First-Order Logic: Syntax and Semantics

Alan Fern, afern@eecs.oregonstate.edu

February 28, 2020

1 Limits of Propositional Logic

Propositional logic assumes that the world or system being modeled can be described in terms of a fixed, known set of propositions. This assumption can make it awkward, or even impossible, to specify many pieces of knowledge.

For example, consider the following general statement about people.

"If a person is rich then they have a nice car."

We could encode this knowledge in propositional knowledge by creating a large set of rules, one rule for each person who is currently in the world,

$$\begin{aligned} BobIsRich &\Rightarrow BobHasNiceCar \\ JaneIsRich &\Rightarrow JaneHasNiceCar \\ PatIsRich &\Rightarrow PatHasNiceCar \\ \vdots &\ \vdots\ \ \vdots \end{aligned}$$

We see that propositional logic requires that we turn the concise and general English sentence into many statements about specific people. This is a completely impractical and unintuitive way to represent and communicate such knowledge.

As another example, consider the following general statement about natural numbers.

"If n is a natural number, then n+1 is also a natural number."

We could try to encode this statement in propositional logic as,

$$\begin{aligned} Natural1 &\Rightarrow Natural2 \\ Natural2 &\Rightarrow Natural3 \\ \vdots &\ \vdots\ \ \vdots \end{aligned}$$

However, we would need to have an infinite number of propositional formulas, one for each natural number. We could place a bound on the natural numbers we are willing to consider, but this is unsatisfactory. For example, with such a bound we would not be able to correctly answer many simple queries about the typical interpretation of natural numbers, e.g. "Does there exist a largest natural number?". Any true AI should certainly be able to reason about the natural numbers or other infinite domains, yet propositional logic does not allow for this.

2 Upgrading Propositional Logic

In both of these examples we need the ability to directly talk about objects (e.g. people or numbers) and to write down logical statements that generalize (or **quantify**) over those objects. First-order logic gives us this ability.

The examples in the last section can be encoded in **first-order** logic as

$$\forall\, x\ Rich(x) \Rightarrow \exists\, y\ [Owns(x, y) \wedge Car(y) \wedge Nice(y)]$$

and

$$\forall\, x\ Natural(x) \Rightarrow Natural(x + 1)$$

where $\forall$ and $\exists$ are **universal** and **existential** quantifiers, respectively.

As we will see, the syntax and semantics of first-order (FO) logic allow us to explicitly represent objects and relationships among object, which provides us with much more representational power than the propositional case. First-order logic, for example, can be used to represent number theory, set theory, and even the computations of Turing machines.

3 Syntax of FO Logic

Well-formed formulas (wff) of FO logic are composed of six types of symbols (not counting parentheses).

1. **constants symbols**, which will be interpreted as representing objects, e.g. Bob might be a constant.
2. **function symbols**, each having a specified arity (i.e. number of input arguments) and will be interpreted as functions from the specified number of input objects to objects. For example, fatherOf is a single-arity function symbol, whose natural interpretation is to return the father of its input. Zero-arity function symbols are thought of as the same as constants.
3. **predicate symbols**, each having a specified arity. Single argument predicates can be thought of as specifying properties of objects. For example, let Rich be a single arity predicate, then Rich(Bob) would be used to denote that Bob is rich. Zero-arity predicate symbols are treated as propositions as in propositional logic, so *first-order logic subsumes propositional logic*. These propositions can be thought of as properties of the world. Multi-arity predicates denote relations among objects. For example, let $Owns$ be a two-arity predicate, then $Owns(Bob, Car)$ may be indended to indicate that Bob owns Car.
4. **variable symbols**, will be used to quantify over objects
5. **universal and existential quantifiers**, will be used to indicate the type of quantification
6. **logical connectives and negation** ($\Rightarrow$, $\wedge$, $\vee$, $\Longleftrightarrow$, and $\neg$), which are the usual way of composing wffs from other wffs.

Using these symbols there are two main types of syntactic objects in first-order logic. First there are **terms**, which will be interpreted as objects. Second, there are **formulas**, which will be interpreted as either true or false.

