The advantage of bidirectional search in artificial intelligence is
a. Faster
b. Reduced exploration
c. Both a & *b*
d. None of the above

The metric used to measure the performance of the Bidirectional Search is

a. Completeness

b. Optimality

c. Time and space complexity

d. All of the Abov*e*

Program generator is present in _____

a) Goal based agent
b) Simple reflex agent
c) Learning agen*t*
d) Utility based agent

An algorithm A is admissible if ___________

a) It is not guaranteed to return an optimal solution when one exists

b) It is guaranteed to return an optimal solution when one exists

c) It returns more solutions, but not an optimal one

d) It guarantees to return more optimal solutions

A search algorithm takes _________ as an input and returns ________ as an output.

a.

Input, output

b.

Problem, *solution*

c.

Solution, problem

d.

Parameters, sequence of actions

A technique that was developed to determine whether a machine could or could not demonstrate the artificial intelligence known as the___

A)Boolean Algebra

B)Turing Test

C)Logarithm

D)Algorithm

Resolution

- Resolution produces proof by refutation. i.e. ***to prove a statement, resolution attempts to show that the negation of statement produces a contradiction with known statements.***
- This approach contrast with technique that we have been using to generate proofs by chaining backward from theorem to be proved axioms.
- It operates on statements that have been converted to a very convenient standard form.
- The formula would be easier to work with if
  - *It were flatter i.e. there was less embedding of components.*
  - *The quantifiers were separated from the rest of formula so they did not need to be consider.*

Resolution

Clause: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.

Conjunctive Normal Form: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**.

Resolution Method in Propositional Logic

The process followed to convert the propositional logic into resolution method contains the below steps:

1. Convert the given axiom into clausal form, i.e., disjunction form.
2. Apply and proof the given goal using negation rule.
3. Use those literals which are needed to prove.
4. Solve the clauses together and achieve the goal.

Algorithm : Propositional Resolution

1. Convert all the propositions of *F* to clause form.

2. Negate *P* and convert the result to clause form. Add it to the set of clauses obtained in step 1.

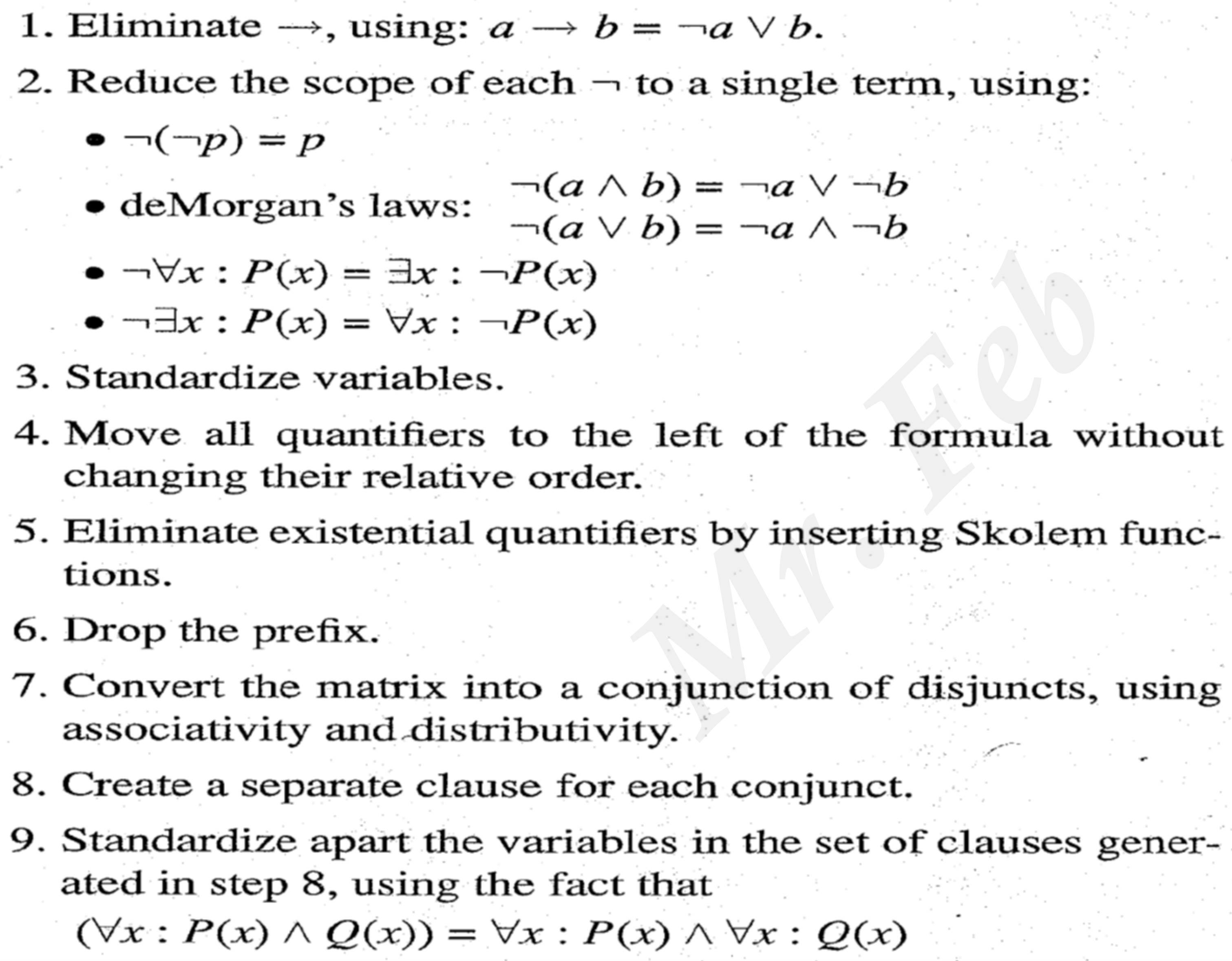
3. Repeat until either a contradiction is found or no progress can be made:

(a) Select two clauses. Call these the parent clauses.

(b) Resolve them together. The resulting clause, called the *resolvent,* will be the disjunction of all of the literals of both of the parent clauses with the following exception: If there are any pairs of literals *L* and ¬*L* such that one of the parent clauses contains *L* and the other contains ¬*L,* then select one such pair and eliminate both *L* and ¬*L* from the resolvent.

(c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

Algorithm : Convert to Clause Form



1. Eliminate $\rightarrow$, using: $a \rightarrow b = \neg a \vee b$.
2. Reduce the scope of each $\neg$ to a single term, using:
   - $\neg(\neg p) = p$
   - deMorgan's laws: $\neg(a \wedge b) = \neg a \vee \neg b$
     $\neg(a \vee b) = \neg a \wedge \neg b$
   - $\neg \forall x : P(x) = \exists x : \neg P(x)$
   - $\neg \exists x : P(x) = \forall x : \neg P(x)$
3. Standardize variables.
4. Move all quantifiers to the left of the formula without changing their relative order.
5. Eliminate existential quantifiers by inserting Skolem functions.
6. Drop the prefix.
7. Convert the matrix into a conjunction of disjuncts, using associativity and distributivity.
8. Create a separate clause for each conjunct.
9. Standardize apart the variables in the set of clauses generated in step 8, using the fact that
   $(\forall x : P(x) \wedge Q(x)) = \forall x : P(x) \wedge \forall x : Q(x)$

Resolution Method in FOPl/ Predicate Logic

Resolution method in FOPL is an uplifted version of propositional resolution method.

In FOPL, the process to apply the resolution method is as follows:

1. Convert the given axiom into CNF, i.e., a conjunction of clauses. Each clause should be dis-junction of literals.
2. Apply negation on the goal given.
3. Use literals which are required and prove it.
4. Unlike propositional logic, FOPL literals are complementary if one unifies with the negation of other literal.

