Logic Concept and Logic Programming

Logic was considered to be a branch of philosophy. Logic helps in inveshng and classifying the structure of statements and arguments through the study of formal system of inference.

Formally Logic is concerned with the principles of drawing valid inference from a given set of true statements.

Symbolic Logic is divided into two branches

① Propositional Logic

② Predicate Logic.

Proposition refers to a declarative statements that is either true or false (but not both) in a given context One can infer a new proposition from a given set of proposition in the same context using logic.

Propositional Calculus :⇒ Propositional Calculus refers to a language of propositions in which a set of rules are used to combine simple propositions to form compound propositions with the help of certain logical operators. These logical operators are often called connectives. For eg: not, and, or, implies, equivalence.

Well formed formula: A well formed formula is defined as a symbol or a string of symbols generated by the formal grammer of a formal language.

* The smalles unit (or an atom) is considered to be wff.
* If α is wff then $\sim\alpha$ is also wff.
* If α & β are wff then $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$ $(\alpha \rightarrow \beta)$ $(\alpha \leftrightarrow \beta)$ are also wff.

Equivalence Laws :⇒

Equivalence relations or laws are used to reduce or simplify a given WFF or to derive a new formula from the existing formula. Logically equivalent ($\equiv$)

Commutative Law	$A \vee B \equiv B \vee A$ $A \wedge B \equiv B \wedge A$
Associative Law	$A \vee (B \vee C) \equiv (A \vee B) \vee C$ $A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C$
Double Negation	$\sim(\sim B) \equiv B$
Distributive Laws	$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
De Morgan's Laws	$\sim(A \vee B) \equiv \sim A \wedge \sim B$ $\sim(A \wedge B) \equiv \sim A \vee \sim B$
Idempotence	$A \vee A \equiv A$ $A \wedge A \equiv A$

Propositional Logic :⇒

Propositional Logic deals with the validity, satisfiability and Unsatisfiability of a formula and the derivation of a new formula using equivalence Law. Each row of a truth table for a given formula α is called its interpretation under which the value of a formula may be either true or false. A formula α is said to be a tautology if and only if the value of α is true for all its interpretation

* A formula α is said to be valid if and only if it is a tautology
* A formula α is said to be satisfiable if there exists at least one interpretation for which α is true.
* A formula α is said to be unsatisfiable if the value of α is false under all interpretation

Example: Show that the following is a valid argument
"If it is humid then it will rain and since it is humid today it will rain".

Solution: Propositional atoms:

A : It is humid

B : It will rain.

Now formula (α)

$\alpha : [(A \to B) \wedge A] \to B$

A	B	$A \to B = (X)$	$X \wedge A = (Y)$	$Y \to B$
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

Although it is an easy method for evaluating consistency, inconsistency or validity of a formula, the limitation of this method lies in the fact that the size of the truth table grows exponentially.

Some other methods that are concerned with proofs and deductions are as follows.

* Natural deduction System.
* Axiomatic System
* Semantic tableau method.
* Resolution refutation method.

Scanned by CamScanner

5.3.6 $\forall x : [\neg \text{Roman}(x) \vee \neg \text{know}(x, \text{Marcus})] \vee$
$[\text{hate}(x, \text{Caesar}) \vee (\forall y : \forall z : \neg \text{hate}(y, z) \vee \text{think crazy}(x, y))]$

(3) Standardize variables so that each quantifier binds a unique variable. Since variables are just dummy names. For eg.

$\forall x : P(x) \vee \forall x : Q(x)$
would be converted to.

$\forall x : P(x) \vee \forall y : Q(y)$

Pg #6

(4) Move all quantifiers to the left of the formula without changing their relative order.

$\forall x : \forall y : \forall z : [\neg \text{Roman}(x) \vee \neg \text{know}(x, \text{marcus})] \vee$
$[\text{hate}(x, \text{Caesar}) \vee (\neg \text{hate}(y, z) \vee \text{think crazy}(x, y))]$

At this point, the formula is in what is known as prenex normal form. It consists of a prefix of quantifiers followed by a matrix, which is quantifier free.

5. Elliminate existential qualifiers.

6. Drop the prefix.

$[\neg \text{Roman}(x) \vee \neg \text{know}(x, \text{marcus})] \vee$
$[\text{hate}(x, \text{Caesar}) \vee (\neg \text{hate}(y, z) \vee \text{think crazy}(x, y))]$.

