

PY1003 Introduction to Logic
Lecture 11

A bit of terminology

- Predicate logic is sometimes called First-Order Logic.
- An atomic sentence of Sentential logic is a single sentence letter: A, B etc.. An atomic sentence of Predicate logic is a single predicate letter followed by an appropriate number of individual letter or letters, e.g. Fa, Gb etc.. (We will look later on at predicate letters which are followed by more individual letters than one.)
- Compound sentences of Predicate can be constructed from the atomic sentences using connectives, just as in Sentential logic, e.g. $Fa \rightarrow Gb$. Quantified sentences can also be combined using connectives, e.g. $\forall xFx \wedge \forall xGx$.
- Individual letters (a, b, c, etc.) are sometimes called *individual constants* or just *constants*.
- Individual variables (x, y, z, etc.) can be *bound* or *free*.
 - A variable is bound just in case it falls within the scope of a quantifier expression containing the same variable letter. For instance, in the sentence $\exists xFx$ the variable x is bound by the existential quantifier expression $\exists x$. But in $\exists yFx$ the variable x is not bound.
 - A free variable is one which is not bound by any quantifier.
- An *open sentence* is a sentence which contains free variables but is otherwise an ordinary sentence of Predicate logic. An open sentence is the result of removing a quantifier from a sentence containing variables which were previously bound by that quantifier. For instance, $\forall x(Fx \wedge Gx)$ is a quantified sentence, and $Fx \wedge Gx$ is the corresponding open sentence. An open sentence is not a fully grammatical sentence of Predicate Logic.

Quantifier Scope

Compare:

- (1) If everybody likes bananas, then Peter is happy.
- (2) Everybody is such that if he/she likes bananas, then Peter is happy.

These are very different claims: consider a scenario where Steve, and nobody else, likes bananas, and Peter is not happy. Then the antecedent of (1) is false, so (1) comes out true (by the truth table for \rightarrow , which we use to translate 'if...then...'). But sentence (2) seems to be false in this situation: there is somebody, namely Steve, for whom it is *not* the case that if he likes bananas then Peter is happy.

Translating (1) and (2), we get:

- (1') $\forall xBx \rightarrow Ha$
- (2') $\forall x(Bx \rightarrow Ha)$

The difference between our two original sentences is here captured as a difference in the scope of the quantifier. Quantifiers, like negation, always take the narrowest possible scope. So in (1') the $\forall x$ has narrow scope, governing only the antecedent of the conditional. In (1'), the \rightarrow has the widest

scope; the scope of the \rightarrow is the whole sentence. But in (2') the $\forall x$ has the widest scope; here, the scope of the $\forall x$ in (2') is the whole sentence.

Again compare:

(3) $\forall x(Fx \wedge Gx)$

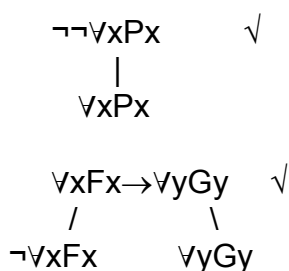
(4) $\forall x Fx \wedge Gx$

Sentence (3) is a fully grammatical sentence which says that everything is F and G, whereas (4) is not a fully grammatical sentence, because (4) contains a free variable – there is no quantifier governing the variable x which appears at the end. This is because the scope of the $\forall x$ is just the first conjunct.

Trees for Predicate logic

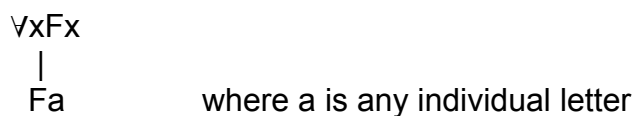
Just as in Sentential logic, we can use truth trees for Predicate logic to test for consistency, tautology, contradiction and validity, and to exhibit models and counterexamples. All the rules we learned for Sentential trees also apply to Predicate trees.

In order to know which rule to apply to a sentence of Predicate, we need to find the connective or quantifier expression with the widest scope. When we do this and it transpires that one of the familiar rules is applicable, because one of the familiar connectives has the widest scope, then we just apply that rule as usual.



However, where a quantifier expression or a negated quantifier expression has the widest scope, we will need some new rules.

THE RULE FOR \forall



Justification for this rule: if everything is F, then whatever individual you pick, that individual is F.

The rule for \forall is called Universal Instantiation. “Instantiating” a quantified sentence is the process of removing the quantifier and replacing every occurrence of the variable letter in the remaining open sentence with an individual letter.

Note that it *must* be the same individual letter throughout. To instantiate $\forall x(Fx \rightarrow Gx)$, we must write $Fa \rightarrow Ga$, or $Fb \rightarrow Gb$, etc.. We would *never* write $Fa \rightarrow Gb$.

