Predicate Logic / Firs order Logic / First order Predicate Logic(FOPL)

First-Order Logic

- In propositional logic, we can only represent the facts, which are either true or false. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.
- **"Some humans are intelligent", or**
- **"Sachin likes cricket."**
- To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

First-Order Logic

- First-order logic is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in appropriate way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.

First-Order Predicate Calculus

- FOPL does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
- First-Order Logic includes:
  - Objects: people, numbers, places, ideas (atoms)
  - Relations: relationships between objects
    - Example: father(fred, mary)
    - Properties: properties of atoms
    - Example: red(ball)
  - Functions: father-of(mary), next(3), (any value in range)

Logics in General

- Ontological Commitment:
  - What exists in the world — TRUTH
  - PL : facts hold or do not hold.
  - FOL : objects with relations between them that hold or do not hold

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	degree of truth $\in [0, 1]$	known interval value

Syntax of FOL: Basic elements

- Constant Symbols:
  - Stand for objects
  - e.g., KingJohn, 2, UCI,...

- Predicate Symbols
  - Stand for relations
  - E.g., Brother(Richard, John), greater_than(3,2)...

- Function Symbols
  - Stand for functions
  - E.g., Sqrt(3),...

Syntax of FOL: Basic elements

- Constants KingJohn, 2, UCI,...
- Predicates Brother, >,...
- Functions Sqrt,
- Variables x, y, a, b,...
- Connectives $\neg, \Rightarrow, \wedge, \vee, \Leftrightarrow$
- Equality =
- Quantifiers $\forall, \exists$

Syntax: First-Order Logic

- **Atomic sentences:**

➢ Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.

➢ We can represent atomic sentences as

Predicate(term1,term2, ……, term n).

- **Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).**

Chinky is a cat: => cat (Chinky).

- **Complex Sentences:**

➢ Complex sentences are made by combining atomic sentences using connectives.

Syntax: First-Order Logic

- **First-order logic statements can be divided into two parts:**

1. **Subject:** Subject is the main part of the statement.
2. **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

- **Example: "x is an integer."**,
- it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.
- “x is greater than 3”
- x |subject
- is greater than 3” | predicate

Syntax: First-Order Logic

- Examples:

- Father(x): unary predicate
- Brother(x,y): binary predicate
- Sum(x,y,z): ternary predicate
- P(x,y,z,t): n-ary predicate

Syntax: First-Order Logic

- **Quantifiers in First-order logic**

➢ A quantifier is a language element which generates quantification.

➢ These are the symbols that permit **to determine or identify the range and scope of the variable in the logical expression**. There are two types of quantifier:

1. **Universal Quantifier** (for all, everyone, everything)
2. **Existential quantifier**,(for some, at least one).

Syntax: First-Order Logic

- **Universal Quantifier**

➢ Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

➢ The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

➢ If x is a variable, then ∀x is read as:

  – **For all x**
  – **For each x**
  – **For every x.**

- Example:
- **All man drink coffee.** ∀x man(x) → drink (x, coffee).
- It will be read as: There are all x where x is a man who drink coffee.

Syntax: First-Order Logic

- **Existential Quantifier**

➢ Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

➢ It is denoted by the logical operator ∃, When it is used with a predicate variable then it is called as an existential quantifier.

➢ If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

  – **There exists a 'x.'**
  – **For some 'x.'**
  – **For at least one 'x.'**

- Example:
- **Some boys are intelligent.** ∃x: boys(x) ∧ intelligent(x)
- It will be read as: There are some x where x is a boy who is intelligent.

Points to remember:

1. The main connective for universal quantifier ∀ is implication →.
2. The main connective for existential quantifier ∃ is and ∧.

Properties of Quantifiers:

1. In universal quantifier, ∀x∀y is similar to ∀y∀x.
2. In Existential quantifier, ∃x∃y is similar to ∃y∃x.
3. ∃x∀y is not similar to ∀y∃x.

