Que:- What is intelligence?

Ans:- Intelligence has been defined in many different ways including one's capacity for logic, understanding, self-awareness learning, emotional knowledge, planning, creativity and problem solving.
It can be more generally described as the ability to perceive information, and retain it as knowledge to be applied towards adaptive behaviour with in an environment or content.

Artificial intelligence is intelligence in machines. It is commonly implemented in computer systems using program software.

Que:- What is Artificial Intelligence?

Ans:- The theory and development of computer ~~science~~ systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.

Que:- What is True AI?

Ans:- We have short term memory and

computers have long term memory that's why we are not able to do fast calculation. Humans have pattern recognization.

Artificial Intelligence problem :—
Algorithm for playing chess.
3 lit water jug & 4 lit water jug. No measuring marks on them. Water is available to fill water in their jugs. Get exactly 2 lit of water in the 4 lit jug.

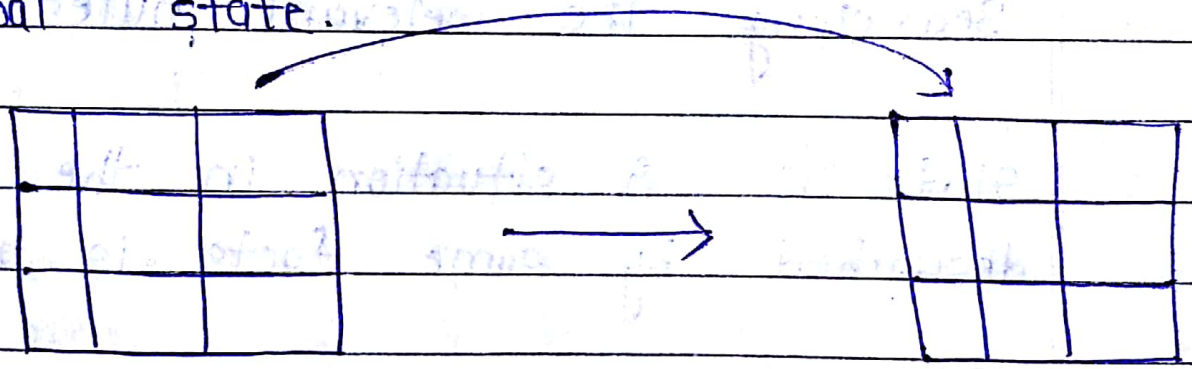
27/07/16

Solving AI problem :—
Problem: Playing chess / To write a program that could play chess/

For the programmer 3 things have to be specified —

1.) The starting position of the chess board.
2.) The rules that define the legal moves.
3.) The board position that represent a win for one side or the other.

Formal representation of the problem :-
The starting chess board position can be represented by 8×8 array. where symbols occupies the appropriate position as in the official chess opening position.
Our goal is any position that the opponent does not have any legal move and his/her King is under attack.
The legal moves that provides the way of getting from initial state to goal state.



Left side serve as a position to be matched against the current board position.

Right side describe the changes that have been made by reflecting the move.

X Suppose if we write the rules like this then we have to write rules for 10^{120} possible chess board position

Nobody can supply the complete set of rules.

$60 \times 60 \times 24 \times 365 \times 10^{-9} \times ? = 10^{120}$

It takes too much time. It can not be done without comming any mistake.

* No problem can handle all these rules.
✓ Storing the rules.
✓ Searching the relevant rules.

State :- A situation in the. would described by some facts is a state.

In physics :-

A state is a set of measurable physical parametens fully describing some part of the universe.

In computer Science, A process state is a set of value currently in registers, buffers and stack so stack is a Snapshot of a process

in time.

categories :-

Instantaneous (last for only sometime)/non-Instantaneous

→ continuous / Discrete

↓ state transition

Problem state :- List of facts / information required to solve the problem in one bundle.

28/07/16

Good state space representation for water jug problem :-

quantity of jug (x, y), x = quantity of water 3 lit jug (0,1,2,3)

initial state (0,0). y = quantity of water 4 lit jug (0,1,2,3,4)

(0,0) initial state

(3,0)

(0,3)

(3,3)

(2,4)

(2,0)

(0,2) Goal state

require rule for state transition which is called production rules.

Production rule -1 —

$(n, m) \rightarrow (3, m)$

if $(n > 3)$

Corresponding rule 4 —

$(n, m) \rightarrow (n, 4)$ Fill 4 lit Jug

if $(n < 4)$

$(n, m) \rightarrow (n, 0)$ empty the 4 lit jug

if $(m > 0)$

$(n, m) \rightarrow (0, m)$

if $(n > 0)$ empty the 3 lit jug

pour some water out of 4 lit jug to 3 lit jug :—

$(n, m) \rightarrow (n-d, m)$

if $(n > 0)$

1/00/16

$(n, m) \rightarrow (n, m-d)$ pour some water out of 4L jug

if $m > 0$

$(n, m) \rightarrow (n-d, m)$ pour some water out of 3L jug

if $n > 0$

$(n, m) \rightarrow (n-(4-m), 4)$ pour the water from 3L jug to 4L

if $(n+m) > 4$

$(m, n) \rightarrow (3, m-(3-n))$ corresponding rule

if $(n+m) > 3$ pour the water from 4L jug to 3L jug until it become full.

$(4, 0) \rightarrow (0, 2)$ special rule specific solution

Necessary things to provide formal description of AI problem

* Define problem stall space.
* Problem state space should contain all the possible configuration of the relevant objects.

* Specify one more states within the space that describe possible situation from where which problem solving process may start.
  These are called initial states.

* Specify one on more states that would be acceptable as solution to the problem. These states are called goal

states.

* Specify a set of rules that describe the action available also called state transition rules or production rules.

2/08/16

* What are the unstated assumptions are present in the informal problem description.

- How general should the rules be made?
- How much work require to solve the problem should be pre-computed and represented in the rules.
- General steps required to build a system which solve AI problem.

1.) Define the problem precisely.
2.) Analyse the problem - to find the appropriateness of the techniques to be applied in order to solve the problem.

3.) Isolate and represent the problem domain knowledge.

4.) Apply the best technique to solve it.

5.) Good control strategy

✓ State space search technique :-

We can 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 the chessboard.
We can play chess by starting at an initial state using a set of rules to move from one state to another and attempting to end up one of the goal state.

The state space representation structure corresponds to the structure of problem solving in two way :-

(i) It allows a formal definition of a problem as the need to convert some given situation into some desired situation using set of permissible operation.

(ii.) It permit us to define the process

soluting problem as a combination of known technique and search.

3/08/16

AI problem characteristic :—

1.) Is the given problem is decomposable or not?

Breaking the problem into smaller sub-programs, each of which can be solve using small collection of specified rules.

Integration

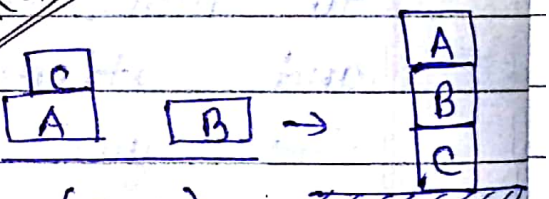
$$\int (x^2 + 3x)\, dx$$

$$\int x^2\, dx + \int 3x\, dx + c$$

$$\frac{x^3}{3} + \frac{3x^2}{2} + c$$

Here subproblem are independent and the problem is decomposable.

Block world



on (C, A)

on (B, C) &
on (A, B)

clear (x) → on (x, table) (block has nothing on it)

pickup x
and put it on the table

clear (x) and clear (y) → on (x, y)

put x on y

Subproblems are not independent so the problem is not decomposable.

2.) Can solution steps be ignored or atleast undone if they prove unwise?

Ignorable	Recoverable	Irrecoverable
Mathematical theorem proving Ignorable in which solution steps can be ignored. This type of problem can be solved using a simple control structure that never backtracks easy to solve	8 puzzle problem Start 2 8 3 1 6 4 7 5 Goal 1 2 3 8 4 7 6 5 The 8 puzzle is a square tray in which are placed 8 tiles, the remaining ninth square is uncovered. each tiles has a number on it. A tile that in adjacent to the blank space can be side into that space. A game consist of a starting position and a specified goal position. The goal is to transform the starting position into goal position by sliding the tiles around	Problem of playing chess Irrecoverable in which sol^n steps can not undon. Irrecoverable problem will be solved by a system that expends a great deal of effort making each decision since that decision must be final. planning process.

type of problem can be solved by a slightly more complicated control strategy (It does sometime make mistakes need to be recovered using back tracking. back tracking can be implemented using stack.

4/08/16

3) Is the problem universe predictable?

8-puzzle	card game, bridge game
Certain outcome after the state transition. Everytime we make a move & we know that exactly what will happen It is possible to plan the entire sequence of moves & be confident about what will be the	uncertain outcome plan revision after the feedback from the environment + probability

resulting state.
We can plan to avoid having to undo actual moves (Backtracking can be used during planning)
planning <u>without</u> feedback from environment.

4.) Is a good solution absolute or relative ?

Problem I	Problem II
Consider the following facts on a database -	Travelling salesman • Hard to solve • This type of problem called relative. • It is best path problem. • Much more exaustive search will required to be performed.
1.) Marcus was a man.	
2.) Marcus was a pompeian.	
3.) Marcus was born in 40AD	
4.) All men are mortal.	
5) All pompeii died when the valcano erupted in 79 AD.	
6.) No mortal lives longer than 150 years.	
7.) It is now 2016 AD.	
Is marcus alive now?	

Sol:- (2), 15) & (7)

1, 4 → Marcus is mortal

- easy to solve.
- This type of problem called absolute.
- It is any path problem
- It can be solved in a resonable ammount of time (good heuristic can be used to explore the better path.

5.) Is a solution in a state or path ?

Water Jug
The bank president ate a dish of pasta salad with fork.

Bank <

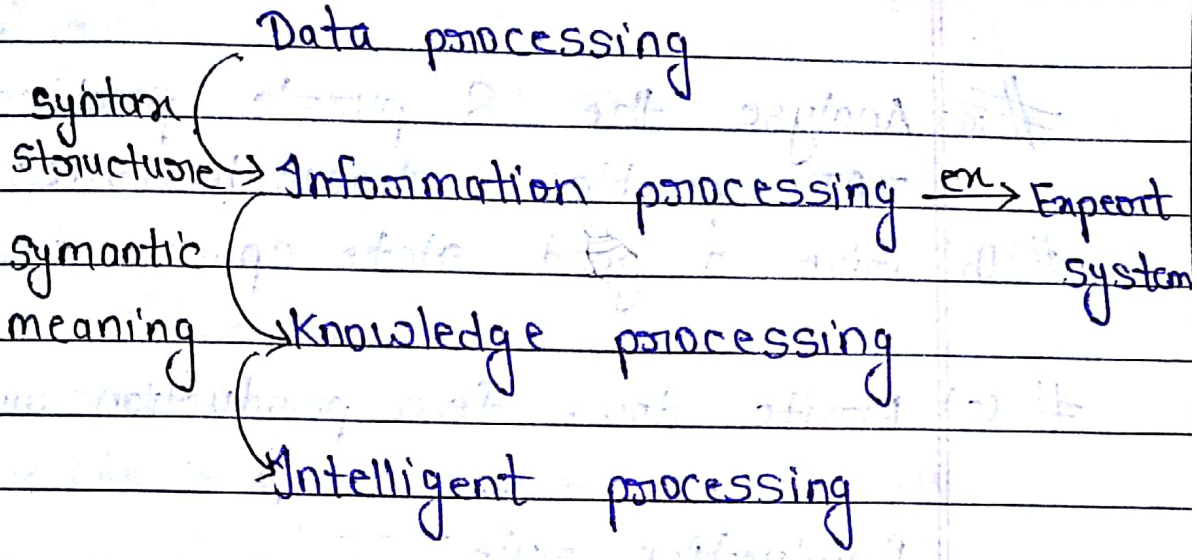
dish <

pasta salad <

with the fock <

6.) What is the role of knowledge?

Knowledge required for playing chess game by the computer.



