UNIT 2

CHAPTER 1

First Order Logic (FOL)

First Order Logic (FOL), also known as predicate logic or first-order predicate calculus, is a formal system used in mathematics, philosophy, linguistics, and computer science. It extends **propositional logic** by introducing **quantifiers** and the ability to refer to **objects** and their **relationships** in a domain. First-order logic allows for the expression of more complex statements than propositional logic, which only deals with true or false statements without considering the structure within those statements.

Key Components of First Order Logic

1. **Constants**: These are specific entities or objects in the domain. For example, in a domain of people, constants could be individual names like John, Alice, or Mary.
2. **Variables**: These represent elements from the domain. Examples include x, y, or z, which can refer to any object in the domain.
3. **Predicates**: Predicates are functions that represent relationships or properties of objects. For instance, Likes(John, Pizza) is a predicate expressing that John likes pizza.
4. **Functions**: Functions map objects to other objects in the domain. For example, MotherOf(x) refers to the mother of an individual x.
5. **Quantifiers**: There are two types of quantifiers:
    - **Universal quantifier (∀)**: Denotes "for all" and is used to express that a statement is true for every element in the domain. Example: $\forall x, P(x)$ means "for all x, P(x) is true."
    - **Existential quantifier (∃)**: Denotes "there exists" and is used to state that there is at least one element in the domain for which a statement is true. Example: $\exists x, P(x)$ means "there exists an x such that P(x) is true."

6. **Logical Connectives**: These are the same as in propositional logic:
    - **AND (∧)**
    - **OR (∨)**
    - **NOT (¬)**
    - **IMPLIES (→)**
    - **IFF (↔)** (if and only if)

Syntax of First Order Logic

A well-formed formula (WFF) in FOL is constructed by combining predicates, variables, constants, quantifiers, and logical connectives according to the rules of syntax. Here's an example of a syntactically valid formula in FOL:

- ∀x (Human(x) → Mortal(x))

This formula states that "for all x, if x is human, then x is mortal."

Semantics of First Order Logic

The **semantics** of FOL describe how to interpret the formulas and how to assign truth values to them. A formula is evaluated with respect to a **domain** (the set of all objects under consideration) and an **interpretation** (which assigns meaning to constants, functions, and predicates).

Example 1: Basic Relationships

Let's consider a simple domain of people, with the following constants:

- John, Mary, Peter (constants representing individuals)

Predicates:

- Parent(x, y) (x is a parent of y)
- Likes(x, y) (x likes y)

Example FOL statements:

1. **John is the parent of Peter**:
   Parent(John, Peter)
2. **Mary likes John**:
   Likes(Mary, John)

Example 2: Universal and Existential Quantifiers

1. **"Everyone likes pizza"**:
   $\forall x$ (Person(x) $\rightarrow$ Likes(x, Pizza))
   This means that for every person x, x likes pizza.

2. **"There exists someone who likes John"**:
   $\exists x$ (Person(x) $\land$ Likes(x, John))
   This means that there is at least one person who likes John.

Example 3: Complex Statements with Multiple Predicates

Let's add more predicates:

- Teacher(x) (x is a teacher)
- Student(x) (x is a student)
- Teaches(x, y) (x teaches y)

Now, we can express more complex statements:

1. **"Every teacher teaches some student"**:
   $\forall x$ (Teacher(x) $\rightarrow$ $\exists y$ (Student(y) $\land$ Teaches(x, y)))
   This means that for every x who is a teacher, there exists a y who is a student and x teaches y.
2. **"There is a student who is taught by every teacher"**:
   $\exists y$ (Student(y) $\land$ $\forall x$ (Teacher(x) $\rightarrow$ Teaches(x, y)))
   This means that there exists a student y who is taught by every teacher x.

Example 4: Logical Connectives in FOL

Let's consider a domain where we have two predicates:

- Rich(x) (x is rich)
- Happy(x) (x is happy)

1. **"If John is rich, then John is happy"**:
   Rich(John) $\rightarrow$ Happy(John)
2. **"John is rich or happy"**:
   Rich(John) $\lor$ Happy(John)
3. **"John is not rich"**:
   $\neg$Rich(John)
4. **"John is rich if and only if he is happy"**:
   Rich(John) $\leftrightarrow$ Happy(John)
   This means John is rich if and only if he is happy, i.e., John is rich precisely when he is happy, and vice versa.

