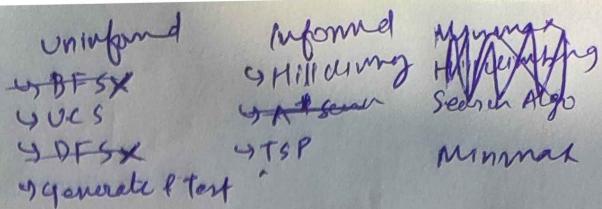


U-2

A!



12m

① Uniformed cost search:

→ Uniformed search: that does not contain any domain knowledge.

: operates on brute force way

: search tree is searched without any info about search space.

: known as blind search.

→ Uniformed cost search

* used for weighted Tree / Graph Traversal

* Goal is to find the path to goal-node with lowest cost or optimal path.

* It is based on path cost.

* Priority Queue is used for implementation
 ↳ High priority : minimum cost.

* Backtracking is there to avoid sticking.

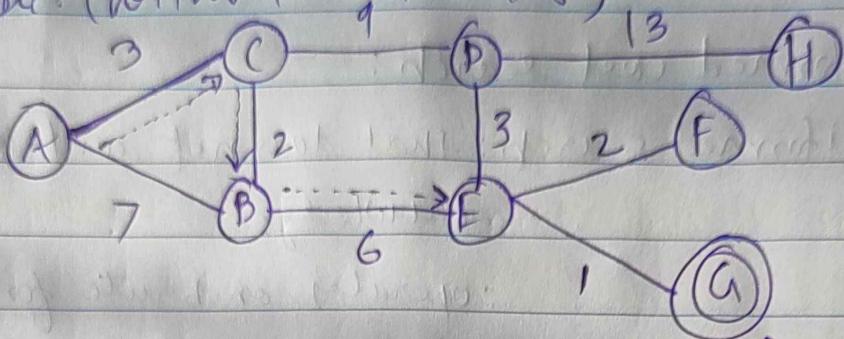
→ Advantages:

- (i) Optimal soln (the path chosen is least in cost)
- (ii) Completeness (can find you the solution)
- (iii) Flexibility (used in wide range)

→ Disadvantages:

- (i) Time complexity (slow & inefficient)
- (ii) Space complexity (requires large space)
- (iii) can stuck in infinite loop

→ Example: (without Backtracking)



Step 1:

Take the starting node as A.

Step 2

At Node A, $AC = 3$ and $AB = 7$, using priority queue, we can select the path with minimal cost. So, AC is chosen.

Step 3

At Node C, CA is already visited, $CD = 9$, $CB = 2$, using priority queue, we can select the path with minimal cost. So, CB is chosen.

Step 4

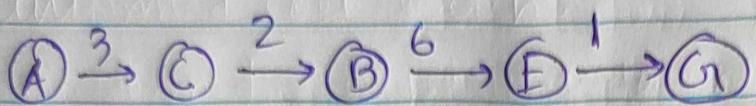
Similarly, BC is already visited. Among $BA = 7$ and $BE = 6$, BE will be chosen.

Step 5

At E, ~~EB~~ EB is already visited. $ED = 3$, $EF = 2$, $EG = 1$. EG will be chosen.

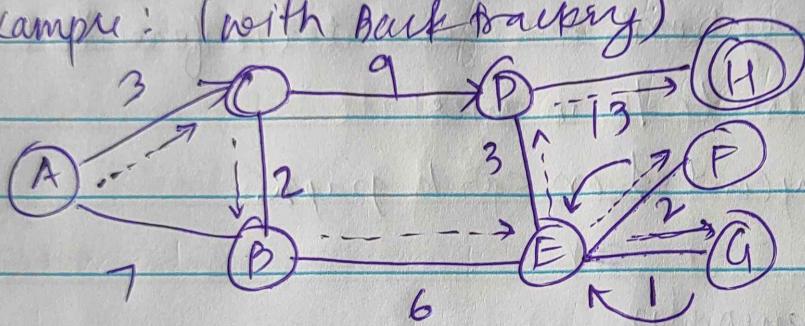
Since EG is goal node it will be terminated here.

