Artificial Intelligence Notes

Contributor: Deepak

[Aryabhatta (DU)]

Computer Science Notes

Date: / /
Page No.

Artificial Intelligence

PEAS
- Performance
- Environment
- Actuators
- Sensors

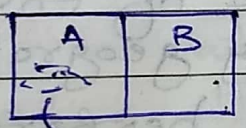
Think like human	Thinking Rationally
Act like human	Acting Rationally

⟹ Rational agent

Four approaches (चार तरीके होना)

✓ **Agents** - Anything that perceive its environment through sensors & act upon through Actuators

Percept Sequence

eg- 

environment consisting of two squares A & B

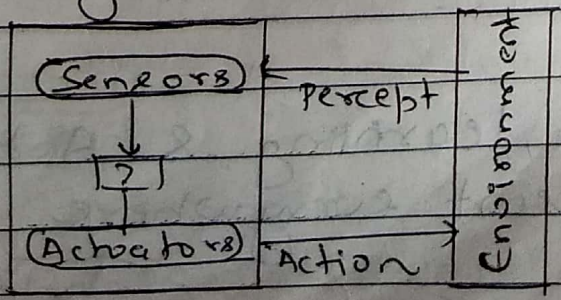
Vaccum cleaner (Agent)

Performance measure

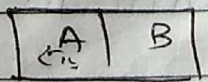
(Shd be coined Accordingly)

According to one which is required in environment i.e what actually one wants in the environment rather than how agent shd behave

Agent



(general structure)



vaccum cleaner

(left, right, clean, do nothing)

Date: / /
Page No.

Percept seq.	Action
[A, clean]	Right
[A, dirty]	Suck
[B, clean]	left
[B, dirty]	Suck
[A, clean] [A, clean]	Right
[A, clean] [A, clean] [A, dirty]	Suck

Percept Sequence

Rationality (How to know agent is rational (intelligent))

1. Performance measure
2. Agents prior knowledge of environment (Basically geography of Env.)
3. Actions that agent can perform
4. Agent's percept sequence till date

Definition - Rational Agent

For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built in knowledge the Agent has.

- Rationality
  - omniscience, learning & Autonomy?
    - being present everywhere
      - Detail in Book

Definition Book

- ~~Specifying the task Environment~~
  1. Performance
  2. Environment

Date: / /
Page No.

3) Actuators
4) Sensors

3 Taxi driver (Automated) (Agent) — Performance measure: Safe, fast, legal, maximize profit, comfortable trip

Environment	Actuators	Sensors
Road, other traffic, Pedestrian	Steering, Breaks, Horn, Accelerators	Camera, GPS, sensor, speedometer, odometer, fuelmeter

3 • Agent- Medical diagonosis system

P	E	A	S
Healthy patient, Reduced cost	whole Hospital	Reports, References, Test Display Questions	Readings + Patient's Answer

3 • Agent- Satellite Image Analysis System

P	E	A	S
correct Image categorization	Downlink from Satelite	Display of scene categorization	Pixels, Array

Task Environment

• Properties of task Environment
kind

1) Fully observable vs. partially observable
Eg. chess
agent fully observe the environment before next move

Automated taxi
partially observe

Date: / /
Page No.

A task environment is effectively fully observable if the sensor detects all aspects that are relevent to the choice of Action

2) Single Agent Vs Multiagent
(agent in crossword puzzle (x,o)
) chess Automatic taxi conceding other taxi as agent also.

• competetive Vs cooperative } Book
chess taxi

3) Deterministic Vs stochastic
(current state + action taken by agent → determine → The next state
chess
) taxi
if pedestrian comes in front of taxi, breaks will be applied bt we cannot determine whether taxi hit him or not

4) Episodic Vs sequential
satellite image analysis (chess & taxi driving
In an episodic task environment the agent's experience is divided into atomic episodes ~~the~~ each next episode does not depend on the action taken in previous episode

5) Static Vs Dynamic
while agent performing, the env. is not changing (chess) (crossword puzzle)
while agent is deliberating (performing) the environment is changing continously (taxi)

Date: / /
Page No.

(chess) (taxi driving)

5) Discrete Vs Continuous
7) Known Vs Unknown

the env. is fully known to agent (eg. vaccum cleaner)

) taxi driver

(6) Structure of agent

Agent = Architechture + Program

(Actuators + sensors) to take current percept in the form of input from the sensor & return an action to the actuators

agent program (current percept) / Agent function (History of percept sequence)

Table Driven Agent Program

```
fn Table-Driven-Agent (Percept) returns an action)
    Persistent: Percept, a seqn, initially empty. Table, a table of actions
    append percept to the end of percepts
    action ← lookup (Percept, table)
    return action
```

program to build a program

Date: / /
Page No.

Agent Construction

table construction
(for eg - we created table in vaccom cleaner)
↳ exponential probabilities

table - Driven Agent

function vaccom-Agent ([location, status]) return an action)
    if status = dirty
        then return suck
    else if location = A, then
        return right
    else if location = B then,
        return left

↳ build based on previous program

- Kind of agent program -

P = set of possible percept
T = Total no. of percept agent will receive

$$\sum_{t=1}^{T} p^t$$

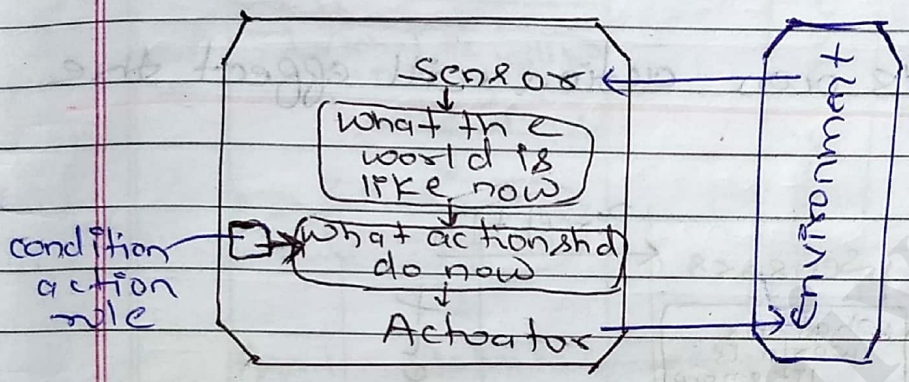
↳ total no. of entries in table

Date: / /
Page No.

1. Simple reflex agent
2. Model based reflex agent
3. Goal based reflex agent
4. Utility based reflex agent

Text diagrams & programs

a) Simple reflex



Simple reflex agent
focus on current percept & ignores the previous percept history

for fully observable environment

b) Simple reflex agent fn

fn simple-reflex-agent (percept) returns an action
static: rules: a set of condn-action rules
state ← Interpret-Input (percept)
rule ← RULE-MATCH (state, rules)
action ← RULE-ACTION (rule)
return action

Date: / /
Page No.

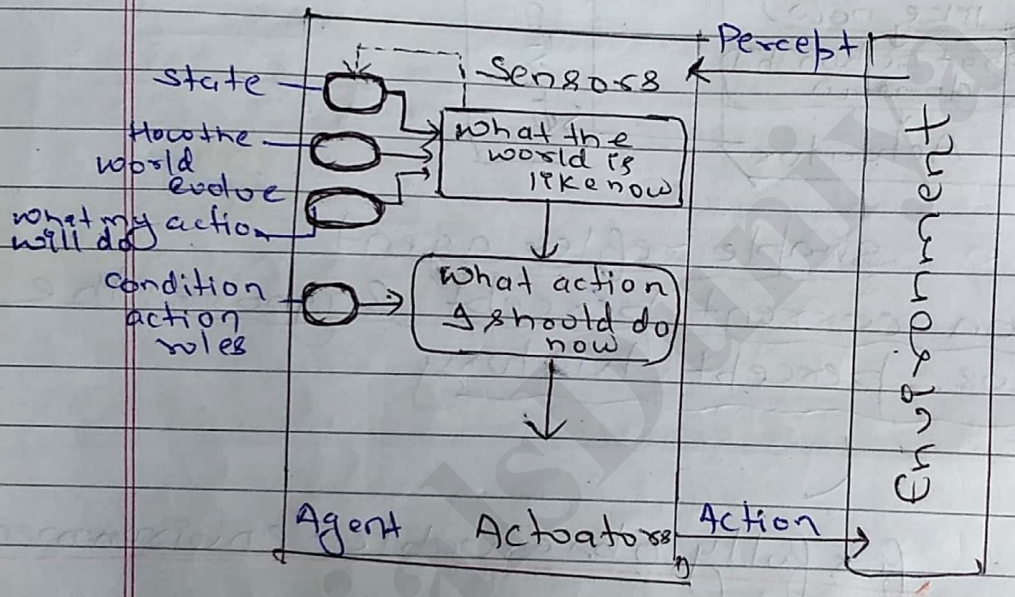
Model based Reflex agent

something which can handle partially observable environment

↳ has to keep track of internal state of
↳ has to " " " " percept environment.

① How the world evolve independently from agent

② How agent's own action will effect the world.

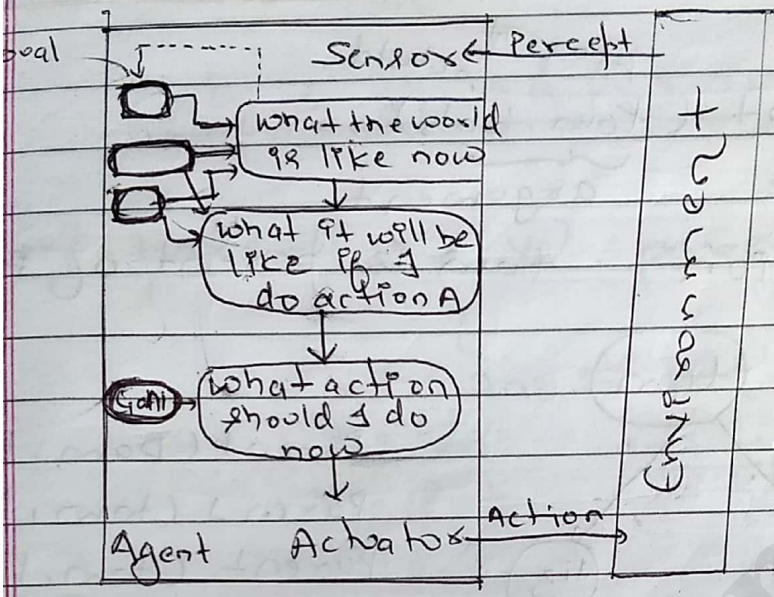


- way to handle partial observability
- For this some sort of internal state has to be maintained
- Updating this internal state information will require
  - How the world evolve independently of agent
  - How the agent's own action effect the world
- This knowledge about "How the world works" is called model.

Date:
Page No.

Through update state, new internal state description can be created.

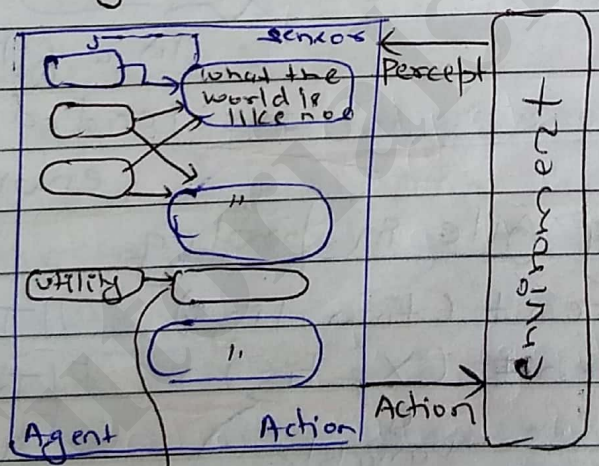
Goal Based agent



Along with knowing something abt the current state of Env, the agent need some sort of goal info that describes the situation that are desirable.

goal based agent are more flexible as compared to reflex based agent

Utility Based Agent



- Background info. used in process
- current internal state

How happy I will be in such a state

fn Reflex-agent-with-state (percept)
static: state, rules, action
state ← update-state (state, action, percept)
rule ← rule-match (state, rules)
action ← rule.action (rule)
return action

Model based agent ←

Date: / /
Page No.

Prolog Relations

(

basically it is a language for symbolic non-numeric computations

many binary

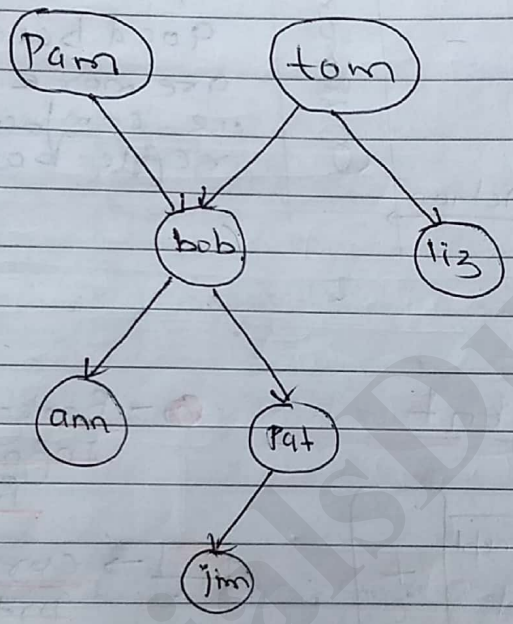
binary facts

parent (tom, bob)
parent — child

Name of relation

argument

tom is parent of Bob



parent (Pam, bob)
parent (tom, bob)
parent (tom, liz)
parent (bob, ann)
parent (bob, Pat)
parent (Pat, jim)

in a file & loaded in prolog environment

After loading file in prolog.

Query -? parent (tom, liz) //True

parent (X, liz) //True & gives an X

X — variable

tom

Atom (constant type of things)

parent (liz, Y)

False

liz is not parent of any child

parent (X, Y) gives all defined relation

then press ";" to get another one

give one one

Date: / /
Page No.

Ans -? parent(Y, jim), parent (X, Y)

Now,

Y = pat

X = bob

grandparent

Now,

bob is grandparent of jim

Ans -? parent (X, Y), parent (Y, jim)

Ans :- Pat

X = bob

Y = Pat

Ans -? parent (X, Ann), Parent (X, pat)

X = bob

(Do ann & pat have same parent)

Important Points—

- A prolog system consist of clauses, each clause terminate with full stop (.)
- The argument of relations can be constant/atoms or general objects such as X and Y called as variables.
- A prolog query can consist of one or more goals.
- Answer to a query can be +ve or -ve
- If several ans satisfy the query/question then prolog will find as many of them desired by user.

Date: / /
Page No.

(unary clause) (binary clause)

•) female(pam) or gender(pam, feminine)
male(tam) gender(tom, masculine)
;
;

Now, child(bob, pam)
└ we have to define it for every parent relation.

Is there are way to do better?

