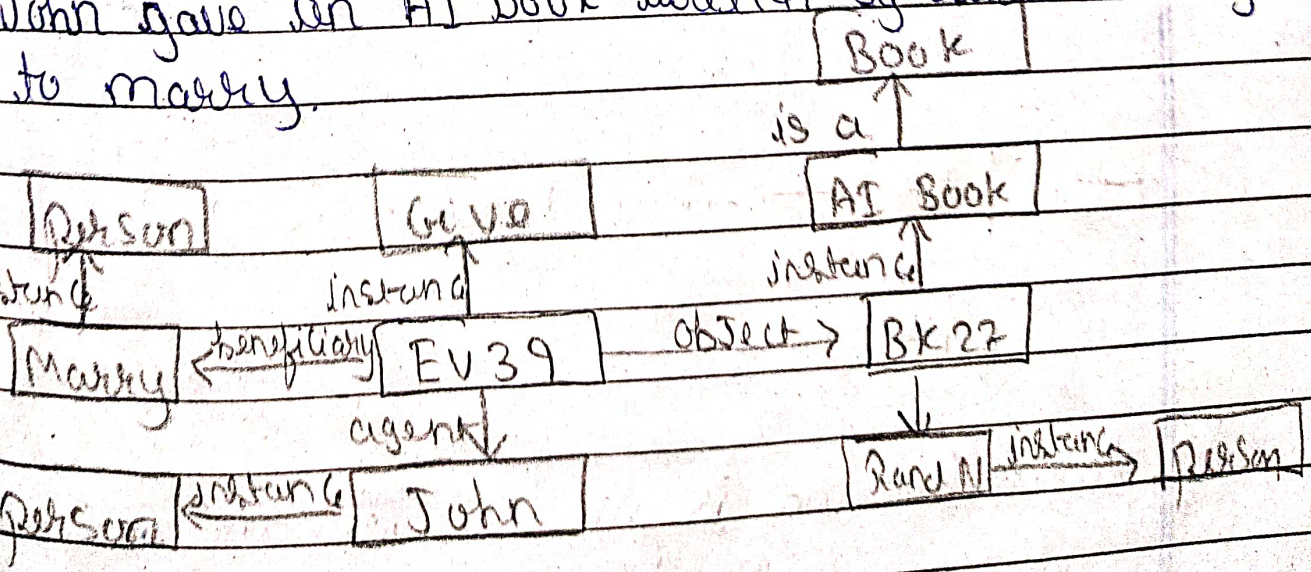


C.S: Computer science

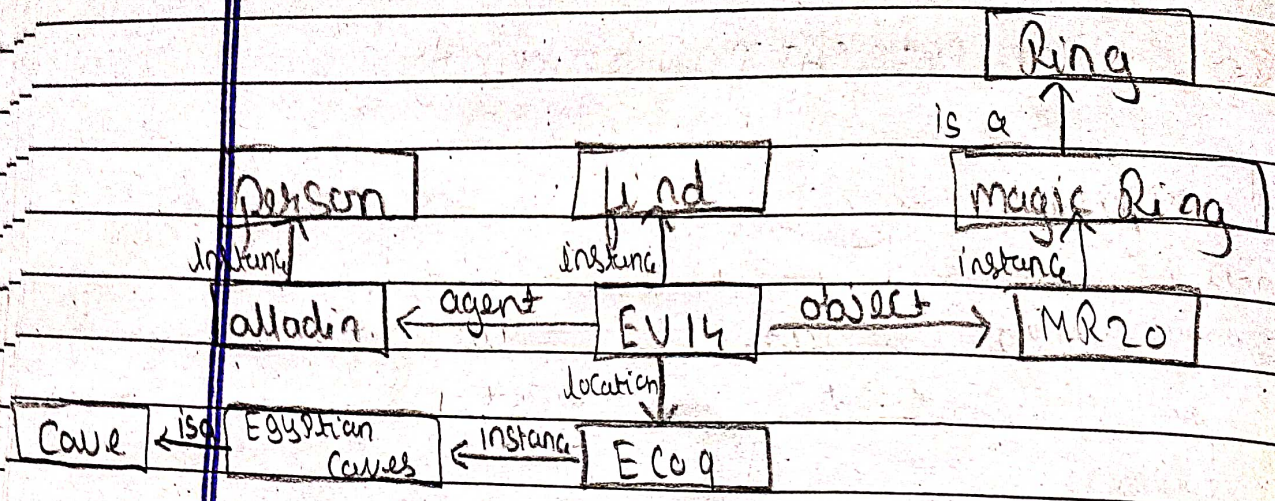
5. I owe a tan leather Jacket



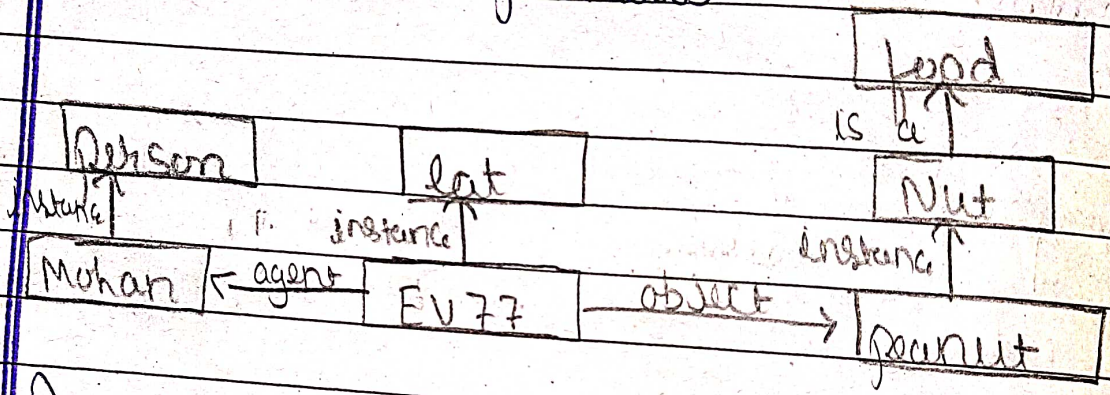