→ Path to Goal node (G)



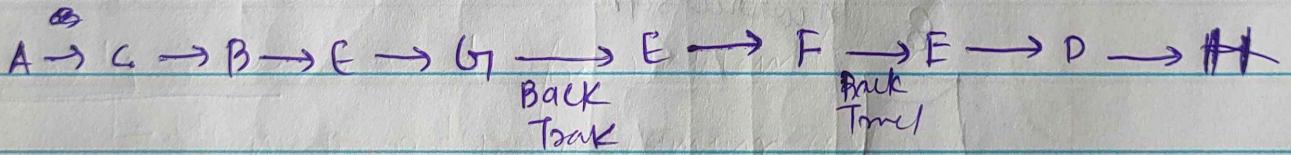
$$\text{Cost} = 3 + 2 + 6 + 1 = 12$$

→ Example: (with Backtracking)



Path will be: A → C → B → E → G {we haven't reached the goal node}

Now:



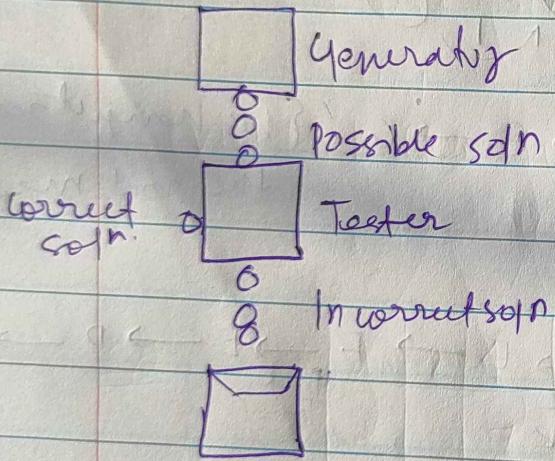
$$\text{Cost} = 3 + 2 + 6 + 1 + 1 + 2 + 2 + 3 + 13$$

$$= 33$$

Informed search

② Generate and Test

- ↳ A simple algo.
- ↳ guarantees to find a solution if done systematically and there exists a solution.
- ↳ It is a depth first search procedure
- ↳ Possible solutions are generated before test.
- ↳ Can be implemented in tree
- ↳ can be implemented on search graph



S1 Generate a possible soln.

S2 Test the soln

S3 If solution found THEN Done

ELSE return to step 1

③ Hill climbing search

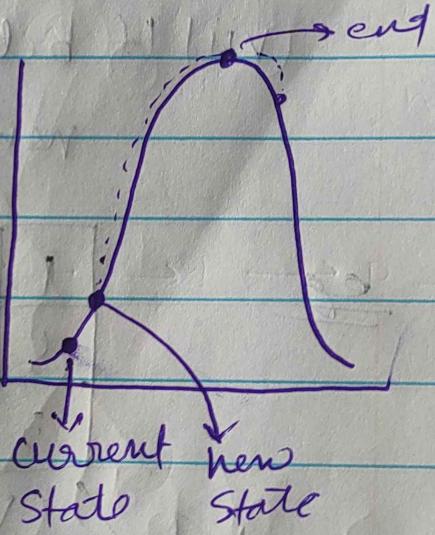
- ↳ Local search Algo (local domain)
- ↳ uses greedy approach (if best solⁿ avail., it continues)
- ↳ No Backtrack (no best move, it won't backtrack)
- ↳ Type of generate and test method.
- ↳ The feedback from the testing method is used to decide which direction to move in search space
- ↳ It is like DFS
- ↳ Always moves in a single direction

e.g. → there's a hill given

→ Have to find the best pos.

→ best pos. means moving upwards from current pos.

→ always checks the previous state after moving to next state.



→ If new state is better than current state.

 new state = current state

→ If new state is not better than current state

 hill climbing will stop if
 search is completed

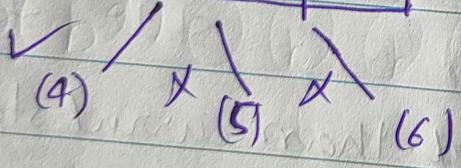
Algorithm:

- (i) Evaluate initial state
- (ii) If initial state = goal state
found the solution
stop
- (iii) If ~~not~~, current state = initial state
- (iv) Apply operator and get new state
- (v) Evaluate the new state
- (vi) If new state = goal
found soln.
stop
- (vii) If better than current state, then
new state = current state

eg

1	2	4
5		7
3	6	8

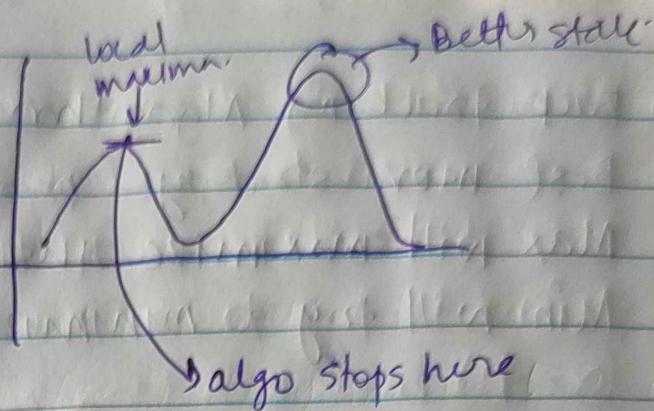
(start state)



We have to choose the least heuristic value and discard the remaining values.

