AI CT2

Artificial Intelligence (SRM Institute of Science and Technology)

U-2

AI

Uninformed
↳ BFS
↳ UCS
↳ DFS
↳ generate & test

Informed
↳ Hill climbing
↳ A*
↳ TSP

Minimax Search Algo

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.
* Back Tracking is there to avoid stucking.

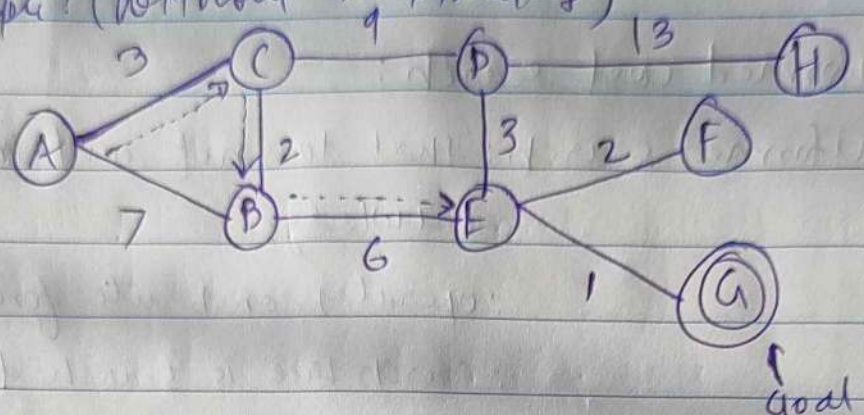
→ Advantages:

(i) Optimal soln (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 is already visited. ED = 3, EF = 2, EG = 1
EG will be chosen.

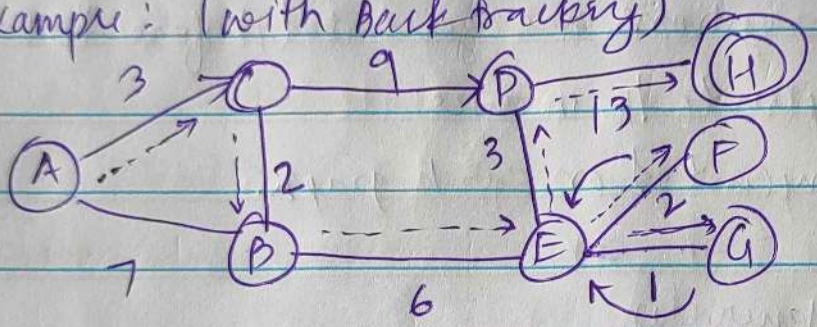
Since EG is goal node it will be terminated here

→ Path to Goal node (G)

(A) $\xrightarrow{3}$ (C) $\xrightarrow{2}$ (B) $\xrightarrow{6}$ (E) $\xrightarrow{1}$ (G)

Cost = 3+2+6+1 = 12

→ Example: (with Back Tracking) ← Goal state



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

Now:

A → C → B → E → G ⟶ E → F ⟶ E → D → ##
(Back Track; Back Track)

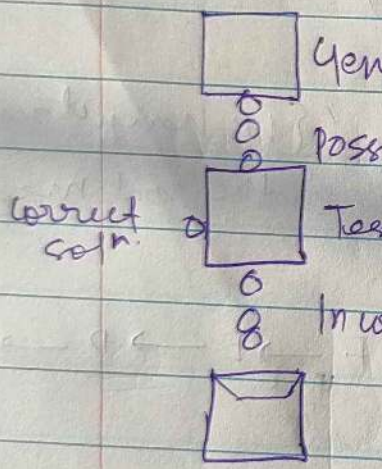
Cost = 3+2+6+1+1+2+2+3+13

= 33

Informed search

② Generate and Test.

- A simple algo.
- guarentees 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.

