AI-Module-4-Important-Topics

- AI-Module-4-Important-Topics
    - 1. Knowledge based agents
        - What Are Knowledge-Based Agents?
        - Key Components
        - What Can Knowledge-Based Agents Do?
        - Architecture of knowledge based agent
    - 2. Propositional Logic
        - Key Facts About Propositional Logic
        - Syntax of propositional logic
        - Example 1
        - Example 2
    - 3. First Order Logic
        - First order Logic
        - Parts of first order logic
        - Syntax of FOL
        - Examples of FOL
        - FOL Question
    - 4. Unification in FOL
        - Example of Unification in FOL
        - Conditions for Unification
        - Unification Algorithm
        - Unification Example
    - 5. Theorem Proving by Resolution in FOL
        - Example
        - Steps for resolution

    - Resolution-Example
- 6. CNF Conversion Problem
    - Properties
    - CNF (Conjunctive Normal Form)
    - Example
- 7. Forward Chaining and Backward Chaining
    - Inference Engine
    - Forward chaining
    - Backward chaining
    - Difference between Forward and Backward Chaining

1. Knowledge based agents

A **knowledge-based agent** is a type of intelligent system that uses knowledge about the world to make decisions and act effectively. Here's how they work and what they consist of:

What Are Knowledge-Based Agents?

- They store **knowledge** about the world and use it to reason and decide what to do.
- They update their knowledge when they observe something new.
- They can represent the world formally (e.g., using logic) and behave intelligently based on this knowledge.

Key Components

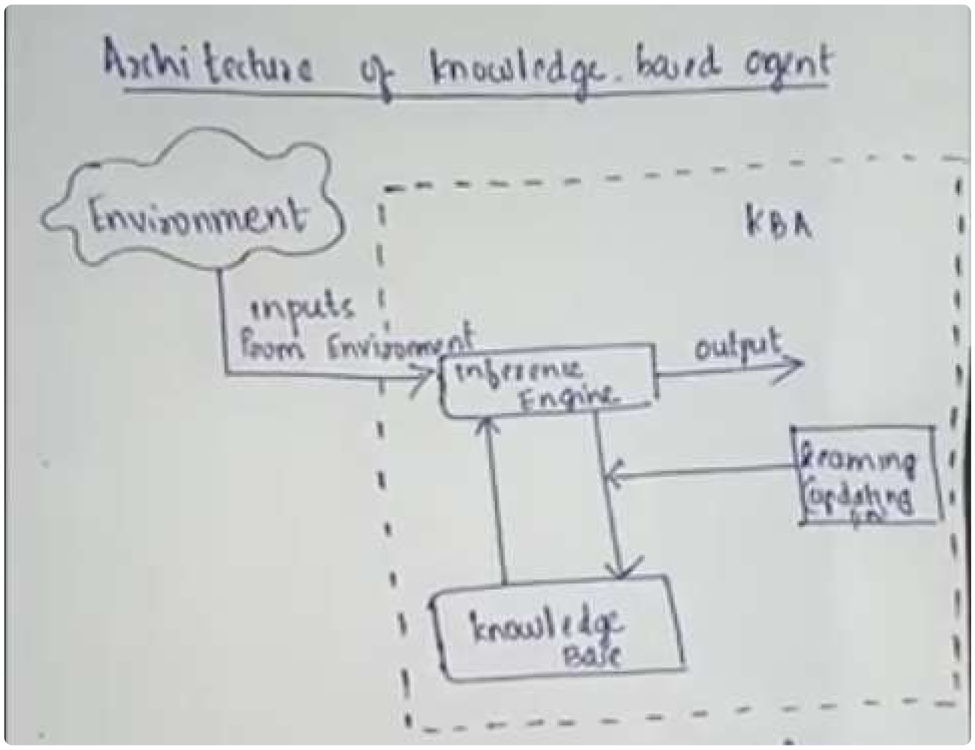
1. **Knowledge Base (KB):**
   A storage system where the agent keeps facts about the world.
2. **Inference System:**
   A reasoning mechanism that uses the knowledge base to deduce new facts or decide on actions.

What Can Knowledge-Based Agents Do?

1. **Represent Information:**
   They can store details about states, actions, and more.
2. **Respond to Observations:**
   They can interpret new inputs (percepts) and update their knowledge.

3. **Update Their Knowledge:**
   They refine their understanding of the world as they learn more.
4. **Reason About the World:**
   They can deduce new facts or conclusions based on existing knowledge.
5. **Choose Actions Intelligently:**
   They use their reasoning to decide the best action to take in a given situation.

Architecture of knowledge based agent



- **Input Handling:**
  - The agent receives **input (percepts)** from the environment.
  - This input is passed to the **inference engine** for processing.
- **Inference Engine:**
  - The inference engine reasons using the **knowledge base (KB)**.
  - It checks if the output (or desired action) already exists in the knowledge base.
- **Action Decision:**
  - If the required information is present in the KB, the agent uses it to decide on an action.
  - If not, the agent updates its KB with new knowledge and reasons further to determine an appropriate action.
- **Knowledge Base (KB):**

- Stores facts, rules, and previously deduced information.
- Continuously updated as the agent learns from its environment.

2. Propositional Logic

- Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions
- A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.
- Example
    - It is sunday
    - 3+3 = 7
    - 5 is a prime number

Key Facts About Propositional Logic

1. **Also Known as Boolean Logic:**
    - Works with values **0 (false)** and **1 (true)**.
2. **Symbols for Propositions:**
    - Propositions are represented with symbols like **A, B, C, D**.
3. **True or False Only:**
    - Propositions can **only** be true or false, not both.
4. **Logical Connections:**
    - Includes **objects, relationships, and logical operators** (like AND, OR, NOT).
5. **Connectives:**
    - Used to combine sentences (e.g., "AND," "OR").
6. **Special Terms:**
    - A statement that is **always true** is called a **tautology**.
    - A statement that is **always false** is called a **contradiction**.
    - **Commands or questions** are **not propositions** because they can’t be true or false.

Syntax of propositional logic

- The syntax of propositional logic defines the allowable sentences for knowledge representation
  - There are two types of proposition
  - Atomic proposition and compound proposition
- **Atomic propositions**
  - Atomic proposition are the simplest propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.
  - 2+2 is an atomic proposition and true fact
- **Compound proposition**
  - It is constructed by combining simpler or atomic proposition using parenthesis and logical connectives
  - Example
    - It is raining today, and street is wet

Example 1

- Tom is Hardworking
  - Here the variable is Tom
  - Hardworking(Tom)
- Tom is an intelligent student
  - Intelligent(Tom)
- If Tom is hardworking and intelligent then tom scores high marks
  - Hardworking(Tom) ^ Intelligent(Tom) -> ScoresHighMarks(Tom)

Example 2

Represent the following assertion in propositional Logic:
"A person who is radical(R) is electable(E) if he/she is conservative(C), but otherwise is not electable"

Answer
E↔C: Electability (E) depends on conservativeness (C). This bi-conditional means:

- If a person is conservative (C), then they are electable (E).
- If a person is not conservative , then they are not electable

- If a person is radical (R), then the above relationship between electability and conservativeness applies.
- R -> E<->C

3. *First Order Logic*

First order Logic

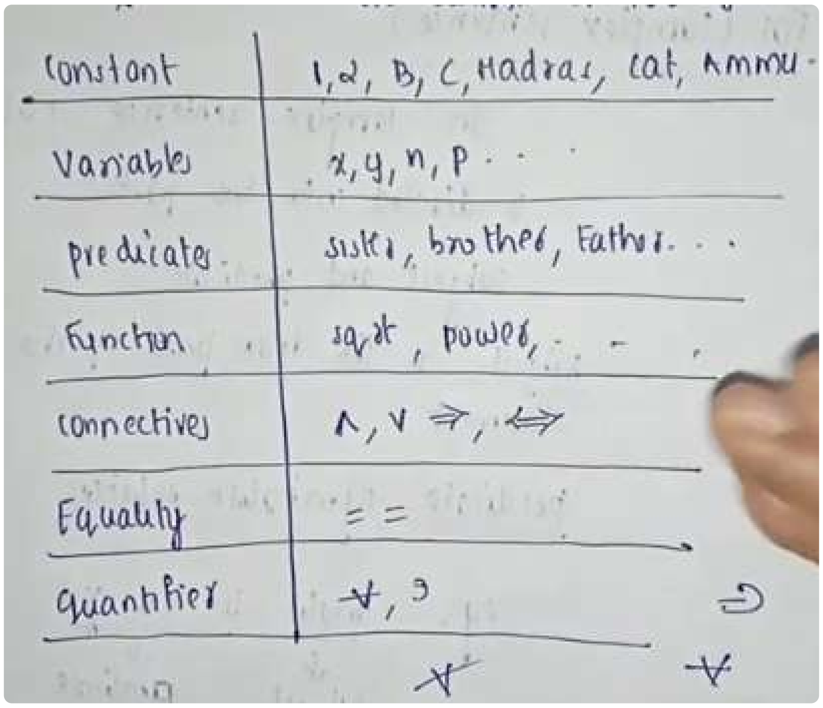
- It is a powerful language that develops information about the objects in more easy way and can also express the relationship between those objects
- It is the another way of knowledge representation in AI and it is an extension to propositional logic
- It is also known as predicate knowledge or first 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 propositionals, 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