* ~~Knowledge~~ The rules that determine the legal moves.
* Control strategy mechanism.
* Good strategy and tactics to constraint the search (to specify the execution).

* Knowledge required to reading / understanding the news-paper column.

7.) Does the task require interaction with a person?

Solidary - in which computer is given a problem description and produce an answer with no intermediate communication and no demand for an explanation of the reasoning process.

conversational in which there is intermediate communication bet" a person & the computer either to provide additional

assistance to the computer or to provide additional information to the user or both.

#(a) Analyse the 8 puzzle problem w.r.t AI problem characteristics (7 marks)

#(b) Give a good state space representation? (3 marks).

#(c) Write down few production rule? (4 marks)

Production rule :—

(c) 1.) Non decomposable (subproblem is not independent).

2.) Recoverable.

3.) Problem universe is predictable.

4.) Absolute.

5.) path

6.) Not much

7.) Solidary

17/08/18

Towers of Hanoi (a good state space representation) :—

1.) Non decomposable (subproblems are not independent)

2.) Recoverable (solution steps can be undone)

3.) Problem universe is predictable.
4.) Good solution is absolute.
5.) Solution is path to a state.
6.) Only limited domain knowledge is required.
7.) Solidary

chess game :—
1.) Non decomposable
2.) Irrecoverable
3.) Predictable or non predictable both

22/08/16

Three missionaries and three cannibals are on one side of the river with a boat. They are all want to get to other side of the river. The boat can carry only one or two people at a time. At no time should there be more cannibals than missionaries on either side of the river as this would probably result in the missionaries being eaten.

(i) Give a good state space representation.
(ii) What would be the initial state and final state (goal state) as per above/ your representation.

(iii) Write down all possible production rules.

(iv) General problem search tree [without any cycle]

Initial state: (3,3,1) (0,0,0) ← complement of each other.

Goal state : (0,0,0) (3,3,1) ✓ can be considered as good state representation.

(mmm, ccc) ✗ not a good representation

There is not distinction any either cannibal or missionaries

(3,3,0) preclude info. regarding the boat position

(3,3,1) still not a good representation.

General rules :—

① 1C →

② 2C →

③ 1M →

④ 2M →

⑤ 1M, 1C →

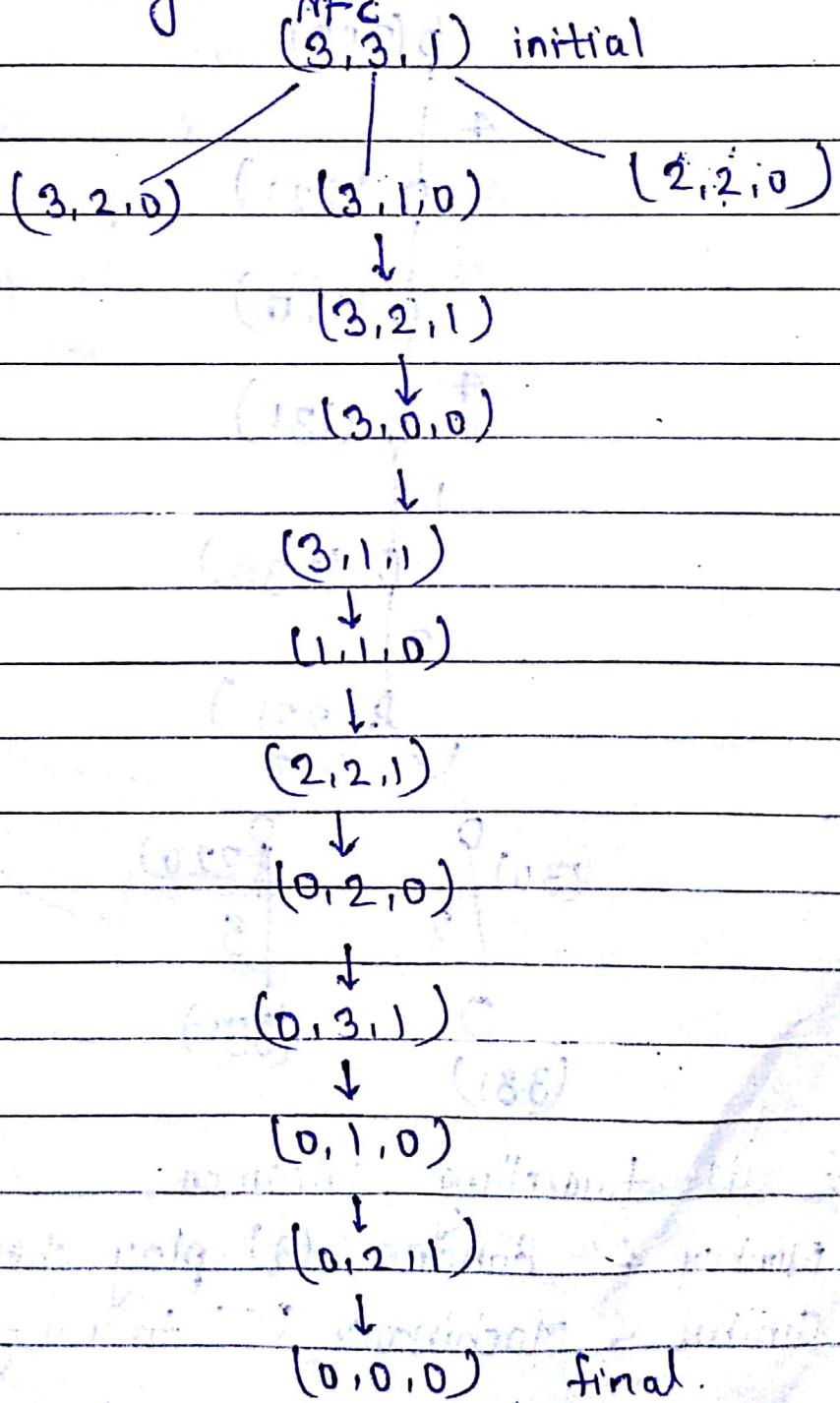
Problem search tree:

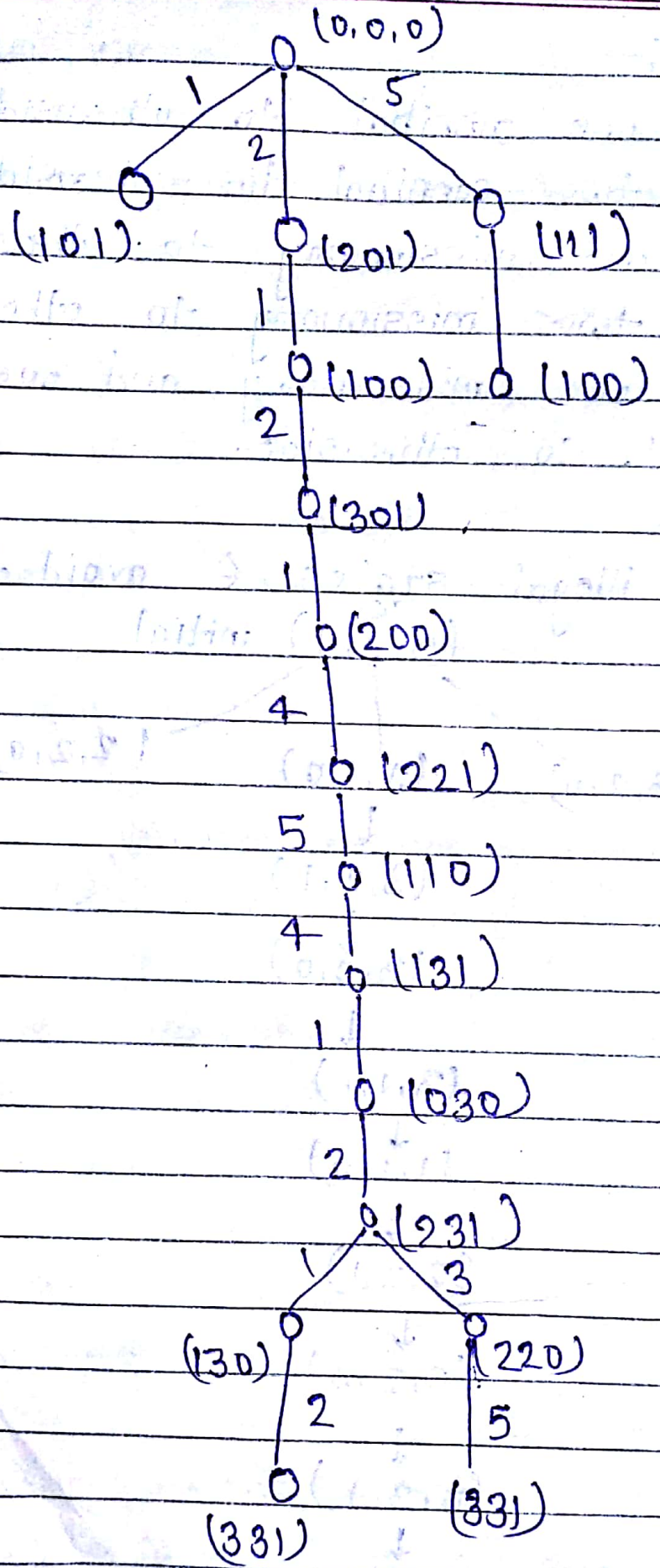
starting side (3,3,1) (0,0,0) destination side

Rules :—

(i) Move one cannibal to otherside
(ii) Move two cannibal to otherside
(iii) Move one missionary to otherside
(iv) Move two missionary to otherside
(v) Move one missionary and one ~~mission~~ cannibal to otherside.

state illegal states & avoid them





Assignment :- (1) travelling salesman
(2) Monkey & Banana (3) play chess
(4) Canibal & Machinary (5) tower of Honai

Monkey and Banana problem :-

Hungry monkey find the banana hanging from ceiling. also finds a stick & a chair and with the help these two monkey will be able to strike the banana.

((mx, my, mz), (Sx, Sy, Sz), (Bx, By, Bz), (Cx, Cy, Cz))

* Walk into the room.
* Move the chair.
* Hungry status.
* climb up/down the chair.
* swing the stick.
* Eat Banana.

What is a search process?

Every search process can be viewed as a traversal through a directed graph in which each node represents a problem state & each arc represents a relationship between the states it connects.

What is the objective of search process?

To find a path through directed graph starting at an initial state and ending in one or more goal states.

Five important issues that arise out in all search techniques method :—

1.) The direction in which to conduct the search.
2.) Topology of search process.
3.) How each node of the process will be represented.
4.) Selecting the applicable rule.
5.) Using heuristic function to guide the search.

Issue

Forward versus Backward reasoning :-

The objective of a search process is to discover a path through problem space starting from the initial configuration to a goal state. There are two direction in which search process could proceed.

(i) Forward from the start state.
(ii) Backward from the goal state.

We have two choice to identify the factors that influence the direction of the search.

How to choose particular one?

• Depending upon the topology of the problem space, it may be significantly more efficient perform search in one direction rather than that other.

• There are three factors:

(a) Are more possible start states or goal states.

It is desirable to move from smaller set of states to a larger set of states.

(b) In which direction the branching factor is higher?
It is better to proceed in the direction of lower branching factor.

(c) Will the program be asked to justify its reasoning process to the user?
It is important to proceed in the direction that corresponds more closely with the user will think.

Bidirectional search ~~strategy~~ strategy: simultaneously search forward from initial state and backward from goal state until path meet somewhere in between.

The bidirectional search may ineffective when the two searches bypass each other resulting in more work that it would have taken for one of them.

on double the work

II Topology of the search process: tree, graph

In AI search process can be viewed

as the application of production rules.

Problem tree vs problem graph :—
A simple way to implement any search strategy is to perform as a tree traversal.
Each node of the tree is expanded by the production rules to generate a set of successor node and each of which in turn be expanded, continuing until a node that represent a solution is found.
A tree search procedure can be converted to graph search procedure by modifying the action performed at each time a node is generated.

(i) Examine the set of nodes that have been created so for to see if the new node is already exist.