7. Convert the matrix into a conjunction of disjuncts. [I.e $(a \wedge b) \vee c = (a \vee c) \wedge (b \wedge c)$] and simply remove the parentheses.

$\neg \text{Roman}(x) \vee \neg \text{know}(x, \text{marcus}) \vee \text{hate}(x, \text{Caesar})$
$\vee \text{hate}(y, z) \vee \text{think crazy}(x, y)$.

8. Create a separate clause corresponding to each conjunct.

9. Rename the variable so that no two clauses make reference to the same variable.

In the end we will have a set of clauses, each of which is a disjunction of literals.

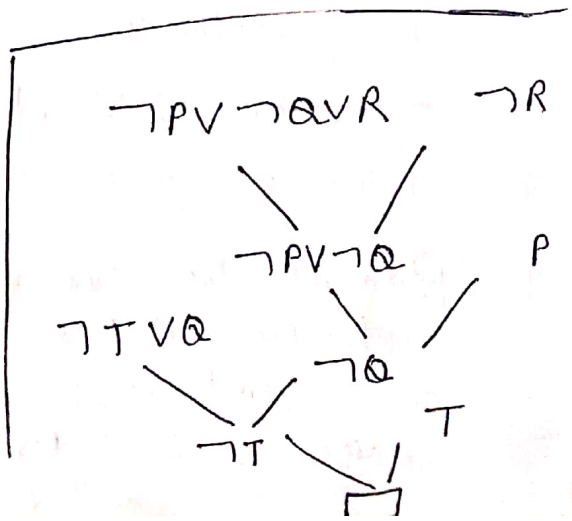
Resolution in Propositional Logic :→

Algorithms : 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 step1.
3. Repeat until either a contradiction is found or no progress can be made:
   (a) Select two clause. 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 clause 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.

For eg.

Given Axioms	Converted to clause form.
P.	P.
$(P \wedge Q) \rightarrow R$	$\neg P \vee \neg Q \vee R$
$(S \vee T) \rightarrow Q$	$\neg S \vee Q$
	$\neg T \vee Q$
T	T



Computable Functions AND PREDICATES :→

(1) Marcus was a man.
man(Marcus).

(2) Marcus was a Pompeian.
Pompeian(Marcus).

(3) Marcus was born in 40 A.D
born(Marcus, 40).

4. All men are mortal.
$\forall x : man(x) \rightarrow mortal(x)$.

5. All Pompeians died when the volcano erupted in 79 A.D.
$erupted(Volcano, 79) \wedge \forall x : [Pompeian(x) \rightarrow died(x, 79)]$

6. No mortal lives longer than 150 yrs.
$\forall x : \forall t_1 : \forall t_2 : mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \rightarrow dead(x, t_2)$

7. gt is now 1991
now = 1991.

8. Alive means not dead.
$\forall x : \forall t : [alive(x,t) \rightarrow \neg dead(x,t)] \wedge$
$[\neg dead(x,t) \rightarrow alive(x,t)]$

9. If someone dies, then he is dead at all later times.
$\forall x : \forall t_1 : \forall t_2 : died(x, t_1) \wedge gt(t_2, t_1) \rightarrow dead(x, t_2)$

Says that one is dead in all years after the one in which one died.

Resolution : Resolution produces proofs by refutation. In other words, to prove a statement, resolution attempts to show that the negation of the statement produces a contradiction with the known statement.

① **conversion to clause form.**

Suppose we know that all Romans who know marcus either hate Caesar or think that anyone who hates anyone is crazy. In wff.

$$\forall x : [\text{Roman}(x) \wedge \text{know}(x, \text{marcus})] \rightarrow [\text{hate}(x, \text{Caesar}) \vee (\forall y : \exists z : \text{hate}(y, z) \rightarrow \text{thinkcrazy}(x, y))]$$

Algorithm Convert to clause form.

① Eliminate $\rightarrow$ using the fact that $a \rightarrow b$ is equivalent to $\neg a \vee b$.

② ~~Reduce the scope of~~

$$\forall x : \neg[\text{Roman}(x) \wedge \text{know}(x, \text{marcus})] \vee [\text{hate}(x, \text{Caesar}) \vee (\forall y : \neg(\exists z : \text{hate}(y, z)) \vee \text{think crazy}(x, y))]$$

② Reduce the scope of each $\neg$ to a single term, using the fact that $\neg(\neg p) = p$.

demorgan's law

$\neg(a \wedge b) = \neg a \vee \neg b$.