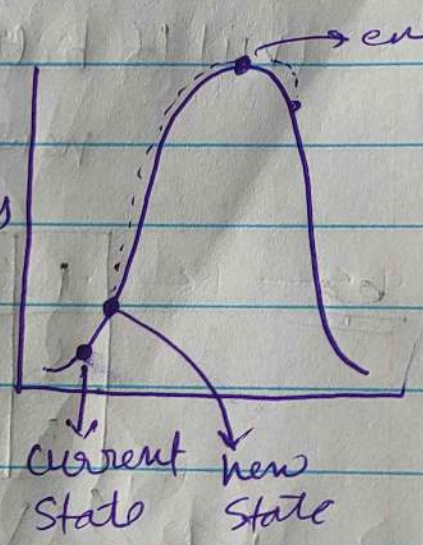
S2 Test the soln

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 Back Track (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

eg:- → 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 & search is completed

(c) 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

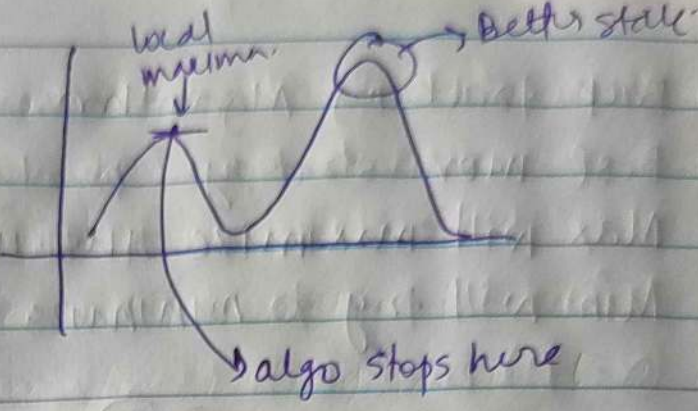
(Start state)

✓ (4) ✗ (5) ✗ (6)

We have to choose the least heuristics value and discard the remaing values.

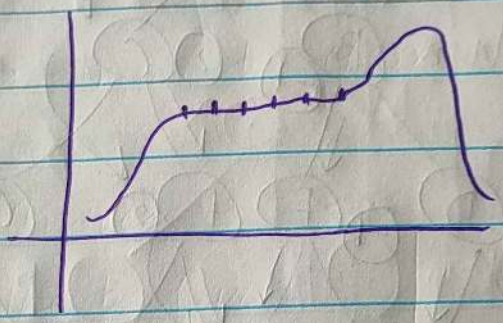
Limitations:

(1) Local Maxima:



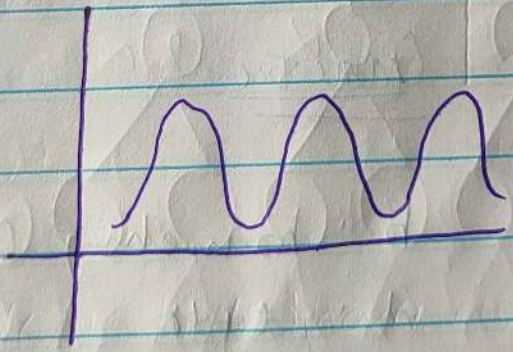
don't have idea about global maxima

(2) Plateau:



algo get same value for some time and algo stops

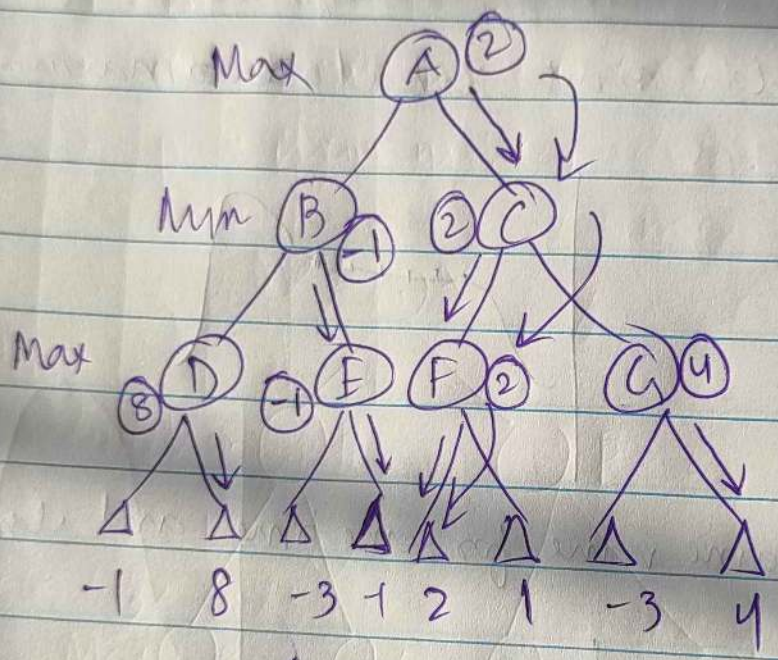
(3) Ridge:



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.