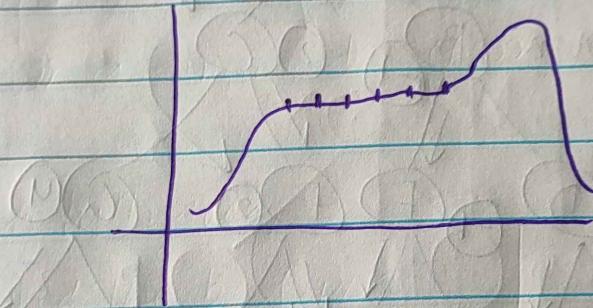
4. Limitations:

(1) Local Maxima:



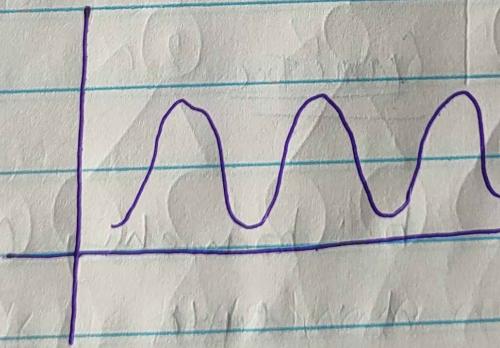
don't have idea about global ~~state~~ maximum

(2) Plateau:



algo get same value for some time and algo stops

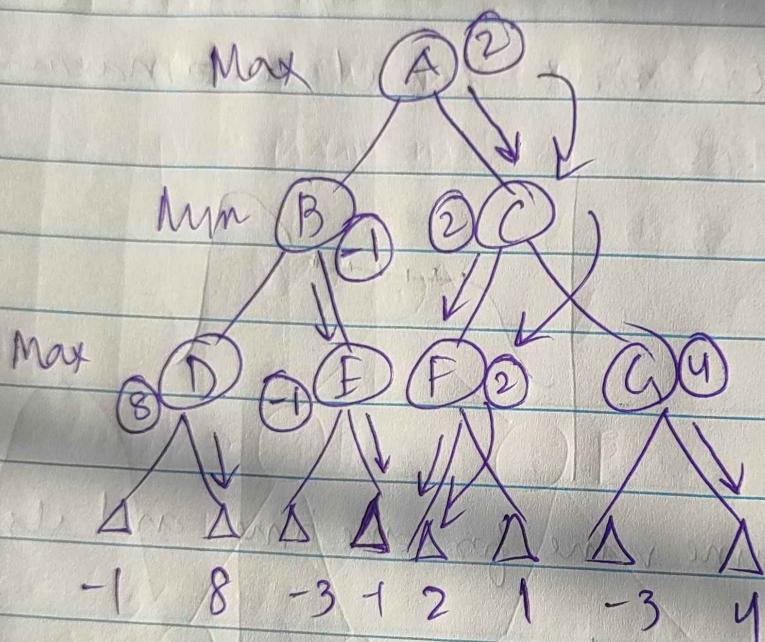
(3) ~~Ridge~~ Ridge:



all successor / peaks having same values

④ Minimax Algorithm

- ↳ Back-Tracking Algorithm
- ↳ Best move strategy (priority to win the game)
- ↳ Max will maximize the utility (Best move)
- ↳ Min will try to minimize the utility.



Time $\Rightarrow \mathcal{O}(b^d)$
(completeness)

b : branching factor (choices)
d : depth

↳ Used in game playing

↳ 2 players
 \rightarrow Max
 \rightarrow Min

↳ used in tic-tac-toe, checkers, chess, etc.

Properties:

- ① Complete solution
- ② optimal
- ③ Time complexity: $O(b^d)$
- ④ Space complexity = $O(b^d)$

Drawback:

① slow for complex game as chess.