Algorithm : Resolution

1. Convert all the statements of *F* to clause form.

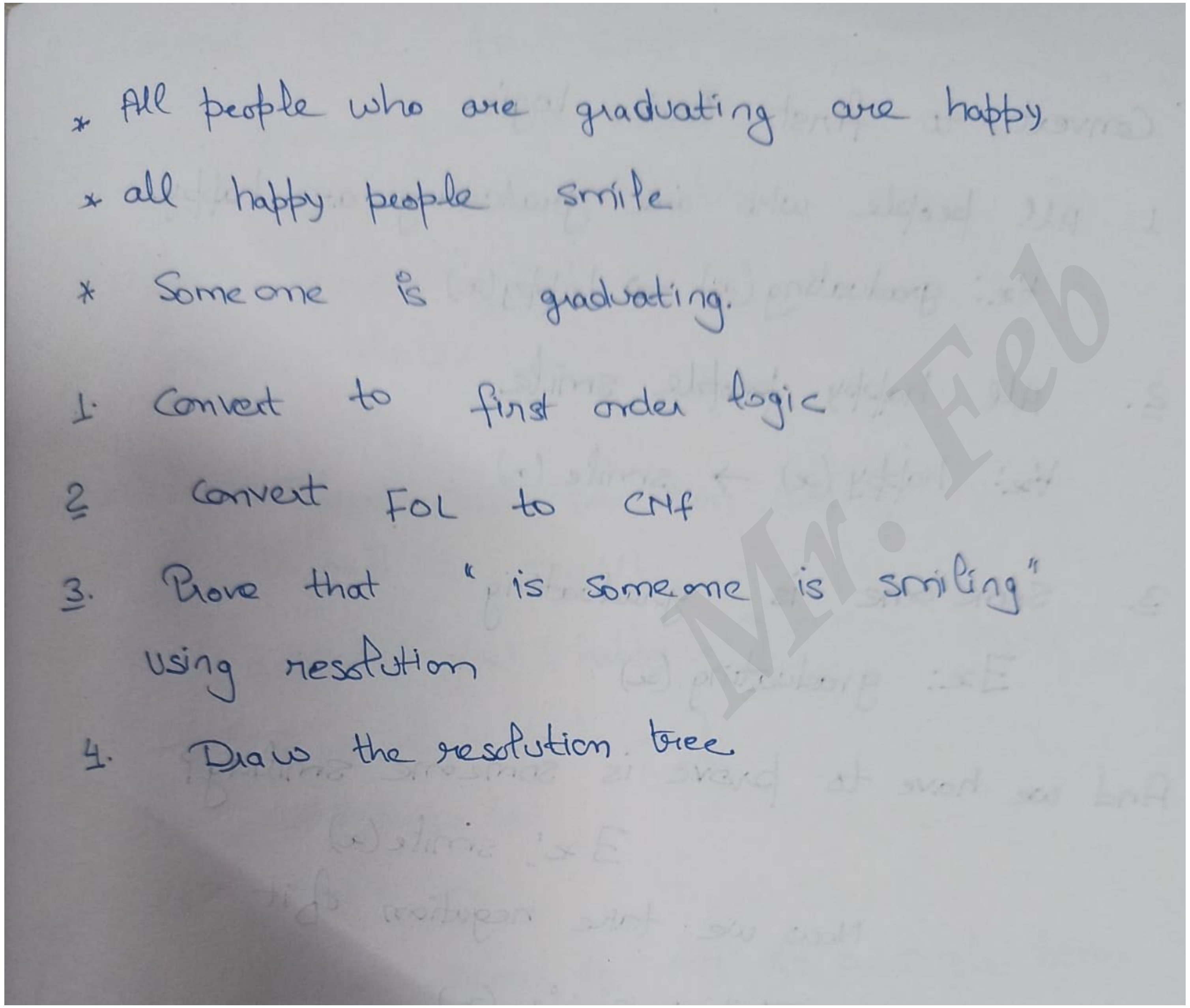
2. Negate *P* and convert the result to clause form. Add it to the set of clauses obtained in 1.

3. Repeat until either a contradiction is found, no progress can be made, or a predetermined amount of effort has been expended.

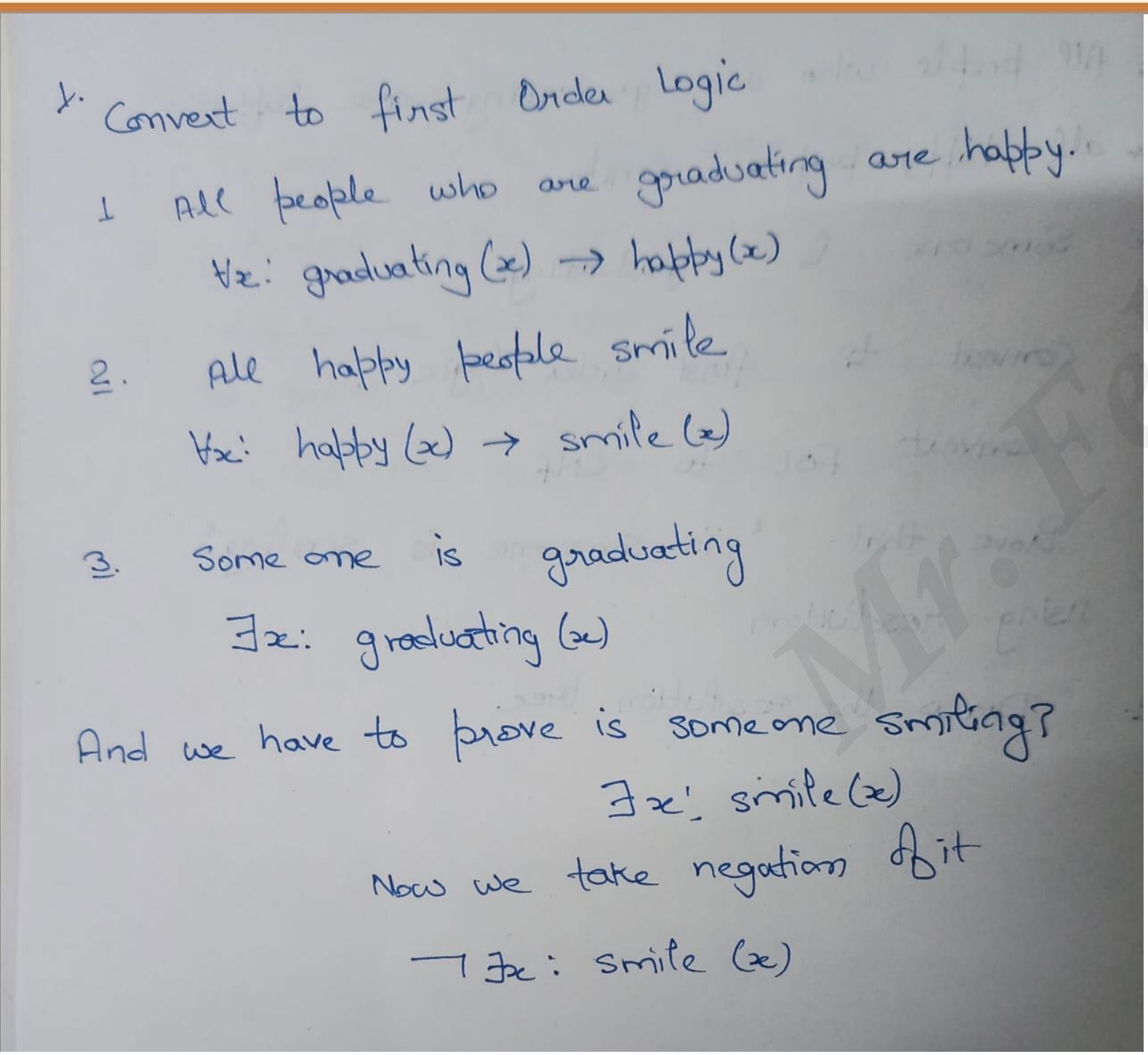
(a) Select two clauses. Call these the parent clauses.

(b) Resolve them together. The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exception: If there is one pair of literals *T*1 and ¬*T*2 such that one of the parent clauses contains *T*2 and the other contains *T*1 and if *T*1 and *T*2 are unifiable, then neither *T*1 nor *T*2 should appear in the resolvent. If there is more than one pair of complimentary literals, only one pair shold be omitted from the resolvent.

(c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.


