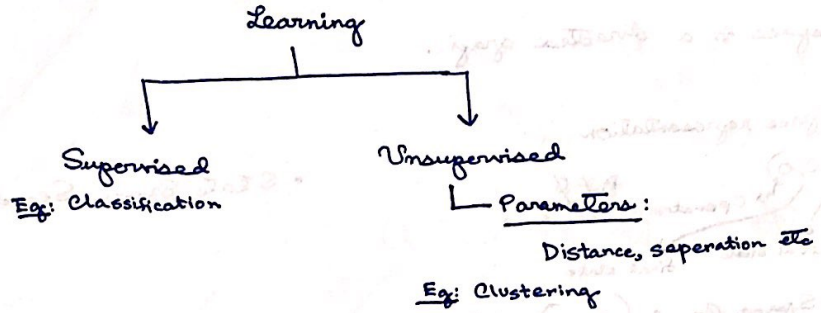
Artificial Intelligence

Characteristics of Intelligent Machines —

1) Perception capability
2) Learning from experience
3) Natural language understanding

- Whatever intelligent machine we develop is called agent/ rational agent.



Constraint Satisfaction: Generally solved using backtracking.

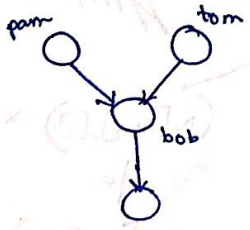
- Propositional Logic } Differentiated using Quantifiers
- Predicate Logic }

- First Order Predicate Logic (FOPL)
  - The quantifiers are associated with the variable
- Second Order Predicate Logic
  - The quantifiers are associated with functions.

Importance of Knowledge

- Facts
- Rules

- PROLOG always always follows universal quantifier.



fact: parent(tom, bob).

rule: offspring(X, Y) :- parent(Y, X).

↓

Prolog Syntax

Agents

Performance Measure — Performance metrics need to be specified.

- What is rational at any given time depends on four things : PPT
- Environments — PPT
- Types of Agents — PPT

Problem Solving

State space is a directed graph.

- State Space representation

(A, K, C, O) $A \neq \emptyset$

A → states, K → initial state, C → Operator, O → final state

- State Space Search Algorithms
- State-Space Graph (A, E) → Directed Graph

A → no. of nodes

- Problems —

i) Wolf-goat-cabbage Problem

— State-Space Representation

Initial State: $S(L, [w, g, c], [\,])$

Final State: $S(R, [\,], [w, g, c])$

Operators: w, g, c, alone (a)

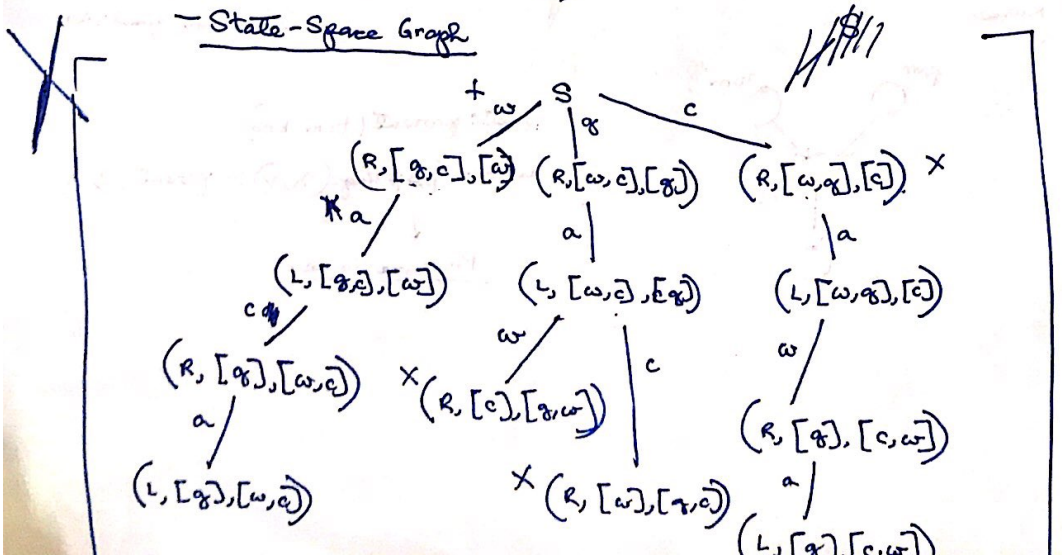
$A = \{\}$

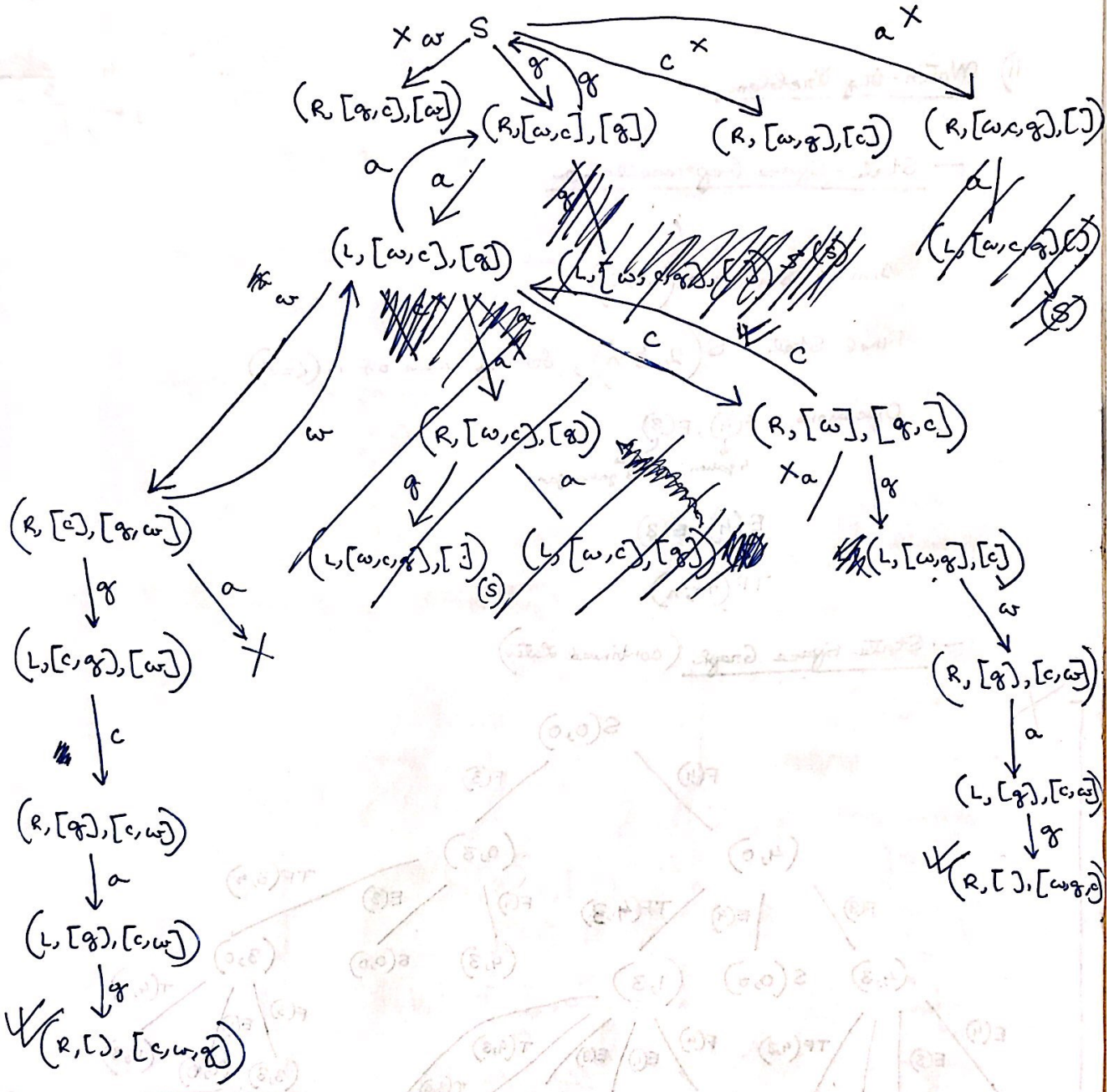
$K = \{\{w, g, c\}, \{\emptyset\}\}$

$C = \{\{\emptyset\}, \{w, g, c\}\}$

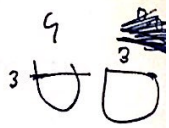
$O = \{l \xrightarrow{w} r, l \xrightarrow{g} r, l \xrightarrow{c} r\}$

— State-Space Graph





ii) Water-Jug Problem



— State-Space Representation

Initial State : $S(0,0)$

Final State : $S(2, n)$, for any value of n ($<=3$)

