Lecture 7: Predicate Logic

In this lecture, we introduce predicate logic, also called first-order logic.

Free Variables

Remember that a *proposition* is a statement that can be true or false, such as '4 is even'. Many simple propositions are built from a *subject*, in this case the number 4, and a *predicate*, which is something that can be applied to a subject, in this case 'is even'.

What about the statement 'n is even'?

We say that this statement contains a *free variable*. Only when we assign a value to the variable do we get a proposition. This is the same situation as with evaluating arithmetic terms: $2 + 2$ is something we can evaluate directly, but $2 + x$ has a free variable x, and we can only evaluate it in an environment. Note that $x - x$ is still a term with a free variable x, even though its evaluation in any environment that defines x will be zero, so we can say $x - x \equiv 0$.

Statements with free variables are another way of describing functions[1]: 2 is a number, $f(x) = x+2$ is a function that gives you a number, but you have to provide an input number first.

A statement with one or more free variables, in mathematics, is also called a *predicate* as you can apply it to values to get a proposition. For example, $P(n) : 1 + \ldots + n = n(n + 1)/2$ is a predicate that is true for all natural numbers[2] $n \geq 1$.

What about these three statements over the natural numbers?

- $n(n + 1)$ is even.
- For every natural number n, the term $n(n + 1)$ is even.
- The product of a natural number with its successor is even.

The first is a predicate with one free variable n, and it evaluates to true in every environment (that is, it is a tautology). The third statement does not contain any variables, but the second one does, even though they seem to say more or less the same thing. Predicate logic gives a meaning to the second statement that makes it say the same as the third, because the variable here is not free.

Quantifiers

We define two quantifiers that turn predicates (of one free variable) into propositions by *binding* the variable.

Definition 1 (quantifiers)**.** If $P(x)$ is a predicate over some domain D with a free variable x, then in predicate logic,

- The statement 'for all x in D, $P(x)$ [holds]' is a proposition. In symbols: $\forall x : D.P(x)$.
- The statement 'there exists an x in D for which $P(x)$ [holds]' is a proposition. In symbols: $\exists x : D.P(x)$.

The variables behind the quantifiers $\forall, \exists$ are called *bound variables* within the scope of the

[1] Functional programming makes this connection explicit.

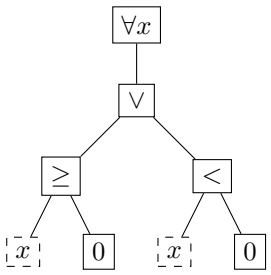
[2] It also holds for $n = 0$ if you make the convention that the sum on the left, for $n = 0$, is zero.

quantifiers.

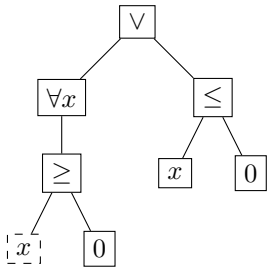
If the domain is clear, we can just write $\forall x.P(x)$ and $\exists x.P(x)$.

By convention, the scope of a quantifier is the smallest phrase after the quantifier that it could match, with even higher precedence than $\neg$. So, $\forall x.A(x) \vee B(x)$ means $(\forall x.A(x)) \vee B(x)$ and if you want the other meaning (which you almost always do!) then you have to write $\forall x.\,(A(x) \vee B(x))$.

Writing out two examples as terms,



$\forall x.\,(x \geq 0 \vee x < 0)$



$\forall x.x \geq 0 \vee x < 0$

The x terms in dashed boxes are bound variables (a variable is bound if and only if, on the path from its node to the root of the term tree, there is a node higher up that binds this variable like the $\forall x$ here), whereas in the term on the right, one of the x terms is free. Using the same variable both bound and free in the same term, while legal syntax, is extremely ugly — remember, mathematics is meant to be read by humans — and you should avoid this in your own writing. The short version is it doesn't hurt to always bracket what you are trying to capture with a quantifier.

We need to be a bit more pedantic than this, however. If you are doing proofs in arithmetic, or almost any other theory except logic itself, then you are working on two layers with arithmetic (or something else) on the lower layer and logic on the upper layer. Quantifiers, however, only apply to statements in logic. Thus in arithmetic over the integers (or reals), $\forall x.x + 1 \geq x$ can only mean $\forall x.(x + 1 \geq x)$ because x or $x + 1$ themselves are arithmetic terms, not logical ones, so you cannot apply quantifiers to them directly ($\forall x.x$ and $\forall x.x + 1$ are not valid term strings). In mathematics, unlike in C programming, arithmetic terms do not have truth values.