Applications of First Order Logic

1. **Artificial Intelligence**: FOL is used in knowledge representation and reasoning. Systems like **Prolog** use FOL for representing facts and rules, allowing machines to reason about the world.
2. **Databases**: FOL underpins the relational model in databases. Queries in SQL can be seen as expressions in a form of first-order logic.
3. **Automated Theorem Proving**: FOL is the foundation of many formal methods used in verification of software and hardware systems.
4. **Natural Language Processing**: FOL helps in building systems that can understand and infer information from natural language texts.

Rules of Inference in First Order Logic (FOL)

Inference rules are the logical tools that allow us to derive conclusions from premises. In First Order Logic (FOL), inference rules help us deduce new facts or theorems from known facts using logical reasoning. FOL extends propositional logic by adding quantifiers and predicates, allowing more expressive reasoning about objects, their properties, and relationships.

1. Universal Instantiation (UI)

Rule: If a statement is true for all elements in a domain, then it is true for any specific element of that domain.
Notation:

- From $\forall x P(x)$
- Infer $P(a)$ for any specific individual a.

Example:

- Let $P(x)$: "x is a teacher."
- Suppose $\forall x(Teacher(x) \rightarrow Likes(x,Mathematics))$ (All teachers like Mathematics).
- From this, we can conclude that if Ramesh is a teacher, then $Likes(Ramesh,Mathematics)$ holds.

2. Existential Instantiation (EI)

Rule: If there exists at least one element in the domain for which a statement is true, then we can instantiate that statement with a specific element.

Notation:

- From ∃xP(x)
- Infer P(a)P(a)P(a) for some specific individual a.

Example:

- Let Q(x): "x is a student."
- Suppose ∃x(Student(x)∧Likes(x,Cricket)) (There exists a student who likes cricket).
- We can conclude that there is some specific student, say Priya, such that Student(Priya)∧Likes(Priya,Cricket).

3. Universal Generalization (UG)

Rule: If a statement is true for an arbitrary element of a domain, then it is true for all elements of that domain.

Notation:

- From P(a) for arbitrary a,
- Infer ∀xP(x).

Example:

- Let R(x): "x is an Indian citizen."
- If we show that R(Anjali) (Anjali is an Indian citizen) is true for an arbitrary individual, we can infer ∀xR(x) (All individuals are Indian citizens).

4. Modus Ponens (MP)

Rule: If a conditional statement is true and its antecedent is true, then its consequent is also true.

Notation:

- From P→Q and P
- Infer Q.

Example:

- Let P: "Ravi is a student."
- Let Q: "Ravi will pass the exam."
- If we know P→Q (If Ravi is a student, then he will pass the exam) and P (Ravi is a student), we can conclude Q (Ravi will pass the exam).

5. Modus Tollens (MT)

Rule: If a conditional statement is true and its consequent is false, then its antecedent is also false.

Notation:

- From P→Q and ¬Q

Infer ¬P.

Example:

- Let P: "Ram is a doctor."
- Let Q: "Ram is knowledgeable."
- If we have P→Q(If Ram is a doctor, then he is knowledgeable) and we know ¬Q (Ram is not knowledgeable), we can conclude ¬P (Ram is not a doctor).

6. Disjunctive Syllogism (DS)

Rule: If a disjunction is true and one of its disjuncts is false, then the other disjunct must be true.

Notation:

- From P∨Q and ¬P
- Infer Q.

Example:

- Let P: "Rohan is attending the festival."
- Let Q: "Sita is attending the festival."
- If we know P∨Q (Either Rohan or Sita is attending) and ¬P (Rohan is not attending), we can conclude Q(Sita is attending the festival).

7. Hypothetical Syllogism (HS)

Rule: If two conditional statements are true, then we can infer a new conditional statement.

Notation:

- From P→ Q and Q→ R,
- Infer P→R.

Example:

- Let P: "Karan is studying."
- Let Q: "Karan will pass the exam."
- Let RR: "Karan will get a job."
- If we have P→Q (If Karan studies, then he will pass) and Q→R (If Karan passes, then he will get a job), we can conclude P→R (If Karan studies, he will get a job).

8. Constructive Dilemma (CD)

Rule: Given two implications and a disjunction, we can conclude another disjunction.

Notation:

- From (P→Q)∧(R→S) and P∨R,
- Infer Q∨S.

Example:

- Let P: "Meena is cooking."