(ii) If it does not simply add it to the graph just as for a tree.

(iii) If it does already exist then do the following two things :

(a) Set the node that is being expanded

to point to the already existing node corresponds to its successor. rather than to the new one. the new one can simply to thrown away.

06/09/16

The monkey and banana problem:

A hungry monkey finds himself in a room in which a bunch of bananas is hanging from the ceiling, The monkey unfortunately, cannot reach the bananas. However, in the room there are also chair & a stick. The ceiling is just right height so that the monkey standing on a chair could knock the bananas down with the stick. The monkey knows how to move around, carry other things around to reach for banana and wave the stick in air. What is the best sequence of action the monkey to take to acquire lunch?

1.) chess
2) Water jug
3.) 8-puzzle

4.) Travelling salesman
5.) missionaries & cannibals
6) Tower of Honai (3 discs)
7.) Monkey & banana.

⇒ Analyse w.r.t. 7 problem characteristics.
⇒ Good state space representation.
⇒ Possible initial and goal state.
⇒ Production rules
⇒ Generate problem tree upto a possible level.

issue 2

Problem Tree → Problem graph

(b) If you keep track of the best path to each node then:

* check to see if the new path is better or worse than old one.
* If it is ~~no~~path is worse than donothing.
If it is new path is better than set new path as the current path.
~~to use~~ to get the node and propagate ~~the worst path~~ Corresponding changes down through successors as necessary.

A cycle is a path through the path graph in which a given node appears more than once/ones.

issue-3

How each node in the search process will be represented?

(i) How can individual objects and facts be represented.

(ii) How can you combine the representation of individual objects into a representation of complete problem state.

(iii) How can the sequence of problem states that arise in a search process be represented efficiently?

issue-4
7/09/16

selecting the applicable rules:
A clever search involves choosing from the set of applicable rules that can be applied at particular point, the one that is most likely leads to a solution.
Extracting the set of applicable rules for a current position require some kind of matching betn the current position and the pre-condition of the rules. (This way of selecting rules may not be always good idea for two reasons.)

(a) Scanning all the rules at every step of the search would be hopefully inefficient.

(b) It will not always obvious whether or not a rules preconditions ~~one~~ are satisfied by the current state.

Remedy: Use the current state as the index for selecting the set of applicable rules.

issue-5: Using a heuristic function to the search: A heuristic is defined as a technique that aids in discovering solution to problem eventhough there is no guarentee that it will never lead in the wrong direction.

Depth first search :—

It will pursue a single branch of a search tree until if it yields a solution. or until some pre-defined depth has been reached only then go back and explore other branches.

The alternative at same level are ignored completely as long as there is a hope of reaching the goal state using original choice pursuing the single branch.

12/09/16

Depth first search Algorithm :—

(1.) Form a one element list consisting of the root node (the node at the top of the tree and with no parent

(2.) Until the list is empty or the goal node has been reached, repeat the followings.

2(a) Determine if the first node on the list is a goal node or not.

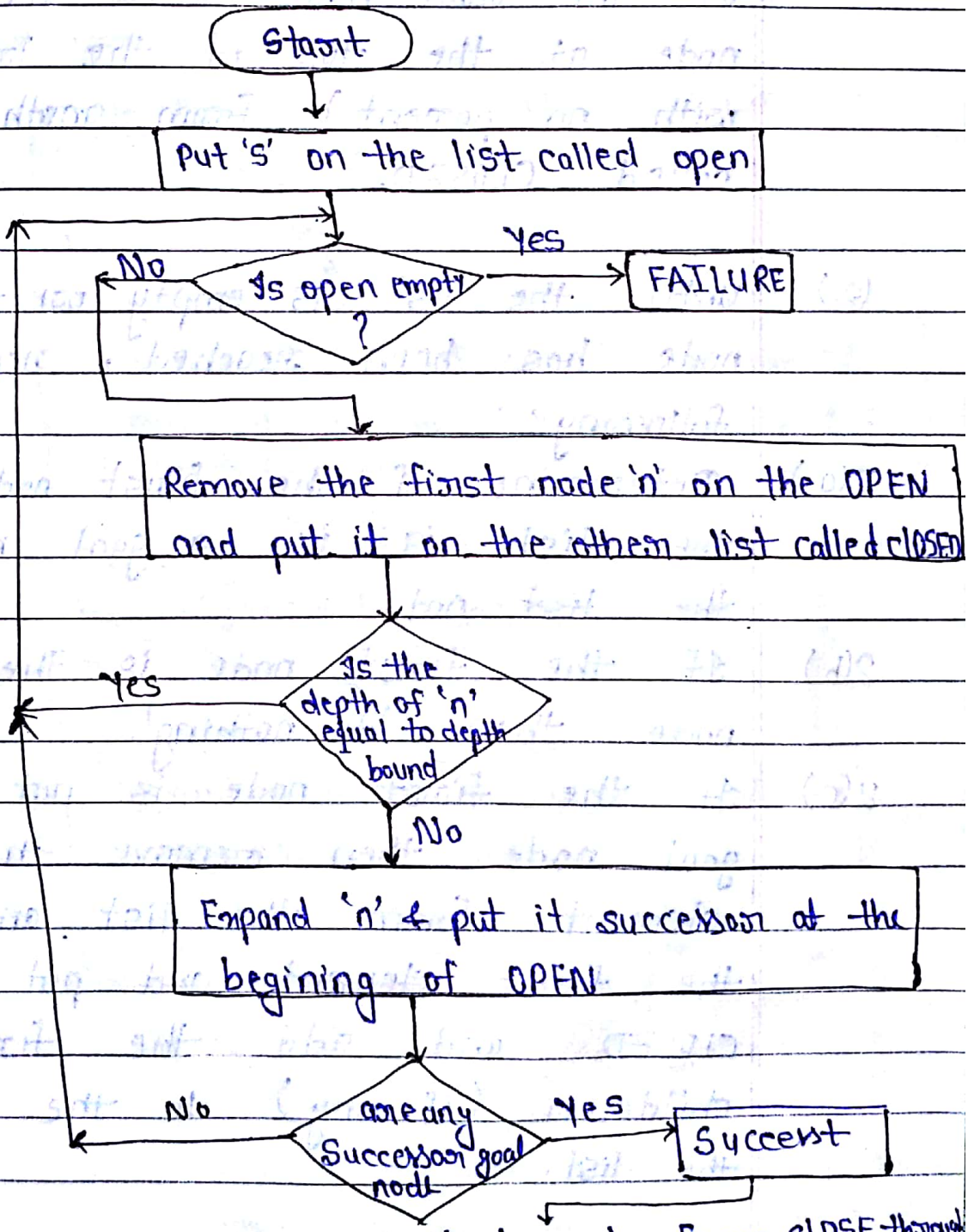
2(b) If the first node is the goal, node then 'do nothing'

2(c) If the first node is not the goal node, then remove the first element from the list and add the first element, children (if any) to the front

of the list. |*

(3) If the goal node has been found "annaunce sucess" else " announce failure".

Flow chart (Two dimensional representation of an algorithm) for Depth first search :-



Date :

OPEN } are single element list
CLOSED }

S → start (initial) node

n → current node

Breadth first Search Algorithm :—

(1.) Form a one element list consisting of the root node called OPEN (the node at the top of the tree and with no parent). Form another list called CLOSED.

(2.) Until the list is empty or the goal node has been reached, repeat the following:

2(a) Determine if the first node on the list is ~~is~~ a goal node or ~~the list~~ not.

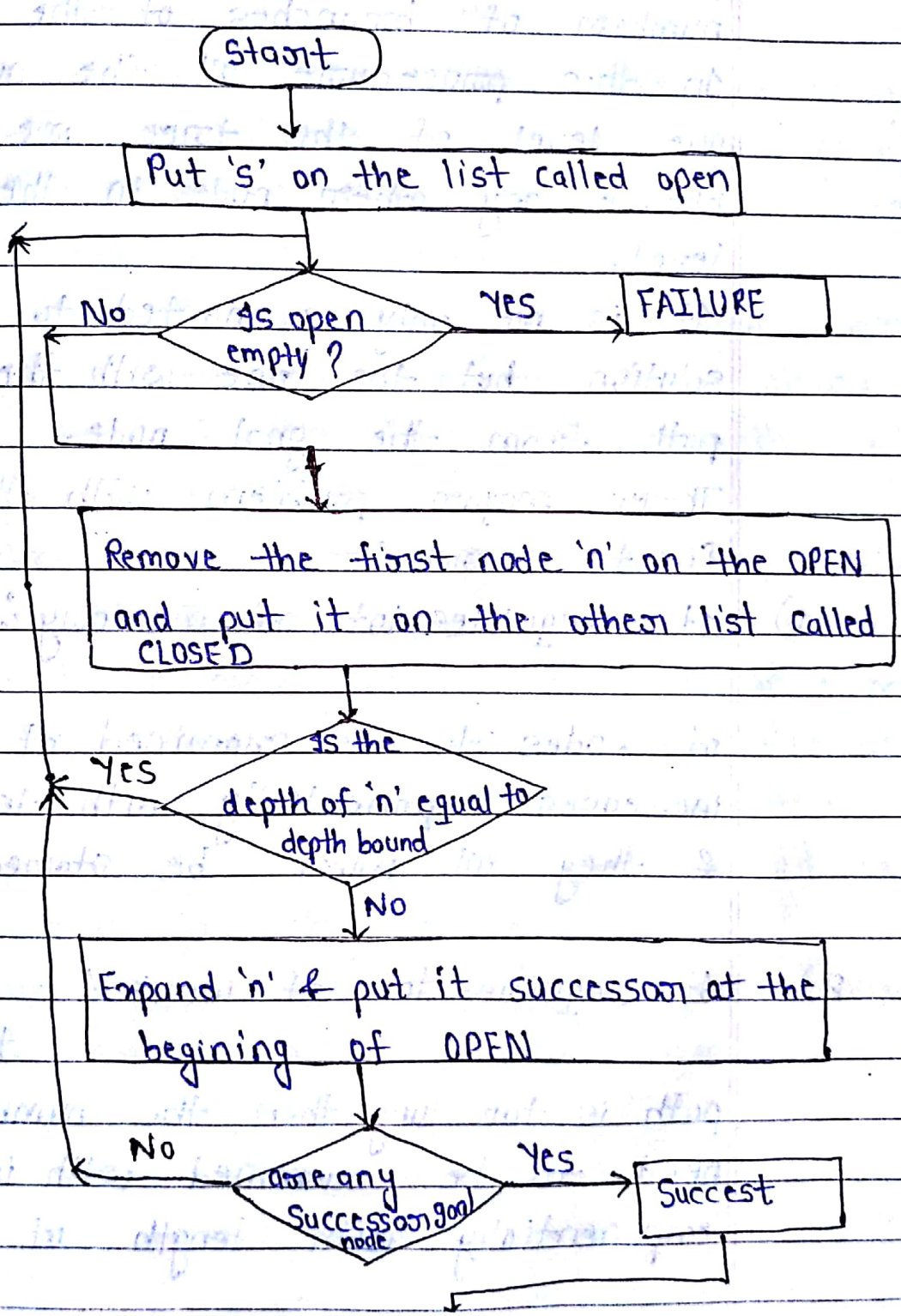
2(b) If the first node is the goal node then 'do nothing'.

2(c) If the first node is not the goal node then remove the first element from the list and ~~add the first element~~ and put it on CLOSED and add the first element children (if any) to the End of the list.

Date :

(3.) If the goal node has been found 'announce success' else 'announce failure'.

Flow chart for Breadth first search :-



Breadth first search :—

A breadth first search procedure is guaranted to find a solution (if exist) provided that there are a finite number of branches of the tree.
In this procedure all the nodes on one level of the tree are examined before any other node in the next level.
It is not only guaranted to find a solution but the one with the shortest path from the goal node.
Three major problem with the breadth first search :—

1.) It requires lot of memory :—

The number of nodes to be examined at each level increases exponentially with level number & they all must be stored.

2.) It require lot of work :—

If the shortest path is too long then the number of nodes need to be examined with increased exponentially with length of the path.

