CS532, Winter 2010
Lecture Notes:
First-Order Logic: Syntax and Semantics

Dr. Alan Fern, afern@cs.orst.edu

January 8, 2010

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,

$$\begin{array}{rcl} \text{BobIsRich} & \Rightarrow & \text{BobHasNiceCar} \\ \text{TonyIsRich} & \Rightarrow & \text{TonyHasNiceCar} \\ \text{JonIsRich} & \Rightarrow & \text{JonHasNiceCar} \\ \vdots & \vdots & \vdots \end{array}$$

We see that propositional logic requires that we basically compile the concise general statement into many statements about specific people. This is a completely impractical and unintuitive way to represent, reason about, 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,

$$\begin{array}{rcl} \text{Natural1} & \Rightarrow & \text{Natural2} \\ \text{Natural2} & \Rightarrow & \text{Natural3} \\ \vdots & \vdots & \vdots \end{array}$$

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 questions about natural numbers, e.g. "Does there exist a largest natural number?". Any true AI should certainly be able to reason about the natural numbers, yet propositional logic does not allow for this.

1.1 Upgrading Propositional Logic with Quantifiers

In both of these examples we need the ability to directly talk about objects (e.g. people and numbers) and to write down logical statements that generalize (or quantify) over those objects. *First-order logic (FOL)* is a logic that gives us the ability to quantify over objects. You will often see FOL called *first-order predicate logic* or *first-order predicate calculus*. These are all different names for the same thing.

The examples in the last section can be encoded in FOL

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

and

$$\forall\, x\ \ (\text{Natural}(x) \Rightarrow \text{Natural}(x + 1))$$

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

As we will see, the syntax and semantics of first-order 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.

2 Syntax of FO Logic

Figure 1 gives the grammar for the syntax of first-order logic, which we will now describe.

We will first describe each type of symbol used to construct a first-order logic formula.

Logical connectives ($\Rightarrow$, $\wedge$, $\vee$, and $\iff$), negation ($\neg$), and parentheses. These will be used to recursively build complex formulas, just as was done for propositional logic.

Constants symbols are strings that will be interpreted as representing objects, e.g. *Bob* might be a constant.

Variable symbols will be used as "place holders" for quantifying over objects.

Predicate symbols each have an associated arity (i.e. number of arguments), which might be zero or some other finite value. Predicates will be used to denote properties of objects and relationships among them.

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.

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.

Multi-arity predicates denote relations among objects. For example, let Owns be a two-arity predicate, then Owns(Bob, Car) indicates that Bob owns Car.

$$
\begin{array}{rcl}
\textit{Formula} & \rightarrow & \textit{PrimitiveFormula} \\
& | & (\textit{Formula Connective Formula}) \\
& | & \neg\ \textit{Sentence} \\
& | & \textit{Quantifier Variable Formula} \\
\\
\textit{PrimitiveFormula} & \rightarrow & \textit{Predicate}(\textit{Term},\ldots,\textit{Term}) \\
\\
\textit{Term} & \rightarrow & \textit{Function}(\textit{Term},\ldots,\textit{Term}) \\
& | & \textit{Constant} \\
& | & \textit{Variable} \\
\\
\textit{Connective} & \rightarrow & \Rightarrow \mid \wedge \mid \vee \mid \iff \\
\textit{Quantifier} & \rightarrow & \forall \mid \exists \\
\textit{Constant} & \rightarrow & \text{any string that is not used as a variable predicate or function} \\
\textit{Variable} & \rightarrow & \text{any string that is not used as a constant predicate or function} \\
\textit{Predicate} & \rightarrow & \text{any string that is not used as a constant variable or function} \\
\textit{Function} & \rightarrow & \text{any string that is not used as a constant variable or predicate}
\end{array}
$$

Figure 1: The BNF grammar for first-order logic.

Function symbols each have a specified arity (i.e. number of input arguments) and will be interpreted as functions mapping the specified number of input objects to objects. For example, let FatherOf be a single-arity function symbol, then the natural interpretation of FatherOf(Bob) would be Bob's father.

Zero-arity function symbols are considered to be constants.

Universal and existential quantifier symbols will be used to quantify over objects.

For example, $\forall\ x$ Alive$(x) \Rightarrow$ Breathing(x) is a universally quantified statement that uses the variable x as a placeholder.

Given all of these syntactic pieces we can now describe the two types of strings that are used in FO logic: 1) terms, which are used to reference objects, and 2) formulas, which are used to encode knowledge.