- Let Q: "Meena is preparing dinner."
- Let R: "Rahul is studying."
- Let S: "Rahul is learning."
- If we know (P→Q)∧(R→S) (If Meena cooks, she prepares dinner, and if Rahul studies, he learns) and P∨R (Either Meena is cooking or Rahul is studying), we can conclude Q∨S (Either Meena is preparing dinner or Rahul is learning).

9. Addition

Rule: From a single statement, you can derive a disjunction (or statement) by adding any proposition.
Notation:

- From: P
- You can infer: P ∨ Q (for any proposition Q)

Example:

- Premise: LovesCricket(Arjun)
- Conclusion: LovesCricket(Arjun) ∨ LovesFootball(Arjun)
  (Arjun loves cricket or Arjun loves football)

10. Conjunction
Rule: You can combine two statements into a conjunction (and statement) if both are true.
Format:

- From: P and Q
- You can infer: P ∧ Q

Example:

- Premise 1: Indian(Arjun)
- Premise 2: LovesCricket(Arjun)
- Conclusion: Indian(Arjun) ∧ LovesCricket(Arjun)
  (Arjun is Indian and Arjun loves cricket)

11. Simplification
Rule: If you have a conjunction, you can simplify it to derive either of the individual statements.
Format:

- From: P ∧ Q
- You can infer: P or Q

Example:

- Premise: Indian(Arjun) ∧ LovesCricket(Arjun)
- Conclusion 1: Indian(Arjun)
  (Arjun is Indian)

- Conclusion 2: LovesCricket(Arjun)
  (Arjun loves cricket)

12. Destructive Dilemma
Rule: A destructive dilemma allows you to infer a conclusion based on two conditional statements and a negation of their consequents.
Format:

- From:
  1. P→ Q (If P, then Q)
  2. R→ S (If R, then S)
  3. ¬Q (Not Q)
  4. ¬S (Not S)
- You can infer: ¬P∨¬R (Either not P or not R)

Example:

1. Premise 1: If Arjun is a student, then he studies hard.
   (Student(Arjun)→StudiesHard(Arjun))
2. Premise 2: If Priya is a student, then she studies hard.
   (Student(Priya)→StudiesHard(Priya))
3. Premise 3: Arjun does not study hard.
   ¬StudiesHard(Arjun)
4. Premise 4: Priya does not study hard.
   ¬StudiesHard(Priya)

Conclusion: Either Arjun is not a student or Priya is not a student.
¬Student(Arjun)∨¬Student(Priya)

Differentiate Between FOL and Prepositional Logic

1. Expressiveness

- Propositional Logic: Deals with simple propositions that can be either true or false. Each proposition is treated as an atomic unit without further structure.
- First-Order Logic: Extends propositional logic by introducing quantifiers and predicates, allowing for the expression of more complex statements about objects and their relationships.

2. Components

- Propositional Logic: Consists of propositions (e.g., P, Q) and logical connectives (AND, OR, NOT).

- First-Order Logic: Includes predicates (which represent properties of objects), constants, variables, functions, and quantifiers (∀ for "for all," ∃ for "there exists").

3. Quantifiers

- Propositional Logic: Lacks quantifiers; it cannot express statements about "all" or "some."
- First-Order Logic: Uses quantifiers to make generalizations about properties of objects (e.g., "All humans are mortal" can be expressed as $\forall x\ (\text{Human}(x) \rightarrow \text{Mortal}(x))$).

4. Domain of Discourse

- Propositional Logic: Does not consider any domain of objects; it simply evaluates the truth of propositions.
- First-Order Logic: Operates over a domain of objects, allowing for statements about these objects.

5. Inference

- Propositional Logic: Inference rules (like modus ponens) are based on the truth values of entire propositions.
- First-Order Logic: Inference can be more complex due to the use of predicates and quantifiers, allowing for more powerful reasoning.

6. **Complexity**

- Propositional Logic: Generally simpler and easier to work with, especially for automated reasoning.
- First-Order Logic: More expressive but also more complex , which can make automated reasoning more challenging.

7. **Examples**

- Propositional Logic: "It is raining" (P).
- First-Order Logic: "For every person, if they are a teacher, then they are knowledgeable" ($\forall x\ (\text{Teacher}(x) \rightarrow \text{Knowledgeable}(x))$).