6. John gave an AI Book written by Russel Norvige to marry.



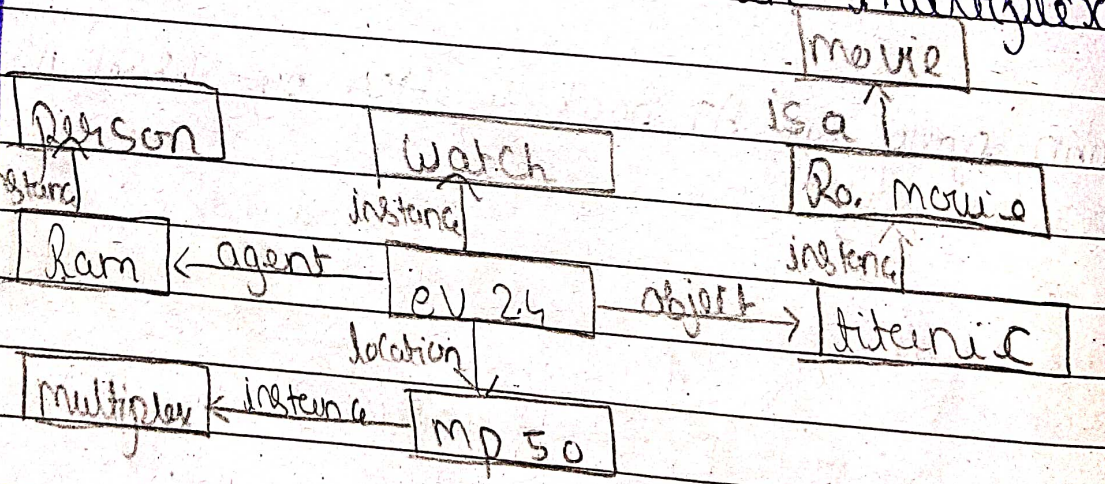
7. Alladin finds the magic ring in egyptian