➢ Example:

- 1) ∀x∃y: For every x there exists a y (such that...)
  Everyone is married to someone (ie. For every person there exists a person to whom he/she is married)
- 2) ∃y∀x: There exists a y (such that) for every x...
  Someone is married to everyone (ie. There is someone who is married to everyone)

Some Examples of FOL using quantifier:

1. All birds fly.
   the predicate is "fly(bird)."
   ∀x bird(x) →fly(x).

2. Every man respects his parent.
   the predicate is "respect(x, y)," where x=man, and y= parent.
   Since there is every man so will use ∀, and it will be represented as follows:

∀x man(x) → respects (x, parent).

3. Not all students like both Mathematics and Science.

the predicate is "like(x, y)," where x= student, and y= subject.

Since there are not all students, so we will use ∀ with negation, so following representation for this:

¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

Free and Bound Variables

- The quantifiers interact with variables which appear in a suitable way. There are two types of variables in First-order logic:

➢ **Free Variable:** A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: ∀x ∃(y): [P (x, y, z)], where z is a free variable.

➢ **Bound Variable:** A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

➢ A variable x is bound in a wff ϕ if and only if it is in the scope of a quantifier ∀x or ∃x in ϕ; otherwise it is free.

Example: ∀x ∀y: [A (x) V B(y)], here x and y are the bound variables.

Syntax: First-Order Logic

- **Well Formed Formula (wff)**:
- A well-formed formula, is obtained by composing atoms with logical connectives and quantifiers. Therefore, a well-formed formula is a predicate with the following properties:

➢ All atomic formulas (atoms) are well-formed formulas.

➢ Combines using propositional connectives such as conjunction, disjunction and negation.

➢ If x is a variable and F is a well formed formula function, then both quantifiers are well formed formulas.

- A well-formed formula (wff) is a sentence containing no “free” variables. That is, all variables are “bound” by universal or existential quantifiers.

Syntax: First-Order Logic

- **Well Formed Formula**: A well-formed formula (wff) is a sentence containing no “free” variables. That is, all variables are “bound” by universal or existential quantifiers.
- **Quantifiers**

➢ **Universal quantification**

• (∀x)P(x) means that P holds for **all values of x** in the domain associated with that variable

• E.g., (∀x) dolphin(x) → mammal(x)

➢ **Existential quantification**

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

• E.g., (∃ x) mammal(x) ∧ lays-eggs(x)

Syntax: First-Order Logic

- **Some Predicate Calculus Equivalence**

Suppose p and q are predicates; X and Y are variables

- $\neg \exists X\ p(X) = \forall X \neg p(X)$
- $\neg \forall X\ p(X) = \exists X \neg p(X)$
- $\exists X\ p(X) = \exists Y\ p(Y)$
- $\forall X\ p(X) = \forall Y\ p(Y)$
- $X\ (p(X) \wedge q(X)) = \forall X\ p(X) \wedge \forall Y\ q(Y)$
- $\exists X\ (p(X) \wedge q(X)) = \exists X\ p(X) \wedge \forall Y\ q(Y)$

Examples

- All men are mortal

Examples

- All men are mortal
  - $\forall$ x [Man(x) -> Mortal(x)]

Examples

- All men are mortal

  $\forall$ x [Man(x) -> Mortal(x)]

- Socrates is a man

  Man(Socrates)

- Socrates is mortal

  Mortal(Socrates)

- All purple mushrooms are poisonous

  $\forall$ x [(Purple(x) ^ Mushroom(x)) -> Poisonous(x)]

- A mushroom is poisonous only if it is purple

  $\forall$ x [(Mushroom(x) ^ Poisonous(x)) -> Purple(x)]

Examples

- No human enjoys golf

$$\forall x\,[\,Human(x) \rightarrow \neg Enjoys(x, Golf)\,]$$

- Some professor that is not a historian writes programs

$$\exists x[\mathrm{Professor}(x) \wedge \neg Historian(x) \wedge Writes(x, \mathrm{Programs})]$$

- All birds fly. ∀x bird(x) →fly(x).
- Every man respects his parent. ∀x man(x) → respects (x, parent).
- Some boys play cricket. ∃x boys(x) ^ play(x, cricket).
- Not all students like both Mathematics and Science.

