SYMBOLIC LOGIC

PROPOSITIONAL LOGIC IN ARTIFICIAL INTELLIGENCE

Logic is concerned with reasoning and the validity of arguments. In general, in logic, we are not concerned with the truth of statements, but rather with their validity. That is to say, although the following argument is clearly logical, it is not something that we would consider to be true:

- All lemons are blue
- Mary is a lemon
- Therefore, Mary is blue

This set of statements is considered to be valid because the conclusion (Mary is blue) follows logically from the other two statements, which we often call the premises.

1. Propositional Logic
2. Predicate Logic/ First order Logic

PROPOSITIONAL LOGIC IN ARTIFICIAL INTELLIGENCE

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:

a) It is Sunday.

b) The Sun rises from West (False proposition)

c) 3+3= 7(False proposition)

d) 5 is a prime number.

Syntax of propositional logic:

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

1.Atomic Propositions

2.Compound propositions

- **Atomic Proposition:** Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

Example:

a) 2+2 is 4, it is an atomic proposition as it is a **true** fact.

b) "The Sun is cold" is also a proposition as it is a **false** fact.

- **Compound proposition:** Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

Example:

a) "It is raining today, and street is wet."

b) "Ankit is a doctor, and his clinic is in Mumbai."

LOGICAL CONNECTIVES

Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

1. **Negation:** A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.

2. **Conjunction:** A sentence which has ∧ connective such as, **P ∧ Q** is called a conjunction.
   Example: Rohan is intelligent and hardworking. It can be written as,
   P= Rohan is intelligent, **Q= Rohan is hardworking. → P∧ Q**.

3. **Disjunction:** A sentence which has ∨ connective, such as **P ∨ Q**. is called disjunction, where P and Q are the propositions.
   Example: "Ritika is a doctor or Engineer"
   Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as **P ∨ Q**.

4. **Implication:** A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as
   If it is raining, then the street is wet.
   Let P= It is raining, and Q= Street is wet, so it is represented as P → Q

5. **Biconditional:** A sentence such as **P⇔ Q is a Biconditional sentence, example If I am breathing, then I am alive**
   P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

LOGICAL CONNECTIVES

Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A ∨ B
→	Implies	Implication	A → B
⇔	If and only if	Biconditional	A⇔ B
¬ or ~	Not	Negation	¬ A or ¬ B

For Negation:

P	¬ P
True	**False**
False	**True**

For Conjunction:

P	Q	P∧ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

For disjunction:

P	Q	P ∨ Q.
True	True	**True**
False	True	**True**
True	False	**True**
False	False	**False**

For Implication:

P	Q	P→ Q
True	True	**True**
True	False	**False**
False	True	**True**
False	False	**True**

For Biconditional:

P	Q	P⇔ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**True**

PRECEDENCE OF CONNECTIVES

Precedence	Operators
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction(AND)
Fourth Precedence	Disjunction(OR)
Fifth Precedence	Implication
Six Precedence	Biconditional

PROPERTIES OF OPERATORS

- **Commutativity:**
  - P∧ Q= Q ∧ P, or
  - P ∨ Q = Q ∨ P.
- **Associativity:**
  - (P ∧ Q) ∧ R= P ∧ (Q ∧ R),
  - (P ∨ Q) ∨ R= P ∨ (Q ∨ R)
- **Identity element:**
  - P ∧ True = P,
  - P ∨ True= True.
- **Distributive:**
  - P∧ (Q ∨ R) = (P ∧ Q) ∨ (P ∧ R).
  - P ∨ (Q ∧ R) = (P ∨ Q) ∧ (P ∨ R).
- **DE Morgan's Law:**
  - ¬ (P ∧ Q) = (¬P) ∨ (¬Q)
  - ¬ (P ∨ Q) = (¬ P) ∧ (¬Q).
- **Double-negation elimination:**
  - ¬ (¬P) = P.

Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic. Example:
  - **All the girls are intelligent.**
  - **Some apples are sweet.**
- Propositional logic has limited expressive power.
- In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

RULES OF INFERENCE

In artificial intelligence, we need intelligent computers which can create new logic from old logic or by evidence, **so generating the conclusions from evidence and facts is termed as Inference**.

Inference rules

Inference rules are the templates for generating valid arguments. Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal.

In inference rules, the implication among all the connectives plays an important role. Following are some terminologies related to inference rules:

- **Implication:** It is one of the logical connectives which can be represented as $P \rightarrow Q$. It is a Boolean expression.
- **Converse:** The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as $Q \rightarrow P$.
- **Contrapositive:** The negation of converse is termed as contrapositive, and it can be represented as $\neg Q \rightarrow \neg P$.
- **Inverse:** The negation of implication is called inverse. It can be represented as $\neg P \rightarrow \neg Q$.