Unification

Unification is a process in which two terms, which may include variables, constants, and functions, are made identical by finding a substitution for the variables. A substitution is a mapping of variables to terms. For instance, if we have a variable (X) and we substitute it with a constant (a), the variable (X) is unified with (a).

Basic Terminology -

- Term: A term can be a constant, a variable, or a compound term (a function with arguments).
  Substitution: A mapping from variables to terms.
- Unifier: A substitution that, when applied to two terms, makes them identical.
- Most General Unifier (MGU): The simplest unifier that can be used to unify two terms, meaning that any other unifier can be derived from the MGU by further substitution

Unification Algorithm -

The unification algorithm attempts to find the MGU for two terms. The algorithm involves recursively applying substitutions until the terms become identical or a conflict is found that prevents unification.

Steps in the Unification Algorithm:

1. Initialization: Start with the two terms you want to unify.
2. Decompose Compound Terms: If the terms are compound (i.e., functions with arguments), break them down into their constituent parts.
3. Check for Conflicts: If a variable is being unified with a term that contains that variable (occurs check), unification fails.
4. Apply Substitutions: Continuously apply the substitutions and simplify the terms until they are identical or no further simplification is possible.

Example 1:

Suppose we have the following expressions:

Expression 1: f(a, X, g(Y))
Expression 2: f(Z, b, g(h))
To unify these expressions, we need to find a substitution that makes them equal. The unification process involves matching corresponding parts and finding a set of variable assignments that satisfy both expressions:

Match f(a, X, g(Y)) with f(Z, b, g(h)):

X unifies with b (X/b)
Y unifies with h (Y/h)
Z unifies with a (Z/a)

The resulting substitution is: {X/b, Y/h, Z/a}. Applying this substitution to both expressions gives us:

f(a, b, g(h)) = f(a, b, g(h))
The expressions are now unified.

<u>Example 2:</u>

Let's consider a more complex example involving predicates and quantifiers in first-order logic:

Expression 1: ∀x P(x, f(Y))
Expression 2: P(Z, f(a))

To unify these expressions, we need to find a substitution that makes them equal. The unification process involves matching the quantifiers and predicates and finding a set of variable assignments:

Match ∀x P(x, f(Y)) with P(Z, f(a)):
x unifies with Z (x/Z)
Y unifies with a (Y/a)

The resulting substitution is: {x/Z, Y/a}. Applying this substitution to both expressions gives us:

∀Z P(Z, f(a)) = P(Z, f(a))
The expressions are now unified.

Applications in AI:

1. Logic Programming (Prolog):
- Unification is the core mechanism for pattern matching in Prolog. When a query is made, Prolog uses unification to match the query with facts and rules in the database to infer new information or find solutions.

2. Theorem Proving:
- Automated theorem provers use unification to match premises with conclusions of rules, thereby deriving new statements and ultimately proving or disproving theorems.

3. Natural Language Processing (NLP):
- In NLP, unification is used for parsing and understanding sentences. Feature structures representing grammatical properties are unified to check for agreement and syntactic correctness.

4. Type Inference in Programming Languages:
- Unification is used in type inference algorithms to determine the type of expressions in statically typed languages like Haskell and ML.

Lifting

Lifting refers to the process of extending a function or predicate to operate over a broader set of structures, often involving the transformation of arguments or the inclusion of quantifiers. It allows predicates or functions to take other predicates or functions as inputs, effectively elevating them to a higher level of abstraction.

Key Points:

1. Higher-Order Logic: Lifting is often associated with higher-order logic, where functions can accept other functions or predicates as arguments.
2. Expressiveness: Lifting increases the expressiveness of logical systems, enabling more complex reasoning about relationships and properties.
3. Quantification: It often involves quantifying over predicates or functions, allowing statements about properties that hold for a set of objects.

Importance of Lifting

- Expressive Power: Lifting enables the representation of statements about entire sets of objects, not just individual ones. This can be crucial in many applications like model checking, type theory, and programming language semantics.
- Generalization: It allows for the generalization of properties, enabling reasoning about classes of objects rather than specific instances.

Example

Let’s illustrate lifting with a concrete example involving predicates and quantification.

Example Scenario:

1. **Base Predicate:** Consider the predicate P(x), which states "x is a cat."
2. **Lifted Predicate:** We want to create a lifted predicate Q that asserts something about all cats.

   We can define Q in terms of P:

   $$Q(P) \equiv \forall x (P(x) \implies R(x))$$

Here, R(x) might represent "x is a pet."