Game Tree

Time (complexity) ⇒ $O(b^d)$ b: branching factor (choices)
d: depth

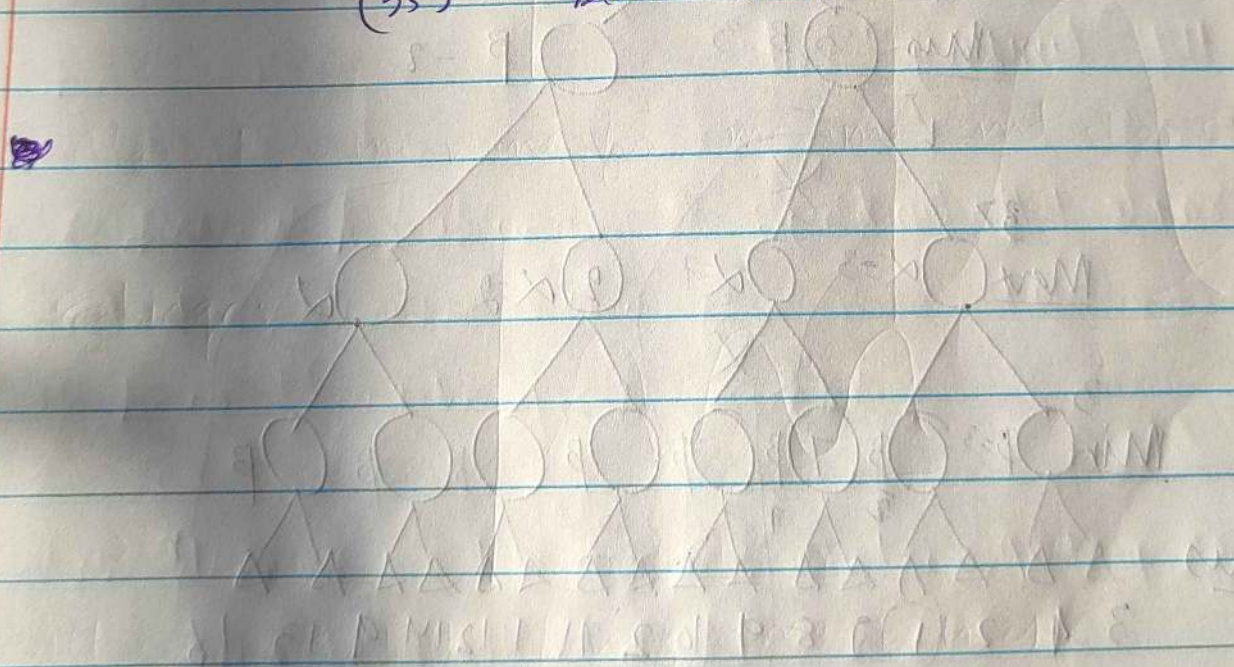
↳ Used in game playing
↳ 2 players → Max
→ Min
↳ used in tic-tac-toe, checkers, chess, etc.

Properties:

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

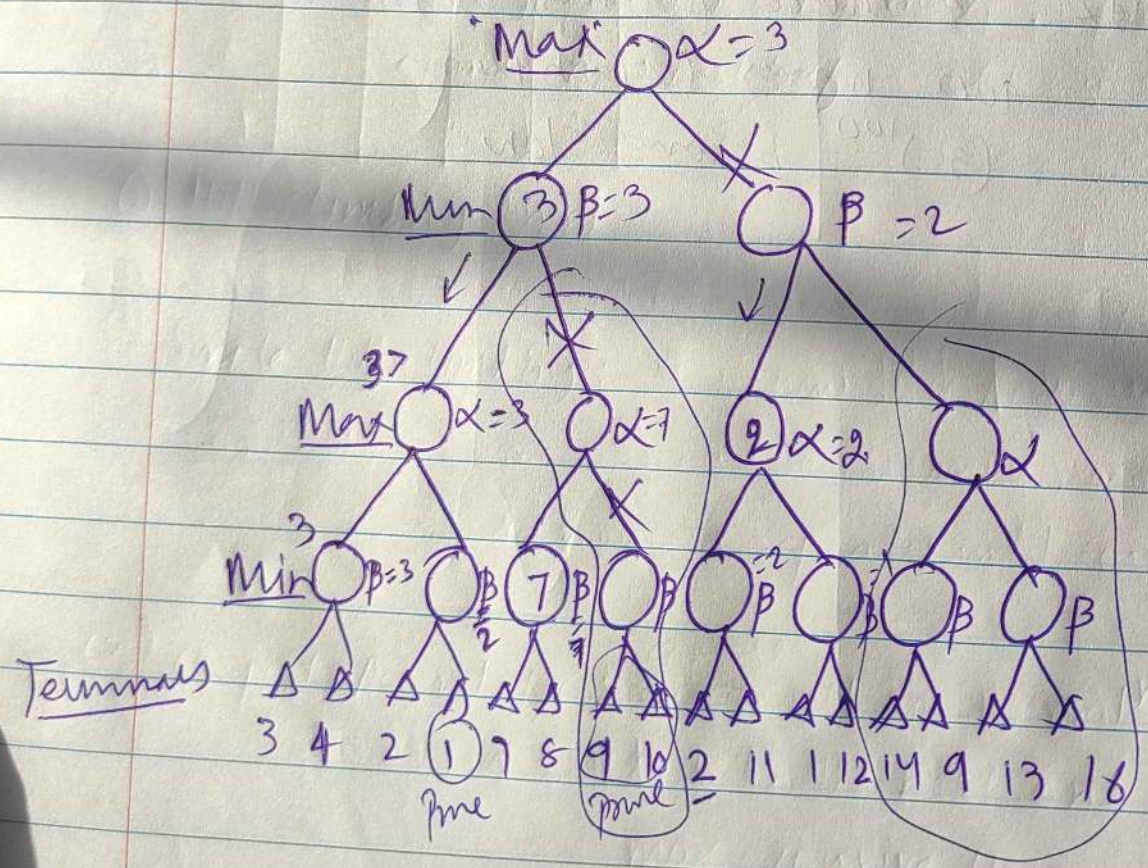
Uniform:

① Slow for complex game as chess.
ex: 35 choice for chess avg.
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 nodes
  - $\beta \rightarrow$ min nodes.



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

Ans

① A* search Algo

↳ Informed search algo
↳ use 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 shorty path.

② Memory Bounded :

RBFS : Recursive best first search algo.
: runs in linear space
: mimics the operation of BFS
: 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.

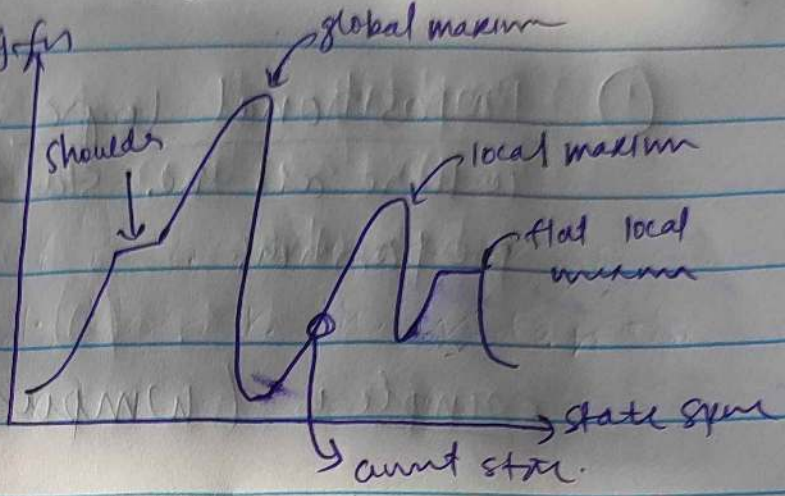
③ Hill climbing Types :

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

(2) Steepest-Ascent-HC : consde all possible soln 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 curnt state.