RULES OF INFERENCE

From the above term some of the compound statements are equivalent to each other, which we can prove using truth table:

P	Q	P → Q	Q→ P	¬ Q → ¬ P	¬ P → ¬ Q.
T	T	T	T	T	T
T	F	F	T	F	T
F	T	T	F	T	F
F	F	T	T	T	T

RULES OF INFERENCE

Types of Inference rules:

1. Modus Ponens:

The Modus Ponens rule is one of the most important rules of inference, and it states that if P and P → Q is true, then we can infer that Q will be true. It can be represented as:

Notation for Modus ponens: $\frac{P \rightarrow Q, \quad P}{\therefore Q}$

Example:

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q

Statement-2: "I am sleepy" ==> P

Conclusion: "I go to bed." ==> Q.

Hence, we can say that, if P→ Q is true and P is true then Q will be true.

2. Modus Tollens:

The Modus Tollens rule state that if P→ Q is true and **¬ Q is true, then ¬ P** will also true. It can be represented as:

Notation for Modus Tollens: $\frac{P \rightarrow Q, \quad \sim Q}{\sim P}$

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q

Statement-2: "I do not go to the bed."==> ~Q

Statement-3: Which infers that "**I am not sleepy**" => ~P

RULES OF INFERENCE

3. Hypothetical Syllogism:

The Hypothetical Syllogism rule state that if P→R is true whenever P→Q is true, and Q→R is true. It can be represented as the following notation:

Example:

Statement-1: If you have my home key then you can unlock my home. **P→Q**
Statement-2: If you can unlock my home then you can take my money. **Q→R**
Conclusion: If you have my home key then you can take my money. **P→R**

4. Disjunctive Syllogism:

The Disjunctive syllogism rule state that if P∨Q is true, and ¬P is true, then Q will be true. It can be represented as:

Notation of Disjunctive syllogism: $\frac{P \vee Q,\ \neg P}{Q}$

Example:

Statement-1: Today is Sunday or Monday. ==>P∨Q
Statement-2: Today is not Sunday. ==> ¬P
Conclusion: Today is Monday. ==> Q

RULES OF INFERENCE

5. Addition:

The Addition rule is one the common inference rule, and it states that If P is true, then P∨Q will be true.

Notation of Addition: $\frac{P}{P \vee Q}$

Example:

Statement: I have a vanilla ice-cream. ==> P

Statement-2: I have Chocolate ice-cream.

Conclusion: I have vanilla or chocolate ice-cream. ==> (P∨Q)

6. Simplification:

The simplification rule state that if **P∧ Q** is true, then **Q or P** will also be true. It can be represented as:

Notation of Simplification rule: $\frac{P \wedge Q}{Q}$ Or $\frac{P \wedge Q}{P}$

Proof by Truth-Table:

P	Q	$P \wedge Q$
0	0	0
1	0	0
0	1	0
1	1	1

7. Resolution:

The Resolution rule state that if P∨Q and ¬ P∧R is true, then Q∨R will also be true. **It can be represented as**

Notation of Resolution $\frac{P \vee Q,\ \neg P \wedge R}{Q \vee R}$

TRANSLATING BETWEEN ENGLISH AND LOGIC NOTATION

1. **"It is raining and it is Tuesday."**

might be expressed as: $R \wedge T$,

Where R means "it is raining" and T means "it is Tuesday."

2. "it is raining in New York" or "it is raining heavily" or even **"it rained for 30 minutes on Thursday"** , then R will probably not suffice.

To express more complex concepts like these, we usually use **predicates**.

Hence, for example, we might translate

"it is raining in New York" as: **N(R)** We might equally well choose to write it as: **R(N)**

- When we write **N(R)**, we are saying that a property of the rain is that it is in New York, whereas with **R(N)** we are saying that a property of New York is that it is raining. Which we use depends on the problem we are solving.
- It is likely that if we are solving a problem about New York, we would use **R(N)**, whereas if we are solving a problem about the location of various types of weather, we might use **N(R)**.

Translating between English and Logic Notation

Jeffery is happy.

Solomon and Kevin are both dogs.

Carlos is happier than Sue, but sadder than Fred.

James is a troublemaker when Kevin dislikes him.