Thus, Q(P) states: "For all x, if x is a cat (i.e., P(x) holds), then x is a pet (i.e., R(x) holds)."

3. **Interpretation:** The lifted predicate Q(P) generalizes the specific property of being a cat to a broader claim about the relationship between being a cat and being a pet.
4. **Using Lifting:**
   - We can check if this lifted predicate holds for specific instances. For example, if we know that:
     - P(Whiskers) is true (Whiskers is a cat).
     - We assume R(Whiskers) is true (Whiskers is a pet).
   - In this case, since $P(\text{Whiskers}) \implies R(\text{Whiskers})$ holds, we can conclude that Q(P) is satisfied for this instance.

Forward chaining

Forward chaining is a data-driven approach that starts with the available facts and applies inference rules to derive new facts until a goal is reached or no new facts can be generated.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Mechanism:

1. Start with a set of known facts.

2. Apply inference rules (if-then rules) to derive new facts.
3. Repeat the process until no new facts can be derived or the goal is reached.

Example 1:

Consider the following facts and rules:

1. **Facts:**
    - F1:Bird(Tweety)
    - F2:CanFly(Tweety)
2. **Rules:**
    - R1:∀x(Bird(x) $\Longrightarrow$ CanFly(x)) (If x is a bird, then x can fly.)
    - R2:∀x(CanFly(x) $\Longrightarrow$ Pet(x)) (If x can fly, then x is a pet.)

Forward Chaining Process:

1. Start with F1 and F2.
2. Apply R1:
    - From Bird(Tweety), we conclude CanFly(Tweety). (Already known)
3. Apply R2:
    - From CanFly(Tweety), we derive Pet(Tweety).

Result:

- New fact derived: Pet(Tweety)

Forward chaining has successfully derived new information using the available facts and rules.

Example 2:

Let's consider a simple medical diagnosis system with a set of rules for common illnesses.

Facts:

1. F1:HasFever(John)
2. F2:HasCough(John)

Rules:

1. R1:∀x(HasFever(x)∧HasCough(x) $\Longrightarrow$ HasFlu(x))
2. R2:∀x(HasFlu(x) $\Longrightarrow$ NeedsRest(x)

Forward Chaining Process:

1. Start with facts F1and F2.
    - F1:HasFever(John)
    - F2:HasCough(John)
2. Apply Rule R1 (If someone has a fever and a cough, they have the flu):
    - Since HasFever(John)∧HasCough(John) is true, we can derive:
    - HasFlu(John).
3. Apply Rule R2 (If someone has the flu, they need rest):
    - Since HasFlu(John) is true, we can derive:
    - NeedsRest(John).

New Facts:

- HasFlu(John)
- NeedsRest(John)

In this case, using forward chaining, we were able to infer that John needs rest based on his symptoms.

Backward Chaining

Backward chaining is an inference method used in logic programming and artificial intelligence where reasoning starts with a goal or query and works backward to infer facts or conditions that must hold true to satisfy the goal. The system works by recursively breaking down the goal into sub-goals, looking for rules that support the goal, and checking if the premises of those rules are already known facts or need further derivation.
Unlike forward chaining (where facts are used to derive new facts), backward chaining starts with the desired conclusion and works back to determine whether the facts support that conclusion.

Mechanism:

1. **Start with the Goal**: Identify the target or query you want to prove (the goal).
2. **Find Applicable Rules**: Search for inference rules that can lead to the goal (conclusion) if certain conditions (premises) hold.
3. **Check Premises**: For each rule, check if the premises (antecedents) are facts. If not, treat each premise as a new sub-goal.
4. **Recursion**: Repeat the process for each sub-goal, until all premises are proven true (or false).
5. **Success/Failure**: If all premises are satisfied, the original goal is true. If any premise cannot be satisfied, the goal is false.

<u>Example 1:</u>

We want to determine if John has the flu.

Facts:

1. HasFever(John)
2. HasCough(John)

Inference Rules:

1. R1:$\forall x(\text{HasFever}(x) \land \text{HasCough}(x) \implies \text{HasFlu}(x))$ *(If a person has a fever and a cough, they have the flu.)*

Goal: Prove HasFlu(John).