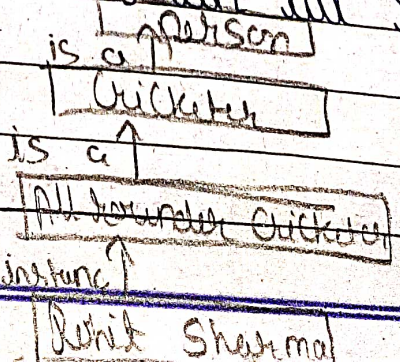
8 Mohan eats peanuts



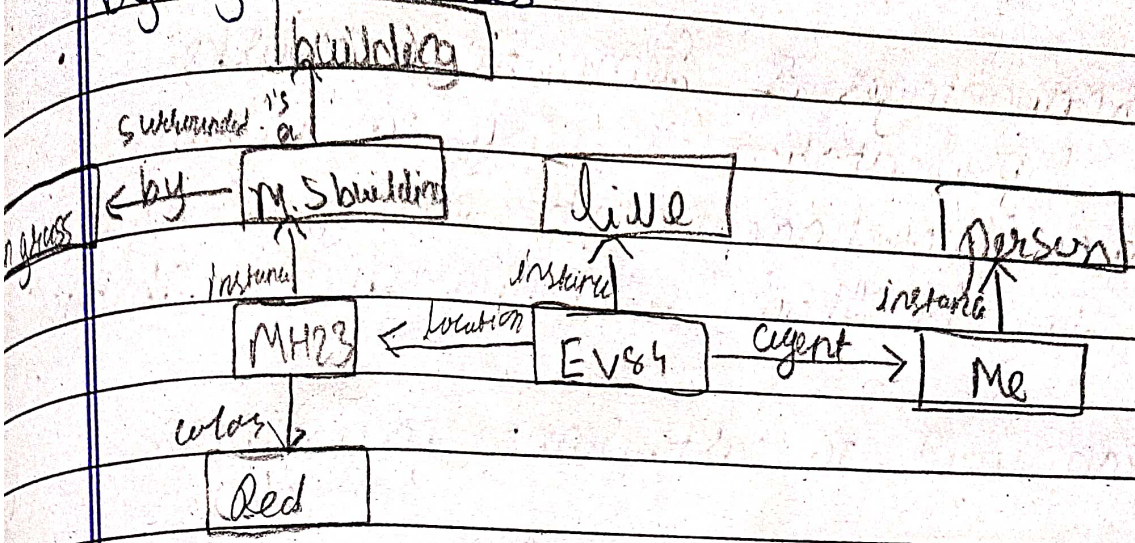
9 Ram watches titanic in multiplex



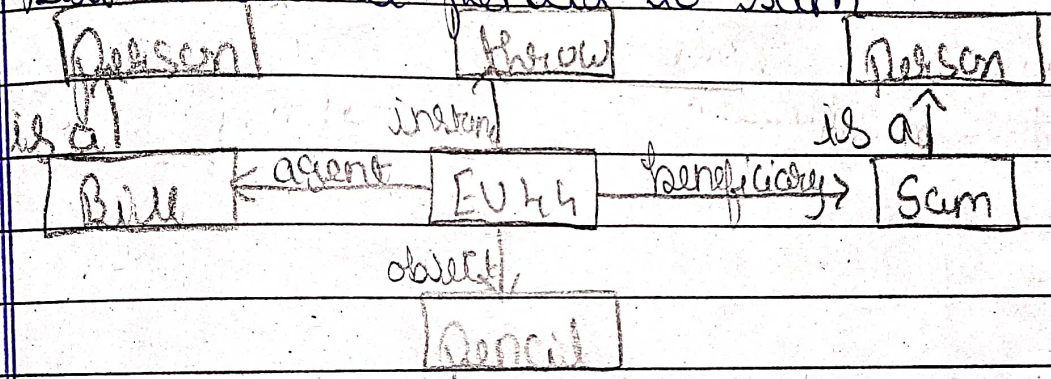
10 Rohit Sharma is an all rounder cricketer