Rule

•) child(Y, X) :- parent(X, Y)
head body
conclusion condition } rule

For all X & Y, if X is parent of Y then Y is the child of X.
conclusion condition

•) for sister

```
sister(X, Y) :- parent(Z, X),
                parent(Z, Y), female(Y).
```

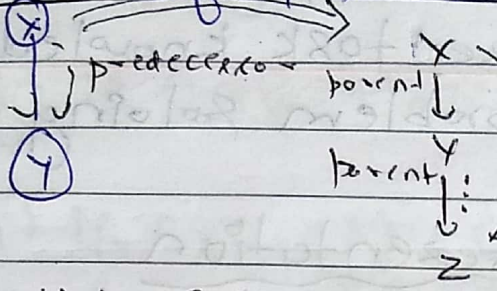
But if we query ?- sister(X, pat) assume that pat is female it will give out pat output
X - ann
X - pat

But ann is the answer ∴ pat is not a sister of itself.
∴ we have to specify that X, Y should be different. i.e. different(X, Y)

Recursion in prolog (for

Date: / /
Page No.

For all X & Z, X is predecessor of Z if there is a Y such that X is a parent of Y & Y is a predecessor of Z — (2)



For all X & Z, X is a predecessor of Z if X is a parent of Z — (1)

↳ predecessor(X,Z) :- parent(X,Z) // base case

predecessor(X,Z) :- parent(X,Y),
predecessor(Y,Z

Queries

?predecessor(pam,x) ⇒ Bald
and - bob, ann, pat, jim

Comment -

Singleline: %. . .

multipleline: /* . . .
. . . */

1) Mother(X,Y) :- parent(X,Y), gender(X, Feminine

2) GrandFather(X,Y) :- parent(Z,Y), parent(X,Z)

X
Z
Y

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

Date: / /
Page No.

Chapter

Problems, Problem spaces & search

1. Precise definition of the problem
2. Analyzing the problem
3. Isolate & represent task knowledge
4. Choose the best problem solving techniq

(To build a system to solve a pr we need to do fo this

⇒ State - space Representation

S.S representation forms the basis of m of the AI methods. Its structure corresp to the structure of problem solving in tw important ways -

① It allows for a formal defn of a problem as the need to convert some given situa into some desired situation using a set o permissible operations

② It permits us to define a process of solvi a particular problem as a combination of k techniques (eg. search techniques)

Jugs — Soln

4l 3l — 2l in 4l Jug

(x, y)

4l 3l

(0,0) → (2,0)

⇒ Production Rules

1. (x,y) if (x<4) → (4,y) fill the 4lt. j
2. (x,y) if (y<3) → (x,3) fill the 3lt. jug
3. (x,y) if (x>0) → (x-d,y) Poor some wat out of 4l jug

5. (x,y) if (x>0) → (0,y) Empty 4l jug on the ground

6. (x,y) if (y>0) → (x,0) Empty the 3l jug on the ground

7. (x,y) if (x+y≥4) & if (y>0) → (4, y-(4-x)) Pour the water from 3l jug into 4l jug

8. (x,y) if (x+y≥3) if (x>0) → (x-(3-y), 3) Pour the water from 4l jug into 3l jug until 3l jug is full

9. (x,y) if (x+y≤4) & (y>0) → (x+y, 0) Pour all the water from 3l jug into 4l jug

10. (x,y) if (x+y≤3) & (x≥0) → (0, x+y) Pour all the water from 4l jug into 3l jug

11. (0,2) → (2,0) Pour 2l from 3l jug into 4l jug

12. (2,y) → (0,y) Empty the 2l from 4l jug on the ground

4l jog	3l Jug	Role
0	0	→ 2
0	3	→ 9
3	0	→ 2
3	3	→ 7
4	2	→ 5
0	2	→ 11
2	0	

Solution of the problem

Que) 3 Issues with the conversion of informal problem statement into a formal problem discription. [like one shown above] (Pg. 27)

Date: / /
Page No.

Prolog

objects Relationship

likes (Deepak, Fruits)

?- likes(X, Fruits)
variable

likes(Xabc, Fruits)

likes('abc', Fruits)
in single quotes

instantiation is abt variable & Unification is about clauses, fact, rule

searching through the knowledge base to find a match related to fact & rule based on query

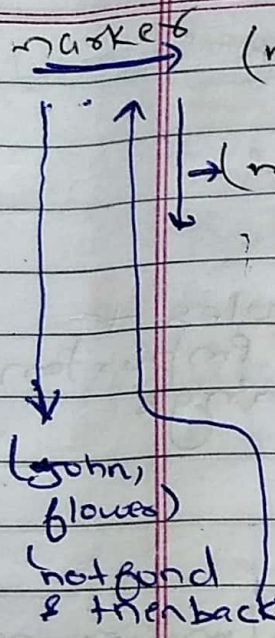
?- likes (mary, X), likes (john, x)

mary and john like something

for eg -
if it found (mary, flower) then it will place its marker on it & instantiate X with flower & then find (john, flower), if it found this then it will display X = flower & if (john, flower) is not in knowledgebase, then it will undo X & it will backtrack in knowledge base upto (mary, flower) & then find another (mary, y) & repeat process

first time it is found in knowledge base

Date: / /
Page No.



In conjuction the situation goes like this.

1) If the first goal succeeds, the variable X is instantiated
2) Next, attempt to satisfy the second goal.
3) If the second goal fails.
4) It will backtrack, forgett the previous instantiation of the variable & attempt to resatisfy the goal.

5) Problem :-
Monkey & lion.

Date: / /
Page No.

Production System :-

(full def^n in book)

Production System consist of -

(1) A set of rules
(2) One or more Knowledge database
(3) Control strategy
(4) A rule applier.

(set of rules + other important things)

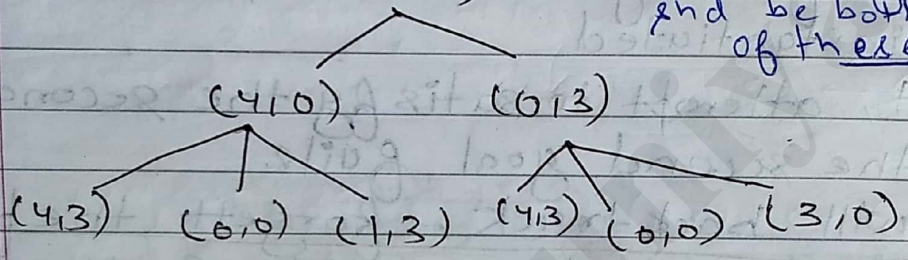
(~) control strategy

Systematic

motion/movement

control strategy and be both of these.

Jug problem (0,0)



control strategy now can be BFS & DFS. & Heuristic.

- DFS is going to require less memory than BFS & soln can be found on a particular path.
- BFS - we can explore each level

Heuristic Search is basically used in combinatorial explosion. It may not give the best ans but can give good ans.

TSP - (travelling salesman problem)
(a traveller has to start at a point & has to go 10 cities & come back to the starting point.

(10! to compute all the distances. (10 × 9 × 7 ... ×1)

Date: / /
Page No.

use

we will nearest neighboor ~~problem~~ to solve the problem. i.e the salesman will go to the nearest city.

complexity = N^2

Problem characterstics

1. wether problem is decomposable / Non-decomposable.

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

decomposable

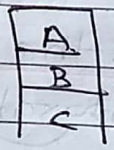
like $\int x^2 + \int 3x + \int \sin^2 x + \int \cos^2 x$

Block problem -

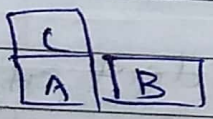
clear(x) → ON(x, table)

block x has nothing on it

Goal →



i.e



clear(C) → A C B

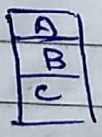
Non-decomposable

we cannot move A untill we have clear it from top.

ii) clear(x) and clear(y) ⇒ ON(x, y)

[Put x on y]

A B C →



clear(B) and clear(C) → on(B, C)

depends on each other so non-decomposible

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

on(B,C) ON(A,B)

comb^n of these subgoln into one goln will get

2) Advantages & disadvantages of heuristic
3) " " " of BFS & DFS
4) Algo for BFS & DFS

Date: / /
Page No.

(ii) Can solution step be ignored or undone?
8 puzzle : recoverable
chess : unrecoverable
Theorem proving : ignorable

(iii) Is the universe predictable?
eg- chess - predictable
cards - unpredictable

(iv) Is the good soln absolute or relative?
(v) any path problem vs best path problem

(vi) Is the solution a state or a path?
Natural lang understanding, water jug problem
eg.
given interpretation of
"The bank president ate a dish of pastasalad with the fork".
which type of bank — what is this. Diff from dog salad — whether used fork as a tool, or has eaten fork also like "with the vegetables"

3 Sb kuch interpret krne k baad jo soln aaya us pe concern h sirf. So it is a "state" while in water jug problem, sb kuch pta hona chahiye. So it lies in a "path"

9) What is the role of knowledge?
How much knowledge is required to solve a problem. eg: Sentiment analysis will require higher knowledge while tic tac toe will require lesser.

Date: / /
Page No.

Prolog :-

likes (mary, food)
likes (mary, wine)
likes (john, wine)
likes (john, chocolates)

?- likes (mary, X),
likes (john, X).

Syntactical Details -

term — constant
— variable
— structure

Anonymous variable

check
?- likes (_, John)
?- likes (John, _)

Compound terms

date (24, 1, 2019) → structure

- Function
- component of str.

can be used in libraries eg.

likes (John, Book (Algo, 2018, cormen))
structure

owns (mary, book)
owns (john, book)
owns (john, withering_height)
owns (mary, moby_tenee)

⚠ it is confusing as we are unable to clarify what it signifies

owns (john, book (withering_height, (cormen))
still confusing

owns (john, book (withering_height, author (emily, brook)))
structure form

Mathematical operators-

+ (x, * (y,z)) :- x + y * z   check +(3*(5,7)) Not working

clause clause   1st means ?

A B

reigns(rhodri, 844, 878)
reigns(anarwd, 878, 916)
reigns(A, 916, 950)
reigns(B, 950, 979)
reigns(C, 979, 985)
reigns(D, 985, 986)
reigns(more, 986, 999)

greater than equal to in prolog >=

equal check X =:= Y same of ==

Ab agar ruler clause nikalna ho to agr. pass krke X & Y (ruler name, year doing the), pehle dekhna h A≤Y≤B and X = name

ruler(X,Y) :- reigns (X, A, B), Y>A, Y=<B

pop(usa, 203)
pop(india, 548)
pop(china, 800)
pop(brazil, 108)

?- ruler(rhodri, 845)
?- ruler(more, 999)
?- ruler(X, 980)
Ans = C

area(usa, 3)
area(india, 2)
area(china, 1)
area(brazil, 5)

Population density of country X is Y (pop. divided by area). If pop. of X is P and area of X is A, Y is calculated.

density(X, Y) :- pop(X, P), area(X, A), Y is P/A

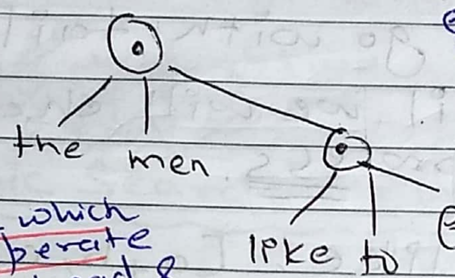
(•) LIST
↳ used in search specifilly using recursion

Empty list —denoted by→ [ ]

head, tail
↳ first one letter will be head or rest is tail

We use dot (•) at root

eg- [the, men, [like, to, fish]]
the = head; like = head; to, fish = tail; men, [like, to, fish] = tail



→ [a, b, c]
a = head; b, c = tail

eg- [a| bc] (pipe which seperate head & tail)

→ [ ] – head none, tail none

(•) list1 list2 Instantiation

[X, Y, Z] [John, likes, fish]
X = John = head
Y = likes
Z = fish] tail

[cat] [X|Y]
X = cat
Y = [ ]

[X, Y|Z] [mary, likes, wine]
X = mary } head
Y = likes
Z = [wine]
↳ by default any change karenge to list me dege

[[the, Y]|Z] [[X, hare], [is, here]] ?
X = the
Y = hare
Z = [[is, here]]

[golden|T] [golden, norfolk]
T = [norfolk]

? [vale, horse] [horse|X]
X = None (not Instantiate)
(head are not same so false)

Date: / /
Page No.

Searching in a list (Recursive) (X|Y)

[a, b, c, d, e, f, g, h]
head — tail

First we will search head, if its the element we want, we will return it else we will go with tail [b, c, d, e, f, g, h] and in this tail we will check head and repeat this process.

If list is like [a, b, c, d, e, f, g, h]
head, tail

Now if we have to search 'b', we will take head & then apply above procedure.

member(a, [a|b, c, d, e, f, g])
true

(1) member(X, [X|_])
(Anonymous
X is the member of list that has X as its head. // unifying with head

member(X, [Y|_]) :- X = Y
if X = Y, then X is the head
then clause will satisfy

(2) member(X, [_|Y]) :- member(X, Y)
X is the member of the list if X is the member of tail of list i.e Y

member(X, [X|_]).
member(X, [_|Y]) :- member(X, Y).
?- member(d, [a, b, c, d, e, f]) - Yes

Date: / /
Page No.

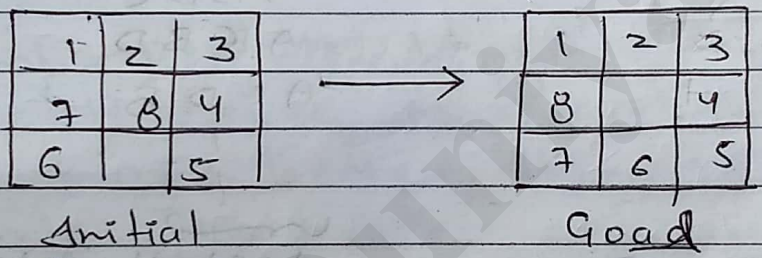
Program –

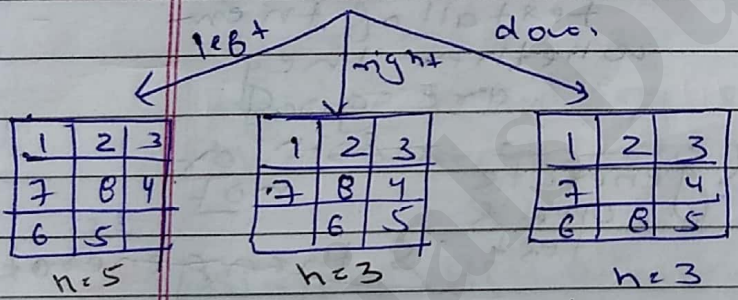
```
member(X, [X|_])
member(X, [_|Y]) :- member(X,Y)

?- member(d, [a, b, c, d, e, f])
```

true

(•) Heuristic Search Technique

weak because it may not give best possible answer.





Which move is best?

Blind search techniques used an arbitary ordering of operation. Heuristic search technique make use of domain specific information in the form of heuristic.

Approach 1) A simple heuristic for 8 puzzle problem could be the no. of tiles in the correct position, the higher the no., the better the result.

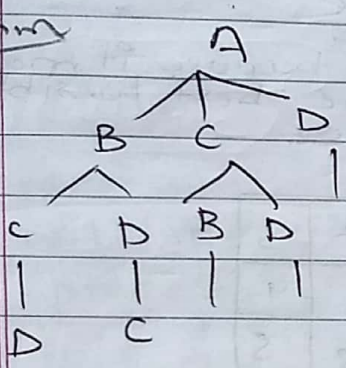
Approach 1) Another approach could be no. of tiles in the incorrect position. The best move is the one with the lowest no. returned by the heuristic

Date: / /
Page No.

