AI-Series-2-Important-Topics

- AI-Series-2-Important-Topics
  - 1. Arc consistency and domain size
    - Example:
    - Reducing Domain Size:
    - Advantage:
      - Arc consistency Algorithm
        - AC-3 algorithm for arc consistency
        - Function based algorithm
        - Complexity of AC-3
  - 2. Game definition
    - Adversarial Search
    - Types of games in AI
      - Perfect information
      - Imperfect information
      - Deterministic Games
      - Non Deterministic Games
  - 3. Most general unifier finding problem
    - Unification
    - Example
    - Conditions for Unification
    - Unification Algorithm
    - Unification Example
  - 4. Modus ponens and modus tollens
  - 5. Propositional logic
    - Propositional Logic in Artificial Intelligence
      - Example Propositions:

1. Arc consistency and domain size

- A variable is **arc consistent** if every value in its domain satisfies the variable's binary constraints.
- If variable xi is arc consistent with respect to another variable xj, every value in xi's domain must correspond to a value in xj's domain that satisfies the constraint between them.

Example:

- Suppose the constraint is $y = x^2$ where the domains of x and y are sets of digits.
- Valid pairs of values are: $(0, 0), (1, 1), (2, 4), (3, 9)$.
- To make x arc consistent with respect to y:

    - We reduce x's domain to $0, 1, 2, 3$ because these are the only values of x that have corresponding valid values in y.
- To make y arc consistent with respect to xxx:
    - We reduce yyy's domain to {0,1,4,9} because these are the squares of x's values.

Reducing Domain Size:

- Arc consistency reduces the domain size by eliminating values that cannot participate in valid solutions, which makes the problem easier to solve by focusing on feasible options.

Advantage:

- Arc consistency helps detect conflicts (failures) earlier than simpler methods like forward checking.

Arc consistency Algorithm

AC-3 algorithm for arc consistency

1. A queue data structure is used.
2. Initially, the queue contains all arcs (pairs of variables) in the CSP (constraint satisfaction problem).
3. The algorithm pops an arbitrary arc (xi,xj) from the queue and checks if xi is arc consistent with xj.
4. If the domain of xi remains unchanged, the algorithm moves to the next arc.
5. If the domain of xi is reduced:
    1. The neighbors of xi(variables connected by constraints) are added back to the queue.
    2. For example, if $y = x^2$
        - x's domain could be {0,1,2,3,4,5,...}
        - y's domain is smaller because it's based on x^2, so y's domain is {0,1,4,9,...}
6. If the domain of xi is reduced to nothing, the algorithm detects a failure and returns.
7. Otherwise, the process continues until all variables are arc consistent, resulting in a CSP that is equivalent to the original but potentially easier to solve.

Function based algorithm

```
def Revise(CSP, xi, xj):
    revised = False
```

```
    for x in domain[xi]:
        if no value y in domain[xj] satisfies the constraint between xi and
xj:
            domain[xi].remove(x)
            revised = True
    return revised
```

Complexity of AC-3

- Best Case -> O(d^3)
- Worst Case -> O(d^3)

2. Game definition

Adversarial Search

- It is a type of search where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.
  - We plan to win the game
  - People who play with us are planning against us
- In game playing, instead of searching the solution with one agent, more than one agent is involved
  - This is called multi agent environment
  - Each agent are opponent to each other.
  - Each agent should closely analyze the actions of the other agents and perform actions based on others performance
- Searches in which two or more players with conflicting goals and trying to explore the same space for the solution is called **Adversarial search** often known as **Games

Types of games in AI

Perfect information

- A game with perfect information is that in which agent can look into the complete board
- Agent have all the info of the game and they can see each others move.
- Eg: Chess, checkers

Imperfect information

- If in a game agent do not have any info about the games, such type of game is called imperfect information.
- Eg: Battle ship, tic tac toe

Deterministic Games

- Deterministic games are those games which follows strict pattern and set of rules for the games
- Eg: Chess

Non Deterministic Games

- Non deterministic games are those which is mainly based on luck or chances. Such games are called stochastic games
- Eg: Monopoly, dice or card games

3. Most general unifier finding problem

Unification

- The process of finding a substitution for predicate parameters is called unification
- Unification is a process of making two different logical atomic expressions identical by finding a substitution.
- Unification depends on the substitution process
- It takes 2 literals as input and makes them identical using substitution
- We need to know
  - That 2 literals can be matched
  - The substitution is that makes the literals identical
- There is a simple algorithm called unification algorithm that does this.

Example

- Lets say there are two different expressions, P(x,y) and P(a,f(z)).
- In this example, we need to make both above statements identical to each other.

- For this we will perform the substitution
  - P(x,y) .... (i)
  - P(a,f(z))......(ii)
- Substitute x with a and y with f(z) in the first expression, and it will be represented as a/x and f(z)/y
- With both the substitutions, the first expression will be identical to the second expression and substitution set will be [a/x,f(z)/y]
- The substitution variables are called **Most General Unifier or MGU**

Conditions for Unification

- Following are some basic conditions for unification
  - Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified
  - Number of arguments in both expressions must be identical
  - Unification will fail if there are 2 similar variables present in the same expression

Unification Algorithm

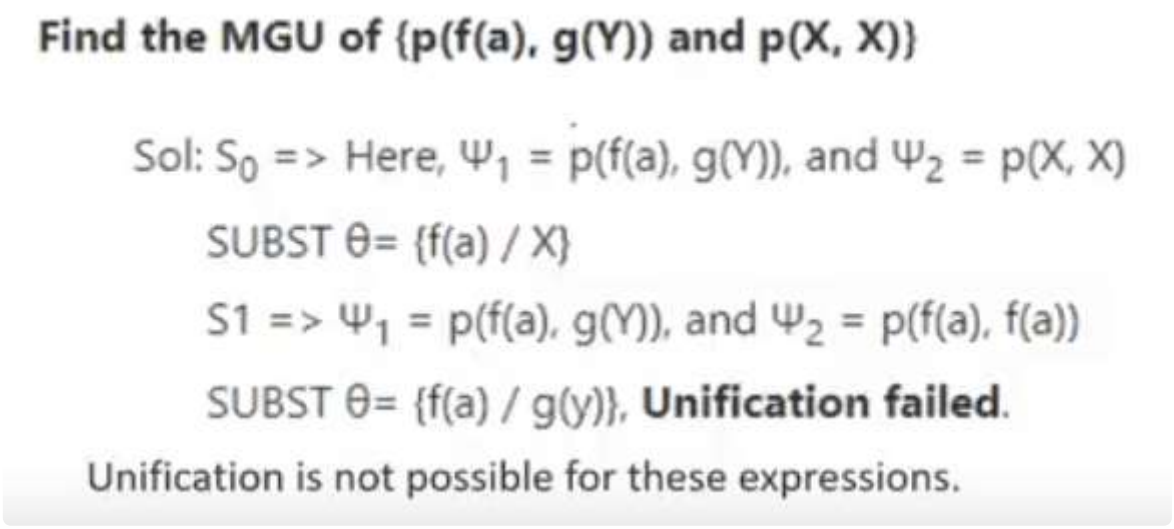
- Initialise the substitution set to be empty
- Recursively unify atomic sentences
  - Check for identical expression match
  - If one expression is a variable vi and the other is a term ti which does not contain variable vi, then
    - Subtitute ti/vi in the existing subtitutions
    - Add ti/vi to the substitution set list
    - If both the expressions are functions, then function name must be similar and the number of arguments must be same in both the expression

Unification Example

- Example 1

- P(x) and P(y): substitution = (x/y) "substitute x for y"
- P(x,x) and P(y,z): (z/y)(y/x) y for x, then z for y
- P(x,f(y)) and P(Joe,z): (Joe/x, f(y)/z)
- P(f(x)) and P(x): can't do it!
- P(x) ∨ Q(Jane) and P(Bill) ∨ Q(y): (Bill/x, Jane/y)

- Example 2



  - p(f(a),g(Y)) and p(X,X)
  - f(a) = X
  - Then it becomes p(f(a),f(a))
  - So its not possible to get p(f(a),g(y))
  - Hence unification failed
- Example 3

2. Find the MGU of {p(b, X, f(g(Z))) and p(Z, f(Y), f(Y))}

Here, Ψ_1 = p(b, X, f(g(Z))) , and Ψ_2 = p(Z, f(Y), f(Y))

S_0 => { p(b, X, f(g(Z))); p(Z, f(Y), f(Y))}

SUBST θ={b/Z}

S_1 => { p(b, X, f(g(b))); p(b, f(Y), f(Y))}

SUBST θ={f(Y) /X}

S_2 => { p(b, f(Y), f(g(b))); p(b, f(Y), f(Y))}

SUBST θ= {g(b) /Y}

S_2 => { p(b, f(g(b)), f(g(b)); p(b, f(g(b)), f(g(b))} **Unified Successfully.**

And Unifier = { b/Z, f(Y) /X , g(b) /Y}.

- Example 4

3. Find the MGU of {p (X, X), and p (Z, f(Z))}

Here, Ψ_1 = {p (X, X), and Ψ_2 = p (Z, f(Z))

S_0 => {p (X, X), p (Z, f(Z))}

SUBST θ= {X/Z}

S1 => {p (Z, Z), p (Z, f(Z))}

SUBST θ= {f(Z) / Z}, **Unification Failed.**

- Example 5

4. Find the MGU of UNIFY(prime (11), prime(y))

Here, Ψ_1 = {prime(11) , and Ψ_2 = prime(y)}

S_0 => {prime(11) , prime(y)}

SUBST θ= {11/y}

S_1 => {prime(11) , prime(11)} , **Successfully unified.**

Unifier: {11/y}.