2.1 Terms

Terms are used in FOL to reference objects. Terms are defined recursively as either:

- a constant
- a variable
- an n-arity function symbol applied to n terms

It is easy to see that if there is at least one function symbol with arity greater than zero, then there will be an infinite (but countable) number of terms. Intuitively terms are used to index or name objects in the world. We will see exactly how this is done when we discuss FOL semantics, where we will define interpretations for terms.

If Bob, 1, and 0 are constants and FatherOf and Plus are function symbols, then the following are all terms:

- Bob
- FatherOf(Bob)
- FatherOf(FatherOf(Bob))
- 1
- Plus(0,1)
- Plus(0,Plus(1,1))

2.2 Formulas

Formulas are used in FOL to state properties and will be interpreted as **true** or **false** when we discuss FOL semantics. Formulas are defined recursively as:

- *primitive formulas* (sometimes called *atoms*), which are simply predicate symbols applied to the appropriate number of terms,
- a logical combination of formulas, or
- an expression of the form "quantifier variable formula"

For example, if TallerThan is a predicate symbol then the following are some example formulas:

- TallerThan(Bob,FatherOf(Bob))
  is a primitive formula, which has the intended interpretation that *Bob* is taller than his father.
- 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.
- $\exists\, x$ TallerThan(FatherOf(x), x)
  is a quantified formula that says there exists a person who is shorter than their father.

Any string that is not formed according the above syntactic rules is not a well-formed formula.

2.3 Additional Terminology

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

Ground formulas are formulas without variables.

Ground atoms are atoms without variables.
Ground atoms are often used to state specific facts such as Rich(Bob).

Closed formulas are formulas in which all variables are associated with quantifiers.

- In contrast, a formula has **free variables** if it is not closed.
  For example, if x is a variable then $\text{Rich}(x)$ has x as a free variable, while $\forall\, x\ \text{Rich}(x)$ is closed.

Typically free variables are treated as universally quantified in first-order logic. Our primary reason for introducing the concept of free variables is to help define the semantics of formulas (described in Section 3).

2.4 Illustrative Examples

Although we have not yet defined the semantics of first-order logic lets consider some example formulas along with their intuitive natural language interpretations.

- "Not all birds can fly."

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

- "All birds cannot fly."

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

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

$$(\exists\, x\ \text{Solves}(x, \text{problem})) \Rightarrow \text{Solves}(\text{Hilary}, \text{problem})$$

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

$$\neg[\exists\, x\ \text{TakesCalculus}(x) \wedge (\forall\, y\ \text{TakesAI}(y) \Rightarrow \text{SmarterThan}(x, y))]$$

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

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

3 Semantics of First-Order 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 properties of objects and the relationships among objects.

3.1 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 $n > 0$, 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.
- If p is a predicate symbol of arity 0, i.e. a simple proposition, then $I(p)$ is equal to either **true** or **false**.

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:

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

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

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

Another possible interpretation M_2 might be,

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

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.

Extended Models. Before we define the semantics of strings, we will need to introduce one more piece of notation for dealing with variables. Given a model $M =< D, I >$, 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. $I(x) = o$).

3.2 Interpreting Terms

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. First, we define recursively the semantics for a term t:

- If t is a constant or variable, then $t^M = I(t)$. 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. This is important to remember. Terms are never interpreted as **true** or **false**, but are exclusively used in FO logic to name or index objects in the domain of a model.

Given the model M_1 from Section 3.1, we can make the following interpretations:

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

and

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

3.3 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 relative to a model is either **true** or **false** and is given by the following recursive definition:

- If ϕ is a primitive 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 3.1:

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

and the atomic formula TallerThan(Bob,FatherOf(Bob)). To compute the interpretation TallerThan(Bob, FatherOf(Bob)) we need to check whether

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

Since $\langle \text{Bob}^M, \text{FatherOf}(\text{Bob})^M \rangle = \langle \text{BOB}, \text{JON} \rangle$ which is in $I(\text{TallerThan})$, 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

$$\exists x \text{TallerThan}(x, \text{FatherOf}(x)) \ .$$

Consulting the above definition of the semantics we get that,

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

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

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

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

3.4 Entailment in First-Order 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 (i.e. the models that KB is true in) are also models of ϕ. For example, KB might be a set of formulas[1] 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.

Computing whether KB $\models \phi$ holds or not (i.e. deduction) is much more difficult for first-order logic than in the propositional case. The primary reason for this is that entailment is a statement about all models and the space of first-order models in infinite (uncountably infinite in fact), whereas the space of propositional models is finite. Our next topic will be about deduction.

[1] Often we think of a knowledge base as a set of formulas. However, in reality the set of formulas are treated semantically as a conjunction of all those formulas and hence a single formula.

4 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.

Finally, your book gives several nice examples of first-order knowledge bases. You should read about these and understand them.