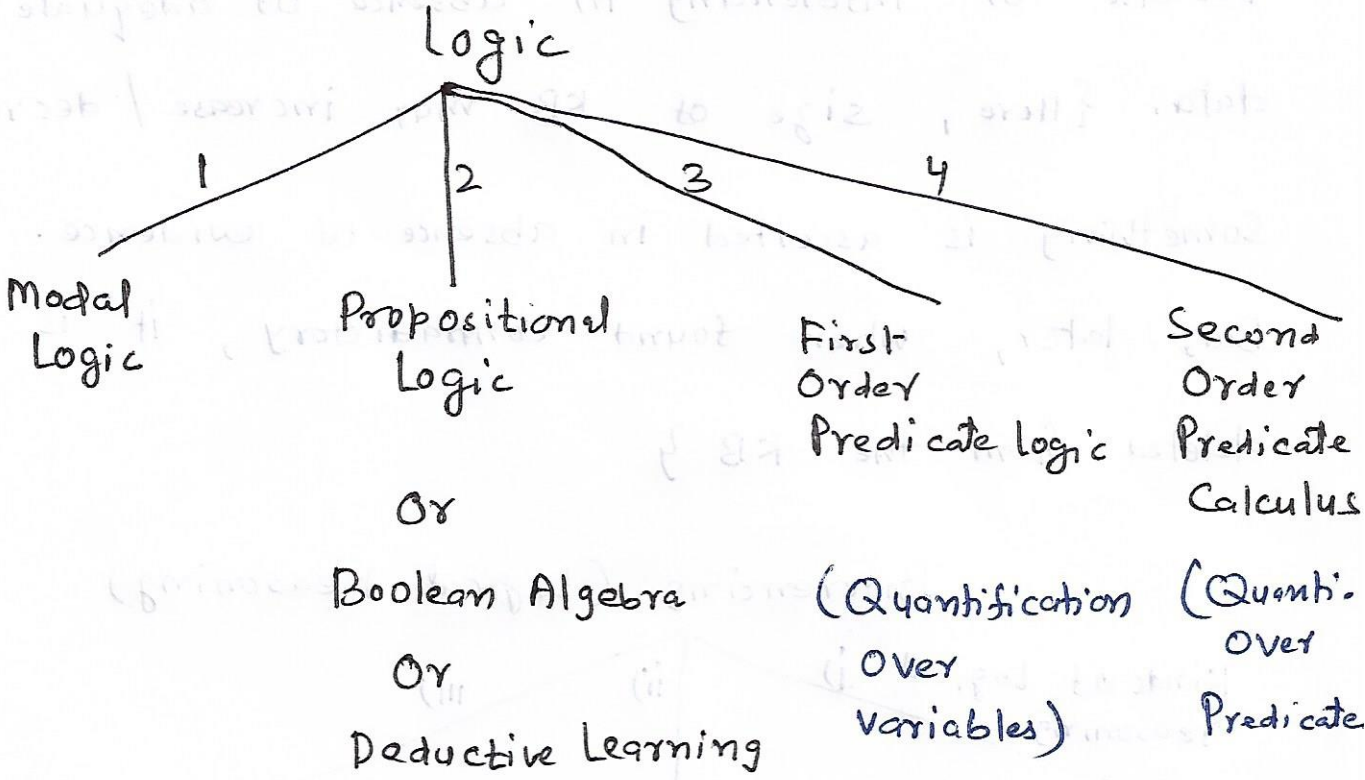
Part II Knowledge Representation



Modal Logic includes :

i) Assertive sentences (facts)