- 

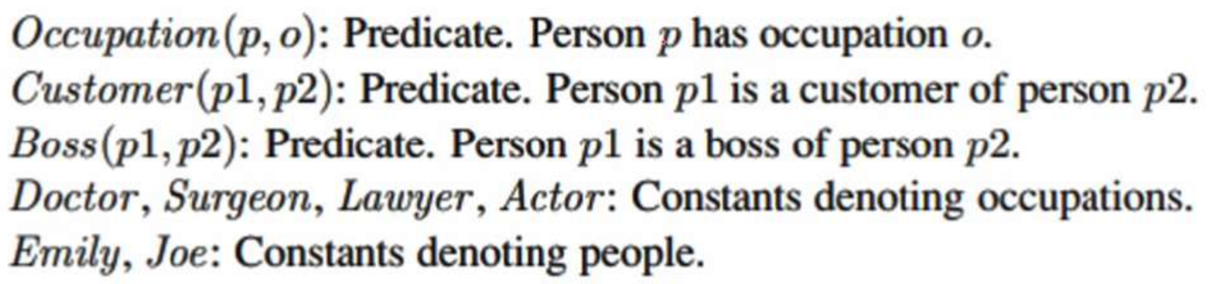
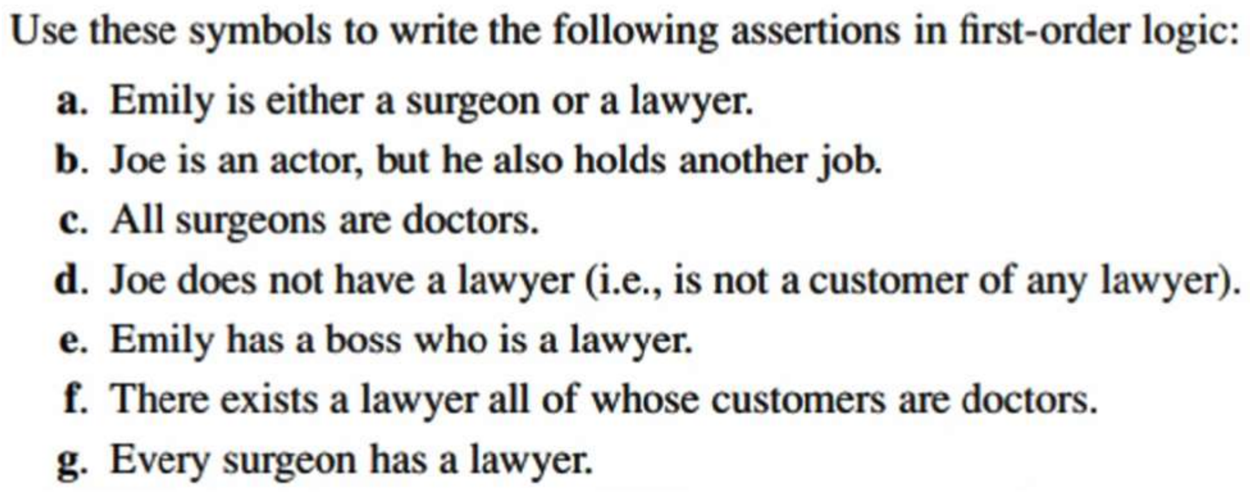
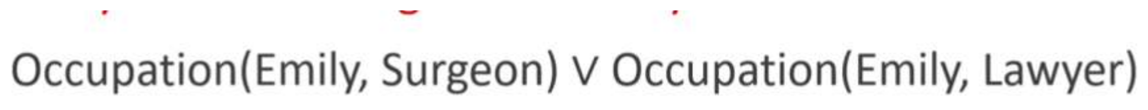
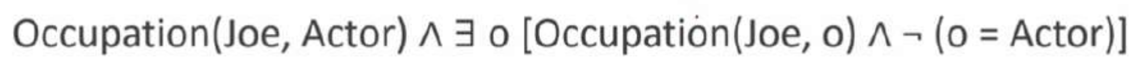
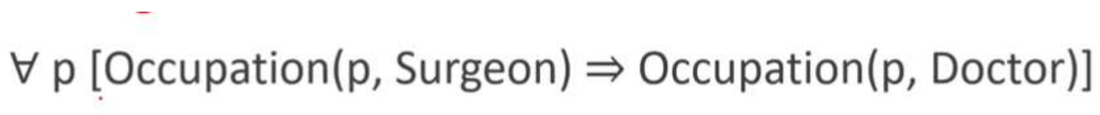
Constant	1, 2, B, C, Madras, cat, Ammu.
Variable	x, y, n, p . . .
predicate	sister, brother, Father . . .
Function	sqrt, power, . . .
connectives	∧, ∨ ⇒, ⇔
Equality	==
Quantifier	∀, ∃

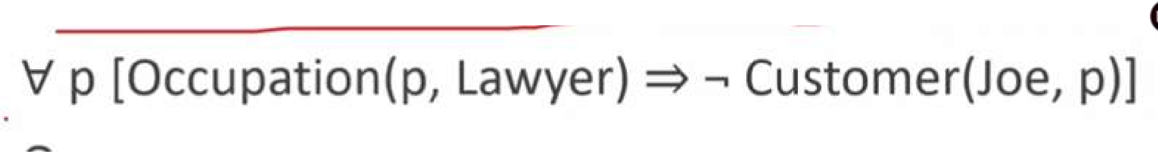
Examples of FOL

1. **All birds fly**
    1. In this question, the predicate is fly(bird)
    2. Since there are all birds who fly, it will be represented as follows
        1. ∀x bird(x) -> fly(x)
2. **Every man respects his parent**
    1. In this question, the predicate is respect(x,y) where x = man, y=parent
    2. Since there is every man so will use **∀** and will be represented as follows
        1. ∀x man(x) -> respects(x,parent)
3. **Some boys play cricket**
    1. In this question, the predicate is play(x,y) where x=boys, and y=game.
    2. Since there are **some boys** so we will use ∃x boys(x) -> play(x,cricket)
4. Not all students like both Mathematics and Science
    1. In this question the predicate is like(x,y) where x=student and y=subject
    2. Since there are not all students, we will use ∀ with negation
    3. ¬∀(x) [student(x) -> like(x,Mathematics) ^ like(x,Science)]
5. Only one student failed in mathematics
    1. In this question, the predicate is failed(x,y) where x=student, and y=subject.

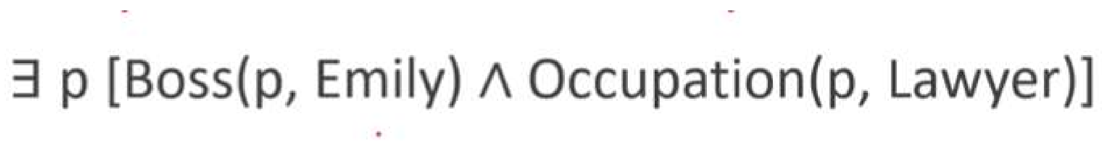
2. Since there is only one student who failed in mathematics, so we will use the following representation for this
3. ∃x [student(x)-> failed(x,Mathematics) ^ ∀(y)[¬(x==y) ^ student(y) -> ¬failed(x,Mathematics)]]

FOL Question

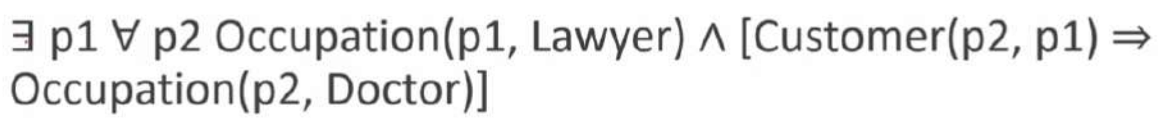
- 
  Occupation(p, o): Predicate. Person p has occupation o.
  Customer$(p1, p2)$: Predicate. Person $p1$ is a customer of person $p2$.
  Boss$(p1, p2)$: Predicate. Person $p1$ is a boss of person $p2$.
  Doctor, *Surgeon*, *Lawyer*, *Actor*: Constants denoting occupations.
  Emily, *Joe*: Constants denoting people.
- 
  Use these symbols to write the following assertions in first-order logic:
  a. Emily is either a surgeon or a lawyer.
  b. Joe is an actor, but he also holds another job.
  c. All surgeons are doctors.
  d. Joe does not have a lawyer (i.e., is not a customer of any lawyer).
  e. Emily has a boss who is a lawyer.
  f. There exists a lawyer all of whose customers are doctors.
  g. Every surgeon has a lawyer.
- Emily is either a surgeon or a lawyer
  - 
    Occupation(Emily, Surgeon) ∨ Occupation(Emily, Lawyer)
- Joe is an actor but he also holds another job
  - 
    Occupation(Joe, Actor) ∧ ∃ o [Occupation(Joe, o) ∧ ¬ (o = Actor)]
- All surgeons are doctors
  - 
    ∀ p [Occupation(p, Surgeon) ⇒ Occupation(p, Doctor)]
- Joe does not have a lawyer (Joe is not a customer of any lawyer)


∀ p [Occupation(p, Lawyer) ⇒ ¬ Customer(Joe, p)]

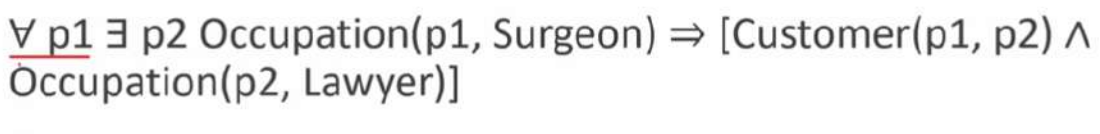
- Emily has a boss who is a lawyer


∃ p [Boss(p, Emily) ∧ Occupation(p, Lawyer)]

- There exists a lawyer all of whose clients are doctors (All of whose customers are doctors)


∃ p1 ∀ p2 Occupation(p1, Lawyer) ∧ [Customer(p2, p1) ⇒ Occupation(p2, Doctor)]

- Every surgeon has a lawyer


∀ p1 ∃ p2 Occupation(p1, Surgeon) ⇒ [Customer(p1, p2) ∧ Occupation(p2, Lawyer)]

4. Unification in FOL

Unification is a method used to make two logical expressions (literals) look exactly the same by finding a suitable replacement (substitution) for their variables.

