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 *therefore* X12 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.

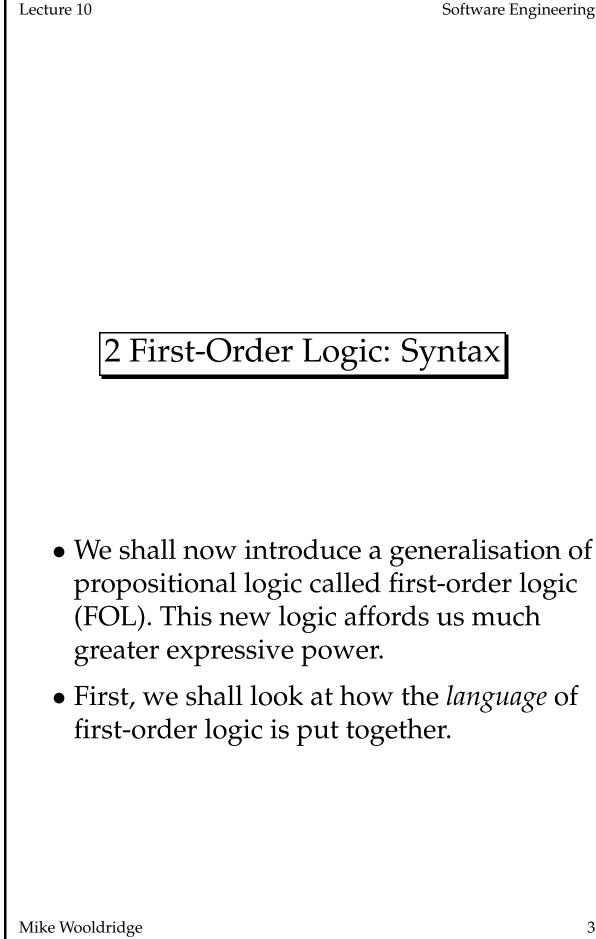
(Later, we shall see how we can do this kind of reasoning formally.)

- The only way we could represent these statements in propositional logic would be:
 - let *p* be all monitors ready;

– let *q* be ...

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.





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 Z.
- We often use constants in maths; we introduce them by writing things like

```
Let S 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'.

Lecture 10 Software Engineering • The second simplest kind of term is a 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. • Lets just formalise this before going any further. • **Definition:** A *constant* of type *T* is a name that denotes some particular object in the set T. • **Definition:** A *variable* of type *T* is a name that can denote any value in the set *T*.

Lecture 10

- 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:

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

this function takes *n* arguments; the first is of type T1, the second is of type T2, 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.

Software Engineering

Lecture 10

• 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 τ_1, \ldots, τ_n be terms of type T_1, \ldots, T_n respectively. Then

 $f(\tau_1,\ldots,\tau_n)$

is a functional term.

ing
t
1
8

• In maths, we have many functions; the obvious ones are

 $+ - / * \sqrt{-} \sin \cos \ldots$

• The fact that we write

2 + 3

instead of something like

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.:

 $(\mathbf{x}+3) * \sin 90$

(which might just as well be written

```
times(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* ∈ ℕ, which takes arguments of types *T*₁,...,*T_n*. Then if *τ*₁,...,*τ_n* are terms of type *T*₁,...,*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:

 $> < = \geq \leq \neq \in \subset \subseteq \ldots$

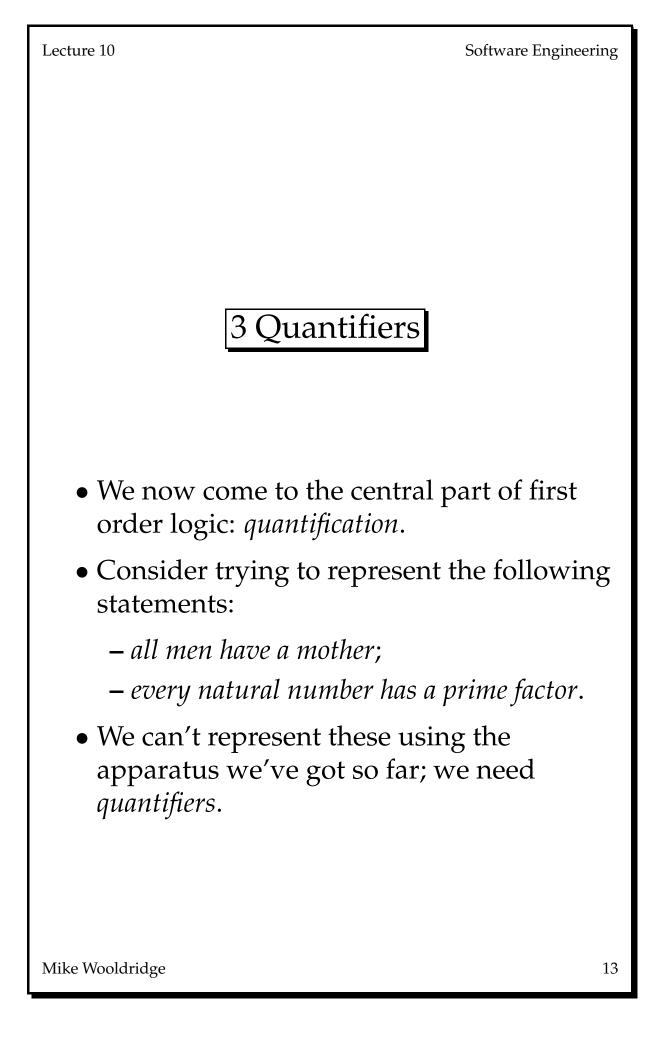
- Once again, the fact that we are normally write x > y instead of gt(x, y) is just convention.
- We can build up more complex predicates using the connectives of propositional logic:

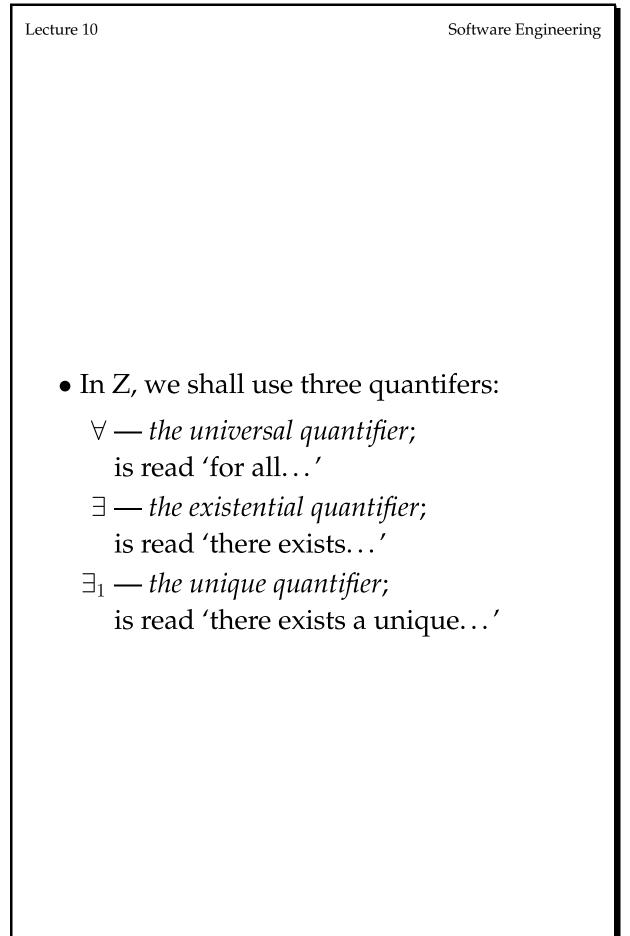
 $(2 > 3) \land (6 = 7) \lor (\sqrt{4} = 2)$

- 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 follolwing are properties:

Man(x)Mortal(x)Malfunctioning(x).