Whenever he eats sandwiches that have pickles in them, he ends up either asleep at his desk or singing loud songs

First-Order Logic (Predicate Logic)

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise 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 logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
  - **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, Wumpus, ...
  - **Relations: It can be unary relation such as:** red, round, is adjacent, **or n-any relation such as:** the sister of, brother of, has color, comes between
  - **Function:** Father of, best friend, third inning of, end of, ...
- As a natural language, first-order logic also has two main parts:
  - **Syntax**
  - **Semantics**

First-Order Logic (Predicate Logic)

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

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:

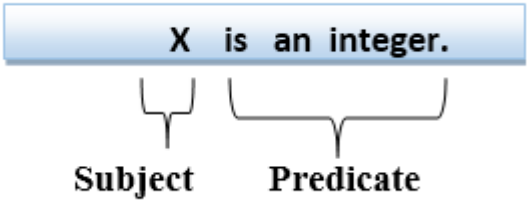
- Complex sentences are made by combining atomic sentences using connectives.

First-Order Logic (Predicate Logic)

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

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

Consider the statement: "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.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- 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:
  - **Universal Quantifier, (for all, everyone, everything)**
  - **Existential quantifier, (for some, at least one).**

First-Order Logic (Predicate 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.

Note: In universal quantifier we use implication "→".

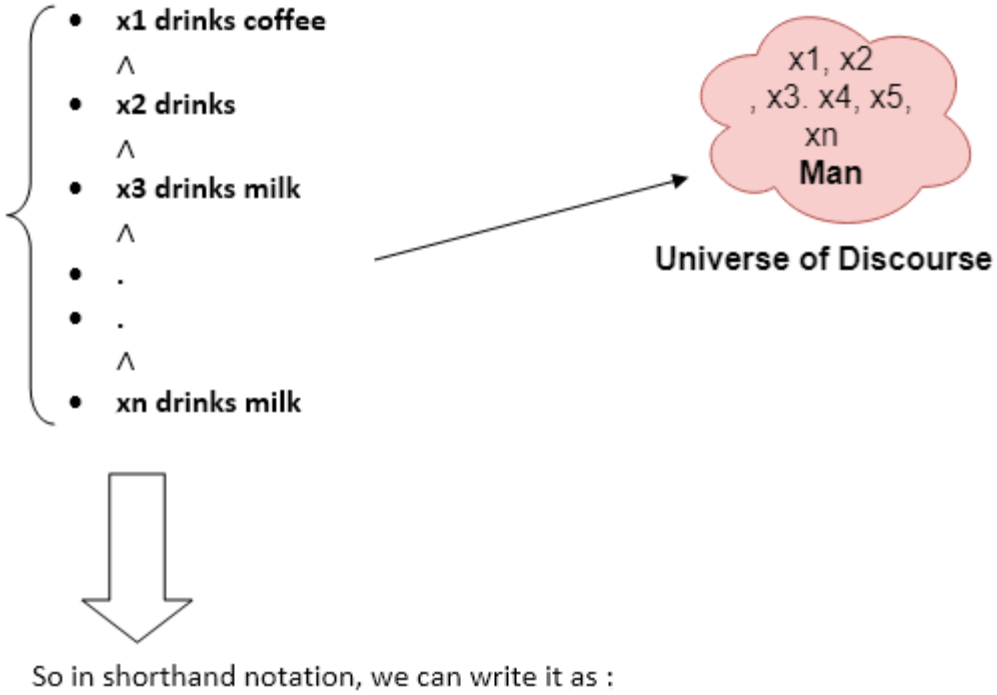
If x is a variable, then ∀x is read as:

- **For all x**
- **For each x**
- **For every x.**

Example:

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as seen:



∀x man(x) → drink (x, coffee).

First-Order Logic (Predicate 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 ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

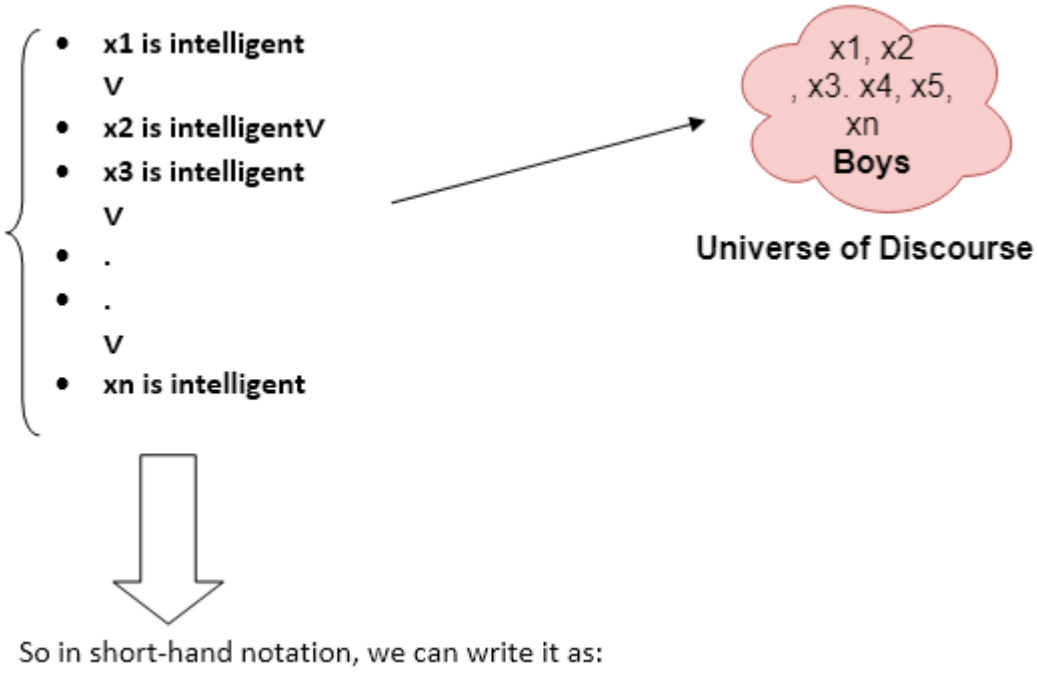
Note: In Existential quantifier we always use AND or Conjunction symbol (∧).