$$
\begin{array}{rcl}
\textit{Formula} & \rightarrow & \textit{AtomicFormula} \\
& | & \textit{Formula Connective Formula} \\
& | & \textit{Quantifier Variable Formula} \\
& | & \neg\ \textit{Formula} \\
& | & (\textit{Formula}) \\
\\
\textit{AtomicFormula} & \rightarrow & P(T_1, \ldots, T_n), \text{ where } P \in \text{PRED}, T_i \text{ are Terms, and } n \text{ is the arity of } P \\
\\
\textit{Term} & \rightarrow & c, \text{ where } c \in CONST \\
& | & v, \text{ where } v \in \text{VAR} \\
& | & F(T_1, \ldots, T_n), \text{ where } F \in \text{FUNC}, T_i \text{ are Terms and } n \text{ is arity of } F \\
\\
\textit{Connective} & \rightarrow & \Rightarrow \mid \wedge \mid \vee \mid \iff \\
\textit{Quantifier} & \rightarrow & \forall \mid \exists
\end{array}
$$

Figure 1: The BNF grammar for first-order logic. The well-formed formulas (wffs) are defined relative to given sets of predicate symbols PRED, constant symbols CONST, function symbols FUNC, and variables VAR.

- **Terms** are defined recursively as. The atomic terms are either constant symbols or variable symbols. Terms can also be compositions of a function symbol applied to the appropriate number of terms. So for example, Bob, $FatherOf(Bob)$, $FatherOf(FatherOf(Bob))$, X, and $FatherOf(X)$ are all terms, where Bob is a constant, X is a variable, and $FatherOf$ is a function symbol.

- **Formulas** can be primitive formulas, which are simply predicates applied to the appropriate number of terms. We can also compose formulas from other formulas via logical connectives and negation. We can also generate formulas by applying a quantifier over a variable to the formula, where typically the formula will involve a term with the variable. For example, $TallerThan(Bob, FatherOf(Bob))$ is a primitive formula, which indicates that Bob is taller than his father. Another example

  $$TallerThan(Bob, FatherOf(Bob)) \wedge TallerThan(FatherOf(FatherOf(Bob)), Bob)$$

  is a logical combination of formulas that says Bob is taller than his father but shorter than his grandfather. Finally, $\exists\ x\ TallerThan(FatherOf(x), x)$ is a quantified formula that says there exists a person who is shorter than their father.

Figure 1 gives the grammar for the syntax of first-order logic, which shows how these symbols are used to build terms and formulas.

4 Additional Terminology and Examples

It will be helpful to be familiar with some common terminology regarding formulas and terms.

Atoms are atomic formulas.

Ground atoms/terms are atoms/terms without variables.
Ground terms are used to refer to specific objects, e.g. Bob or $FatherOf(Bob)$. Ground atoms are often used to state specific facts such as $Rich(Bob)$ or $TallerThan(Bob, FatherOf(Bob))$.

Ground formulas are formulas without variables.

Closed formulas are formulas in which all variables are associated with quantifiers. For example, $\forall x\ Number(x) \Rightarrow Number(Successor(x))$ is a closed formula. While $\forall x\ GreaterThan(x, y) \Rightarrow LessThan(y, x)$ is not a closed formula when y is a variable, since y does not have an associated quantifier.

Free Variables are variables in a formula that do not have an associated quantifier, such as y in the second example above. Typically free variables are treated as being implicitly universally quantified in first-order logic. Our primary reason for introducing the concept of free variables is to help define the semantics of formulas.

Although we have not yet defined the semantics of first-order logic lets consider some example formulas along with their intuitive natural language interpretations. Our convention will be to capitalize at least the first letter of constant symbols and use lowercase for variables.

- "Not all birds can fly."

$$\begin{array}{l}\neg(\forall x\ Bird(x) \Rightarrow Fly(x))\ , \quad \text{which is the same as} \\ \exists x\ Bird(x) \wedge \neg Fly(x)\end{array}$$

- "All birds cannot fly."

$$\begin{array}{l}\forall x\ Bird(x) \Rightarrow \neg Fly(x)\ , \quad \text{which is the same as} \\ \neg(\exists x\ Bird(x) \wedge Fly(x))\end{array}$$

- "If anyone can solve the problem, then Hilary can."

$$(\exists x\ Solves(x, Problem)) \Rightarrow Solves(Hilary, Problem)$$

- "Nobody in the Calculus class is smarter than everyone in the AI class"

$$\neg\exists x\ \ (TakesCalculus(x) \wedge (\forall\ y\ TakesAI(y) \Rightarrow SmarterThan(x, y)))$$

- "John hates all people who do not hate themselves."

$$\forall x\ \ (Person(x) \wedge \neg Hates(x, x)) \Rightarrow Hates(John, x)$$

5 Semantics of FO Logic

As for all logics, the first step in defining the semantics is to define the models of first-order logic. Recall that one of the benefits of using first-order logic is that it allows us to explicitly talk about objects and relations among them. Thus, our models will contain objects along with information about the relationships among objects. After defining the models of FO logic we will then talk about the interpretation of strings in FO logic. That is, what is the interpretation of wff and terms given a model.