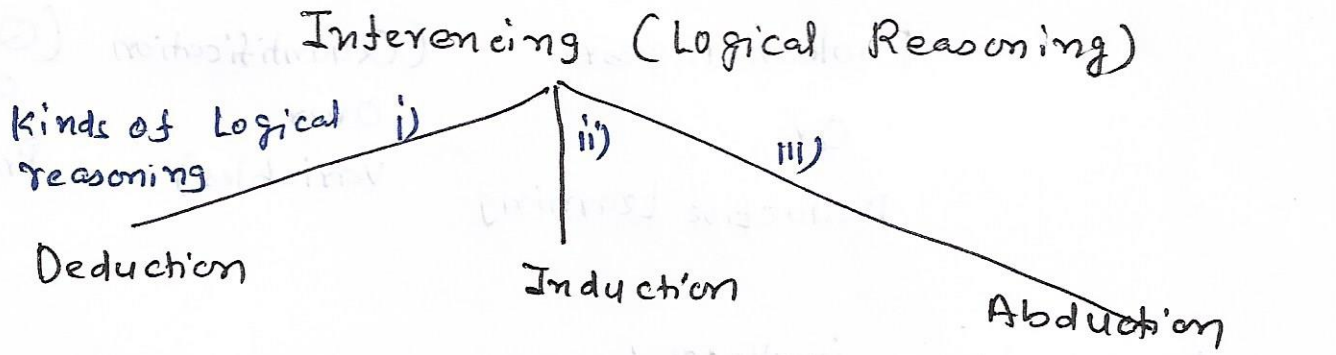
ii) Modal sentences

Possibility - e.g. If I were president...

Belief Sentences - I suppose ...
I believe ...
I expect ...

2, 3, 4 are known as "Monotonic Logic systems". This is because the size of KB goes on increasing through interence. It never decreases.

I is a Non-Monotonic Logic system. An asserted fact may be deleted later. This is because of inferencing in absence of adequate data. {Here, size of KB may increase / decrease Something is asserted in absence of evidence. But, later, when found contradictory, it is deleted from the KB }



eg. All men are mortal

John is a man

⇒ John is mortal

Applications

Mathematicians use this reasoning style

(Generalizing from finite knowledge) i.e. learning a rule from several examples.

e.g. The grass has been wet every time it has rained".

Rule: When it rains, the grass gets wet

Scientists use this style of reasoning

$A \rightarrow B$,

Suppose, we do know whether $B \rightarrow A$.

This is assumed to be true in absence of any knowledge.

This is Abductio

Diagnosticians and Detectives use this style of reasoning

Propositional Logic

Sentences	Proposition
It is raining	RAINING
It is sunny	SUNNY
It is windy	WINDY
It is raining, then it is not sunny	RAINING → ¬SUNNY

Predicate Calculus (First Order Logic)

Motivation

i) To represent relationship between objects.

eg. Sky is blue : P

Screen is blue : Q

blue (sky).

blue (screen).

ii) Handling "For All" and "There exists" kind of sentences

Operators in FOPL

$\forall$ – For All [Universal Quantifier]

$\exists$ – There exists [Existential Quantifier]

$\rightarrow$ Implication

$\neg$ $\wedge$ $\vee$: NOT AND OR

Facts	Representation in Logic
1. Marcus was a man	man (Marcus)
2. Marcus was a Pompeian	pompeian (Marcus)
3. All Pompeians were Romans	$\forall x$: pompeian (x) $\rightarrow$ roman (x)
4. Caesar was a ruler	ruler (Caesar)
5. All Romans were either loyal to Caesar or hated him	$\forall x$: Roman (x) $\rightarrow$ loyalto (x, Caesar) $\vee$ hate (x, Caesar)
6. Everyone is loyal to someone	i) $\forall x: \exists y:$ loyalto (x, y)
	ii) $\exists y: \forall x:$ loyalto (x, y)
	iii) $\forall x: \exists y:$ loyalto (y, x)

Person Who is loyal to someone

Person to whom x is loyal

Everyone has a loya perso

There is someone to whom

possibilities

i) $\forall x : \exists y : \text{loyalto}(x, y)$

ii) $\exists y : \forall x : \text{loyalto}(x, y)$

iii) $\forall x : \exists y : \text{loyalto}(y, x)$

i) matches our interpretation

We should be careful about scope of the quantifiers and ambiguity

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

Ambiguity

i) Only rulers people try to assassinate are those to whom they are not loyal

ii) Only thing people try to do is to assassinate rulers to whom they are not loyal

With interpretation i), the representation is:

$\forall x : \forall y : \text{person}(x) \wedge \text{ruler}(y) \wedge \text{tryassassinate}(x, y) \rightarrow \neg \text{loyalto}(x, y)$

8. Marcus tried to assassinate Caesar

tryassascinate (Marcus, Caesar)

Question : Was Marcus loyal to Caesar ?