$\neg(a \vee b) = \neg a \wedge \neg b$

Predicate logic: is a logical extension of propositional Logic, which deals with the validity, satisfiability and unsatisfiability of a formula along with the inference rule for derivation of a new formula. Predicate Calculus is the study of predicate systems; when inference rules are added to predicate calculus, it becomes predicate logic

Predicate Calculus has three more logical notions in addition to propositional calculus. These are:

* **Term:** A term is defined as either a variable or constant or n-place function. A function is defined as a mapping that maps n term to a single term.
* **Predicate:** A predicate is defined as a relation that maps n-terms to a truth value
* **Quantifiers:** Quantifiers are used with variable There are two types of quantifiers, namely Universal (∀ forall), Existential ∃ there exist.

Pg# 8

WFF in predicate logic :→

* Atomic formula $P(t_1 \ldots t_n)$ (also called an ato is well formed formula. Where P is a predicate symbol and $t_1 \ldots t_n$ are the terms.
* If α and β are well formed formula then $\neg(\alpha)$, $(\alpha \vee \beta)$, $(\alpha \vee \beta)$ $(\alpha \rightarrow b)$ $(\alpha \leftrightarrow b)$ are well formed formula.

* If α is a well formed formula and X is a free variable in α then $(\forall x)\alpha$ and $(\exists x)\alpha$ are both well formed formula.
* wff may be generated by applying the rules described above a finite number of times.

First Order Predicate Calculus :⇒ If the quantification in predicate formula is only on simple variables and not on predicates or functions then it is called first order predicate ~~and functions~~. Calculus. On the other hand if the quantification is over first order predicate and functions then it becomes second order predicate calculus.

The first order predicate calculus is a formal language in which a wide variety of statements are expressed. The formulae in predicate calculus are formed using rules similar to those used in propositional calculus. When inference rules are added to first order predicate calculus, it becomes first-order predicate logic (FOL).

Each formula α is evaluated to be true or false under a given interpretation I over a given Domain D.

* $(\forall x)\, P(x)$ = true if and only if $P(x)$ = true, $\forall x \in D$ otherwise it is false.
* $(\exists x)\, P(x)$ = true if and only if $\exists c \in D$ such that $P(c)$ = true otherwise it is false.

Page #9

(6) Evaluate the truth value of an FOL formula $\alpha : (\forall x)(\exists y)\, p(x,y)$ under the following Interpretation I:

* $D = \{1, 2\}$.
* $p(1,1) = F$, $p(1,2) = T$, $p(2,1) = T$, $p(2,2) = F$

Let us denote true by T and false by F. For $x = 1$ then $\exists\, 2 \in D$ such that $p(1,2) = T$ and for $x = 2$ then $\exists\, 1 \in D$ such that $p(2,1) = T$. Hence α is true under interpretation I.

(11) Evaluate $\alpha : (\forall x)[\, p(x) \rightarrow q(f(x), c)]$

* $D = \{1, 2\}$
* $c = 1$ (c is constant from domain D)
* $f(1) = 2$ $f(2) = 1$
* $p(1) = F$, $p(2) = T$
* $q(1,1) = T$, $q(1,2) = T$, $q(2,1) = F$, $q(2,2) = T$.

For $x = 1$

$p(1) \rightarrow q(f(1), 1)$

$\trianglelefteq\ p(1) \rightarrow q(2,1)$

$\trianglelefteq\ F \rightarrow q(2,1)$

$\trianglelefteq\ T$

For $x = 2$

$p(2) \rightarrow q(f(2), 1) \trianglelefteq p(2) \rightarrow q(1,1) \trianglelefteq T \rightarrow q(1,1)$

$T \rightarrow T \trianglelefteq T$.

α is true for all values of $x \in D$ under the interpretation I

Predicate Logic

Predicate logic as a way of Representing knowledge

(1) Marcus was a man.

man (Marcus).

(2) Marcus was a pompein

Pompeian (Marcus)

(3) All Pompeian were Romans.

$\forall x$: Pompeian(x) → Roman(x).

(4) Caesar was a ruler

ruler (Caesar).

(5) All Romans were either loyal to Caesar or hated him.