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.

FIRST-ORDER LOGIC (PREDICATE LOGIC)

1. All birds fly.

In this question the predicate is "**fly(bird)**."
And since there are all birds who fly so it will be represented as follows.

∀x bird(x) →fly(x).

2. Every man respects his parent.

In this question, 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. Some boys play cricket.

In this question, the predicate is "**play(x, y)**," where x= boys, and y= game. Since there are some boys so we will use **∃, and it will be represented as**:

∃x boys(x) → play(x, cricket).

FIRST-ORDER LOGIC (PREDICATE LOGIC)

4. Not all students like both Mathematics and Science.

In this question, 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)].

5. Only one student failed in Mathematics.

In this question, the predicate is "**failed(x, y),**" **where x= student, and y= subject**.
Since there is only one student who failed in Mathematics, so we will use following representation for this:

∃(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)].

INFERENCE IN FIRST-ORDER LOGIC

Substitution:

Substitution is a basic procedure that is applied to terms and formulations. It can be found in all first-order logic inference systems. When there are quantifiers in FOL, the substitution becomes more complicated. When we write **F[a/x]**, we are referring to the substitution of a constant "**a**" for the variable "**x**."

[Note: first-order logic can convey facts about some or all of the universe's objects.]

Equality:

In First-Order Logic, atomic sentences are formed not only via the use of predicate and words, but also through the application of equality. We can do this by using **equality symbols**, which indicate that the two terms relate to the same thing.

Example: Brother (John) = Smith.

In the above example, the object referred by the **Brother (John)** is close to the object referred by **Smith**. The equality symbol can be used with negation to portray that two terms are not the same objects.

Example: ¬(x=y) which is equivalent to x ≠y.

INFERENCE IN FIRST-ORDER LOGIC

FOL inference rules for quantifier:

First-order logic has inference rules similar to propositional logic, therefore here are some basic inference rules in FOL:

1. Universal Generalization:

- Universal generalization is a valid inference rule that states that if premise P(c) is true for any arbitrary element c in the universe of discourse, we can arrive at the conclusion x P. (x).
- It can be represented as:

$$\frac{P(c)}{\forall x\, P(x)}$$

- If we want to prove that every element has a similar property, we can apply this rule.
- x must not be used as a free variable in this rule.

Example: Let's represent, P(c): **"A byte contains 8 bits"**, so "**All bytes contain 8 bits.**" for ∀ x P(x) , it will also be true.

INFERENCE IN FIRST-ORDER LOGIC

FOL inference rules for quantifier:

First-order logic has inference rules similar to propositional logic, therefore here are some basic inference rules in FOL:

2. Universal Instantiation:

- A valid inference rule is universal instantiation, often known as universal elimination or UI. It can be used to add additional sentences many times.
- The new knowledge base is logically equal to the existing knowledge base.
- **We can infer any phrase by replacing a ground word for the variable**, according to UI
- The UI rule say that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from **∀ x P(x) for any object in the universe of discourse.**
- It can be represented as

$$\frac{P(c)}{\forall x\, P(x)}$$

Example: 1 IF "Every person like ice-cream"=> ∀x P(x) so we can infer that "John likes ice-cream" => P(c)

INFERENCE IN FIRST-ORDER LOGIC

Example: 2 Let's take a famous example,

"All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL: **∀x king(x) ∧ greedy (x) → Evil (x),**

We can infer any of the following statements using Universal Instantiation from this information:

- **King(John) ∧ Greedy (John) → Evil (John),**
- **King(Richard) ∧ Greedy (Richard) → Evil (Richard),**
- **We can infer any phrase by replacing a ground word for the variable, according to UI**
- **King(Father(John)) ∧ Greedy (Father(John)) → Evil (Father(John)),**

INFERENCE IN FIRST-ORDER LOGIC

FOL inference rules for quantifier:

First-order logic has inference rules similar to propositional logic, therefore here are some basic inference rules in FOL:

3. Existential Instantiation:

- Existential instantiation is also known as Existential Elimination, and it is a legitimate first-order logic inference rule.
- It can only be used to replace the existential sentence once.
- Although the new KB is not conceptually identical to the old KB, it will be satisfiable if the old KB was.
- This rule states that for a new constant symbol c, one can deduce P(c) from the formula given in the form of x P(x).
- The only constraint with this rule is that c must be a new word for which P(c) is true.
- It's written like this:

$$\frac{\exists x\, P(x)}{P(c)}$$

INFERENCE IN FIRST-ORDER LOGIC

Example: 1

From the given sentence: **∃x Crown(x) ∧ OnHead(x, John),**

So we can infer: **Crown(K) ∧ OnHead(K, John)**, as long as K does not appear in the knowledge base.

- The above used K is a constant symbol, which is known as **Skolem constant**.
- The Existential instantiation is a special case of **Skolemization process**.

4. Existential introduction

1.An existential generalization is a valid inference rule in first-order logic that is also known as an existential introduction.

2.This rule argues that if some element c in the universe of discourse has the property P, we can infer that something in the universe has the attribute P.

3.It's written like this:

$$\frac{P(c)}{\exists x P(x)}$$

Example: Let's say that,

"Priyanka got good marks in English."

"Therefore, someone got good marks in English."

First-Order Logic (Predicate Logic)

Free and Bound Variables

There are two types of variables based upon their interaction with the quantifiers in a First Order Logic in AI, namely free and bound variables.

1. Free Variables:

Free variables are those variables that do not come under the scope of the quantifier. For instance, in an expression $\forall x \exists y P(x,y,z)$, z is a free variable because it doesn't come under the scope of any quantifier.

2. Bound Variables:

Bound variables are those variables that occur inside the scope of the quantifier. For instance, in an expression $\forall x \exists y P(x,y,z)$, x and y are bound variables because they occur inside the scope of the quantifiers.

First-Order Logic (Predicate Logic)

In logic, a **Well-Formed Formula (WFF)** is a string of symbols that is syntactically correct and can be interpreted as a meaningful statement. Essentially, it's a formula that follows the rules of a specific logical language, making it a valid expression within that system.

- **Symbols:**

WFFs are built from symbols like propositional variables (e.g., P, Q), logical connectives (e.g., ∧ for AND, ∨ for OR, ¬ for NOT), and parentheses for grouping.

- **Formation Rules:**

Specific rules dictate how these symbols can be combined to create a WFF. These rules vary depending on the type of logic (propositional, predicate, etc.).

- **Meaningful Statement:**

A WFF represents a proposition (a statement that can be true or false) and can be assigned a truth value (true or false) based on the meaning of its components and the rules of the logic system.

First-Order Logic (Predicate Logic)

Convert the following sentences to Predicate Logic

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Pompeians were either loyal to Caesar or hated him.
6. Every one is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.
9. All men are people.

Using Predicate Logic to prove that,

Was Marcus loyal to Caesar?

First-Order Logic (Predicate Logic)

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Pompeians were either loyal to Caesar or hated him.
6. Every one is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.
9. All men are people.

First-Order Logic (Predicate Logic)

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\forall x: Pompeian(x) \rightarrow Roman(x)$
4. $ruler(Caesar)$
5. $\forall x: Roman(x) \rightarrow loyalto\,(x, Caesar) \vee hate\,(x, Caesar)$
6. $\forall x: \exists y: loyalto(x, y)$
7. $\forall x: \forall y: person(x) \wedge ruler(y) \wedge tryassassinate(x, y) \rightarrow \neg loyalto\,(x, Caesar)$
8. $tryassassinate(Marcus, Caesar)$
9. $\forall x: man(x) \rightarrow person(x)$

First-Order Logic (Predicate Logic)

$$nil$$

$$\downarrow (1)$$

$$man(Marcus)$$

$$\downarrow (9)$$

$$person(Marcus)$$

$$\downarrow (8)$$

$$person(Marcus) \wedge tryassassinate(Marcus, Caesar)$$

$$\downarrow (4)$$

$$person(Marcus) \wedge tryassassinate(Marcus, Caesar) \wedge ruler(Caesar)$$

$$\downarrow (7, \text{Substitution})$$

$$\neg loyalto(Marcus, Caesar)$$

First-Order Logic (Predicate Logic)