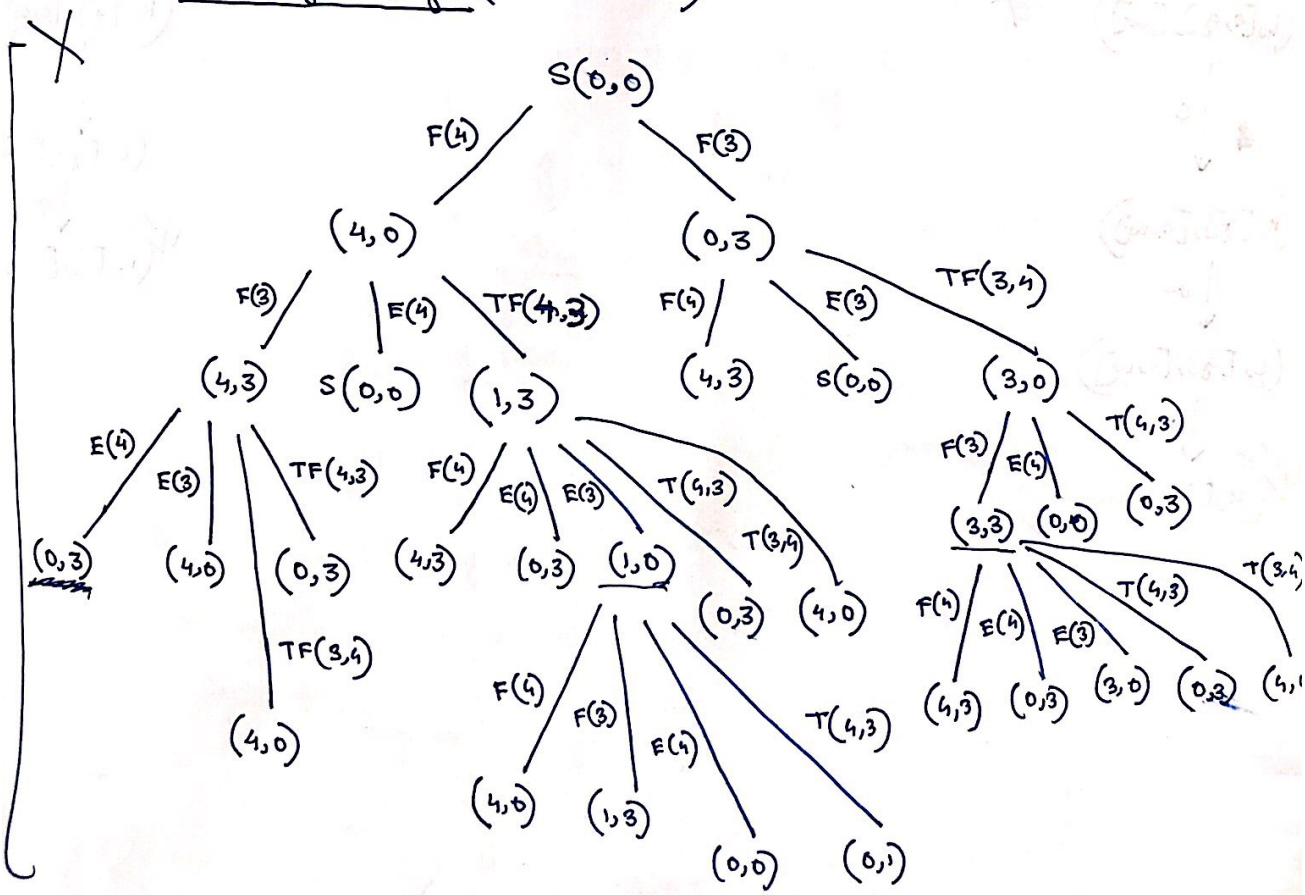
Operators : $F(4)$, $F(3)$

$F(4)$ → 4 gallon jar, $F(3)$ → 3 gallon jar

$E(4)$, $E(3)$

$TF(m,n)$

— State Space Graph (continued Later)



iii) Missionaries and Cannibals

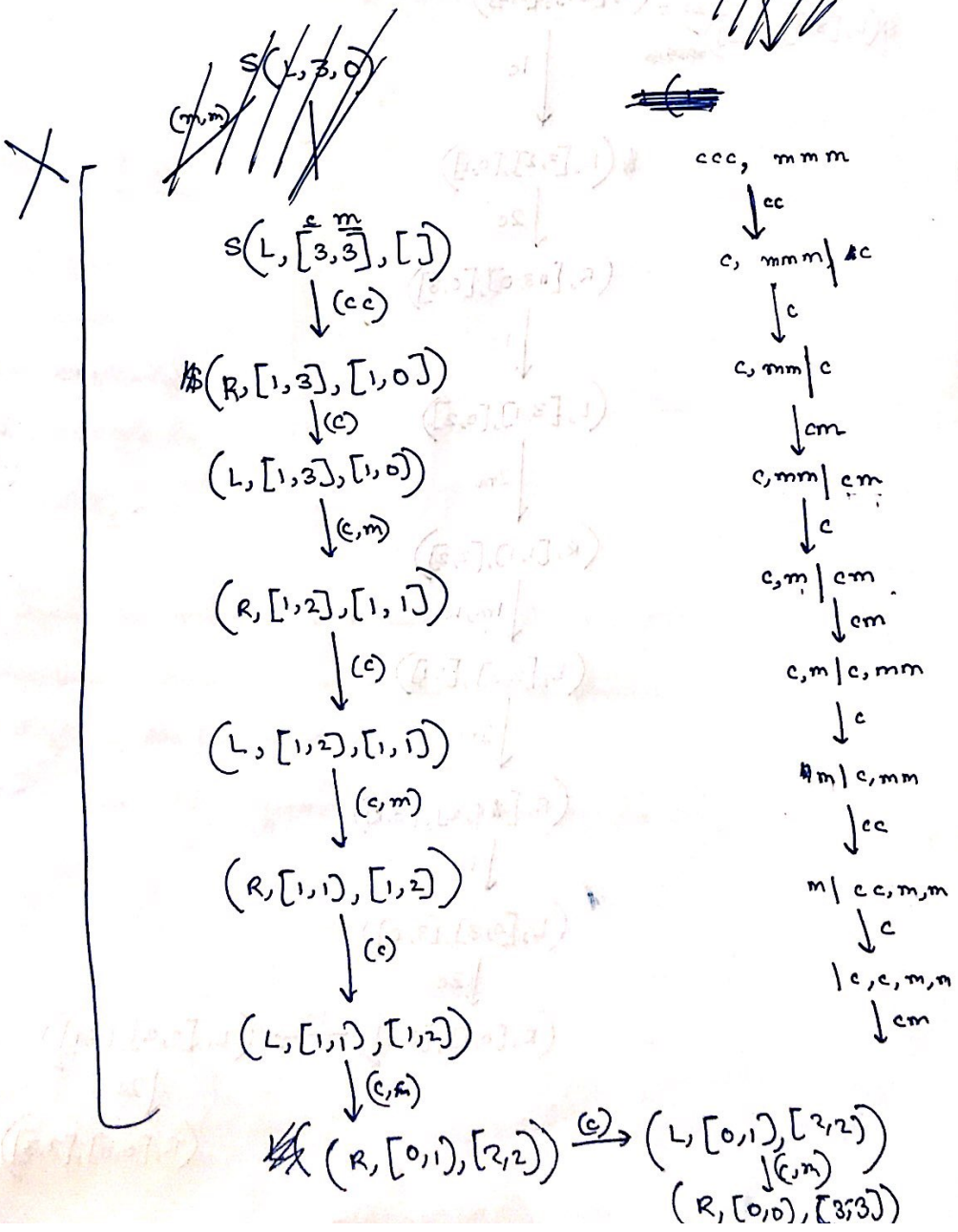
— State-Space Representation

Initial State : S(L, [3,3], [ ])

Final State : S(R, [ ], [3,3])

Operators : (m,m)
(m,c)
(c,c)
(c)
(m)

— State Space Graph (Next Page)