- Example 6

5. Find the MGU of Q(a, g(x, a), f(y)), Q(a, g(f(b), a), x)}

Here, Ψ_1 = Q(a, g(x, a), f(y)), and Ψ_2 = Q(a, g(f(b), a), x)

S_0 => {Q(a, g(x, a), f(y)); Q(a, g(f(b), a), x)}

SUBST θ= {f(b)/x}

S_1 => {Q(a, g(f(b), a), f(y)); Q(a, g(f(b), a), f(b))}

SUBST θ= {b/y}

S_1 => {Q(a, g(f(b), a), f(b)); Q(a, g(f(b), a), f(b))}, **Successfully Unified.**

Unifier: [a/a, f(b)/x, b/y].

- Example 7

6. UNIFY(knows(Richard, x), knows(Richard, John))

Here, Ψ_1 = knows(Richard, x), and Ψ_2 = knows(Richard, John)

S_0 => { knows(Richard, x); knows(Richard, John)}

SUBST θ= {John/x}

S_1 => { knows(Richard, John); knows(Richard, John)}, **Successfully Unified.**

Unifier: {John/x}.

4. Modus ponens and modus tollens

1.	**Modus ponens**	p $p \rightarrow q$ $\therefore q$
2.	**Modus tollens**	$\neg q$ $p \rightarrow q$ $\therefore \neg p$

5. Propositional logic

Propositional Logic in Artificial Intelligence

Propositional logic (PL) is a simple form of logic used in AI for representing facts or knowledge in a clear, mathematical way. A **proposition** is just a statement that can either be **true** or **false**—nothing more, nothing less.

Example Propositions:

- "It is Sunday."
- "3 + 3 = 7" (false).
- "5 is a prime number" (true).

Key Features of Propositional Logic:

- **Boolean Logic:** It works with two values—**true** (1) and **false** (0), so it's often called boolean logic.
- **Symbols:** We use letters like A, B, or C to represent these propositions.
- **Logical Operators:** Connect propositions together, like "AND," "OR," or "NOT." For example, "It is raining **AND** the street is wet" combines two propositions.
- **Tautology and Contradiction:** A formula that's always true is a **tautology**, and one that's always false is a **contradiction**.

Syntax of Propositional Logic:

There are two main types of propositions:

1. **Atomic Propositions:** The simplest form, which stands alone (e.g., "2 + 2 = 4").
2. **Compound Propositions:** These are built by combining atomic propositions using logical connectives (e.g., "It is raining **and** the street is wet").

Drawbacks of Propositional Logic:

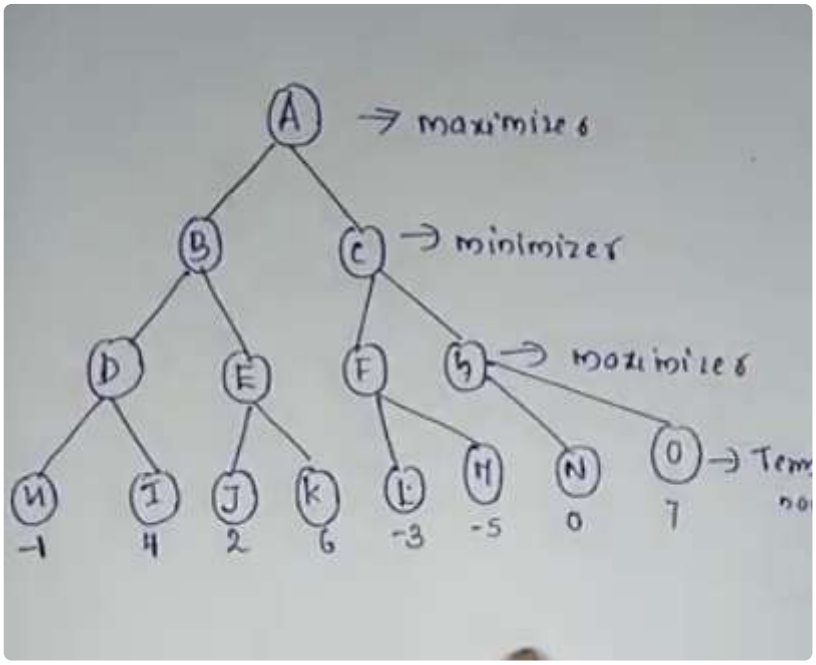
1. **Limited Expressiveness:** Propositional logic can only represent simple facts, not complex relationships. For example, you can't say "All dogs are animals" in propositional logic.
2. **No Variables:** Unlike other logics (like predicate logic), propositional logic can't handle variables like "x" that allow for more detailed and flexible statements.
3. **Scalability Issues:** As the number of propositions grows, the size of the truth tables (used to evaluate propositions) grows rapidly, making it hard to manage for larger problems.

6. Minimax Algorithm

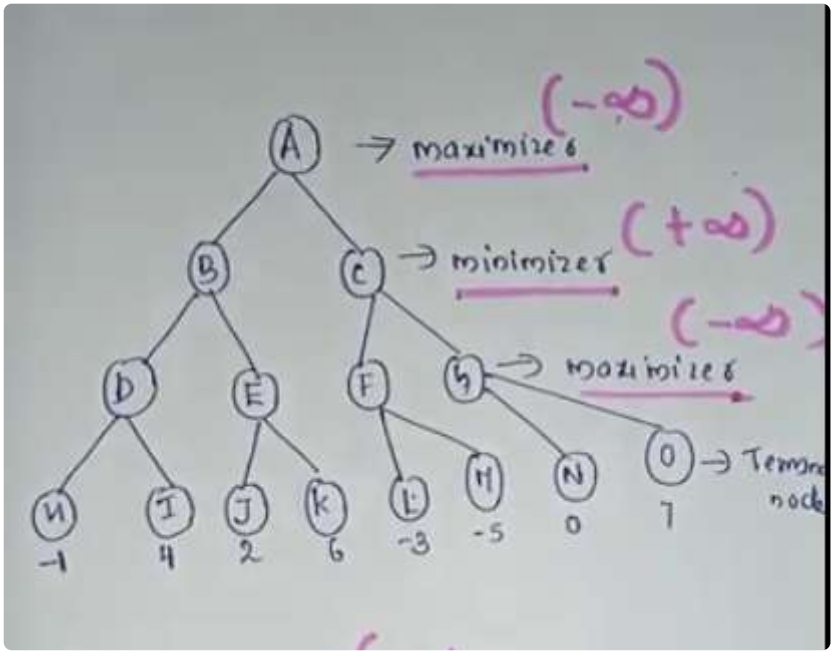
- **Mini-Max Algorithm** is a backtracking algorithm which is used in decision making and to get game strategy.
- It provides optimal move to all the players
- The only condition is that success is only acheived by a single player
- Minimax algorithm is mostly used for game playing in AI, **such as chess and various 2 player games.**
- In this algorihtm two players play the game, one is called **MAX and other is called MIN**

- The player who got minimum value and player who got maximum value will get a optimised value
- This algorithm perform **depth-first search algorithm**

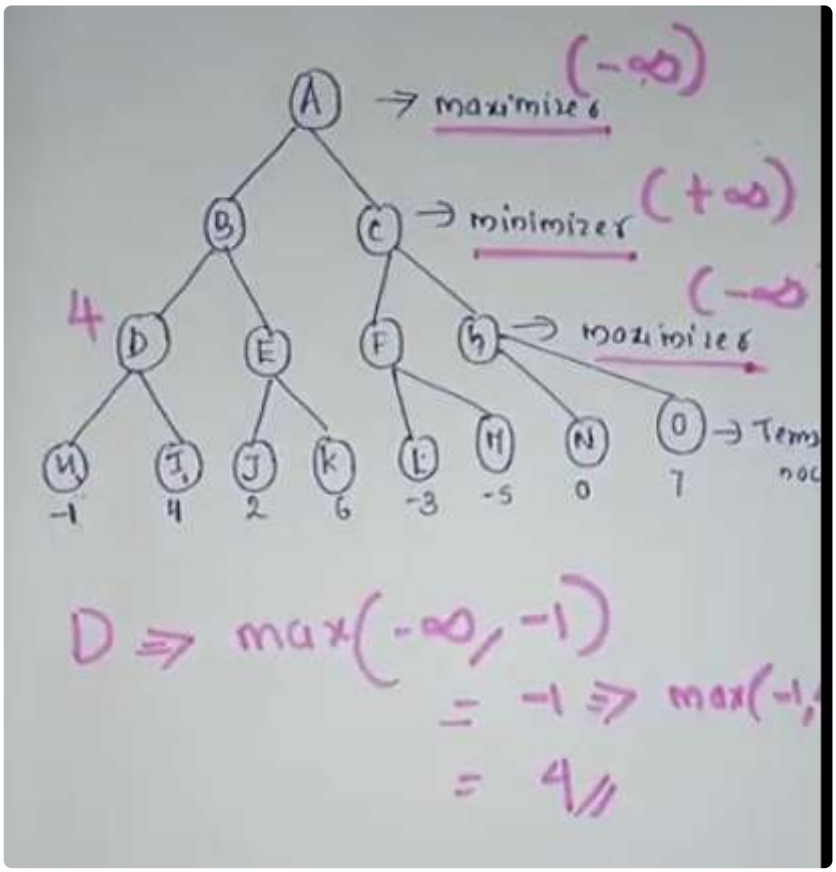
Implementation (Game Tree)