2. Was Marcus hates Caeser?

$$\neg loyalto(Marcus, Caesar)$$
$$\downarrow (2)$$
$$Pompeian(Marcus)\ \neg loyalto(Marcus, Caesar)$$
$$\downarrow (3)$$
$$Roman(Marcus)\ \neg loyalto(Marcus, Caesar)$$
$$\downarrow (53)$$
$$hate(Marcus, Caesar)$$

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\forall x:\ Pompeian(x) \rightarrow Roman(x)$
4. $ruler(Caesar)$
5. $\forall x:\ Roman(x) \rightarrow loyalto\ (x, Caesar) \vee hate\ (x, Caesar)$
6. $\forall x:\ \exists y: loyalto(x, y)$
7. $\forall x: \forall y: person(x) \wedge ruler(y) \wedge tryassassinate(x, y) \rightarrow \neg loyalto\ (x, Caesar)$
8. $tryassassinate(Marcus, Caesar)$
9. $\forall x: man(x) \rightarrow person(x)$

REPRESENTING INSTANCE AND ISA RELATIONSHIPS

- Specific attributes instance and isa play important role particularly in a useful form of reasoning called property inheritance.
- The predicates instance and isa explicitly captured the relationships they are used to express, namely class membership and class inclusion.
- Fig. 4.2 shows the first five sentences of the last section represented in logic in three different ways.
- The first part of the figure contains the representations we have already discussed. In these representations, class membership is 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.

REPRESENTING INSTANCE AND ISA RELATIONSHIPS

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 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, are described as shown in sentence 3.
- The implication rule states that if an object is an instance of the subclass Pompeian then it is an instance of the superclass Roman.
- Note that this rule is equivalent to the standard set-theoretic definition of the subclass-superclass relationship.
- 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

It may be necessary to compute functions as part of a fact. In these cases a computable predicate is used. A computable predicate may include computable functions such as +, -, *, etc. For example, gt(x-y,10) →bigger(x) contains the computable predicate gt which performs the greater than function. Note that this computable predicate uses the computable function subtraction.

Example: Consider the following statements:

1. Marcus was a man.
2. Marcus was Pompeian.
3. Marcus was born in 40 A.D.
4. All men are mortal.
5. All Pompeians died when the volcano erupted in 79 A.D.
6. No mortal lives longer than 150 years.
7. It is now 2024.
8. Alive means not dead.
9. If someone dies, he is dead at all later times.

COMPUTABLE FUNCTIONS AND PREDICATES

1. Marcus was a man.

   man(Marcus)

2. Marcus was 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) /\ $\forall x$: Pompeian(x) $\rightarrow$ died(x,79)

6. No mortal lives longer than 150 years.

   $\forall x \forall t1 \forall t2$: mortal(x) /\ born(x,t1) /\ gt(t2-t1,150) $\rightarrow$ dead(x,t2)