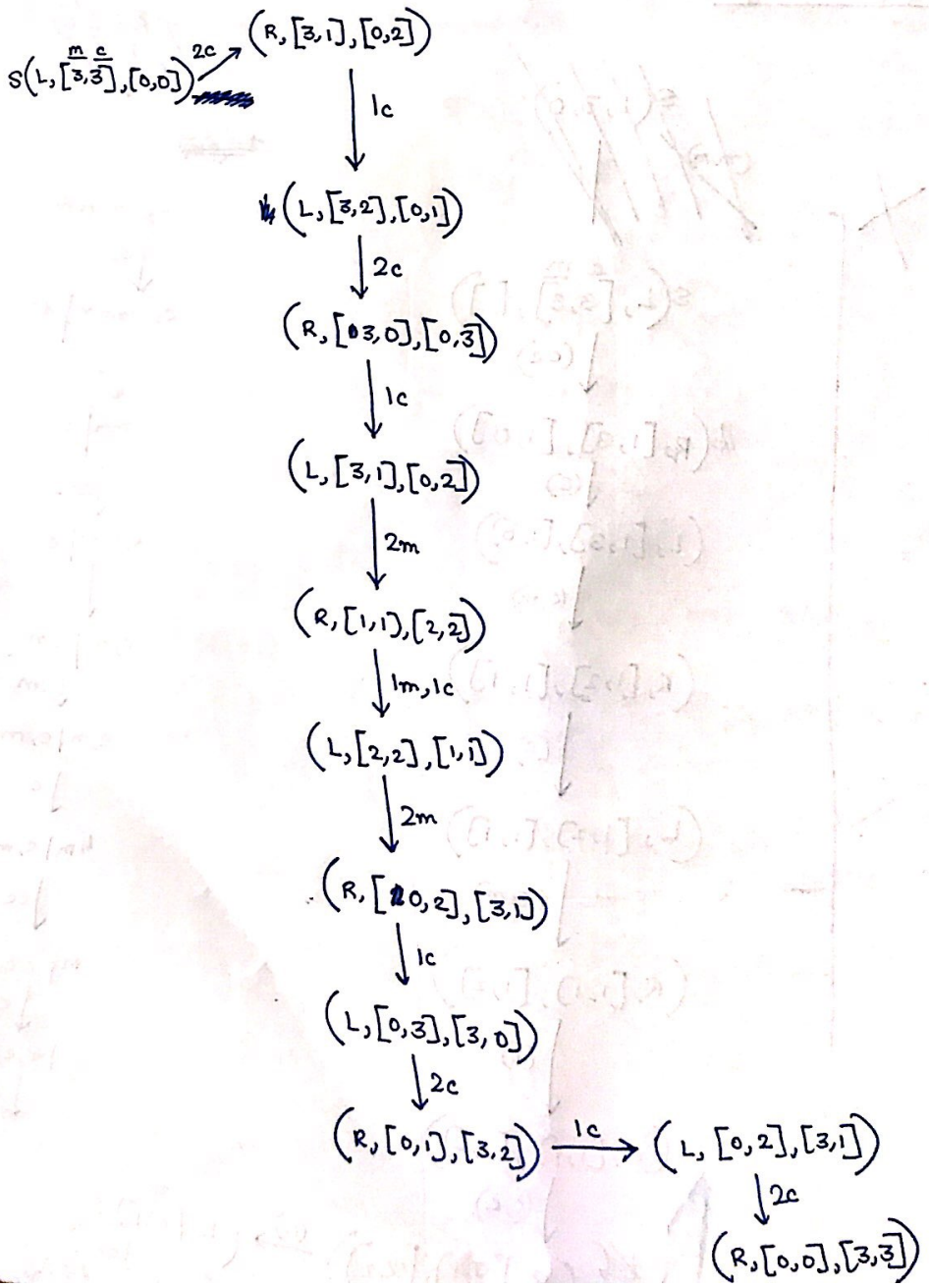
Search Methods

- Blind or uninformed Search, eg. BFS, DFS.
- Informed or Directed Search (Heuristic Search)
  - We know which one may be a promising option to reach our goal

(Only start state info is given and the option is chosen purely randomly and we don't know which one may be a promising option to take to reach our goal)

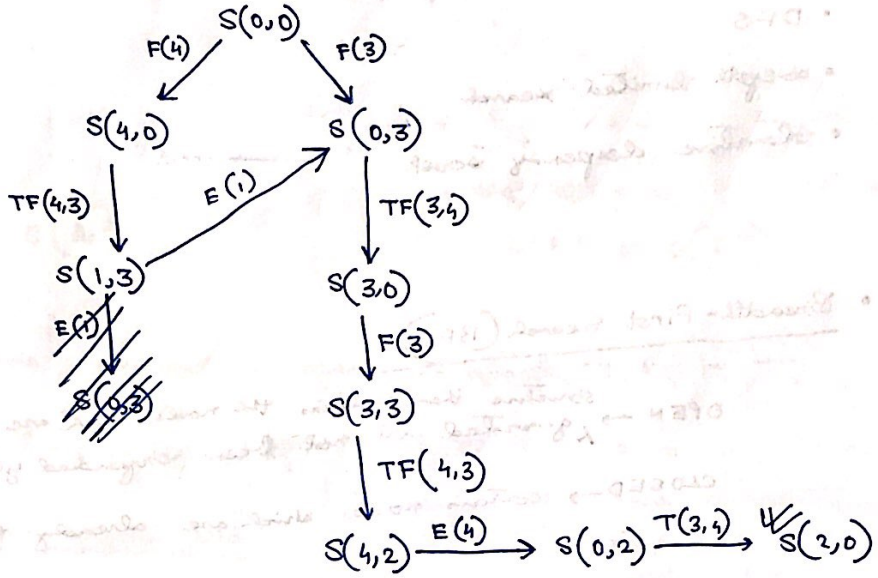
Missionaries and Cannibals (continued)

State Space Graph — (Solution) ~~Not optimal~~



Water - Jug Problem (continued)

State Space Graph —



Search Methods (continued)

- **Search Strategies:**
  - i) Completeness
  - ii) Time Complexity
  - iii) Space complexity
  - iv) Optimality

Time and space complexity are measured in terms of —

b — maximum branching factor of the search tree.

d — depth of the least-cost solution

m — maximum depth of the state space (may be ∞)

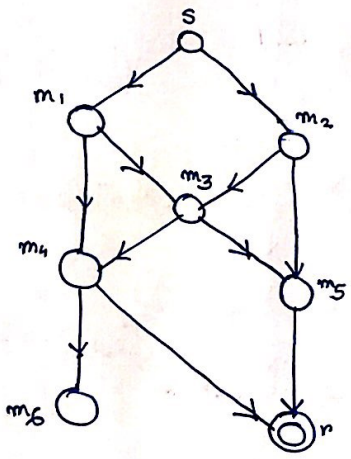
Uninformed or Blind Search

- BFS
- Uniform-cost search ---> Same as Djistra
- DFS
- depth-limited search
- Iterative deepening search

- **Breadth-First Search (BFS)**

OPEN → structure that contains the nodes which are generated but not been expanded yet (Behaves like a queue)

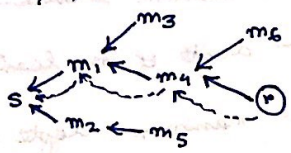
CLOSED → contain nodes which are already expanded.



OPEN : ~~S~~ ~~m1~~ ~~m2~~ ~~m4~~ ~~m3~~ ~~m3~~ ~~m5~~ ~~m6~~ ~~n~~

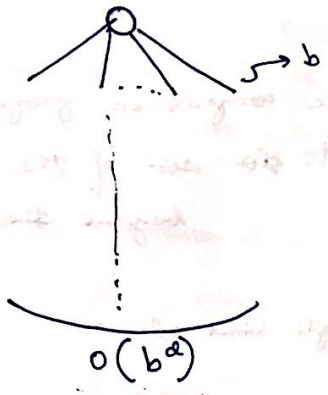
CLOSED : S ~~m1~~ ~~m2~~ ~~m4~~ ~~m5~~ ~~m3~~

OPEN	CLOSED
S	S
~~m1~~ m2	S m1
~~m2~~ m3 m4	S m1 m2
~~m3~~ m4 m5	S m1 m2 m3
m4 m5	
m5 m6 (n)	



path: S → m1 → m4 → (n)

Worst Case Complexities:



for BFS, b and m are same.

∴ in the end, it can be $O(b^d)$ or $O(b^m)$.

∴ Time complexity = Space complexity = $O(b^d)$.

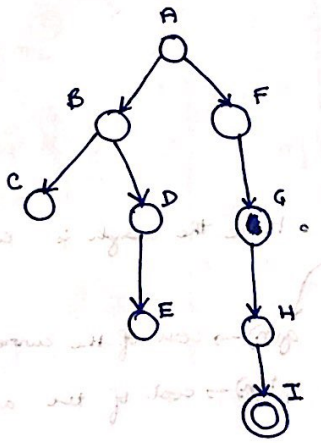
- If the branching factor (b) is infinite, then there may be a situation where completeness is not attained in case of BFS.

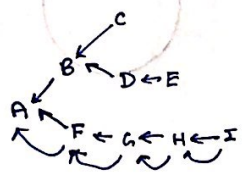
- **Depth-First Search (DFS)**

Time Complexity: $O(b^m)$

Space Complexity: $O(mb)$

- May not be complete.





OPEN	CLOSED
A	A
F	A B
D F	A B
F	D
F	
F	F
G	G
H	H
I	

- **Depth-Limited Search:**

This algorithm tries to attain the completeness property, but fails to do so if the goal lies beyond l.

Time: $O(b^l)$

Space: $O(bl)$

Special case of DFS with depth limit $= l = \infty$.

- **Iterative Deepening DFS:**

Makes use of both BFS and DFS.

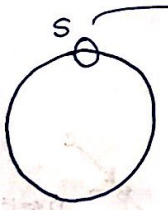
Nodes generated: $b^d \cdot 1 + b^{d-1} \cdot 2 + \ldots + 1 \cdot d$

$= O(b^d)$

$\therefore$ Time complexity $= O(b^m)$

$\therefore$ Space complexity $= O(b^d)$

Uniform-cost search

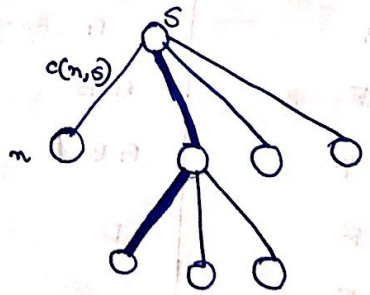


- Here the graph is weighted

$g(n) \rightarrow$ cost of the currently known best path from s to n

$g^*(n) \rightarrow$ cost of the actual best path

$g^*(n) \leq g(n)$



Informed Search

In uniform cost, we do not consider how far we are from the goal state. But in informed search, we take this into account.

$h(n)$ is a function which gives an estimated value of the cost of the remaining path, from n to goal node.

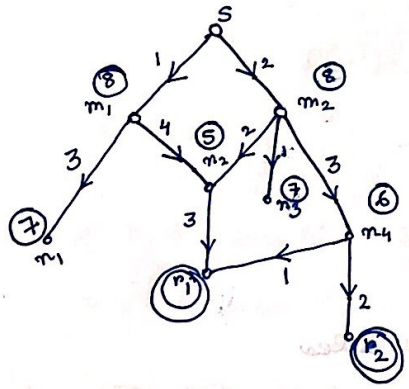
It is not guaranteed that this heuristic will find the best result.