* All people who are graduating are happy
* all happy people smile
* Someone is graduating.

1. Convert to first order logic
2. Convert FOL to CNF
3. Prove that " is someone is smiling" using resolution
4. Draw the resolution tree



1. Convert to first Order Logic

1 All people who are graduating are happy.

$\forall x: graduating(x) \rightarrow happy(x)$

2. All happy people smile

$\forall x: happy(x) \rightarrow smile(x)$

3. Some one is graduating

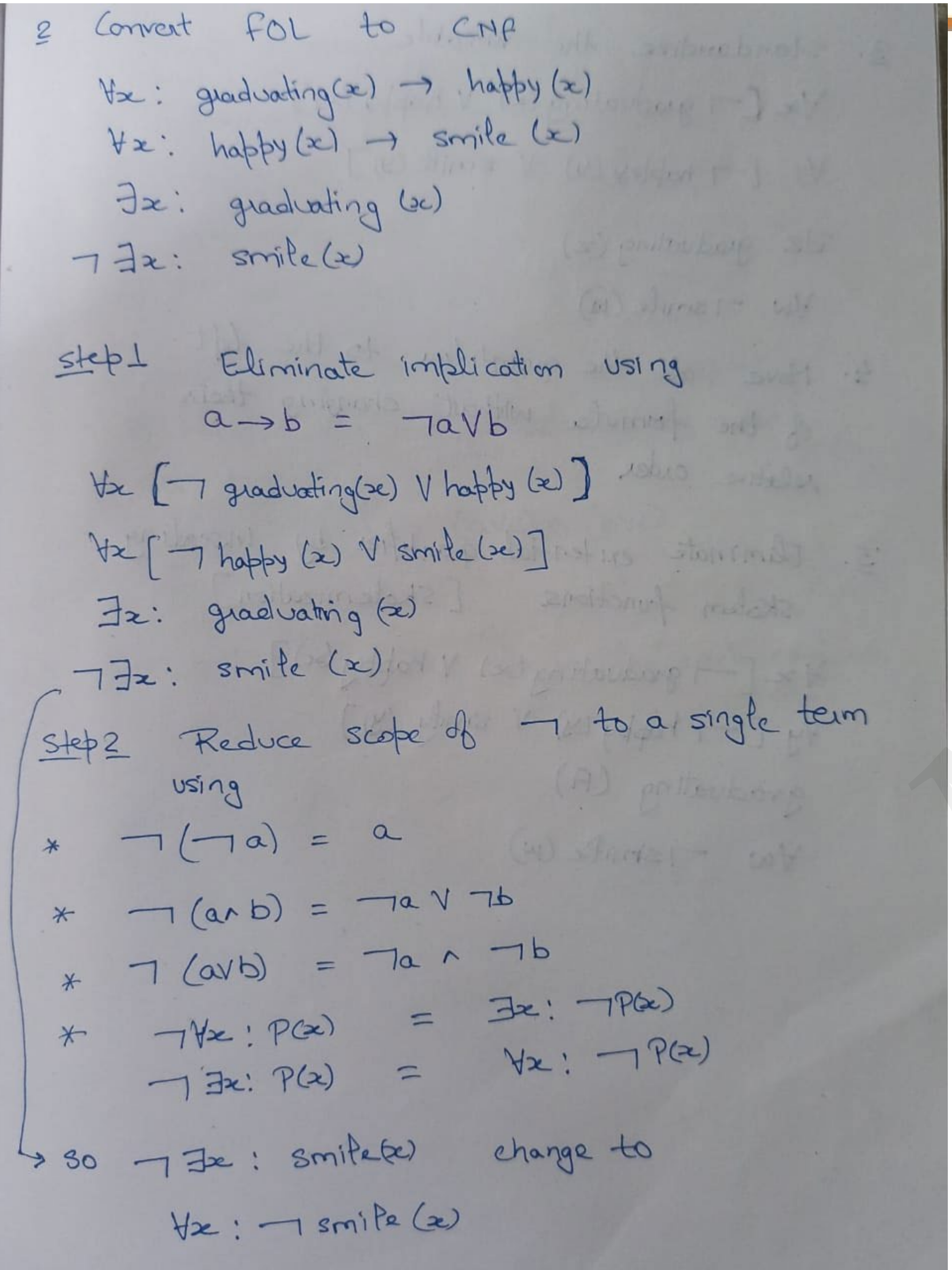
$\exists x: graduating(x)$

And we have to prove is someone smiling?

$\exists x: smile(x)$

Now we take negation of it

$\neg \exists x: smile(x)$



2 Convert FOL to CNF

$\forall x: \text{graduating}(x) \rightarrow \text{happy}(x)$

$\forall x: \text{happy}(x) \rightarrow \text{smile}(x)$

$\exists x: \text{graduating}(x)$

$\neg \exists x: \text{smile}(x)$

<u>step 1</u> Eliminate implication using

$a \rightarrow b = \neg a \vee b$

$\forall x [\neg \text{graduating}(x) \vee \text{happy}(x)]$

$\forall x [\neg \text{happy}(x) \vee \text{smile}(x)]$

$\exists x: \text{graduating}(x)$

$\neg \exists x: \text{smile}(x)$

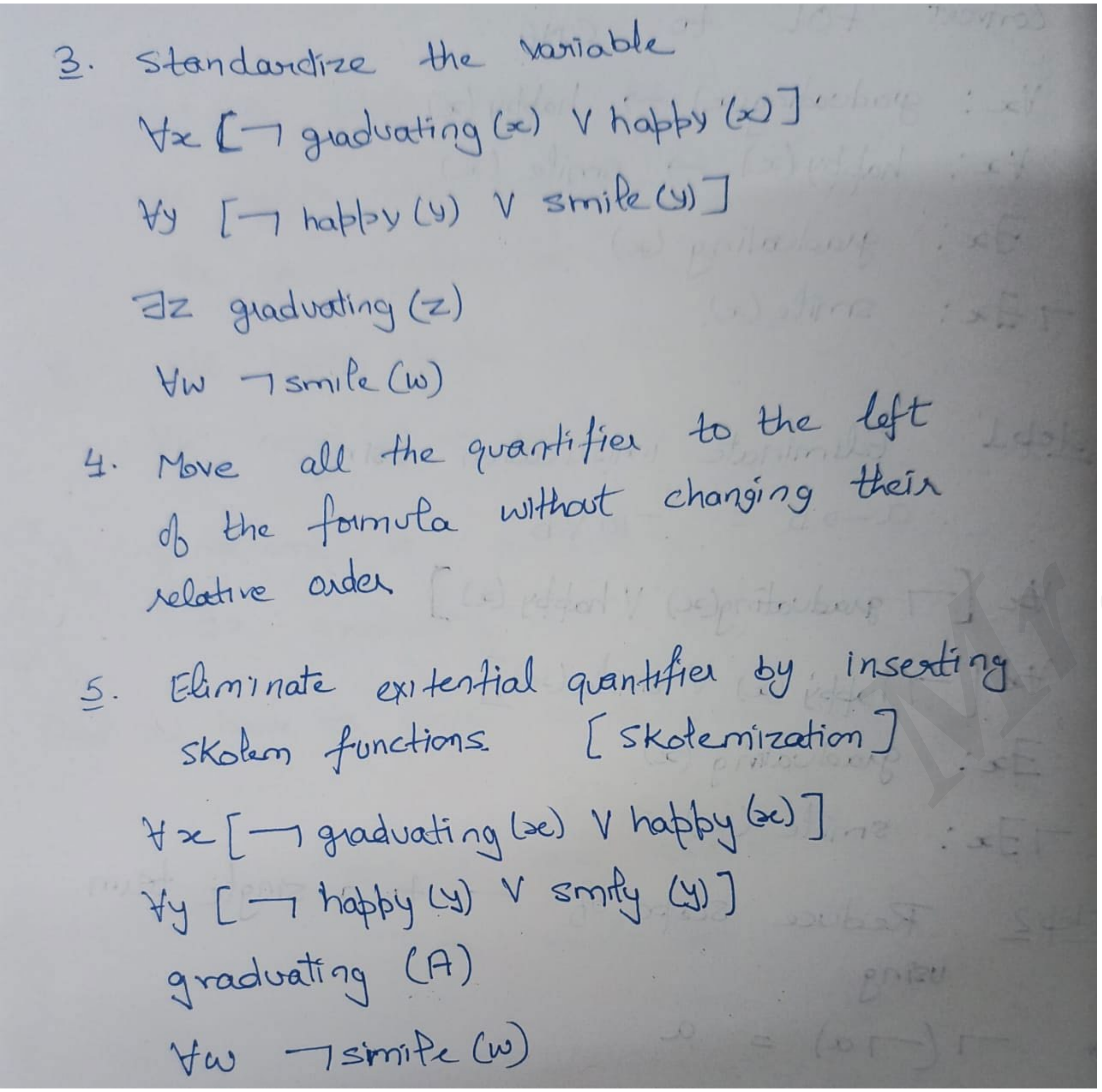
<u>Step 2</u> Reduce scope of $\neg$ to a single term using