ex: 35 choice for one's turn

100 moves for both player

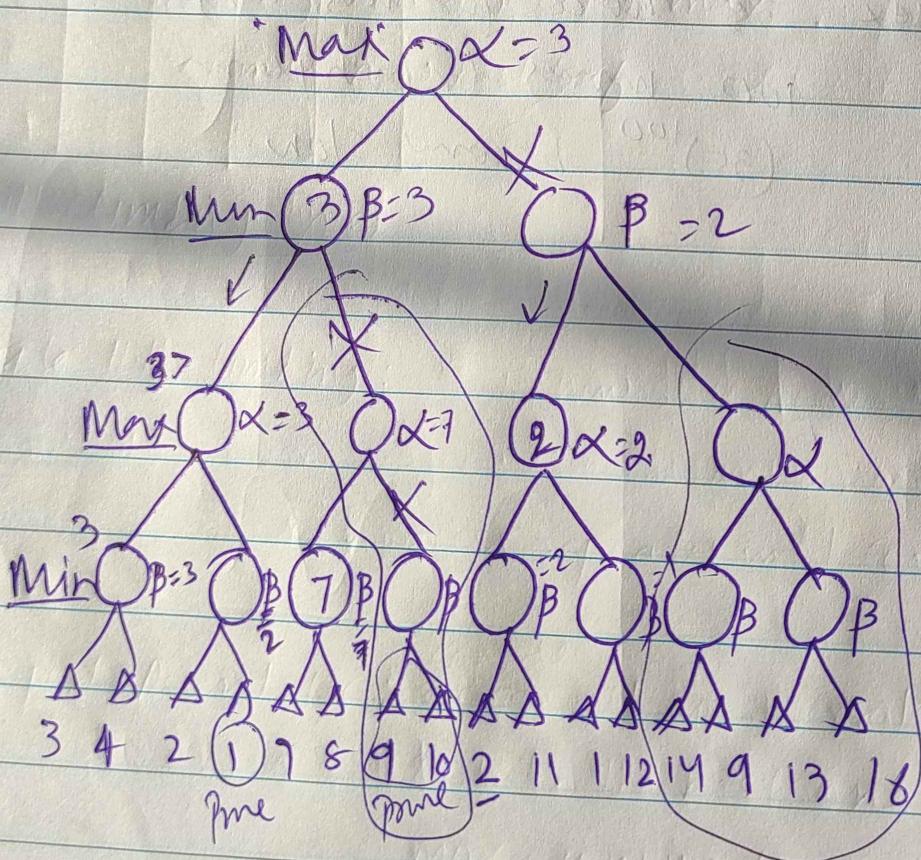
$(35)^{100}$ becomes slow.

⑤ Alpha-Beta Pruning

- ↳ Advanced type of Minimax algo.
- ↳ Reduces the search by exploring less nodes.
- ↳ If we have best path, the remaining paths are pruned.
- ↳ $(\alpha - \beta)$ means

$\alpha \rightarrow$ max node

$\beta \rightarrow$ min node



Time Complexity : $O(b^{d/2})$

4m

① A* search Algo

↳ Informed search algo

- ↳ uses to find the shortest path through search space
- ↳ It uses $h(n)$ & cost to reach the node from start node $g(n)$

↳ Less search Tree

↳ provides optimal result faster

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

$f(n)$: estimated cost of solⁿ.

$h(n)$: heuristic value, cost of node n to goal state

$g(n)$: cost to reach node from start state.

Adv: Best Algo.

optimal & complete

can solve complex problem

disadv: not always produces shortest path.

② Memory Bounded:

PBFS: Recursive best first search algo.

: runs in linear space

: mimics the operation of BFS

drawback: uses so much memory

SMA*: Simple Memory Bounded A*

: If memory is full, adds the new node

: memory bound is set to max

: expands new best leaf, deletes oldest worst leaf

③ Hill Climbing Types:

(1) Simple Hill Climbing: if new state better than curr state,
new state = curr state.
process repeats until found the soln.

(2) Steepest-Ascent-HC: consider all possible sol'n available
Selects the best one
more expensive than simple HC
gives better result

(3) Stochastic HC: similar to simple HC
selects the random neighbour and compares
it to the curr state.

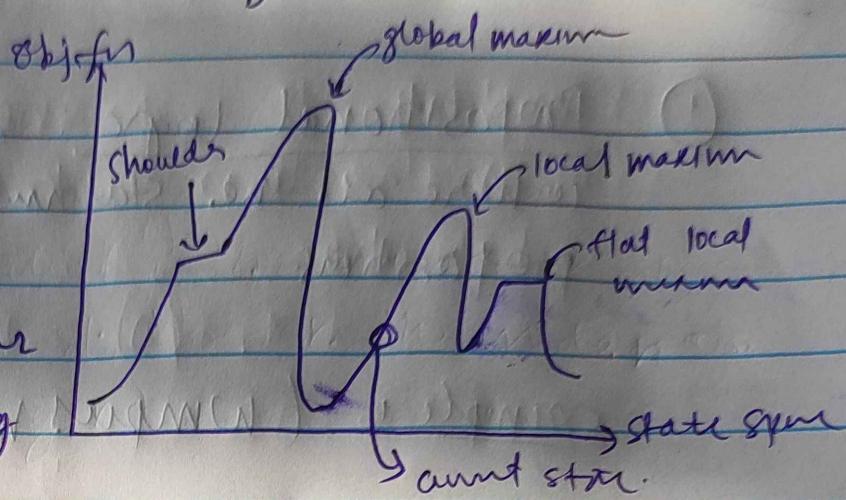
④ State Space Diagram Hill climbing

* X-axis: State space

* Y-axis: Obj. fn

* Local Maximum:

↳ state which is better
than neighbouring
state



* Global Maxm

↳ state which is the
best possible state.