Note also that $\forall xFx$ is NOT “used up” when the Universal Instantiation rule is applied. We can use it again, for any other individual letter. Universally quantified sentences are the *only* sentences that do not get used up when a rule is applied to them. Instead of just ticking off a universally quantified sentence when we apply the rule for \forall (which would look like the sentence was finished with), we write *a tick with a letter next to it*, to show that we have instantiated the sentence using that letter:

$$\begin{array}{l} \forall xFx \quad \checkmark^a \\ | \\ Fa \end{array}$$

If we later instantiate it again using another letter, we will give the sentence another tick with the second letter next to it:

$$\begin{array}{l} \forall xFx \quad \checkmark^a \checkmark^b \\ | \\ Fa \\ | \\ Fb \end{array}$$

Consider the following argument:

Everyone is mortal
Therefore David is mortal

We can show that it is valid using a tree. Translating it gives us: $\forall xMx \vdash Ma$
And we begin the tree:

$$\begin{array}{l} \forall xMx \\ \neg Ma \end{array}$$

Now we can instantiate $\forall xFx$ with any individual letter. We *could* pick a new one, say c , and build onto our tree as follows:

$$\begin{array}{l} \forall xMx \quad \checkmark^c \\ \neg Ma \\ | \\ Mc \end{array}$$

But this isn't getting us anywhere; the branch obviously isn't going to close if we keep doing this. However, the branch will close immediately if we instantiate $\forall xMx$ using the individual letter which already appears in our tree, namely a :

$$\begin{array}{l} \forall xMx \quad \checkmark^a \\ \neg Ma \\ | \\ \underline{Ma} \end{array}$$

When instantiating universally quantified sentences, we should always use 'old' individual letters if we can, i.e. individual letters that already appear in the tree.

But sometimes this won't be possible. Consider using a tree to test whether the sentence $\forall x(Fx \wedge \neg Fx)$ is a contradiction. The tree begins:

$$\forall x(Fx \wedge \neg Fx)$$

But now we need to instantiate, and there are no 'old' individual letters around for us to use. So, *in these special circumstances*, we introduce a new individual letter, i.e. one which has not appeared in the tree so far:

$$\begin{array}{c} \forall x(Fx \wedge \neg Fx) \quad \checkmark^c \\ | \\ Fc \wedge \neg Fc \end{array}$$

Then we proceed to apply the rule for \wedge , as usual:

$$\begin{array}{c} \forall x(Fx \wedge \neg Fx) \quad \checkmark^c \\ | \\ Fc \wedge \neg Fc \quad \checkmark \\ | \\ Fc \\ \hline \neg Fc \end{array}$$

Every branch is closed, so there is no way for the starting sentence to be true, so it is a contradiction.

Note that in order to know whether the rule for \forall is the correct rule to apply to a sentence, we need to know the *scope* of the quantifier. Remember our two sentences:

$$(1') \forall x Bx \rightarrow Ha$$

$$(2') \forall x (Bx \rightarrow Ha)$$

To (1'), we would apply the rule for the connective with the widest scope, \rightarrow :

$$\begin{array}{c} \forall x Bx \rightarrow Ha \quad \checkmark \\ / \quad \backslash \\ \neg \forall x Bx \quad Ha \end{array}$$

But in (2'), the expression which has the widest scope is the universal quantifier, so we apply the rule for \forall :

$$\begin{array}{c} \forall x (Bx \rightarrow Ha) \quad \checkmark^a \\ | \\ Ba \rightarrow Ha \end{array}$$

THE RULE FOR \exists

$$\begin{array}{l} \exists xFx \\ | \\ Fc \end{array}$$

where c is a 'new' individual letter (i.e. one which has not appeared in the tree so far)

Justification for this rule: if something is F , then we can give it a name, c , and say that c is F .

This rule is called Existential Instantiation. Just like in the universal case, we instantiate by removing the quantifier and replacing every occurrence of the variable letter in the remaining open sentence with an individual letter. (Again, it *must* be the same individual letter throughout.)

Note that once we have applied the rule for an existentially quantified sentence, we *do* tick it off as usual. *Universally* quantified sentences are the only ones which aren't fully used up when a rule is applied to them.

It is important to remember to use a new name when applying the Existential Instantiation rule. Consider the argument:

There is someone such that if he/she is happy then he/she is rich.
Poppy is happy
 Therefore, Poppy is rich

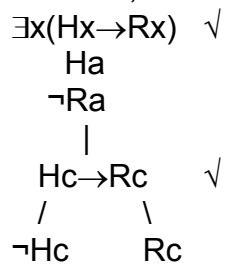
This is clearly invalid. Translating, we get: $\exists x(Hx \rightarrow Rx), Ha \vdash Ra$
 And we begin our tree as follows:

$$\begin{array}{l} \exists x(Hx \rightarrow Rx) \\ Ha \\ \neg Ra \end{array}$$

Now suppose we allow ourselves to instantiate the first line using the old individual letter, a :

$$\begin{array}{l} \exists x(Hx \rightarrow Rx) \quad \checkmark \\ Ha \\ \neg Ra \\ | \\ Ha \rightarrow Ra \quad \checkmark \\ / \quad \backslash \\ \underline{\neg Ha} \quad \underline{Ra} \end{array}$$

We've just "proved" that the argument is valid! This shouldn't be possible, because the argument is *not* valid. Had we stuck to using a new name when we instantiated, this would not have happened:



This tree is finished and has open branches, so we have shown that the argument we started with is invalid.
 (NB: don't worry yet about constructing counterexamples for Predicate arguments.)

How do we know the tree is finished? Because the only sentences on the open branches which have not been ticked off are either atomic sentences or negated atomic sentences. There are no rules we can apply to these sentences.

Supplementary reading

Colin Howson, *Logic With Trees*
 Chapter 6, sections 1-3 and Chapter 8, section 1