1. **What is unification?**
   - It's a way to replace variables in logical expressions to make two different expressions identical.
2. **How does it work?**
   - Unification takes two literals as input.
   - It finds and applies substitutions to make them match.
3. **What do we need to check?**
   - Can the two literals be matched?
   - What substitutions will make them identical?
4. **The Unification Algorithm**

- A step-by-step process exists to figure this out, called the *unification algorithm*. It's a simple and systematic way to perform unification.

Example of Unification in FOL

Let’s work through an example to make unification clear:

1. **The Expressions**
   We have two different logical expressions:
   - $P(x, y)$ (Expression 1)
   - $P(a, f(z))$ (Expression 2)
     Our goal is to make these two expressions identical.
2. **How to Unify Them?**
   To make the two expressions identical, we find substitutions for the variables in Expression 1:
   - Replace x with a
   - Replace y with $f(z)$
3. **The Substitution Process**
   After substitutions, Expression 1 becomes:
   - $P(a, f(z))$
     Now, both expressions are identical.
4. **The Substitution Set**
   The set of substitutions we used is:
   - $[a/x, f(z)/y]$
5. **Most General Unifier (MGU)**
   - The substitutions a/x and $f(z)/y$ are called the **Most General Unifier (MGU)** because they are the simplest way to unify the two expressions.

Conditions for Unification

For two logical expressions to be unified, the following conditions must be met:

1. **Same Predicate Symbol**
   - The predicate symbol in both expressions must be the same.
     Example: $P(x)$ and $Q(x)$ cannot be unified because $P \neq Q$.
2. **Same Number of Arguments**

- Both expressions must have the same number of arguments.
  Example: $P(x, y)$ and $P(a)$ cannot be unified because one has two arguments while the other has one.

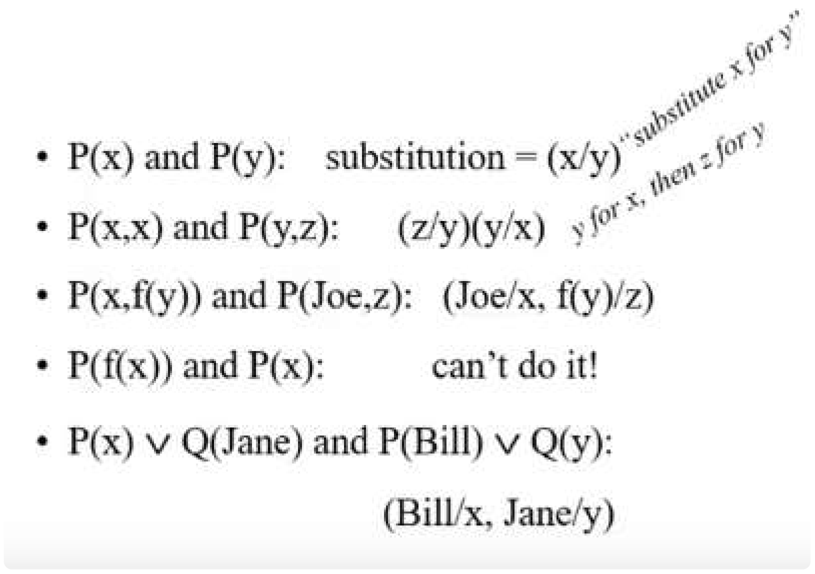
3. **No Repeated Variables in the Same Expression**
   - Unification will fail if two identical variables appear in the same expression during substitution.
     Example: $P(x, x)$ and $P(a, b)$ cannot be unified because x would need to be replaced by both a and b, which is contradictory.

Unification Algorithm

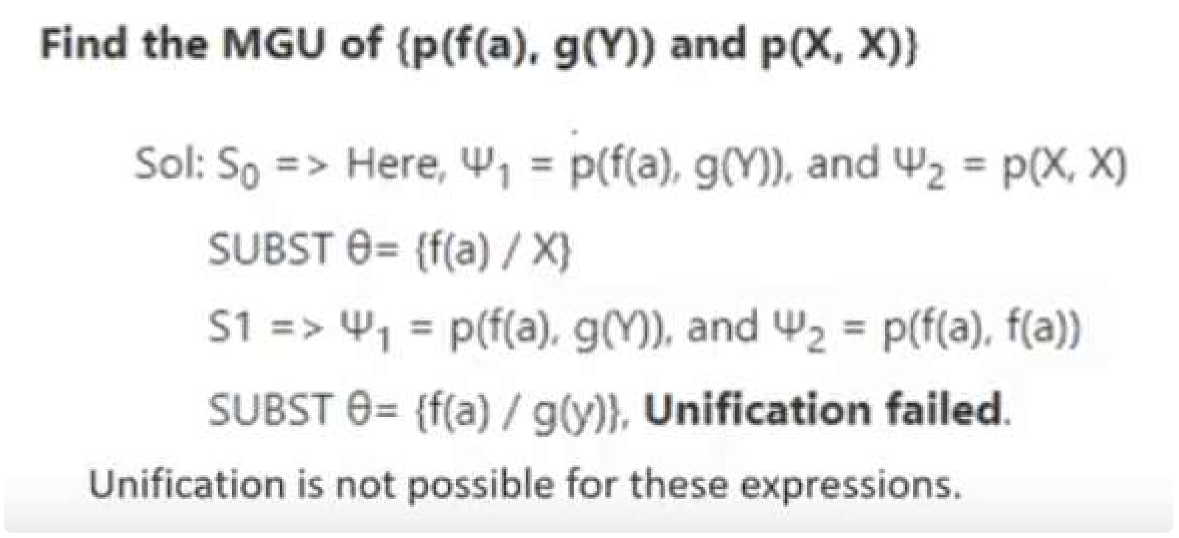
1. Initialize the substitution set to be empty.
2. **Recursive Unification of Atomic Sentences**
   - Check if both expressions are already identical.
   - If one expression is a variable v_i and the other is a term t_i, and t_i does not contain v_i, then:
     - Substitute t_i/v_i in the existing substitutions.
     - Add t_i/v_i to the substitution set list.
   - If both expressions are functions:
     - The function names must be identical.
     - The number of arguments must be the same in both expressions.

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
    - 

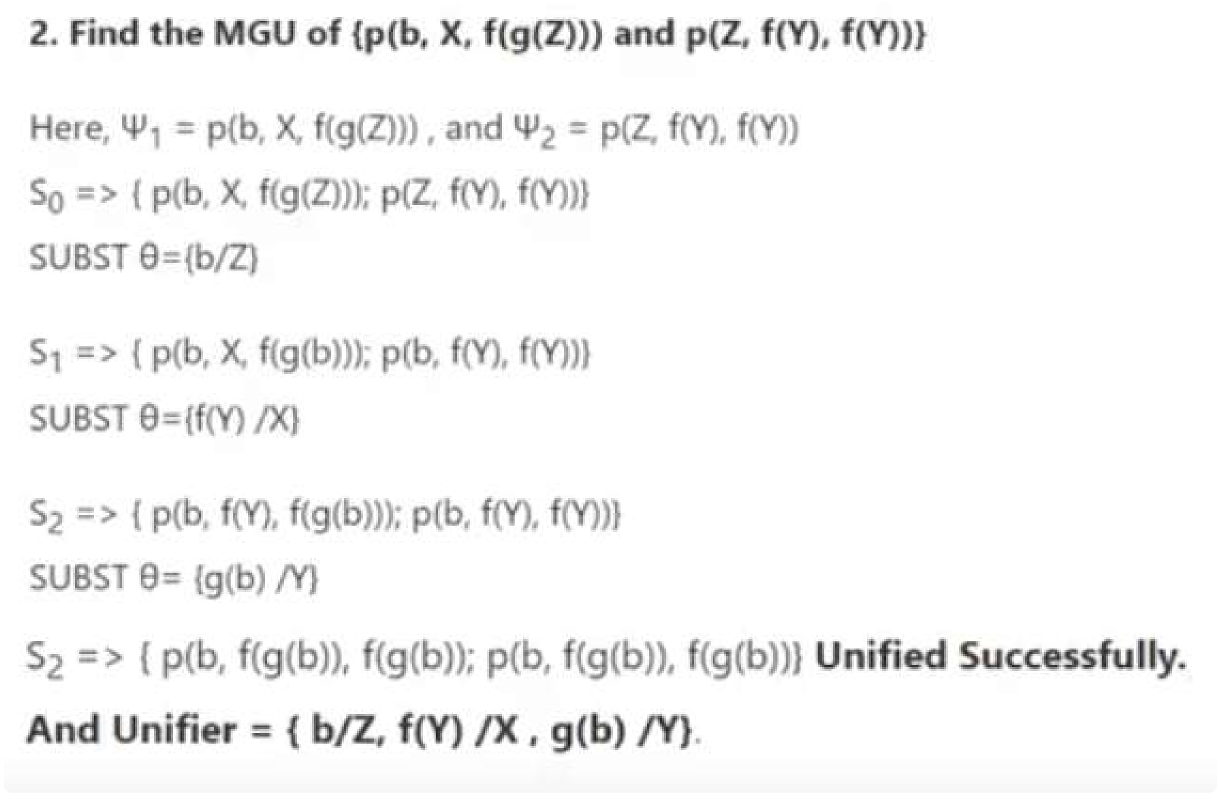
      Find the MGU of {p(f(a), g(Y)) and p(X, X)}

      Sol: S_0 => Here, Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(X, X)

      SUBST θ= {f(a) / X}

      S1 => Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(f(a), f(a))

      SUBST θ= {f(a) / g(y)}, **Unification failed**.