¬∀ (x) *student(x) → like(x, Mathematics) ∧ like(x, Science)].

Representing facts with Predicate Logic - Example

Ram is a man.

man(Ram)

God loves Ram.

loves(God, Ram)

Ram eats an apple.

eats(Ram, apple)

Sita drinks water.

drinks(sita, water)

Every man loves god.

$\forall$ x (x, man(x) --> loves (x, god))

Everyone loves someone

$\forall x \exists y$ Loves (x,y)

There is one person everyone loves

$\exists y \forall x$ Loves (x,y)

All birds have feathers.

$\forall$ x [Feathers(x) $\Rightarrow$ Bird(x)]

Every dog is an animal.

$\forall$ x (dog(x) $\Rightarrow$ animal (x))

Representing facts with Predicate Logic - Example

- Marcus was a man
- Marcus was a Pompeian
- All Pompeians were Romans
- Caesar was a ruler.
- All Romans were either loyal to Caesar or hated him.
- Everyone is loyal to someone.
- Men only try to assassinate rulers they are not loyal to.
- Marcus tried to assassinate Caesar

Representing facts with Predicate Logic - Example

$$\text{Man}(\text{Marcus})$$

$$\text{Pompeian}(\text{Marcus})$$

$$\forall\ x\ \text{Pompeian}(x) \Rightarrow\ \text{Roman}(x)$$

$$\text{Ruler}(\text{Caesar})$$

$$\forall\ x\ \text{Romans}(x) \Rightarrow\ \text{Loyalto}(x,\text{Caesar}) \vee \text{Hate}(x,\text{Caesar})$$

$$\forall\ x\ \exists y\ \text{Loyalto}(x,y)$$

$$\forall\ x\ \forall\ y\ \text{Man}(x) \wedge\ \text{Ruler}(y) \wedge\ \text{Tryassassinate}(x,y) \Rightarrow \neg\ \text{Loyalto}(x,y)$$

$$\text{Tryassassinate}(\text{Marcus},\text{Caesar})$$

Unification

- Unification is a process of making two different logical atomic expressions identical by finding a substitution.
- It takes two literals as input and makes them identical using substitution.
- Let Ψ_1 and Ψ_2 be two atomic sentences and σ be a unifier such that, $\boldsymbol{\Psi_1\sigma = \Psi_2\sigma}$, then it can be expressed as **UNIFY($\boldsymbol{\Psi_1, \Psi_2}$)**.
- **Example:** Let Ψ_1 = King(x), Ψ_2 = King(John),
- **Substitution θ = {John/x}** is a unifier for these atoms and applying this substitution, and both expressions will be identical.

Unification

- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).
- Unification is a key component of all first-order inference algorithms.
- It returns **fail** if the expressions **do not match** with each other.
- The substitution variables are called Most General Unifier or MGU.

➢ **E.g.** Let's say there are two different expressions, **P(x, y), and P(a, f(z))**.

- Substitute x with a, and y with f(z) in the first expression, and it will be represented as **a/x** and f(z)/y.
- With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: **[a/x, f(z)/y]**.

Unification

- 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.

Unification Algorithm

1. Initial predicate symbols must match.
2. For each pair of predicate arguments:

- different constants cannot match.
- a variable may be replaced by a constant.
- a variable may be replaced by another variable.
- a variable may be replaced by a function as long as the function does not contain an instance of the variable.

- When attempting to match 2 literals, all substitutions must be made to the entire literal.
- There may be many substitutions that unify 2 literals, the most general unifier is always desired.