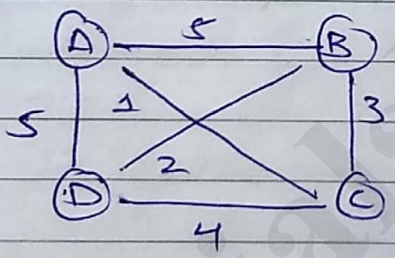
Approach(iv) Another heuristic could be the count of how far away (how many tile movements) each tile is from its correct position. Sum of this count over all the tiles. This will be an another estimate on the no. of moves away from a soln.

- **Generate & Test Strategy**

Algorithm



lexicographical -
A B C D
A B D C
A C B D
A C D B
|
|
test all of them whether they are goal soln or not



generate one & test

& then generate other & test & go on like this

- **Hill climbing**

) simple Hill climbing } Algorithms
) Steepest Ascent" "

generate & test + a direction to move on the basis of a Heuristic function

generate all possible situation & find the best one

Deepak

Date: / /
Page No.

Hill climbing = generate n test + a dirn to move

Heuristic fn estimates how close a given state is to goal state.

Hill climbing

~~Simple Hill climbing Algorithm~~

1) Generation of next state depends upon feedback from the test procedure.
2) Test includes a Heuristic fn that provides a guess as to how good each possible is.

Simple Hill climbing -

1) Use Heuristic to move only to states that are better than the current state. The process ends when all operators have been applied & none of the resulting states are better than the current state

Simple Hill climbing Algo (ss)

1. Evaluate the initial state
2. loop until a soln is found or there are no new operator left to be applied.
   (i) Select & apply a new operation
   (ii) Evaluate the new state
   goal → quit.
   If the new state is better than current state, update the current state.

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

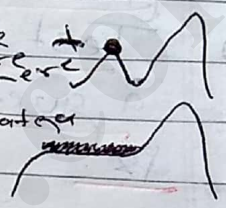
Please Share these Notes with your Friends as well

Date: / /
Page No.

Heuristic fn is a way to inject task specific Knowledge into the control process

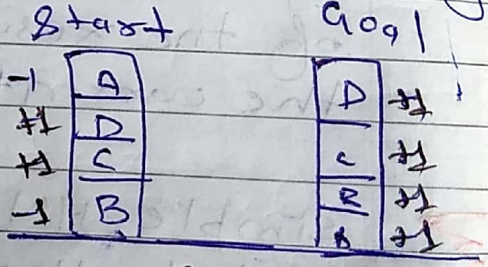
(w) Potential problems with simple Hill climbing

1) local optima (local minima & local maxima)
Local maximum is a state that is better than all of its neighbour but not better than some of the other states far away
2) Plateau
3) Ridge



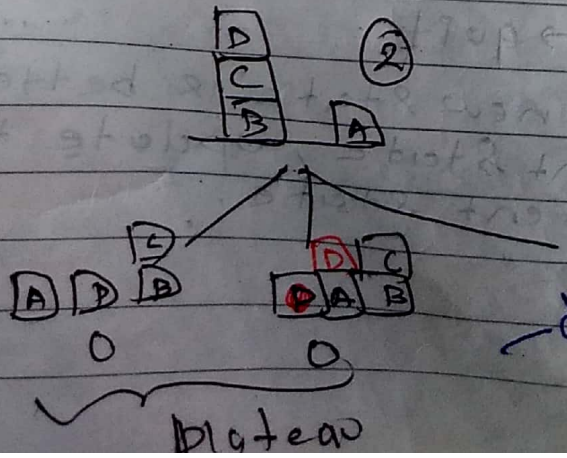
0) Steepest Ascent Hill climbing

Example of Simple Hill climbing.



Local Heuristic is - +1 for each block that is resting at correct position
-1 for each block that is resting at wrong place thing

Now,



disadvantage of simple Hill climbing

Date: / /
Page No.

Hill climbing can be very inefficient for a very large problem space. Local Heuristics may fail, global Heuristic may have to pay for compotational complexity. Simple Hill climbing is often useful when combined with other methods

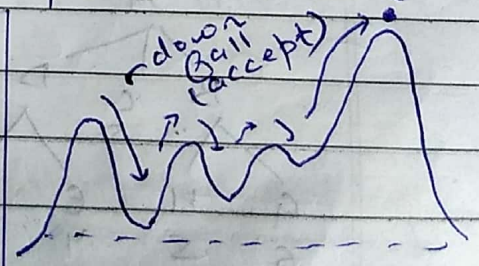
steepest Ascent Hill climbing

Consider all the moves from current state, Select the best one as the next state. It is not just climbing to a better state, but climbing up the steepest slope.

) Simulated Annealing (ω)

current state → opⁿ → new state

It says agr new state previous state se worst hai, to bhi accept kr lo. Kyunki aage home better result mil skta hai.



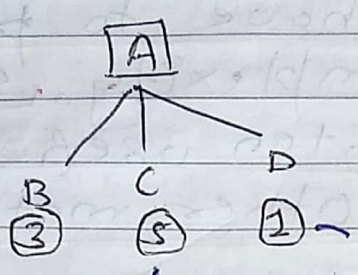
/block It is a variation of hill climbing in which at the

/block begning of the process some downhill moves may be made. This is done to explore the whole space early. This will lower the chance of getting caught at a local maxima

Date: / /
Page No.

o) Best first search

Combine features of breadth & depth first search



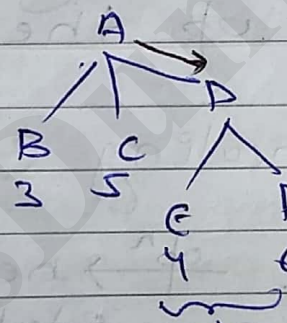
1 move to reach goal

Give more moves to reach goal

estimate not good

on basis of heuristic

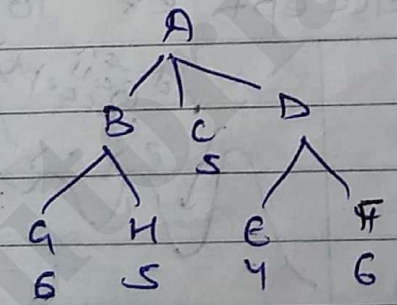
Say for eg we go to D



Nodes can be open or close

Now this is worst coz they require more moves to reach goal

Now, we go to B from D.



Now G-6 & H-5

so we go to E after opening B.

Open - Generated bt not examined

Close - examined

eg- D is close & C is open as well as G, H, E, F

leaf

non leaf

Date: / /
Page No.

In depth first search, not all competing branches have to be expanded. Breadth first do not get trapped on a dead end path.
Best first search is about combining the two. In this a single path is followed at a time bt swith path whenever some competing path look more promising than the current one.

Open Nodes- Nodes that have been generated but have not examined.

Closed Nodes- Nodes that have already been examined

Whenever a new node is generated, check whether it has been generated before
It is similar to steepest Accent but dont throw away the states that are not chosen.

$f' = g + h'$

- f': total estimated cost from start to goal.
- g: cost from initial/start to current state
- h': estimated cost from current to goal state

greedy Dynamic paradigm

Date: / /
Page No.

Algorithm

1. OPEN = {initial state}
2. Loop until a goal is found or there are no nodes left in open.
   i) Pick the best node in open
   ii) Generate its successors
   iii) For each successor
      new → evaluate it
      Add it to open
      record its dater

• A* Algorithm

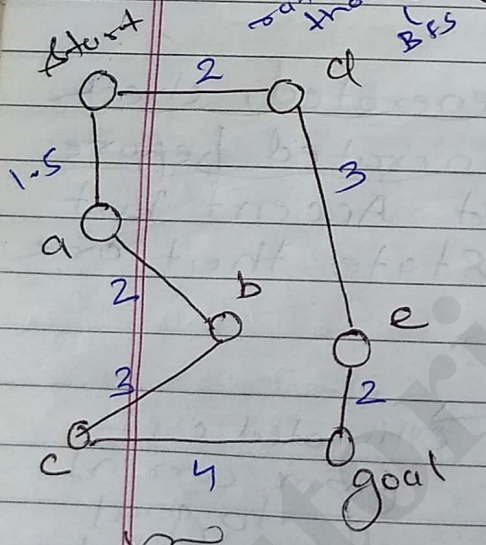
(more restricted than best first search

(more focus on h' rather than BFS

$f' = g + h'$

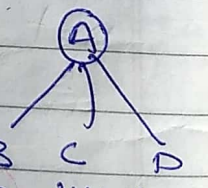
we will segregate them

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



we have been given values of g(n)

if same for each node, it will be Breadth first search than



g.

like

$f(a) = 1.5 + 4$ — we will prefer this

$f(d) = 2 + 4.5$ — given in Question

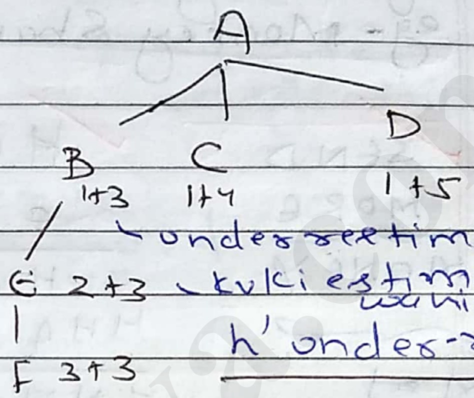
$f(b) = 3.5 + 2$ — move from a to b

$f(d) = 2 + 4.5$ — (back track

set of nodes is maintained as priority queue so we need not track back we can go to previous node we will go to b

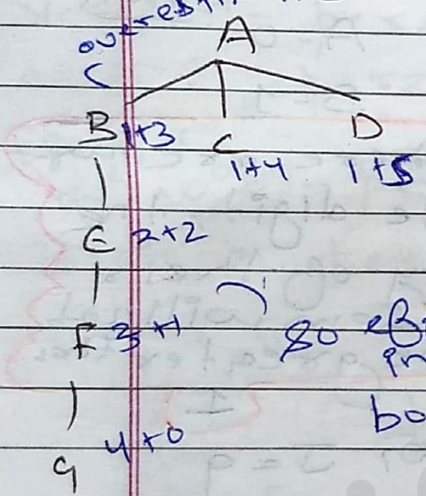
f(c) = 6.5 + 4
f(d) = 2 + 4.5 — we will use this

f(e) = 5 + 2 — go with this
f(c) = 6.5 + 4



underestimate
kuki estimate wahi tha
h' underestimate
h' < h
we are efforting but not moving
i.e algorithm is stuck

h' overestimate h



so effort increases
but best soln can be another path

More the overestimate, more the effort

Date: / /
Page No.

Constraint Satisfaction

- Monkey & banana problem

Cryptarithmetic

Solve the problem by following the constraints

eg- Monkey & banana

eg- (all have diff value)

```
  J E
x B B
-----
  J E
J E A
-----
B A D E
```

we have to find their values

A = 0

B = 1

If there are two single digit, the carry of their sum will not be greater than 1

So, J = 9

E cannot be 0, 1, 9 & 2

```
  SEND
+ MORE
-------
 MONEY
```

M = 1

carry cannot be more than 1)

```
 9567
+1085
-----
10652
```

M = 1

```
  HE
+ EH
----
  HE
HHA -
-----
HNME
```

H = 1
E = 7
A = 9
M = 0
N = 2

Q) WIRE + MORE = MONEY

(

```
 9274
+1074
-----
10348
```

one carry

```
 9762
+1062
-----
10824
```

record sort

sort

```
WRONG
WRONG
-----
RIGHT
```

```
 24153
+24153
-----
 48306
```

```
GERALD
DONALD
-------
ROBERT
```

ODD
ODD
EVEN

D = 5 N = 0 E = 1 O = 6, 8

```
 TWO
+TWO
----
FOUR
```

```
 846
+846
----
1692
```

```
 928
+928
----
1856
```

Date: / /
Page No.

Prolog (LIST)

parent(X,Y) :- child(Y,X)
child(Y,X) :- Parent(X,Y)

prolog with stuck in loop — we have to avoid it

circular predicate definition

Left Recursion –

version1 – person(adam)
(person(X) :- person(Y), mother(X,Y)
it will not come out of here

end of circular loop ?- person(X) – ? not work

version2 person(X) :- person(Y), mother(X,
person(adam)
(it will work ?- person(X)

it will work, go to adam

which version is going to terminate?
(open question)

Prolog program

Q) WAP in prolog for sum

Sum(X,Y) :- S is X+Y
write(S)

(1) Constraint satisfaction reduces the amount of search: The problem of constraint satisfaction like crypt arithmetic with initial set of constraints & restrictions that can be infer. from the rules of arithmetic are augmen
(2) This will reduce the no. of allowable guess
(3) Two step process -
(i) Constraints are discovered & propogate as far as possible throughout the system.
(ii) Then if there is still not soln, the search begins.
(ii).1 A guess abt something is made and added as a new constraint

Constraint propogation terminates for one of the two reason-
1) Detection of contradiction
2) Propogation has run out & there are no further changes that can be made out of current knowledge.

Use of Heuristic in Constraint Satisfaction -
For the second step of CS, the order in which search options are tried may have a substantial effect on the

Date: / /
Page No.

degree of search. For ex. If there is a letter with only two possible value & other with six possible value, then we will try the first one. This is Heuristic.

(2) If there is a letter that participates in many constraints. then it is good idea to prefer it to a letter that participate in few. (Another Heuristic)

Prob:- SEND
MORE
MONEY

Initial state-

1) No two letter have same value
2) the sum of digit must be as shown in the problem

Prolog :-

left recursion -

for satisfying person(x), prolog would first use the rule & generate subgoal person(Y). In trying to satisfy this it would again pick the rule first & generate yet another equivalent subgoal. (case I).

Sol:- The actual trouble is that in order to backtrack prolog has to have has failed after trying the first possibility

Date: / /
Page No.

MAPPING

?- alter([do, you, know, french], X)

ans shd be
no ∃ know germ al

alter predicate shd change the head of the input list into another word & let the head of the output list stand for that word.

(2) Use alter on the tail of the input list and let the tail of the output list stand for the altered tail.