      Unification is not possible for these expressions.
    - 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))}

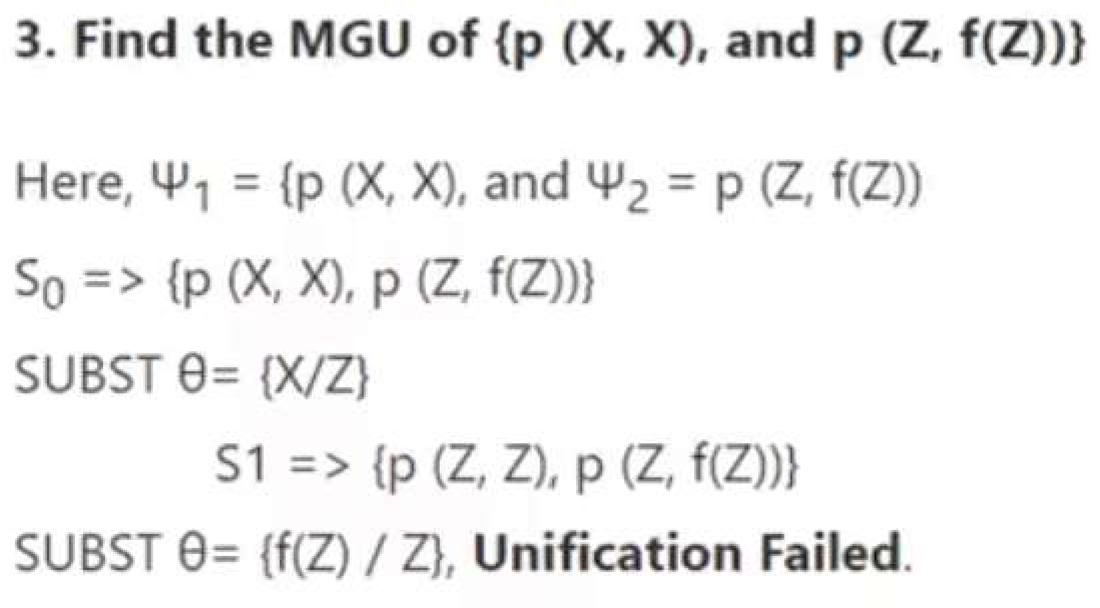
      SUBST θ={f(Y) /X}

      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))

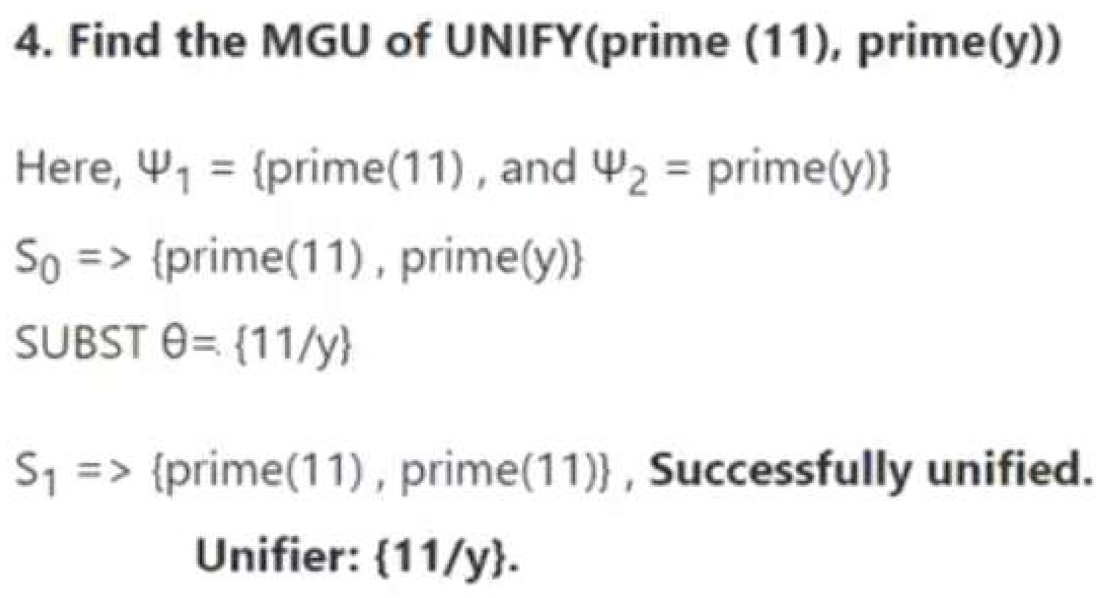
S_0 => {p (X, X), p (Z, f(Z))}

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)}

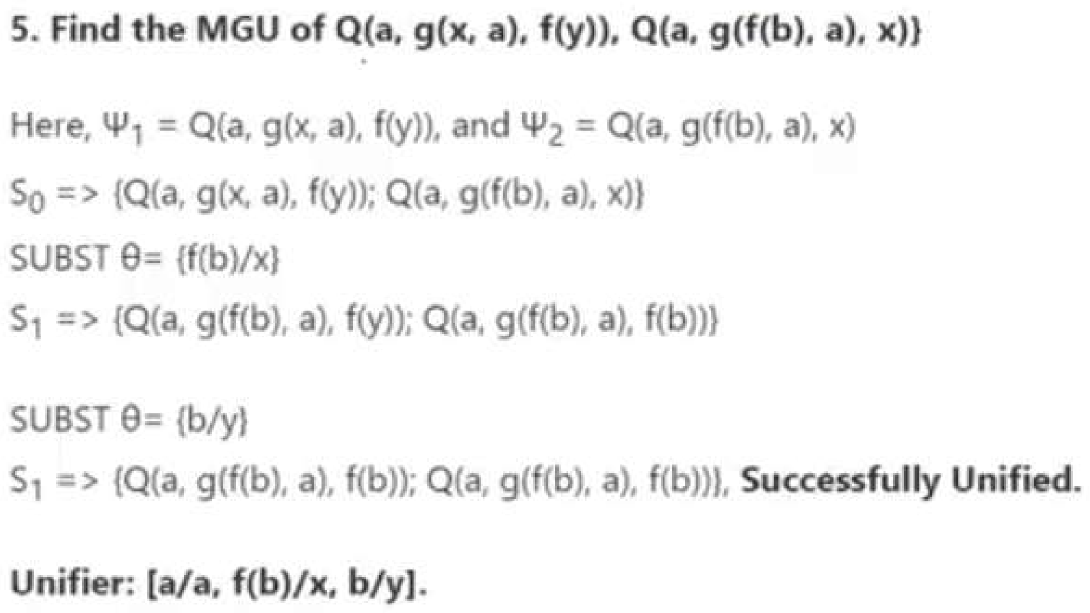
S_0 => {prime(11) , prime(y)}

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))}

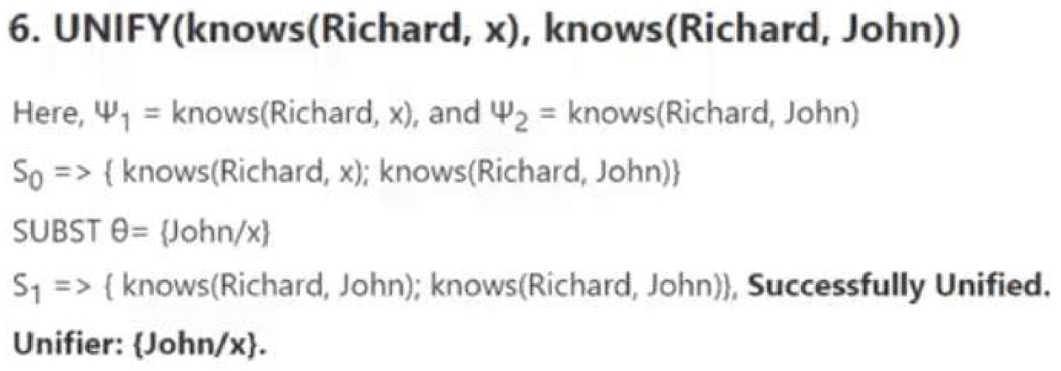
SUBST θ= {b/y}

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.**

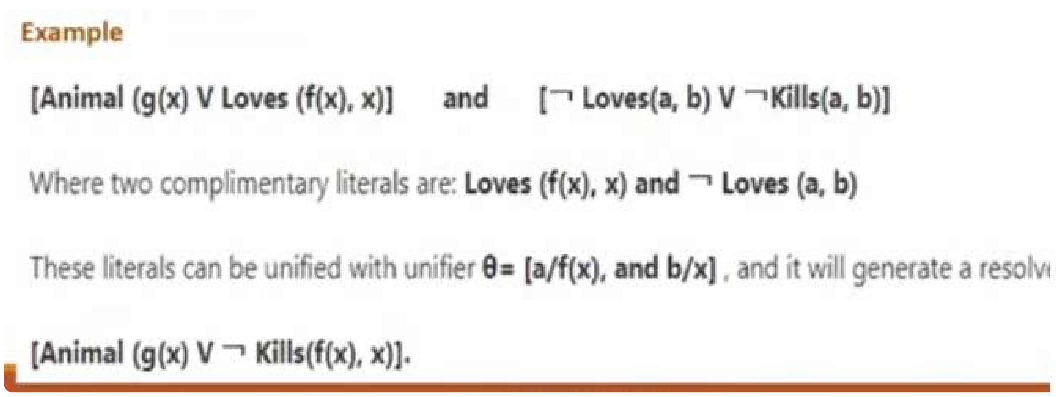
Unifier: {John/x}.