¬loyalto (Marcus, Caesar)

↑ 7, substitution

person (Marcus)

ruler (Caesar)

tryassassinate (Marcus, Caesar)

↑ 8

person (Marcus)

ruler (Caesar)

↑ 4

person (Marcus)

↑ 9, substitution

man (Marcus)

↑ 1

□

How to know that a ~~person~~ man is a person. Needs to be explicitly specified

9. $\forall x$: man (x) → person (x)

How should a program decide whether it should try to prove

loyalto (Marcus, Caesar) OR

¬ loyalto (Marcus, Caesar)

Possibilities

1. Use forward chaining, i.e. using available knowledge, see what are the things that can be infered.

   Problem : branching factor increases with the amount of knowledge

2. Use Heuristic knowledge to decide which answer is more likely and then try to prove that.

   If it can not be proved in some reasonable amount of time, then prove other thing.

3. Prove both things simultaneously and stop when one of the things is proved

Consider the following facts

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 errupted in 79 A.D.

   errupted (volcano, 79) $\wedge$ $\forall x$: pompeian(x) $\rightarrow$ died (x, 79)

6. No mortal lives longer than 150 years

   $\forall x$: $\forall t_1$: $\forall t_2$: mortal(x) $\wedge$ born(x, t_1) $\wedge$ gt($t_2 - t_1$, 150) $\rightarrow$ dead (x, t_2)

7. It is now 2020

   now = 2020

[Equal Quantities can be substituted for each other]

Is Marcus alive now?

We can prove that Marcus is dead by:

i) Killed by volcano

ii) Age > 150 years

Additional knowledge (Relationship bet^n/ alive and dead is to be established)

8. $\forall x : \forall t : \text{dead}(x, t) \rightarrow \neg \text{alive}(x, t)$

9. If someone dies, then he is dead at all later times.

$$\forall x : \forall t_1 : \forall t_2 : \text{died}(x, t_1) \wedge gt(t_2, t_1) \rightarrow \text{dead}(x, t_2)$$

Soln I

¬alive(Marcus, now)

↑ 8, substitution

dead(Marcus, now)

↑ 9, substitution

died(Marcus, t1)
gt(now, t1)

↑ 5, substitution

Pompeian(Marcus)
gt(now, 79)

↑ 2

gt(now, 79)

↑ 7

gt(2020, 79)

↑ Compute gt

Solution II

¬alive (Marcus, now)

↑ 8, substitution

dead (Marcus, now)

↑ 6, substitution

mortal (Marcus)

born (Marcus, t_1)

gt (now − t_1 , 150)

↑ 4, substitution

man (Marcus)

born (Marcus, t_1)

gt (now − t_1 , 150)

↑ 1

born (Marcus, t_1)

gt (now − t_1, 150)

↑ 3, substitution

gt (now − 40 , 150)

↑ 7

gt (2020 − 40, 150)

↑ Compute Minus

gt (1980 , 150)

↑ Compute gt

Resolution

- Proof by refutation (contradiction).
- It is used to intere new clause from old ones.

It works as:

"A clause may be proven to be true, in the context of a set of clauses known to be true, by adding the logical NOT of this clause to the set and seeking a contradiction".

→ Resolution provides a way of finding contradictions by trying a minimum no. of substitutions.

→ If contradiction exists, then eventually it will be found.

If resolution process fails to find a contradiction, the negative of what we seek to prove is logically consistent with the database, thus the clause can not be true.

Algo. Propositional Resolution

1. Convert all the Propositions of F, a set of axioms, to clause form.

2. Negate P, Proposition to be proved by Resolution, 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 parent clauses.

   (b) Resolve them together. The resulting clause, called the resolvent, will be the disjunction of all of the literals of all 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 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.

Propositional Resolution Example

To Prove R

1	P V Q
2	P → R
3	Q → R

Clauses

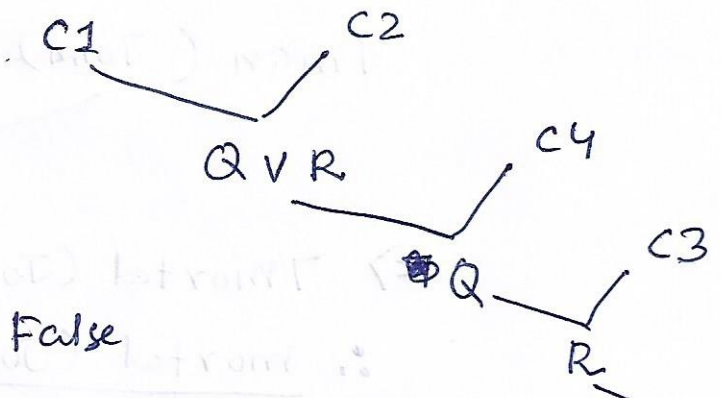
C1 P V Q

C2 ¬P V R

C3 ¬Q V R

To prove R.