(3) If we have reached the end of the input list, then there is nothing more to go onto the output list, so we can terminate the output list with an empty list. []

Prolog Program

```
alter([],[]).
alter([H|T],[X|Y]):-
        change(H,X),
        alter(T,Y).

change(you,i).
change(are,[am,not]).
change(french,german).
change(do,no).
change(X,X).
```

?- alter([You, are, a, computer], X)
X = [i, [am, not], a, computer]

MEA

Means End Analysis

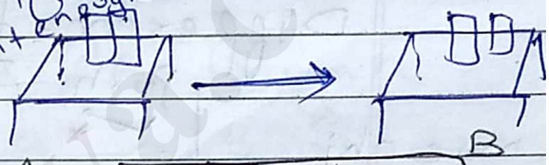
Means → tools (साधन)

A ⟶ B ⟶ D ⟶ G (from B to D: opr^n)

- A to B: what to apply here to reach B
- B to D: op^n to take us from B to D

Operators:

1. Push – when we push, table shd not have any object on it
2. Carry – carry if it is light enough



home robot
Here, we have to move table with objects on it from A to B → goal

3. pickup
5. putdown
6. place
9. walk

Seq^n of operation

walk, PU, PD, PU, PD, push

Operator	Precondition	Results
(1) PUSH (obj, loc)	at(robot, ob) armempty clear(obj) largeenough	at(obj, loc) at(robot, loc)
(2) CARRY (obj, loc)	" except large enough (smallenough)	"
(3) WALK (robot, loc) or walk (loc)	none	at(robot, loc)

defⁿ of operator & subgoaling

(4) Pickup(obj) — at(rob, obj), armempty, small enough — holding(obj)

(5) Putdown(obj) — holding(obj) — ¬holding(obj) or empty arm

(6) Place(obj1, obj2) — at(robot, obj2), holding(obj1) — on(obj1, obj2)

Difference table

	Push	Carry	Walk	Pickup	Putdown	Place
move obj	✓	✓	✓			
move robot						
clear obj				✓		
Get obj on obj						✓
be holding obj		✓		✓		
get armempty					✓	✓

Algo –

Step1 – Until the goal is reach or no more procedures are available

(i) Decribe the current state, the goal state & the diff. b/w two.

(ii) Use the difference that describe a procedure that will hopefully get near the goal.

(iii) Use the procedure & update the current state.

Step2 – If the goal is reached then success otherwise fail

Date: / /
Page No.

Game Playing

(imp topic of AI.

- search
- non search

for two reason

1) Structure task ← success / failure
2) large amt of knowledge is not required

8-puzzle

winning

35^100 moves in chess (for eg.)

↳ Not applicable for all games

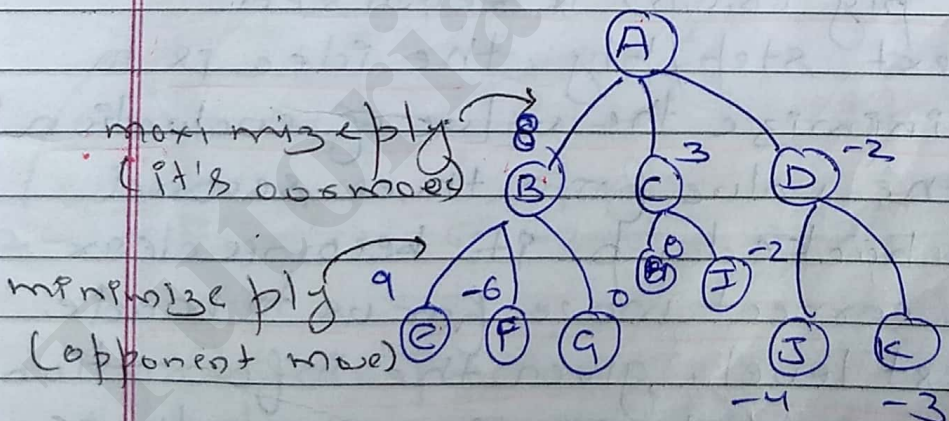
Search
↳ generate & test
↳ Based on evaluative f^n

Plausible move generator
(gives more promising moves instead of all.

1) The Minimax Search Procedure

ply
(moves set

we are computer Here



no. are based of evaluative f^n

We have to make first move, so we go to B. Now its opponent move, but He can take path of E i.e 9, so its our loss. Same is with option C. So computer shd take move D.

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

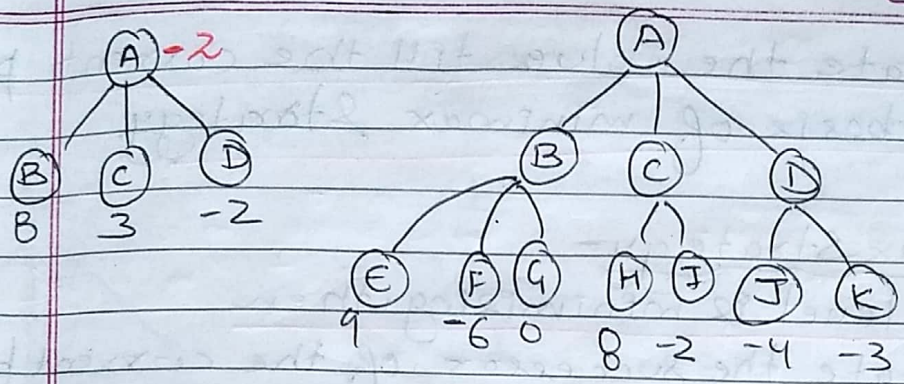
value are in range, maximize
-10 → 10
opponent win
Date 1/1
Page No. 42

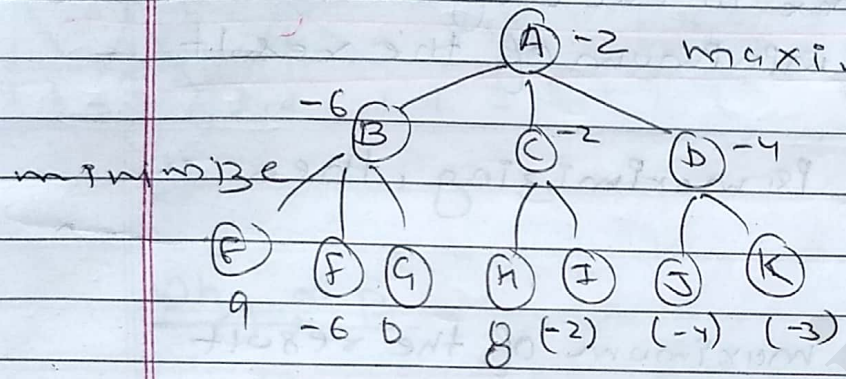
We to maximize our move in such a way that minimizes the opponent's move.
If machine can think of about 30 moves, it means it has min-max tree with 30 levels.

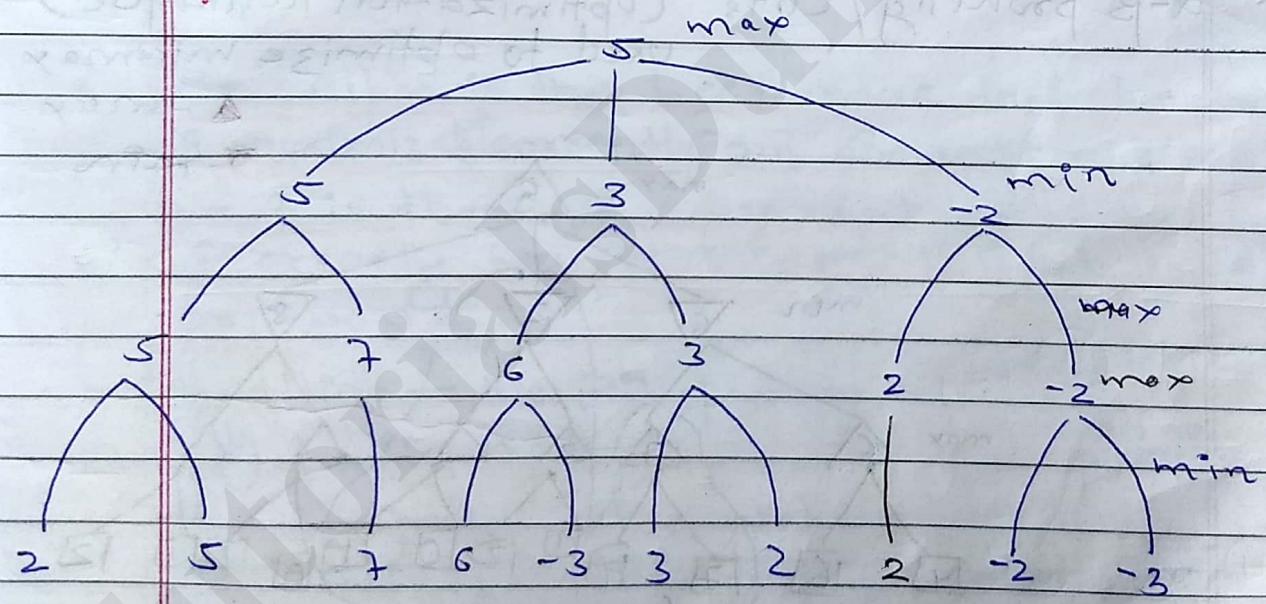
Sub points

The minimax search
1) Depth first Depth limited procedure
2) Start at the current position and use the plausible move generator to generate the set of possible successor position
3) Apply the static evaluation fn & choose the best one.
4) Now back that value up to the starting position, Goal is to maximize the value of the static evaluation fn of the next board position.
5) To carry the search further ahead then one ply (B, C, D) is required.
6) At the next step ply, the idea is to minimize the value of evaluation fn. once the value from the second ply are backed up it becomes clear that the correct move for us to made at the first level, given the information we have available is C, since there is nothing opponent can do from there to produce a value worse than -2

Date: / /
Page No.







Minimax Procedure :-

- Keep on generating the search tree till the required limit, say depth of the tree (d), has been reached from the current position
- Compute the static values at depth d from the current position using evaluation fn

Date: / /
Page No.

propogate the values till the current path on the basis of minimax strategy

Ans

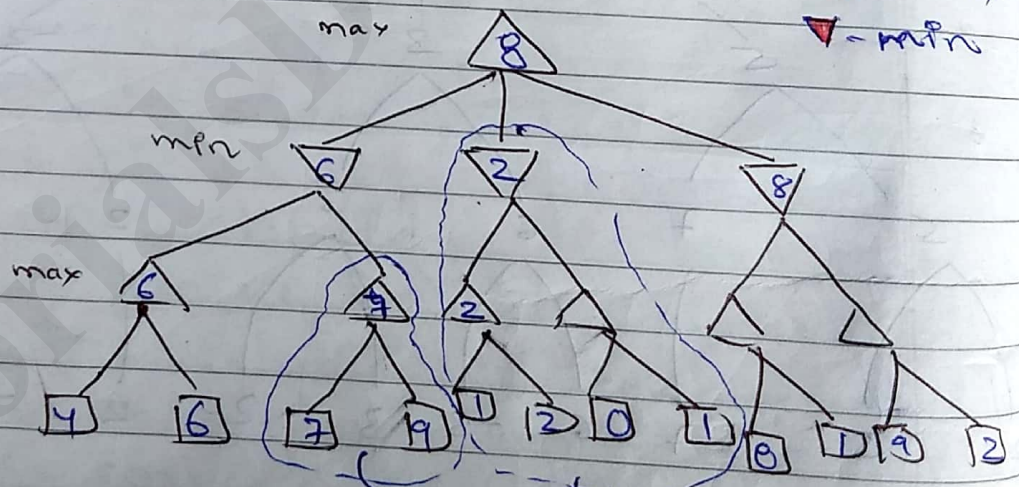
Minimax Strategy -

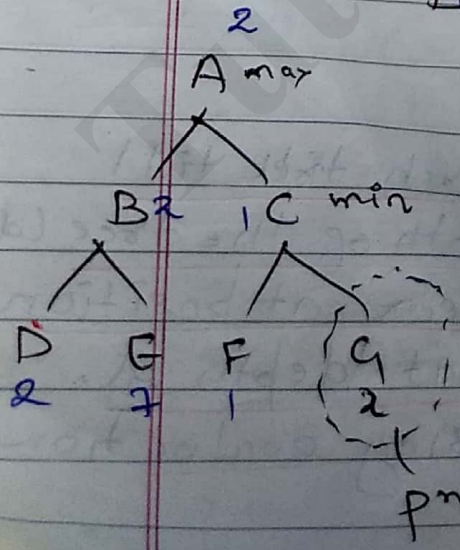
→ If the level is minimizing, then
> generate the successor of the current path
> apply minimax to each of the successor
> return the minimum of the result.

→ If the level is maximizing, then
> ''
> ''
> return the maximum of the result

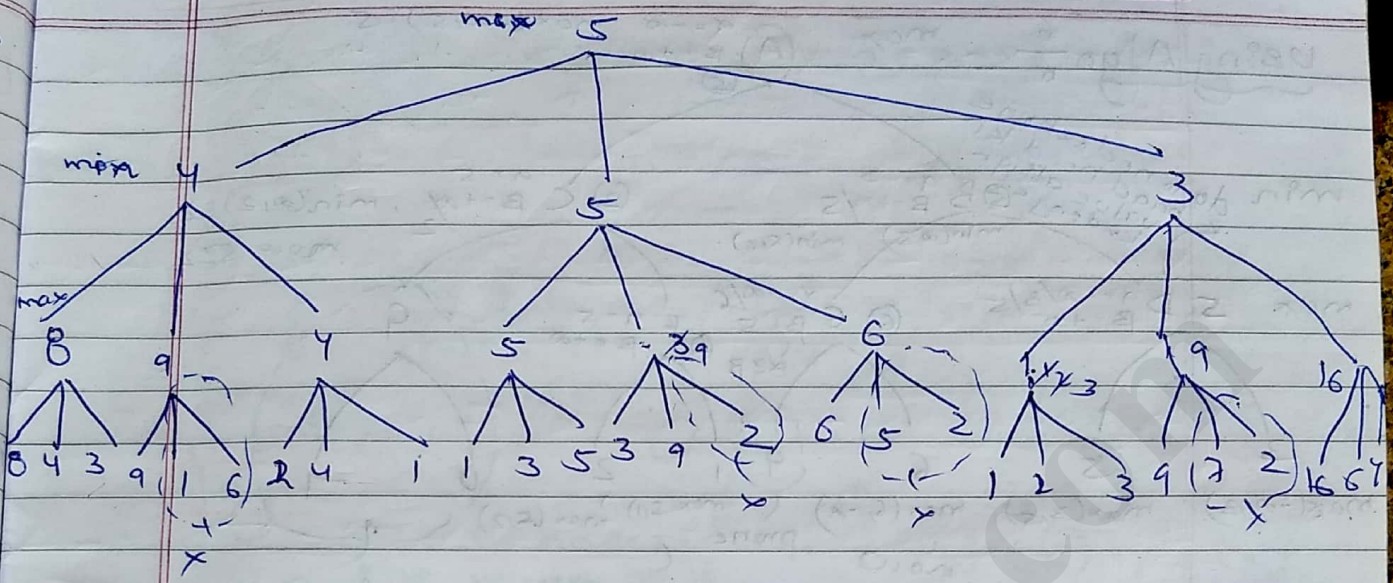
α-β pruning / cuts (optimization technique) used to optimize minimax

▲ - max
▼ - min





Date: / /
Page No.