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

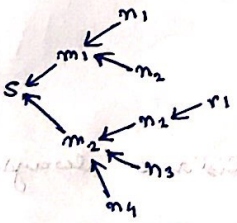
$g(n)$ → cost of the currently known best path from s to n.

$h^*(n)$ → actual cost from n to goal node

A* Algorithm



	S	m_1	m_2	n_1	n_2	n_3	n_4	r_1	r_2
1	0								
2	[0]	9	10						
3		[9]	10	11	10				
4			[10]	11	9	10	11		
5				11	[9]	10	11	7	
6								[7]	



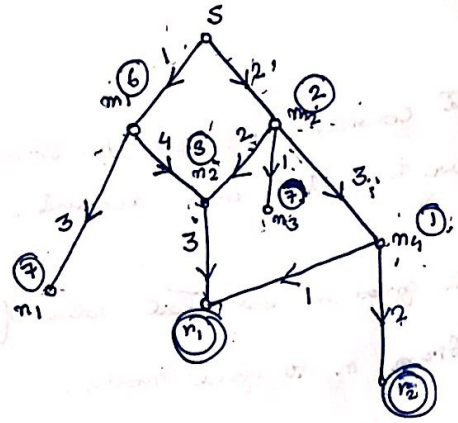
∴ path: $s \to m_2 \to n_2 \to r_1$

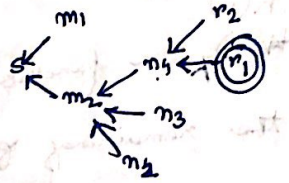
cost: 7

But, this is not optimal.

$s \to m_2 \to n_4 \to r_1$ is the optimal path for r_1 with cost 6.

A* will give the optimal solution if $h(n) \leq h^*(n)$ for all nodes.





path: $S \rightarrow m_2 \rightarrow m_4 \rightarrow r_1$

cost: 6

admissible

	S	m_1	m_2	n_1	n_2	n_3	n_4	r_1	r_2
1	0								
2	[0]	7	4						
3		7	[4]		7	10	6		
4		7			7	10	[6]	6	7
5								[6]	

Heuristic Functions

- 8 Queen Puzzle

- h_1 = the number of misplaced tiles
- h_2 = the sum of the distances of the ~~tiles~~ tiles from their goal positions.
  (Sum of the Manhatten distance)

- Prove that the heuristic sum of the manhatten distance always gives ~~us~~ you the optimal solution for the 8 puzzle problem.

- For any node n, $h_2(n) \geq h_1(n)$, ie. h_2 dominates h_1
- It is always better to use a heuristic function with

higher values, provided it does not overestimate and that the computational time for the ~~the~~ heuristic is not too large

Theorem

1. If A^* uses a consistent heuristic, then prove that $f(n)$ is non-decreasing along any path.

consistent heuristic: The heuristic is said to be consistent if $h(N) \leq c(N, N') + h(N')$ where N' is a child of N.

$c(N, N')$ ↓ cost of the edge between N and N'

Proof:

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

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

$$\leq g(n) + c(n, n') + h(n')$$

$$\leq g(n) + h(n)$$

$$\therefore f(n') \leq f(n)$$

$h(N) \leq c(N, N') + h(N')$

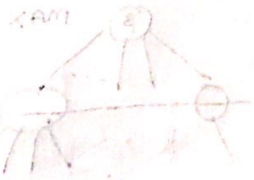
$f(n) = g(n) + h(n)$

$f(n) \leq g(n) + c(n, n') + h(n$

$\leq g(n') + h(n')$

$\leq f(n')$

2. If h is consistent then prove that this rule can be applied to any descendant N' of N.



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

$\leq f(n')$

$\leq c(N, n') + h(n')$

Proof:

$$h(N) \leq c(N, N') + h(N')$$

$$h(N') \leq c(N', N'') + h(N'')$$

$$h(N) \leq c(N, N') + c(N', N'') + h(N'')$$

$$\leq c(N, N'') + h(N'')$$

3. If $h(n)$ is consistent, prove that it is admissible also. ($h(n) \leq h^*(n)$)

Proof:

$$h(N) \leq c(N, N') + h(N')$$

from last proof,

$$h(N) \leq c(N, G) + h(G)$$

($c(N,G) \to h^*(N)$, $h(G) \to 0$)

$$\leq h^*(N)$$

— WAP to implement A* on 8 Queen Puzzle taking heuristic as the sum of manhatten distance.

Game Playing / Adversarial Search

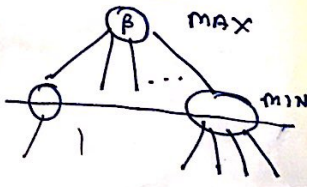
- Evaluation Function (heuristic): Tells how good the situation is for you, or how bad the situation is for your opponent or if it is neutral.

$f(n)$ (state) = large +ve → good for max, bad for min

= large −ve → good for min, bad for max

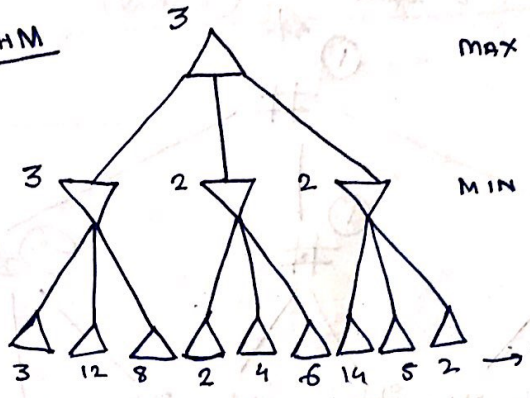
≈ 0 → neutral

Game Tree



MINIMAX ALGORITHM

Consider the following game Tree —



→ Values calculated using evaluation function

The backed up score of the min nodes will be the minimum of all its successors. For max ~~nodes~~ nodes the maximum of all its successors is backed up.

Evaluation function Calculation (Example - Tic-Tac-Toe)

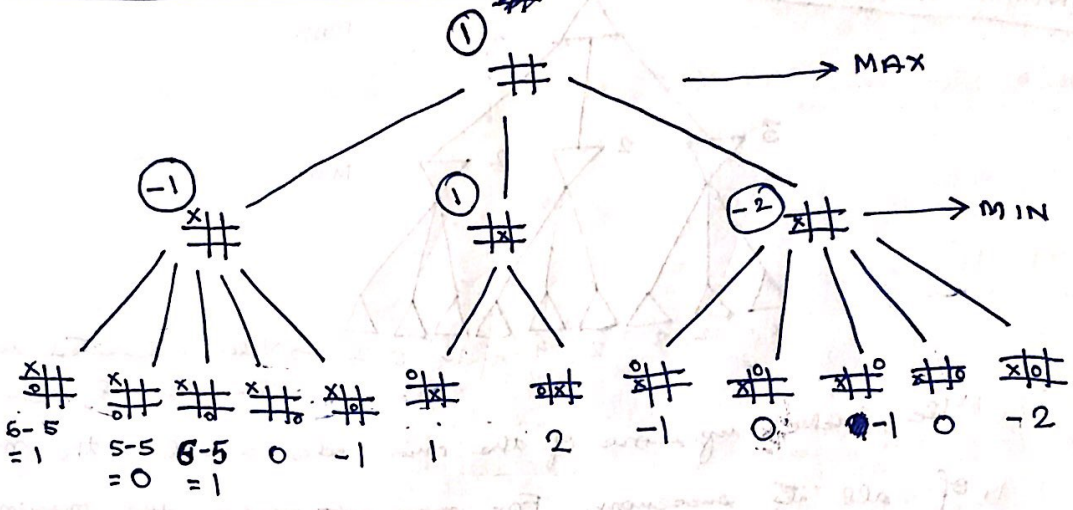
In terms of Tic-Tac-Toe let's assume max marks X and min marks O. The evaluation function e(p) for a given board position p may be calculated as follows —

i) If p is not a winning board position for either player, then e(p) = number of complete rows, columns or diagonals that are still open for MAX
— number of complete rows - columns or diagonals that are still open for MIN

ii) If p is a win for MAX, $e(p) = +\infty$.

iii) If p is a win for MIN, $e(p) = -\infty$.