Negated clause is ¬R. ← C4



∴ ¬R is False

⟹ R

Use of Resolution for interencing (in Predicate Logic)

Example

Consider the following sentences

(a) All men are mortal

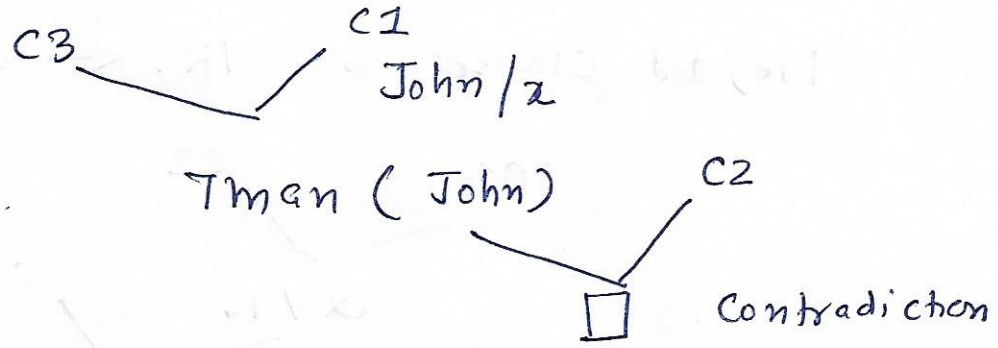
(b) John is a man

Convert the sentences into wff	Clauses for wff
1. $\forall x$: man $(x) \rightarrow$ mortal (x)	C1 : $\neg$man (x) V mortal (x)
2. man (John).	C2 : man (John)

Question : Is John mortal ?

goal : mortal (John)

Negated goal : $\neg$mortal (John) C3



$\Rightarrow$ $\neg$mortal (John) is false

$\therefore$ mortal (John)

Algo. Convert to Clause Form

1. Eliminate "$\rightarrow$".

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

Also,

$$\forall x : \neg[\text{Roman}(x) \wedge \text{know}(x, \text{Marcus})] \vee [\text{hate}(x, \text{caesar}) \vee (\forall y : \neg(\exists z : \text{hate}(y, z) \vee \text{thinkcrazy}(x, y))]$$

2. Reduce the scope of each $\neg$ to a single term using demorgan's laws and relation between quantifiers, such as

i) $\neg \forall x : P(x) \equiv \exists x : \neg P(x)$

ii) $\neg \exists x : P(x) \equiv \forall x : \neg P(x)$

$$\forall x : [\neg \text{Roman}(x) \vee \neg \text{know}(x, \text{Marcus})] \vee [\text{hate}(x, \text{caesar}) \vee (\forall y : \forall z : \neg \text{hate}(y, z) \vee \text{thinkcrazy}(x, y))]$$

Conversion to Clause Form

Consider the following sentences

All Romans who know Marcus either hate Caesar or think that anyone who hates anyone is crazy.

$$\forall x : [\text{Roman}(x) \wedge \text{know}(x, \text{Marcus})] \rightarrow$$

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

A formula is in Conjunctive Normal Form if:

i) It is flatter (less embedding of components)

ii) Quantifiers are separated from the rest of the formula so that they need not be considered.

Resolution requires clauses to be in Conjunctive Normal form with no instances of the connector $\wedge$.

3. Standardize variables so that each quantifier binds a unique variable. e.g.

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

would be converted as: $\forall x : P(x) \vee \forall y : Q(y)$

This step is in preparation for the next-

4. Move all quantifiers to the left of the formula without changing their relative order. This is possible since there is no conflict among variable names.

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

This is called Prenex Normal form, which consists of of a prefix of Quantifiers and a matrix which is quantifier free.

(5) Eliminate existential quantifiers

We can eliminate the quantifier by substituting for the variable a reference to a function that provides the desired value. This is called Skolemizati (Skolemization). We assume that there is a function which gives us this value.

i) $\exists y$: president (y)

can be transformed as president (s1).