-

5. Theorem Proving by Resolution in FOL

- **Resolution** is a method used to prove theorems by showing that something is false, which leads to a contradiction.
- It works by applying one simple rule of inference.
- This method is especially useful when the logical expressions are in **conjunctive normal form (CNF)** or **clausal form**, which are special ways of writing logical statements.

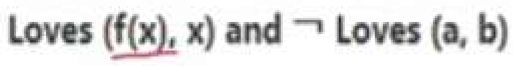
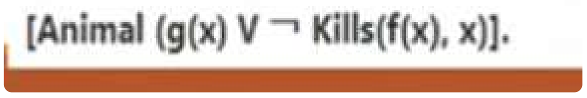
Example

- 
  Example

  [Animal (g(x) V Loves (f(x), x)] and **[¬ Loves(a, b) V ¬Kills(a, b)]**

  Where two complimentary literals are: **Loves (f(x), x) and ¬ Loves (a, b)**

  These literals can be unified with unifier **θ= [a/f(x), and b/x]** , and it will generate a resolv

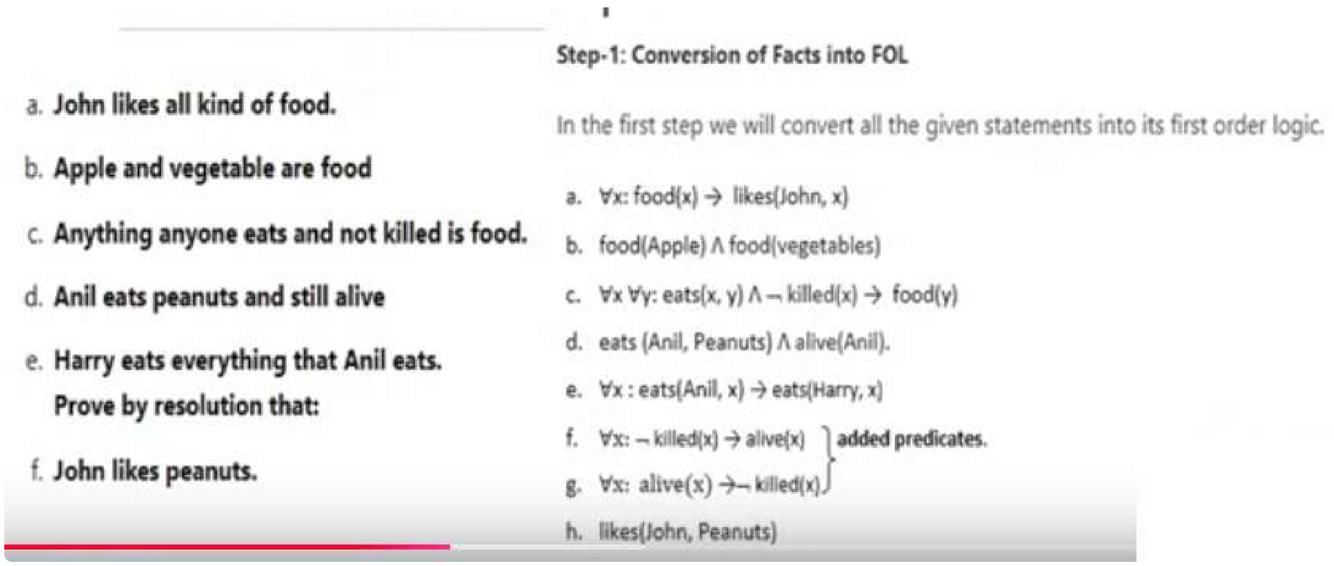
  [Animal (g(x) V ¬ Kills(f(x), x)].
- By unification we are getting contradictions
  - 
    Loves (f(x), x) and ¬ Loves (a, b)
- We can eliminate these contradictions and give the below result
  - 
    [Animal (g(x) V ¬ Kills(f(x), x)].

Steps for resolution

1. Conversion of facts into first order logic
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



a. **John likes all kind of food.**

b. **Apple and vegetable are food**

c. **Anything anyone eats and not killed is food.**

d. **Anil eats peanuts and still alive**

e. **Harry eats everything that Anil eats.**
Prove by resolution that:

f. **John likes peanuts.**

Step-1: Conversion of Facts into FOL

In the first step we will convert all the given statements into its first order logic.

a. ∀x: food(x) → likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y: eats(x, y) Λ ¬ killed(x) → food(y)

d. eats (Anil, Peanuts) Λ alive(Anil).

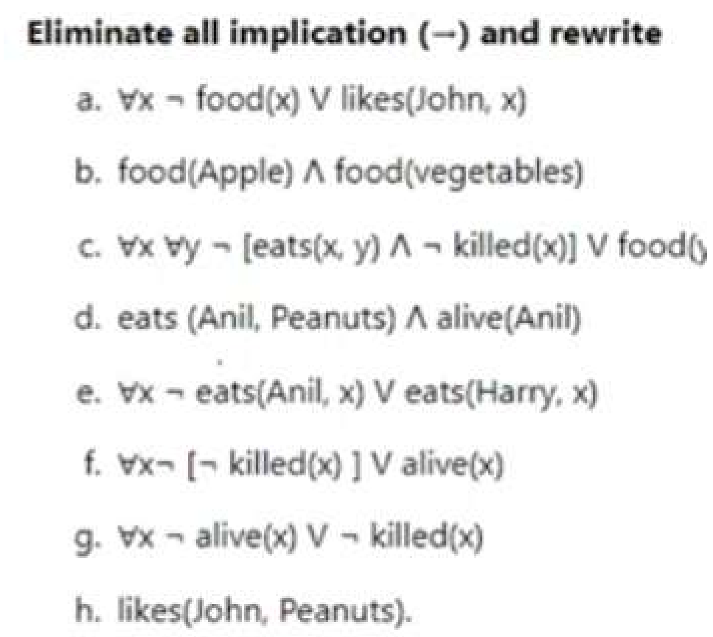
e. ∀x : eats(Anil, x) → eats(Harry, x)

f. ∀x: ¬ killed(x) → alive(x) } **added predicates.**

g. ∀x: alive(x) → ¬ killed(x)

h. likes(John, 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)

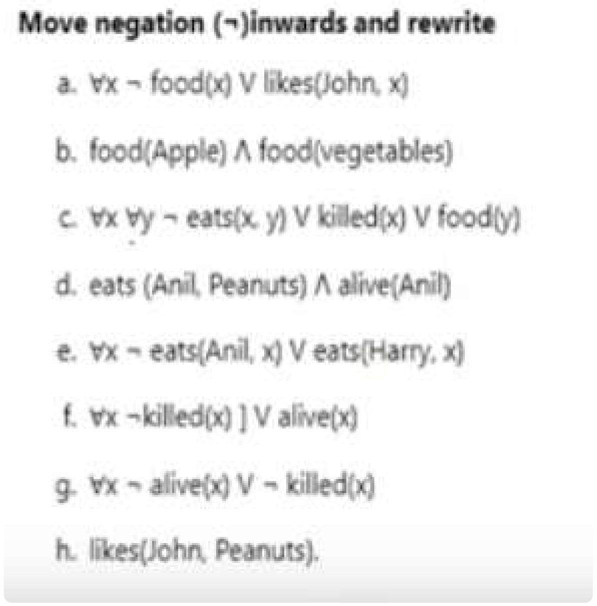
e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

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)

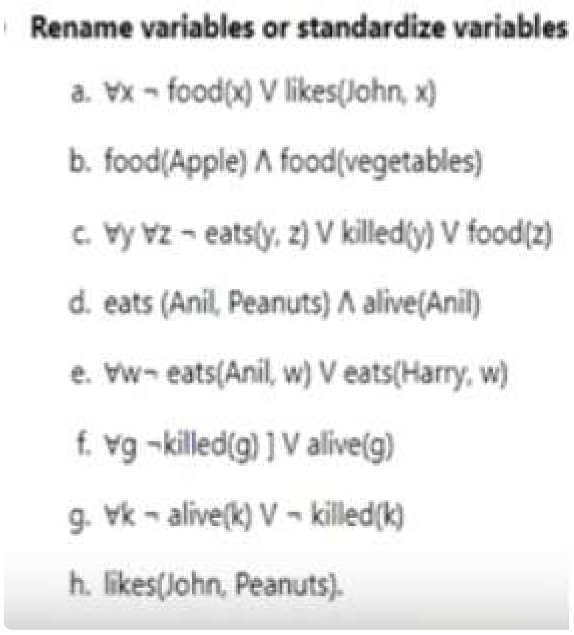
e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

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)