* $\neg(\neg a) = a$
* $\neg(a \wedge b) = \neg a \vee \neg b$
* $\neg(a \vee b) = \neg a \wedge \neg b$
* $\neg \forall x: P(x) = \exists x: \neg P(x)$

  $\neg \exists x: P(x) = \forall x: \neg P(x)$

so $\neg \exists x: \text{smile}(x)$ change to

$\forall x: \neg \text{smile}(x)$



3. Standardize the variable

$\forall x\ [\neg\ \text{graduating}(x) \lor \text{happy}(x)]$

$\forall y\ [\neg\ \text{happy}(y) \lor \text{smile}(y)]$

$\exists z\ \text{graduating}(z)$

$\forall w\ \neg\text{smile}(w)$

4. Move all the quantifier to the left of the formula without changing their relative order

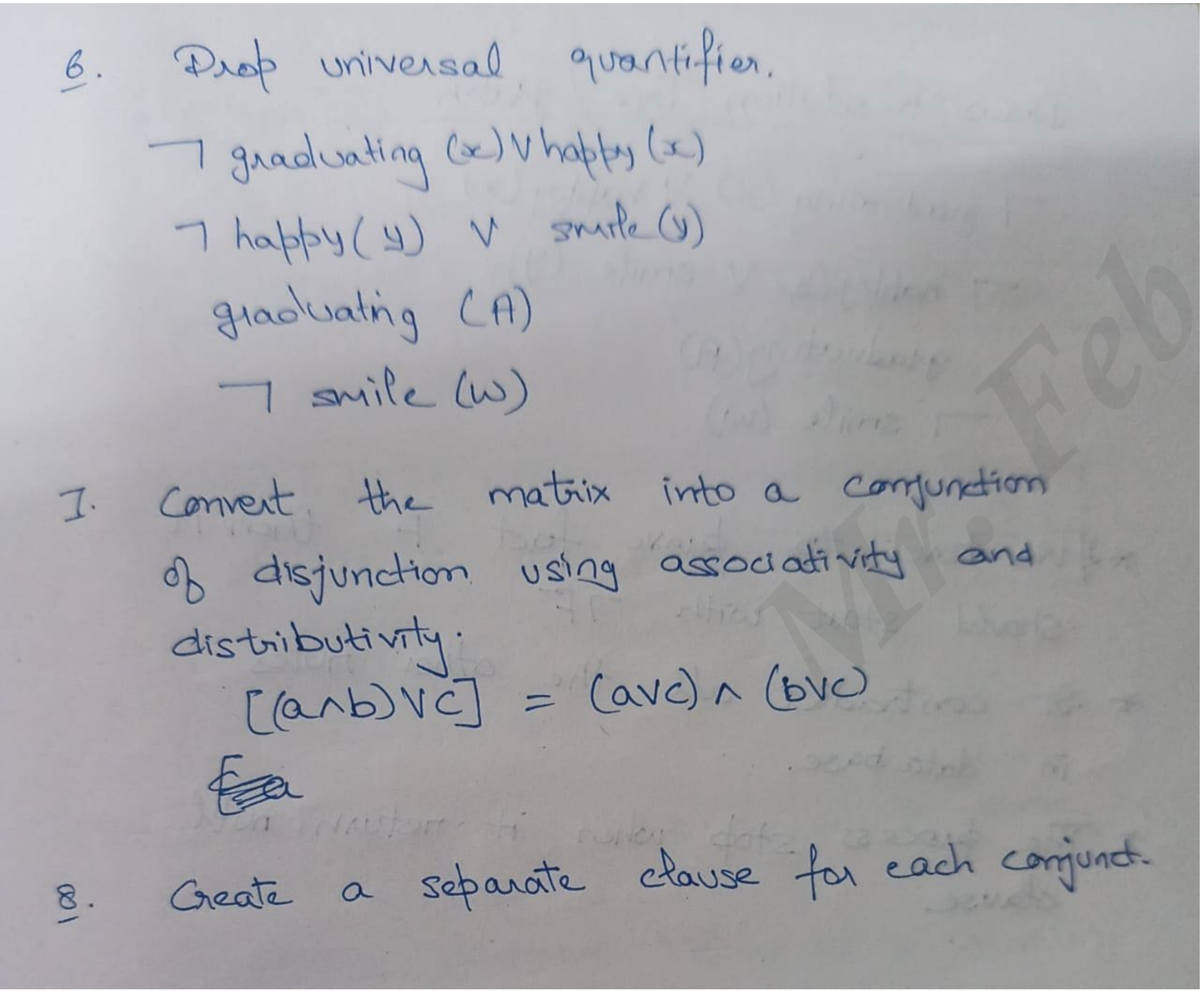
5. Eliminate exitential quantifier by inserting skolem functions [Skolemization]

$\forall x\ [\neg\ \text{graduating}(x) \lor \text{happy}(x)]$

$\forall y\ [\neg\ \text{happy}(y) \lor \text{smify}(y)]$

$\text{graduating}(A)$

$\forall w\ \neg\text{smile}(w)$



6. Drop universal quantifier.

$\neg$ graduating (x) $\vee$ happy (x)

$\neg$ happy (y) $\vee$ smile (y)

graduating (A)

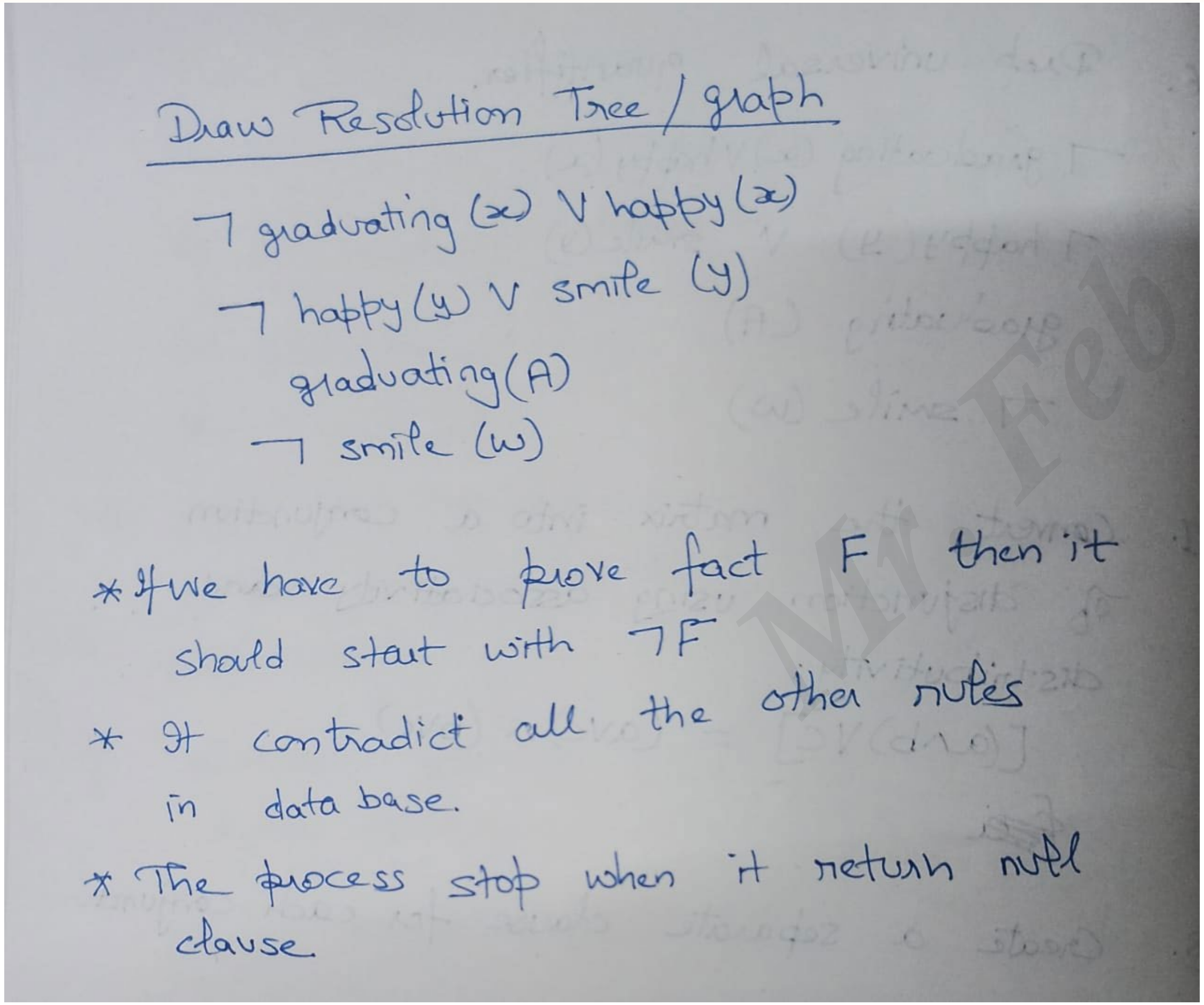
$\neg$ smile (w)