7. It is now 2024.

   now=2024

8. Alive means not dead.

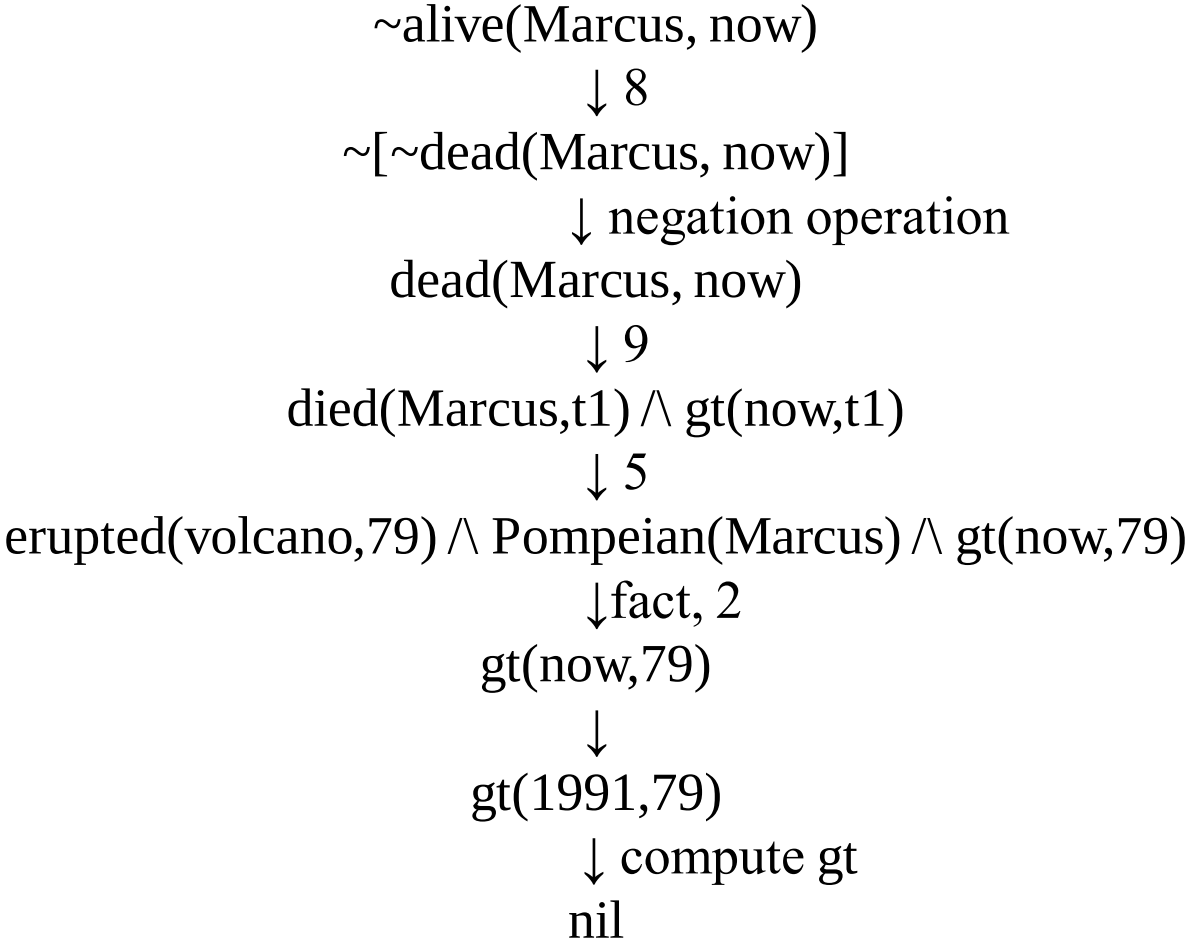
   $\forall x \forall t$: [alive(x,t) $\rightarrow$ ~dead(x,t)] /\ [~dead(x,t) $\rightarrow$alive(x,t)]

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

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

COMPUTABLE FUNCTIONS AND PREDICATES

Suppose we want to answer the question "**Is Marcus alive now?**". We can do this by either proving **alive(Marcus, now)** or **~alive(Marcus, now)**.



RESOLUTION

In the previous sections facts were proved or queries were answered using backward chaining. In this section we will examine the use of resolution for this purpose. Resolution proves facts and answers queries by **refutation**. This involves assuming the fact/query is untrue and reaching a contradiction which indicates that the opposite must be true.

Algorithm: Converting wffs to Clause Form

1. Remove all implies, i.e. → by applying the following: a → b is equivalent to ~a \/ b.
2. Use the following rules to reduce the scope of each negation operator to a single term:
   - ~(~a) = a
   - ~(a /\ b) = ~a \/ ~b
   - ~(a \/ b) = ~a /\ ~b
   - ~∀x: p(x) = ∃x: ~p(x)
   - ~∃x: p(x) = ∀x: ~p(x)
3. Each quantifier must be linked to a unique variable. For example, consider ∀x: p(x) \/ ∀x: q(x). In this both quantifiers are using the same variable and one must changed to another variable: ∀x: p(x) \/ ∀y: q(y).

RESOLUTION

3. Move all quantifiers, in order, to the left of each wff.
4. Remove existential quantifiers by using **Skolem constants or functions**. For example, ∃x: p(x) becomes p(s1) and ∀x ∃y: q(x,y) is replaced with ∀x: q(s2(x), x).
5. Drop the quantifier prefix.
6. Apply the associative property of disjunctions: a ∨ (b ∨ c) = (a ∨ b) ∨ c and remove brackets giving a ∨ b ∨ c.
7. Remove all disjunctions of conjunctions from predicates, i.e. create conjunctions of disjunctions instead, by applying the following rule iteratively: (a ∧ b) ∨ c = (a ∨ c) ∧ (b ∨ c).
8. Create a separate clause for each conjunction.
9. Rename variables in the clauses resulting from step 9 to ensure that no two clauses refer to the same variable.

RESOLUTION

Algorithm: Resolution

1. Convert the wffs to clause form.
2. Add the fact (or query) P to be proved to the set of clauses:
    i. Negate P.
    ii. Convert negated P to clause form.
    iii. Add the result of ii to the set of clauses.
3. Repeat
    i. Select two clauses.
    ii. Resolve the clauses using unification.
    iii. If the resolvent clause is the empty clause, then a contradiction has been reached. If not add the resolvent to the set of clauses.