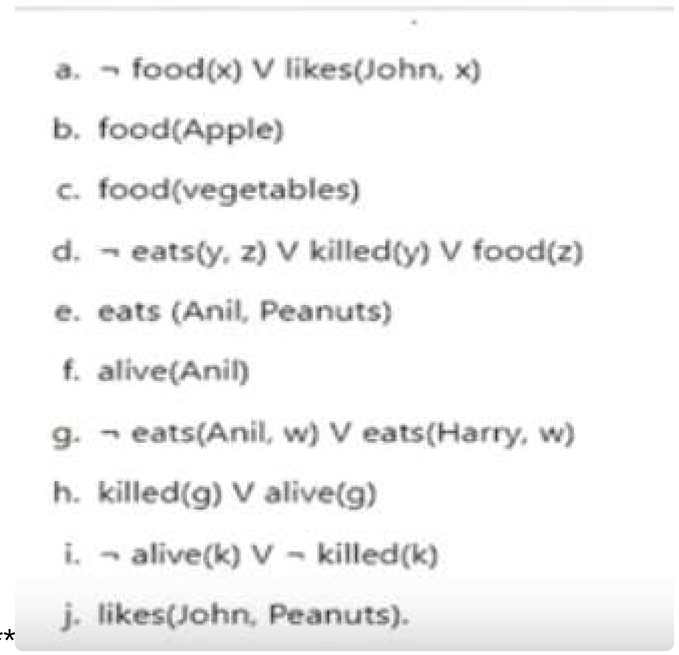
  e. ∀w¬ eats(Anil, w) V eats(Harry, w)

  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)

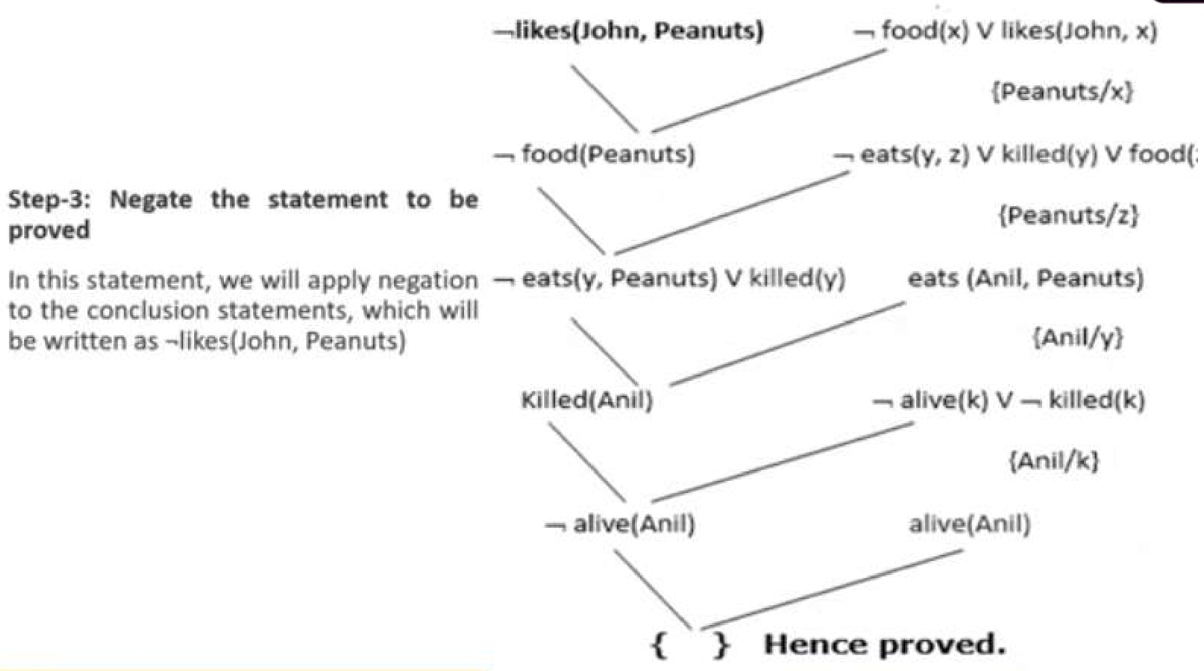
    f. alive(Anil)

    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

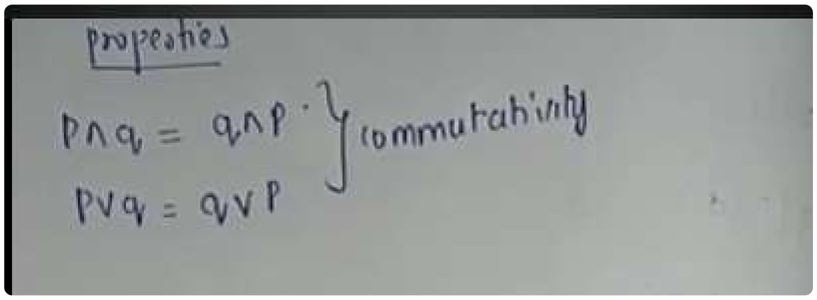


Step-3: Negate the statement to be proved

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

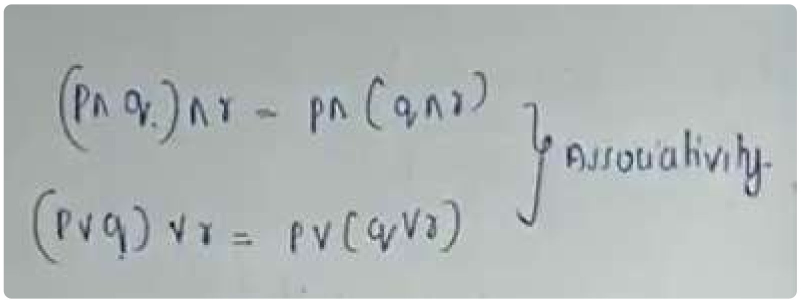
6. CNF Conversion Problem

Properties

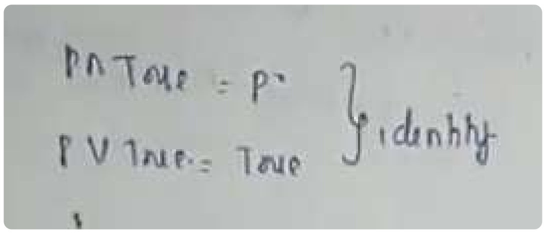


properties

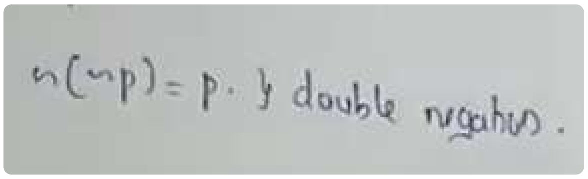
$p \wedge q = q \wedge p$
$p \vee q = q \vee p$ } commutativity



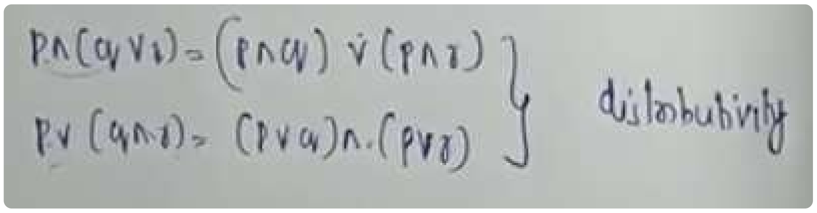
$(p \wedge q) \wedge r = p \wedge (q \wedge r)$
$(p \vee q) \vee r = p \vee (q \vee r)$ } Associativity



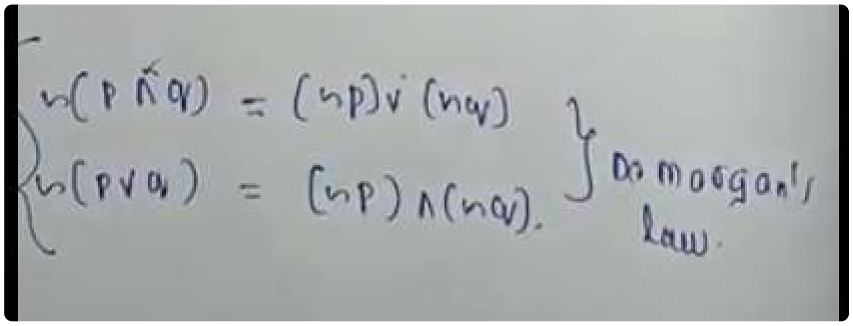
$p \wedge \text{True} = p$
$p \vee \text{True} = \text{True}$ } identity



$\neg(\neg p) = p.$ } double negation.

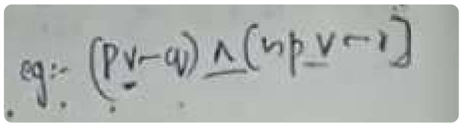


$p \land (q \lor r) = (p \land q) \lor (p \land r)$ }
$p \lor (q \land r) = (p \lor q) \land (p \lor r)$ } distributivity



$\neg(p \land q) = (\neg p) \lor (\neg q)$ }
$\neg(p \lor q) = (\neg p) \land (\neg q).$ } De morgan's law.