Unification Examples

- P(x) and P(y): substitution = (x/y) (“substitute x for y”)
- P(x,f(y)) and P(Joe,z):(Joe/x, z/f(y))
- P(x) V Q(Jane) and P(Bill) V Q(y): (Bill/x, y/Jane)
- Man(Marcus) and ¬Man(spot) ((Not unified)
- Man(Marcus) and Man(Marcus, X) (Not unified)
- Man(Marcus) and Man(X) (X is substitute to Marcus so unified)
- Man(hate(X,Y)) and Man(hate(Marcus, spot))=(Marcus/X and spot/Y)

Normal Forms

- There are different normal forms are there to represent the facts.

1. Conjunctive Normal Form (CNF)
   conjunction of disjunctions of literals (conjunction of clauses) For example, (A v –B) ^ (B v –C v –D)
2. Disjunctive Normal Form (DNF)
   disjunction of conjunctions of literals (disjunction of terms) For example, (A ^ B) v (A ^ -C) v (A ^ -D) v (-B ^ -C) v (-B ^ -D)
3. Horn Form
   conjunction of *Horn clauses* (clauses with at most 1 positive literal)
   For example, (A v –B) ^ (– B v –C v –D)

Literals L ::= A (positive literal)
| ¬A (negative literal)

Convert first-order logic expressions to normal form

- **Why CNF (Conjunctive Normal Form)?**

➢ To make our representation simple.

➢ In wff, our representation of fact is very complicated.

➢ In CNF form, simplicity should be there and all quantifiers should not be there.

Convert first-order logic expressions to normal form

- **Rules to convert wff to CNF**

Step-1: Eliminate biconditionals and implications:

- Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
- Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.

Step-2: Move $\neg$ inwards:

- $\neg(\forall\ x\ p) \equiv \exists\ x\ \neg p$,
- $\neg(\exists\ x\ p) \equiv \forall\ x\ \neg p$,
- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$,
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$,
- $\neg\neg\alpha \equiv \alpha$.

Convert first-order logic expressions to normal form

- **Rules to convert wff to CNF**

Step-3: Standardize variables

renaming the variables (if required)

Step-4. **Skolemize**: each existential variable is replaced by a Skolem constant or Skolem function of the enclosing universally quantified variables.

- For instance, ∃x Rich(x) becomes Rich(G1)

where G1 is a new Skolem constant.

Convert first-order logic expressions to normal form

- **Rules to convert wff to CNF**

Step-5: Convert to prenex form by moving all universal quantifiers to the beginning of the wff.

Step-6: Drop universal quantifiers

- For instance, ∀ x Person(x) becomes Person(x).

Step-7. Distribute ∧ over ∨: Use distributive laws and equivalence rules of propositional logic to transform the matrix to CNF.

- $(\alpha \wedge \beta) \vee \gamma \equiv (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$.

Skolem functions

- ∃ y President(y)
- We replace y with a new function func:
- President(func())
- func is called a skolem function.

Example:

" ∀ x ∃ y Father(y,x)

create a new function named foo and replace y with the function.

" ∀ x Father(foo(x),x)

Convert first-order logic expressions to normal form

- **Example : Conversion to CNF**

∀ x: ([roman(x) ^ know(x , Markus)] -> [hate(x, ceaser)) V (∀ y: ∃ z: hate(y,z)) -> thinkcrazy(x,y)]

- Eliminate implications ⇔⇔, ⇒

∀ x: ¬ [roman(x) ^ know(x , Markus)] V [hate(x, ceaser) V ¬ (∀ y: ∃ z: hate(y,z)) V thinkcrazy(x,y)]

- Reduce scope of negations:

∀ x: [¬ roman(x) V ¬ know(x , Markus)] V [hate(x, ceaser) V ∃y: ∀ z: ¬ hate(y,z)) V thinkcrazy(x,y)]

- Standardize variables

Skip this step

Convert first-order logic expressions to normal form

- Skolemization:

∀ x: [¬ roman(x) V ¬ know(x , Markus)] V [hate(x, ceaser) V ∀ y ∀ z: ¬ hate(y,z)) V thinkcrazy(x,y)]

- Prenex form:

(∀x,y,z) [¬ roman(x) V ¬ know(x , Markus)] V [hate(x, ceaser) V (¬ hate(y,z)) V thinkcrazy(x,y)]

- Drop universal quantifiers

¬ roman(x) V ¬ know(x , Markus) V [hate(x, ceaser) V (¬ hate(y,z)) V thinkcrazy(x,y)]

- Convert to CNF

¬ roman(x) V ¬ know(x , Markus) V hate(x, ceaser) V ¬ hate(y,z) V thinkcrazy(x,y)

Convert first-order logic expressions to normal form

- **Example**

∀ x: (¬ literate(x) -> (¬write(x) ^ (¬ ∃ y: (read(x,y) ^ book(y)))

1. Eliminate implications ⇔, ⇒

∀ x: literate(x) V (¬write(x) ^ (¬ ∃ y: (read(x,y) ^ book(y)))

2. Reduce scope of negations:

∀ x: literate(x) V (¬write(x) ^ ∀ y: ¬read(x,y) V ¬book(y))

3. Standardize variables if any

skip this step

□ Prenex form:

∀ x: ∀ y: literate(x) V (¬write(x)) ^ (¬read(x,y) V ¬book(y))

Convert first-order logic expressions

to normal forms

4. Drop universal quantifier

literate(x) V (¬write(x)) ^ (¬read(x,y) V ¬book(y))

5. Conjuction and disjunction

A V (B ^ (C VD)) = (A V B) ^ (A V (C V D))

where (C V D) = X

(literate(x) V ¬write(x)) ^ (literate (x) V (¬read(x,y) V ¬book(y)))

Here, A ^ B so A and B both are considered as separate facts.

i) literate(x) V ¬write(x)

ii) literate (x) V (¬read(x,y) V ¬book(y))

Resolution

- Resolution is a procedure to prove a statement.
- 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 (unification). (From Knowledge base find out fact/term as negative meaning so both will be elemented)

Resolution

- **Example**
- Marcus was a man
- Marcus was a Pompeian
- All Pompeians were Romans
- Caesar was a ruler.
- All Romans were either loyal to Caesar or hated him.
- Everyone is loyal to someone.
- Men only try to assassinate rulers they are not loyal to.
- Marcus tried to assassinate Caesar

Knowledge Base in First Order Logic

$\text{Man}(\text{Marcus})$

$\text{Pompeian}(\text{Marcus})$

$\forall\ x\ \text{Pompeian}(x) \Rightarrow\ \text{Roman}(x)$

$\text{Ruler}(\text{Caesar})$

$\forall\ x\ \text{Romans}(x) \Rightarrow\ \text{Loyalto}(x,\text{Caesar}) \vee \text{Hate}(x,\text{Caesar})$

$\forall\ x\ \exists y\ \text{Loyalto}(x,y)$

$\forall\ x\ \forall\ y\ \text{Man}(x) \wedge\ \text{Ruler}(y) \wedge\ \text{Tryassassinate}(x,y) \Rightarrow$
$\neg\ \text{Loyalto}(x,y)$