Connectors like $=, >, \geq$ turn two arithmetic terms into equations or inequalities, and those do have truth values (at least in environments). So:

- $x + 1$ is an arithmetic term with one free variable. It does not have a truth value, whether you provide an environment or not. You can evaluate it in an environment to get a number value, but that is not a truth value as far as mathematicans are concerned.
- $x + 1 = 2$ is an arithmetic equation with one free variable. This does have a truth value (it is true in the environment $\{x = 1\}$ and false in all other environments). $x+1 > x$ is an arithmetic inequality with one free variable, and is true in all environments, therefore it is a tautology.
- $\forall x.x + 1 > x$ is a logical term with no free variables (the x is bound by the quantifier). Since the inequality it binds is a tautology, it evaluates to true.

If you are doing proofs in logic itself (logic on the lower as well as the upper layer), or you are using logical connectors in your terms such as $\forall x.(x \geq 0 \vee x < 0)$, that is the point when you need to be careful about brackets.

Proofs with Quantifiers

To prove that a statement with a quantifier is true or false, you have some options depending on the domain:

- By convention, the term $\forall x : D.P(x)$ over an empty domain D is always true. For example, a statement 'for all numbers n that are both odd and even, n is a power of 2' is true, as there are no such numbers, just like $A \to B$ is true if A is false.
- If you are trying to show $\forall x : D.P(x)$ over some finite domain D, then you can just try out $P(x)$ on every case. The statement is true if and only if all cases are true.
- If you are trying to show $\forall x : D.P(x)$ over an infininte (or at least, impracticably large) domain there are two main proof tactics.
  - Start with an x and make no assumptions except that x is in D, and calculate. If you reach $P(x)$, then you have shown the statement.
    For example, you can show '$\forall x : \mathbb{N}.\ x(x+1)$ is even' this way, with the exact same proof that you have already seen.
  - If your domain is the natural numbers, you might be able to use proof by induction.
- If you want to show that the statement $\forall x : D.P(x)$ is *false*, then you only have to find a single x in D such that $\neg P(x)$ is true. Such an x is called a *counter-example*.
- The term $\exists x : D.P(x)$ over an empty domain D is false, by convention. (If a domain is empty then there is nothing there to exist.)
- To show $\exists x : D.P(x)$, you have to find a single x in D where $P(x)$ is true. Such an x is called a *witness* for the statement.
- To show $\exists x : D.P(x)$ is *false*, you have to show that for every x in D, the expression $P(x)$ is false.

Indeed, we can pull negations into and out of quantifiers:

Definition 2 (DeMorgan for quantifiers)**.** We have, over any domain D:

- $\neg\left(\forall x.P(x)\right) \equiv \exists x.\neg P(x)$
- $\neg\left(\exists x.P(x)\right) \equiv \forall x.\neg P(x)$

Pulling a negation into or out of a quantifier flips the quantifier.

This means that to show a $\forall$ quantified statement is false, or an $\exists$ quantified statement is true, it is enough to find a single example, whereas for the other two cases if you want to show a $\forall$ quantified statement is true or an $\exists$ quantified statement is false, you need to make an argument that covers the whole domain.

Domains

What is a domain? It is a way of not getting into contradictions. As long as you are doing mathematics over some set, such as the integers, or the real numbers, then everything is fine and you can simply write e.g. $\forall x.(x+1 > x)$ leaving the domain implicit, or you can add a domain if you are temporarily only looking at some subset e.g. $\forall x : \mathbb{N}.x \geq 0$. In this case (your 'universe' is the reals, but you are temporarily looking at the natural numbers) you can also write a domain restriction as

an implication: over the reals, the last statement can be written $\forall x.(x \in \mathbb{N} \to x \geq 0)$ and you can see why forall-quantification over an empty domain is defined to be true: you have an implication where the left-hand side is always false.

You cannot ever write $\forall x.P(x)$ without having some domain in mind, because doing so would allow you to create statements of the form 'this statement is false'. In fact, the domain has to be a set, and 'all possible mathematical objects' is not a set, because sets themselves are mathematical objects, and 'all sets' cannot be a set without hitting something called Russell's paradox: if you could form the set of all sets, then you could also form the subset of all sets that do not contain themselves, and then you have a contradiction both if this set does contain itself, or does not contain itself.

