LECTURE 10: FIRST-ORDER LOGIC

Software Engineering
Mike Wooldridge
1 Why not Propositional Logic?

- Consider the following statements:
  - all monitors are ready;
  - X12 is a monitor.
- We saw in an earlier lecture that these statements are propositions: their meaning is either true or false.
- Propositional logic is the most abstract level at which we can study logic.
- As we shall say, it is too coarse grained to allow us to represent and reason about the kind of statement we need to write in formal specification.
Consider the following argument:

all monitors are ready
X12 is a monitor
\[ \text{therefore } X12 \text{ is ready} \]

Intuitively, we can see that this argument is sound: if we accept that the two premises (i.e., the statements above the line) are true, then we must accept that the conclusion is true also.

(Later, we shall see how we can do this kind of reasoning formally.)

The only way we could represent these statements in propositional logic would be:

– let \( p \) be \textit{all monitors ready};
– let \( q \) be \ldots

And the sense of the argument would be lost; in fact, if we represented the three statement in propositional logic, then we could not derive the conclusion.
We shall now introduce a generalisation of propositional logic called first-order logic (FOL). This new logic affords us much greater expressive power.

First, we shall look at how the language of first-order logic is put together.
2.1 Terms

- The basic components of FOL are called terms.
- Essentially, a term is an object that denotes some object other than true or false.
- The simplest kind of term is a constant.
- A value such as 8 is a constant; the denotation of this term is the number 8 — a value that is contained in the sets \( \mathbb{N} \) and \( \mathbb{Z} \).
- We often use constants in maths; we introduce them by writing things like

  \[ \text{Let } S \text{ be the set } \{1, 2, 3\}. \]

  In this case, we have introduced a constant and made its denotation clear; we have given it an interpretation.

- We can have constants that stand for any kind of object; for example, we could have a constant that stood for (denoted) the individual ‘Michael Wooldridge’. 
• The second simplest kind of term is a \textit{variable}.

• A variable can stand for anything in a set of objects.

• That is, a variable of type $\mathbb{N}$ could stand for any of the natural numbers.

• Let's just formalise this before going any further.

• \textbf{Definition:} A \textit{constant} of type $T$ is a name that denotes some particular object in the set $T$.

• \textbf{Definition:} A \textit{variable} of type $T$ is a name that can denote any value in the set $T$. 
• We can now introduce a more complex class of terms — *functions*.

• The idea of functional terms in logic is similar to the idea of a function in programming: recall that in programming, a function is a procedure that takes some arguments, and *returns a value*.

In Modula-2:

```plaintext
PROCEDURE f(a1:T1; ... an:Tn) : T;
```

this function takes $n$ arguments; the first is of type $T_1$, the second is of type $T_2$, and so on. The function returns a value of type $T$.

• In FOL, we have a set of *function symbols*; each symbol corresponds to a particular function. (It denotes some function.)

• Each function symbol is associated with a natural number called its *arity*. This is just the number of arguments it takes.
• Each function symbol has a return-type associated with it…

• … and each function symbol has an argument type associated with it.

• A functional term is then built up by applying a function symbol to the appropriate number of terms, of the appropriate type.

• Formally …

  **Definition:** Let \( f \) be an arbitrary function symbol of type \( T \), with arity \( n \in \mathbb{N} \), taking arguments of type \( T_1, \ldots, T_n \) respectively. Also, let \( \tau_1, \ldots, \tau_n \) be terms of type \( T_1, \ldots, T_n \) respectively. Then

  \[
  f(\tau_1, \ldots, \tau_n)
  \]

  is a functional term.
• All this sounds complicated, but isn’t. Consider a function \( \text{plus} \), which takes just two arguments, each of which is a natural number, and returns the first number added to the second.

Then:

- \( \text{plus}(2, 3) \) is an acceptable functional term;
- \( \text{plus}(0, 1) \) is acceptable;
- \( \text{plus}(\text{plus}(1, 2), 4) \) is acceptable;
- \( \text{plus}(\text{plus}(\text{plus}(0, 1), 2), 4) \) is acceptable;

but

- \( \text{plus}(-1, 0) \) isn’t;
- and neither is \( \text{plus}(0.1, 2) \).
• In maths, we have many functions; the obvious ones are

\[ + \quad - \quad / \quad \times \quad \sqrt{\quad \sin \quad \cos \quad \ldots} \]

• The fact that we write

\[ 2 + 3 \]

instead of something like

\[ \text{plus}(2, 3) \]

is merely a matter of convention, and is not relevant from the point of view of logic; all these are functions in exactly the way we have defined.