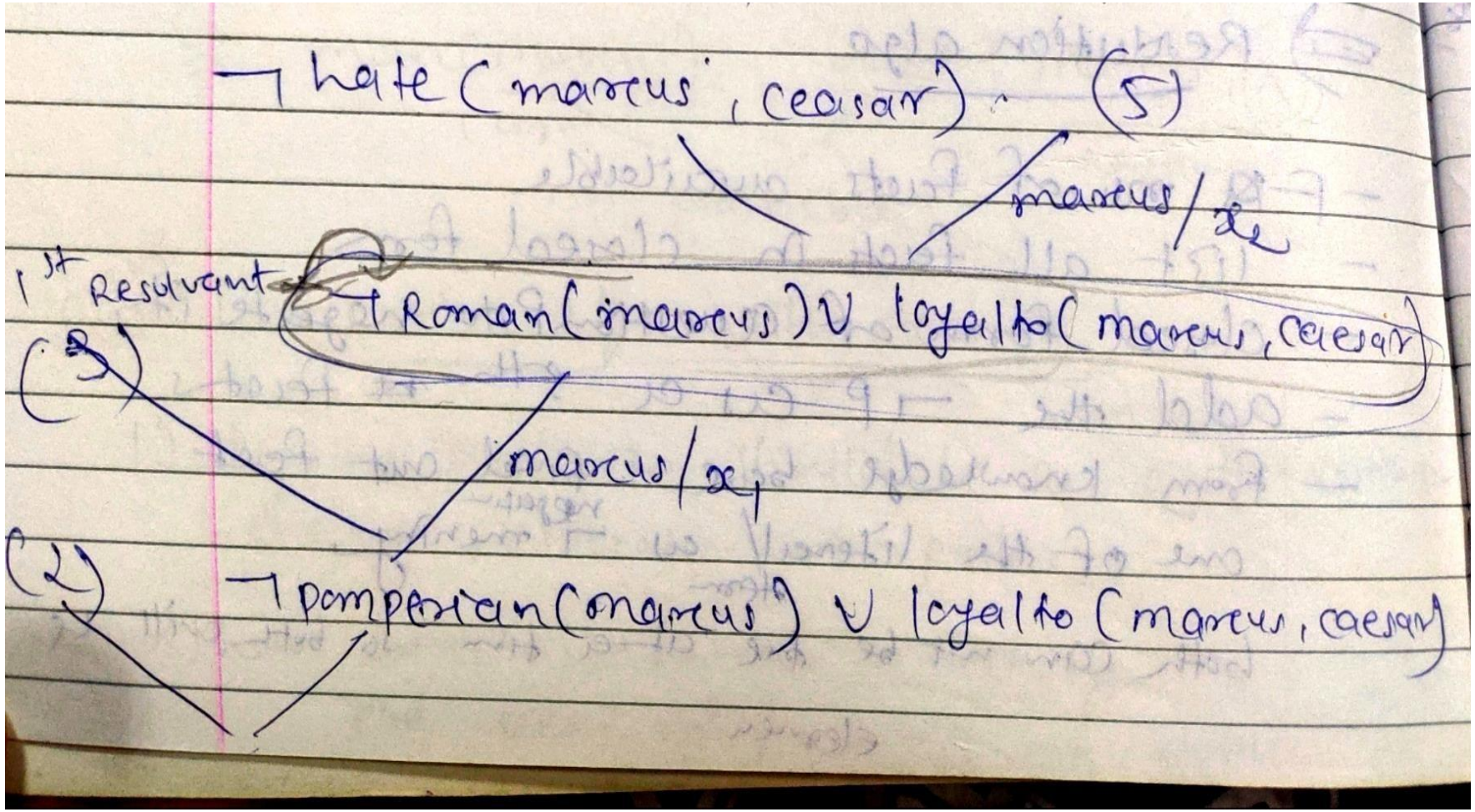
$\text{Tryassassinate}(\text{Marcus},\text{Caesar})$

Knowledge Base in CNF

- Man(Marcus)
- Pompeian(Marcus)
- ¬ Pompeian(x_1) ∨ Roman(x_1)
- Ruler(Caesar)
- ¬ Romans(x_2) ∨ Loyalto(x_2,Caesar) ∨ Hate(x_2,Caesar)
- Loyalto(x_3, $f(x_3)$)
- ¬ Man(x_4) ∨ ¬ Ruler(y_1) ∨ ¬ Tryassassinate(x_4,y_1) ∨ Loyalto(x_4,y_1)
- Tryassassinate(Marcus,Caesar)

