First-Order Logic in Artificial intelligence

Propositional logic is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

a. "Some humans are intelligent", or
b. "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:

1. 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.
2. 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.
3. 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, ...

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.

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).

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

Complex Sentences:

Complex sentences are made by combining atomic sentences using connectives.

Quantifiers in First-order logic: ***"study by your own"***

Note:

1. The main connective for universal quantifier $\forall$ is implication $\rightarrow$.
2. The main connective for existential quantifier $\exists$ is and $\wedge$.

A few Examples::

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.

$\forall x\ bird(x) \rightarrow fly(x)$.

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 $\forall$, and it will be represented as follows:

$\forall x\ man(x) \rightarrow respects\ (x, parent)$.

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 $\exists$, and it will be represented as:

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

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 $\forall$ with negation, so following representation for this:

$\neg\forall (x)\ [\ student(x) \rightarrow like(x, Mathematics) \wedge like(x, Science)]$.

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)].

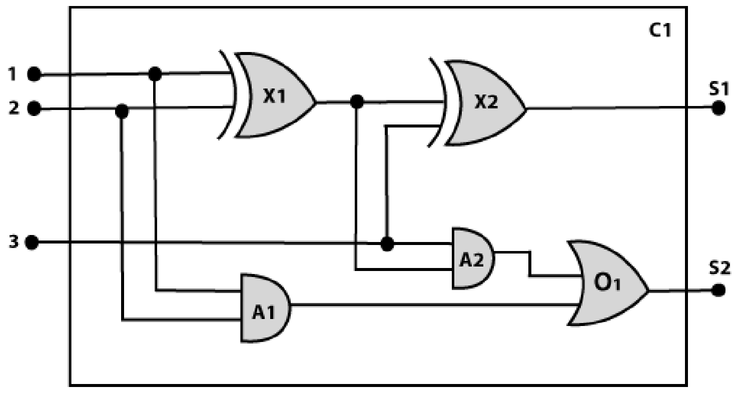
*****Understand what is **free** and **bounded** variable.

Knowledge Engineering in First-order logic:

What is knowledge-engineering?

The process of constructing a knowledge-base in first-order logic is called as knowledge- engineering. In knowledge-engineering, someone who investigates a particular domain, learns important concept of that domain, and generates a formal representation of the objects, is known as knowledge engineer.

Process:



Following are some main steps of the knowledge-engineering process. Using these steps, we will develop a knowledge base which will allow us to reason about digital circuit (One-bit full adder) which is given below

1. Identify the task:

The first step of the process is to identify the task, and for the digital circuit, there are various reasoning tasks. At the first level or highest level, we will examine the functionality of the circuit:

Does the circuit add properly?
What will be the output of gate A2, if all the inputs are high?

At the second level, we will examine the circuit structure details such as:

Which gate is connected to the first input terminal?

Does the circuit have feedback loops?

2. Assemble the relevant knowledge:

In the second step, we will assemble the relevant knowledge which is required for digital circuits. So for digital circuits, we have the following required knowledge:

Basically the knowledge about the Gates (AND, OR, NOT, XOR etc.) and how they work.

3. Vocabulary:

The next step of the process is to select functions, predicate, and constants to represent the circuits, terminals, signals, and gates. Firstly we will distinguish the gates from each other and from other objects. Each gate is represented as an object which is named by a constant, such as, Gate(X1). The functionality of each gate is determined by its type, which is taken as constants such as AND, OR, XOR, or NOT. Circuits will be identified by a predicate: Circuit (C1).

a. For the terminal, we will use predicate: Terminal(x).
b. For gate input, we will use the function In(1, X1) for denoting the first input terminal of the gate X1, and for output terminal we will use Out (1, X1).
c. The function Arity(c, i, j) is used to denote that circuit c has i no of inputs, j no of outputs.
d. The connectivity between gates can be represented by predicate Connect(Out(1, X1), Out(1, X2)), where X1 and X2 are gates.
e. We use a unary predicate On (t), which is true if the signal at a terminal is on.

4. Encode general knowledge about the domain:

few rules to remember here and deduce many more likewise.

a. If two terminals are connected then they have the same input signal, it can be represented as:
   ⇨ **∀ t1, t2 Terminal (t1) ∧ Terminal (t2) ∧ Connect (t1, t2) → Signal (t1) = Signal (2**).
b. Signal at every terminal will have either value 0 or 1, it will be represented as:
   ⇨ **∀ t Terminal (t) →Signal (t) = 1 ∨Signal (t) = 0.**
c. Connect predicates are commutative:
   ⇨ **∀ t1, t2 Connect(t1, t2) → Connect (t2, t1).**
d. Representation of types of gates:
   ⇨ **∀ g Gate(g) ∧ r = Type(g) → r = OR ∨r = AND ∨r = XOR ∨r = NOT.**
e. Output of AND gate will be zero if and only if any of its input is zero.
   ⇨ **∀ g Gate(g) ∧ Type(g) = AND →Signal (Out(1, g))= 0 ⇔ ∃n Signal (In(n, g))= 0.**
f. Output of OR gate is 1 if and only if any of its input is 1:
   ⇨ **∀ g Gate(g) ∧ Type(g) = OR → Signal (Out(1, g))= 1 ⇔ ∃n Signal (In(n, g))= 1**
g. Output of XOR gate is 1 if and only if its inputs are different:
   ⇨ **∀ g Gate(g) ∧ Type(g) = XOR → Signal (Out(1, g)) = 1 ⇔ Signal (In(1, g)) ≠ Signal (In(2, g)).**