- The above is a game tree.
  - The first level node we can name as maximizer
  - 2nd level can be named minimizer
  - 3rd level is named maximizer
  - The last level are terminal nodes
- We can see that all terminal nodes have values
  - But the other nodes at level 1,2 and 3 doesnt have them
  - We need to assign the values to them

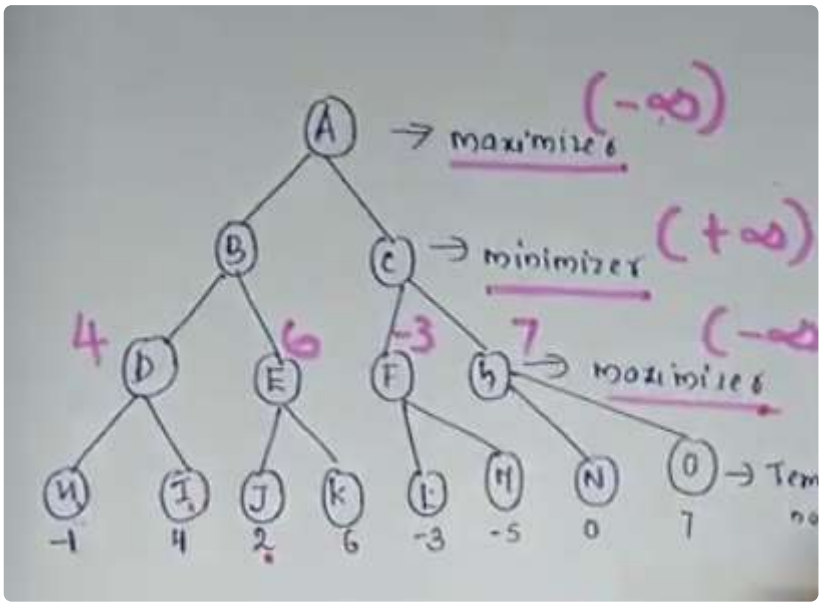


- We will be denoting + Infinity for maximizer
- And - infinity for minimizer

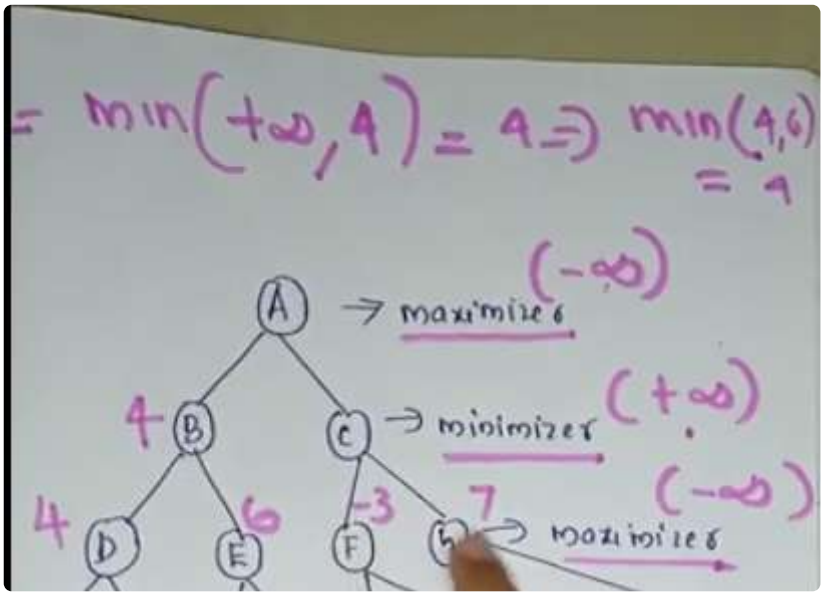


- Lets find the value of D first
  - We can see that D is in maximizer, which is - infinity
- We will first take the child of D with the least value and compare it with - infinity

- -1 is the child of D with least value
  - Doing max comparison
  - D => max(-infinity,-1)
  - The larger number is -1
- Now we need to take the other child of D which is larger
  - Doing max comparison
  - D => max(-1,4)
  - **The value is 4**

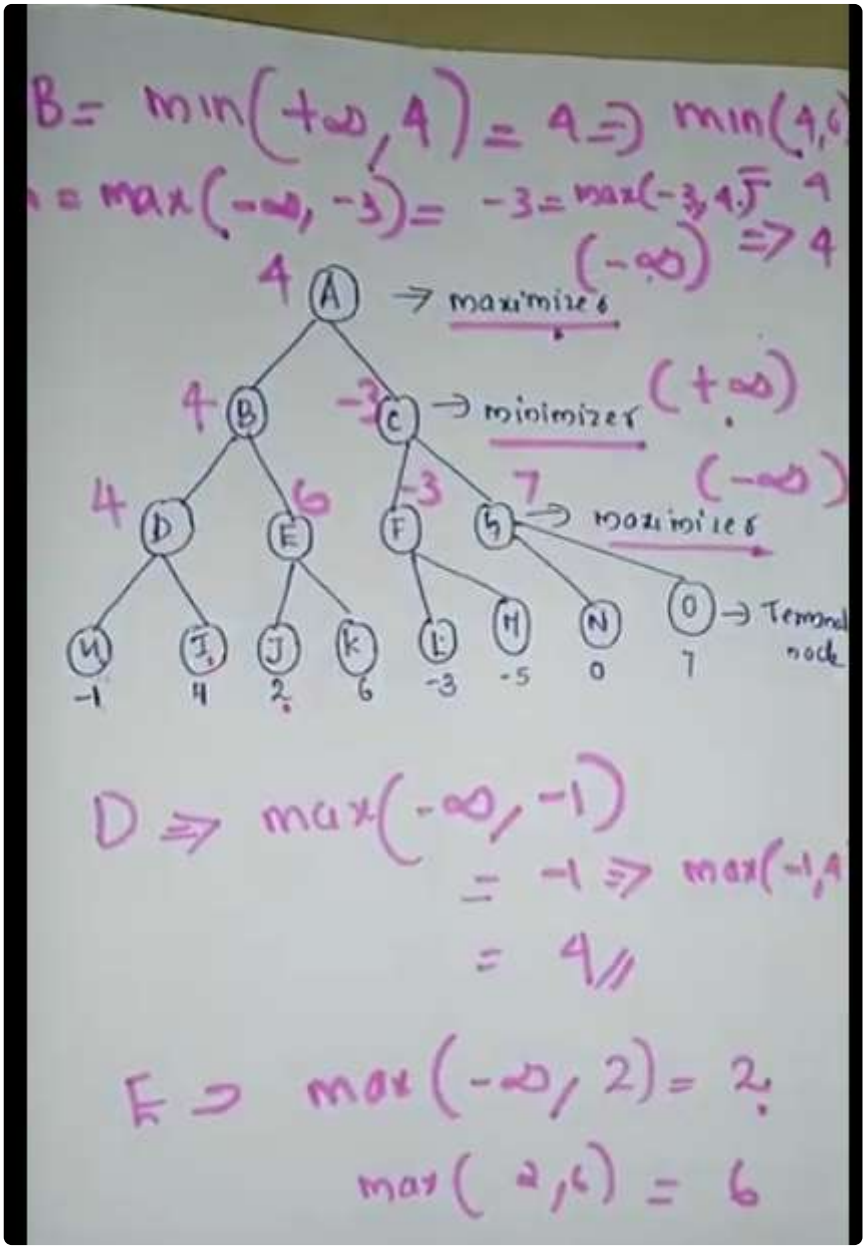


- Similarly, we can find the values of all the nodes in the third level



- Now lets find the values in the 2nd level, lets start with B

    - We can see that its a minimzer, with +infinity
- We have to take the smaller child of B and compare it with +infinite
    - The minimum of +infinite and 4 is 4.
    - Now compare this 4, with the other child of B
    - min(4,6) is 4



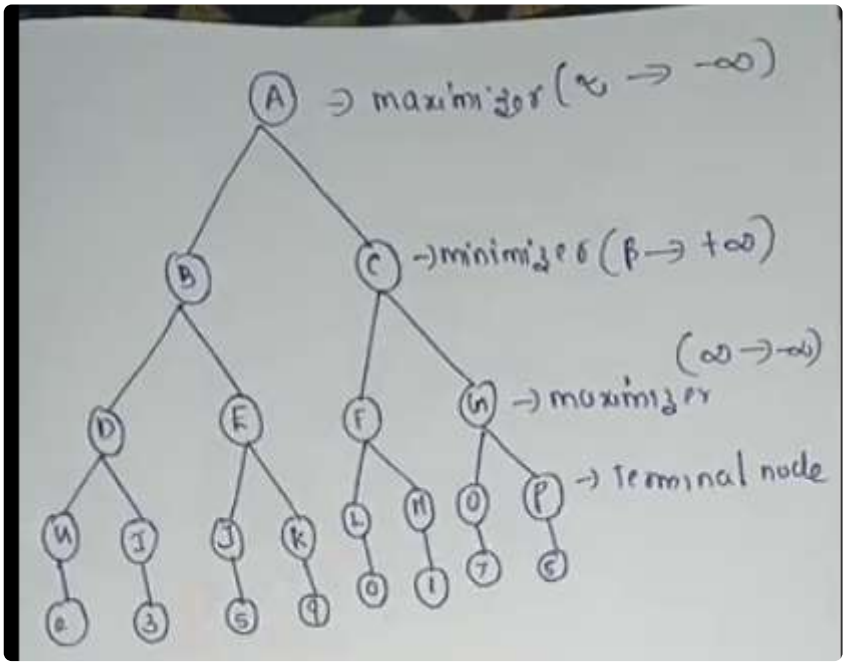
- Similarly, we will get remaining values on the 2nd level
- For the first level, its the same as the 3rd level