7. Convert the matrix into a conjunction of disjunction using associativity and distributivity.

$[(a \wedge b) \vee c] = (a \vee c) \wedge (b \vee c)$

~~Ea~~

8. Create a separate clause for each conjunct.



Draw Resolution Tree / graph

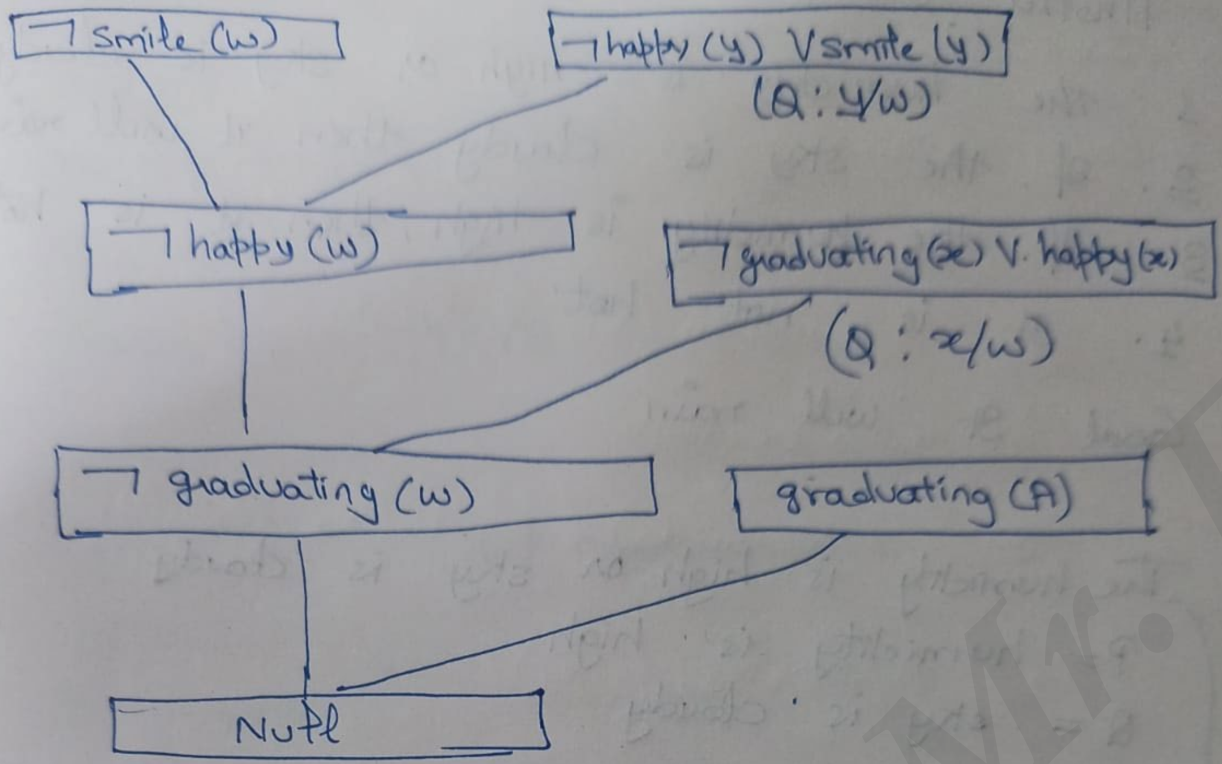
$\neg$ graduating (x) $\vee$ happy (x)

$\neg$ happy (y) $\vee$ smile (y)

graduating(A)

$\neg$ smile (w)

* If we have to prove fact F then it should start with $\neg F$
* It contradict all the other rules in data base.
* The process stop when it return null clause.



Hence we can say ¬ smile (w) ie. some is not smiling is wrong.
and given statement "some one is smiling" is correct.

Example of FOPL resolution

Consider the following knowledge base:

1. Gita likes all kinds of food.
2. Mango and chapati are food.
3. Gita eats almond and is still alive.
4. Anything eaten by anyone and is still alive is food.

Goal: Gita likes almond.

Resolution Method in Propositional Logic

Consider the following Knowledge Base:

1. The humidity is high or the sky is cloudy.
2. If the sky is cloudy, then it will rain.
3. If the humidity is high, then it is hot.
4. It is not hot.

Goal: It will rain

Resolution Method in Propositional Logic

1. *The humidity is high or the sky is cloudy.*
Let, P: Humidity is high.
Q: Sky is cloudy.
It will be represented as **P V Q.**

2. If the sky is cloudy, then it will rain.
Let, R: It will rain.
It will be represented as be **Q → R.**

3. If the humidity is high, then it is hot
P: Humidity is high. …**from(1)**
Let, S: It is hot.
It will be represented as **P → S.**

4. It is not hot.
¬S: It is not hot.

Resolution Method in Propositional Logic

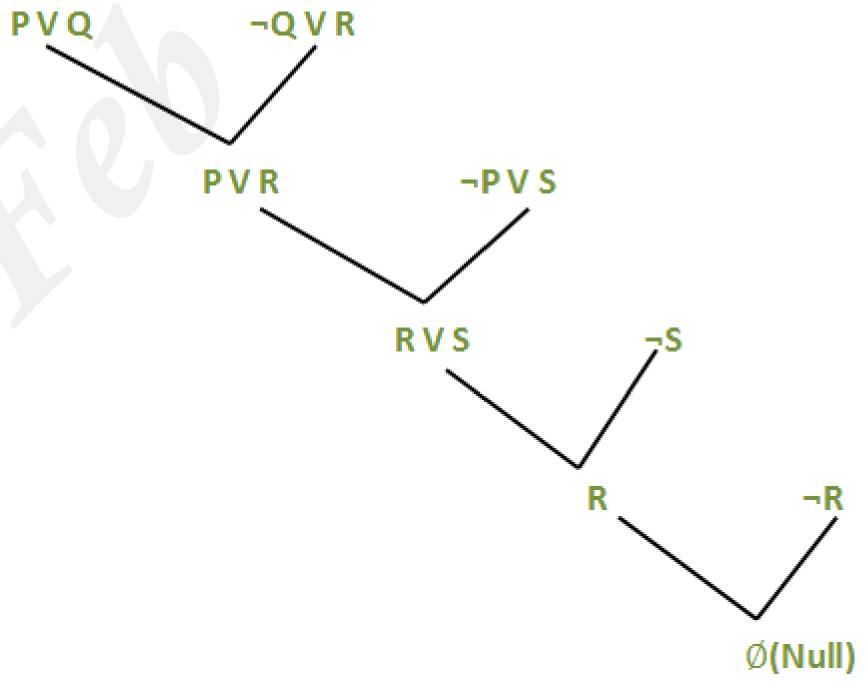
Applying resolution method:

In (2), Q → R will be converted as (¬Q V R)

In (3), P → S will be converted as (¬P V S)

Negation of Goal (¬R): It will not rain.