5.1 First Order Models

More formally, a first-order model is a tuple $\langle D, I \rangle$ where D is a non-empty domain of objects and I is an interpretation function. The domain D is simply a set of objects or elements and can be finite, infinite, even uncountable. The interpretation function I assigns a meaning or interpretation to each of the available constant, function, and predicate symbols as follows:

- If c is a constant symbol then $I(c)$ is an object in D. Thus, given a model, a constant can be viewed as naming an object in the domain.
- If f is a function symbol of arity n then $I(f)$ is a total function from D^n to D. That is the interpretation of f is a function that maps n domain objects to the domain D.
- If p is a predicate symbol of arity n then $I(p)$ is a subset of D^n; that is, a predicate symbol is interpreted as a set of tuples from the domain. If a tuple $O = \langle o_1, \cdots, o_n \rangle$ is in $I(p)$ then we say that p is true for the object tuple O.

For example, suppose that we have one predicate $TallerThan$, one function $FatherOf$, and one constant Bob. A model M_1 for these symbols might be the following:

$$\begin{aligned} D &= \{ BOB, JON, NULL \} \\ I(Bob) &= BOB \\ I(TallerThan) &= \{ \langle BOB, JON \rangle \} \end{aligned}$$

Recall that $I(FatherOf)$ is a function, so to give the interpretation of $FatherOf$ we will just show the value for each input

$$\begin{aligned} I(FatherOf)(BOB) &= JON \\ I(FatherOf)(JON) &= NULL \\ I(FatherOf)(NULL) &= NULL \end{aligned}$$

Another possible interpretation M_2 might be,

$$\begin{aligned} D &= \{ BOB, JON \} \\ I(Bob) &= BOB \\ I(TallerThan) &= \{ \langle BOB, JON \rangle, \langle JON, BOB \rangle \} \\ I(FatherOf)(BOB) &= BOB \\ I(FatherOf)(JON) &= JON \end{aligned}$$

In the above, it is important to note the distinction between Bob which is a constant (a syntactic entity) and BOB which is an object in the domain (a semantic entity). The second interpretation is not what we might have in mind (e.g. the objects are fathers of themselves and $TallerThan$ is inconsistent), but it is still a valid model. It is the job of the knowledge base to rule out such unintended models from consideration by placing appropriate constraints on the symbols.

Before we define the semantics of wff and terms, we will need to introduce one more piece of notation for dealing with variables. Given a model $M = \langle D, I \rangle$, a variable x, and object $o \in D$ we define the **extended model** $M[x \rightarrow o]$ as a model that is identical to M, except that I is extended to interpret x as o, (i.e. $I(x) = o$).

5.2 First-Order Interpretations

We are now ready to define the meaning or interpretations of strings (terms and formulas) relative to a given model $M = \langle D, I \rangle$. Recall that we will denote the interpretation of a string ϕ relative to a model M by ϕ^M.

5.2.1 Interpreting Terms

Given a term and a model, the term is always interpreted as an object. The interpretation of a term t is defined recursively:

- If t is a constant or variable, then we have

$$t^M = I(t) \ .$$

  Note that if t is a variable, we assume that M has been "extended" to interpret that variable.

- If t is of the form $f(t_1, \cdots, t_n)$ where f is a function symbol and the t_i are terms, we have

$$t^M = I(f)(t_1^M, \cdots, t_n^M) \ .$$

So we see that each term t will be interpreted as a distinct object of the domain D.

Given the model M_1 from Section 5.1, we can make the following interpretations:

$$FatherOf(Bob)^{M_1} = I(FatherOf)(Bob^{M_1}) = I(FatherOf)(BOB) = JON$$

and

$$FatherOf(FatherOf(Bob))^{M_1} = I(FatherOf)(FatherOf(Bob)^{M_1}) = I(FatherOf)(JON) = NULL$$

5.2.2 Interpreting Formulas

Given the ability to interpret terms, we can now define the interpretation of a formula ϕ relative to M. The interpretation of any formula is either true or false and is given by the following recursive definition:

- If ϕ is an atomic formula of the form $p(t_1, \cdots, t_n)$, where p is a predicate and the t_i are terms we have

$$\phi^M = \begin{cases} true & \text{if } \langle t_1^M, \cdots, t_n^M \rangle \in I(p) \\ false & \text{otherwise} \end{cases}$$