3.) Irrelevant or redundant operators will

greatly increase the number of nodes that must be explored.

Hill climbing Algorithm :-
It is the variation of DFS.

1.) Form a one element list initially consisting of the root node called OPEN (the node at the top of the tree / with no parent) Form another list called CLOSED.

2.) Until the list is empty or the goal nodes has been reached, repeat the followings:

a.) Determine if the first node on the list is goal node or not.

b.) If the first node is goal node, do nothing. go to step 3.

c.) If not, remove the first element from the list and sort. the first element children (if any) with respect to estimated remaining distance to goal node & add to the front of the list.

3.) If the goal node has been found, "announce success" else "announce failure".

Hill climbing Search :-
Sort the first element (if any) with respect estimated remaining distance and add then to from of the list.

Hill climbing search technique is not a good idea for three kinds of problem.

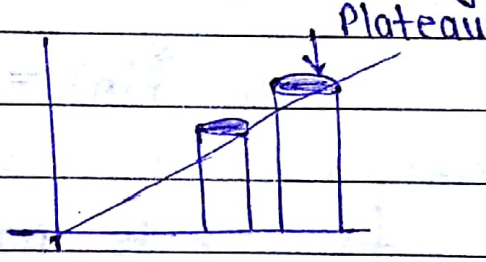
Foot hill problem :-

It occurs whenever secondary peaks occur. These secondary peaks draw the hill climbing search procedure like magnet. there an optimal point is found but it is local not global and the user is left with false with sense of accomplishment.
A local maxima is a state that is better than all it's neighbour. but it is not better than some other state further away.
local maxima are particularly

frustating because they often occur almost within the sight of a solution and in this case they are called foot hills.

Plateau :-

A plateau problem is flat area of the search space in which a whole set of neighbour states have the same value. On plateau, it is not possible to determine the best direction in which to move by making local comparison



ridge problem :

A ridge is an area of the search space that is higher than surrounding area, but it can not be traversed by single move in any one direction.

Best first search :-

Add the first element children (if any) to the list and sort the entire list. elements with respect to the remaining distance to the goal node.

Hill climbing :- choose the best child Successor

A* Algorithm :

Evaluation function :-

The best way to use heuristic information is to use some criteria to reorder to all the nodes in the list at every step. In order to apply such an ordering procedure, we need some measure by which to evaluate the promise of a node. such measures are called evaluation function.

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$$

where

$\hat{f}(n) \rightarrow$ is an estimate of the cost of minimal cost path constrained to go through 'n' [. The value of evaluation function at any node 'n'] from the starting node to goal node

$\hat{h}(n) \rightarrow$ is an estimate of the cost of minimal cost path from node 'n' to goal node.

$\hat{g}(n) \rightarrow$ is the cost of the path from the start node 's' to the node 'n'.

Step 1: Put the start node 's' on the list called 'OPEN' and closed set 'CLOSED' to an empty list.

Step 2: If 'OPEN' is empty exit with Failure
else
Continue

Step 3: Pick up a node from 'OPEN' with lowest 'f' value and call it as BEST_NODE. Remove the BEST_NODE from the 'OPEN' and put it into 'CLOSED'.

*

Step 5: Expand the 'BEST_NODE' by generating all of its successors and for each successor repeat the following:

(a.) Set a pointer from each successor to the 'BEST_NODE' [This is useful for recovering the path. Once the solution is (good node is discovered) found].

(b) Compute g(successor) = g(BEST_NODE) + cost of getting from BEST_NODE to successor.

(c.) Check if the successor is same as any node on / 'OPEN'
'CLOSED' - contains expanded node
'OPEN' - contains unexpanded node

if so call that node as 'OLD' and add it to list of BEST_NODE Successors check whether it is cheaper to get OLD via its current parent or via the BEST_NODE by comparing the values of g(OLD) and g(successor).
if 'OLD' is cheaper then do nothing.
else (if the successor is cheaper)
Reset OLD's parent link to point to the BEST_NODE and record the newly found cheaper path g(OLD) plus as well as update $\hat{f}$(OLD).

(d.) check the successor was not on OPEN, check if it on 'CLOSED', if so call this node as OLD and add it to the list of BEST_NODE's successor.
check if the new path or old path is better and according reset the parent list & update g value.

If we have found better path to 'OLD' then this newly discovered information must be propegated to OLD's successors ↓ perform DFS starting from OLD and change each node's g value when it is visited &
Terminate the DFS where / either node

with no successors (leaf node) or a node in which leaf an equivalent or better path has already been found. / has reached

(e) if successor was neither on OPEN or on CLOSED then put it on OPEN and direct the pointers from BEST_NODE to it also compute.

$\hat{f}$(successor) = g(successor) + $\hat{h}$(success

Step 4: If the BEST NODE is a goal node then exist with Success & the Solution path is obtained by tracing back through the pointers else continues.

3/10/16

Assignment-II

2	8	3
1	6	4
7		5

start state

1	2	3
8		4
7	6	5

Goal state

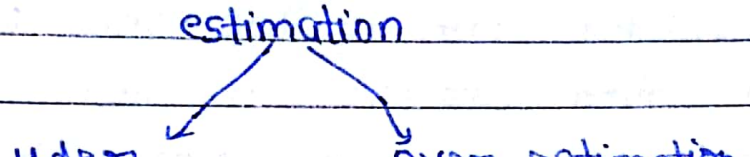
Generate problem tree using DFS & BFS

2	8	3
1		4
7	6	5

2	8	3
1	6	4
7	5	

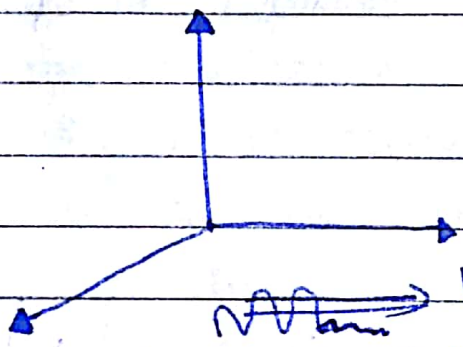
2	8	3
1	6	4
	7	5

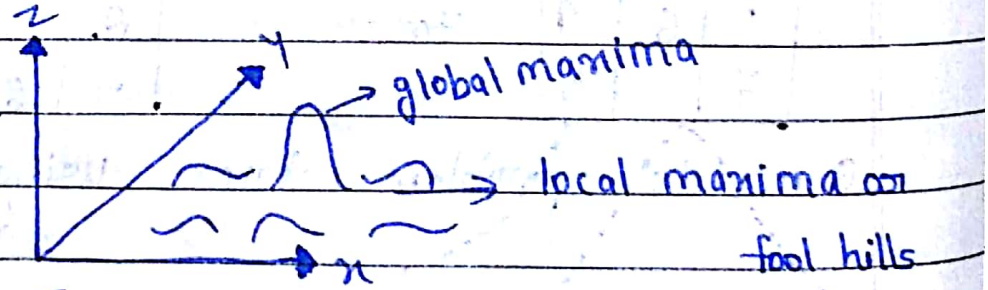
Weak AI / Strong AI
Weak method / Strong method
Hard computing / Soft computing

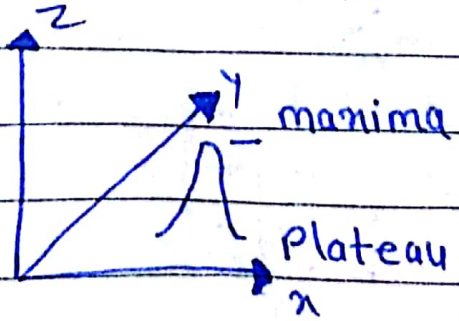


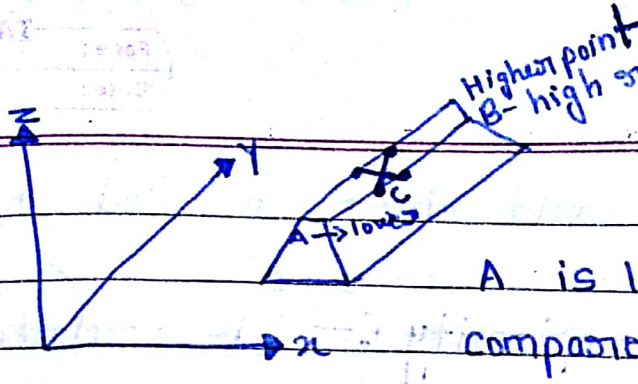
4/10/16

3-D representation of problem that fools hill climbing :-









A is lower position when compared and which is global maxima.

Properties of search method :-

Complexity → Time complexity $O(b^d)$ (d: depth, b: branching factor)

Complexity → Space complexity

Time complexity :- The time required to find / search the goal node.

Space complexity :- Memory that search method need to use.

5/10/16

Properties of search method :-

Complexity

Completeness

Optimality

Admissibility

Tentative vs Irrevocability

complexity :- It is useful to describe how efficient that a search method is, over time & space

Time complexity :- of a search method is related to the length of time that the

Date :

method could take to find the goal state.

Space Complexity :- is related to the ammount of memory that the method needs to use.

Breadth first search:

Time complexity $O(b^d)$
↓ Big O

b - Branching factor of the problem tree

d - depth of the goal node in the tree.

Depth First search: Efficient in space
Not Efficient in time

Completeness :- A Search method is described being complete if it is guaranteed to find goal state if one exists.

Breadth first search is complete

Depth first search is not complete because it may explore a path of infinite length and never find the goal node that exists on another path.

Optimality :- A search method is optimal if it is guaranteed to find the best solution that exist.
it will find the path to a goal node that involves taking the least number of steps.
BFS is optimal & DFS is not optimal

Admissibility :- A search method is admissible if it is guaranteed to find the best solution in quikest possible time.

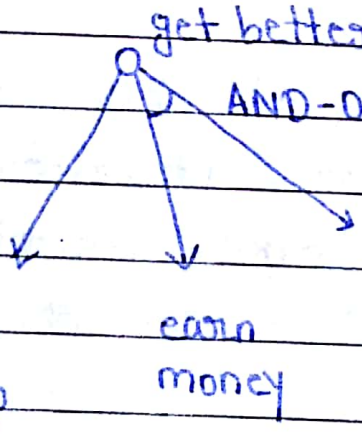
Tentative Vs Irrevocability :- Search method that use back tracking are described as tentative.
Search method that do not use back tracking & therefore examine just one path are described as irrevocability

Problem Reduction :-

So far we have considered several search strategies for OR graph (Represent the problem tree that problem is not decomposible)

AND-OR graph structure is very usefull representing the soln of the problem that can be solved by decomposing them into subproblems all of which must be solved. This decomposition or reduction generate area that we call them as AND arc.

When we need a mobile or laptop.



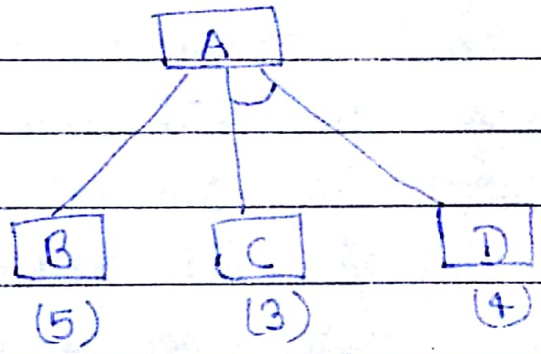
One AND arc may point to any number of successors nodes all of which must be solved inorder for the arc to the point to the solution.

Just as in OR graph several arc may emerge from a single node indicating variety of ways that the problem can be solved.