7. Alpha beta pruning problem

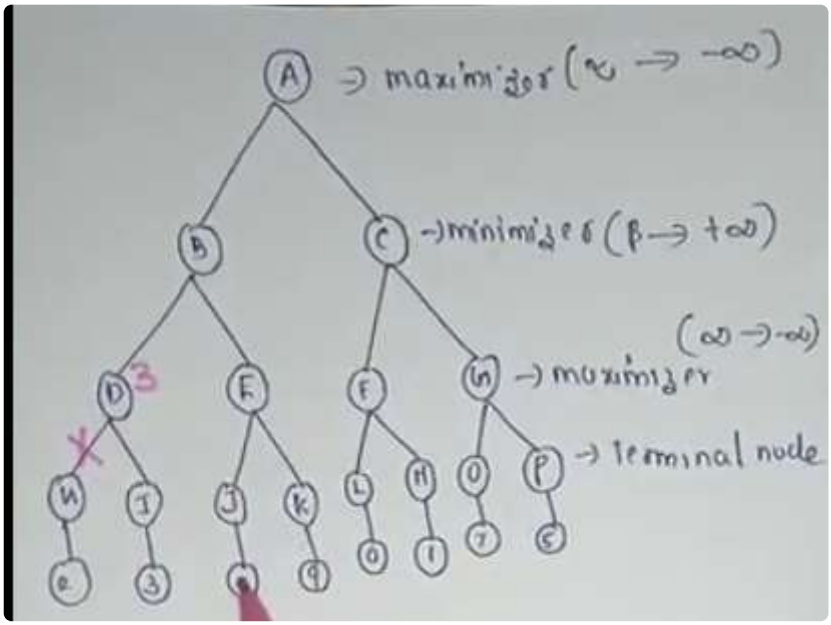
Alpha-Beta Pruning

- Its a **modified version of mini-max algorithm**
- In this Alpha-Beta pruning algorithm, two threshold values are considered that is **alpha and beta**
- Backtracking is also included
- This algorithm does not consider all the nodes it just considered the nodes which satisifes certain conditions and the rest of the nodes are eliminated
- Alpha
  - The maximiser value (-infinity)
- Beta
  - The minimizer value (+infinity)
- Condition
  - Alpha >=Beta

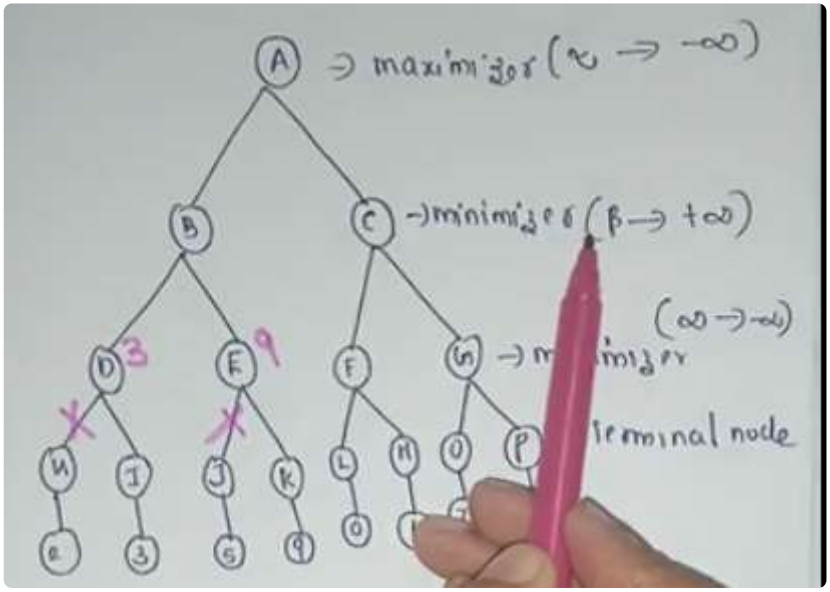
Example of Alpha-Beta Pruning



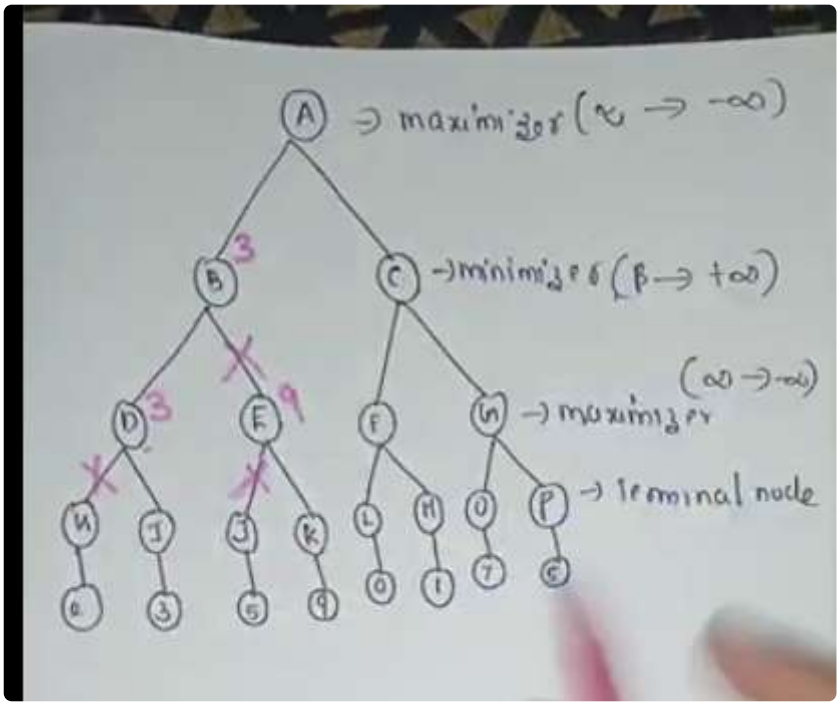
- Just like min-max algorithm, we need to assign the maximizer and minimizer, which are alpha and beta
- We will be moving from left to right



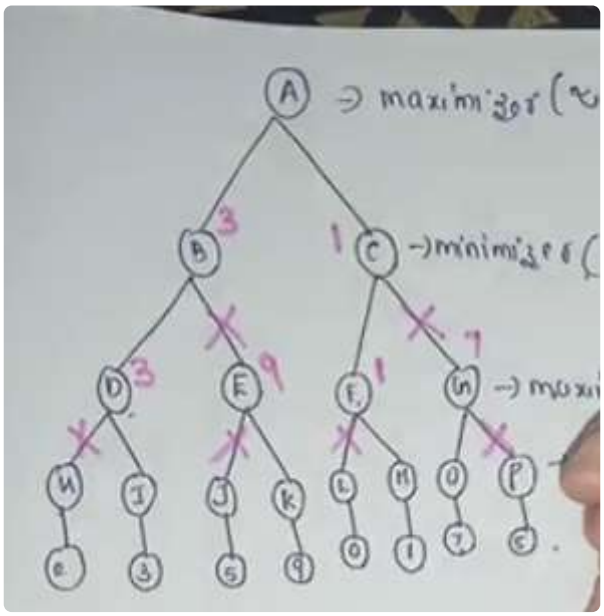
- Starting from the left, taking H and I
  - The values are 2 and 3, maximum is 3, taking that
  - Pruning H from the tree



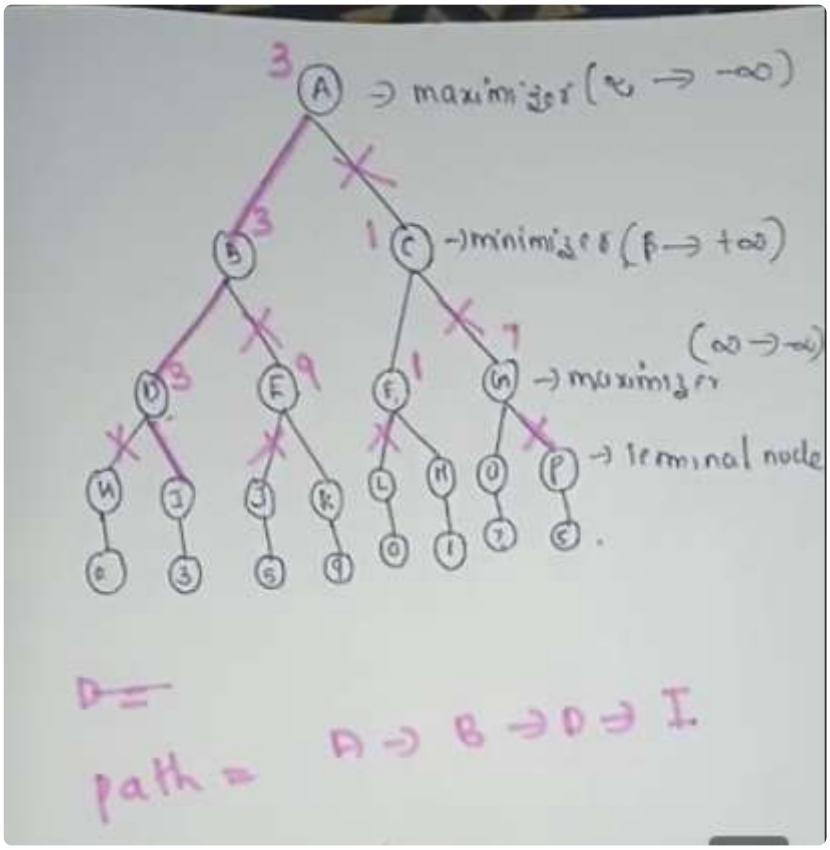
- Taking J and K, maximum value is 9
- Pruning J