* Current state: present state

* Shoulder: plateau state

* flat local: plateau state

⑤ Blind search

Uninformed Search

- no prior knowledge used
- finds soln slow
- always complete soln
- high cost
- less efficient
- Ex → DFS (depth-fs)
- BFS (breadth fs)

Heuristic Search

Informed Search

- prior knowledge is used
- finds soln quickly.
- may or may not
- low cost
- More efficient
- A* search
- Hill climbing

U-3

A1

12m

①

Propositional logic: simplest form of logic where all the statements are made by proposition.

example: (Wumpus World Problem)

	4	Stench	Breeze	PIT
3		Stench Breeze Gold	PIT	Breeze
2		Stench	Breeze	
1		Gold	PIT	Breeze
	1	2	3	4

Agent (1,1)

→ 4x4 rooms

→ Task: find the gold in 4x4 room and return to the pos and get out of the rooms.

→ PIT & Wumpus: static.

Cond: (1) Room adjacent to wumpus are smelly i.e. stench.

(2) Rooms adjacent to pits has breeze

(3) There will be a glith in room, if the room has gold.

(4)

Action: left move, right move, grab, clear.

Senses: stench, breeze, glith

A = agent $P = pit$
 B = breeze $OK = safe$
 V = $U\downarrow$, $U\uparrow$ $P = pit$
 $V = visited$ $W = wumpus$

S1

	4					
	3					
	2					
	1	(1,1) OK				
	1	1	2	3	4	

Safe

S2

	4					
	3					
	2	(2,1) OK V A	P?			
	1	(1,1) OK V	B \square	(2,1) P?		
	1	1	2	3	4	

Go back to (1,1) cuz there's a breeze and can have PIT.

S3

	4					
	3					
	2	(1,2) OK V A	(2,2) P?			
	1	(1,1) OK V	(2,1) P?			
	1	1	2	3	4	

At (2,2) no breeze, no stench, Agent can visit.

S4

	4					
	3					
	2		(2,3) OK V A			
	1		V OK	(2,1) OK V A		
	1	1	2	3	4	

senses the glitter at (2,3)

② Unification

- ↳ making the expression identical.
- ↳ to make them look identical, we need to do substitution.

eg $P(x, f(y)) \quad P(a, f(g(z)))$

$$x = a$$

$$y = g(z)$$

unification: $[a/x, g(z)/y]$

Substitute x and y with a and $g(z)$

$$P(a, f(g(z))) \quad P(a, f(g(z)))$$

Now, 1st exp. is identical to no 2nd exp.

Set will be $[a/x, g(z)/y]$.

→ Cond:

(1) Predicate symbol must be same & diff predicate symbol cannot be unified.

$$P(n, f(y)) , \quad P(a, f(g(z)))$$

(2) No., of arguments in both must be same

(3) Unification will fail if there are two similar var. in same expression

4) Alg2: Unify (L_1, L_2)

S1: If $L_1 \neq L_2$ is var or const then

(1) If L_1, L_2 are identical, return NIL

(2) else if L_1 is var, then if L_1 occurs in L_2
then return FAIL, else return $\{L_2/L_1\}$

(3) else if L_2 is var, then if L_2 occurs in L_1
then return FAIL, else return $\{L_1/L_2\}$

(4) else return FAIL

S2: If initial symbol is not same, return FAIL

S3: If $L_1 \neq L_2$ have diff arguments, return FAIL

S4 Set SUBST to NIL

S5 Loop: { if $S = \text{fail}$, return fail
if S is not equal to NIL, then
 $SUBST = APPEND(S, SUBST)$ }

S6 Return SUBST

example : $P(x, g(x))$

(i) $P(z, y)$

→ unifies with $[y/z, g(x)/y]$

(ii) $P(z, g(z))$

infers wtr $[y/z, z/x]$

(iii) $P(\text{name}, f(\text{name}))$

does not infer.

(f & g does not match)

⑤ Forward chaining: starts with atomic sentences and applies inference rules in fwd dir to extract more data until goal is reached.