④ State Space Diagram Hill climbing



* X-axis: state space
* y-axis: Obj-fn
* Local Maxim:
  ↳ state which is better than neighbouring state
* Global Maxim
  ↳ 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
→ find soln quickly.
→ may or may not
→ Low cost
→ More efficient
→ ex→ A* search
Hill climbing

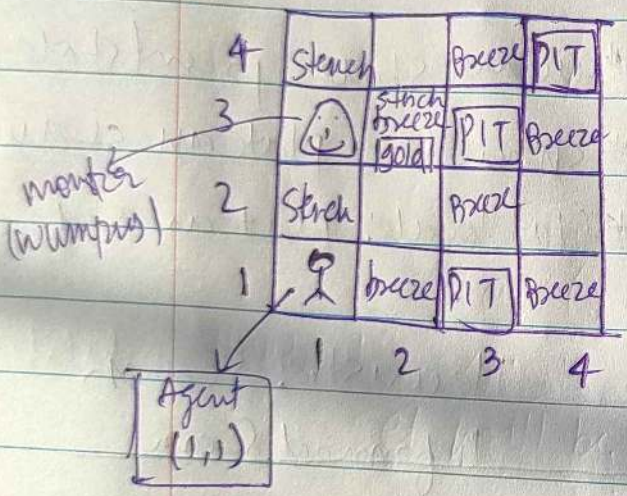
U-3

AI

12m

① Propositional logic: simplest form of logic where all the statements are made by propositions.

example: (wumpus world problem)



→ 4×4 rooms

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

→ PIT & wumpus: static.

Cond: (1) Room adjectnt to wumpus are smelly ie. stench.

(2) Rooms adjacent to pits has breeze

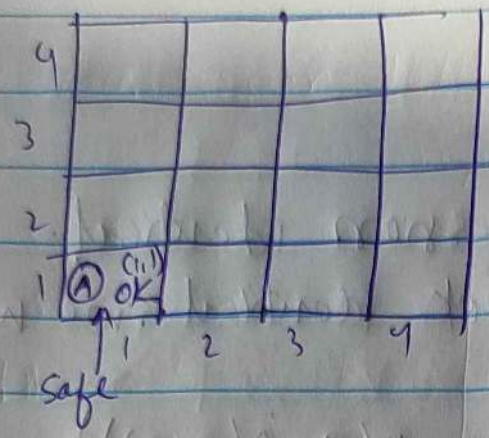
(3) There will be a glitter in room, if the room has gold.

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

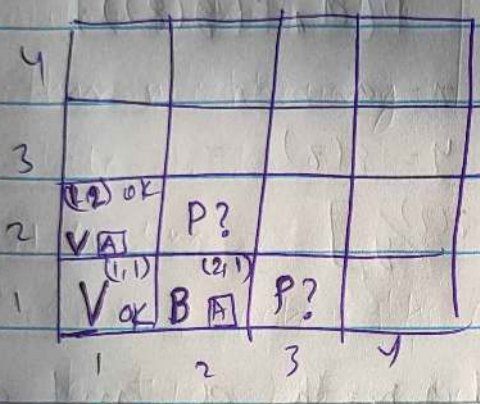
Snses: stench, breeze, glitter

A = agent P = Pit
B = breeze ok = safe
G = Glitter P = Pit
V = visited W = wumpus

S1

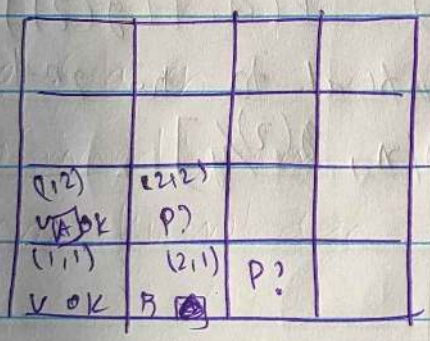


S2



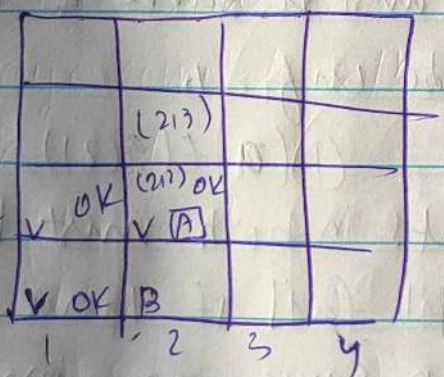
Go back to (1,1) as there's a Breeze and can have PIT.