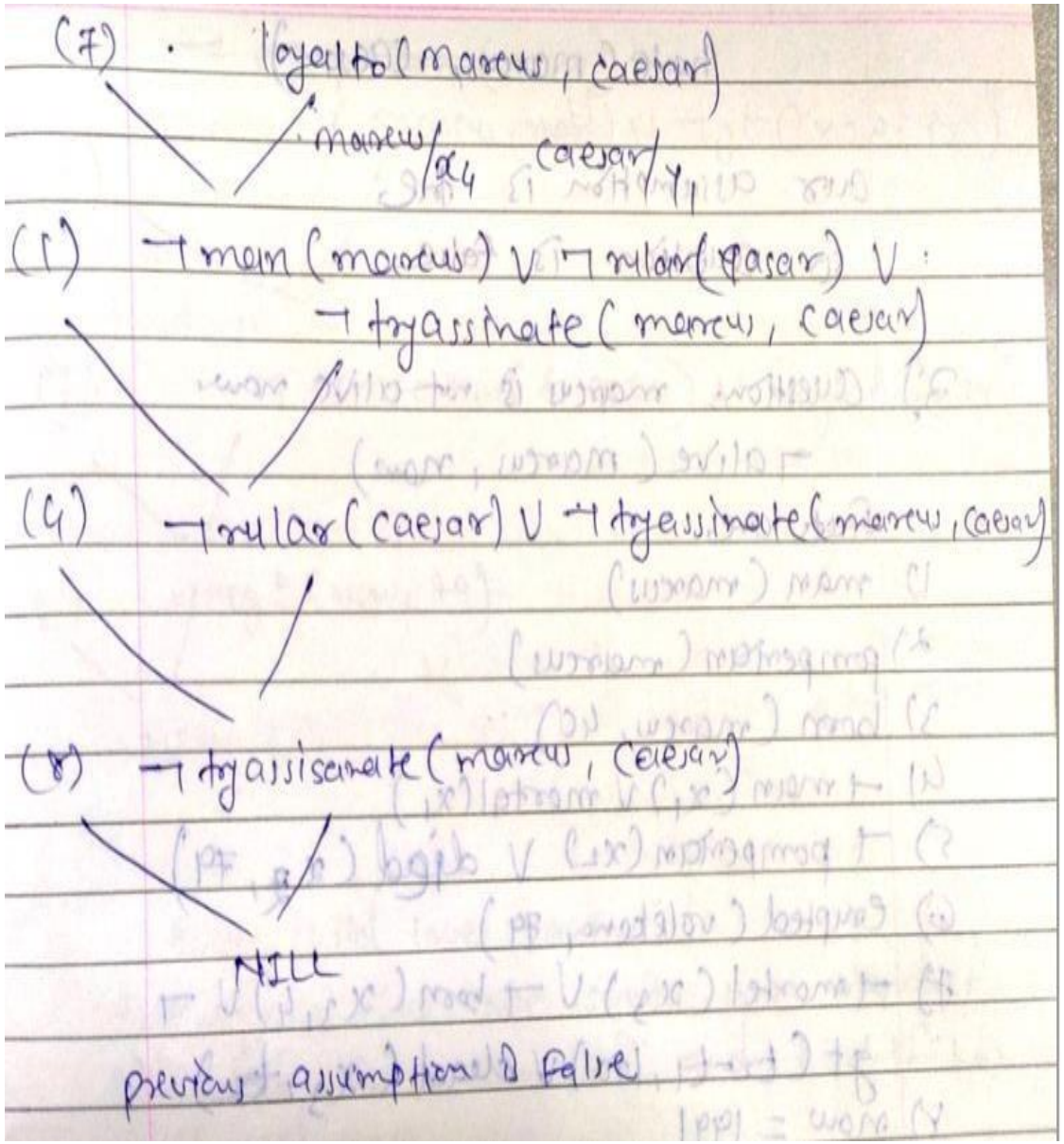
Knowledge Base in CNF

- Query: Does Marcus hate Caesar?
  - Negation of Query ¬Hate(Marcus, Caesar)



Resolution

- Does Marcus hate Caesar?