Backward Chaining Steps:

1. **Start with the Goal:** We want to prove HasFlu(John).
2. **Find Applicable Rule:** The rule R1 suggests that HasFlu(x) can be inferred if both HasFever(x)∧HasCough(x) are true. So, we break this down into two sub-goals:
    - HasFever(John)
    - HasCough(John)
3. **Sub-goal 1:** Prove HasFever(John).
    - The fact HasFever(John) is directly given, so it is true.
4. **Sub-goal 2:** Prove HasCough(John).
    - The fact HasCough(John) is also directly given, so it is true.
5. **Conclusion:** Since both sub-goals are proven true (i.e., HasFever(John) and HasCough(John)), we can apply the rule R1 to infer that:
    - HasFlu(John) is true.

Thus, backward chaining confirms that John has the flu.

<u>**Example 2:**</u>

We are trying to determine if "Tweety can fly."

Facts:

1. Bird(Tweety)-Tweety is a bird.
2. Penguin(Tweety)-Tweety is a penguin.

Inference Rules:

1. R1:∀x(Bird(x)∧¬Penguin(x) ⟹ CanFly(x)) *(If x is a bird and not a penguin, then x can fly.)*
2. R2:∀x(Penguin(x) ⟹ ¬CanFly(x)) *(If x is a penguin, then x cannot fly.)*

Goal: Prove CanFly(Tweety) — Can Tweety fly?

Backward Chaining Steps:

1. **Start with the Goal**: We want to prove CanFly(Tweety).
2. **Find Applicable Rule**: There are two rules related to flying:
    - R1:Bird(x)∧¬Penguin(x) ⟹ CanFly(x)
    - R2:Penguin(x) ⟹ ¬CanFly(x)

    Since we know that Tweety is a penguin (Penguin(Tweety)), rule R2 seems relevant to this situation. Let's try to use that rule to prove or disprove the goal.
3. **Apply Rule R2**: The rule R2 tells us that if Tweety is a penguin, then Tweety cannot fly. So we break this down into a sub-goal:
    - **Sub-goal**: Prove Penguin(Tweety).
4. **Sub-goal 1**: Prove Penguin(Tweety).

   - The fact Penguin(Tweety) is directly given, so it is true.
5. **Conclusion from Rule R2**: Since Penguin(Tweety) is true, we can apply the inference rule R2, which tells us that:
   - ¬CanFly(Tweety) — Tweety cannot fly.

Final Answer:

- Backward chaining concludes that **Tweety cannot fly** based on the fact that Tweety is a penguin.

Resolution in FOL

Resolution is a fundamental inference rule in Propositional Logic and First-Order Logic (FOL) used for theorem proving. It allows us to deduce new clauses from existing clauses until we either prove or disprove a given goal. In FOL, resolution is applied to clauses in Conjunctive Normal Form (CNF) and is used in automated reasoning systems to prove statements by contradiction.

Resolution works by unifying and combining two clauses that contain complementary literals. If the goal of resolution is to show that a particular statement is true, the process continues until a contradiction (the empty clause) is derived, proving that the negation of the goal cannot be true.

Steps in the Resolution Process:

1. **Convert to Clause Form (CNF):** Convert all given facts and the negation of the goal into conjunctive normal form (CNF), which is a conjunction of disjunctions of literals.
2. **Apply Unification:** Find pairs of clauses with complementary literals (i.e., one clause contains a literal, and another contains its negation).
3. **Resolve Clauses:** Eliminate the complementary literals and combine the remaining literals to form a new clause.
4. **Repeat:** Continue resolving clauses until either an empty clause (contradiction) is derived or no more resolutions are possible.
5. **Derive Contradiction:** If the empty clause is derived, the original statement (goal) is proven true (by contradiction).

<u>Example 1:</u>

Prove that Tweety can fly, given the following knowledge base:

Facts (Knowledge Base):

1. Birds can fly:
   $\forall x(\text{Bird}(x) \Longrightarrow \text{CanFly}(x))$

2. Tweety is a bird:
   Bird(Tweety)

Goal: Prove CanFly(Tweety) — Tweety can fly.