After applying Proof by Refutation (Contradiction) on the goal, the problem is solved, and it has terminated with a **Null clause (Ø).** Hence, the goal is achieved. Thus, It is not raining.



Another Example

1.If it rains, Joe brings his umbrella
2.If Joe has an umbrella, he doesn't get wet
3.If it doesn't rain, Joe doesn't get wet
4.prove that Joes doesn't get wet

Another Example

Given Axioms	Converted to Clause Form	
P	P	(1)
$(P \wedge Q) \rightarrow R$	$\neg P \vee \neg Q \vee R$	(2)
$(S \vee T) \rightarrow Q$	$\neg S \vee Q$	(3)
	$\neg T \vee Q$	(4)
T	T	(5)

Goal: we want to prove R.

Another Example

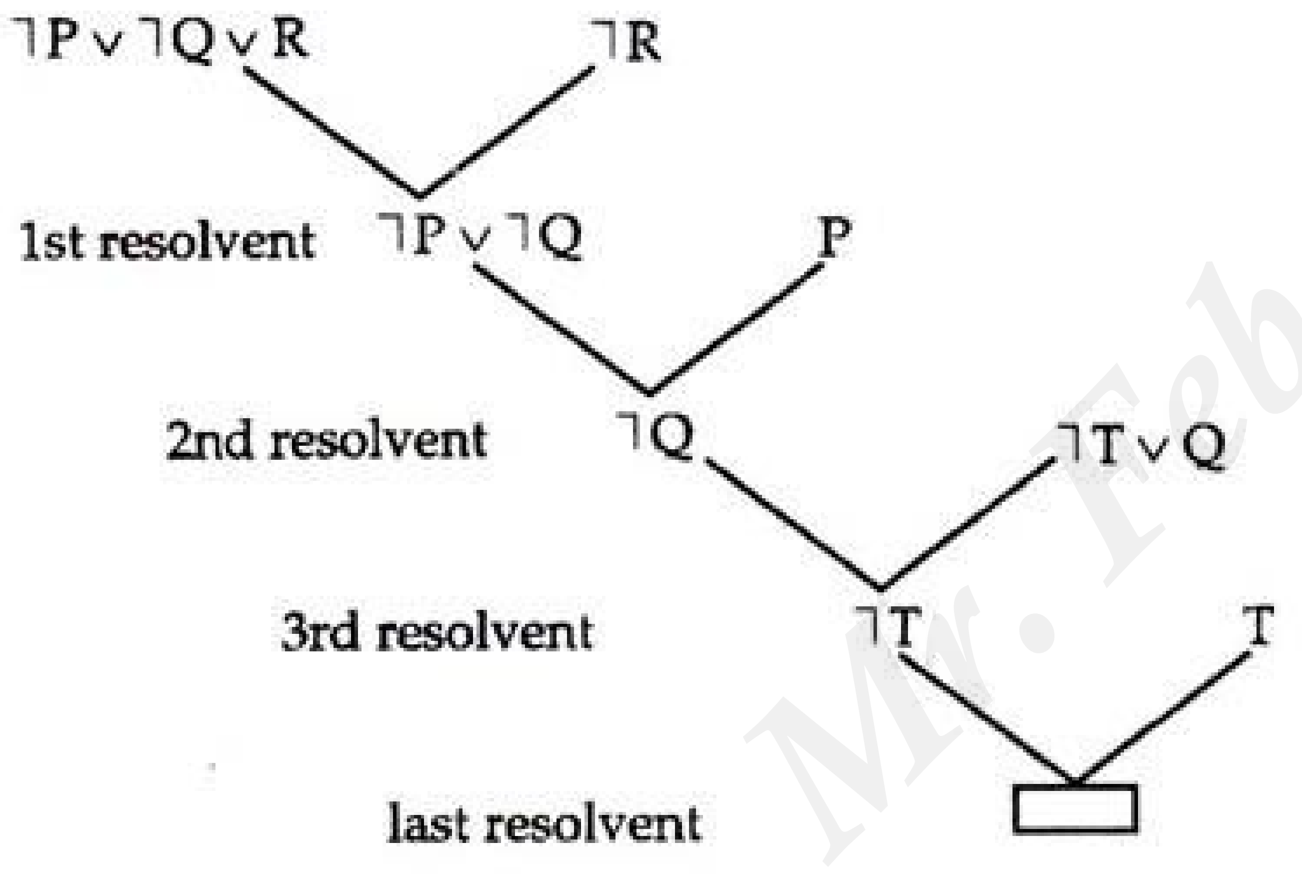


Fig. 6.5 : *Resolution process in Propositional logic*

Unification

Unification is a matching procedure that compares two literals and discover whether there exists a set of substitutions that makes them identical.

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 two similar variables present in the same expression.

Finding General Substitutions

Given :