$\forall x$: Roman(x) → loyalto (x, Caesar) ∨ hate(x, Caesar)

(6) Every one is loyal to Someone

$\forall x \rightarrow y$: loyalto (x, y)

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

$\forall x : \forall y$: person(x) ∧ ruler(y) ∧ tryassassinate (x, y) → ¬ loyalto (x, y).

(8) Marcus try to assassinate Caesar.

trytoassassinate (Marcus, Caesar).

(9) All men are people

$\forall$: man(x) → people(x).

Three ways of Representing Class membership.

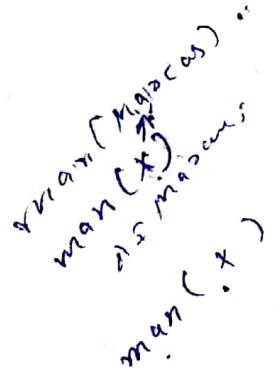
(1) man (Marcus)
(2) Pompeian (Marcus)
(3) $\forall x$: Pompeian(X) → Roman(X)
(4) ruler (Caesar)
(5) $\forall x$: Roman (X) → loyalto (X, Caesar) V hate (X, Caesar)

1. Instance (Marcus, man)
2. Instance (Marcus, pompeian)
3. $\forall x$: Instance (X, Pompeian) → instance (X, Roman)
4. Instance (Caesar, ruler)
5. $\forall x$: instance (X, Roman) → loyal to (X, Caesar) V hate (X, Caesar)

1. Instance (Marcus, man)
2. Instance (Marcus, pompeian)
3. isa (Pompeian, Roman)
4. Instance (Caesar, Ruler)
5. $\forall x$: Instance (X, Roman) → Loyalto (X, Caesar) V hate (X, Caesar)
6. $\forall x$: $\forall y$: $\forall z$: Instance (X, Y) ∧ isa (Y, Z) → Instance (X, Z)

gorithm : 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 L1 and L2 have different number of arguments, then return {FAIL}.
4. Set SUBST to NIL.
5. For i ← 1 to no. of arguments in L1:
   (a) Call unify with the ith argument of L1 and 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.