A* algo

The best - fit is not adequate for searching AND-OR graph

Main step: everytime choose a best node (the lowest f value) for expansion



Assumptions:

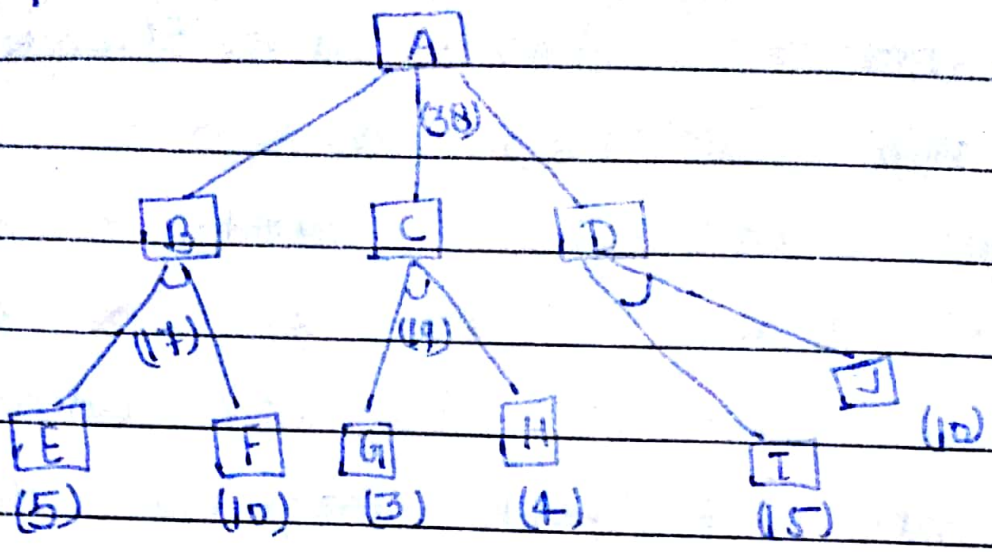
? The numbers at each node represents f value here we consider only h(n) and ignore g(n).

? Every operation has a uniform cost for simplicity.

Say: Each arc with single success has cost of '1'

Each AND arch with multiple successor has a cost of '1' each of its

components.



Heare the problem is that the choice of which node to expand next must depend not only on the f value of that node, but also whether that is part of the current best path from the initial node.

Before describing an algorithm for searching an AND-OR graph we need to exploit a value that we call it futility (a threshold value for 'f') Any solution with a cost above it is to expressive to be practical.

Problem Reduction Algorithm :-

Step-1 Initialize the graph with starting (initial) node.

Step-2 Repeat the following steps until the starting node is labelled as solved or until its cost goes above FUTILITY.

2(a) Traverse the graph starting at the initial node and following the current best path and accumulate the set of nodes that are on that path and have not yet been expanded or labeled as 'SOLVED'.

2(b) Pickup one of these un-expanded node and expand it. If there are no successors then assign FUTILITY as the value of the node ~~otherwise~~ else. add its successor to the graph. calculate the f value of each of the successor.

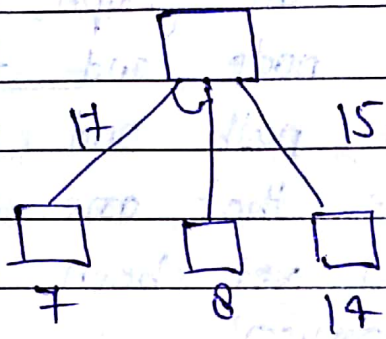
If the f value of any node is 0 (zero) then label that node as 'SOLVED'

2(c) The changes in the f' estimat of the nearly expanded node to reflect new information provided by its successor propagate this change backward through graph If any p* node contains a successor

A* algo :- OR graph AND-OR graph x

TAYALS
Page :
Date :

are whose descendent are all solved then label the node itself as 'SOLVED'. At each node that is visited while grouping the graph decide which of its successor are is the most promising and mark it as the part of the current best path.



most promising is the lowest value

This may cause current best path to change.

AO* Algorithm :-

- It's only applicable for AND-OR graph OR graph.

- AO* will use a single structure 'GRAPH'.
- Each node in the graph will point both down to its immediate successor and upto its immediate processor.

Each node in the graph will also

associated with it an h 'value' and estimated of the cost of the path from itself to a set of solution nodes.

24/10/16

AO* Algorithm :-

step 1 :- Let the GRAPH consist of the node representing the initial state and call this node as 'INIT'. Compute h'(INIT)

Step 2 :- Until INIT is labeled as SOLVED or until INIT's h value becomes greater than FUTILITY repeat the following procedure.

step 2(a) :- Trace the labelled arc from INIT and select for expansion one of the not yet expanded nodes that occur on this path call selected node as 'NODE'.

step 2(b) :- Generate SUCCESSORS of 'NODE' If there are none then assign 'FUTILITY' as the h value of 'NODE' (This is equivalent to say that NODE is not solvable).

If there are successors then for each successor that is not an ancestor of NODE do the following.

(i) Add the successor to GRAPH.

(ii) If the successor is a terminal node, then label it as 'SOLVED' and assign it h value to zero (0)

(iii) If the succession is not a terminal node then compute the h value of the successor.

Step 3 :- Propagate the newly discovered information up in the graph by doing the following:

Let 'S' be a set of nodes that have been labeled as 'SOLVED' or whose h value have been changed. So there is a need to propagate these values back to their parents.

Initialize 'S' to 'NODE'.

until 'S' is empty repeat the following:

3(a) If possible select from 'S', a node none of whose descendant in GRAPH occurs in 'S'. If there is no such node then select any node. call this node as 'CURRENT' and

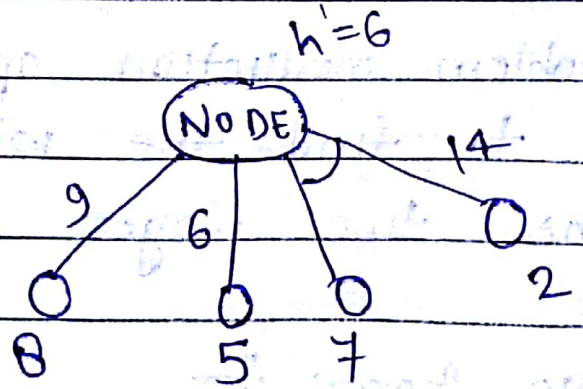
remove it from 'S'.

3(b) : Compute the cost of each arc, emerging from CURRENT Assign CURRENT's new h value as the minimum cost arc emerging from it.

3(c) Mark the best path out of CURRENT by marking the arc that had minimum cost computed.

3(d) Mark 'CURRENT' as SOLVED if all the node connected to it through newly labeled as SOLVED.

3(e) If ('CURRENT' has been SOLVED) OR (the cost of CURRENT was just changed) then it new status must be propagated up in the graph and so add all the ancestors of 'CURRENT' to 'S'.



Game Playing :—

Game playing is a good domain to explore machine intelligence for two reasons.

1.) They provide structured task in which it is easy to measure success or failure.

2.) They did not require large ammount of knowledge. (applicable only for simple games)

Here we are interested only in two players games and the games players know what they have done & what they can do.

Good | Aim — one of the two player wins or at least draw the game.

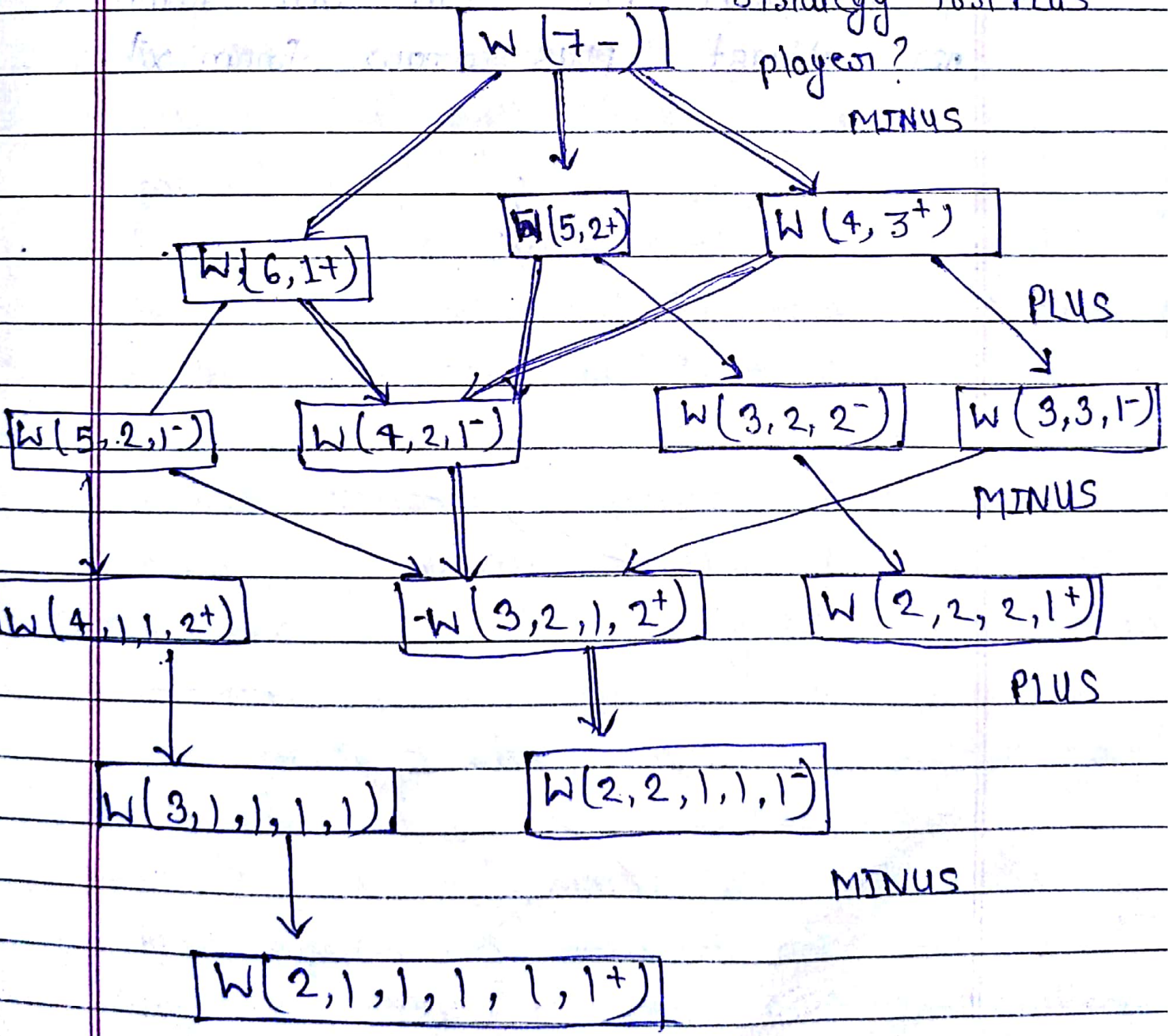
The problem reduction approach can be applied to find the winning strategy for the two player games.

Grundy's Game :—

Two players have infront of them a single pile of object say a stack

of pennies. The first player divide the original stack into two stack that must by unequal. Each player alternatively does the same to same single stack until every stack has either just one penny or two. The player who first can not play is loser.

find the winning strategy for PLUS player?



Date :

Here we name the two players 'PLUS' & 'MINUS'. We have to find a playing strategy for PLUS player starting with some configuration x^t

The superscript 't' stands for either '+' or '-'

'+' indicates it is PLUS turn to moves next '-' indicates it is minus turn to move next would like to prove that PLUS can win from x^t or atleast PLUS draw from x^t.

Date :

26/10/16

Game playing :-

Two things can be done to improve the effectiveness of the search based problem solving programming.

(1.) Improve the generate procedure so that only good moves are generated

(2.) Improve the test procedure so that best moves (path) will be recognised and explored.

Mini Max procedure

- It is a depth first & depth limited procedure.
- look ahead procedure.

Every recursive call the procedure should move towards the base case.

Suppose situation analyser that convert overall judgement about a situation into a single overall quality Number (it can be +ve or -ve).

+ve number by convention favour on player.

-ve number favours the other player.

The degree of favour goes with absolute value of the quality number.