• Using functions, constants, and variables, we can build up expressions, e.g.:

\[ (x + 3) \times \sin 90 \]

(which might just as well be written

\[ \text{times}(\text{plus}(x, 3), \sin(90)) \]

for all it matters.)
2.2 Predicates

- In addition to having terms, FOL has relational operators, which capture relationships between objects.
- The language of FOL contains a stock of predicate symbols.
- These symbols stand for relationships between objects.
- Again, each predicate symbol has an associated arity…
- … and each argument has a type.
- **Definition:** Let $P$ be a predicate symbol of arity $n \in \mathbb{N}$, which takes arguments of types $T_1, \ldots, T_n$. Then if $\tau_1, \ldots, \tau_n$ are terms of type $T_1, \ldots, T_n$ respectively, then $P(\tau_1, \ldots, \tau_n)$ is a predicate, which will either be true or false under some interpretation.
• EXAMPLE. Let \( gt \) be a predicate symbol with the intended interpretation ‘greater than’. It takes two arguments, each of which is a natural number. Then:

– \( gt(4, 3) \) is a predicate, which evaluates to \( true \);
– \( gt(3, 4) \) is a predicate, which evaluates to \( false \).

but

– \( gt(-1, 2) \) isn’t a predicate.

• The following are standard mathematical predicate symbols:

\[
> \leq \geq \neq \in \subseteq \ldots
\]

• Once again, the fact that we are normally write \( x > y \) instead of \( gt(x, y) \) is just convention.

• We can build up more complex predicates using the connectives of propositional logic:

\[
(2 > 3) \land (6 = 7) \lor (\sqrt{4} = 2)
\]
• So a predicate just expresses a relationship between some values.

• What happens if a predicate contains *variables*: can we tell if it is true or false? Not usually; we need to know an *interpretation* for the variables.

• A predicate that contains no variables is a proposition.

• Predicates of arity 1 are called *properties*.

• EXAMPLE. The following are properties:

\[
\begin{align*}
\text{Man}(x) \\
\text{Mortal}(x) \\
\text{Malfunctioning}(x).
\end{align*}
\]

• Predicate that have arity 0 (i.e., take no arguments) are called *primitive propositions*. 
3 Quantifiers

• We now come to the central part of first order logic: quantification.

• Consider trying to represent the following statements:
  – all men have a mother;
  – every natural number has a prime factor.

• We can’t represent these using the apparatus we’ve got so far; we need quantifiers.
• In Z, we shall use three quantifiers:

\( \forall \) — *the universal quantifier*;

is read ‘for all…’

\( \exists \) — *the existential quantifier*;

is read ‘there exists…’

\( \exists_1 \) — *the unique quantifier*;

is read ‘there exists a unique…’
• The simplest form of quantified formula in Z is as follows:

\[
\text{quantifier signature} \bullet \text{predicate}
\]

where

- quantifier is one of \( \forall, \exists, \exists_1 \);
- signature is of the form

\[
\text{variable} : \text{type}
\]

- and predicate is a predicate.
• EXAMPLES.
  
  – $\forall x : Man \bullet Mortal(x)$
  ‘For all $x$ of type $Man$, $x$ is mortal.’
  (i.e. all men are mortal)
  
  – $\forall x : Man \bullet \exists! y : Woman \bullet MotherOf(x, y)$
  ‘For all $x$ of type $Man$, there exists a unique $y$ of type $Woman$, such that $y$ is the mother of $x$.’
  
  – $\exists m : Monitor \bullet MonitorState(m, \text{ready})$
  ‘There exists a monitor that is in a ready state.’
  
  – $\forall r : Reactor \bullet \exists t : 100 \ldots 1000 \bullet Temp(r) = t$
  ‘Every reactor will have a temperature in the range 100 to 1000.’
• More examples:

  – \( \exists n : \mathbb{N} \bullet n = (n \times n) \)
    ‘Some natural number is equal to its own square.’

  – \( \exists c : EC \bullet \text{Borders}(c, \text{Albania}) \)
    ‘Some EC country borders Albania.’

  – \( \forall m, n : \text{Person} \bullet \neg \text{Superior}(m, n) \)
    ‘No person is superior to another.’

  – \( \forall m : \text{Person} \bullet \neg \exists n : \text{Person} \bullet \text{Superior}(m, n) \)
    Ditto.
4 Comments

• Note that universal quantification is similar to conjunction:

$$\forall n : \{2, 4, 6\} \bullet Even(n)$$

is the same as

$$Even(2) \land Even(4) \land Even(6).$$

• In the same way, existential quantification is the same as disjunction:

$$\exists n : \{7, 8, 9\} \bullet Prime(n)$$

is the same as

$$Prime(7) \lor Prime(8) \lor Prime(9).$$
The universal and existential quantifiers are in fact duals of each other:

\[ \forall x : T \cdot P(x) \iff \neg \exists x : T \cdot \neg P(x) \]

Saying that everything has some property is the same as saying that there is nothing that does not have the property.

\[ \exists x : T \cdot P(x) \iff \neg \forall x : T \cdot \neg P(x) \]

Saying that there is something that has the property is the same as saying that its not the case that everything doesn’t have the property.
5 Decidability

• In propositional logic, we saw that some formulae were tautologies — they had the property of being true under all interpretations.

• We also saw that there was a procedure which could be used to tell whether any formula was a tautology — this procedure was the truth-table method.

• A formula of FOL that is true under all interpretations is said to be valid.

• Now we can’t use truth tables to tell us whether a formula of FOL is valid.

• Is there any other procedure that we can use, that will be guaranteed to tell us, in a finite amount of time, whether a FOL formula is, or is not, valid?

• The answer is no.

• FOL is for this reason said to be undecidable.