Consider the following game tree (Partial)

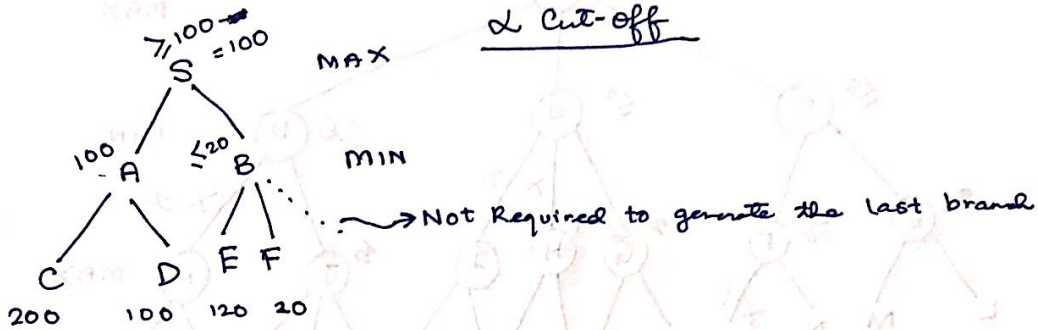


Algorithm MINIMAX
{
v = value (root);
return v;
}

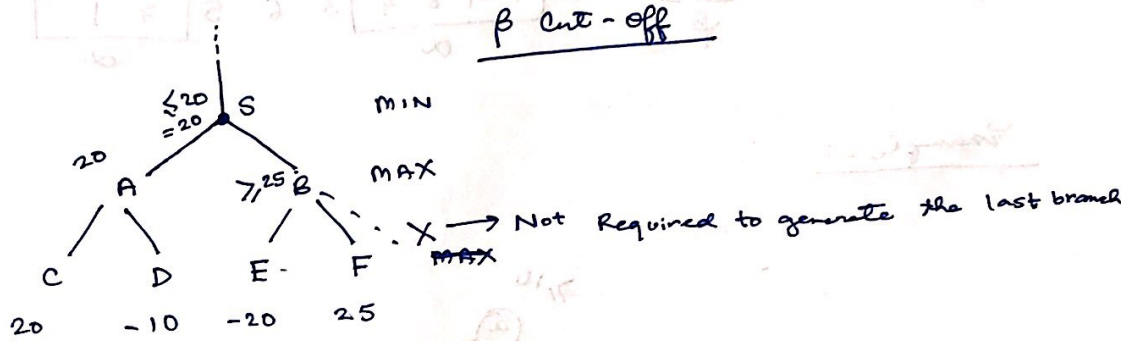
Algorithm value(n)
{
if n is a terminal then
return e(n);
if n is a MAX node then
cvalue = - ∞;
if n is a MIN node then
cvalue = + ∞;
For each successor n_i of n do
{
generate n_i;
if n is a MAX node than
cvalue = max (cvalue, value(n_i))
else
cvalue = min (cvalue, value(n_i))
}
return (cvalue);
}

Alpha (α) - Beta (β) Pruning / Cut off

α Cut-off



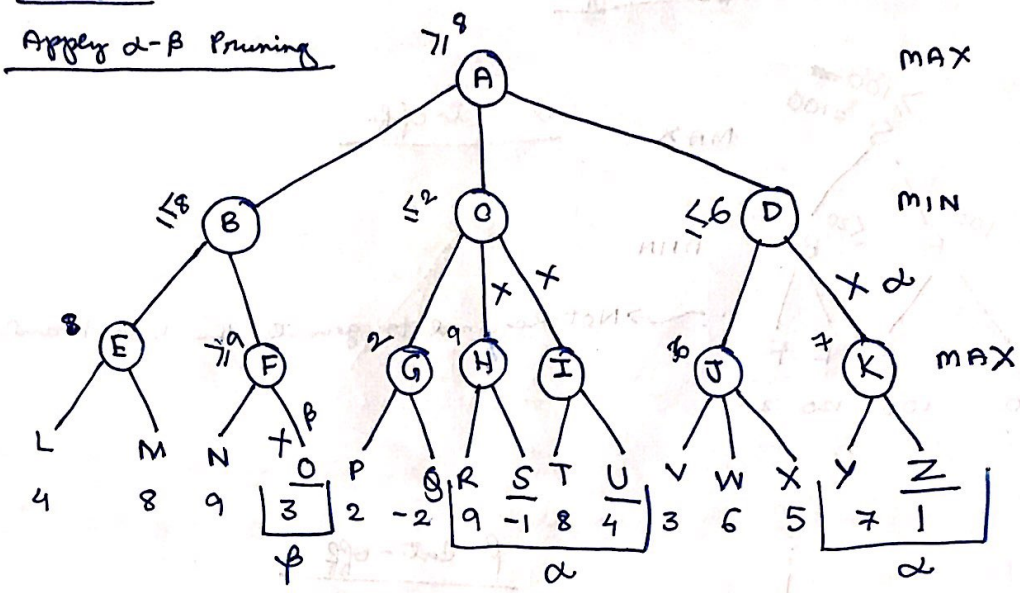
β Cut-off



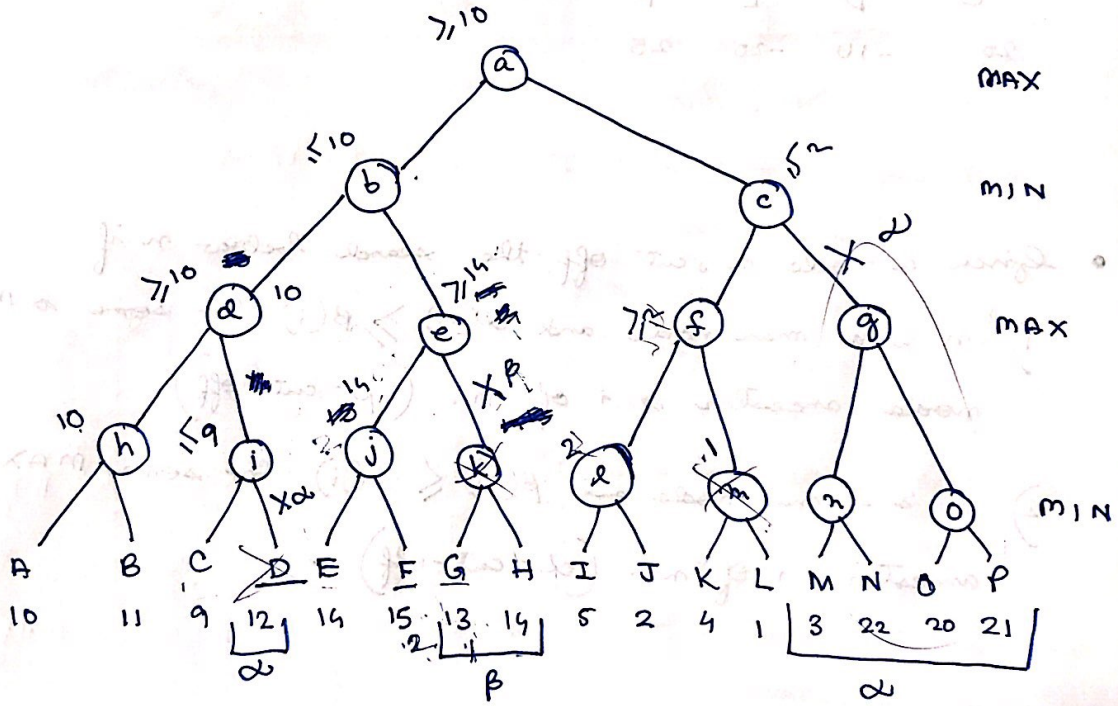
- Given a node n, cut off the search below n if
  1) n is a max node and $\alpha(n) \geq \beta(i)$ for some MIN node ancestor i of n. (β-cut-off)
  2) n is a min node and $\beta(n) \leq \alpha(i)$ for some MAX node ancestor i of n. (α-cut-off)

Example 1:

Apply α-β Pruning



Example 2:



Knowledge Representation (Rich and Knight – Book)

- Representations and Mapping

  2 entities –

  a) Facts

  b) Representation of facts in some chosen format

Example:

Fact: Tommy is a dog

Logical representation: dog(Tommy)

- Approaches to Knowledge Representation
- Inheritable Knowledge
  - Semantic Network
- Inferential Knowledge
  - i) Forward Reasoning
  - ii) Backward Reasoning
  - iii) Resolution
- Procedural Knowledge
- Formalized Symbolic Logic
  - Propositional ~~Calculus~~ Logic
  - First Order Predicate Logic

Logic
- Syntax
- Symantics : whether true or false

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

P=T
Q=T
R=T
} Semantic : True (T)

P=T
Q=T
R=T

F → T

T ∨ T
= T

- Properties of Sentence
  - Satisfiable
  - Valid
  - Contradiction
  - Equivalence
  - Logical Consequence
- Inference Rules
  - Modus Ponens : From P, P → Q infer Q.
  - Modus Tollens
  - Chain Rule
- Resolution in Propositional Calculus

• Resolution works in negation.

- Resolution rule for Propositional calculus :

fact, rule } clause

$\{l\} \cup \Sigma_1$

$\{\neg l\} \cup \Sigma_2$

CNF : $(\{l\} \cup \Sigma_1) \wedge (\{\neg l\} \cup \Sigma_2)$

$\Downarrow$

$\Sigma_1 \cup \Sigma_2$ → inference

Example:

Given set of wffs :

1. BAT_OK 2. ¬MOVES 3. (BAT_OK ∧ LIFTABLE) → MOVES

Wff to be proved : ¬LIFTABLE

From ③, we get 4. ¬BAT_OK V ¬LIFTABLE V MOVES

Negation of the wff to be proved is 5. LIFTABLE

Resolving 5 with 4, 6. ¬BAT_OK V MOVES

Resolving 2 with 6, .. 7. ¬BAT_OK

∴ ① and ⑦ Contradict.

∴ ⑤ is incorrect.

∴ ¬LIFTABLE [Proved]

Steps:

1. Convert the wff to CNF
2. Convert the negation of the wff to be proved, ω, to CNF.
3. Combine all the CNF clauses in a single set Γ.
4. . . . .

- Converting wff to CNF —

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

1. Eliminate implication —

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

2. Reduce the scope of ¬ by using De-Morgan's Law —

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

3. Convert to CNF by associative & distributive laws —

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

4.

$$\{(P \vee \neg R), (\ldots\ldots)\}$$

Examples:

1. $\neg R \rightarrow P$
   $P \rightarrow Q$
   $\neg R \rightarrow Q$
   $R \vee Q$

Resolving $P \vee P$ & $\neg P \vee Q$ yields $R \vee Q$. This can be proved by rule of chaining. So chaining is a special example of resolution.