Resolution in Predicate Logic :→

Algorithm : Resolution

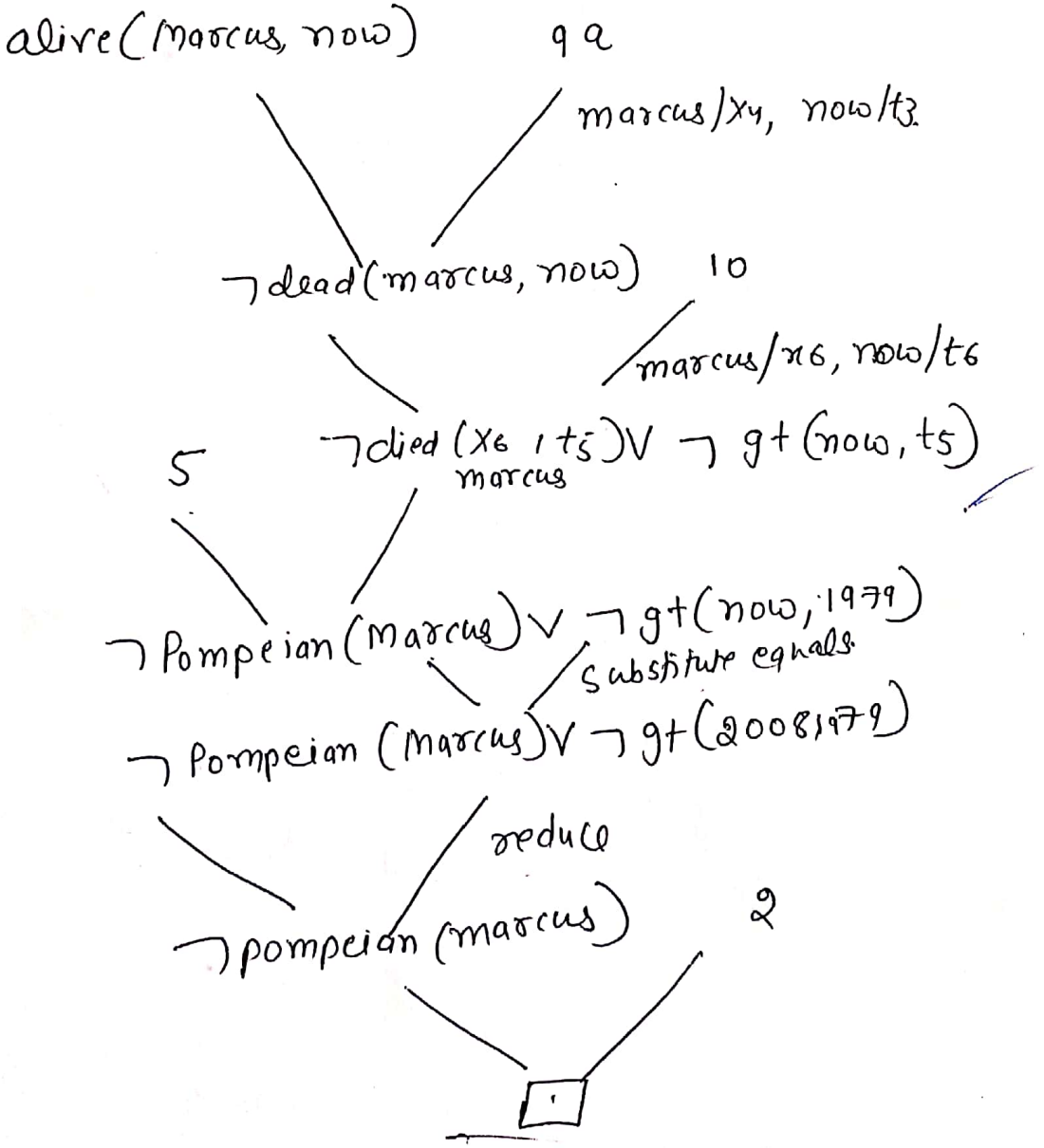
1. Convert all the statements of F to clause form.
2. Negate P and convert the result to clause form. Add it to the set of clauses obtained in 1.
3. Repeat until either a contradiction is found, no progress can be made, or a predetermined amount of effort has been expended.
   (a) Select two clauses, call these the parent clause.
   (b) Resolve them together. the resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitution performed and with the following exception: If there is one pair of literals T_1 and $\neg T_2$ such that one of the parent clauses contains T_2 and the other contains T_1 and if T_1 & T_2 are unifiable, then neither T_1 nor T_2 should appear in the resolvent. We call T_1 & T_2 Complementary literals. Use the substitution produced by the unification to create the resolvent. If there are more than one pair of complimentary literals, only one pair should 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.

Prove : ¬ alive (marcus, now).

(10)

Axiom in clause form :

1. man (Marcus).
2. Pompeian (Marcus).
3. born (Marcus, 1940).
4. ¬ man (x_1) ∨ mortal (x_1)
5. ¬ Pompeian (x_2) ∨ died (x_2, 1979).
6. erupted (Volcano, 1979).
7. ¬ mortal (x_3) ∨ ¬ born (x_3, t_1) ∨ ¬ gt (t_2 - t_1, 150) ∨ dead (x_3, t
8. now = 2008.
9(a) ¬ alive (x_4, t_3) ∨ ¬ dead (x_4, t_3)
9b. dead (x_5, t_4) ∨
10. ¬ died (x_6, t_5) ∨ ¬ gt (t_6, t_5) ∨ dead (x_6, t_6)



