First Order Logic in Knowledge Representation and Reasoning

Introduction

- In the topic of Propositional logic, we have seen that how to represent statements using propositional logic.
- But unfortunately, in propositional logic, we can only represent the facts, which are either true or false.

- PL is not sufficient to represent the complex sentences or natural language statements.

Introduction

- 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 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:

- 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, ...

First-Order logic:

- As a natural language, first-order logic also has two main parts:
  - **Syntax**
  - **Semantics**

Syntax of First-Order logic:

- The syntax of FOL determines which collection of symbols is a logical expression in first-order logic.
- The basic syntactic elements of first-order logic are symbols.
- We write statements in short-hand notation in FOL.

Basic Elements of First-order logic:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	$\land$, $\lor$, $\neg$, $\Rightarrow$, $\Leftrightarrow$
Equality	==
Quantifier	$\forall$, $\exists$

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:

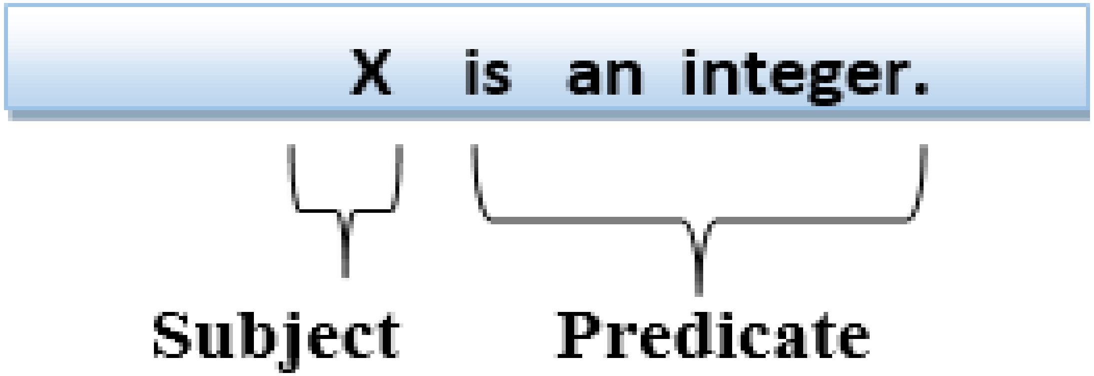
- Complex sentences are made by combining atomic sentences using connectives.

First-order logic 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.

Complex Sentences:

- **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.

Quantifiers in First-order logic:

There are two types of quantifier:

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

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 "→".

Universal Quantifier:

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 as below:

Example:

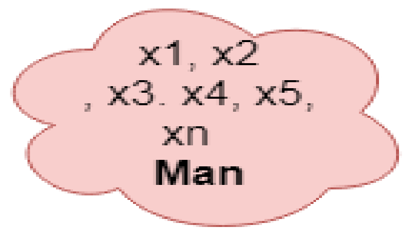
- x1 drinks coffee

  ∧
- x2 drinks

  ∧
- x3 drinks milk

  ∧
- .
- .

  ∧
- xn drinks milk



Universe of Discourse

So in shorthand notation, we can write it as :

∀x man(x) → drink (x, coffee).

It will be read as: There are all x where x is a man who drink coffee.

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 (∧).

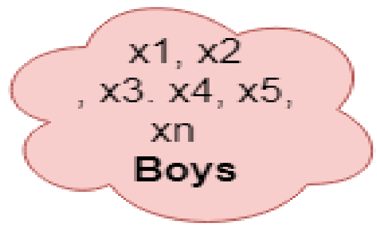
Existential Quantifier:

- 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.

- **x1 is intelligent**
  V
- **x2 is intelligentV**
- **x3 is intelligent**
  V
- **.**
- **.**
  V
- **xn is intelligent**



Universe of Discourse

So in short-hand notation, we can write it as:

∃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/Properties of Quantifiers:

- The main connective for universal quantifier $\forall$ is implication $\rightarrow$.
- The main connective for existential quantifier $\exists$ is and $\wedge$.
- In universal quantifier, $\forall x \forall y$ is similar to $\forall y \forall x$.
- In Existential quantifier, $\exists x \exists y$ is similar to $\exists y \exists x$.
- $\exists x \forall y$ is not similar to $\forall y \exists x$.

Some Examples of FOL using Quantifier:

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.

   $$\forall x \; \mathbf{bird(x)} \rightarrow \mathbf{fly(x).}$$

2. **2. Every man respects his parent.**
   In this question, the predicate is "**respect(x, y),"** **where x=man, and y= parent**.

Some Examples of FOL using Quantifier:

- Since there is every man so will use ∀, and it will be represented as follows:

  $\forall x$ 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**:

$\exists x$ boys(x) → play(x, cricket).

Some Examples of FOL using Quantifier:

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:
   $\neg\forall (x)$ [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 which are given below:

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

Example: $\forall x \exists (y)[P(x, y, z)]$, where z is a free variable.

Free and Bound Variables:

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

Example: $\forall x$ [A (x) B(y)], here x and y are the bound variables.

Knowledge Engineering in FOPL

- Knowledge engineering is the process where a knowledge engineer investigates a specific domain, learn the important concepts regarding that domain, and creates the formal representation of the objects and relations in that domain.

Knowledge Engineering Process

1) Identify the task
2) Assemble the relevant knowledge
3) Decide on a vocabulary of constants, predicates, and functions:
4) Encode general knowledge about the domain
5) Encode description of the specific problem instance
6) Raise queries to the inference procedure and get answers
7) Debug the knowledge base

Identify the task

A knowledge engineer should be able to identify the task by asking a few questions like:

- ❑Do the knowledge base will support?
- ❑What kinds of facts will be available for each specific problem?
- ❑The task will identify the knowledge requirement needed to connect the problem instance with the answers.

Assemble the relevant knowledge:

- A knowledge engineer should be an expert in the domain. If not, he should work with the real experts to extract their knowledge.
- This concept is known as **Knowledge Acquisition**.

Note: Here, we do not represent the knowledge formally. But to understand the scope of the knowledge base and also to understand the working of the domain.

Decide on a vocabulary of constants, predicates, and functions:

- Translating important domain-level concepts into logical level concepts.
- **Here, the knowledge engineer asks questions like:**
- What are the elements which should be represented as objects?
- What functions should be chosen?
- After satisfying all the choices, the vocabulary is decided. It is known as the **Ontology of the domain**. Ontology determines the type of things that exists but does not determine their specific properties and interrelationships.

Encode general knowledge about the domain:

In this step, the knowledge engineer pen down the axioms for all the chosen vocabulary terms.

Note: Here, misconceptions occur between the vocabulary terms.

Encode description of the specific problem instance:

- We write the simple atomic sentences for the selected vocabulary terms. We encode the chosen problem instances.

Raise queries to the inference procedure and get answers:

- It is the testing step. We apply the inference procedure on those axioms and problem-specific facts which we want to know.

Debug the knowledge base:

- It is the last step of the knowledge engineering process where the knowledge engineer debugs all the errors.