Step 1: Negate the Goal:

Negate the goal: ¬CanFly(Tweety)

Step 2: Convert to CNF:

1. ∀x(Bird(x) ⟹ CanFly(x)) is equivalent to: ¬Bird(x)∨CanFly(x) (Clause 1)
2. Bird(Tweety) is already in CNF:
   Bird(Tweety) (Clause 2)
3. The negated goal is ¬CanFly(Tweety):
   ¬CanFly(Tweety) (Clause 3)

Step 3: Apply Resolution:

- **Resolve Clause 1** and **Clause 2**:
  - Clause 1: ¬Bird(x)∨CanFly(x)
  - Clause 2: Bird(Tweety)

  Unify x with "Tweety" in Clause 1, giving: ¬Bird(Tweety)∨CanFly(Tweety)
  Resolve ¬Bird(Tweety) with Bird(Tweety) from Clause 2, leaving: CanFly(Tweety)
- **Resolve the result with Clause 3**:
  - Result: CanFly(Tweety)
  - Clause 3: ¬CanFly(Tweety)

  Resolving CanFly(Tweety) with ¬CanFly(Tweety) results in the **empty clause**.

Step 4: Conclusion:

Since the empty clause is derived, we have proven that **Tweety can fly**.

Practice Questions

1. **Translate the following English sentences into FOL:**
   - a) Every student in the class loves mathematics.
   - b) There is a person who knows every language.
   - c) All cars in the garage are red.
   - d) There is a cat that chases every mouse.
   - e) No student in the class is taller than the teacher.
2. **Given the domain of "people," with the predicates Parent(x, y) (x is the parent of y), and Likes(x, y) (x likes y), express the following in FOL:**
   - a) Every parent loves their child.
   - b) There exists a person who loves everyone.
   - c) If someone is a parent, they like their child.
   - d) No one likes everyone.
3. **Consider the predicates:**
   - Loves(x, y) – x loves y
   - Animal(x) – x is an animal
   - Person(x) – x is a person
   - Owns(x, y) – x owns y

Write FOL expressions for the following statements:
   - a) Every person owns a dog.
   - b) Some animals are not owned by anyone.
   - c) If a person owns a pet, they love that pet.
   - d) There is a person who loves every animal.

4. **Identify the error in the following FOL statements (if any) and rewrite them correctly:**
   - a) $\forall x\ (Student(x) \lor Teaches(John, x))$ (John teaches every student)
   - b) $\exists x\ (Teacher(x) \rightarrow \forall y\ (Student(y) \land Teaches(x, y)))$ (There is a teacher who teaches all students)
   - c) $\forall x\ (Person(x) \land Likes(x, Pizza) \rightarrow \exists y\ Likes(y, Pizza))$ (If a person likes pizza, someone likes pizza)
5. **Write FOL expressions for these statements:**
   - a) Every natural number has a successor.
   - b) There is a smallest natural number.
   - c) For every even number, there exists another number that is half of it.
   - d) The sum of two odd numbers is even.
6. **Translate the following into natural language:**
   - a) $\forall x\ (Animal(x) \rightarrow \exists y\ (Person(y) \land Owns(y, x)))$
   - b) $\exists x\ (Student(x) \land \forall y\ (Course(y) \rightarrow Enrolled(x, y)))$

- c) ¬∃x (Person(x) ∧ ¬Likes(x, Music))
- d) ∀x (Doctor(x) → ∀y (Patient(y) → Treats(x, y)))

7. **Consider the predicates:**
   - Richer(x, y) – x is richer than y
   - Brother(x, y) – x is the brother of y

Express the following statements in FOL:

- a) There exists a person who is richer than all their brothers.
- b) Everyone has a brother who is richer than them.
- c) No one is richer than themselves.

8. **Domain: Planets, Spacecraft**
   - Let Planet(x) represent that x is a planet, Spacecraft(x) represent that x is a spacecraft, and LandedOn(x, y) represent that x landed on planet y. Write FOL expressions for the following:
   - a) Every spacecraft has landed on a planet.
   - b) Some planets have not been visited by any spacecraft.
   - c) There exists a spacecraft that has landed on every planet.
9. **Given the predicates:**
   - Male(x), Female(x) (x is male, x is female)
   - Parent(x, y) (x is a parent of y)
   - Ancestor(x, y) (x is an ancestor of y)

Write the following in FOL:

- a) Every parent is an ancestor.
- b) If x is a male, and y is a child of x, then x is the father of y.
- c) A person’s ancestor includes all their parents and grandparents.

Solution

1. **Translate the following English sentences into FOL:**

   a) Every student in the class loves mathematics.

   $\forall x\,(Student(x) \land InClass(x) \rightarrow Loves(x,Mathematics))$

   b) There is a person who knows every language.