S3



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

S4



Senses the glitter at (2,3)

② Unification

↳ making the expression identical.

↳ to make them look identical, we need to do substitution.

e.g $P(x, f(y))$ $P(a, f(g(z)))$

$x = a$

$y = g(z)$

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

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

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

Now, 1st exp. is identical to 2nd exp.

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

→ Cond:

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

(P)$(x, f(y))$, (P)$(a, f(g(z)))$

(2) No. of agents in both must be same

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

4) Algo : Unify (L1, L2)

S1: If L1 & L2 is var or const then
(i) If L1 & L2 are identical, return NIL
(2) else if L1 is var, then if L1 occurs in L2 then return FAIL, else return {(L2/L1)}
(3) else if L2 is var, then if L2 occurs in L1 then return FAIL, else return {(L1/L2)}
(4) Else return FAIL

S2: If initial symbol is not same, return FAIL

S3: If L1 & L2 have diff arguments, return FAIL

S4 Set subst to NIL

S5 Loop: { if S = 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 $[x/z, g(x)/y]$

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

unifies with $[x/z, z/x]$

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

does not unifies.

(f & g does not match)

① Forward chaining: starts with atomic sentences and applies inference rules in fwd dirn 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 → F

Rule 2: If A & E then G || A & E → G

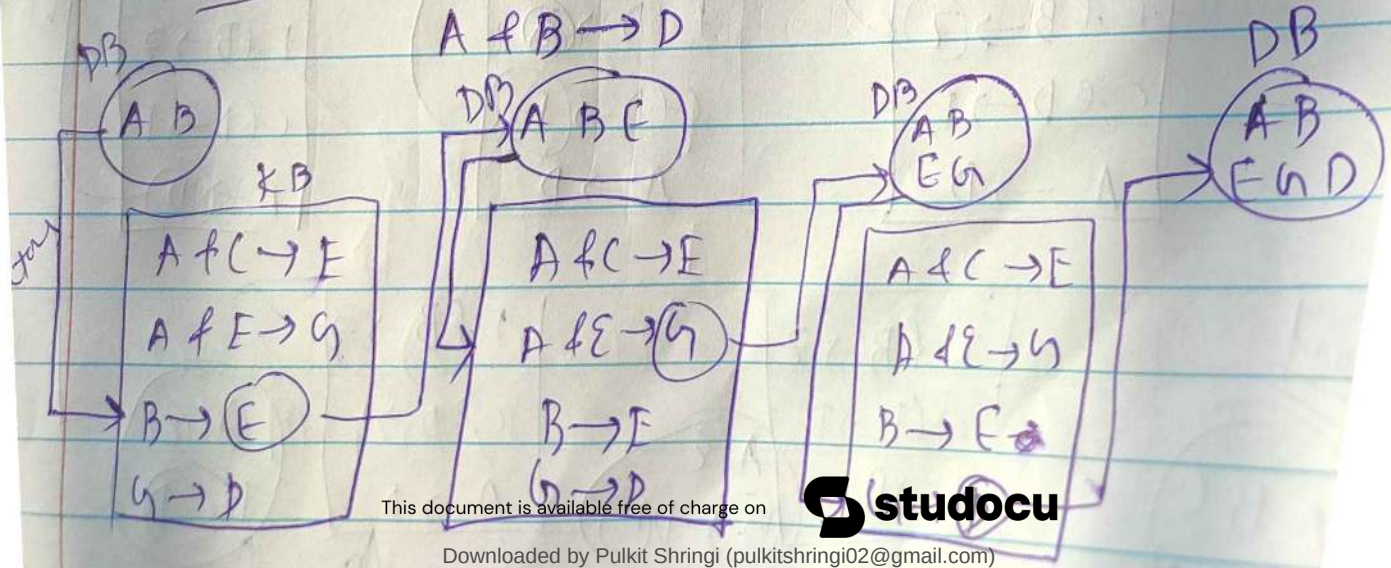
Rule 3: If B then E || B → E

Rule 4: If G then D || G → D

DB

A B

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

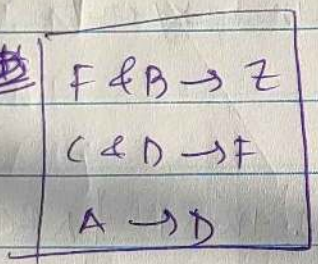


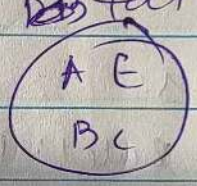
Backward Chaing : start with the goals & works backward.