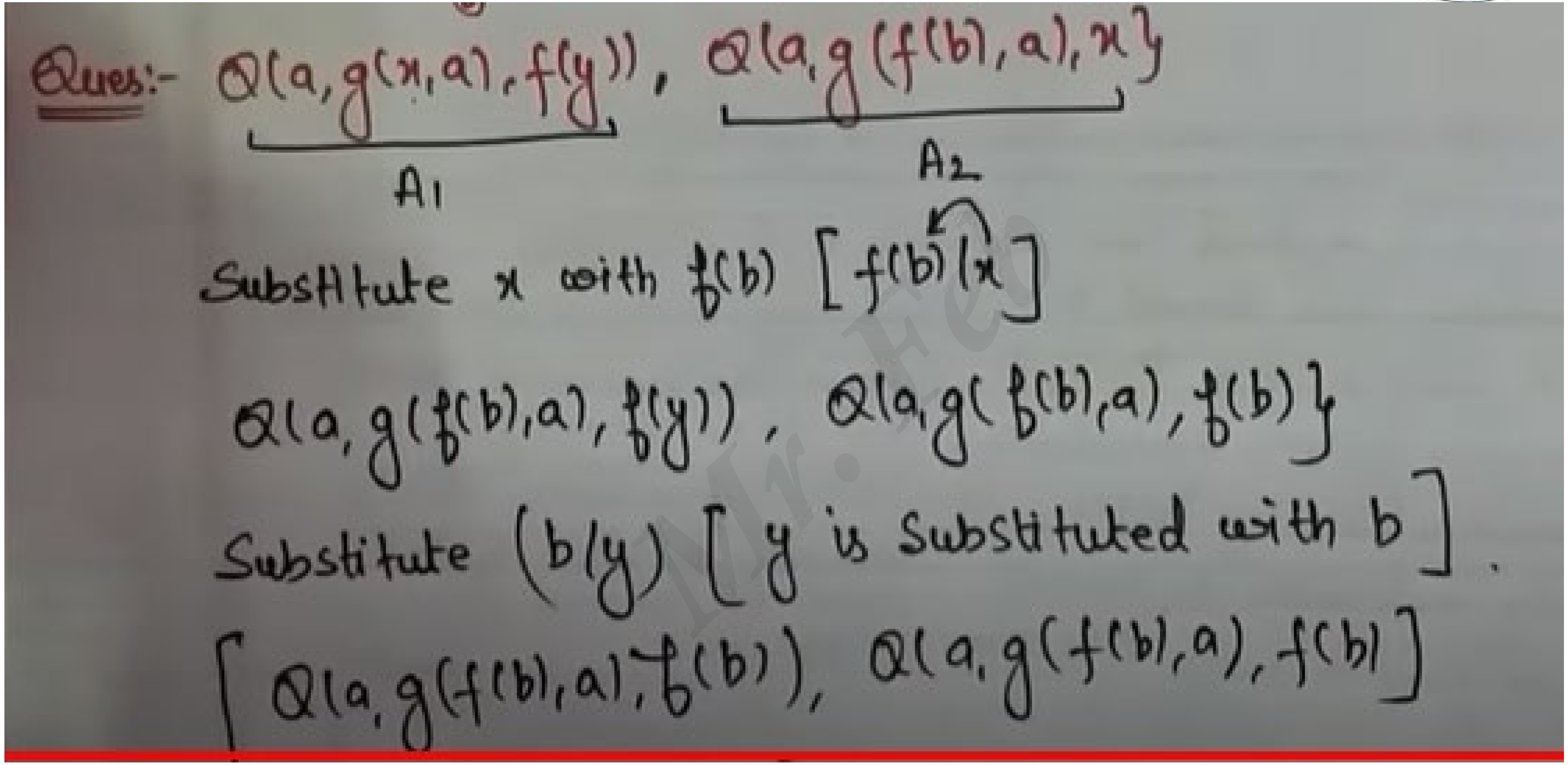
$hate(x,\ y)$
$hate(Marcus,\ z)$

We could produce :

$(Marcus/x, z/y)$
$(Marcus/x, y/z)$

Algorithm : Unify (L1, L2)

```
1. If L1 or L2 are both variables or constants, then:
   (a) If L1 and L2 are identical, then return NIL.
   (b) Else if L1 is a variable, then if L1 occurs in L2 then return {FAIL}, else   return (L2/L1).
   (c) Else if L2 is a variable then if L2 occurs in L1 then return {FAIL}, else        return (L1/L2).
   (d) Else return {FAIL}.
2. If the initial predicate symbols in L1 and L2 are not identical, then return {FAIL).
3. If LI and L2 have a different number of arguments, then return {FAIL}.
4. Set SUBST to NIL.
5. For i ← 1 to number of arguments in L1:
   (a) Call Unify with the /th argument of L1 and the ith argument of L2, putting result in S.
   (b) If S contains FAIL then return {FAIL}.
   (c) If S is not equal to NIL then:
           (i) Apply S to the remainder of both L1 and L2.
           (ii) SUBST : = APPEND(S, SUBST).
6. Return SUBST.
```



Ques:- $\underbrace{Q(a, g(x, a), f(y))}_{A_1}$, $\underbrace{Q(a, g(f(b), a), x\}}_{A_2}$

Substitute x with $f(b)$ $[f(b)/x]$

$Q(a, g(f(b), a), f(y))$, $Q(a, g(f(b), a), f(b)\}$

Substitute (b/y) [y is Substituted with b].

$[Q(a, g(f(b), a), f(b))$, $Q(a, g(f(b), a), f(b)]$

Resolution Method in FOPl/ Predicate Logic

Example:

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)

g. ∀x: alive(x) → ¬ killed(x)

$\}$ **added predicates.**

h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.

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)

g. ∀x: alive(x) →¬ killed(x)

added predicates.

h. likes(John, Peanuts)

- **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(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).

- **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(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).

- **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).

- **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).

$\Longrightarrow$

- **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).

•**Eliminate existential instantiation quantifier by elimination.**

In this step, we will eliminate existential quantifier ∃, and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.

Drop Universal quantifiers.
In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.

- **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).

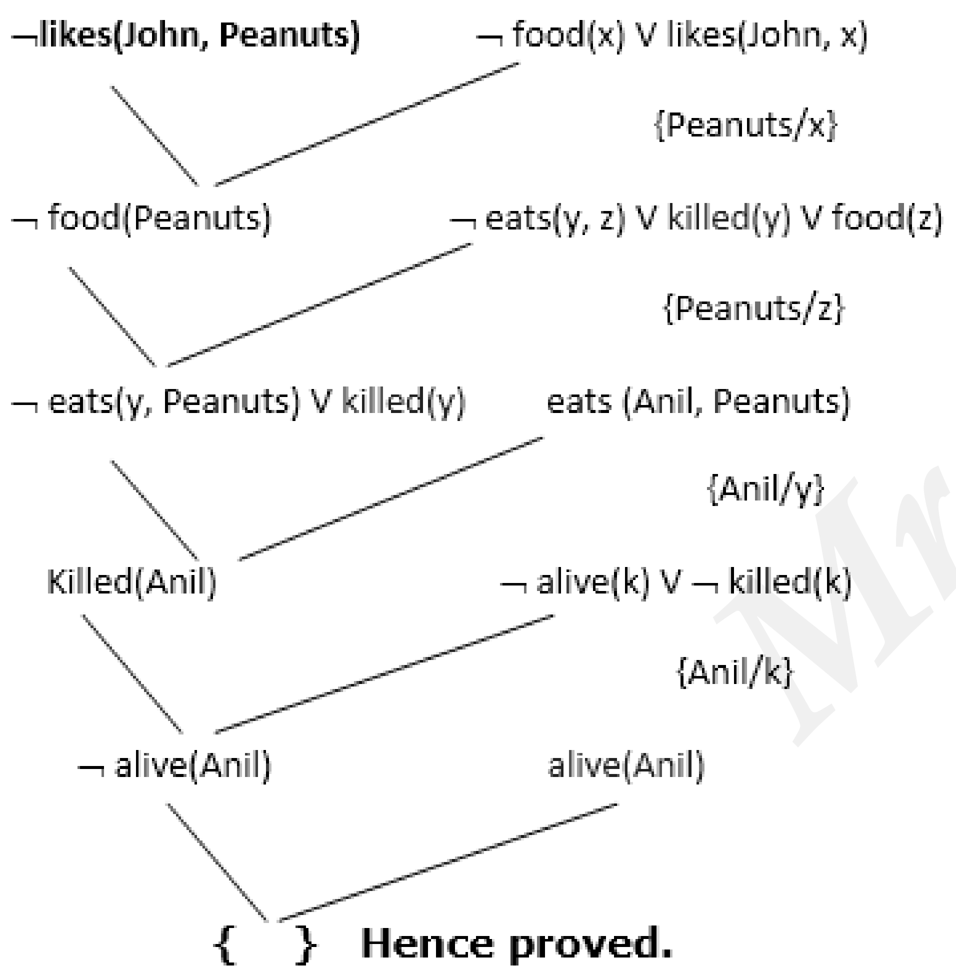
a. ¬ food(x) V likes(John, x)
b. food(Apple)
c. food(vegetables)
d. ¬ eats(y, z) V killed(y) V food(z)
e. eats (Anil, Peanuts)
f. alive(Anil)
g. ¬ eats(Anil, w) V eats(Harry, w)
h. killed(g) V alive(g)
i. ¬ alive(k) V ¬ killed(k)
j. likes(John, Peanuts).

•**Distribute conjunction ∧ over disjunction ¬.** This step will not make any change in this problem.

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)

Step-4: Draw Resolution graph:



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).

Nil means negation is a contradictory statement and given goal statement is true.