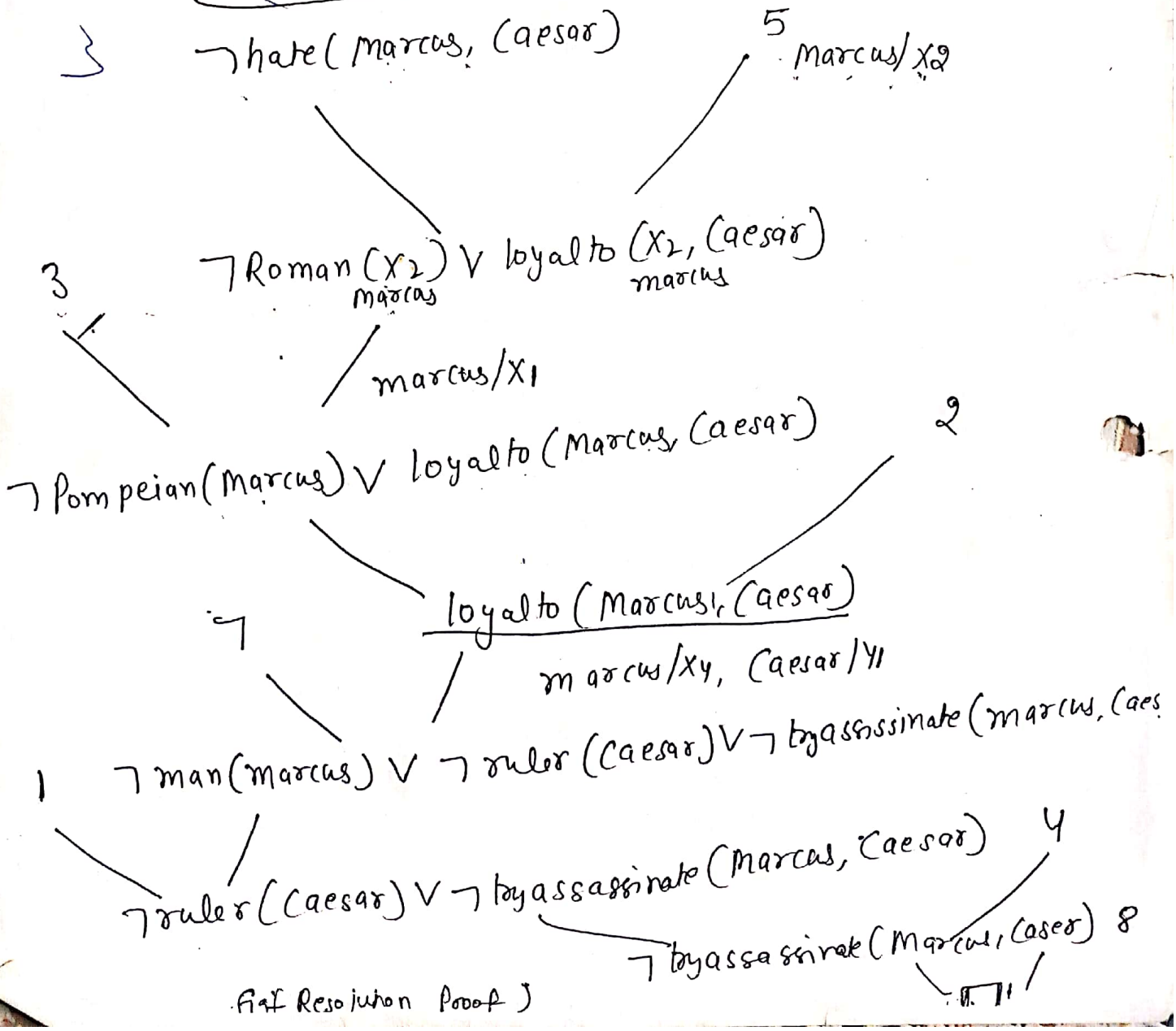
Problems :⇒

Axioms in CLF (Clause form)

1. man(marcus).
2. Pompeian(Marcus).
3. ¬ Pompeian(X_1) ∨ Roman(X_1). Repeat
4. ruler(Caesar).
5. ¬ Roman(X_2) ∨ loyalto(X_2, Caesar) ∨ hate(X_2, Caesar).
6. loyalto(X_3, f1(X_3)).
7. ¬ man(X_4) ∨ ¬ ruler(y_1) ∨ ¬ trytoassissinate(X_4, y_1) ∨ ¬loyalto(X_4, y_1)
8. trytoassissinate(marcus, Caesar).

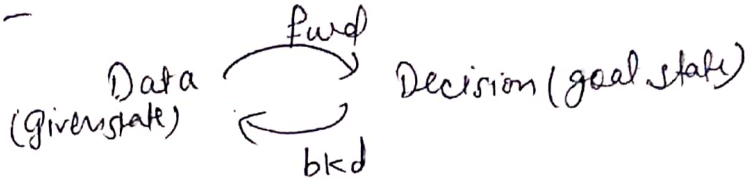
Prove: hate(marcus, Caesar).

P ¬P = □



(First Resolution Proof)

Forward chaining:-



→ Start with the given state & connect it with the next state closest to goal state. Do so till the goal state reached

→ Fwd chaining starts with the available data & use inference rules to extract more data until a goal is reached.

→ generate next level of tree by finding all the rules whose Left side matches the root node & using right side → create new configuration. Adding the new Information to its data.