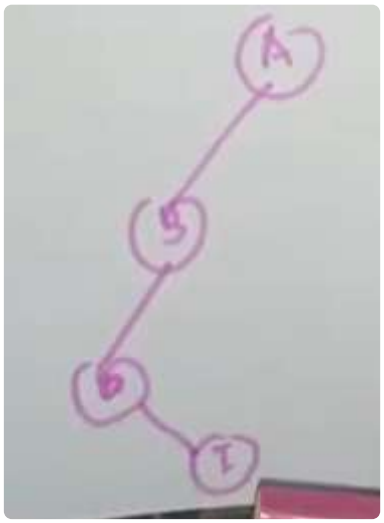


- Taking D and E
- Minimum value is 3
- Pruning E



- Similarly, pruning, we get the above values

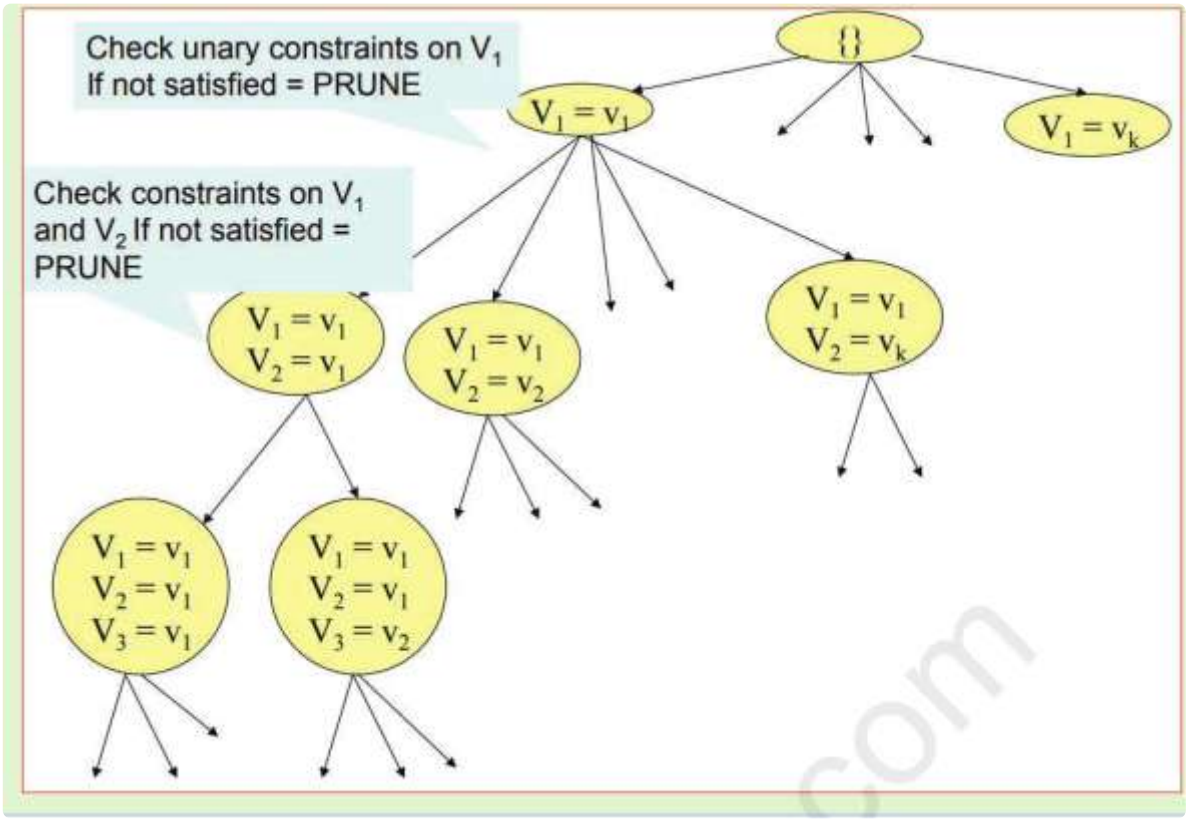


- Similarly, Comparing 3 and 1, max value is 3, assigning 3 to A
- When we go through the unpruned areas in the tree we get the following path
- Path = A -> B -> D -> I
- Giving us the following tree
  - 

8. Backtracking search in csp

Backtracking search is a basic method used to solve **constraint satisfaction problems (CSP)**, such as puzzles like Sudoku. It tries to assign values to variables step by step while respecting certain constraints, and if it runs into an issue, it goes back and tries a different path.

Here's how it works:

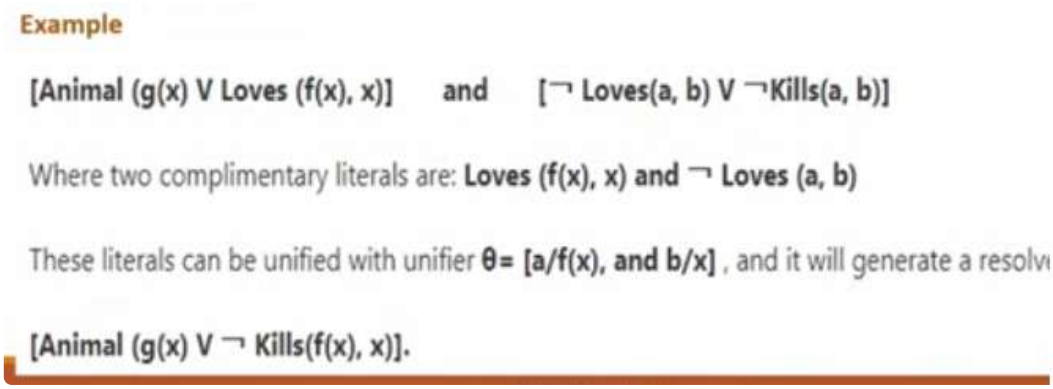
- **Partial Assignments**:
  - It doesn't try to solve the whole problem at once. Instead, it makes a guess for one variable at a time.
  - As it makes these guesses, it goes deeper into the decision tree, much like **depth-first search (DFS)**.
- **Branching Factor**
  - If there are **n** variables, each with **d** possible values, at the top level of the tree, the algorithm has **n × d** choices.
  - As the algorithm assigns values, the number of choices (branching factor) gets smaller, like having **(n-1) × d** choices at the next level, and so on.
  - Eventually, you end up with **n! × d^n** possible outcomes.
- **Commutative Property**:
  - In CSPs, the **order in which you assign values** doesn't change the result (as long as the choices satisfy the constraints).
- 

- Here we are checking for unary constraint (Where only one variable is involved)
- V1=v1 is unary, so we can proceed to larger ones
- Then we check constraints on
  - V1= v1
  - V2=v1
- And proceed like so, if any of them fails, then backtrack

9. Resolution problem

- Resolution is a method used in logic to prove that something is true by showing that assuming the opposite leads to a contradiction. This is called a "proof by contradiction."
- The resolution technique uses a single rule of reasoning that works best with a specific way of organizing logical statements called "conjunctive normal form" (CNF). In this form, statements are expressed as a series of clauses, which makes it easier to work with them and find contradictions.

Example



- By unification we are getting contradictions

  Loves (f(x), x) and ¬ Loves (a, b)

- We can eliminate these contradictions and give a result

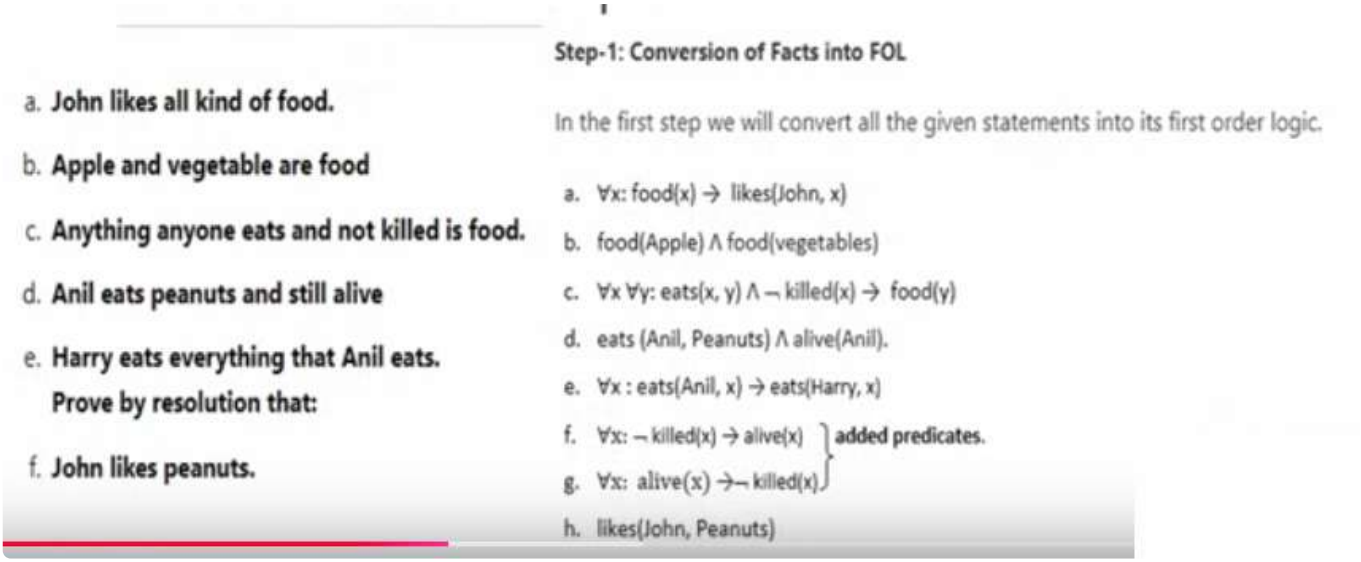
[Animal (g(x) V ¬ Kills(f(x), x)].