Why α-β

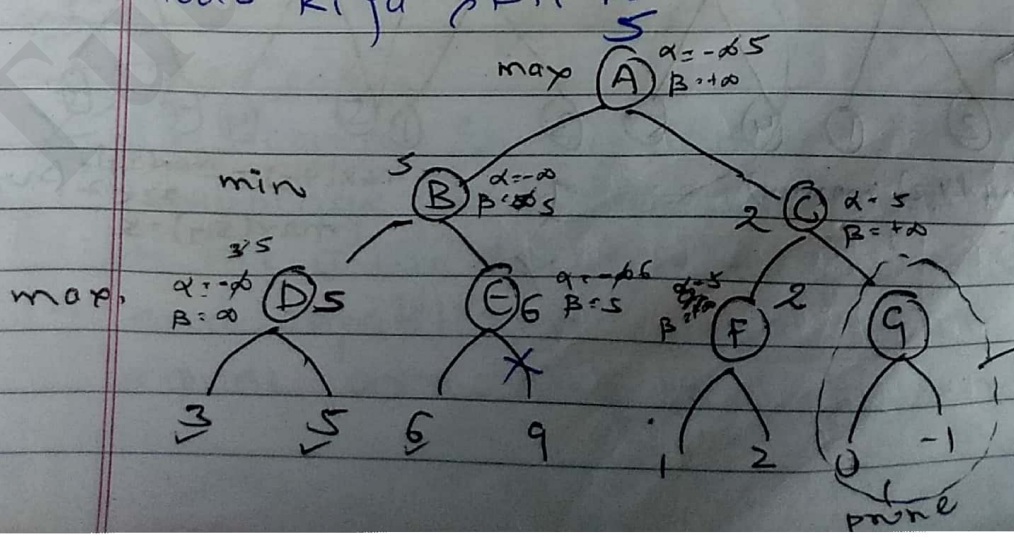
∵ we pokes two extra values in the process

- α- Alpha is the best value that he us a maximizer currently can Guaranty and that level or above
- β- β is the best value that minizer currently can guarantee at that level or above.

Condition :- We will prune only when α ≥ β

Initially -
α = -∞
β = +∞

α or β ki value niche ja skti hai upar kbi nhi, upr value skti node ki ja skti h

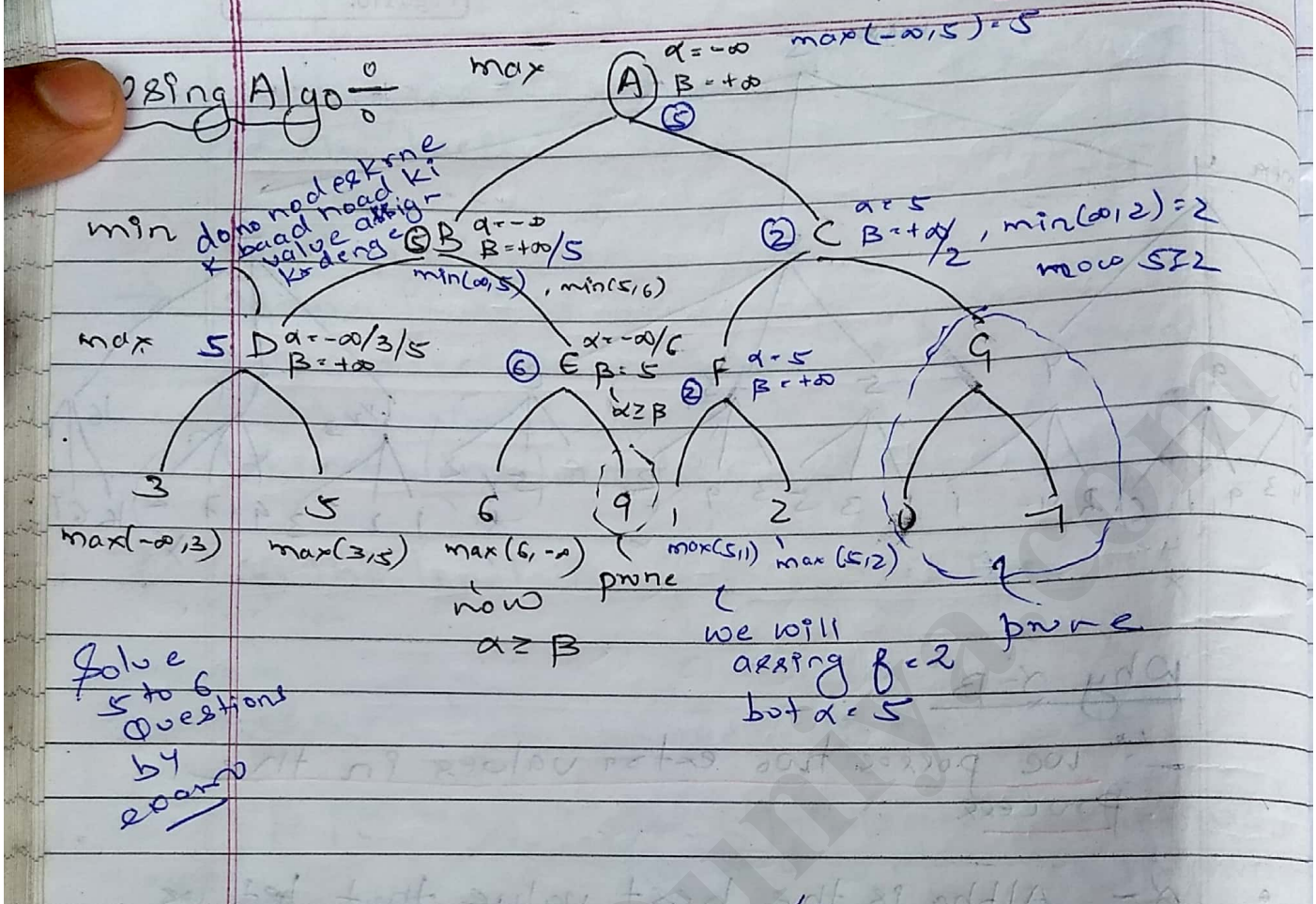


5b now level max pe range we will update α & at min level we will update β

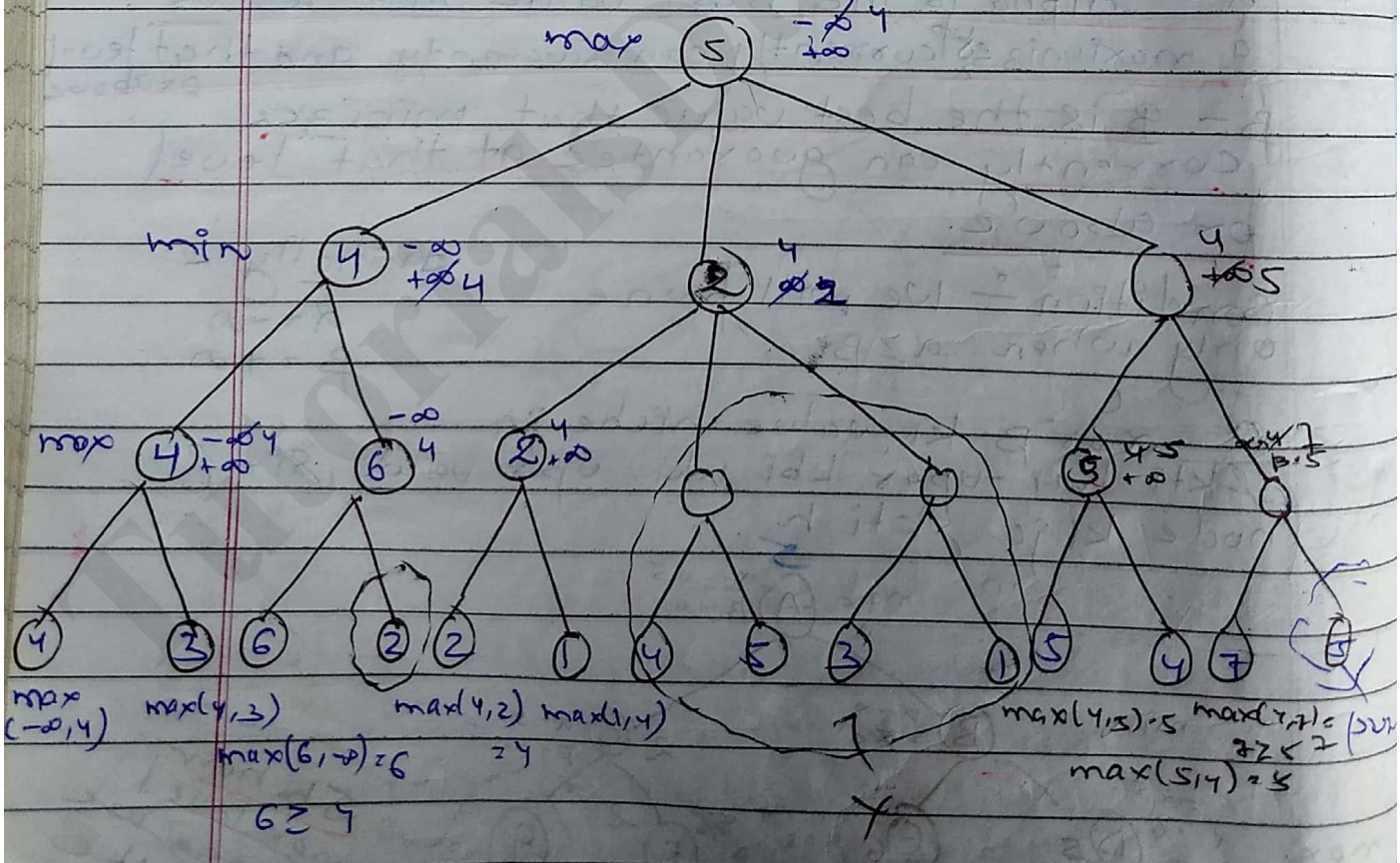
Date: / /
Page No.

46

Pruning Algo



Solve 5 to 6 Questions by yourself



Date: / /
Page No.

Knowledge Representation

Logic

means of formal reasoning

we use symbolic logic

FOPL
first order predicate logic

as one of the most imp knowledge representation method.

Propositional logic (atomic meaning)

Quantifier
$\forall(x)$, $\exists(x)$
(forall x) (there exist some)

There are 5 way for knowledge representation. FOPL is one of them

Ans

FOPL - imp for AI
(1) Sound theoritical foondation
(2) Flexible
(3) Widely Accepted in AI

Logic - formal method for reasoning

FOPL - Predicates, functions, variables, constant, logical connectives, quantifiers
(Syntax of FOPL)

syntax of FOPL

Predicates -
Predicates symbol denotes the relation or mappings from the elements of domain to the value twe or false. Capital letters and capatilized word such as P, Q, R, EQUAL are used to represent predicates.

Date: / /
Page No.

Functions - Functions denote the relation defined on a Domain D. They map n elements to a single element of the domain

Variables - Represented by small letters can hold variable values.

Quantifiers - $\forall(x)$ & $\exists(x)$

Logical connectives - $\vee, \&, \veebar, \rightarrow, \leftrightarrow$

(highest priority ⟶ lowest priority)

Well formed formula (WFF)
(pronounced as Woffs.

E1.)
E2.)
E3.)
E4.)

Soln

Example -

E1: All employees earning $1400 or more per year pay taxes
E2: Some employees are sick today
E3: No employees earn more than president

Soln -
E(x) for x is an employee
P(x) for x is on president
I(x) for the Income of x
GE(u,v) for u is greter than equal to v
S(x) for x is sick today
T(x) for x pays tax.

E2': $\exists x (E(x) \rightarrow S(x))$
E1': $\forall x [(E(x) \& GE(I(x), 1400)) \rightarrow T(x)]$
E3': $\forall y [E(x) \& P(y) \rightarrow \neg GE(I(x), I(y))]$

Date: / /
Page No.

Example -

E1 - All employees of AI software company are programmer

Soln - E(x) : Employee of AI Soft. company
P(x) : Programmer
$\forall x [E(x) \rightarrow P(x)]$

Example -

E1.) Every Natural no. is an integer no.
E2.) There exist no. that is even no.
E3.) For every no. x, there exist a no. y such that x < y - relationship
E4.) Every womens loves a child

Soln -
N(x) : Natural no.
I(x) : Integer no.
E(x) : Even no.
Q(x,y) : x is less than y
w(x) : women
C(x) : child
Love(x,y) : x loves y

E_1' : $\forall x (N(x) \rightarrow I(x))$
E_2' : $\exists x (E(x))$
E_3' : $\forall x \exists y (Q(x,y))$
E_4 : $\forall x \exists y (w(x) \& C(y) \rightarrow loves(x,y))$

N.B.

Example -
E1 - Every man is mortal
E2 - John is man therefore john is mortal

M(x) : Man is mortal
$\forall x (M(x))$

ex- 1) Every man is mortal
2) John is a man
3) Therefore John is mortal

E(x): x is man
M(x): x is mortal

S_1: $\forall x\,(E(x) \rightarrow M(x))$
S_2: $E(John)$
S_3: $E(John) \rightarrow M(John)$

#

Well formed formula

wff: An atomic formula is wff
If P & Q are wff then
$\neg P$, $P \wedge Q$, $P \vee Q$, $P \rightarrow Q$, $P \leftrightarrow Q$
$\forall x\,P(x)$, $\exists x\,P(x)$ is also wff

$\forall P\ P(x) \rightarrow Q(x)$
Man(nJohn)
} Not a well-formed formula

?

E: $\forall x\,((A(a,x) \vee B(f(x))) \wedge (C(x) \rightarrow D(x))$

a = 2
f(1) = 2 f(2) =

A(2,1) = true A(2,2) false
B(1) = true B(2) false
C(1) = true C(2) false
D(1) = false D(2) = true
D = {1,2}

Ex=1 ((A(2,1) ∨ B(f(1))) ∧ C(x)) → D(x)
true false true false

= false

Date: / /
Page No.

Ex: $\exists z\,((A(z,z) \vee B((f(z)) \wedge C(z)) \rightarrow D(z))$

false, true → true; false, true → true; true → false

= false

Prolog

$$fact(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times fact(n-1) & \text{otherwise} \end{cases}$$

fact(N, R)
fact(0, 1).
fact(N, R) :- N1 is N-1, fact(N1, R1), R is N * R1

To check

To execute: ? fact(5, R)

N1 = 4
fact(4, R1)
R1 = 5 * R1
N1 = 3
fact(3, R1)
R = 4 * R1
:
f(0, 1)

•) $$fib(n) = \begin{cases} 0 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$

fib(1, 1).
fib(2, 1).
fib(N, R) :- N1 is N-1, N2 is N-2, fib(N1, R1), fib(N2, R2), R is R1 + R2, write(R).

•) $$GCD(m, n) = \begin{cases} GCD(n, m) & \text{if } n > m \\ m & \text{if } n = 0 \\ GCD(n, m \bmod n) & \text{otherwise} \end{cases}$$

Date: / /
Page No.

(Ex) GCD

```
gcd(X,X,X).
gcd(X,Y,D) :- X<Y, gcd(Y,X,D).
gcd(X,Y,D) :- X>Y, Y1 is X-Y, gcd(Y,Y1,D).
```

Resolution

(principle in FOPL domain)

Facts: ≡≡≡ set of clauses

conclusion - A

Resolution is way to decide, whether these n facts result in conclusion A

Procedure

We have n statement & we have to prove a conclusion X. Now we add the negation of X into the n statement so we have n+1 statement. Now, if these n+1 statements are inconsistence, it means there is problem in negation of X, so conclusion X is correct.

disjunction - ∨ - or
conjuction - ∧ - and

Steps -
1. add negation of conclusion to KB
2. change all to clause form
3. Resolve
4. Basis of results

Add

clause form -
P(x) - literal
or
P(x) ∨ Q(x) - literal & disjunct
only these are allowed.