2. P
   $P \rightarrow Q$
   Q

Resolving R and $\neg R \vee P$ yields P. This can be proved by Modus Ponens. So Modus Ponens can be refered to as a special example of resolution.

- Predicate Calculus
- Atoms and Wffs

- Quantification

Example:

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

Example:

1. Represent the following English sentences in First Order Predicate Logic (FOPL) form:

E1: all employees earning $1400/ more per year pay taxes

E2: some employees are sick today

E3: no employee earns more than the President

$E(x) \rightarrow x$ is an employee

$i(x) \rightarrow$ income of x

$T(x) \rightarrow x$ pay taxes

$GE(u, v) \rightarrow u \geq v$.

E1 $\forall x : \left(E(x) \wedge GE(i(x), 1400)\right) \rightarrow T(x)$

$S(x) \rightarrow x$ is sick

~~$\forall x : (E(x) \wedge S(x))$~~

E2 $\exists x : E(x) \rightarrow S(x)$

$P(y) \rightarrow y$ is the president

E3 $\forall x : E(x) \rightarrow GE(i(P(y)), i(x))$

$\forall x : GE(i(x), i(P(x))) \rightarrow \neg E(x)$

$\nexists x : E(x) \rightarrow GE(i(x), i(P(x)))$

$\forall xy : E(x) \wedge P(y) \rightarrow \sim GE(i(x), i(y))$

2.

i) Marcus was a man

ii) Marcus was a Pompeian

iii) All Pompeian were Roman

iv) Caesar was a ruler.

v) All Romans were either loyal to Caesar or hated him

vi) Everyone is loyal to someone

vii) People only try to assassinate rulers they are not loyal to.

viii) Marcus tried to assassinate Caesar.

i) man(Marcus)

ii) pompeian(Marcus)

iii) $\forall x$: pompeian(x) $\rightarrow$ roman(x)

iv) ruler(caesar)

v) $\forall x$: roman(x) $\rightarrow$ loyalto(x, caesar) $\vee$ hated(x, caesar)

vi) $\forall x : \exists y$: loyalto(x, y)

vii) $\forall xy$: person(x) $\wedge$ ruler(y) $\wedge$ trytoassassinate(x, y) $\rightarrow \neg$ loyalto(x, y)

viii) trytoassassinate(Marcus, Caesar)

Converting arbitrary wff to CNF

- Pollen Function
  ↓
  Needed to eliminate the Existential Quan

Unification

- Name of the predicate as well as the args of the predicate must be considered.

  eg. man(john) & ¬man(john) is a contradiction

  man(John) & ¬man(Spot) is not.

- Constant can be matched with a constant

  - P(x,x) & P(y,z) to be unified.

    — Substitute y for x.

    P(y,y) , P(y,z)

    — Substitute z for y.

    P(z,z), P(z,z)

  - Resolve the following clauses:

    1. Man(Marcus)
    2. ¬Man(x_1) ∨ Mortal(x_1)

    Substitute x_1 for Marcus

    Man(Marcus) and Man(x_1) can be unified.

    by this substitution.

∴ the resolvent is Mortal(Marcus).

Example

Consider the FOPL forms in the answer of ~~Question~~ Question 2.

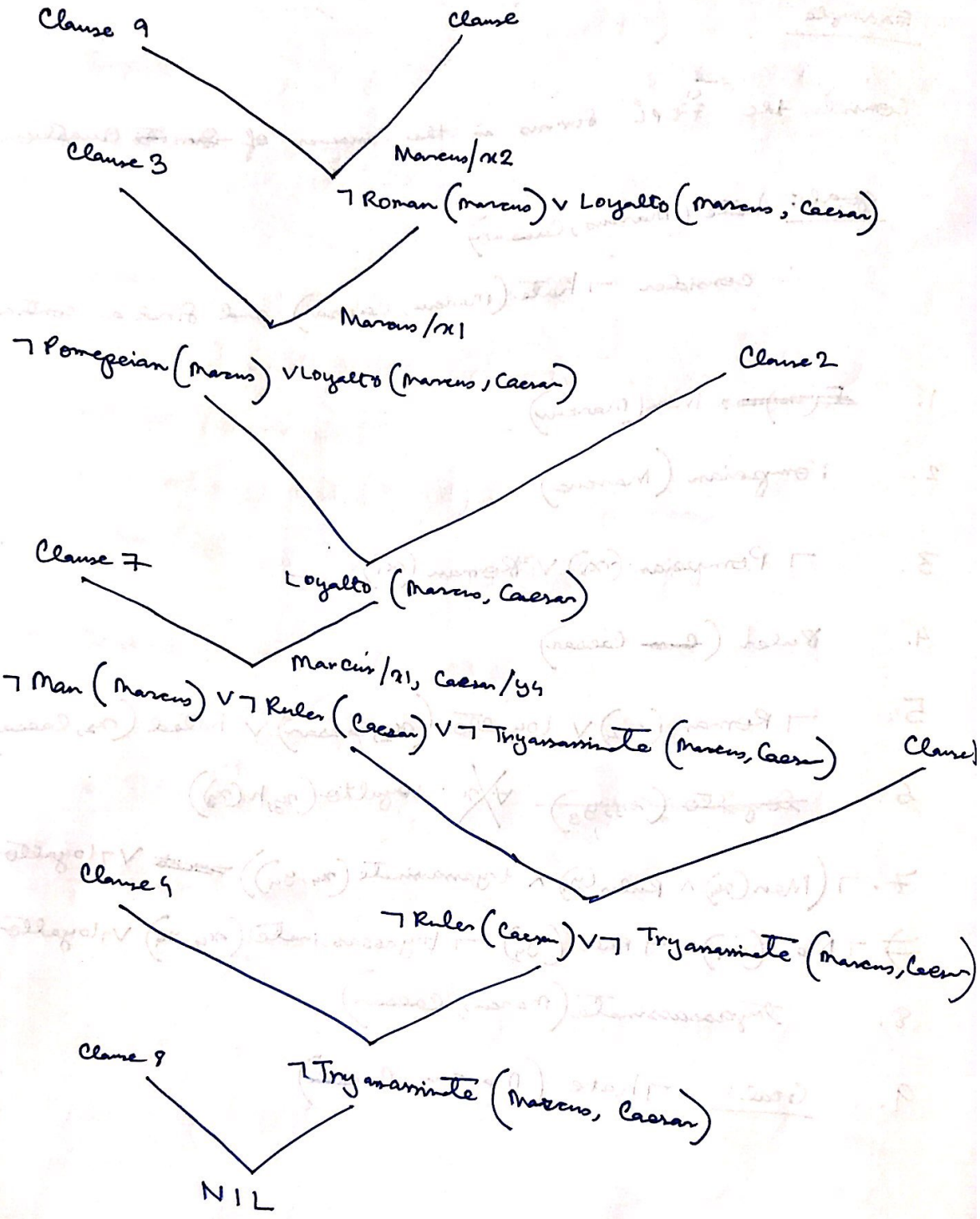
goal: hate(Marcus, Caesar)

∴ consider ¬hate(Marcus, Caesar) and find a contradiction.

1. ~~F(x)~~ → Man(Marcus)
2. Pompeian(Marcus)
3. ¬Pompeian(x_1) ∨ Roman(x_1)
4. Ruler(~~Caesar~~ Caesar)
5. ¬Roman(x_2) ∨ loyalto(x_2, Caesar) ∨ hated(x_2, Caesar)
6. ~~loyalto(x_3, y_3)~~ ~~∀x~~ : loyalto(x_3, h.(x_3))
7. ¬(Man(x_4) ∧ Ruler(y_4) ∧ tryassassinate(x_4, y_4)) ~~→~~ ∨ ¬loyalto(x_4, y_4)

⇒ ¬Man(x_4) ∨ ¬Ruler(y_4) ∨ ¬tryassassinate(x_4, y_4) ∨ ¬loyalto(x_4, y_4)

8. tryassassinate(Marcus, Caesar)
9. Goal: ¬hate(Marcus, Caesar)



∴ the goal is true .

Example:

Everyone who enters in a theatre has bought a ticket.
Person who does not have money can't buy ticket.
Vinod enters a theatre.