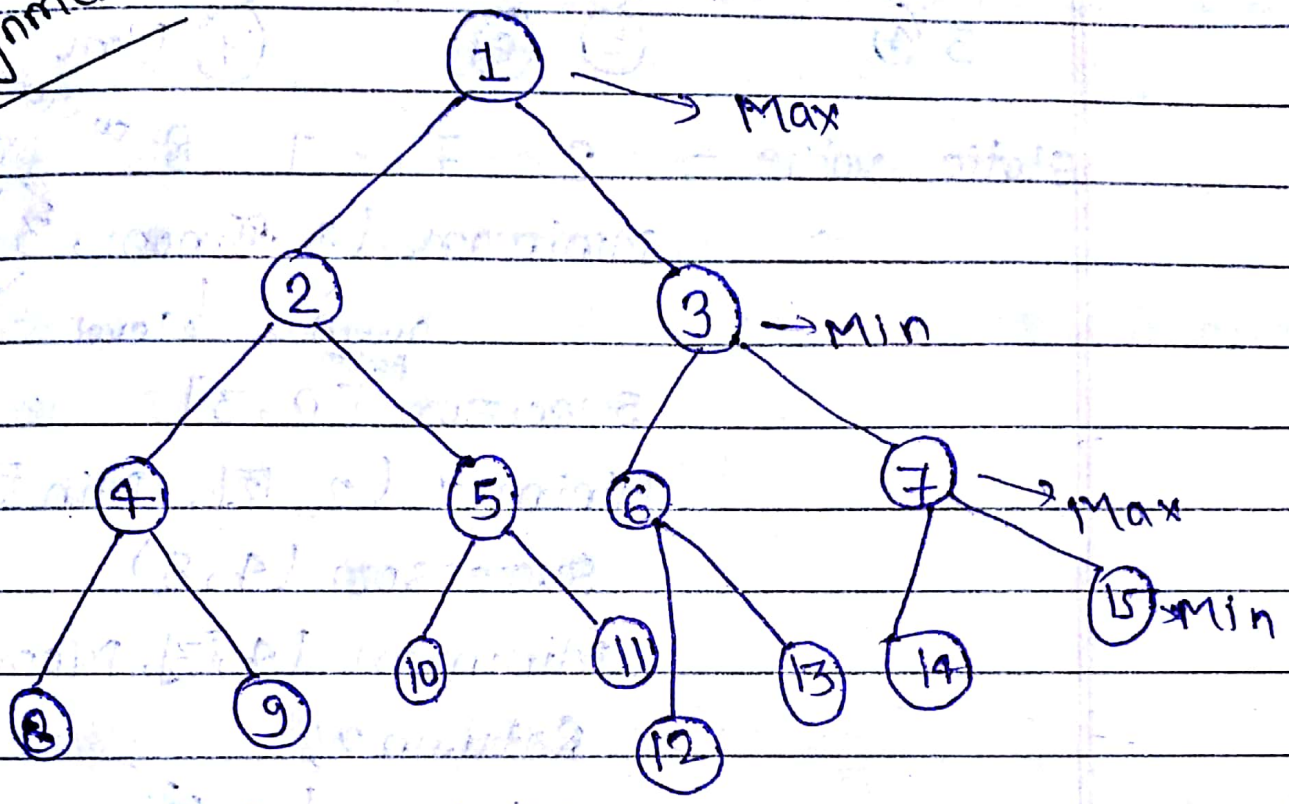
Date :

At the end of limited exploration of possible moves, one find the static evaluation score produced by the situation analyser.
The player hoping to +ve number is called maximizing player.
The player hoping for -ve number is called minimizing player.

Miniman procedure :—
Step-1 : Determine if limit of search has been reached.
or
If the level is a minimizing level
or
if the level is a maximizing level.
1(a) : If the limit of search has been reached, compute the static score of the current position relative to the appropriate player & Report the result.
1(b) : If the level is the minimizing level then use miniman on the children of the current node and report the minimum of the result.

1(c): If the level is the maximizing level then use minimax on the children of the current node & Report the maximum of result.

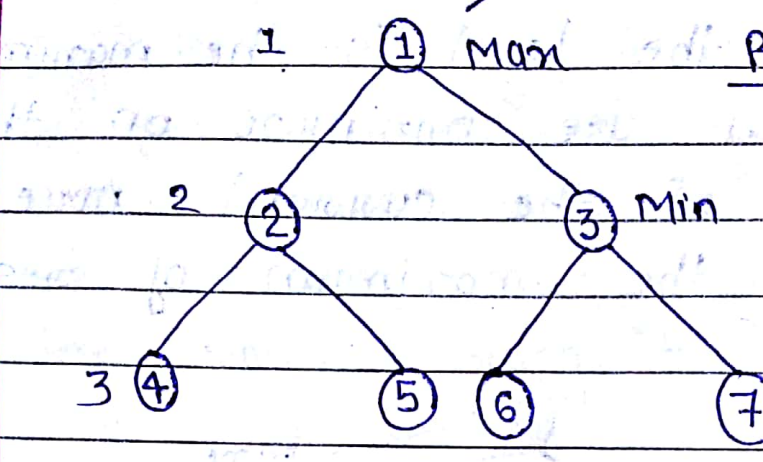
Assignment-III



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

27/10/16

Particular



Pruning

Static value - 2 7 1 8

current level

level of the terminal node

Minimax (A, 1, max, 3)

Current position — level — limit

Succesor [2,3]

Minimax (2, 2, Min, 3)

Succesor (4,5)

Minimax (4, 3, Max, 3)

Return 2;

Minimax (5, 3, max 3)

Return 7;

Return 2;

Minimax (3, 2, Min, 3)

Minimax (3, 2, Min, 3) ×

Succesor (6,7)

Minimax (6, 3, Max, 3)

Return 1;

Minimax (7, 3, Max, 3)

Return 8

Return 1;

Return 2;

If the position is a final one [terminal node] then compute static value of the current position and return the value.

If min is on move then 2.1 generate the successor of the current position.

2.2 Apply Miniman to each of these successor with max to move.

2.3 Return minimum of the result obtained from 2.2.

3/11/16

Miniman with α-β pruning

Step-1 Determine if the level is the top level or
if the limit of search has been reached or
if the level is the minimizing level. or
if the level is maximizing level.

1(a) If the level is the top level then assign the value $-\infty$ to α and $+\infty$ to β

1(b) If the limit of search has been reached then compute the static value to the appropriate player &

Date :

report the result.

1(c.) If the level is the minimizing level:

(i) until all children are examined with minimax or α is greater than β repeat the following:

(i)(*) Set β to the smallest of given β value and the smallest value so far reporting by minimax on working on the children.

(i)(**) Use minimax on the next child of the current position handling this new application minimax the current α-β value.

(i)(***) Report the β.

1(d) If the level is the maximizing level:

1(d).1 Until all children are examined with minimax or α is greater than β repeat the following:-

(1d1.1) Set α to the largest of the given α value and the largest value so far reported by minimax on working on the children.

(1d1.2) Use minimax on the next child of the current position handling this new application minimax the current α-β value.

Date :

Id2 Report the α :

(1) If the current level = Top level then
$\alpha = -\infty$
$\beta = +\infty$

(2) If the current level = limit of search then Calculate ...

(3) If min is on move then :-

(a) generate a list of successors of the current position.

(b) if $\alpha > \beta$ or Successor list is empty then terminate
Return β

(c) $\beta_c = \alpha - \beta$ (successor[1])

(d) β = minimum (β_c, β)

(e) delete first element of the successor and go back to step (3b)

(4) If max is on move then-

(a) Generate a list of successors of the current position.

(b) if $\alpha > \beta$ or successor list is empty the terminate
Return α

(c) α_c = miniman with $\alpha - \beta$ (successor[1])

(d) α = Maximum (α_c, α)

(e) delete first element of the successor and go back to step (4b).

alphabeta [1, max, 1, 3, α, β)

$\alpha = -\infty$

$\beta = +\infty$

succ [2,3]

α_c = Alphabeta (2, min, 2, 3, $-\infty$, $+\infty$)

succ [4, 5]

β_c = Alphabeta (4, max, 3, --)

= 2

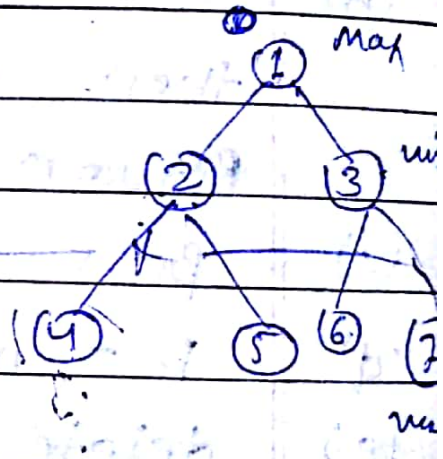
β = Min (2, $+\infty$)

= 2

β_c = Alphabeta (5, max, 3, 3)

β = min (7, 2)

= 2



X ——— X

AlphaBeta (1, max, $-\infty$, $+\infty$, 3)

succ (2, 3)

Apply AlphaBeta ber 2

" " " " 3

α_c = AlphaBeta (2, min, $-\infty$, $+\infty$, 3)

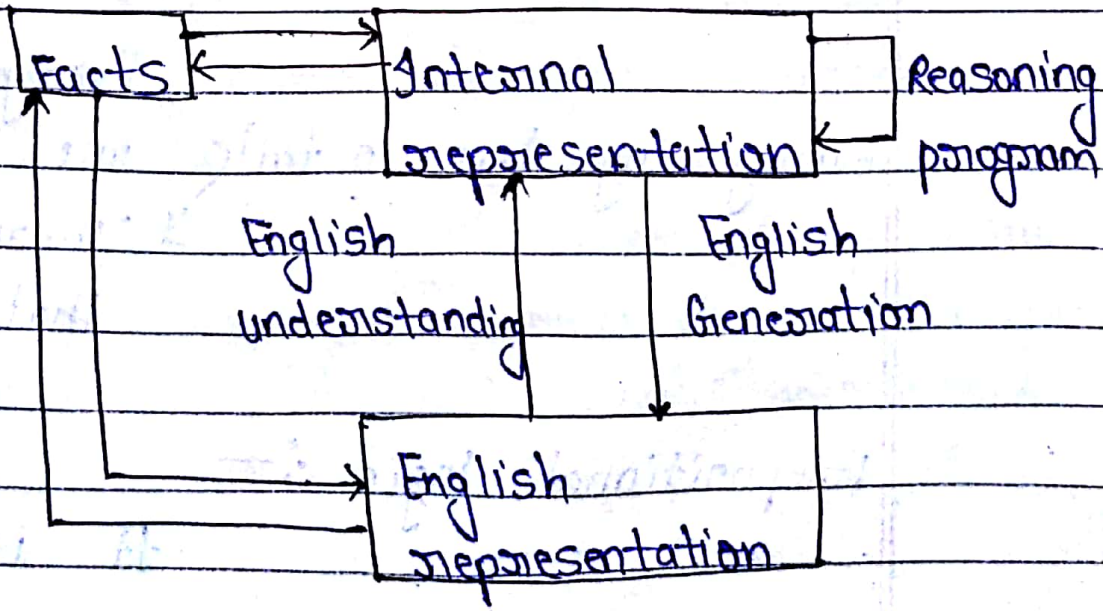
succ (4, 5)

β_c = AlphaBeta (4, $-\infty$, $+\infty$, 3)

Knowledge representation :-

So far we have seen the general methods of manipulating the knowledge using search.
A variety of ways of representing knowledge have been exploited by AI problems.
Two different entities that are pertain to all type knowledge representation.
Fact: Truth in some relevant world
[These are the things we want to represent]

Representation formalism: Representation of facts in some choosen formalism [These are the things we are actually able to manipulate].
Each representation is associated with some mapping functions.



1.) Mapping function from facts to some chosen representation

2.) Mapping function from chosen representation to fact.

Mathematical logic to represent the knowledge :

Spot is a dog. → dog(spot) → Mapping from english representation to predicate logic

All dogs have tails. → $\forall x \; dog(x) \longrightarrow has\ tail(x)$ → mapping from predicate logic to english sentence

Soln :— $dog(spot) \longrightarrow has\ tail(spot)$

The available mapping function is not always one to one.