h. Output of NOT gate is invert of its input:
   ⇨ **∀ g Gate(g) ∧ Type(g) = NOT → Signal (In(1, g)) ≠ Signal (Out(1, g)).**
i. All the gates in the above circuit have two inputs and one output (except NOT gate).
   ⇨ **∀ g Gate(g) ∧ Type(g) = NOT → Arity(g, 1, 1)**
   ∀ g Gate(g) ∧ r =Type(g) ∧ (r= AND ∨r= OR ∨r= XOR) → Arity (g, 2, 1).
j. All gates are logic circuits:
   ⇨ **∀ g Gate(g) → Circuit (g).**

5. Encode a description of the problem instance:

Now we encode problem of circuit C1, firstly we categorize the circuit and its gate components. This step is easy if ontology about the problem is already thought. This step involves the writing simple atomics sentences of instances of concepts, which is known as ontology. For the given circuit C1, we can encode the problem instance in atomic sentences as below:

Since in the circuit given above, there are two XOR, two AND, and one OR gate so atomic sentences for these gates will be:

- ⇨ For XOR gate: Type(x1)= XOR, Type(X2) = XOR
- ⇨ For AND gate: Type(A1) = AND, Type(A2)= AND
- ⇨ For OR gate: Type (O1) = OR.

And then represent the connections between all the gates.

6. Pose queries to the inference procedure and get answers:

In this step, we will find all the possible set of values of all the terminal for the adder circuit. The first query will be:

What should be the combination of input which would generate the first output of circuit C1, as 0 and a second output to be 1? And that can be represented as,

⇨ ∃ i1, i2, i3 Signal (In(1, C1))=i1 ∧ Signal (In(2, C1))=i2 ∧ Signal (In(3, C1))= i3
∧ Signal (Out(1, C1)) =0 ∧ Signal (Out(2, C1))=1

7. Debug the knowledge base:

Now we will debug the knowledge base, and this is the last step of the complete process. In this step, we will try to debug the issues of knowledge base.

In the knowledge base, we may have omitted assertions like 1 ≠ 0.

Inference in First-Order Logic:

Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences. Before understanding the FOL inference rule, let's understand some basic terminologies used in FOL.

a. **Substitution:**

Substitution is a fundamental operation performed on terms and formulas. It occurs in all inference systems in first-order logic. The substitution is complex in the presence of quantifiers in FOL. If we write F[a/x], so it refers to substitute a constant "a" in place of variable "x".

b. **Equality:**

First-Order logic does not only use predicate and terms for making atomic sentences but also uses another way, which is equality in FOL. For this, we can use equality symbols which specify that the two terms refer to the same object.

Example: Brother (John) = Smith.

As in the above example, the object referred by the Brother (John) is similar to the object referred by Smith. The equality symbol can also be used with negation to represent that two terms are not the same objects.

Example: $\neg(x=y)$ which is equivalent to $x \neq y$.

FOL inference rules for quantifier:

As propositional logic we also have inference rules in first-order logic, so following are some basic inference rules in FOL:

1. Universal Generalization
2. Universal Instantiation
3. Existential Instantiation
4. Existential introduction

1. Universal Generalization:

- Universal generalization is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as ∀ x P(x).
- It can be represented as: $\frac{P(c)}{\forall x\, P(x)}$.
- This rule can be used if we want to show that every element has a similar property.
- In this rule, x must not appear as a free variable.

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

2. Universal Instantiation:

- Universal instantiation is also called as universal elimination or UI is a valid inference rule. It can be applied multiple times to add new sentences.
- The new KB is logically equivalent to the previous KB.
- As per UI, we can infer any sentence obtained by substituting a ground term for the variable.
- The UI rule state 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: $\frac{\forall x\, P(x)}{P(c)}$

Example:1.

IF "Every person like ice-cream"=> ∀x P(x) so we can infer that
"John likes ice-cream" => P(c)
Example: 2.

"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),

So from this information, we can infer any of the following statements using Universal Instantiation:

King(John) ∧ Greedy (John) → Evil (John),

King(Richard) ∧ Greedy (Richard) → Evil (Richard),

King(Father(John)) ∧ Greedy (Father(John)) → Evil (Father(John)),

3. Existential Instantiation:

- Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.
- It can be applied only once to replace the existential sentence.
- The new KB is not logically equivalent to old KB, but it will be satisfiable if old KB was satisfiable.
- This rule states that one can infer P(c) from the formula given in the form of ∃x P(x) for a new constant symbol c.
- The restriction with this rule is that c used in the rule must be a new term for which P(c) is true.
- It can be represented as: $\frac{\exists x\, P(x)}{P(c)}$

Example:

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

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 called Skolem constant.
- ⇨ The Existential instantiation is a special case of Skolemization process.

4. Existential introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the property P.
- It can be represented as: $\frac{P(c)}{\exists x P(x)}$

Example: Let's say that,

"Priyanka got good marks in English."
"Therefore, someone got good marks in English."