   $\exists x\,(Person(x) \land \forall y\,(Language(y) \rightarrow Knows(x,y)))$

   c) All cars in the garage are red.

   $\forall x\,(Car(x) \land InGarage(x) \rightarrow Red(x))$

   d) There is a cat that chases every mouse.

   $\exists x\,(Cat(x) \land \forall y\,(Mouse(y) \rightarrow Chases(x,y)))$

   e) No student in the class is taller than the teacher.

   $\forall x\,(Student(x) \land InClass(x) \rightarrow \neg Taller(x,Teacher))$

2. **Given the domain of "people," with the predicates Parent(x, y) (x is the parent of y), and Likes(x, y) (x likes y), express the following in FOL:**

   a) Every parent loves their child.

   $\forall x \forall y(Parent(x,y) \rightarrow Likes(x,y))$

   b) There exists a person who loves everyone.

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

   c) If someone is a parent, they like their child.

   $\forall x \forall y(Parent(x,y) \rightarrow Likes(x,y))$

   d) No one likes everyone.

   $\forall x \exists y(\neg Likes(x,y))$

3. **Consider the predicates:**
   - Loves(x, y) – x loves y
   - Animal(x) – x is an animal
   - Person(x) – x is a person
   - Owns(x, y) – x owns y

Write FOL expressions for the following statements:

a) Every person owns a dog.

$$\forall x(Person(x) \rightarrow \exists y(Animal(y) \land Dog(y) \land Owns(x,y)))$$

b) Some animals are not owned by anyone.

$$\exists x(Animal(x) \land \forall y(Person(y) \rightarrow \neg Owns(y,x)))$$

c) If a person owns a pet, they love that pet.

$$\forall x \forall y((Person(x) \land Owns(x,y) \land Animal(y)) \rightarrow Loves(x,y))$$

d) There is a person who loves every animal.

$$\exists x(Person(x) \land \forall y(Animal(y) \rightarrow Loves(x,y)))$$

4. **Identify the error in the following FOL statements (if any) and rewrite them correctly:**

a) $\forall x$ (Student(x) $\lor$ Teaches(John, x)) **(John teaches every student)**

Error: The use of the disjunction ($\lor$) is incorrect because it doesn't properly capture the idea that John teaches every student. Instead, it says that for every x, either x is a student or John teaches x, which is not what we want.

$$\forall x(Student(x) \rightarrow Teaches(John,x))$$

Explanation: This correctly states that for every x, if x is a student, then John teaches x.

b) $\exists x$ (Teacher(x) $\rightarrow$ $\forall y$ (Student(y) $\land$ Teaches(x, y))) **(There is a teacher who teaches all students)**

Error: The implication ($\rightarrow$) inside the existential quantifier is unnecessary and changes the meaning of the statement. The implication suggests that if someone is a teacher, then they teach all students, but this is not equivalent to saying there exists a teacher who teaches all students.

$$\exists x(Teacher(x) \land \forall y(Student(y) \rightarrow Teaches(x,y)))$$

Explanation: This correctly states that there exists a teacher x such that for every y, if y is a student, then x teaches y.

c) ∀x (Person(x) ∧ Likes(x, Pizza) → ∃y Likes(y, Pizza)) **(If a person likes pizza, someone likes pizza)**

Error: This statement is valid as written but somewhat redundant. If a person likes pizza, it is trivially true that someone likes pizza because that person (x) is the "someone." However, the use of the existential quantifier (∃) for y adds unnecessary complexity. The existential quantifier for y is redundant since xxx already satisfies the condition.

∀x(Person(x)∧Likes(x,Pizza)→Likes(x,Pizza))

Explanation: This simplifies the statement, acknowledging that if a person x likes pizza, then x likes pizza, which is trivially true.

5. **Translate the following into natural language:**

a) ∀x (Animal(x) → ∃y (Person(y) ∧ Owns(y, x)))

"Every animal is owned by some person."

b) ∃x (Student(x) ∧ ∀y (Course(y) → Enrolled(x, y)))

"There is a student who is enrolled in every course."

c) ¬∃x (Person(x) ∧ ¬Likes(x, Music))

"There is no person who does not like music" or "Everyone likes music."

d) ∀x (Doctor(x) → ∀y (Patient(y) → Treats(x, y)))

"Every doctor treats every patient."

REST QUESTION --- PRACTICE YOURSELF