$T(x) \rightarrow x$ enters in a theatre

$M(x) \rightarrow x$ has bought a ticket

$P(x) \rightarrow x$ has money.

i) $\forall x : T(x) \rightarrow M(x)$

ii) $\forall x : \neg P(x) \rightarrow \neg M(x)$

iii) $T(\text{Vinod})$

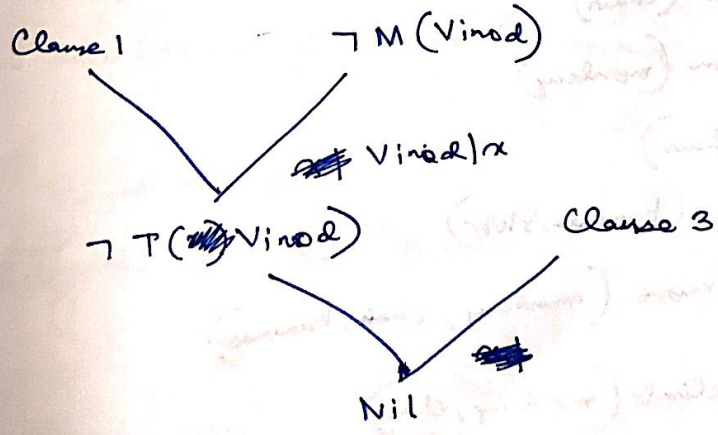
CNF:

i) $\neg T(x) \vee M(x)$.

ii) $P(x) \vee \neg M(x)$

iii) $T(\text{Vinod})$

iv) Goal: $\neg M(\text{Vinod})$



Solution (Alternative Way):

FOPL:

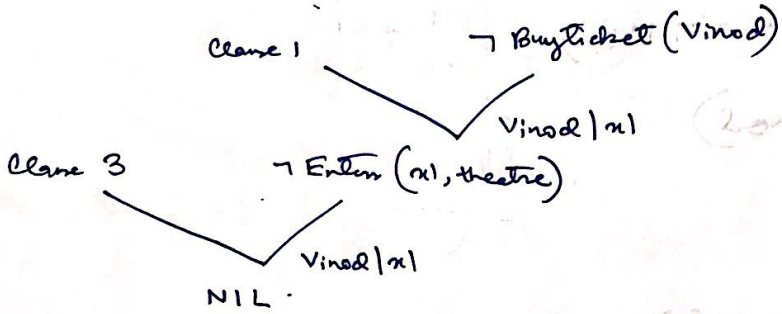
1. $\forall x : \text{Enters}(x, \text{theatre}) \rightarrow \text{BuyTicket}(x)$
2. $\forall x : \text{Person}(x) \wedge \neg \text{HaveMoney}(x) \rightarrow \neg \text{BuyTickets}(x)$

$\text{Enters}(\text{Vinod}, \text{theatre})$.

CNF:

1. $\neg \text{Enters}(x1, \text{theatre}) \vee \text{BuyTicket}(x1)$:
2. $\neg \text{Person}(x2) \vee \text{HaveMoney}(x^2) \vee \neg \text{BuyTicket}(x2)$
3. $\text{Enters}(\text{Vinod}, \text{theatre})$

Goal: $\neg \text{BuyTicket}(\text{Vinod})$



Example:

Monkey Banana Problem

CNF Representation:

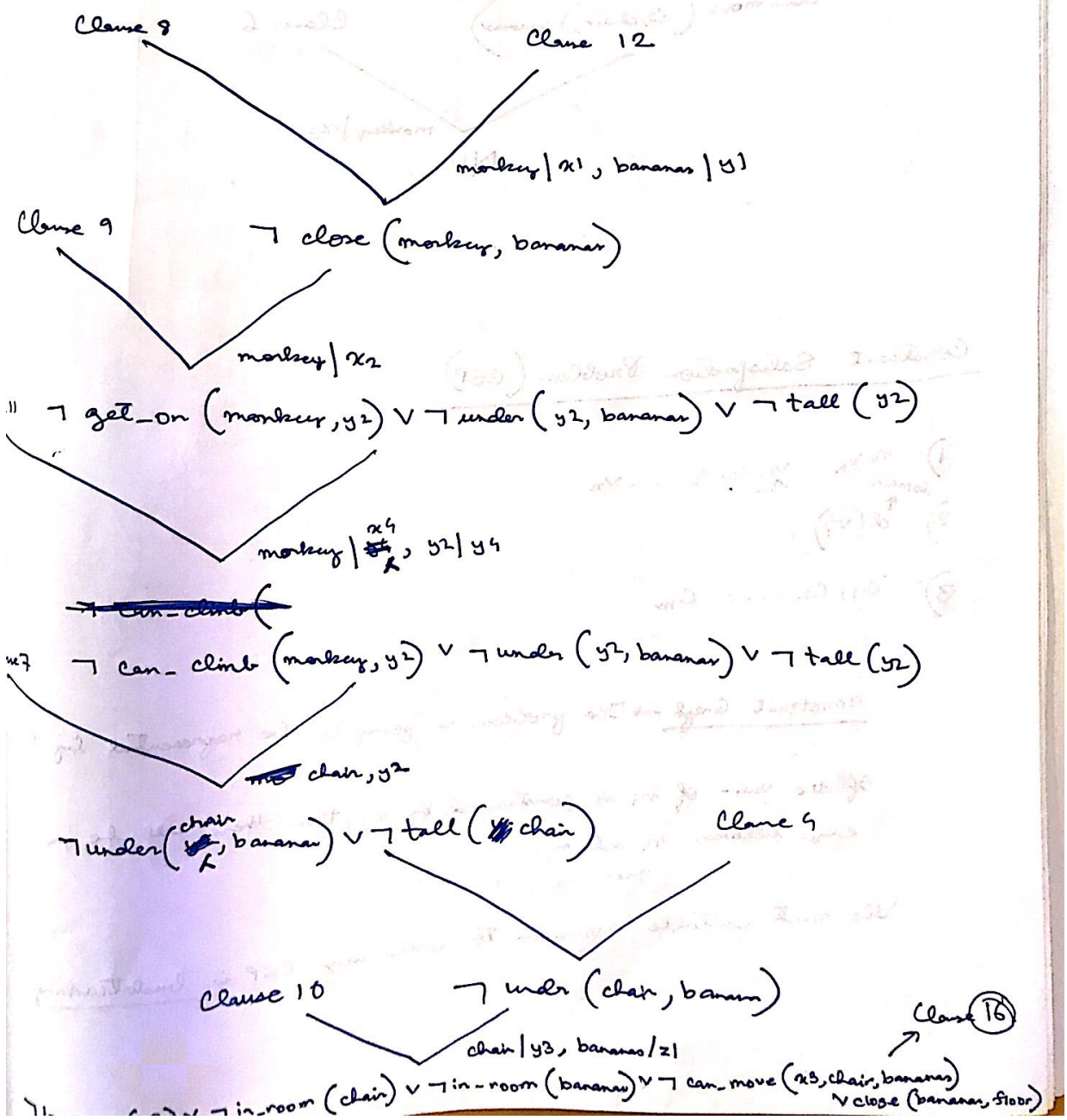
1. in_room (bananas)
2. in_room (chair)
3. in_room (monkey)
4. tall (chair)
5. ¬ close (bananas, floor)
6. can_move (monkey, chair, bananas)
7. can_climb (monkey, chair)
8. ¬ close (x1, y1) ∨ can_reach (x1, y1)

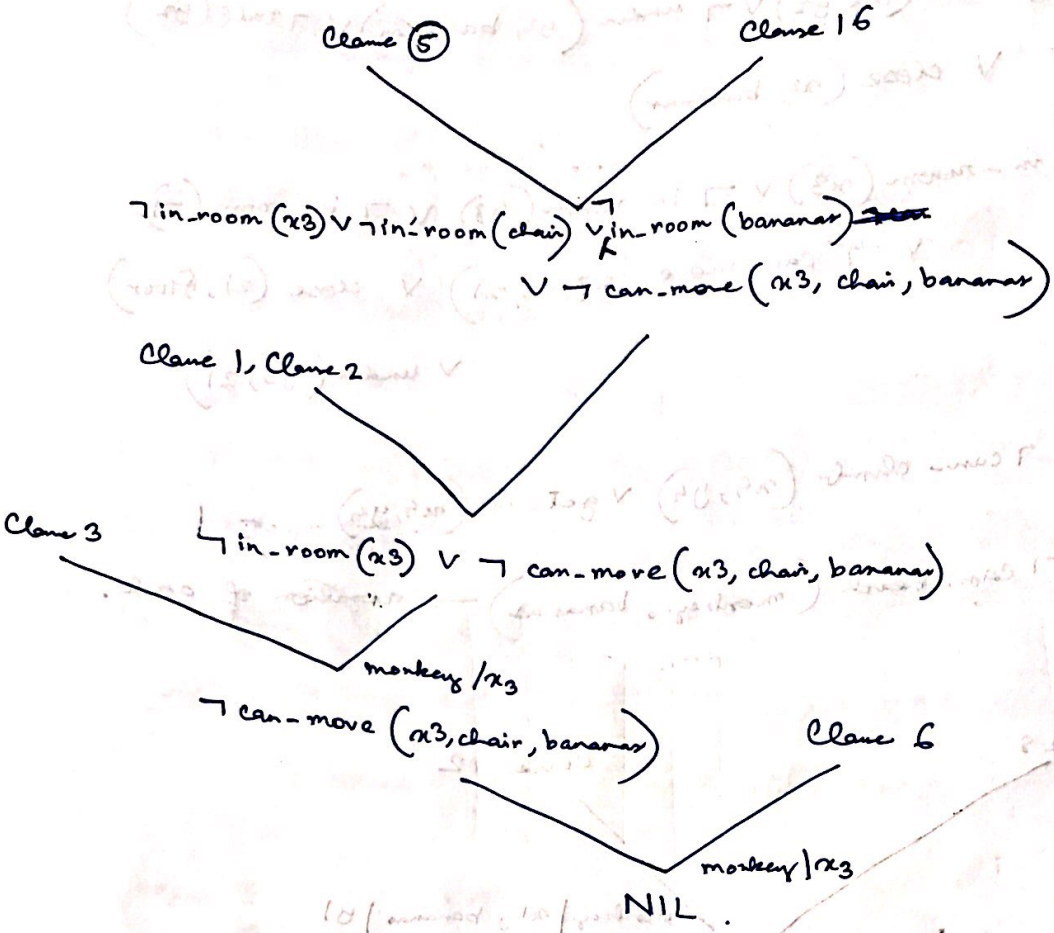
9. $\neg get\_on(x2, y2) \vee \neg under(y2, bananas) \vee \neg tall(y2) \vee close(x2, bananas)$

10. $\neg in\_room(x3) \vee \neg in\_room(y3) \vee \neg in\_room(z1) \vee \neg can\_move(x3, y3, z1) \vee close(z1, floor) \vee under(y3, z1)$

11. $\neg can\_climb(x4, y4) \vee get\_on(x4, y4)$

12. $\neg can\_reach(monkey, bananas)$ → negation of goal.