All dog have tails
- every dog has atleast one tail.
- Each dog has several tails.

Every dog has a tail
- Every dog has atleast one tail.
- There is a tail that every dog has.

Propositional logic :—

It is appealing to

represent the knowledge for two reasons.

1) It is simple to deal with
2.) There exist a decision procedure for it.

English	Propositional logic
It is raining	RAINNY
It is sunny	SUNNY
It is windly	WINDY

⇒ If it is rainny then it is not sunny.

RAINNY → ¬ SUNNY

negation

Limitation propositional logic :—

(1.) Socrates was a man

SOCRATES MAN

Plato was a man.

PLATO MAN

These two are totally seperate assertion and it is not possible to draw any conclusion about the similarities between socrates & Plato.

Date :

Limitation 2: Very difficult to represent certain facts using proposition logic.

All men are mortal.

9/11/16

Predicate logic to represent the logic :-

① Marcus was a man.

man (Marcus)

② Marcus was a pompeian

pompeian (Marcus)

③ All pompeian were Romans

$\forall x$ pompeian $(x) \longrightarrow$ Roman (x)

④ Caesar was ruler.

Ruler (caesar)

⑤ All Romans were either loyal to Caesar or hate him.

$\forall x$ Roman $(x) \longrightarrow$ ((loyal to $(x,$ caesar $) \vee$ hate $(x,$ caesar $) \wedge x$ (loyal to $(x,$ cae

$\wedge$ hate $(x,$ caesar $)$)

⑥ Everyone is loyal to someone.

$\forall x \exists y$ loyal to (x, y)

(7) People only try to assasinate rulers they are not loyal to.

$\forall x \forall y$ people$(x) \wedge$ ruler$(y) \wedge$ try to assasinate$(x,y) \rightarrow \sim$ loyal to(x,y)

(8) Marcus tried to assasinate caesar

try to assasinate (Marcus, caesar)

(9) All men are people. $\forall x$ man$(x) \rightarrow$ person(x)

Was marcus loyal to caesar?

$\sim$ loyal to (Marcus, caesar)

↓(7)

Person (Marcus) $\wedge$ Ruler (caesar) $\wedge$ try to assasinate (Marcus, caesar)

↓(4)

Person (Marcus) $\wedge$ try to assasinate (Marcus, caesar)

↓8

Person (Marcus)

↓(9) Now no way to satisfy this

Man (Marcus)

In order to prove the goal, we need to use rules of inference to transform one goal into another goal and so on until there is no unsatisfied goal remains

Page :
Date :

12/11/16

Three problems in converting English sentence to logical statement.

1.) Many English sentences are ambiguous. chosing the correct interpretation may be difficult (more than one meaning)

2.) There are often choice of ways of representing the knowledge.

Simple representation are desirable but may preclude certain kinds of information.

3.) Even in simple situation a given set of sentences is unlikely to contain all the necessary information to reason about the topic at hand.

Ex:- All men are people.

Was Marcus loyal to Caesar?
loyal to (Marcus, caesar)
~ loyalto (Marcus, caesar)

It is very difficult to know which

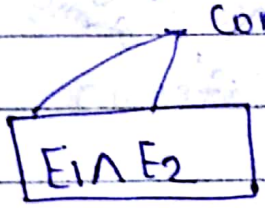
sentence to be deduced.

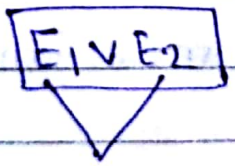
Computational predicate :—

When the no. of facts are not very large and if the facts themselves are sufficiently unstructured, then we can express as combination of individual predicates consider the following facts:

$9 > 1$ gt(3,1) $15 < 90$ lt(15,90) ↓ these are all the computable predicate.	there are infinitely large + it will be very inefficient to store them. But can be easily compute each one as we need it.

Revisit:





Predicates are function that maps objects arguments into TRUE or FALSE values.

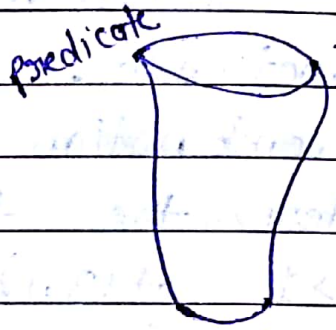
Quantifiers:—

Universal Quantifier: $\forall$ (for all)

Existential Quantifier: $\exists$ (there exist)

Date :

1.) Domain's object are terms.
2.) Variables ranging over a domains objects are terms.
3.) Functions are terms. [The argument to the function & the value returned are objects].



Atomic formula

eats (John, food(x))

literal

~ eats (John, food(x)).

atomic formula

Negation of atomic formula

connectives $\vee, \wedge, \sim, \rightarrow$ | literals | Quantifiers $\forall \exists$

well formed formula (WFF)

Modus Ponens :- If there is an axiom $E_1 \rightarrow E_2$ and there is another axiom E_1 then E_2 logically follow.

Modus Tolens :- If there is an axiom $E_1 \rightarrow E_2$ and there is another axiom $\sim E_2$ then $\sim E_1$ logically follows.

Page :
Date :

Resolution :- If there is an axiom $E_1 \vee E_2$ and there is a another axiom $\neg E_2 \vee E_3$ then $E_1 \vee E_3$ logically follows Resolvant of ~~there~~ is $E_1 \vee E_2$ & $\neg E_2 \vee E_3$.

Resolution Proof :-

Proving strategy: To show that the negation of a theorem can not be true. So the theorem is true.

Step1: Assume that the negation of the theorem is true.

Step2: Show that axioms and assumed negation of the theorem together determines something to be true that can not be true.

Step3: Conclude that the assumed negation of the theorem can not be true since it leads to contraction.

Step4: Concluds that the theorem must be true since the assumed negation of the theorem can not be true.

Proving a theorem by solving its negation can not be true is called proof by refutation.

Resolution Procedure :- The resolution operate

by taking two clauses that contains the same literal. The literal must occure +ve form in one clause & -ve form in the other clause.
The resolved is obtained by combining all the literals of the two parent clauses accepts the ones that cancel.
If the resolvant clause that is produced is an empty clause then contraction has been found.
If there is no contradiction, it is possible that this procedure never ends.

Conversion of WFF into clause form / canonical form / conjunctive normal Form:-
Sequence of steps:
Step 1: Eliminate "$\rightarrow$"
$a \rightarrow b$ is equivalent to $\neg a \vee b$
Step 2: Reduce the scope of negation using
(i) $\neg(\neg p) = p$
(ii) demorgan's law
$\neg(a \wedge b) = \neg a \vee \neg b$
$\neg(a \vee b) = \neg a \wedge \neg b$

(iii) Standard correspondence between quantifiers:

$\sim \forall x P(x) = \exists x \sim P(x)$

$\sim \exists x P(x) = \forall x \sim P(x)$

Step 3: standardize the variable.

Each quantifier binds to unique variable

$\forall x P(x) \vee \forall x Q(x)$ | X

$\forall x P(x) \vee \forall y Q(y)$ | ✓

Step 4: Move all the quantifiers to the left of the formula.

Now the formula is in prenex normal form (i.e the prefix of quantifiers followed by a matrix)

Step 5: Eliminate the existential qualifier

$\exists y$ president (y)

↓

Present (S_1) → skolem constant

S_1 is a function of no argument that produce a value which satisfy president

- - - - -

If the existential quantifier occurs within the scope of universal quantifier then the value that satisfy the predicate

may depend on the values of universally quantified variable.

$\forall x \exists y$ father of (y, x)

$\forall x$ father of $(S_2(x), x)$

↳ Skolem function

Step6: Drop the prefix (of quantifier) matrix remain

Step 7: Convert the matrix into conjunction of disjunct. using associative and distributive properties.

$(\vee \vee \vee) \wedge (\neg \vee \neg \vee \vee) \wedge (\neg \vee \neg \vee \neg)$

Step8:- call Each conjunct a seperate clause.

Step9: Standardize apart the variable in the set of clauses generated.

Resolution in ~~Properties~~ propositional logic:- Procedure for producing proof by resolution of given preposition S with respect to set anxioms 'F'.

1.) Convert all prepositions of 'F' into clause form.

2.) Negate 'S' and convert the result into clause form. Add it to the set of

clauses obtained in step1.

3.) Repeat the following either a contradiction is found or no progress can be made.

3.1 Select two clause call them as the parent clauses.

3.2 Resolve them together, the resulting clause is called resolvent will be disjunction of all literal of the parent clause with following exception.

3.2(a) If there are any pair of literal L and ¬L such that one of the parent clause contain L and the other contain ¬L then select one such pair & eliminate both L & ¬L from the resolvent.

3.3 If the resolvent is the empty clauses then a contradiction has been found if not add the resolvent to the set of clauses available to the procedure.

Prove R with respect to given axioms.

① P

② $(P \wedge Q) \rightarrow R$

③ $(S \vee T) \rightarrow Q$

④ T

(ii) $P \to q = \neg P \vee q$

$\neg(P \wedge Q) \vee R$

$\neg P \vee \neg Q \vee R$ (clause form)

(iii) $(S \vee T) \to Q$

$\neg(S \vee T) \vee Q$

$(\neg S \wedge \neg T) \vee Q$

$(\neg S \vee Q) \wedge (\neg T \vee Q)$

Clause form

(1) $\neg R$

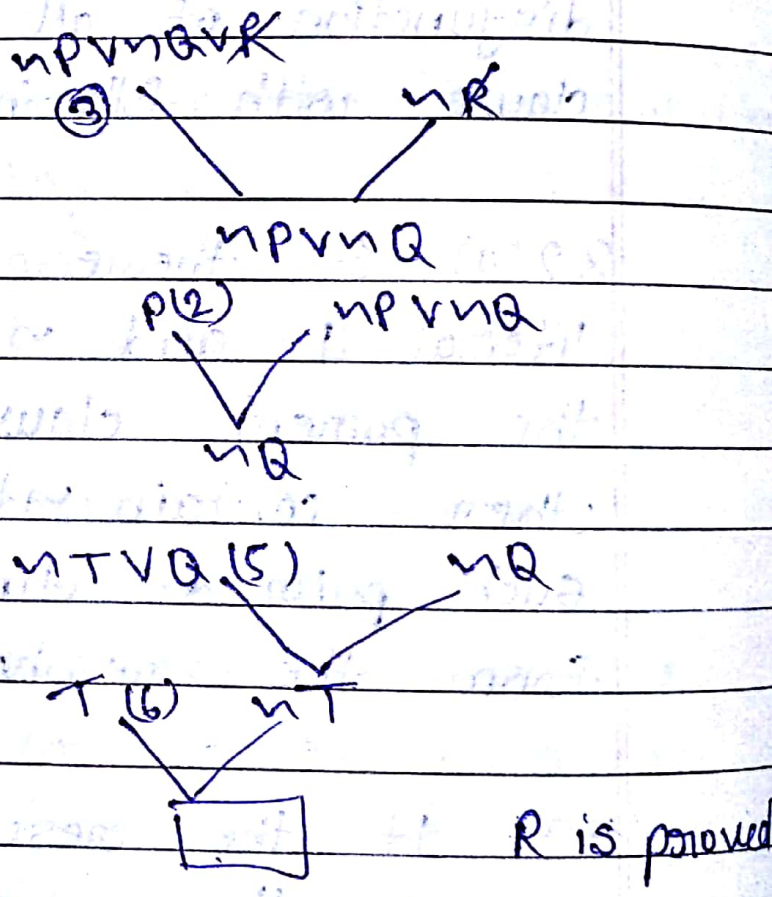
(2) P

(3) $\neg P \vee \neg Q \vee R$

(4) $\neg S \vee Q$

(5) $\neg T \vee Q$

(6) T



R is proved

Resolution in Predicate logic :-

Given : set of statement 'F'

S (has to be proved)

1.) convert all the statements 'F' into predicate logic and then into clause form.

2.) Convert S into predicate logic statement negate it and convert it into clause form. Add these clauses into set of clauses obtain in step 1.