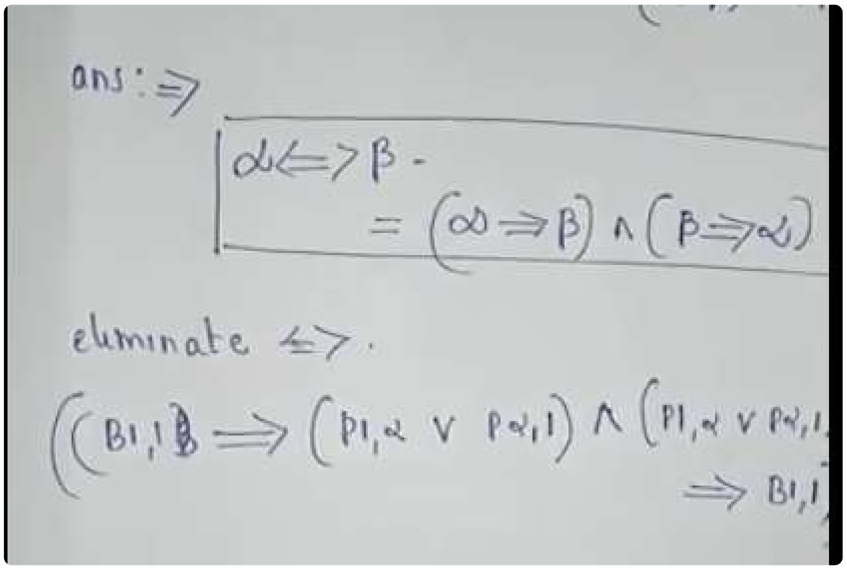
CNF (Conjunctive Normal Form)

- **Conjunctive Normal Form (CNF)** is a way of writing logical statements where:
  - The statement is a series of **AND** operations (denoted by $\land$) connecting several **OR** operations (denoted by $\lor$).
- In simpler terms, it's an **AND** of **OR** clauses.
- **Example 1**
  - Consider the statement:
  - $(A \lor B) \land (C \lor D)$
  - This is already in CNF because it's an AND ($\land$) of two OR clauses: $(A \lor B)$ and $(C \lor D)$.
  - **Example 2**
    - 

      eg:- $(p \lor \neg q) \land (\neg p \lor \neg r)$

Example

- Convert the sentence B1 1 <=> (P1,2 V p2 1) into CNF
- Lets change the implies sign
  - if A <=>B
    - = (A=>B) ^ (B=>A)

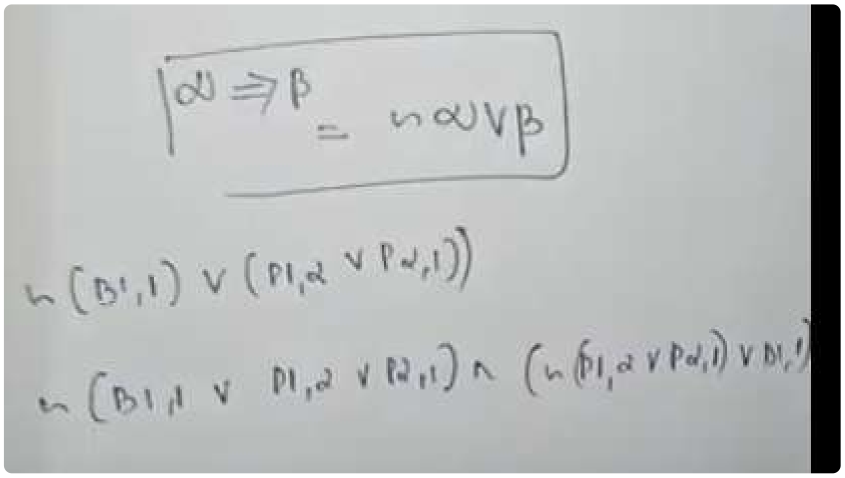
- Applying this
    - 

      ans: ⇒

      $$\boxed{\alpha \Leftrightarrow \beta = (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}$$

      eliminate ⇔.

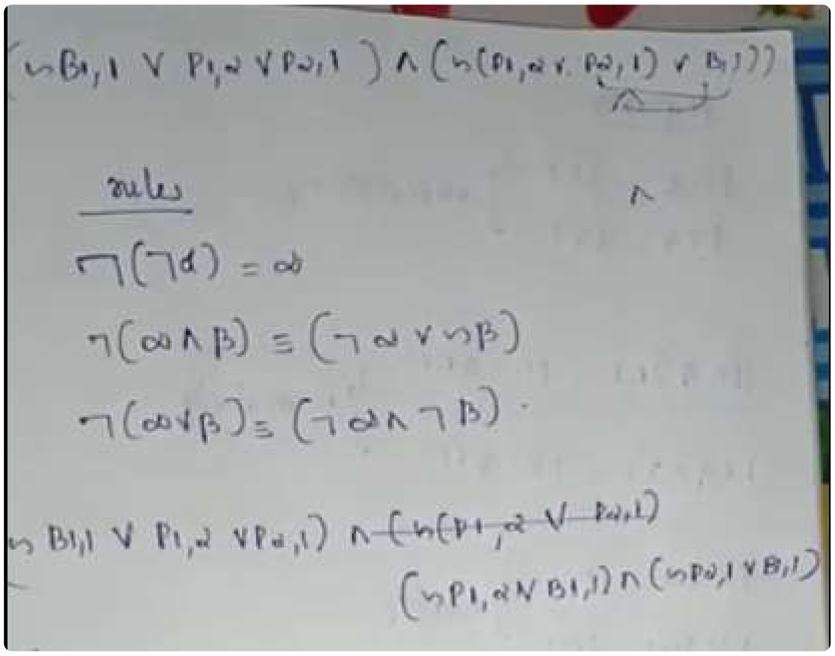
      $((B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1}) \wedge (P_{1,2} \vee P_{2,1}$

      $\Rightarrow B_{1,1}$
- if A => B
    - = ¬AVB
- Applying this
    - 

      $$\boxed{\alpha \Rightarrow \beta = \neg\alpha \vee \beta}$$

      $\neg(B_{1,1}) \vee (P_{1,2} \vee P_{2,1}))$

      $\neg(B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
- Applying distributive property

- 

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1}))$

rules

$\neg(\neg\alpha) = \alpha$

$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$

$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$

$\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge$ ~~$(\neg(P_{1,2} \vee P_{2,1})$~~

$(\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

---

7. Forward Chaining and Backward Chaining

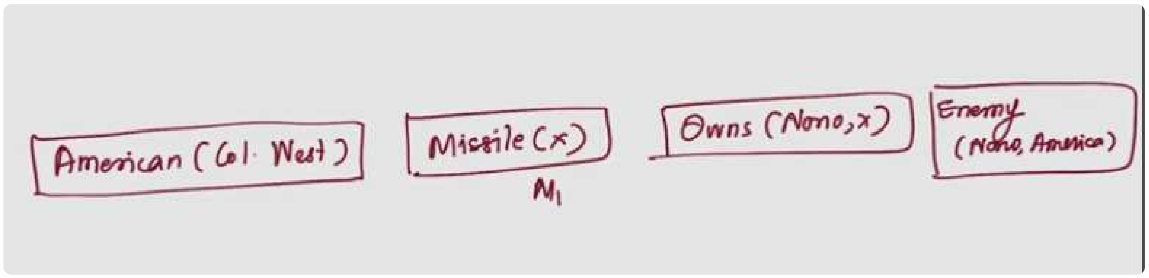
Inference Engine

- Component of the intelligent system that applies logical rules to the knowledge base to infer new information from known facts
- Proceeds in two modes
  - Forward chaining
  - Backward chaining

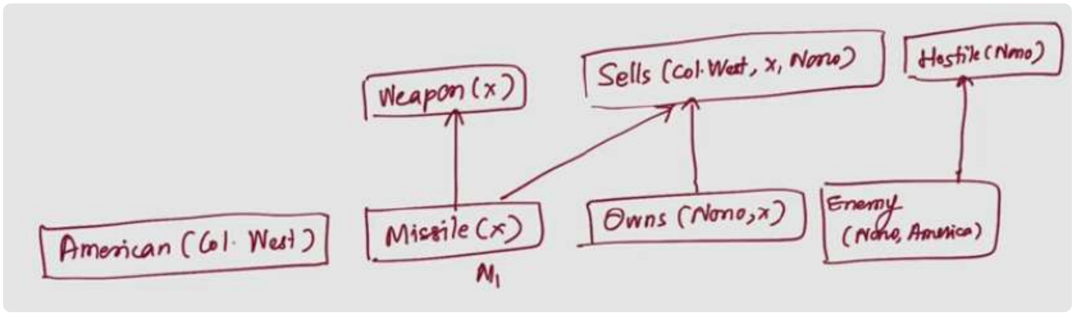
Forward chaining

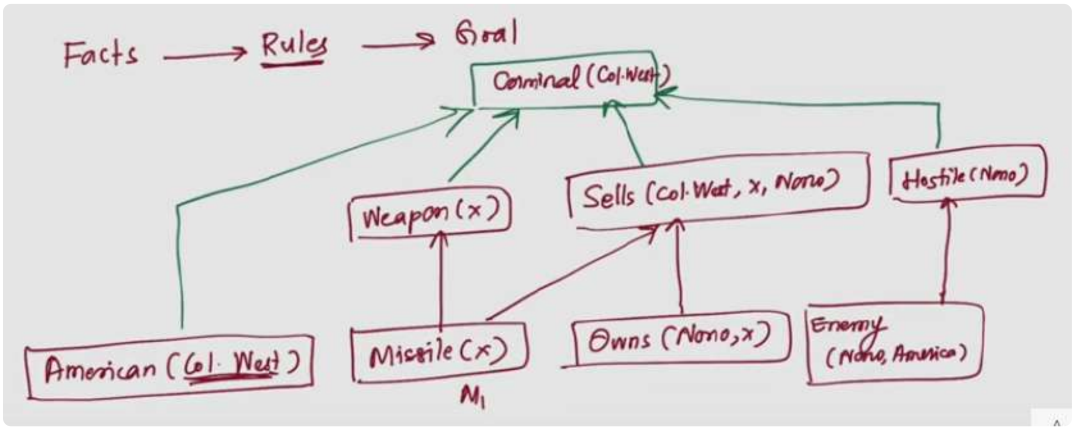
- Starts with atomic sentences in the knowledge base and applies inference rules in the forward direction to extract more data until a goal is reached.
- Starts with known facts, triggers all rules whose premises are satisfied and add their conclusions to the known facts. This repeats until problem is solved
- Example
  - The law says that it is a crime for an american to sell weapons to hostile nations. The country Nono, an enemy of america has some missiles and all of its missiles were sold by Col West who is an american.
  - Prove that Col West is a criminal