Explanation of Resolution graph:

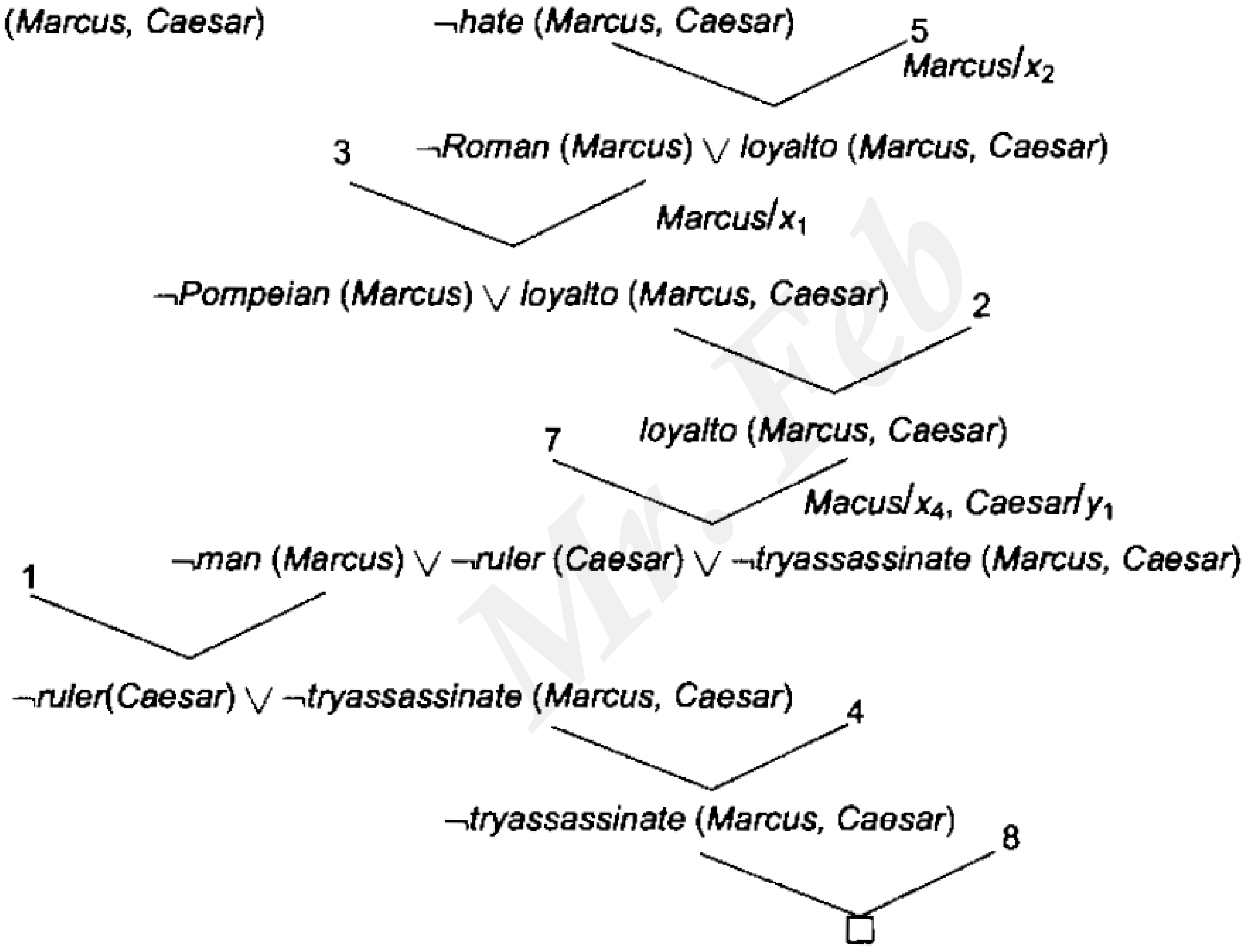
- In the first step of resolution graph, **¬likes(John, Peanuts)** , and **likes(John, x)** get resolved(canceled) by substitution of **{Peanuts/x}**, and we are left with ¬ **food(Peanuts)**
- In the second step of the resolution graph, ¬ **food(Peanuts)** , and **food(z)** get resolved (canceled) by substitution of **{ Peanuts/z}**, and we are left with ¬ **eats(y, Peanuts) V killed(y)** .
- In the third step of the resolution graph, ¬ **eats(y, Peanuts)** and **eats (Anil, Peanuts)** get resolved by substitution **{Anil/y}**, and we are left with **Killed(Anil)** .
- In the fourth step of the resolution graph, **Killed(Anil)** and ¬ **killed(k)** get resolve by substitution **{Anil/k}**, and we are left with ¬ **alive(Anil)** .
- In the last step of the resolution graph ¬ **alive(Anil)** and **alive(Anil)** get resolved.

Axioms in clause form:

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\neg Pompeian(x_1) \lor Roman(x_1)$
4. $ruler(Caesar)$
5. $\neg Roman(x_2) \lor loyalto(x_2, Caesar) \lor hate(x_2, Caesar)$
6. $loyalto(x_3, fl(x_3))$
7. $\neg man(x_4) \lor \neg ruler(y_1) \lor \neg tryassassinate(x_4, y_1) \lor loyalto(x_4, y_1)$
8. $tryassassinate(Marcus, Caesar)$

Prove: $hate\ (Marcus,\ Caesar)$

Prove: *hate* (*Marcus*, *Caesar*)

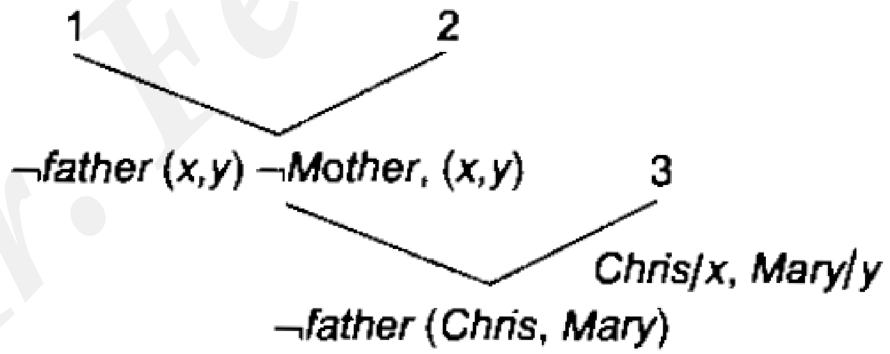

¬hate (Marcus, Caesar)
5
Marcus/x_2
¬Roman (Marcus) ∨ loyalto (Marcus, Caesar)
3
Marcus/x_1
¬Pompeian (Marcus) ∨ loyalto (Marcus, Caesar)
2
loyalto (Marcus, Caesar)
7
Macus/x_4, Caesar/y_1
¬man (Marcus) ∨ ¬ruler (Caesar) ∨ ¬tryassassinate (Marcus, Caesar)
1
¬ruler(Caesar) ∨ ¬tryassassinate (Marcus, Caesar)
4
¬tryassassinate (Marcus, Caesar)
8
□

Given:

1. $\neg father(x, y) \vee \neg woman(x)$

   (i.e., $father(x, y) \rightarrow \neg woman(x)$)
2. $\neg mother(x, y) \vee woman(x)$

   (i.e., $mother(x, y) \rightarrow woman(x)$)
3. $mother(Chris, Mary)$
4. $father(Chris, Bill)$



The Need to Standardize Variables

Procedural Knowledge

Procedural or imperative knowledge clarifies how to perform a certain task. It lays down the steps to perform. For example, - how to copy an array using javascript in a procedural way.

Example

```
var a=[1, 2, 3, 4, 5];
var b=[];
for(var i=0;i < a.length;i++) {
   b.push(a[i]);
}
console.log(b);
```

Output

```
[1, 2, 3, 4, 5]
```

Declarative Knowledge

Declarative or functional knowledge clarifies what to do to perform a certain task. It lays down the function to perform. For example, - how to copy an array using javascript in a declarative way.

Example

```
var a=[1, 2, 3, 4, 5];
var b=a.map(function(number){
   return number*1
});
console.log(b);
```

Output

```
[1, 2, 3, 4, 5]
```

The prior difference between them is that the **declarative** representation just specifies the knowledge but not the mechanism to implement the knowledge. On the contrary, the **procedural** representation provides the control information along with the knowledge.

BASIS FOR COMPARISON	PROCEDURAL KNOWLEDGE	DECLARATIVE KNOWLEDGE
Basic	Includes the knowledge of how a particular thing can be accomplished.	Includes the basic knowledge about something.
Alternate name	Interpretive knowledge	Descriptive knowledge
Stated by	Direct application to the task and difficult to articulate formally.	Declarative sentences and easily articulated.
Popularity	Less common	Generally used
Ease of sharing the knowledge	Hard to communicate	Can be easily shared, copied, processed and stored.
Taken from	Experience, action, and subjective insight.	Artifact of some type as a principle, procedure, process and concepts.
Nature	Process oriented	Data-oriented
Represented by	Set of rules	Production systems
Feature	Debugging is difficult	Validation is quite simple