The above points are not too much of a problem in practice, at least not at the level we are working, but the following is a very practical problem. Is the proposition $\forall x.x \geq 0$ true? It is true in the natural numbers, and false in the integers. So if you write a quantifier without an explicit domain, this is fine as long as you understand that the truth value of your proposition might vary across different domains. Where this makes a difference for a proof, you need to be careful.

Some books write $\forall x \in \mathbb{N}.x \geq 0$ using the 'element of' symbol instead of a colon to denote a domain. This is fine as long as you understand that the symbol, when it appears this way behind a quantifier, is doing something different than when it appears anywhere else.

Renaming

The names of free variables are important: $x > 0$ and $y > 0$ are two different predicates, that for example describe different areas of the 2D plane. Bound variables however you can rename most of the time. Instead of having to say 'the product of any natural number with its successor is even' you can say 'for any natural number n, the term $n(n+1)$ is even' but the name n is just a convention here — if you said 'for any natural number x, the term $x(x+1)$ is even' you would mean exactly the same thing.

So, we would like to state a rule like $\forall x.P(x) \vdash \forall y.P(y)$ that lets us rename bound variables, but there is a trap here. What if $P(x)$ is the statement $x \geq y$? Then the first term is $\forall x.x \geq y$ which is true in some domains and false in others (for example it is true over the natural numbers in the environment $y = 0$, but false over the natural numbers in the environment $y = 1$), but after renaming, we would get $\forall y.y \geq y$ which is a tautology. So the rule needs an extra clarification.

Definition 3 (α-renaming)**.** In a term $\forall x.P(x,\ldots)$ or $\exists x.P(x,\ldots)$, you can rename the bound variable to any other name that does not appear as a free variable in P.

The term you get this way is equivalent (in terms of both syntax and semantics) to the original one. This procedure is called α-renaming in functional programming and related mathematical theories (such as the lambda calculus).

Nested Quantifiers

You can nest statements with quantifiers, for example over the integers $\forall x.\exists y.x + y = 0$ is the statement that every integer has an inverse under addition.

It is legal syntax-wise, but very bad style, to nest quantifiers over the same variable; in this case the innermost quantifier 'shadows' the variable of the outside ones. You should always α-rename any statements you come across that do this.

Nested quantifiers let you express a lot of mathematical facts in a precise way. Looking just at addition over the integers:

- $\forall x.\forall y.x+y = y+x$. This is the commutative law of addition, spelled out this reads: 'for any two integers x, y, the values $x+y$ and $y+x$ are equal'. Nested $\forall$ quantifiers mean 'for any possible way of picking values for all of these variables together'.
- $\exists x.\forall y.x+y = y$. This statement reads: 'there exists one number x, such that for any number y, if you add x to y then you get the same y back'. The number in question is of course 0, and more generally a number with this property is called a neutral element of an operation (for example, 1 is the neutral element of multiplication).
- $\forall x.\exists y.x+y = 0$. This statement reads: 'for any number x, there exists a number y, so that if you add x and y then you get 0.' A number with this property is called an inverse of an operation, and the inverse of x (for addition) is $(-x)$.

This shows that there is a big difference between statements of the form $\forall x\exists y$ and $\exists x\forall y$. In the first case, the statement says that for any x you can find a y, but the value of y might depend on x (the inverse of 2 is different from the inverse of 3). In the second case, the statement says you can find one single x that works for all y at the same time (the same number 0 can be added to any other number without changing it). That is a stronger statement. In fact, it is true in general that over any domain and for any predicate P

$$\exists x.\forall y.P \quad \vDash \quad \forall y.\exists x.P$$

but the other way round does not hold, for example $\exists x.\forall y.x + y = 0$ is false, as there is no one number x that you can add to any number y to get 0. (There is, of course, a number like this if you are multiplying instead of adding.)

Uniqueness Proofs

There are many mathematical statements saying 'there is at most one object with property P' or 'there is exactly one object with property P'. Some books introduce an extra quantifier $\exists!$ to mean 'there is exactly one', but we can build this out of the more basic quantifiers ourselves, in a way that shows how we would prove such a statement. There are two ways in mathematics to do a uniqueness proof, that is a proof that there is at most one object with property P:

1. Pick any two objects a, b that both have property P. (In mathematics, 'pick any two' does not mean they have to be distinct.) Then show that $a = b$.
2. Pick any two distinct objects a, b with property P, that is $P(a)$ holds and $P(b)$ holds and $a \neq b$. Show that this leads to a contradiction.