~~Properties:~~

(i) moves from bottom to top

(ii) process of making a conclusion, by starting from initial state and reach the goal state

(iii) used in expert system

examples:

Rule 1: If A & C then F || A & C \rightarrow F

DB
AB

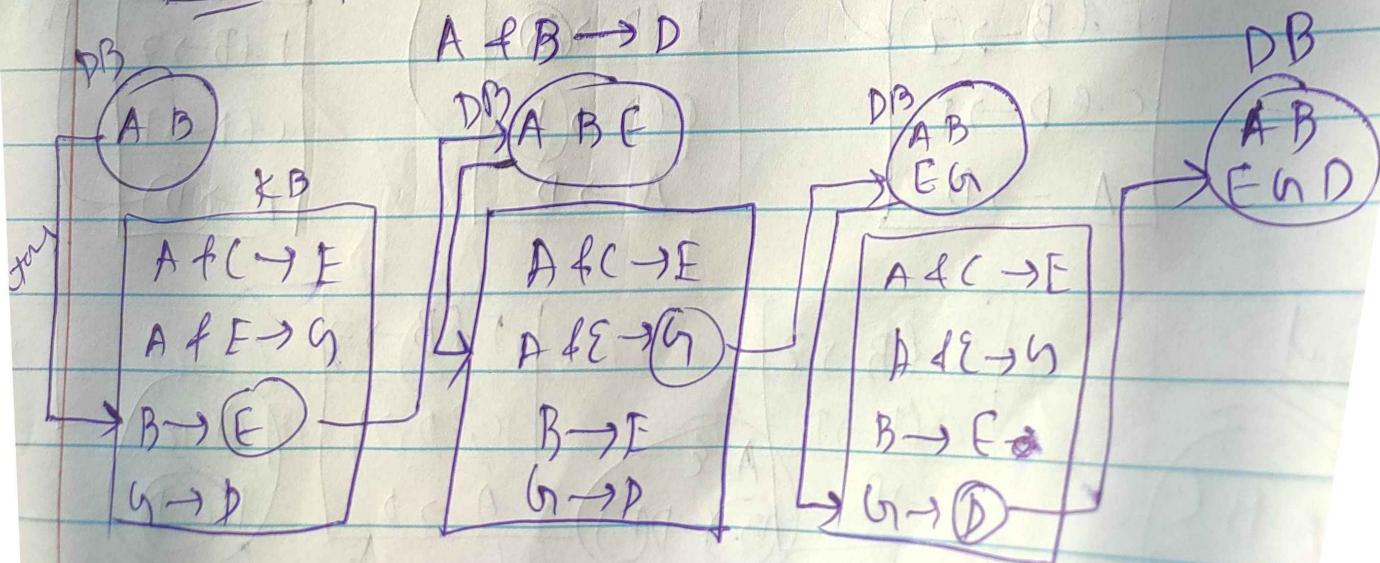
Rule 2: If A & E then G || A & E \rightarrow G

Rule 3: If B ~~&~~ E then E || B \rightarrow E

Rule 4: If C then D || C \rightarrow D

problem: prove A & B are true, then D is true.