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



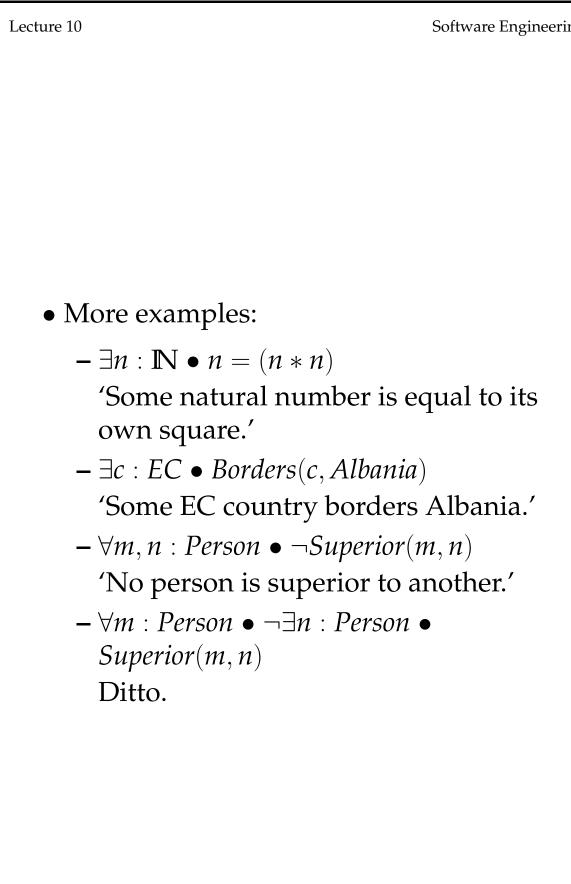


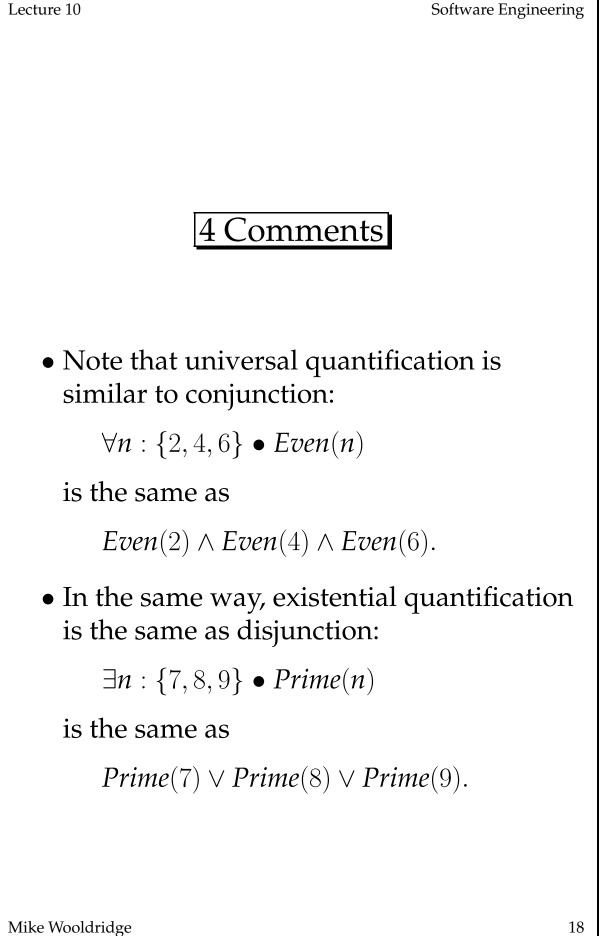
```
Lecture 10
                                            Software Engineering
   • The simplest form of quantified formula in
     Z is as follows:
         quantifier signature • predicate
     where
       – quantifier is one of \forall, \exists, \exists<sub>1</sub>;
       - signature is of the form
             variable : type
       - and predicate is a predicate.
Mike Wooldridge
                                                           15
```

• EXAMPLES.

- -∀x : Man Mortal(x)
 'For all x of type Man, x is mortal.'
 (i.e. all men are mortal)
- $\forall x : Man \bullet \exists_1 y : Woman \bullet MotherOf(x, y)$ 'For all *x* of type *Man*, there exists a unique *y* of type *Woman*, such that *y* is the mother of *x*.'
- *¬*∃*m* : *Monitor MonitorState*(*m*, *ready*)
 'There exists a monitor that is in a ready state.'
- $\forall r : Reactor \bullet \exists_1 t : 100 \dots 1000 \bullet Temp(r) = t$

'Every reactor will have a temperature in the range 100 to 1000.'





```
Lecture 10
                                             Software Engineering
   • The universal and existential quantifiers
     are in fact duals of each other:
         \forall x: T \bullet P(x) \iff \neg \exists x: T \bullet \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 \bullet P(x) \iff \neg \forall x: T \bullet \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*.