- If ϕ is of the form $\phi_1 \circ \phi_2$ where $\circ$ is a logical connective, we have

$$\phi^M = \phi_1^M \circ \phi_2^M$$

- If ϕ is of the form $\neg\phi_1$ where ϕ_1 is a formula, we have

$$\phi^M = \neg\phi_1^M$$

- If ϕ is of the form, $\exists x\ \phi_1$ where ϕ_1 is a formula (that may or may not involve the variable x), we get
$$\phi^M = \begin{cases} true & \text{if there exists an } o \in D \text{ such that } \phi_1^{M[x \to o]} = true \\ false & \text{otherwise} \end{cases}$$
- If ϕ is of the form, $\forall x\ \phi_1$ where ϕ_1 is a formula (that may or may not involve the variable x), we get
$$\phi^M = \begin{cases} true & \text{if for all } o \in D \text{ we have } \phi_1^{M[x \to o]} = true \\ false & \text{otherwise} \end{cases}$$

Notice the use of extended models $M[x \to o]$ to define the semantics of quantified formulas.

As an example, consider the model M_1 from Section 5.1:

$$\begin{aligned} D &= \{\ BOB, JON, NULL\ \} \\ I(Bob) &= BOB \\ I(TallerThan) &= \{\ \langle BOB, JON\rangle\ \} \end{aligned}$$

Recall that $I(FatherOf)$ is a function; to give the interpretation of $FatherOf$, we will just show the value for each input

$$\begin{aligned} I(FatherOf)(BOB) &= JON \\ I(FatherOf)(JON) &= NULL \\ I(FatherOf)(NULL) &= NULL \end{aligned}$$

and the atomic formula $TallerThan(Bob, FatherOf(Bob))$. To compute the interpretation

$$TallerThan(Bob, FatherOf(Bob))^M;$$

we need to check whether

$$\langle Bob^M, FatherOf(Bob)^M\rangle \in I(TallerThan)\ .$$

Since $\langle Bob^M, FatherOf(Bob)^M\rangle = \langle BOB, JON\rangle$ which is in $I(TallerThan)$, we get that

$$TallerThan(Bob, FatherOf(Bob)^M = true.$$

For the model M_2, we get that this same formula is $false$ (verify this on your own).

For an example involving quantifiers, consider the formula

$$\exists x TallerThan(x, FatherOf(x))\ .$$

Consulting the above definition of the semantics we get that,

$$[\exists x TallerThan(x, FatherOf(x))]^M$$

is $true$ iff we can find an object o in D such that

$$TallerThan(x, FatherOf(x))^{M[x \to o]}$$

is $true$. BOB is such an object (verify on your own) so we see that the interpretation of the quantified formula is true.

Your book gives several nice examples of first-order knowledge bases that you should read about an understand.

6 Entailment in FO Logic

The notion of entailment for first-order formulas is defined exactly as for propositional logic. That is $KB \models \phi$ if all the models of KB are also models of ϕ. For example, KB might be a set of formulas about the natural numbers and ϕ might ask whether there is a largest prime number. If KB accurately captures the natural numbers then ϕ should be entailed.

To get a sense for this we might consider creating such a KB for the natural numbers by first having a function $Succ$ for successor, and a predicate NN for natural number. We could then add a formula such as:

$$\forall x\ NN(x) \Rightarrow NN(Succ(x))$$

If this is the only formula we have then we can see that many possible models satisfy it, including models with only a finite number of objects. This means that if $KB \models \phi$ we can't be confident that ϕ is true for the model of natural numbers that we typically think about. We might then add a predicate called $LessThan$ with the natural intended interpretation and then add a formula:

$$\forall x\ LessThan(x, Succ(x))$$

This captures some additional intuitive structure of the natural numbers, but still allows many models that seem inconsistent with the natural numbers. For example, we could have a model with a single object in the domain $D = \{1\}$ such that $I(Succ(1)) = 1$ and $I(LessThan) = \{\langle 1, 1\rangle\}$. We might try to rule out such models by add a formula such as:

$$\forall x \forall y LessThan(x, y) \Rightarrow \neg LessThan(y, x)$$

which rules out the previous model.

This process of adding formulas to try to narrow down the set of models to what we "have in mind" could go on and was one of the early goals of logicians in the context of the natural numbers (among other structures). We'll see that for the natural numbers this turns out to be impossible in a provable way.

Computing whether $KB \models \phi$ holds or not (i.e. deduction) is much more difficult for first-order logic than in the propositional case, and will be our next topic.

7 Summary

Make sure you can do the following things:

- Determine which strings are well-formed formulas and terms of first-order logic.
- Translate English statements into first-order logic expressions and vice versa.
- Determine the interpretation of any formula or term relative to any model, showing the recursive steps. Likewise given a formula create models that the formula is true or false in.