    Until (a contradiction is found OR no progress can be made)

RESOLUTION

Consider the following wffs:

1. man(Marcus)
2. Pompeian(Marcus)
3. ∀x: Pompeian(x) → Roman(x)
4. ruler(Caesar)
5. ∀x: Roman(x) → loyalto(x,Caesar) \/ hate(x,Caesar)
6. ∀x ∃y: loyalto(x,y)
7. ∀x ∀y: person(x) /\ ruler(y) /\ tryassassinate(x,y) → ~loyalto(x,y)
8. tryassassinate(Marcus,Caesar)
9. ∀x: man(x) → person(x)

Converting these to clause form gives:

1. man(Marcus)
2. Pompeian(Marcus)
3. ~Pompeian(x) \/ Roman(x)
4. ruler(Caesar)
5. ~Roman(x1) \/ loyalto(x1,Caesar) \/ hate(x1,Caesar)
6. loyalto(x2,s1(x2))
7. ~person(x3) \/ ~ruler(y) \/ ~tryassassinate(x3,y) \/ ~loyalto(x3,y)
8. tryassassinate(Marcus, Caesar)
9. ~man(x4) \/ person(x4)

RESOLUTION

Suppose that we want to prove that **Marcus hates Caesar**. We firstly convert this to a wff: **hate(Marcus,Caesar)**. The wff is then negated and converted to clause form: **~hate(Marcus,Caesar)**.

~hate(Marcus,Caesar)
↓ 5
~Roman(Marcus) \/ loyalto(x1,Caesar)
↓ 3
~Pompeian(Marcus) \/ loyalto(Marcus,Caesar)
↓ 2
loyalto(Marcus,Caesar)
↓ 7
~person(Marcus) \/ ~ruler(Caesar) \/ ~tryassassinate(Marcus,Caesar)
↓ 4
~person(Marcus) \/ ~tryassassinate(Marcus,Caesar)
↓ 8
~person(Marcus)
↓ 9
~man(Marcus)
↓ 1
□

RESOLUTION

Consider the wffs we created above:

1. man(Marcus)
2. Pompeian(Marcus)
3. born(Marcus,40)
4. ∀x: man(x) → mortal(x)
5. erupted(volcano,79) /\ ∀x: Pompeian(x) → died(x,79)
6. ∀x ∀t1 ∀t2: mortal(x) /\ born(x,t1) /\ gt(t2-t1,150) → dead(x,t2)
7. now=1991
8. ∀x ∀t: [alive(x,t) → ~dead(x,t)] /\ [~dead(x,t) →alive(x,t)]
9. ∀x ∀t1 ∀t2:died(x,t1) /\ gt(t2,t1) →dead(x,t2)

Suppose we now want to use resolution to prove that **"Marcus is not alive now"**. We firstly have to convert these statements to clause form:

1. man(Marcus)
2. Pompeian(Marcus)
3. born(Marcus,40)
4. ~man(x) \/ mortal(x)
5. erupted(volcano,79)
6. ~Pompeian(x1) \/ died(x1,79)
7. ~mortal(x2) \/ ~born(x2,t1) \/ ~gt(t2-t1,150) \/ dead(x2,t2)
8. now=1991
9. ~alive(x3,t) \/ ~dead(x3,t)
10. dead(x4,t3) \/ alive(x4,t3)
11. ~died(x5,t4) \/ ~gt(t5,t4) \/ dead(x5,t5

RESOLUTION

We want to prove **~alive(Marcus,now)**. We negate this and convert it clause form: **alive(Marcus, now)** and find a contradiction:

alive(Marcus,now)
↓10
dead(Marcus,now)
↓11
~died(Marcus,t4) \/~gt(now,t4)
↓6
~Pompeian(Marcus) \/~gt(now,79)
↓ 2
~gt(now,t4)
↓8
~gt (1991,79)
↓
□

RESOLUTION

1. Consider the following facts:
    1. John likes all kinds of food.
    2. 2. Apples are food.
    3. Chicken is food
    4. Anything anyone eats and is not killed by is food.
    5. Bill eats peanuts and is still alive.
    6. Sue eats everything Bill eats.

a) Convert the wffs for these facts to clause form.

b) Using resolution prove that **"John likes peanuts"**.

2. Consider the following facts:
    1. Steve only likes easy courses.
    2. Science courses are hard.
    3. All the courses in the basketweaving department are easy.
    4. BK301 is a basketweaving course.

a) Convert the wffs for these facts to clause form.

b) Use resolution to answer the question **"What course would Steve like?"**.