Where s1 is a function with no arguments that somehow produces a value that satisfies president

ii) If existential quantifier within the scope of universal quantifier, then it can be transformed as follows:

$\forall x$: $\exists y$: fatherof (y, x)

It is transformed as:

$\forall x$: fatherof (s2(x), x)

[Here, value of y depends upon x]

6) Drop the prefix

All the variables are assumed to be universally quantified (Existential Quantifiers have been removed already)

$$[\neg Roman(x) \vee \neg know(x, marcus)] \vee$$

$$[hate(x, caesar) \vee (\neg hate(y, z) \vee thinkcrazy(x, y))]$$

7) Convert the matrix into a conjunction of disjuncts.

Here, in our example, no And's are present. So, we use associative property of "V" and remove the parenthesis.

$$\neg Roman(x) \vee \neg know(x, Marcus) \vee$$
$$hate(x, caesar) \vee \neg hate(y, z) \vee$$
$$thinkcrazy(x, y)$$

8) Create a separate clause corresponding to each conjunct.

9) Standardize apart the variables in the set of clauses generated in step 8, i.e. rename the variables so that no two clauses make reference to the same variable.

Rules for Unification

1. Two atoms that are identical unify.
2. Two identical lists or expressions converted to list form, unify.
3. A constant and unbounded variable unify, with the variable becoming bound to the constant.
4. An unbound variable unifies with a bound variable, with the unbound variable becoming bound to the bound variable.
5. A bound variable unifies with a constant if the binding on the bound variable does not conflict with the constant.
6. Two unbound variables unify. If either variable becomes bound in subsequent unification steps, the other also gets bound to the same atom (variable or constant)
7. Two bound variables unify if they are both bound (perhaps through intermediate bindings) to the same atom (variable or constant)

For algorithm, refer to the text book.

Examples

(I) tryassassinate (Marcus, Caesar)

hate (Marcus)

Can't unify as predicate names are different.

(II) Unify : P (x, x)

P (y, z)

a. P matches P.

b. First argument in both the predicates are variables.

- Make substitution y/x [or x/y]
- This substitution must be made throughout the predicates giving :

P (y, y)

P (y, z)

- New arguments y and z (variables) will unify with substitution z/y.

The unification procedure has succeeded with composite substitution written as :

(z/y) (y/x)

Apply this first

Apply this substitution to new list

(iii) Unify : hate (x, y)
hate (Marcus, z)

Both ~~with~~ will unify with following substitutions:

(Marcus/x, z/y)
(Marcus/x, y/z)
} MGU – Most General Unifier

(Marcus/x, Caesar/y, Caesar/z)
(Marcus/x, Polonius/y, Polonius/z)
} Not MGU

(iv) f(x, x)
f(g(x), g(x))

Can't unify.

Problem is g(x)/x leads to f(g(g(g ...

Rule : An expression involving variable is not unified with variable.

Consider the following sentences

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.

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

6. Everyone is loyal to someone.

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

8. Marcus tried to assassinate Caesar.

Que Did Marcus hate Caesar? Answer using Resolution

1. man (Marcus)

2. pompeian (Marcus)

3. $\forall x$: pompeian $(x) \rightarrow$ roman (x)

4. ruler (Caesar)

5. $\forall x$: roman $(x) \rightarrow$ loyalto $(x,$ Caesar$)$ V hate $(x,$ Caesar$)$

6. $\forall x: \exists y:$ loyalto (x, y)

7. $\forall x: \forall y:$ man $(x) \wedge$ ruler $(y) \wedge$ try assassinate (x, y) $\rightarrow \neg$ loyalto (x, y)

8. tryassassinate (Marcus, Caesar)

Clause Form

1. man (Marcus)

2. pompeian (Marcus)

3. ¬pompeian (x) ∨ roman (x)

4. ruler (Caesar)

5. ¬roman (x1) ∨ loyalto (x1, Caesar) ∨ hate (x1, Caesa

6. loyalto (x_2, $f(x_2)$)

7. ¬man (x_3) ∨ ¬ruler (y) ∨ ¬tryassassinate (x_3, y) ∨ ¬loyalto (x_3, y)