- Take each statement one by one
- 1. American(X) ^ Weapon(y) ^ Sells(x,y,z) ^ Hostile(z) -> Criminal(x)
- 2. Own(Nono,X)
- 3. Missiles(X)
- 4. Missiles(X) ^ Own(Nono,X) -> Sells(Col West,X,Nono)
- 5. Missile(X) -> Weapon(X)
- 6. Enemy(X,America) -> Hostile(X)
- 7. American(Col West)
- 8. Enemy(Nono,America)

- We basically find the facts, then the rules and form the goals
- **Facts are those without implication (->)**
  - Facts are
    - American(Col West)
    - Missile(X)
    - Owns(Nono,X)
    - Enemy(Nono,American)
  - 

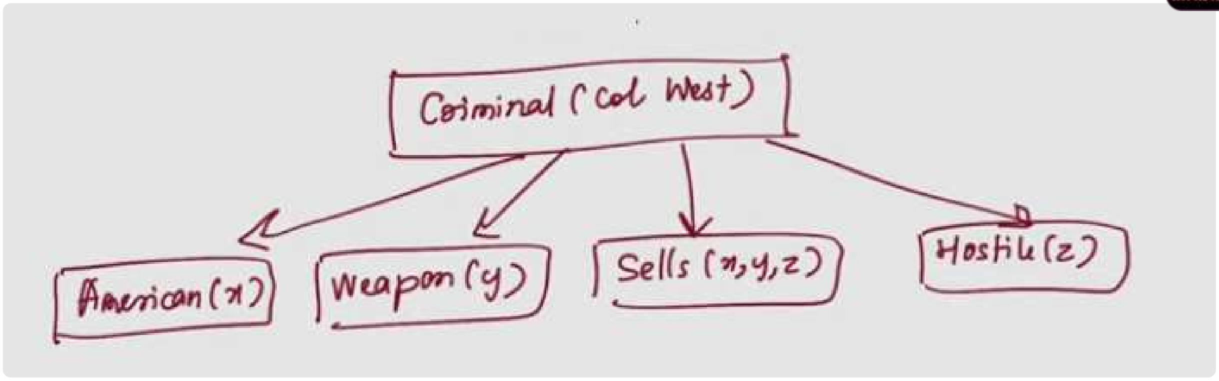
- **Lets FInd the Rules**
  - Take each facts one by one and look where its being used (Refer the numbers of each statement)
    - 7. American is related to 1. American
    - 3. Missiles is related to 5. Missile
    - 2. Own is related to 4. Own
    - 8. Enemy is related to 6. Enemy
  - Ignoring 7->1, since 1st statement is the conclusion
  - Taking 3->5
    - Missile is linked to weapon
  - 2->4

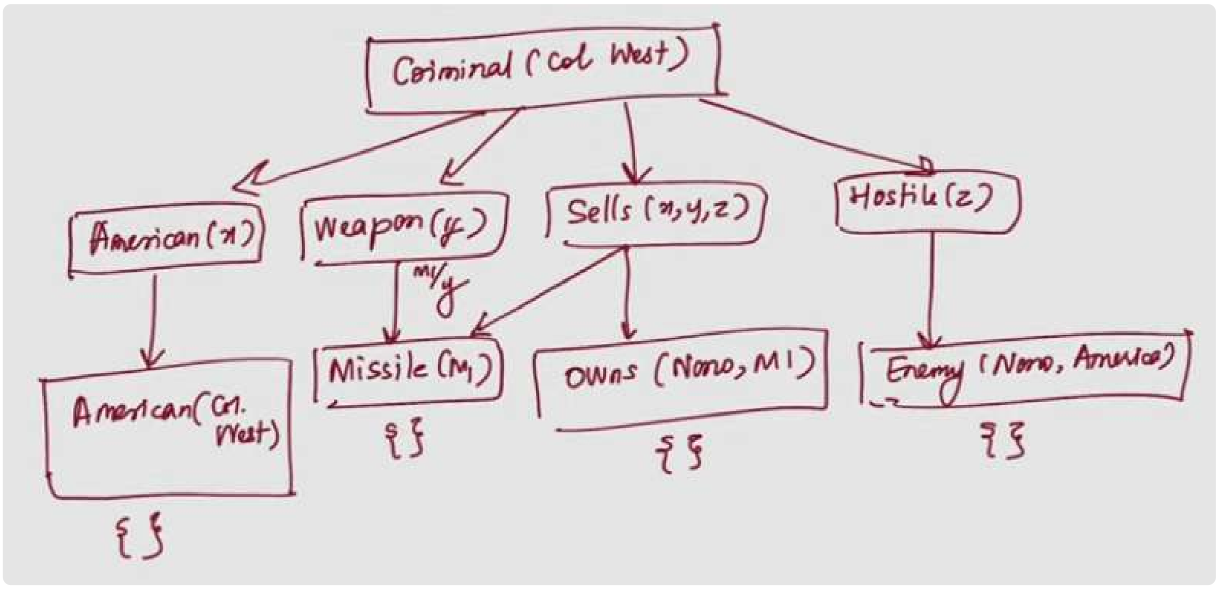
- Missile and Owns is linked to Sells
- 8->6
    - Enemy is linked to Hostile
- The rules are
    - 

- Goal State
    - Taking Statement 1
        - 1. American(X) ^ Weapon(y) ^ Sells(x,y,z) ^ Hostile(z) -> Criminal(x)
        - Connecting all the boxes to Criminal, we get
    - 

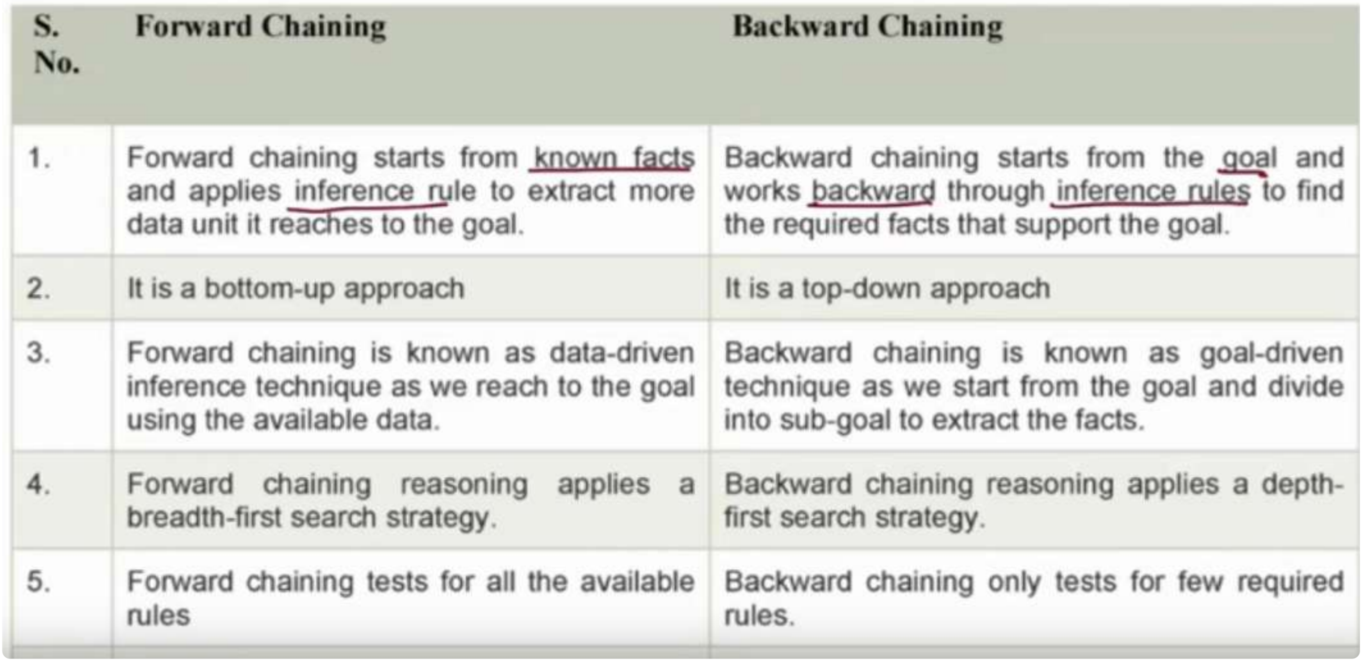
Backward chaining

- Considers the item to be proven a goal.
- We need to find the subgoals from a given gaol.
    - Taking the same example as before.
    - The goal is Col West is a criminal
- Take the Statement
    - American(X) ^ Weapon(y) ^ Sells(x,y,z) ^ Hostile(z) -> Criminal(x)
    - From this we can get the subgoals

- 

- Now lets find the subgoals for American
    - American(Col West)
- Subgoals for weapon
    - Missile(X) -> Weapon(X)
    - Missile (X)
- Subgoals for Sells
    - Missiles(X) ^ Own(Nono,X) -> Sells(Col West,X,Nono)
    - Missiles(X)
    - Own(Nono,X)
- Hostile
    - Enemy(X,America) -> Hostile(X)
- 

Difference between Forward and Backward Chaining



S. No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.