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

examples





④ Fido is dog.
All dogs are annals.
All annals will die.
To prove: fido will die

S1 convert to FOL

1) dog(Fido)
2) $\forall x$: dog(x) → annals(x)
3) $\forall y$: annals(y) → die(y)

To prove: die(fido)

S2 Conver FOL into CNF.

① Elimination of Implication

$P \rightarrow Q = \neg P \vee Q$

② Elimination of Quantifiers

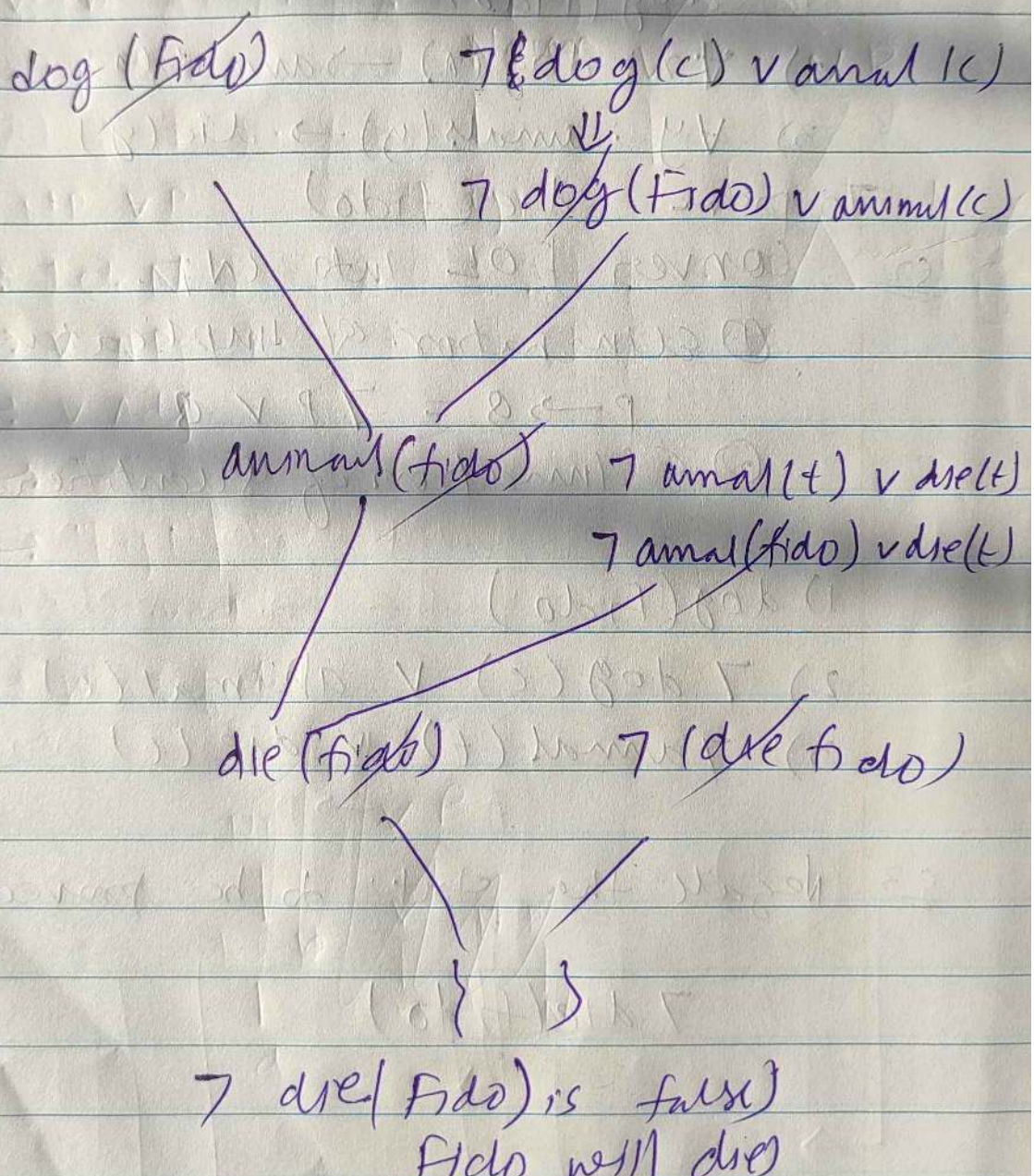
1) dog(Fido)
2) $\neg$ dog(c) V animal(c)
3) $\neg$ annal(t) V die(t)