8. tryassassinate (Marcus, Caesar)

goal: hate (Marcus, Caesar)

Negated Clause : ¬hate (Marcus, Caesar)

¬hate (Marcus, Caesar)

5

Marcus / x1

¬roman (Marcus) ∨ loyalto (Marcus, Caesar)

3

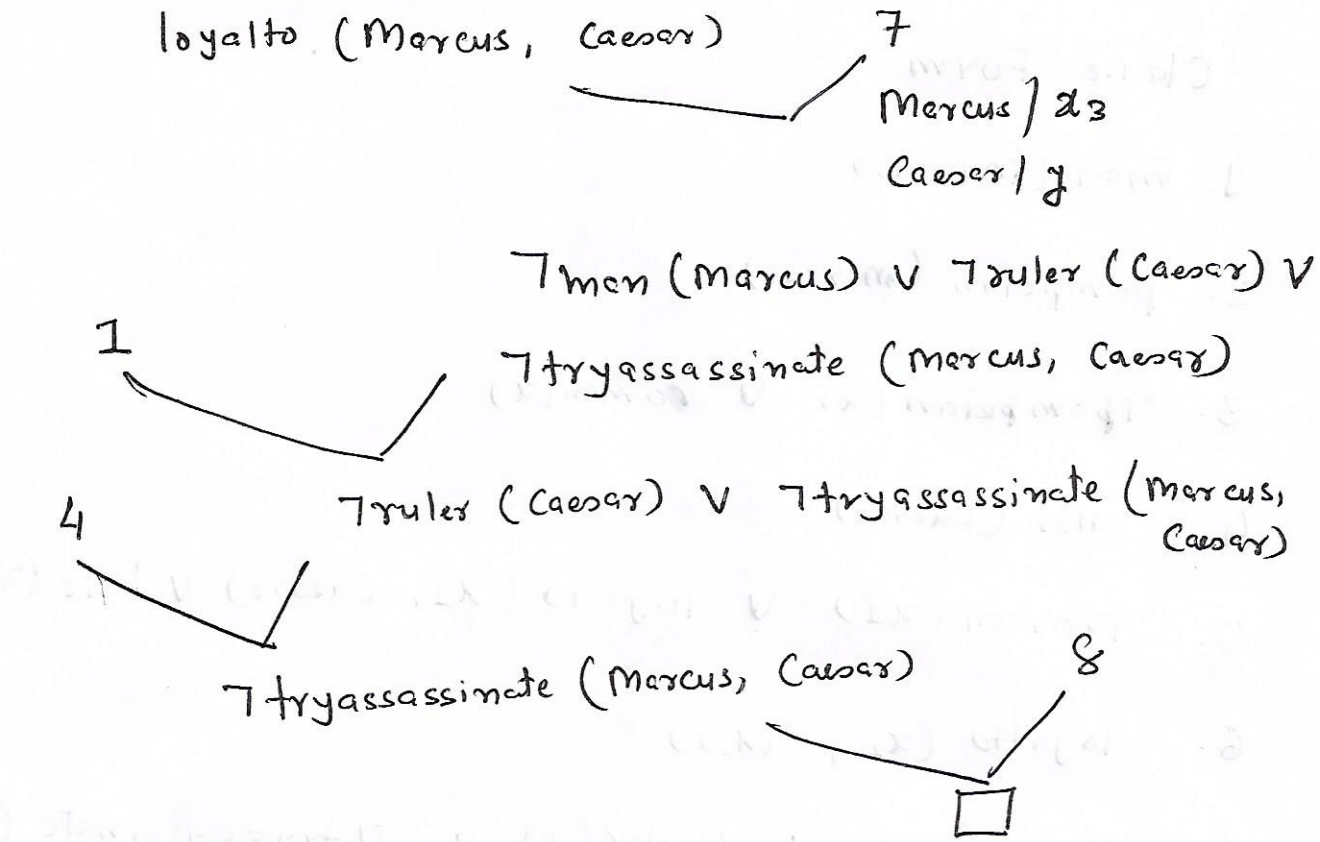
Marcus / x

¬pompeian (Marcus) ∨ loyalto (Marcus, Caesar)

2

loyalto (Marcus, Caesar)

...



$\Rightarrow$ ¬hate (Marcus, Caesar) is false.

$\therefore$ hate (Marcus, Caesar)