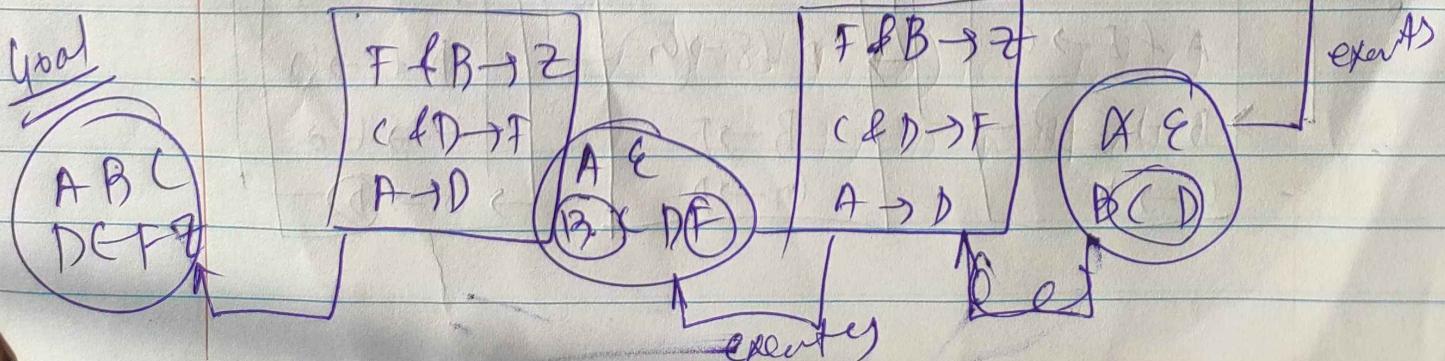
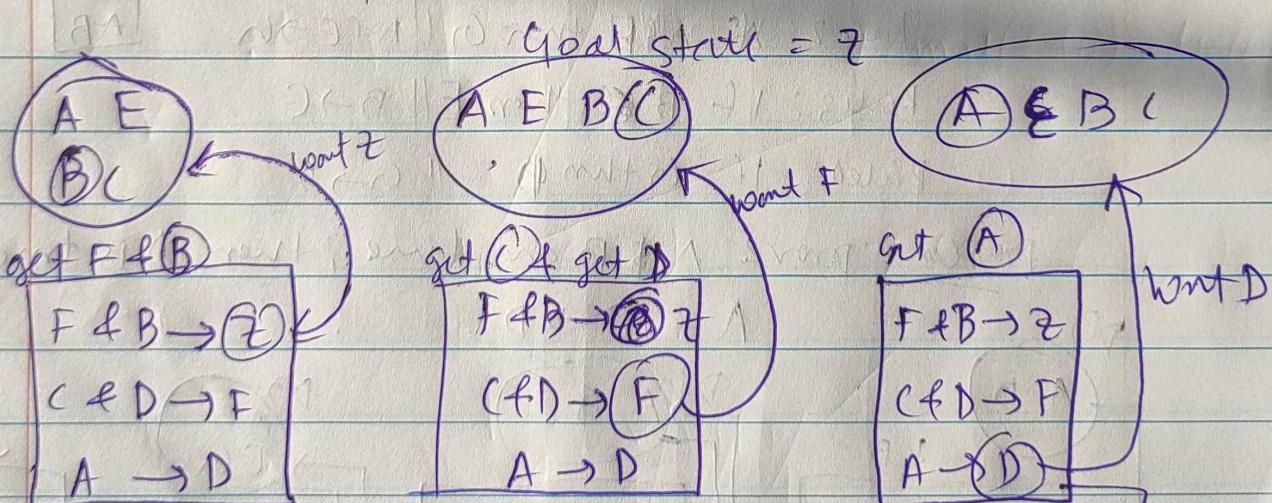
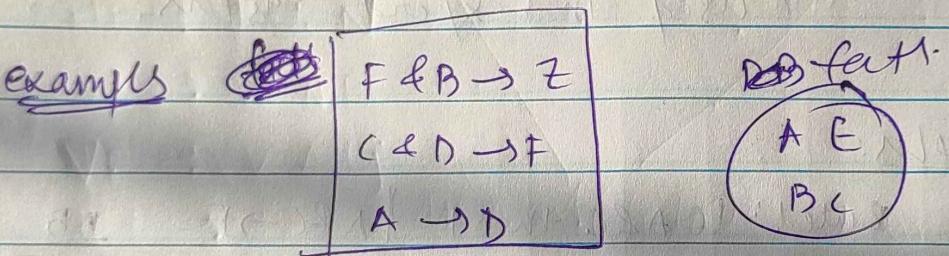
A & B \rightarrow D



Backward Chaining: start with the goals & works backward.

Properties:

- (i) Top-down Approach
- (ii) based on modus ponens inference rule
- (iii) goal is broken in subgoal & subgoals to prove true
- (iv) Game theory, proving tools, assistant
- (v) used in DFS



④ Fido is dog.

All dogs are animals.

All animals will die.

To prove: fido will die

S1 convert to FOL

1) dog(Fido)

2) $\forall x: \text{dog}(x) \rightarrow \text{animal}(x)$

3) $\forall y: \text{animal}(y) \rightarrow \text{die}(y)$

To prove: die(Fido)

S2 convert FOL into CNF.

① Elimination of Implication

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

② Elimination of Quantifiers

1) dog(Fido)

2) $\neg \text{dog}(c) \vee \text{animal}(c)$

3) $\neg \text{animal}(t) \vee \text{die}(t)$

S3 Negate the stat. to be proved:

$\neg \text{die}(\text{fido})$

54: Draw Resolution graph

- 1) dog (Fido)
- 2) $\neg \text{dog} (c) \vee \text{animal} (c)$
- 3) $\neg \text{animal} (t) \vee \text{die} (t)$
- 4) $\neg \neg \text{die} (\text{Fido})$

~~dog (Fido)~~

~~$\neg \neg \text{dog} (c) \vee \text{animal} (c)$~~

~~animal (Fido)~~

~~$\neg \text{animal} (t) \vee \text{die} (t)$~~

~~die (Fido)~~

$\neg \text{die} (\text{Fido})$ is false

Fido will die

Qm:

①



Frames

↳ It is a data structure helps to organize the knowledge

↳ It provides structured way to represent the knowledge by arranging in slots

↳ It consists of slots and each slot is associated with some attributes

↳ each attribute has values associated with that object.