Constraint Satisfaction Problem (CSP)

1) n V_N $V_1, V_2, V_3 \ldots V_n$ (domain)
2) $d(v_i)$
3) $C_1, C_2, \ldots C_m$

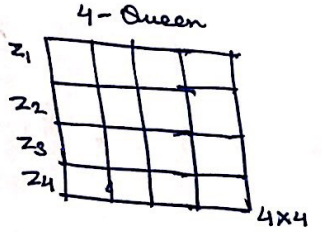
Constraint Graph → The problem is going to be represented by this graph.

If the value of n_i is constrained by n_j, then there will be an edge between n_i and n_j.

The most suitable approach to solve any CSP is backtracking.

N- Queen Puzzle

Let N = 4



Use 4 variables.

z_i → ith row

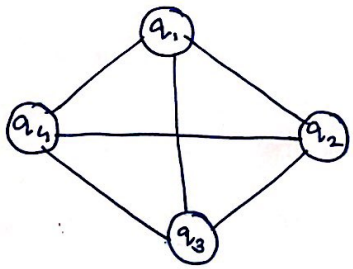
$z_i = j$ → queen at the ith row is placed in the jth column

$z_i \neq z_j$ → vertical constraint

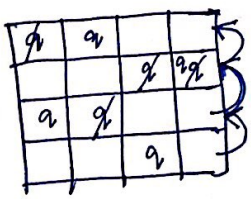
$|i - j| \neq |z_i - z_j|$ → diagonal constraint

Constraint Graph

For 4 variables, there will be 4 nodes.



Solution:



$z_1 = 2$
$z_2 = 4$
$z_3 = 1$
$z_4 = 3$

} 1 Solution

Another solution is by Forward checking.

	3	4
z_1	9	
z_2	x x x	9
z_3	9 x x	x
z_4	x x 9	x

Crossword Puzzle

→ Blocked

The Nos 1, 2 etc. mean that a word can be started here.

	1	2	3	4	5
1	1		2		3
2	#	#		#	
3	#	4		5	
4	6	#	7		
5	8				
6		#	#		#

List of words

AFT	LASER
ALE	LEE
EEL	LINE
HEEL	SAILS
HIKE	SHEET
HOSES	STEER
KEEL	TIE
KNOT	

Variables

1A (Across) = { all 5 letter words }

2D = (down)

Constraints

1A[3] = 2D[1]

1A[5] = 3D[1]

4A[3] = 5D[4]

6D[2] = 8A[1]

7D[2] = 8A[3]

~~3D[3] = 5D[2]~~

3D[3] = 4A[4]

3D[4] = 7A[3]

3D[5] = 8A[5]

Solution:

H	O	S	E	S
#	#	A	#	T
#	H	I	K	E
A	#	L	E	E
L	A	S	E	R
E	#	#	L	

	1	2	3	4	5	
1	IS	A	21	L	3	S
2	#	#		#	A	T
3	#	4		5	S	E
4	6A	#	A7	L	E	E
5	8	A	S	E	R	R
6	E	#	#		#	

Reasoning with Uncertainty

$$P(v_j, v_i) = P(v_j \mid v_i) P(v_i)$$
$$P(v_i, v_j) = P(v_i \mid v_j) P(v_j)$$
} Same

$$\frac{}{k=2}$$

$$P(v_i, v_i) = P(v_i)\, P($$

$$P(v_i \mid v_j) = \frac{P(v_j \mid v_i) P(v_i)}{P(v_j)}$$

(Bayes' Rule)

1. Consider the following joint probabilities:

$P(P, Q, R) = 0.3$, $P(P, Q, \neg R) = 0.2$, $P(P, \neg Q, R) = 0.2$,

$P(P, \neg Q, \neg R) = 0.1$, $P(\neg P, Q, R) = 0.05$,

$P(\neg P, Q, \neg R) = 0.1$, $P(\neg P, \neg Q, \neg R) = 0.05$,

$P(\neg P, \neg Q, \neg R) = 0.0$

Calculate $P(Q \mid \neg R)$.

$P(Q|\neg R)$

$= \alpha P(P, Q, \neg R) + P(\neg P, Q, \neg R)$

$= 0.2 + 0.1$

$= 0.3$

$\therefore \quad P(\neg R) = 0.2 + 0.1 + 0.1$

$= 0.4$

$\therefore \quad P(Q|\neg R)$

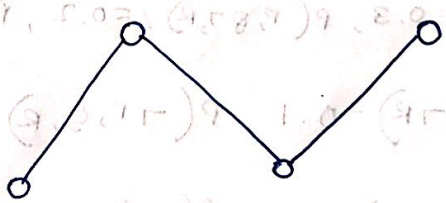
$= \frac{0.3}{0.4} = 0.75$ (Ans)

Conditional Independence

- Bayes Networks

  - A DAG

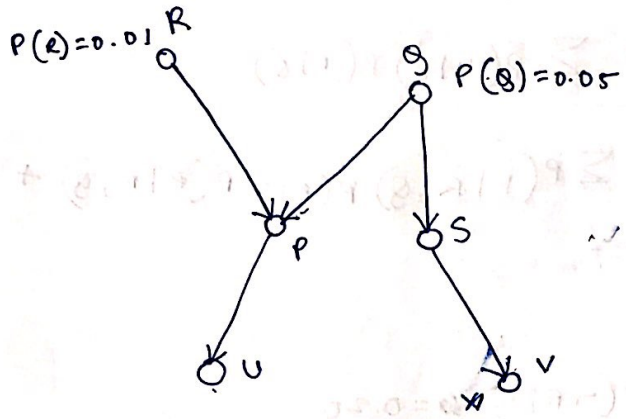
1)



$P(M|L) = P(M|B,L)\,P(B|L) + P(M|\neg B,L)\,P(\neg B|L)$

$= 0.855$

2)



Calculate $P(Q|U)$.

$P(Q|U) = P(Q|P,U)\,P(P|U) +$

$$P(Q|U) = \frac{P(U|Q)\,P(Q)}{P(U)}$$

$$= \frac{(\quad - \quad) \times 0.05}{0.9}$$

=

$P(U|Q) = P(U|$

$P(P|Q) = P(P|R,Q)\,P(R|Q) + P(P|\neg R,Q)\,P(\neg R|Q)$

$= 0.95 \times$

$$P(U|Q) = \sum P(U|P)\,P(P|Q)$$

$$P(P|Q) = \sum P(P|R,Q)\,P(R) = P(P|R,Q) + P(P|\neg R,Q)\,P(\neg R)$$

$$= 0.80$$

So, $P(\neg P|Q) = 0.20$

- $P(U|Q) = P(U|P)\,P(P|Q) + P(U|\neg P)\,P(\neg P|Q) = 0.60$

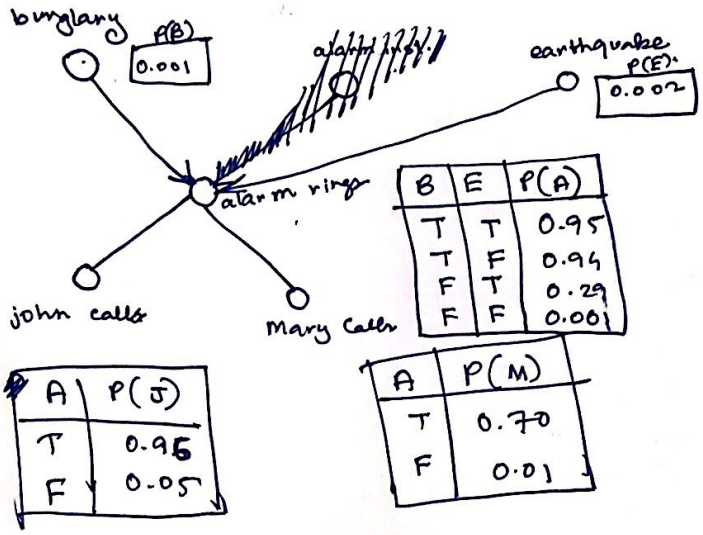
$$P(Q|U) = k * 0.60 * 0.05 = k * 0.03 \quad \text{--- (1)}$$

Similarly, $P(\neg Q|U) = k * P(U|\neg Q)\,P(\neg Q)$

Now, $P(U|\neg Q) = \sum_P P(U|P)\,P(P|\neg Q) = 0.21$

Therefore, $P(\neg Q|U) = k * 0.21 \ldots$

Example:



B	E	P(A)
T	T	0.95
T	F	0.94
F	T	0.29
F	F	0.001

A	P(J)
T	0.95
F	0.05

A	P(M)
T	0.70
F	0.01

Find out $P(J, M, A, \neg B, \neg E)$

Ans: 0.000628

$P(J, M, A, \neg B, \neg E)$

$= P(J|A) * P(M|A) * P(A|\neg B, \neg E) * P(\neg B) * P(\neg E)$

$= 0.96 * 0.7 * 0.001 * 0.$