First-Order Logic

22 December 2020 14:29

A logic that is sufficient for building Knowledge Based Agents

Before, we've used propositional logic as our representation language because it is one of the simplest languages that demonstrates all the important points. Unfortunately, *propositional logic has a very limited ontology*, making only the commitment that the world consists of facts. This makes it difficult to represent even something simple

FOL or *First-Order Predicate Calculus makes a stronger set of ontological commitments*. The main one is that the world consists of objects, that is things with individual identities and properties that distinguish them from other objects.

Among these objects, various relations hold. Some of these relations are functions - relations for which there is only one value for a given input. It is easy to start listing examples of objects, properties, relations and functions

- Objects - people, houses, numbers, theories, colours, baseball games, wars
- Relations - brother of, bigger than, inside of, has colour, occurred after, owns
- Properties - red, round, prime
- Functions - father of, best friend, third inning of, one more than

FOL makes *<u>no commitments to time, categories and events</u>*. A logic that tried to, would only have limited appeal as there are so many different ways of interpreting them. Thus, FOL remains neutral and gives us the freedom to describe these things in a way that is appropriate for the domain. Freedom of choice is a general characteristic of FOL

Syntax and Semantics

In propositional logic, every expression is a sentence, which represents a fact. First-Order Logic has sentences, but it also has terms, which represent objects.
Terms are built from constant symbols, variables, and function symbols.

FOL BNF

1. Sentence -> AtomicSentence | Sentence Connective Sentence | Quantifier Variable, ... Sentence | !Sentence | (Sentence)
2. AtomicSentence => Predicate(Term,...) | Term = Term
3. Term -> Function(term, ...) | Constant | Variable
4. Connective -> => | /\ | \/ | <=>
5. Quantifier -> For All | For Some
6. Constant -> A | John
7. Predicate -> Before | HasColor | Raining
8. Function -> Mother | LeftLegOf

Constant

Which object in the world is referred to by each constant symbol? Each constant symbol names exactly one object, nbut not all objects need to have names, and some can have several names. Thus, the symbol john, in one particular interpretation might refer to a specific king, but the symbol king could refer

Predicate Symbols

An interpretation specifies that a predicate symbols refers to a particular relation in the model

Brother might refer to the relation of brotherhood
A relation is defined by the set of tuples of objects that satisfy it
Thus, a predicate symbol is a symbol that points to this list of satisfying tuples

Function Symbols

- Some relations are functional - that is, any given object is related to exactly one other object by the relation.
- For example, any angle has only one number that is its cosine, and person has only oner person that is his or her father.
- In such cases, it is often more convenient to define a function symbol e.g. cosine that refers to the appropriate relation between angles and numbers.
- In the model, the mapping is just a set of n+1 tuples with a special property, namely that the last element of each tuple is the value of the function for the first n elements.
- A table of cosines is an example of this set of tuples.
- Unlike predicate symbols which hare used to state that relations hold among certain objects, function symbols are used to refer to particular objects without using their names.

Using First-Order Logic

In knowledge representation, a domain is a section of the world about which we wish to express some knowledge

One's husband is one's male spouse:

$$\forall w, h \;\; Husband(h, w) \Leftrightarrow Male(h) \land Spouse(h, w)$$

Male and female are disjoint categories:

$$\forall x \;\; Male(x) \Leftrightarrow \neg Female(x)$$

Parent and child are inverse relations:

$$\forall p, c \;\; Parent(p, c) \Leftrightarrow Child(c, p)$$

A grandparent is a parent of one's parent:

$$\forall g, c \;\; Grandparent(g, c) \Leftrightarrow \exists p \;\; Parent(g, p) \land Parent(p, c)$$

A sibling is another child of one's parents:

$$\forall x, y \;\; Sibling(x, y) \Leftrightarrow x \neq y \land \exists p \;\; Parent(p, x) \land Parent(p, y)$$