↳ Application: CV

Expert systems

Example: a frame rep. car might have slots for making

* Model

* Year

* Colour

* engine type

② Production Rule

S: sentence

NP: noun phrase

VP: verb phrase

Pet: determiner

N: noun

V: verb

Adj: adjective

$S \rightarrow NP VP$

$NP \rightarrow Pet N$

$VP \rightarrow V NP$

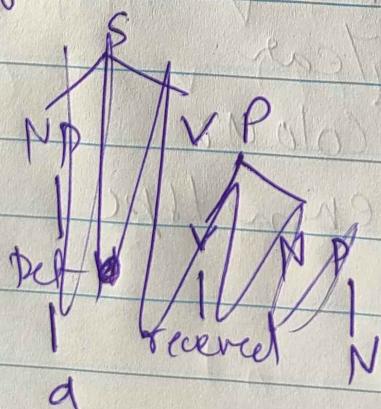
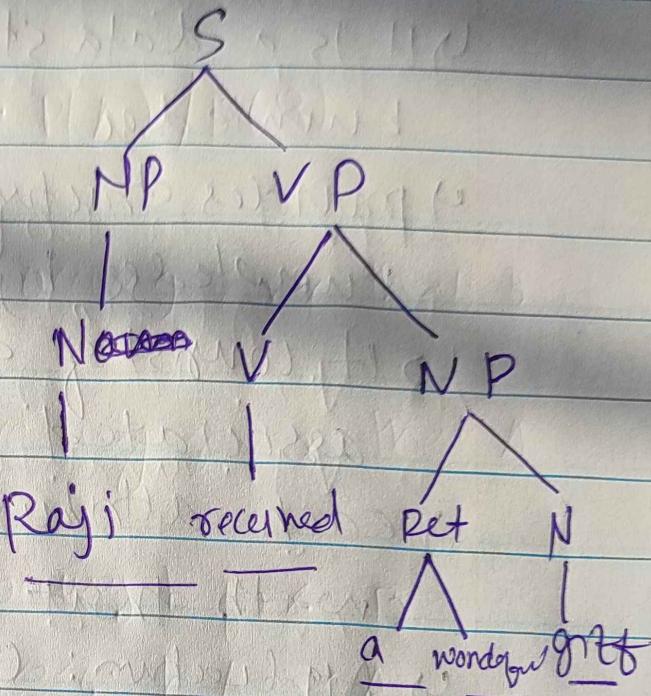
$NP \rightarrow V Adj$

$Pet \rightarrow "a"$

$N \rightarrow "Raj"$

$V \rightarrow "received"$

$Adj \rightarrow "wonderful"$



9

Everyone likes someone.

Let $L(x, y)$ be " x likes y ".

x is subject

y is object

$\forall x \exists y \{ \text{for all } x \text{ there exists a } y \}$
such that x likes y

8

FOPC to English

(i) $\forall x \text{ IsABunny}(x) \Rightarrow \text{IsCute}(x)$

(ii) $\forall x \text{ IsAStuart}(x) \wedge \text{IsTalkingAll}(x) \Rightarrow \text{IsCool}(x)$

(i) for all x , if x is a bunny then x is cute

(ii) for all x , if x is a stuart and x is talking
then x is cool

9

All lions are fierce. Some lion do not drink coffee.

Some fierce do not drink coffee

$F(x) \rightarrow x$ is fierce, $L(x) \rightarrow x$ is lion, $D(x) \rightarrow x$ drinks coffee

① $\forall x ((L(x)) \rightarrow F(x))$ (all lions are fierce)

② $\exists x (L(x) \wedge \neg D(x))$ (some lions don't drink coffee)

③ $L(c) \wedge \neg D(c)$

④ $L(c) \rightarrow F(c)$

⑤ $F(c)$

⑥ $\exists x (F(x) \wedge \neg D(x))$

some fierce do not drink coffee

Bayes theorem

↳ used in ML & AI

$$\hookrightarrow P(A|B) = P(B|A) \cdot P(A) / P(B)$$

$P(A|B) \rightarrow$ Prob. of A occurring given that B is occurred

$P(B|A) \rightarrow$ Prob. of B occurring given that A is occurred

$P(A) \rightarrow$ Prob. of A occurring before B

$P(B) \rightarrow$ Prob. of B occurring before A