3.) Repeat the followings until either contradiction has been found or no progress can be made or a predetermined work has been expended.

3(a) Select two clauses and call them as parent clauses.

3(b) Resolve the parent clauses the resolvent will be the disjunction of all the literal belong to both the parent clauses with following exception:

(i) If there are two literals 'T' and '¬T' appear in different parent clauses and both are unifiable then

(ii) neither of them should appear in the resolvent.

Use the substitution obtained by the unification process to create the final resolvent.

(3c) If the resolvent is an empty clause then contradiction has been found else add the resolvent to the set of clauses available to this procedure.

Unification Algorithm :-

In propositional logic it is easy to determine that two literals can not be true at the same time.
But in predicate logic it is not easy because the arguments are also need to be considered.

Contradiction ← Student (Neha) and ~ Student (Neha)
not-contradiction ← Student (Neha) and ~ student (spot)

So we need a matching process that compares two literals and discover whether there exist a solution that mates them identical.

Date :

Basic Rules :— but identical Predicate can match
Different predicate can not match.
Different constant can not match. but identical constant can match.

A variable can match with
- another variable
- any constant
- predicate expression

create the substitution
Except that the predicate expression must not contain any instance of the variable being matched.

Complications in the matching process:—
P(x,x) P(y,z)
y/x & z/x is not proper
y/x & z/y is proper.

hate(x, y)
hate(Marcus, z)
① Marcus/x, z/y
② Marcus/x, y/z
③ Marcus/x, Caesar/y, Caesar/z

Objective of Unification procedure :— To find atleast one substitution that cause to (2)

literals to match.

Algorithm unify (L_1, L_2)

(1) If L_1 or L_2 are both variable or constant then

(a) If L_1 and L_2 are identical then return NIL.

(b) Else if L_1 is variable then if L_1 occurs in L_2 then return {FAIL} else return (L_2/L_1)

(c) Else if L_2 is a variable then if L_2 occurs in L_1 then return {FAIL} else return (L_1/L_2)

(d) Else return {FAIL}

(2) If the initial predicate symbols in L_1 and L_2 are not identical then return {FAIL}.

(3) If L_1 and L_2 have a different no. of arguments then return {FAIL}

(4) set SUBST to NIL. (At the end of this procedure SUBST will contain all the substitutions used to unify L_1 & L_2).

(5) For i ← 1 to number of arguments in L_1:

(a) Call unify with the ith argument of L_1 and ith argument of L_2, putting

result in S.

(b) If S contain FAIL, then return {FAIL}

(c) If S is not equal to NIL then:

(i) Apply S to remainder of both L_1 & L_2

(ii) SUBST := APPEND (S, SUBST)

(iii) Return SUBST

loyal to (x, y)

- loyal to (Marcus, Caesar)

Consider the following Sentencess :-

(1) John likes all kinds of food.

(2) Apples are food.

(3) Chicken is food.

(4) Anything anyone eats and is not killed by is food.

(5) Bill eats peanuts & still alive.

(6) Sue eats everything Bill eats.

Translate these sentences into formulas in predicate logic. (*) Prove that John likes peanuts using backword chaining.

(**) Convert formulas into clause form

(***) Prove that John likes peanut using resolution.

(1) $\forall x$ food(x) $\rightarrow$ likes (John, x)

(2) food (apples)

(3) food (chicken)

(4) $\forall x \forall y$ eats $(y, x) \wedge \neg$ killed $(y) \rightarrow$ food (x)

(5) eats (Bill, Peanuts) $\wedge$ alive (Bill)

(6) $\forall x$ · eats (Bill, x) $\rightarrow$ (Sue, x)

Clause form :-

$\neg$ food $(x) \vee$ likes (John, x)

food (apple)

food (chicken)

$\neg$ (eats $(y, x) \wedge \neg$ killed $(y)) \vee$ food (x)

eats (Bill, peanuts)

alive (Bill)

$\neg$ eats (Bill, x) $\vee$ eats (Sue, x)

$\neg$ eats $(y, x) \vee$ killed $(y) \vee$ food (x)

$\forall x_3$ alive $(x_3) \rightarrow \neg$ killed (x_3)

$\Rightarrow \neg$ alive $(x_3) \vee \neg$ killed (x_3)

it can be write as

Resolution :-

$\neg$ likes (John, peanuts)

$\neg$ food $(x) \vee$ likes (John, x)

peanuts/x

$\neg$ likes (John, peanuts)

$\neg$ food (peanuts)

$\neg$ eats $(y, x) \vee$ killed $(y) \vee$ food (x)

peanuts/x

$\neg$ eats (y, peanuts) $\vee$ killed (y)

Bill/y

eats (Bill, peanuts)

killed (Bill)

$\neg$ alive $(x_3) \vee \neg$ killed (x_3)

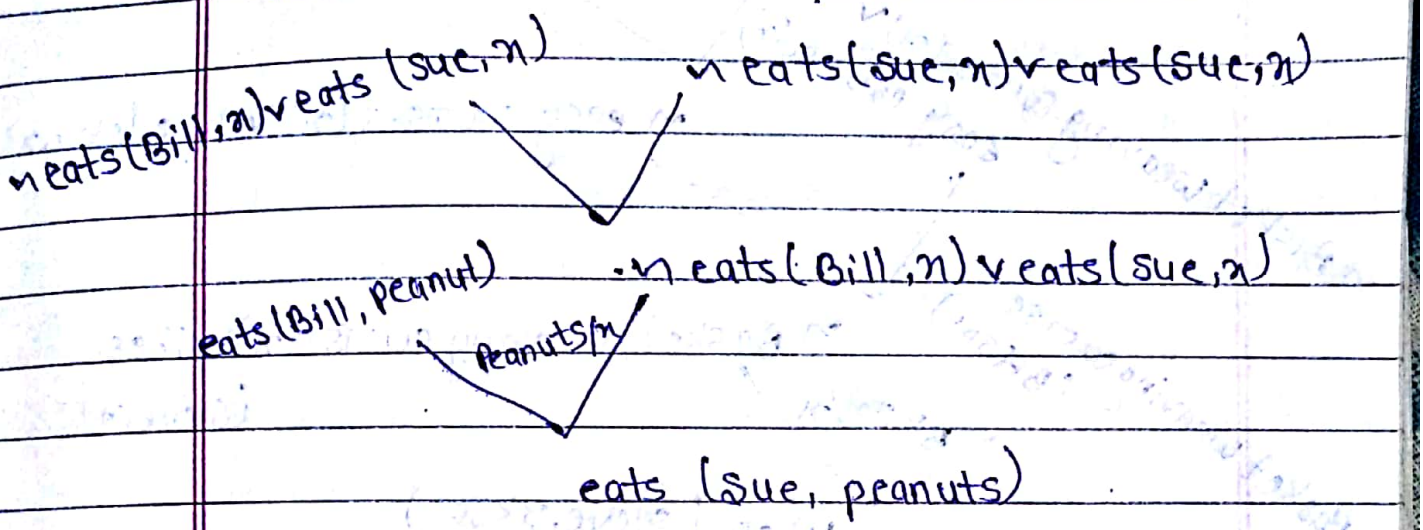
Bill/x_3

alive (Bill)

$\neg$ alive (Bill)

□

Use resolution to answer question what food does sue eats?



15/11/16

Assume the following facts:

(i) Steve only likes easy courses.

(ii) Science courses are hard.

(iii) All courses in the basket weaving department are easy.

(iv) BK301 is a basketweaving course.

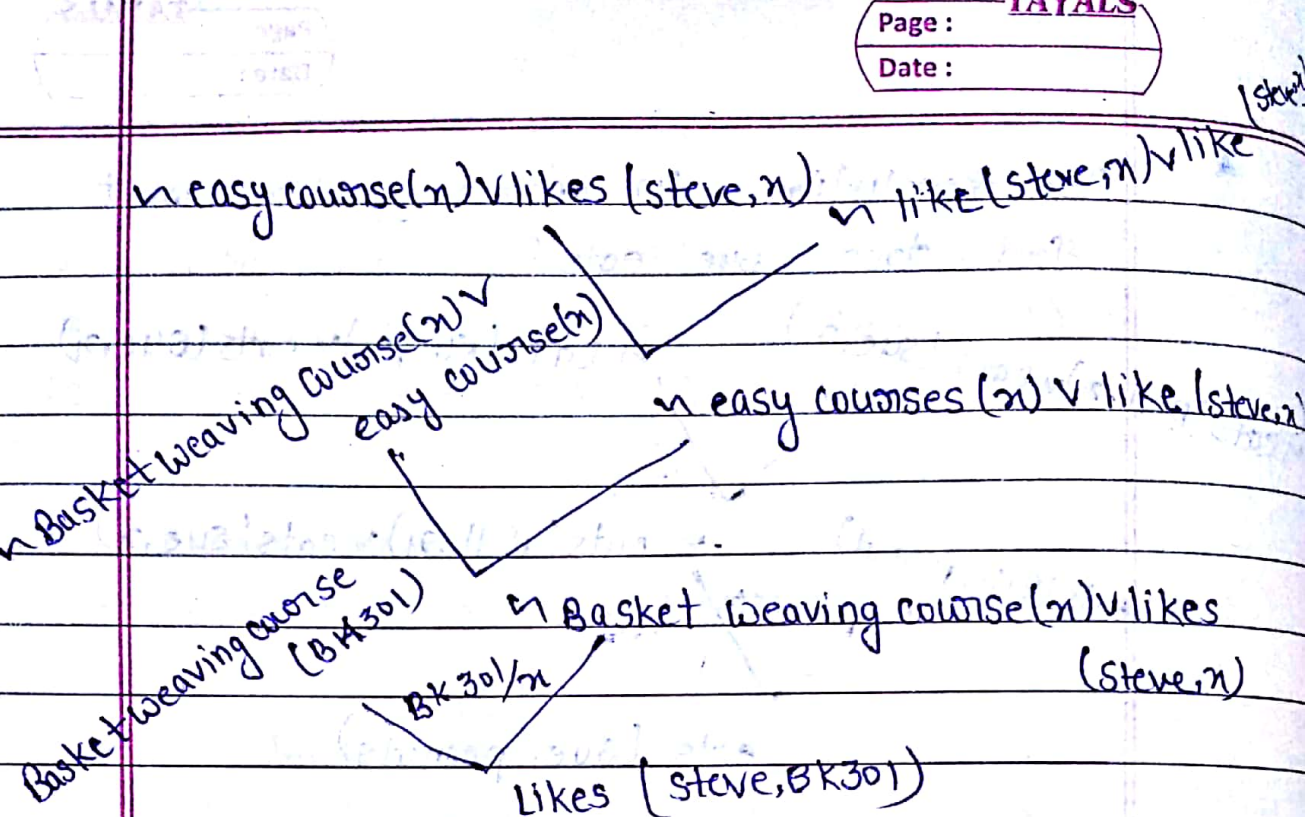
Use resolution to answer the question "What course would steve liks"?

(i) $\forall x$ easy courses $(x) \rightarrow$ likes $(steve, x) \Rightarrow$ $\neg$ easy course $(x) \vee$ like $(steve, x)$

(ii) hard courses (science)

(iii) $\forall x$ Basket weaving courses $(x) \rightarrow$ easy course (x)

(iv) Basket weaving course (BK301) $[\neg$ Basket weaving course $(x) \vee$ easy course $(x)]$



Properties of Good knowledge Representation system :-

* Propositional & predicate logic are useful for representing simple facts. The major advantage of these representation is that they can be combined with powerful inference mechanism called Resolution.

But the major difficulty with these representation is representing the complex knowledge.

(i) Representation Adequacy :- The ability to represent all kinds of facts pertaining to particular domain.

(ii) Inferential Adequacy :- The ability to manipulate the represented structure in such way to drive new structure that corresponds to

new knowledge.

(iii) Inferential Efficiency :— The ability to incorporate into knowledge structure the additional information that can be used to focus the attention of the inference mechanism in a most promising direction.

(iv) Acquisitioned Efficiency :— The ability to acquire new information easily.

Techniques for Representing knowledge