- So hate(Markus, ceaser) is true.

Representing Instance and ISA Relationships

- Specific attributes instance and isa play an important role particularly in a useful form of reasoning called property inheritance.
- The predicates instance and isa explicitly captured the relationships they used to ass inclusion.

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) $\vee$ hate(x, Caesar).

1. instance(Marcus, man).
2. instance(Marcus, Pompeian).
3. $\forall x$: instance(x, Pompeian) $\rightarrow$ instance(x, Roman).
4. instance(Caesar, ruler).
5. $\forall x$: instance(x, Roman). $\rightarrow$ loyalto(x, Caesar) $\vee$ 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). $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).
6. $\forall x$: $\forall y$: $\forall z$: instance(x, y) $\wedge$ isa(y, z) $\rightarrow$ instance(x, z).

Representing Instance and ISA Relationships

- The first part of the figure contains, class membership represented with unary predicates (such as Roman), each of which corresponds to a class.
- Asserting that P(x) is true is equivalent to asserting that x is an instance (or element) of P.
- The second part of the figure contains representations that use the instance predicate explicitly.
- The predicate instance is a binary one, whose first argument is an object and whose second argument is a class to which the object belongs.
- But these representations do not use an explicit isa predicate.
- Instead, subclass relationships, such as that between Pompeians and Romans, described as shown in sentence 3.

Representing Instance and ISA Relationships

- The implication rule states that if an object is an instance of the subclass Pompeian then it is an instance of the superclass Roman.
- The third part contains representations that use both the instance and isa predicates explicitly.
- The use of the isa predicate simplifies the representation of sentence 3, but it requires that one additional axiom (shown here as number 6) be provided.

Computable Functions and Predicates

- To express simple facts, such as the following greater-than and less-than relationships:
- gt(1,O) It(0,1) gt(2,1) It(1,2) gt(3,2) It(2,3)
- It is often also useful to have computable functions as well as computable predicates.
- Thus we might want to be able to evaluate the truth of gt(2 + 3,1)
- To do so requires that we first compute the value of the plus function given the arguments 2 and 3, and then send the arguments 5 and 1 to gt.

Computable Functions and Predicates

- Consider the following set of facts, again involving

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.

x: man(x) → mortal(x)

5) All Pompeians died when the volcano erupted in 79 A.D.

erupted(volcano, 79) ∧ ∀ x : [Pompeian(x) → died(x, 79)]

6) No mortal lives longer than 150 years.

x: t1: At2: *mortal(x) born(x, t1) gt(t2 – t1,150) → died(x, t2)*

7) It is now 1991.

***now* = 1991**

Computable Functions and Predicates

- So, Now suppose we want to answer the question “Is Marcus alive?”
- The statements suggested here, there may be two ways of deducing an answer.
- Either we can show that Marcus is dead because he was killed by the volcano or we can show that he must be dead because he would otherwise be more than 150 years old, which we know is not possible.
- Also, As soon as we attempt to follow either of those paths

rigorously, however, we discover, just as we did in the last example, that we need some additional knowledge. For example, our statements talk about dying, but they say nothing that relates to being alive, which is what the question is asking.

Computable Functions and Predicates

- So we add the following facts:

8) Alive means not dead.

$x: t: [alive(x, t) \rightarrow \neg dead(x, t)] [\neg dead(x, t) \rightarrow alive(x, t)]$

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

$x: t1: At2: died(x, t1)\ gt(t2, t1) \rightarrow dead(x, t2)$

So, based on these facts, we can answer the question "Is Marcus alive?" by proving: $\neg$ ***alive(Marcus, now)***