Steps for resolution

1. Conversion of facts into first order logic (FOL)
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph

Resolution-Example

Prove that John likes peanuts



- **Step 1**
  - We will first take all the statement into first order logic
  - 2 Extra predicates are added
- **Step 2**
  - We need to convert First order into Conjunctive normal form, because it will be easier for doing resolution

Eliminate all implication (→) and rewrite

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

f. ∀x¬ [¬ killed(x)] V alive(x)

g. ∀x ¬ alive(x) V ¬ killed(x)

h. likes(John, Peanuts).

- **Step 3**
  - Move Negation inwards

Move negation (¬)inwards and rewrite

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

f. ∀x ¬killed(x)] V alive(x)

g. ∀x ¬ alive(x) V ¬ killed(x)

h. likes(John, Peanuts).

- **Step 4**
  - Rename variables, make it unique

Rename variables or standardize variables

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀w¬ eats(Anil, w) V eats(Harry, w)

f. ∀g ¬killed(g)] V alive(g)

g. ∀k ¬ alive(k) V ¬ killed(k)

h. likes(John, Peanuts).

- **Step 5**
  - Drop universal quantifier
    - Things like (for all x)
      - ∀y ∀z -

a. ¬ food(x) V likes(John, x)

b. food(Apple)

c. food(vegetables)

d. ¬ eats(y, z) V killed(y) V food(z)

e. eats (Anil, Peanuts)

f. alive(Anil)

g. ¬ eats(Anil, w) V eats(Harry, w)

h. killed(g) V alive(g)

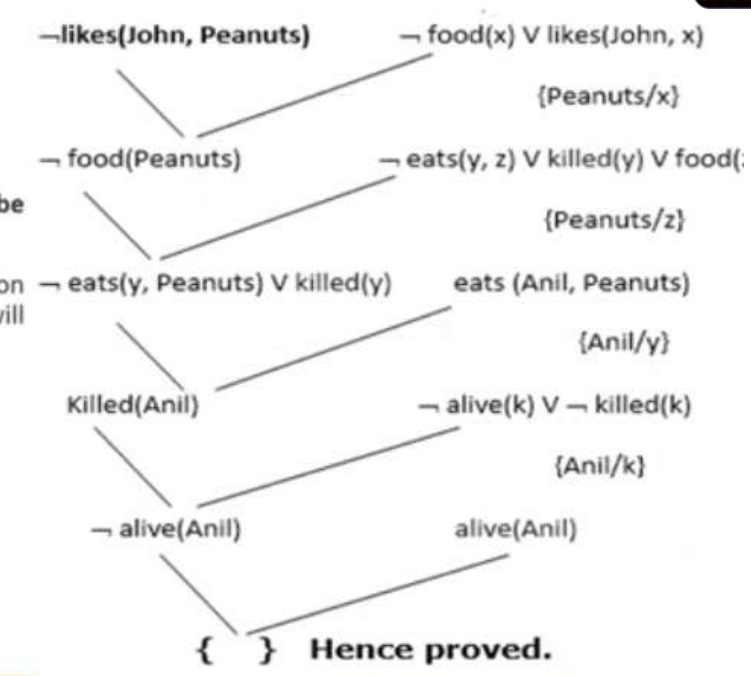
i. ¬ alive(k) V ¬ killed(k)

j. likes(John, Peanuts).

- **
- **Step 6**
  - Negate the statement to be proved
  - We will negate John likes peanuts statement

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)



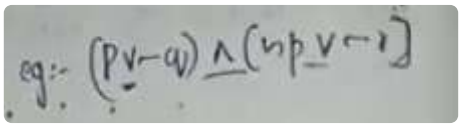
10. Convert to CNF

Conjunctive Normal Form (CNF) is a way of writing logical expressions where the formula is an **AND** of multiple **OR** statements.

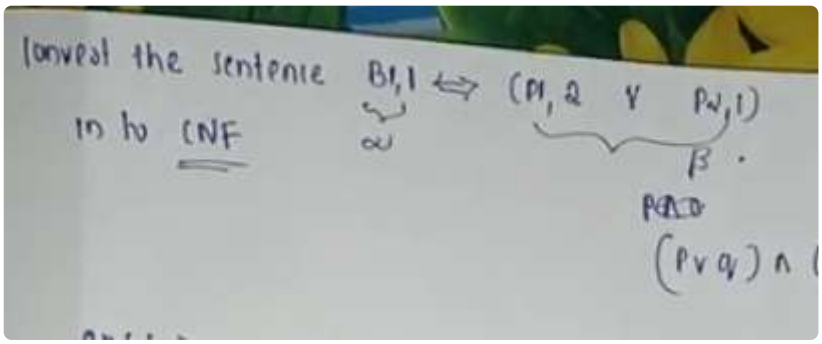
In CNF, a formula looks like this:

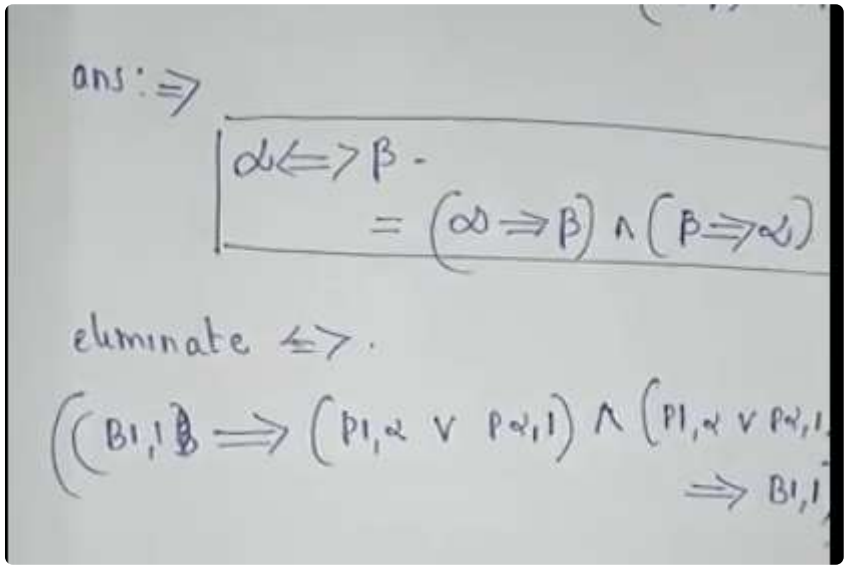
- Several **clauses** (groups of variables connected by ORs), combined using **AND**.

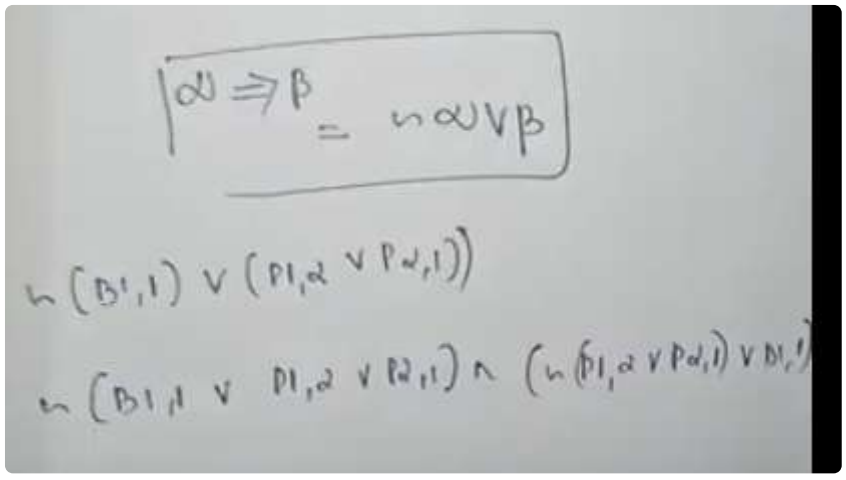
Examples of CNF:

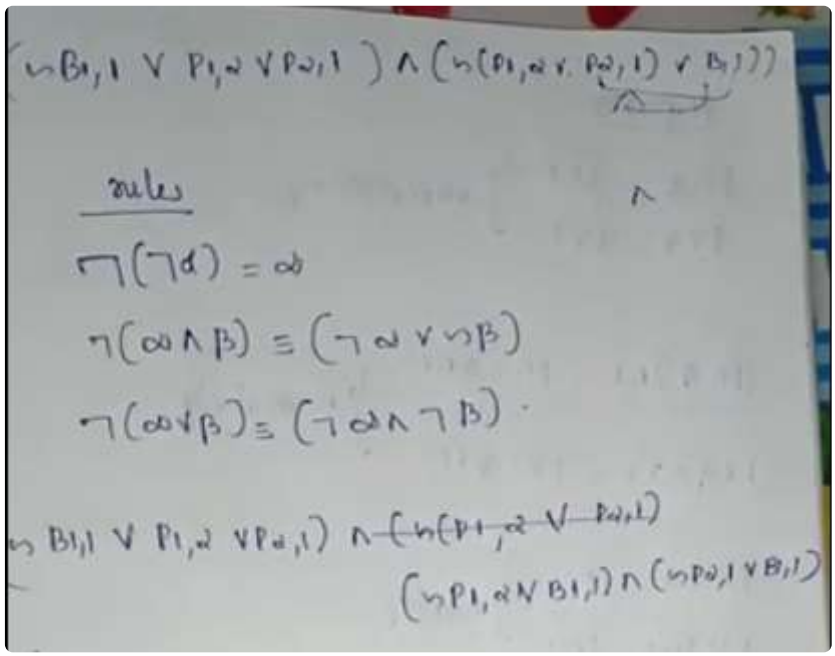
- (A OR B) AND (C OR ¬D)
- 

Example Problem



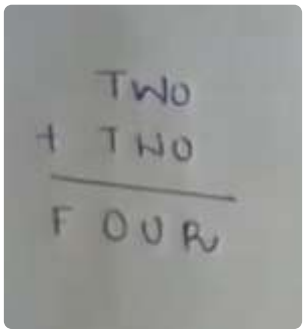
- Convert the sentence B1 1 <=> (P1,2 V p2 1) into CNF
- **Step 1**
  - Lets change the implies sign first
    - A <=>B can be written as (A=>B) ^ (B=>A)
    - Applying this
    - 

- **Step 2**
  - A => B can be written as ¬AVB
    - Applying this
    - 

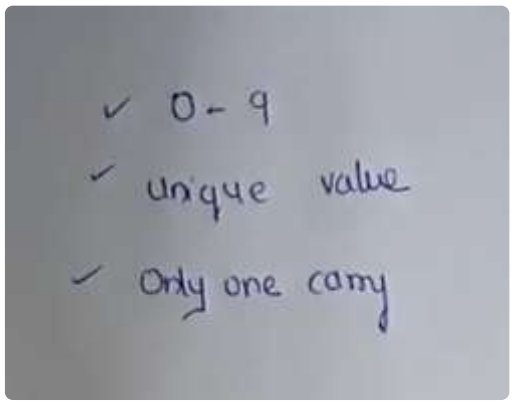
- **Step 3**
  - Applying distributive property
  - 

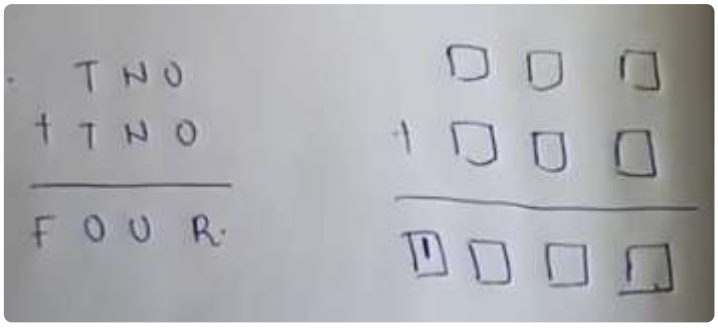
11. Cryptarithmetic problem

Example 1

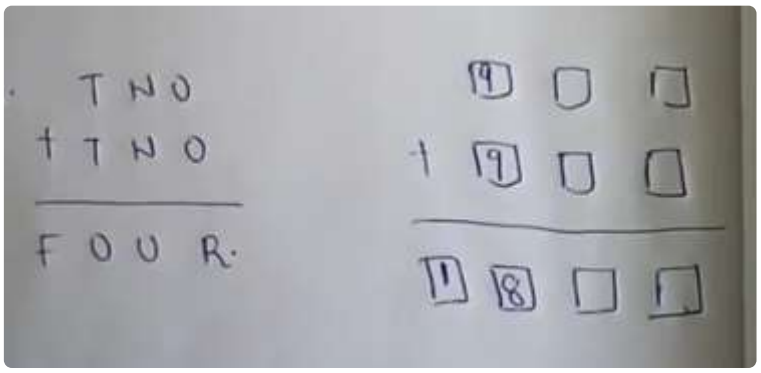
- Consider this example
  - 

  - We need to assign values to each of these letters
  - Value for T, Value for W, etc..
  - We are also given the following conditions for assigning values to these letters

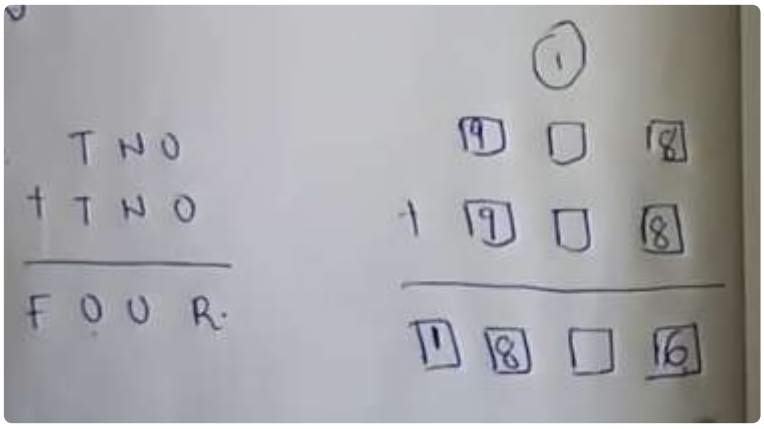
- 

- **Step 1**
  - 

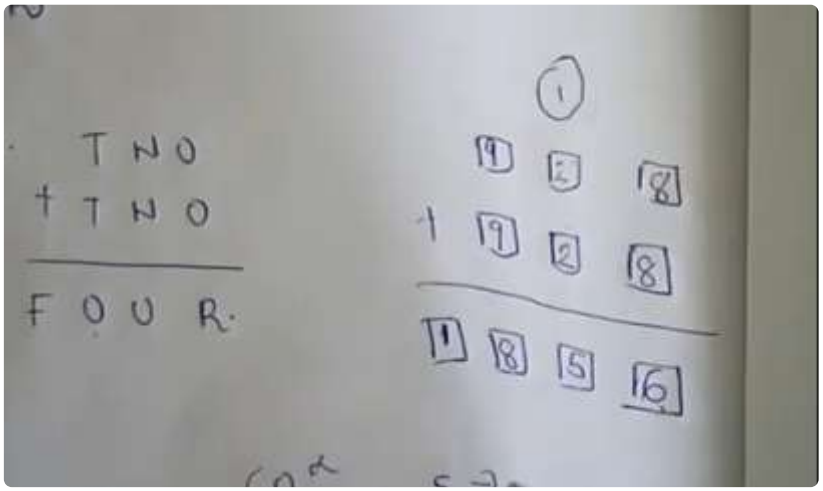
  - From above we can see that
    - O+O = R
    - W+W = U
    - T+T gives O with F as carry
  - From the conditions its said that there can be only one carry
  - Our carry here is F
  - The only value the carry can have is 1
    - Because, 9+9 = 18
    - 1+9 = 10
    - These two are the maximum and minimum values we will get those have carry, and the carry is only having 1
  - So we can assign 1 to F
- **Step 2**

- 

- Lets see T + T
- T+T will give O with carry F
- We know that F is 1
- Which number combinations will give 1 as carry?
  - 5+5 = 10
  - 6+6 = 12
  - 7+7 = 14
  - 8+8 = 16
  - 9+9 = 18
  - Numbers from 5 to 9 gives 1 as carry.
  - You can choose any one of these.
  - Im choosing 9
- So we got Values of W as 9
- Value of O as 8

- **Step 3**
  - 

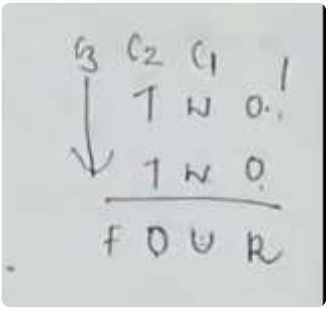
  - Since we got value of O
  - O+O = R
  - 8+8 = 16
  - We can assign R as 6 and we need to put the carry 1.