11 I live in a tall multi storied building surrounded by green trees



12 Bill threw a pencil to Sam

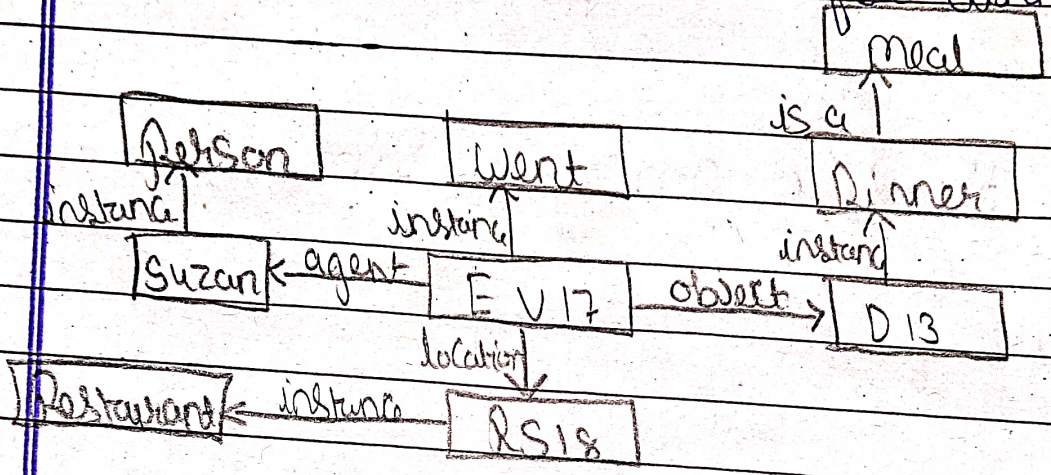


* Problem with Semantic Networks
- Binary relations are easy to represent, but complex sentences make the representation more complex
- Representation of negation is not possible E.g Ram does not know skating
- Representation of disjunction is not possible E.g Ram likes to watch adventures or thriller movies.
- Representation of quantified expression is not possible E.g Everyone likes ice cream
- It requires large amount of memory to

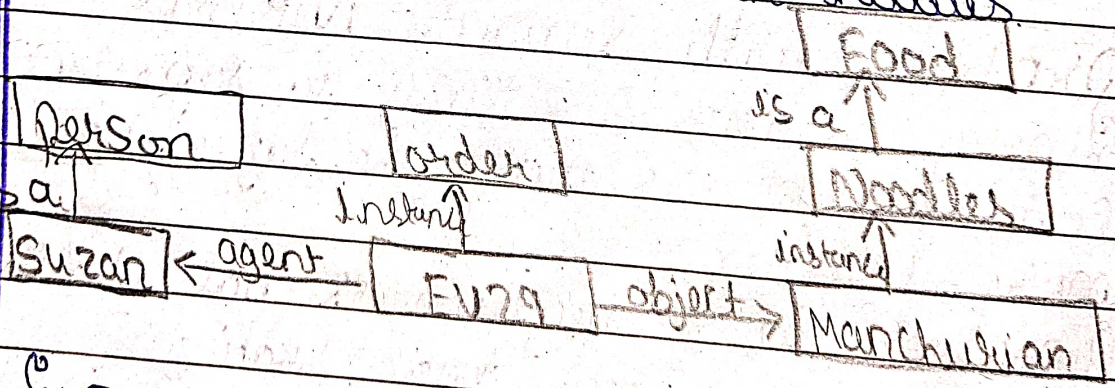
Store the facts

* Disadvantages of Semantic Networks
- The representation is easy to visualize.
- Related knowledge can be easily clustered.
- It can be shown upto any level of abstractio