For example, we can write Euclid's theorem in predicate logic as follows, where the domain is the integers. First, define the predicate DR (division with remainder):

$$DR(a, b, q, r) \quad := \quad a = qb + r \ \wedge \ 0 \leq r \ \wedge \ r < b$$

Then the theorem says

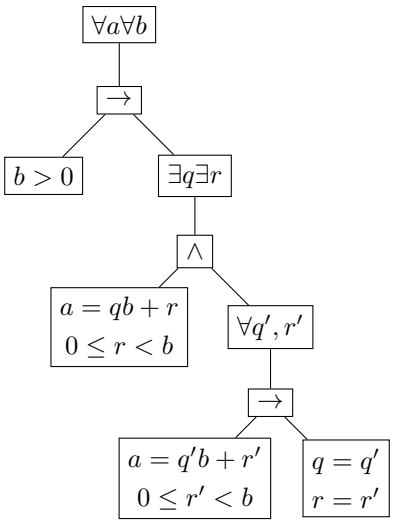
$$\forall a.\forall b.\,(b > 0 \rightarrow \exists q.\exists r.\,(DR(a, b, q, r) \wedge \forall q'.\forall r'.\,(DR(a, b, q', r') \rightarrow q = q' \wedge r = r')))$$

Which we can read as follows:

'Integers a, b, q, r have the division-with-remainder property if $a = qb + r$ and r is between 0 and less than b. For any integers a, b where $b > 0$, there are integers q, r such that a, b, q, r have the division-with-remainder property, and for any other q', r' such that a, b, q', r' also have the division-with-remainder property, in fact $q = q'$ and $r = r'$.'

This of course means that there is exactly one pair (q, r) for any (a, b) that makes up the division-with-remainder property (as long as $b > 0$ of course).

The structure of this theorem becomes clearer if you draw out the term:



This reads roughly as:

- For any a, b, if $b > 0$, then we can find q, r such that
  - q, r are a division of a by b with remainder, and
  - any q', r' that are also a valid division with remainder, are the same as q, r.

But even mathematicians would spell this theorem out in words, as we did originally, because it is easier for humans to read that way.

Predicate Logic as a Formal Language

We have mentioned that logic is usually found as an upper layer to talk about some theory (such as arithmetic) on a lower layer. We can make this into a more formal definition, where elements on the lower layer are called *terms* and elements on the upper layer are called *formulas*.

Definition 4 (predicate logic)**.** To do predicate logic with a theory (such as arithmetic) you need the following:

Lower layer: anything that you can build with the following is called a *term*.

- A set of constants (for example, the integer).
- A set of variables (that are different from the constants).
- A set of functions that map constants to constants (in arithmetic, these would be things like addition).

Upper layer: anything built with the following is a *formula*.

- A set of predicates, that turn terms into truth values. For example, $=$ (equals) for numbers.
- The operations from propositional logic (such as $\wedge, \vee$) and the constants T, F.
- The quantifiers $\forall$ and $\exists$.

Every function and predicate must have a fixed arity (the number of inputs that it takes).

A formula that contains a single predicate (no operations or quantifiers from the upper layer) is called an *atomic formula*, all other formulas are called *compound*.

For example

- For integer arithmetic, $x + 1$ is a term, and $x + 1 > 2$ is an atomic formula. The predicate here is 'greater than', which we could write as $GT(x + 1, 2)$ but using infix notation in this case is more human-friendly.
- $x > 0 \vee x < 0$ is a compound formula, in which x is free, and $\exists x.(x > 0)$ is a compound formula (since it contains a quantifier) in which x is a bound variable.
- For another example, we could take the set of all people as our constants, no functions on the lower layer, but predicates '$S(x) : x$ is a student' and '$E(x) : x$ passes the mathematics exam'. Then '$\forall x.(E(x) \rightarrow S(x))$' means 'everyone who passes the mathematics exam is a student'.

And finally:

Definition 5 (truth set)**.** The *truth set* of a predicate P over domain D is all elements x of D for which $P(x)$ is true.

- The formula $\forall x : D.P(x)$ is true if and only if the truth set of P over D is all of D (this includes the case when D is empty).
- The formula $\exists x : D.P(x)$ is true if and only if the truth set of P over D is not empty (in this case D itself cannot be empty).