S3 Negate the stat: to be proved:

$\neg$ die(fido)

Sol: Draw Resolution graph

1) dog (Fido)
2) ¬dog (c) ∨ animal (c)
3) ¬animal (t) ∨ die (t)
4) ¬die (Fido)



Ans:

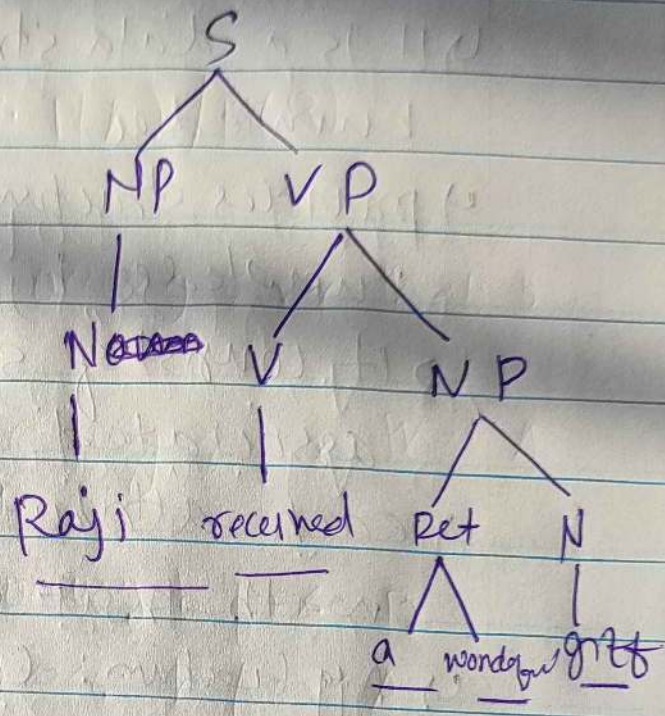
① [redacted] Frames

- It is a data structure helps to organize the knowledge
- provides structured way to represent the knowledge by arranging in slots
- It consist of slots and each slots is associated with some attributes
- each attributes has values associated with that object.
- Application: CV
  Expert systems

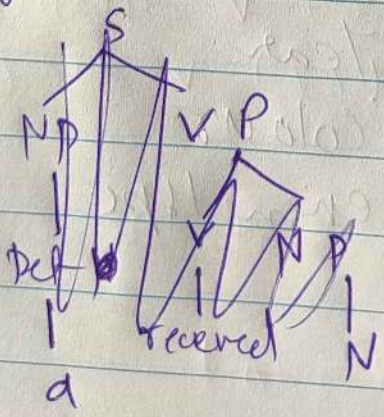
- example: a frame rep. car might have slots for naming
  - Model
  - Year
  - Colour
  - engine type

② Production Rule

S: sentence
NP: noun phrase
VP: verb phrase
Det: determiner
N: noun
V: verb
Adj: adjective



S → NP VP
NP → Det N
VP → V NP
NP → V Adj
Det → "a"
N → "Raji" | "gift"
V → "received"
Adj → 'wonderful'



Q Everyone likes someone.

let L(x,y) be "x likes y"
x is subject
y is object

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

Q FOPL to english

(i) $\forall x$ IsABunny(x) $\Rightarrow$ IsCute(x)

(ii) $\forall x$ IsAStuart(x) $\wedge$ IsTalkingAll(x) $\Rightarrow$ 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

Q All lions are fierce. Some lion does not drink coffee.

Some fierce do not drink coffee

F(x) → x is fierce, L(x) → x is lion, D(x) → 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 dont 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

↳ $P(A|B) = P(B|A) * P(A) / P(B)$

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

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

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

$P(B) \rightarrow$ prob. of B occ. before A