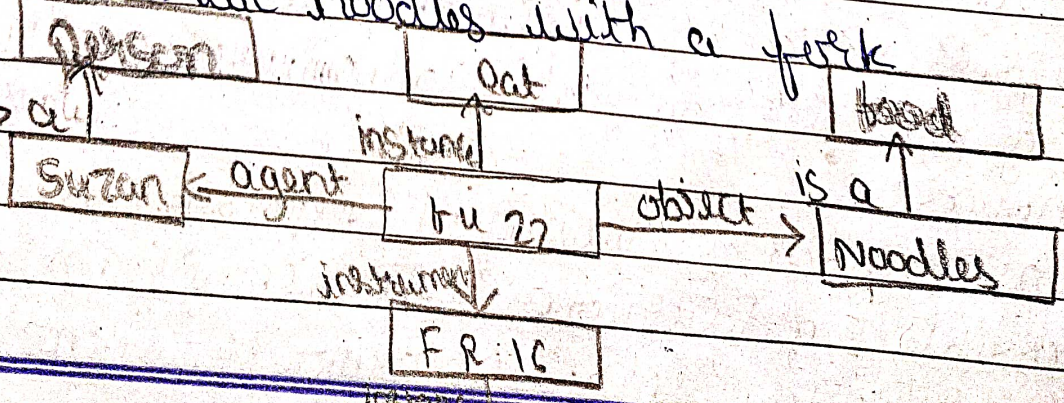
13 Suzan went to restaurant for dinner.



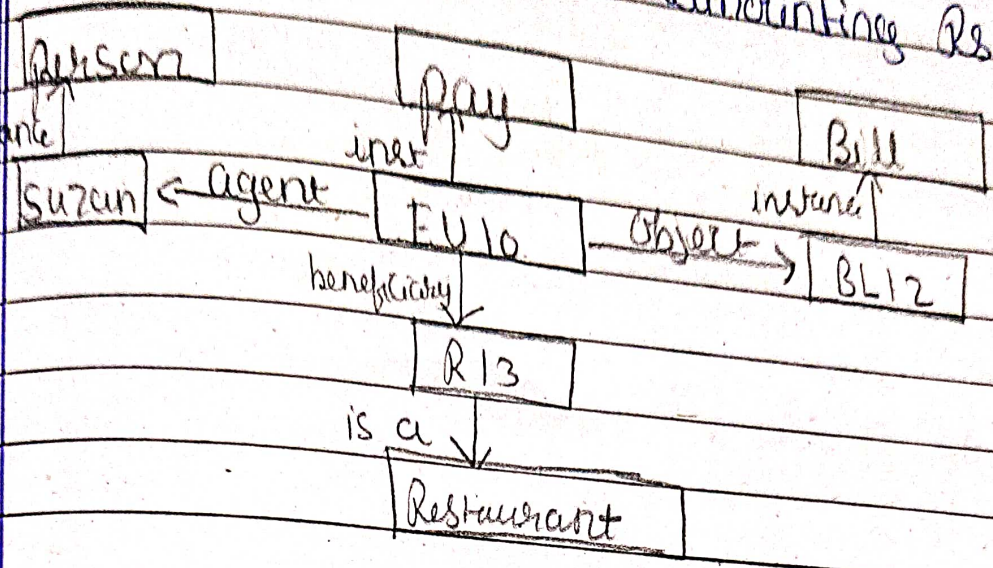
14 Suzan order manchurian noodles



15 Suzan ate noodles with a fork



16 Suzan paid the bill amounting Rs 350



17 Suzan went home in a taxi