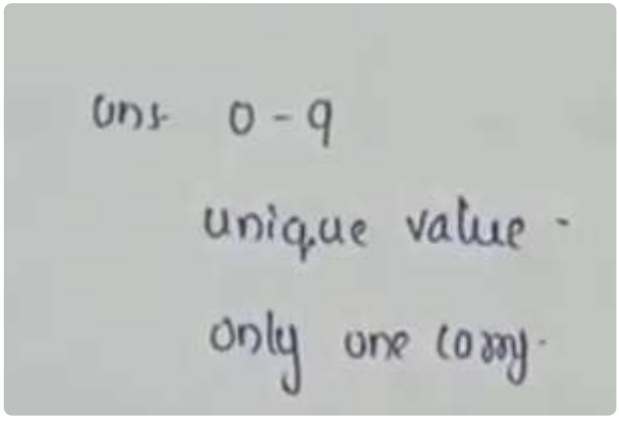
- **Step 4**
  - 

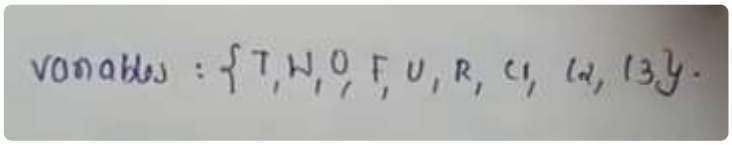
  - W+W = U
  - Since there is a carry, we need to include that as well.
  - W+W+1 = U
  - Our only condition here is that, W+W+1 should not have a carry
  - Possible combinations
    - 0 ❌
      - W = 0 (unique)
      - U = W + W + 1
      - U = 0+0+1 = 1, which is not unique
      - This number is not possible
    - 1 ❌
      - W = 1 (not unique)
      - Cant assign 1, its already given to F
    - 2 ✅
      - W = 2 (Unique)
      - U = 2+2+1 = 5 (unique)
      - 2 and 5 are unique values, so we can assign
- **Solution**
  - T = 9
  - O = 8
  - U = 5
  - W = 2
  - F = 1

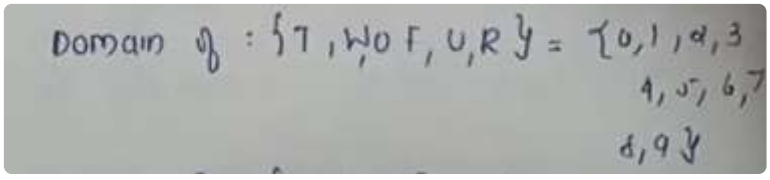
Example 2

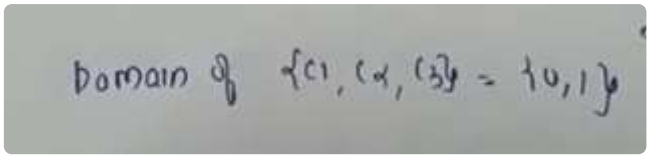
Solve the following crypt arithmetic problem by hand, using strategy of backtracking with **forward checking** and **MRV** and **least constraint value heuristic**



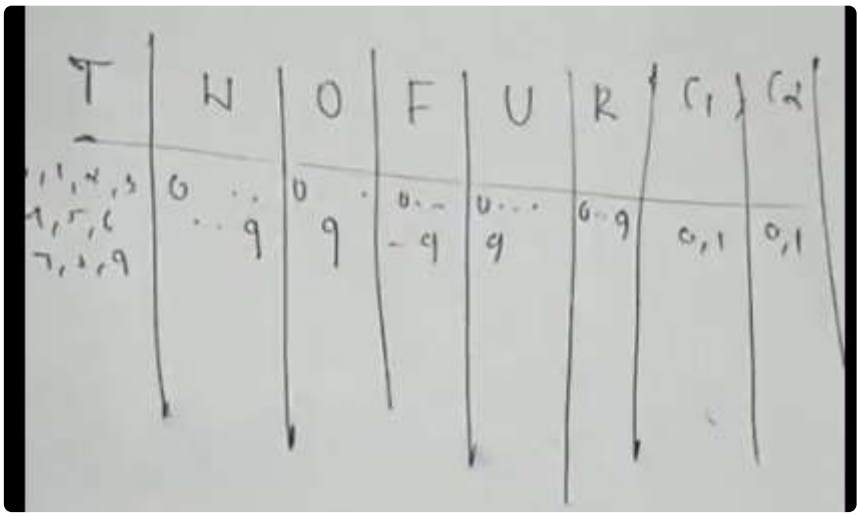
- Constraints
  - 

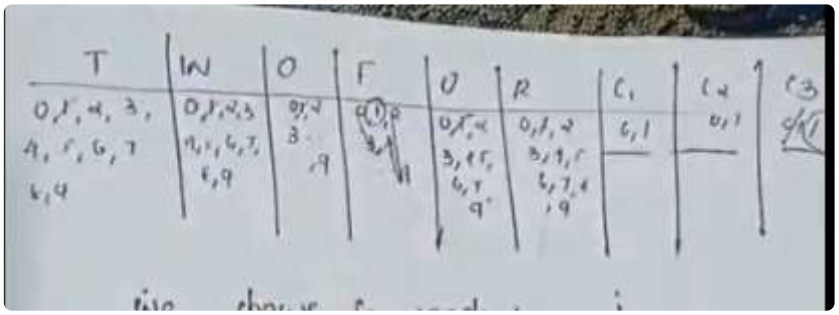
- Variables
  - 

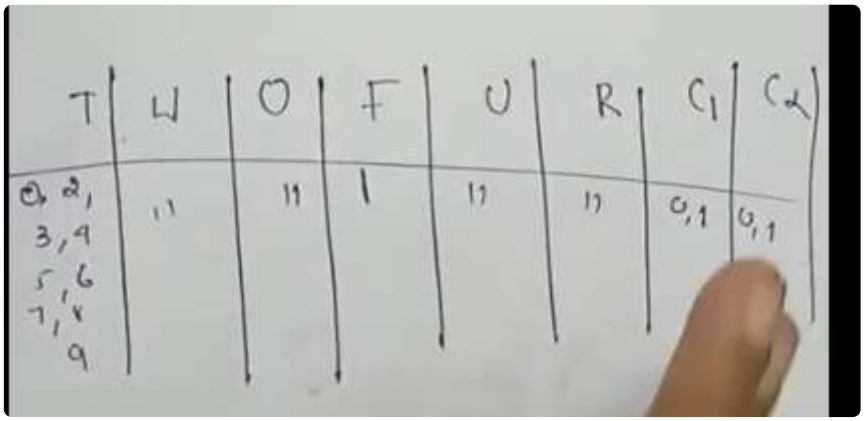
- Domain of
  - 

    - The variables can be from 0 to 9
  - 

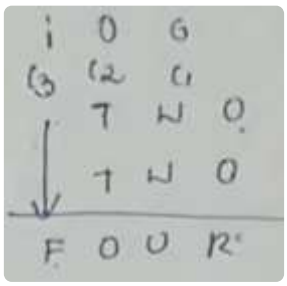
    - For carry its either 0 or 1
- Using MRV, we choose variable with **fewest legal value**
- Create a table with the domain

- 

  - C1,C2,C3 has the fewest legal values
  - Lets choose the last carry, C3
  - Carry can be 1
  - F = 1
- Removing 1, from others domains
  - 

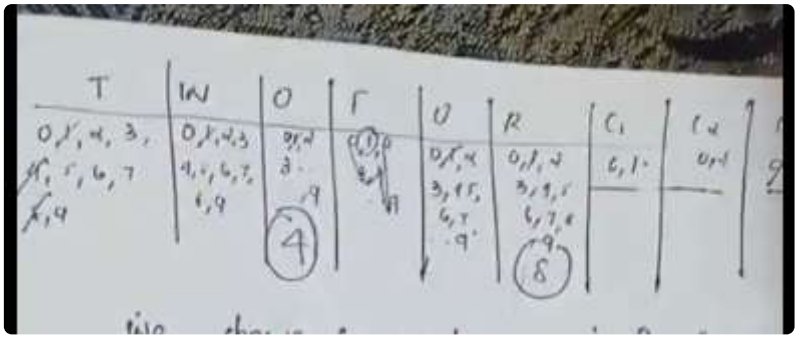
  - 

  - Next lets choose C2
  - Since C2 is not active carry, lets fix it to 0

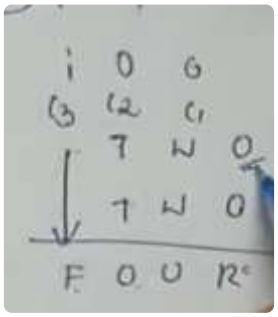


- Lets put O as 4
- R = O+O = 8

- Removing 4 and 8 from the domain

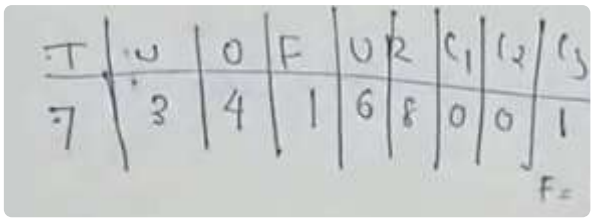


- Lets see W, lets set it to 3
- W = 3
- 3 + 3 = U = 6



  - We know that value of O = 4
  - T must be 7
  - 7+7 = 14

- Result



12. First order Logic

- It is a powerful language that develops information about the objects in more easy way and can also express the relationship between those objects
- It is the another way of knowledge representation in AI and it is an extension to propositional logic
- It is also known as predicate knowledge or first order predicate logic
- First order logic does not only assume that the world contain fact like propositional logic but also assume the following things in the world.
- Other than propositional, first order logic uses the following
    - Objects -> A,B,number, squares, theories, people etc
    - Relations -> it can be a unary relation such as red adjacent between object
    - Function -> Father of, best friend

Parts of first order logic

- As a natural language, First order logic also has 2 main parts
    - Syntax
    - Semantics

Syntax of FOL

- Syntax is the collection of symbols
- Basic Elements of FOL
- Following are the basic elements of FOL

-