Example - Predicate (form)
1. dog(fido)
2. ∀x (dog(x) → animal(x))
3. ∀y (animal(y) → die(y))

Date: / /
Page No.

Conclusion – die(fido)

clause form of above –

1. ~~dog~~(fido)
2. ~~¬dog~~(x) ∨ ~~animal~~(x)
3. ~~¬animal~~(y) ∨ ~~die~~(y)

$P \to \varphi$
$\neg P \vee \varphi$

we will add one more i.e negation of conclusion

4. ~~¬die~~(fido)

Now because in predicate, we have all universal clause

Now, cut statement with its negation.

We are left with empty clause, so it's inconsistence. So our conclusion die(fido) was correct.

Example2 – clause form –

1. $(a \leftarrow b \wedge c) = \neg(b \wedge c) \vee a = \neg b \vee \neg c \vee a$
2. b
3. $(c \leftarrow d \wedge e) = \neg(d \wedge e) \vee c = \neg d \vee \neg e \vee c$
4. $e \vee f$
5. $(d \wedge \neg f)$

conclusion – a

$= b$
$= e \vee f$
$= \neg d \vee f$ X
$= \neg f$
$= d$
$= \neg a$
$(d \wedge \neg f)$

Now, Resolution

(b & c) $\neg a$ $\quad \neg b \vee \neg c \vee a$

$\neg b \vee \neg c$ $\quad b$ (1)

(2) $\neg d \vee \neg e \vee c$ $\quad \neg c$

$\neg d \vee \neg e$ $\quad e \vee f$ (3)

$\neg d \vee f$ $\quad d$

$(d \wedge \neg f)$ = true, so we write d & $\neg f$ as two clause so conclusion is correct

} Tableau

54

$\sim(\sim\alpha \vee \beta)$

Date: / /
Page No.

Semantic tableaux rules

R1- $\alpha \wedge \beta$
↳ α
↳ β
(one after other)

R2- $\sim(\alpha \wedge \beta)$
$\sim\alpha$ $\sim\beta$
(tree form)

R3- $\alpha \vee \beta$
α β

R4- $\sim(\alpha \vee \beta)$
↳ $\sim\alpha$
↳ $\sim\beta$

R5- $\sim\sim\alpha$
↳ α

R6- $\alpha \rightarrow \beta$
$\sim\alpha$ β

R7- $\sim(\alpha \rightarrow \beta)$
↳ α
↳ $\sim\beta$

R8- $\sim(\alpha \leftrightarrow \beta)$
$\alpha \wedge \sim\beta$ $\sim\alpha \wedge \beta$

R9- $\forall x\, \alpha[x]$
↳ $\alpha[t]$
for any ground term t, where t is a term which is free from variables.

R10- $\sim\{\forall x\, \alpha[x]\}$
↳ $\sim\alpha[c]$
where c is a constant not occuring in α

R11- $(\exists x) A[x, t]$
(x: any variable; t: ground term)
$A[g(f), t]$ where g is β^n of t.

R12-

R13-

Step1-
Step2-
Step3-
Step4-
Step5-

Date: / /
Page No.

R12 - $\frac{\exists x \ \alpha[x]}{\alpha[c]}$ for every new constant c.

R13 - $\frac{\neg[(\exists x)\alpha[x]]}{\neg\alpha(t)}$ for any ground term t.

Converting to clause form

Step1 - Eliminate $\rightarrow$, $a \rightarrow b = \neg a \vee b$

Step2 - Reduce the scope of each negation to a single term using

2.1 - $\neg(\neg p) = p$

2.2 - demorgan law $(a \cup b)' = a' \cap b'$, $(a \cap b)' = a' \cup b'$

2.3 - $\neg\forall(x)P(x)$ $\hookrightarrow \neg P(c)$ / $\exists x \neg P(x)$

2.4 - $\neg\exists x P(x)$ $\hookrightarrow \forall x \neg P(x)$

Step3 - Move all quantifiers to the left of the formula without changing their relative order.

Step4 - Eliminate existential quantifier by inserting skolem fn

Step5 - Convert the expression into a conjuction of disjuncts

Skolem fn in FOL

Every existentially quantified variable can be replaced by a unique skolem fn whose argument are all universally quantified variables on which existential depend.

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

Date: / /
Page No.

Example- EveryBody likes Something

likes (x,y)
Person(x).

Now, $\forall x \exists y [Person(x) \wedge likes(x,y)]$

Now, Skolemization

$\forall x [Person(x) \wedge likes(x, S(x))]$

where $S(x)$ = "that which x likes"

Example- Every philosopher writes atleast 1 book (some)

Philosopher - P(x)
write (x,y)

a

$\forall x [Roman(x) \wedge know(x, Marcus)] \rightarrow [hates(x, Ceasar)] \vee (\forall y : \exists z : hate(y,z) \rightarrow thinkCrazy(x,y))$

b

Soln- $\forall x$ in $[a] \rightarrow [b]$

$\forall x ([\neg roman(x) \vee \neg know(x, Marcus)] \rightarrow [b]$

Now,

$\forall x \forall y \exists z [\quad]$

$\forall x \forall y [a] \vee [hate(y, S(y)) \rightarrow thinkCrazy(x,y)]$

Now we can drop this

$[a] \vee [b]$

Date: / /
Page No.

final answer -

[¬Roman(x) V ¬Known(x, Marcus)] U
[hates(x, Ceasar) U ¬hate(y, S(y)) U
thinkcrazy(x, y)]

S(y) = "that y hates".

Now,

Resolution -

man(marcus)
pompeian(marcus)
¬pompeian(x1) V roman(x1).
ruler(Ceasar)
¬roman(x2) V loyalto(x2, Ceasar) V
hate(x2, Ceasar)
Loyalto(x3, f(x3))
¬man(x4) V ¬ruler(y1) U ¬tryassasinate
(x4, y1)
tryassinate(marcus, Ceasar) U ¬loyalto(x4, y1)

To proove - hate(marcus, Ceasar)
So we add.
¬hate(marcus, Ceasar)

Here we can unify x2 with
markus as all quantifiers are
universal.

Date: / /
Page No.

(4) ¬hate(marcus, ceaser) 5

marcus/x2

3 ¬roman(x2) V loyal to(x2, ceaser)

marcus/x1

2 ¬pompien(marcus) V loyal to(marcus, ceaser)

loyal to(marcus, ceaser) 7

x4/marcus
x1/ceaser

8 ¬man(marcus) V ¬ruler(ceaser) V ¬try assinate(marcus, ceaser)

¬man(marcus) V ¬ruler(ceaser) 4

1 ¬man(marcus)

empty

So, Inconsistence

proved is correct

Date: / /
Page No.

Example- (Lucky Student)

(1) Anyone passing his history exam & winning lottery is happy.
(2) Anyone who study or is lucky can pass all this example.
(3) John did not study but he is lucky.
(4) Anyone who is lucky wins the lottery

Conclusion- John is happy.

Done

Example- (Exciting life)

(1) All people who are not poor and are smart are happy.
(2) Those people who reads are not stupid
(3) John can read & his wealthy
(4) Happy people have exciting life

Conclotion- Can anyone be found with an exciting life

- Prolog
  - Hadai

- Rules -
  - $\sim(\forall x) F(x) \equiv \exists x (\sim F(x))$
  - $\sim(\exists x) F(x) \equiv \forall x (\sim F(x))$

  Help while conversion to clausal form

- CNF, DNF, PNF
  - (N)

Date: / /
Page No.

Structured Knowledge

profession (bob, professor)
faculty (bob, engineering)
own (bob, house)
drive (bob, toyota)

we did this till now.

Defining bob in linear set of clauses.

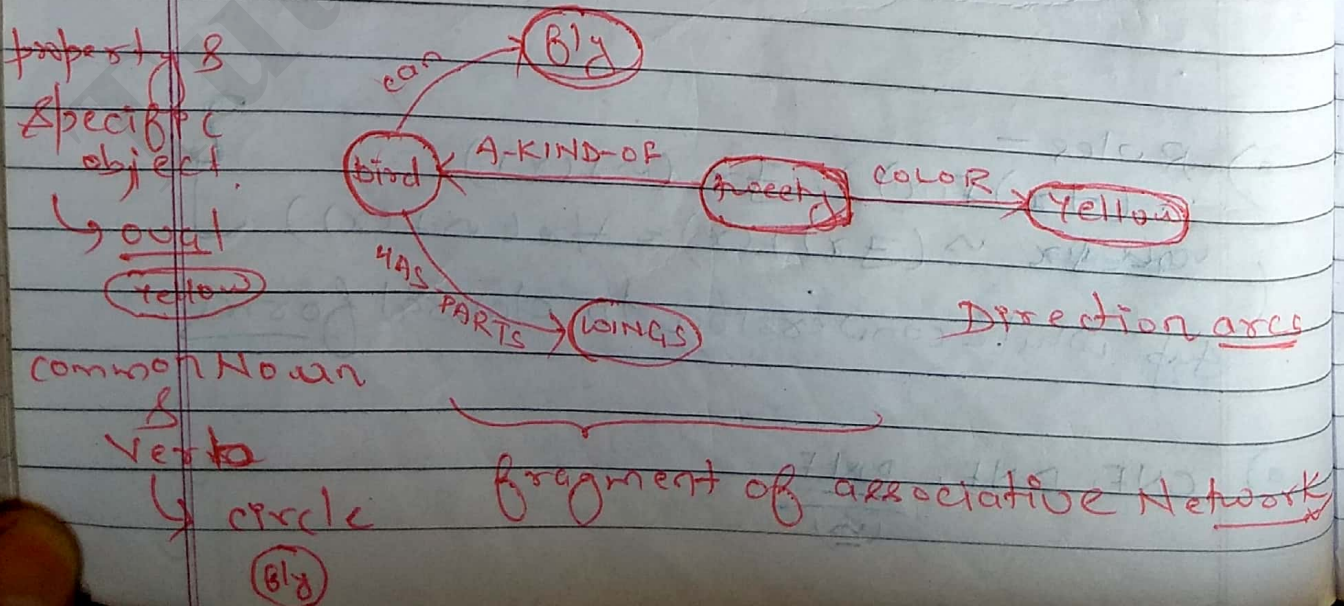
↓

It will not work for large no. of clauses (difficult to represent)

- **Associated Networks** —

3 This provides a mean of structuring and exibiting the knowledge in a different structure.

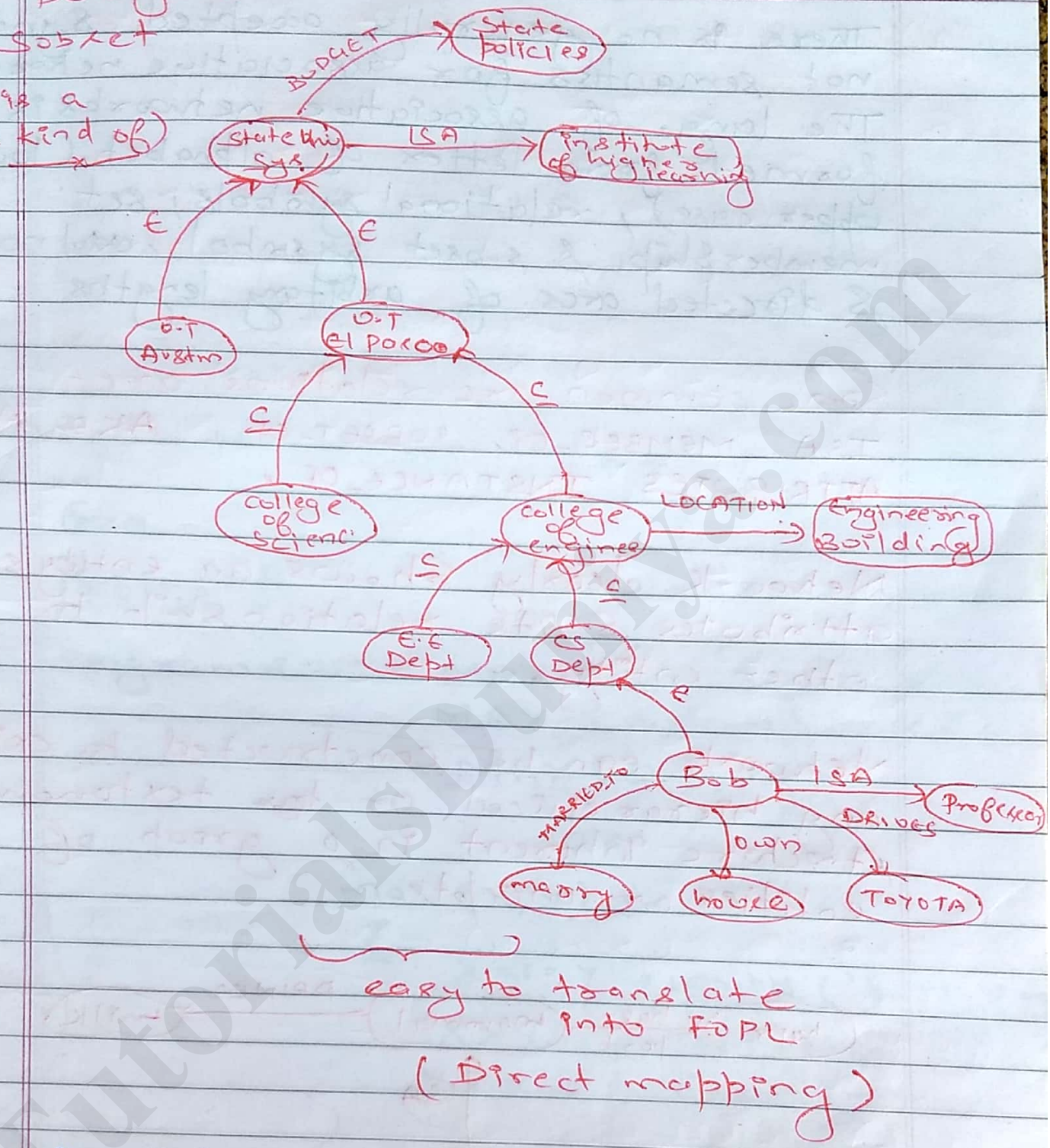
3 Pieces of knowledge are clustered together into symantic groups



Date: / /
Page No.

∈ : belongs to
⊆ : Subset

ISA - is a
AKO - (a kind of)



easy to translate into FOPL
(Direct mapping)

Associative networks are directed graphs with labeled nodes and arcs or arrows. The language used in constructing a network is based on selected domain primitive for objects & relations as well as some general primitives.