1st e.g. Rule base contains following rule set

Rule 1: → if A and C then F

2 :- A ∧ E → G

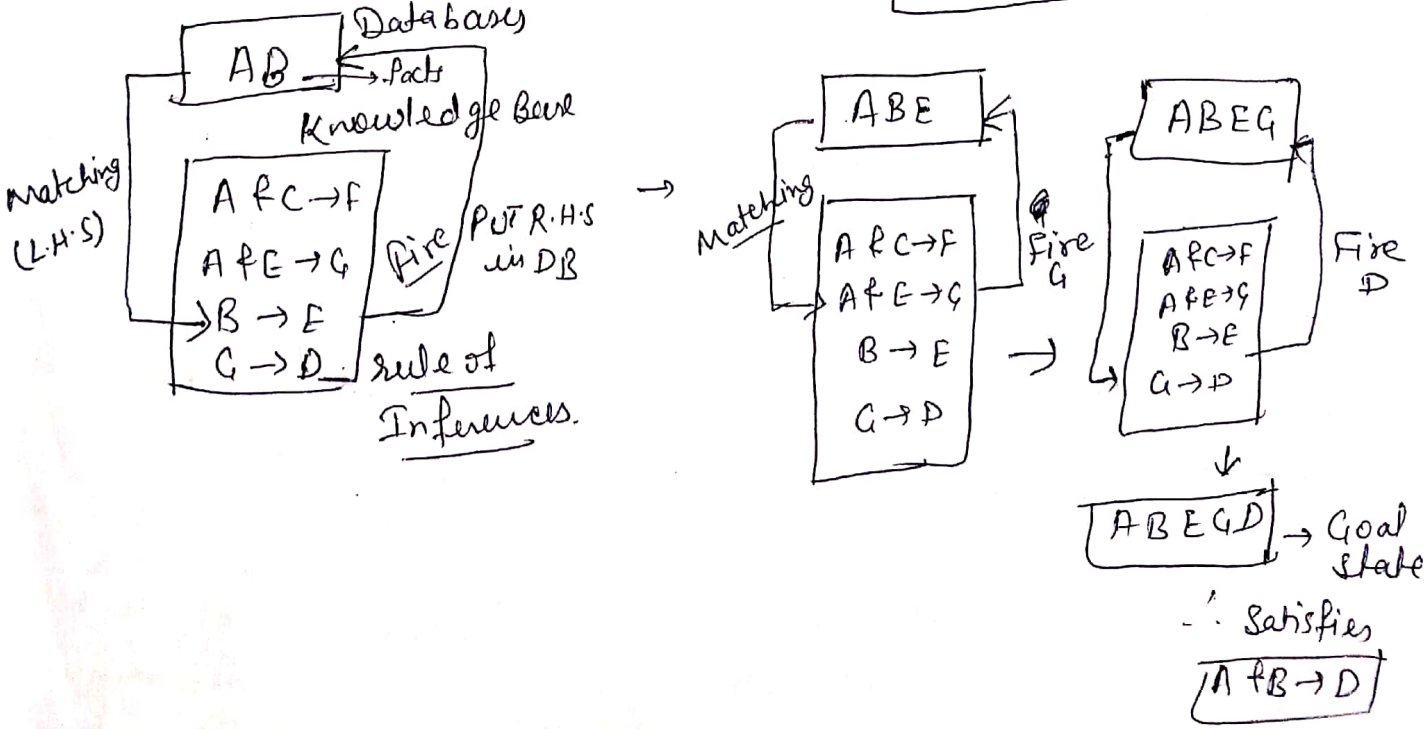
3 → if B Then E

4. → if G then D

Problem :- Prove if A & B true then D is true

What new is given should be added to Database

A & B → D → Goal or Ans.



II nd e.g: Goal is to conclude the color of pet named Teddy.
Given → it barks & eats meat
Rule based contain fours rules
① if x barks & eats meat → Then x is a dog
② if x chirps and Sings → Then x is canary
③ if x is a dog → Then x is Brown
④ if x is a canary → Then x is yellow.
First rule selected becoz it L.H.S matches with the given data.
∴ Consequent (R.H.S) added to the data. i.e. x is a dog.
now III rd rule matches (L.H.S) with data that x is a dog.
R.H.S given as x is Brown

III rd e.g: → washing hands → seq. of steps. with guidance till the goal is reached i.e. clean hands.