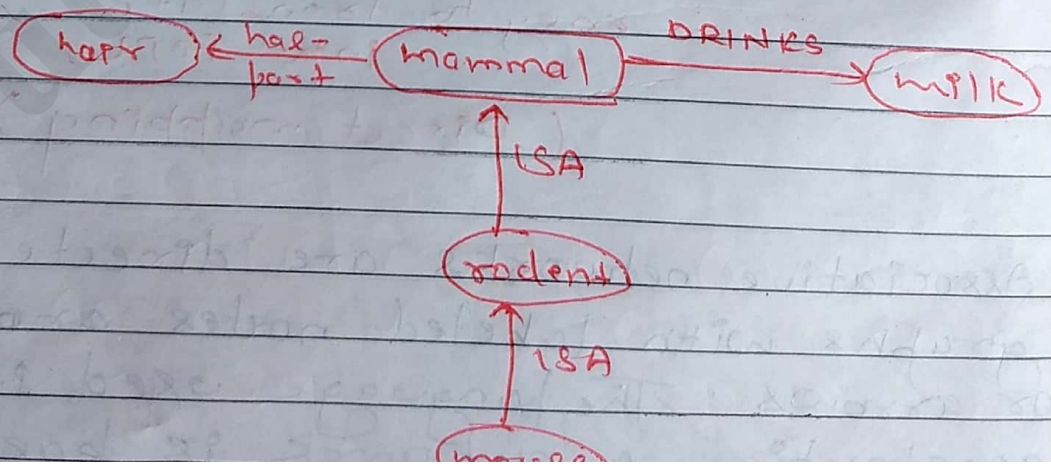
Date: / /
Page No.

There is no generally accepted syntax not semantics for associative network. The lang. of associative networks is formed from letter of alphabet (lower & upper case), relational symbols, set membership & subset symbol, nodal nodes & directed arcs of arbitary lengths.

Some common arc relations are - ISA, MEMBER_OF, SUBSET-OF, AKO, AGENT ATTRIBUTES, INSTANCE_OF.

Network clearly shows an entity's attributes & its relationship to other entity.

Network can be constructed to exibit any hierarchical or ~~tax~~ textonomic structure inherent in a group of entities or concepts.



Date: / /
Page No.

Association network permits the implementation of inheritance. Nodes which are members or subset of others nodes may inherit properties from their higher level ancestor nodes.

b) The assumption is made that unless there is an info. to the contrary, it is reasonable for an entity to inherit characteristics from its ancestors node.

Eg

Prolog ⁰/₀

Program 2-

```
MAX(X,Y,X) :- X>Y
MAX(X,Y,Y) :- Y>X
MAX(5,3,M).
```

or

```
max(X,Y) :-
{
    X=:=Y -> write("both are equal");
    X>Y -> write(X);
    Y>X -> write(Y);
}
```

3 ⌐
4 ⌐ } To check
5 ⌐

Date: / /
Page No.

(7) Mult (N1, N2, R) :- R is N1 * N2.

(6) Power of 2^n :-

Pow(0, N, 0) :- N > 0

Pow(X, 0, 1) :- X > 0

Pow(X, 1, X)

Pow(X, N, Ans) :- N1 is N-1, Power(X, N1, A1), V is A1 * X

Pow(2, 3, V)

N = 2

Pow(2, 2, A1)

A1 = 2

Pow(2, 1, A1)

V = 2 * 2

V = 4 * 2 = 8

9) Pow(X, N, V) :- V is X * Pow(X, N-1, V1)

10) ✓

11) conc(L1, L2, L3)

conc([], L, L)

conc([X|L], M, [X|N]) :-

conc(L, M, N)

Sub

Conc ([1|2,3] [a,b,c] [_1_])

↳ con ([2|3], [a,b,c] , [1|D])
X L M X | N
in next step
[2|2, *1]

Q1) (i) Length of a list [even or odd]

length ([ ], 0).
length ([H|T], L) :- length (T, L1),
L is L1 + 1.

Dry run –

length ([a, b, c, d, e], L) :-
L = 4+1 = 5 length ([b, c, d, e], L1)
L1 = 3+1 = 4
L2 = 2+1 = 3 :- length ([c, d, e], L2)
L3 = 1+1 = 2 :- length ([d, e], L3)
L4 = 1+0 = 1 :- length ([e], L4)
:- length ([ ], 0)

for even – even(L) :- 0 is mod(L,2)
odd (L) :- 1 is mod (L,2)

(ii) Sum of numbers of list

sum ([ ], 0).
sum ([H|T], L) :- sum (T, L1), L is L1 + H.

Date: / /
Page No.

6) Reversing a list

Theory

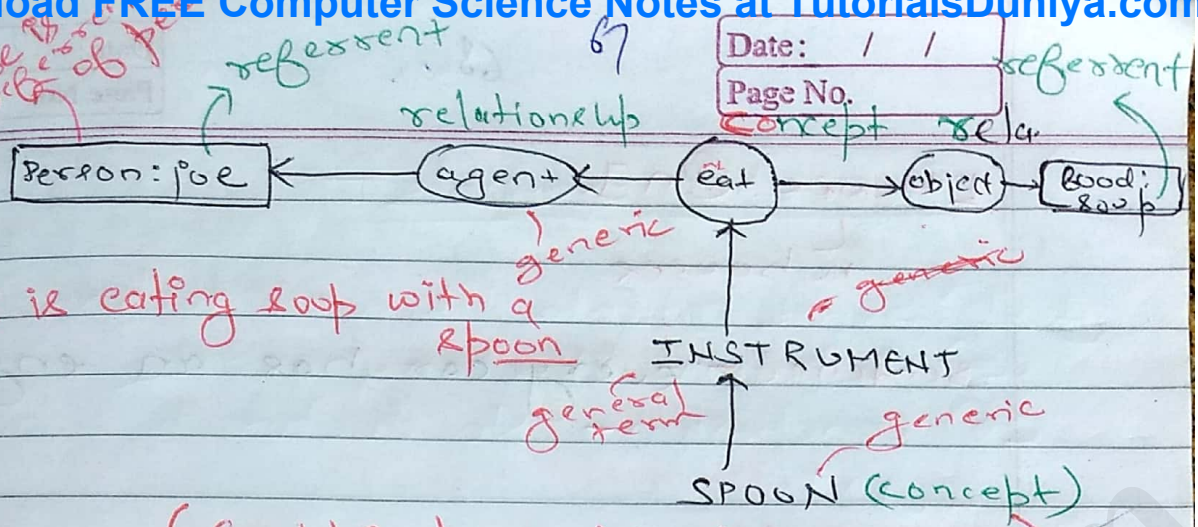
Conceptual Graphs

first building block for associative network

For each state we will have conceptual graph and all these conceptual graphs will make the associative network.

CG & FOPL
↓
logical systems.

Conceptual graph is a primitive building block for associative network. It is a graphical portrail of a mental perception which consist basic concepts & the relationship that exist b/w the concepts. It is roughly equivalent to a graphical diagram of a natural lang. sentence where the words are depicted as concepts & relationships



(Graphical conceptual graph)

[Person, joe] ← (Agent) ← [EAT]

(object)

[Food: Soup]

(INSTRUMENT) → [Spoon]

(Textual conceptual graph)

Symbols of a conceptual graph (concept symbols) refers to entities, actions, properties or events. A concept may be individual or generic. Individual concepts have a type field followed by a referrent field eg. [Person] & [Food].

Eat & spoon have no referent fields since they are generic concepts. In the textual eg square brackets have replaced concept boxes and parenthesis have replaced relation circle.

[House : ?]

which house

[House : #432]

House numbered 432

[House : *x]

which house

any house or some house

Date: / /
Page No.

[House: @n]
↳ house

sentence - every car has an engine

(1) FOPL sentence -
1) It is hot - P
2) It is humid - Q
3) It is rain - R

a) If it is hot and humid then it will rain (P ∧ Q → R)

b) If it is humid then it is hot (Q → P)

c) It is humid now. (Q)

Prolog :-

- Reverse a list

essentially the result

reverse([], Z, Z).
reverse([H|T], Z, Acc) :- reverse(T, Z, [H|Acc])

Dry Run :-

[a|b,c,d] → reverse([b,c,d], R, [a])
[b|c,d] → reverse([c,d], R, [b,a])
[c|d] → reverse([d], R, [c,b,a])
[d] → reverse([], R, [d,c,b,a])

Related call base case
reverse([], Z, [d,c,b,a])
Z = [d,c,b,a] → end

Q $P \rightarrow Q$ If P then Q

Date: / /
Page No.

Theory

Sentence -

1) Some Horses are gentle & have been well trained.

Soln -

H(x) - Horse
G(x) - Gentle
T(x) - Trained

$\exists x (h(x) \wedge g(x) \wedge t(x))$

2) Some horses are gentle only if they have been well trained

Soln - $\exists x [H(x) \& (G(x) \rightarrow T(x))]$

3) Some horses are gentle if they have been well trained

Soln - $\exists x [H(x) \& (t(x) \rightarrow g(x))$

4) All well trained horses are gentle:

Soln - $\forall x (h(x) \& t(x)) \rightarrow g(x)$

5) All gentle horses are well trained.

Soln - $\forall x (h(x) \& g(x)) \rightarrow t(x)$

6) Any Horse is gentle that has been well trained

Soln - $\forall x [h(x) \& (t(x) \rightarrow g(x))$

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

Date: / /
Page No.

TMS

(True maintainance System)

KB - Knowledge Base
IE - Inference engine

till now we talk abt these

$$\frac{P \rightarrow \varphi \quad P}{\varphi}$$

TMS maintain the current belief system

(it do it in the form of structure CSL)

[~~Support~~ list]

- TMS maintain consistency of the knowledge.
- Doesn't perform any inference fn
- It free's the problem solver from any concerns of consistency
- set of beliefs available to the problem solver will be current & consistence
- TMS maintains the currently active belief set.
- Belief revision is the job of TMS.
- TMS maintains dependency records for all conclusion.
- A belief would be removed from the current belief set by making appropriate updates to the dependency record & not by erasing the belief.
- Each preposition or statement having atleast one valid justification is made

Date: / /
Page No.

a part of current belief set.

- Statements lacking acceptable justification are excluded from current belief set.

This process is called Dependency directed backtracking

Now if we have,

$P \rightarrow Q$
P
Q

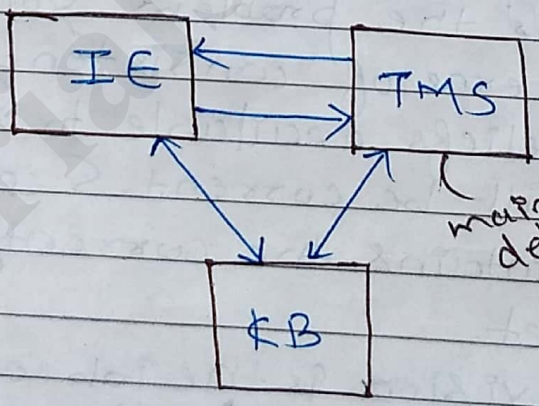
$Q \rightarrow \in$
$\in \rightarrow f$

P is not true (inconsistency)

$P \rightarrow Q$
$\neg P$
Q

Q is also not true so is $\in$ & f

we will not remove these dependency from KB. but are not in the current belief system



- How to maintain dependency network

Dependency N/W

↳ Support list

(SL <inlist>, <outlist>)

Date: / /
Page No.

This is an Example of Table

Node	Status	Meaning	Support list	Comment
n1	IN	cybil is bird	(SL () ())	a premise
n2	OUT	cybil can fly	SL (n1(), n3())	unjustified belief
n3	IN	cybil can't fly	SL (n5() n4())	justified belief
n4	OUT	cybil has wing	(SL () ())	retracted premise
n5	IN	cybil is an ostrich	(SL () ())	a premise

A node can be 1) premise (which is always true,
2) Assumption 3) Datum (single piece of data) 4) Justification

IN - part of current belief system
OUT - Not in " "

But they are in KB

Support list for premise is empty.

Retracted premise - Earlier it was premise, but it is not.

Node n1 & n5 are premises with empty support list since they do not require justification.

For n2 - SL(n1(), n3())

n1() because node n2 was in because of n1 (bird → fly)

n3 kiunke ye node n2 out hua, n3 = P, n2 = ~P. Since n3 = P is IN. So n2 which is ~P must be out (can't fly & fly IN stand be out)

Node n2, the belief that cybil can fly is out because n3 (A valid node - IN) is in outlist of n2.

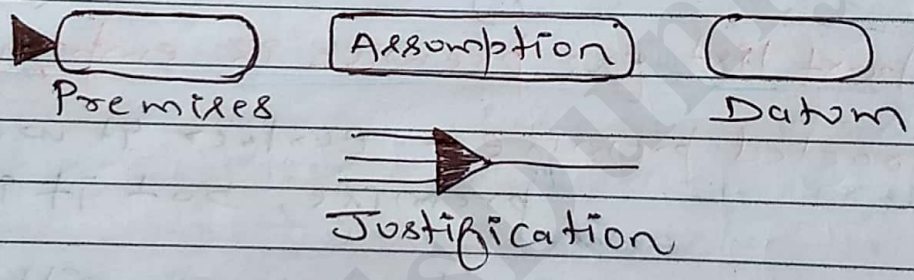
example- soppose it is discovered the cybil is not an ostrich i.e n5 is no longer valid, ie it will be OUT so n3 will be out & therefore n2 will be IN.

(n5 will be retracted)

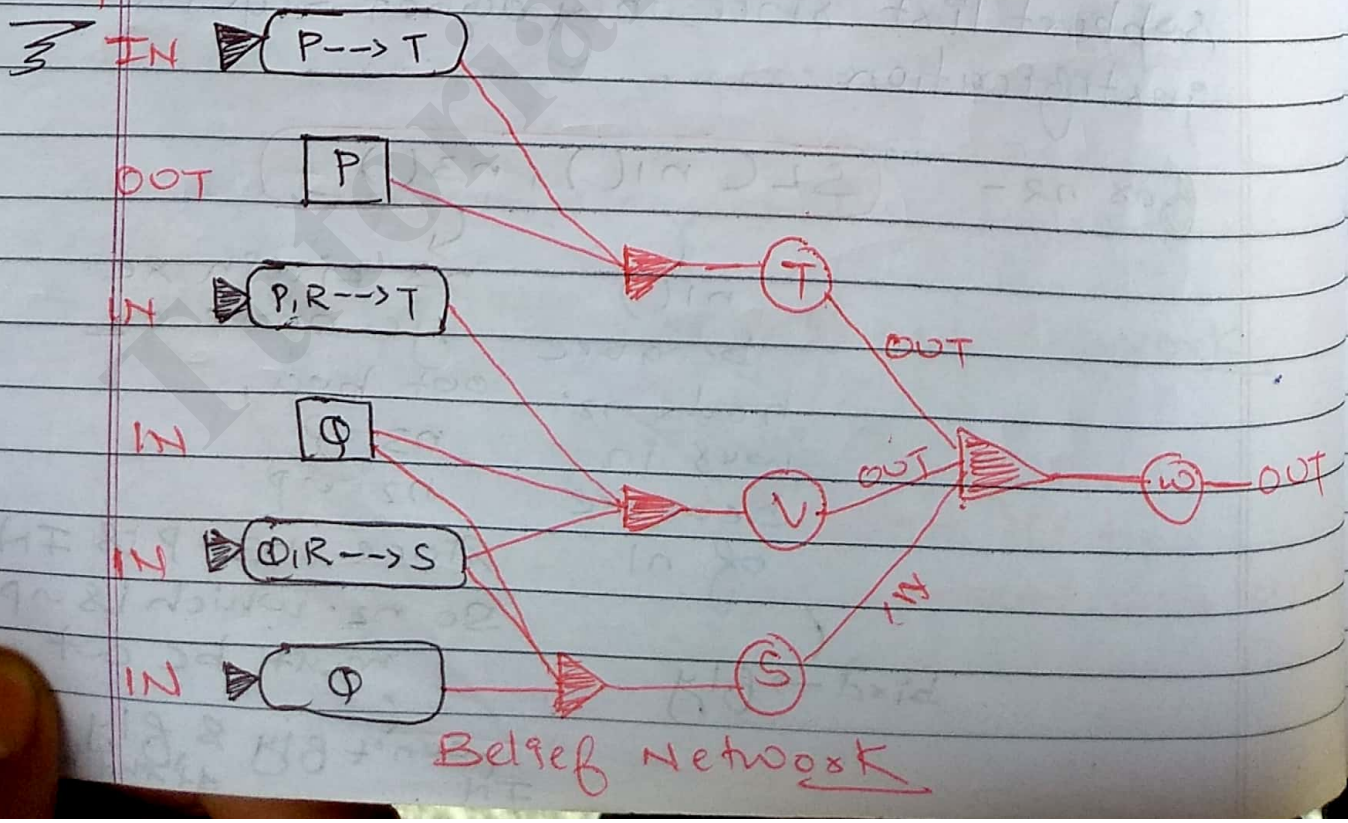
making its status OUT.

n2 will be then justified belief.

•)



Example



Date: / /
Page No.

Scheduling Aircraft

n1 IN type(aircraft) = 377 (SLC) n2( )
n2 OUT type (aircraft) = L400
n3 IN class (crew) = A S2 ((nB1- - -n22) ( ))

(Description)

If n1 is OUT, then we will make n2 IN and according n3 will change

Default Reasoning—

Another form of non-monotonic reasoning. It diminates the need to explicitly store all fact regarding a situation. A Default is expressed as

$$\frac{a(x) : M\,b_1(x) \ldots M\,b_k(x)}{c(x)}$$

where a(x) is a precondition wff
C(x): conclusion wff
M is a consistency operator
$b_i(x)$ are conditions, each of which must be separately consistent for C(x) to hold.

eg:- $\frac{ADULT(x) : M\ DRIVE(x)}{DRIVE(x)}$

If x is an adult then it is true to assume that x can drive, that infer that x can drive.

But later if u learn that x has suffered from blackouts you would be forced to revise or revoke your beliefs

Date: / /
Page No.

Scheduling Aircraft

n1 IN type(aircraft) = 377 (SL() n2())
n2 OUT type(aircraft) = L400
n3 IN class(crew) = A S2((n8,- - -n22)
(1))

(Description)

If n1 is OUT, then we will make n2 IN and according n3 will change

Default Reasoning –

Another form of non-monotonic reasoning. It diminates the need to explicitly store all fact regarding a situation. A Default is expressed as

$$\frac{a(x) : M\,b_1(x) \ldots M\,b_k(x)}{c(x)}$$

where a(x) is a precondition wff
c(x): conclusion wff
M is a consistency operator
$b_i(x)$ are conditions, each of which must be separately consistent for c(x) to hold.

eg- $\frac{ADULT(x) : M\ DRIVE(x)}{DRIVE(x)}$

If x is an adult then it is true to assume that x can drive, that infer that x can drive.

But later if u learn that x has suffered from blackouts you would be forced to revise or revoke your beliefs

Date:
Page No.

Example 2- ① $\frac{BIRD(x) : \neg FLY(x)}{FLY(x)}$

② BIRD (tweety)

(So we have not written FLY(Tweety) but from the above two statement it is clear that tweety can fly

But later we came to know,

OSTRICH(Tweety)

OSTRICH(x) → ¬fly(x)

(we have to revoke our belief

Close world Assumption

- Another form of assumption with regard to incomplete knowledge. Useful in application where most of the facts are known.
- Reasonable to assume that if a preposition cannot be proven, it is false. This means that in KB, if a ground literal P(a) is not provable Negation of P(a) is assumed to hold

eg- flight

(from book

Date: / /
Page No.

Bayesian Probabilistic (6-1, 6-2, 6-2)

$$P(H/E) = \frac{P(H \cap E)}{P(E)}$$ ← conditional probability

Bayes theorem

$$P(D1/E) = \frac{P(E/D1)\,P(D1)}{P(E)}$$

0.95, 0.05, 0.15

Do it by your own

6-3 - Possible world Representation

Prolog -

P16 -

finding a element in list / respective position

bind([], N, x)

element, position

empty

Date: / /
Page No.

Prolog

1) length(list, N).
evenlist(list) :- length(list, N), N mod 2 = 0
oddlist(list) :- length(list, N), N mod 2 = 1

2) Find the element on list

find([], N) :- write('No such element').
find([H|T], 1) :- write(H).
find([H|T], N) :- N1 is N-1, find(T, N1).

3) Delete an element from list

del(Y, [Y], []) || base case
del(X, [X|T], [T]) ||
del(X, [Y|list], [Y|list1]) :- del(X, list, list1).

?- del(ele, List1, List2).

?- del(3, [1|2,3,4,5], L1)
↳ del(3, [2|3,4,5], L1) → 1, 2, 4, 5
↳ del(3, [3|4,5], L1) → 2 4 5
→ L1 = 4 5

Date: / /
Page No.

Program to check sublist

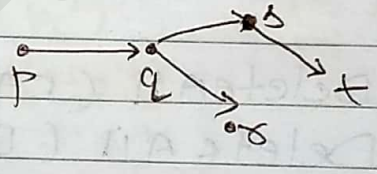
```
sublist([],[]).
sublist([H|T1],[H|T2]) :- sublist(T1,T2)
sublist([_|T1],[T2]) :- sublist([T1],[T2])
```

Program No 9

Path on a cyclic directed graph

```
Path(A,B) :- edge(A,B).
Path(A,B) :- edge(A,X), Path(X,B)
```

```
edge(p,q)
edge(q,r)
edge(q,s)
edge(s,t).
```



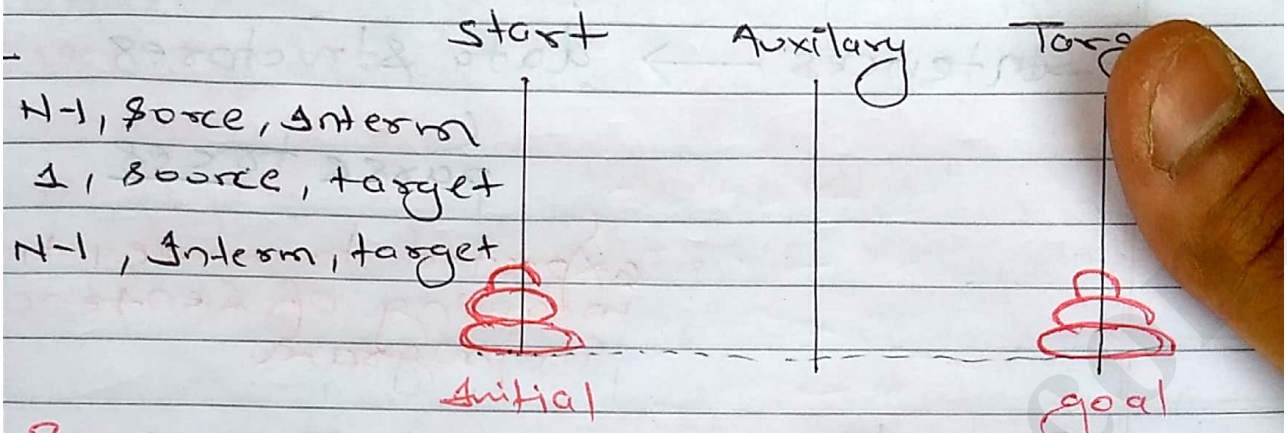
Palindrom

```
Palindrom(list1) :- reverse(list1,list2),
                    compare(list1,list2).

compare([],[]) :- write('list1 is palindrome').

compare([X|list1],[X|list2]) :- compare(list1,list2).

compare([X|list1],[Y|list2]) :- write('not palindrome').
```

6) Merging two ordered list into a single ordered list.

Date: / /
Page No.

Tower of hanoi



Program—

```
move(1,A,C,_):- write('moving top disk from'),
                write('A')
                write('B')
                write('C').

move(N,A,C,B) :- N1 is N-1,
                 move(N1,A,B,C),
                 move(1,A,C,_),
                 move(N1,B,C,A).
```

Date: / /
Page No.

1) NLP as an Application of AI

Sentences → data structures
(parse trees)
expected to convey the meaning of sentences in program.

Linguistic
(we don't need it when we talk about NLP.

Phrase
(A part of a sentence
(act as a single unit

2) Levels of component form of Knowledge
1. Phonological
2. Morphological
(friend + ly = friendly
this type of Knowledge
3. Syntactically
4. Symantical
5. Pragmatic

we need to parse the sentence to determine the structure. (wether it is grammitically correct)

NLU - Natural language understanding

Date: / /
Page No.

General approaches for natural language understanding

1) Use of keyword in pattern matching
2) Combined syntactic & semantic directed analysis (done in toc)
3) Comparing and matching the i/p to the real world situations
(eg - frame & script)

Disadvantage of (1) is that no knowledge structure is created.
Disadvantage of step (3) is that a substantial amount of specific knowledge should be stored. & Advantage is that computation is less cox we are not generating parse tree.
Disadvantage of step (2) is that it is computationally intensive.

Grammer —

terminal point - decomposable
starting point

$G = \langle V_n, V_t, S, P \rangle$

V_n - nonterminal point (can be decomposed)
P - production rules

Alphabets - $V_n + V_T + \epsilon$ (empty string)

Date: / /
Page No.

Φ_N = { S, NP, N, VP, V, ART }

Φ_T = { boy, popsicle, frog, ate, slapped, blew, the, a }

P: S → NP, VP
NP → ART N
VP → V NP
N → boy | popsicle | frog
V → ate | slapped | blew
ART → the | a .

The boy ate a popsicle. correct
The popsicle blew a frog
grammatically correct but not symantically

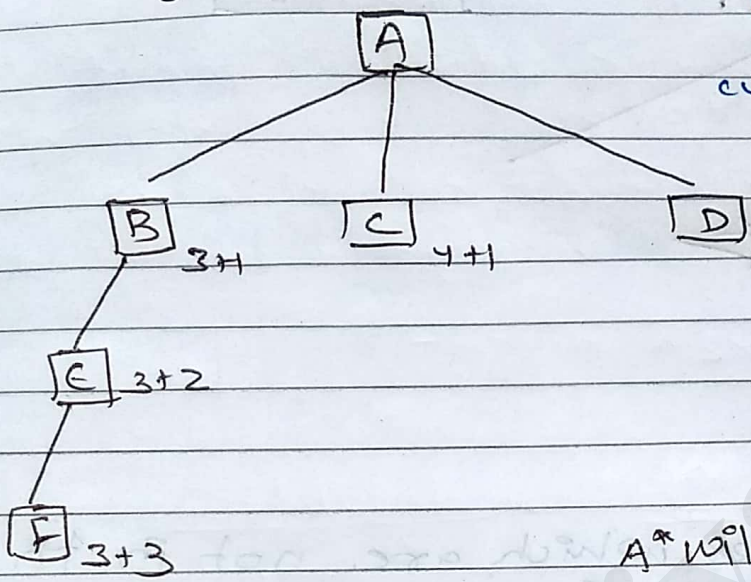
Date: / /
Page No.

Last class

A* Algorithm

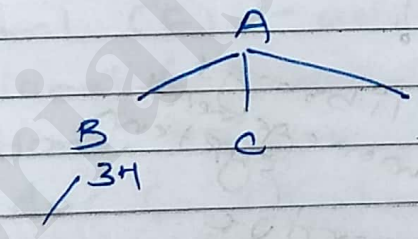
$f = h' + g$

h' → estimation
g → initial-current
current to goal



A* will converge when, h' is perfect or underestimate

Underestimate means hmne h' ki value ko km estimate kiya hai.



overestimated, actually it is 2+1

we converge but not optimally

If we actually have followed path of D, we would have reach the goal state first

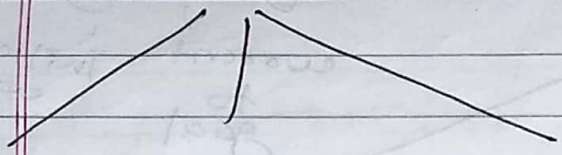
Date: / /
Page No.

Start

3	7	6
5	1	2
4	□	8

Goal

5	3	6
7	□	2
4	1	8



Book

Heuristic

No. of tiles which are not in their goal position & g is depth of node in search space.

Prolog

Cut

(System predicate

—!

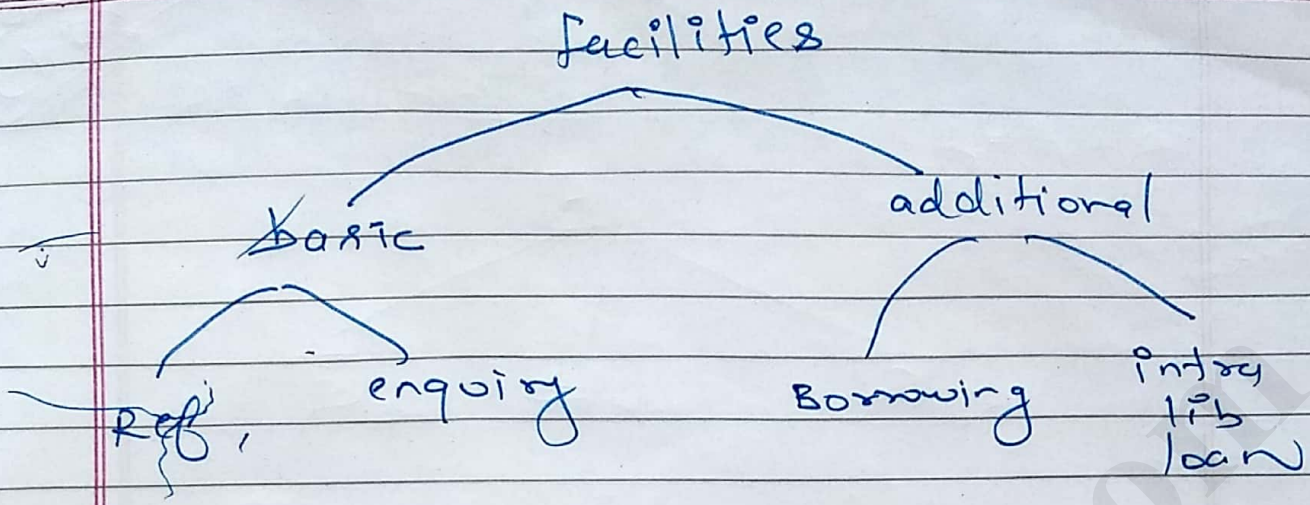
Bor library management system

overdue

(If it gets one overdue, it will not check for other overdue books

It just cont'n backtracking

Date: / /
Page No.




```
List(U,F) :- user(U), facility(U,F)
facility(U,F) :- overdue(U,B)! basic(F)
facility(U,F) :- general)
basic('reference')
basic('enquiry')
general(F) :- basic(F)
general(F) :- additional(F)
additional('borrowing').
additional('inter lib loan')
overdue('SK das', logic)
user('SK das')
user('Rajan').
```

6) John like everything except snakes

```
like(John, Snak) :- !, fail.
like(John, X).
```

more to study

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well