Backward chaining → Start with goal state & then connect it with the next state closest to initial state, cts till the initial state
→ Generate the next level of tree by rule whose Right side matches the root node. if the L.H.S. of rules is true than it is added to list of goals.
e.g: goal → ~~the~~ Conclude the color of pet Teddy.
Given → He barks and eat meat
As above ← Rules
Now the Rules searched & start with the goal state—
goal state matches with ③ & ④ rule.
Both the L.H.S. added to goal list.
→ i.e: if Teddy is a dog, if Teddy is a canary
→ Rule base is again searched with R.H.S.
→ R.H.S. matches with L.H.S. Rule ① & ② seleted & added to goal list.
→ Rule ① matches with the given data ~~or~~ initial state
→ ∴ concluded that Teddy is a dog not a canary
→ goal achied that Teddy is brown ~~&~~ he is a dog ∵ he barks & eat not Sing.

IV e.g: washing hands in bkward Chaining → start with seq. Now leave the last to wipe with tissue. Next step leave the previous two steps i.e. wipe hands & use of soap this way leave the steps until ~~goal~~ initial state ∴ goal is reached. reinforce ment learning.

Propositional Logic :→ A proposition or statement is a sentence which is either true or false. If a proposition is true, then we say its truth value is true, and if a proposition is false, we say its truth value is false.

Propositional Calculus :→ It is a branch of logic, It is also called propositional logic sentential calculus, sentential logic or zero-order logic. It deals with propositions and argument flow.

It is the branch of logic, that studies ways of joining and/or modifying entire propositions, statements or sentences to form more complicated propositions, statements.

Propositional logic :→ It largely involves studying logical connectives such as the words 'and' and 'or' and the rules determining the truth values of the propositions they are used to join, as well as what these rules mean for the validity of arguments and such logical relationship between statements as being consistent or inconsistent with one another.

dicate logic :⇒ To cope with
ficiencies of proponhonal logic, we introduce
two new following feature: predicate and
quantifiers. A predicate is a verb template
that describe a property of objects, or relahonship
among objects represented by the variables.

For eg :⇒ The Car Tom is driving is blue
→ The sky is blue.
→ The cover of the book is blue.

The 'is blue' is a predicate and describes
the property of being blue.
Predicates are often give a name for eg
'isBlue', Blue, B can be used to represent
a predicate.

→ If we adopt B as the name of predicate
for is_blue, sentences that asserts an
object is blue can be represented as.

B(X) – where X represents an
arbitrary object.

Quantifiers :⇒ (1) Universal Quantifiers
corresponds to conjunchon 'and' in that
(∀x) P(X) means that P holds for
all values of x in the domain associated
with that variable.
For eg : (∀x) dolphin(x) → mamal(x).

xistential Quantification – OR.

(∃x){P(x) means that P holds for some value of x in the domain associated with that variable.

For eg (∃x) mammal(x) ∧ lays_eggs(x)

3. Universal quantifiers are usually used with implies to form if-Then rules

For eg (∀x) student(x) → smart(x) means all students are smart.

4. Existential quantifiers are usually used with (and) to specify a list of properties or facts about an individual.

eg :- (∃x) student(x) ∧ smart(x) – implies there is a student who is smart

5. Switching the order of Universal quantifiers does not change the meaning

(∀x) (∀y) P(x,y) = (∀y) (∀x) P(x,y)

Similarly we can change order of existential quantifiers.

Terminology :→ A term is a constant Symbol. a variable symbol, or a function.

For eg father-of().

f(x, .. xn) are terms, where each xi is a term.

atom which has true or false value.
is either an n-place predicate of n terms.
If P and Q are Atom then
$\neg P$, $P \vee Q$, $P \wedge Q$, $P \rightarrow Q$, $P \leftrightarrow Q$ are atoms

3. A sentence is an atom, or if P is a sentence and x is a variable then
$(\forall x) P$ and $(\exists x) P$ are sentences.

4. WFF (A well formed formula) is a sentence containing no "free" variables - i.e all the variables are bound by universal or existential quantifiers.
For eg $(\forall x) P(x, y)$ has x bound as universally quantified variable but y is free

→ Read formula from left to right

$\forall x$ → 'for every object x in the universe the following holds'

$\exists x$ → there exist an object x in the universe which satisfy the following. or for some object x in the universe the following holds