

More on quantifier scope

Some sentences contain more than one quantifier, and the ranges of the quantifiers *overlap*. For instance:

Everyone loves everyone: $\forall x\forall yLxy$

Someone loves someone: $\exists x\exists yLxy$

In each of the above two cases, the scope of the first quantifier is the whole sentence, and the scope of the second is the whole sentence minus the first quantifier.

How do we know this? Well, quantifiers want to take the narrowest scope possible. For the second quantifier, the smallest 'chunk' which it can take as its scope is the open sentence Lxy . The first quantifier cannot take as its scope only the second quantifier, since this does not form a 'chunk' of a sentence.

What exactly counts as a 'chunk'? Consider the sentence:

$\forall x\exists y(Lxy \rightarrow Fa)$

When we are determining scope, we start by looking for connectives or quantifiers with the smallest scope. In Predicate Logic, the 'chunks' which are in the scope of these smallest-scope connectives or quantifiers will be atomic sentences or open sentences. In this case, the only connective or quantifier that deals only with atomic sentences or open sentences is the \rightarrow .

When we look for the quantifier or connective with the next narrowest scope, we cannot separate any 'chunks' that have been created already. The part in the brackets must be treated as one chunk from now on.

The existential quantifier can take this chunk as its scope, making it the quantifier with the next narrowest scope.

Finally, we come to the universal quantifier. We must now treat the existential quantifier together with the part in the brackets as a single chunk, so the universal takes the whole of this chunk in its scope, making its scope the whole sentence.

(Note that the scope of a connective or quantifier conventionally includes the connective or quantifier itself.)

When we have overlapping quantifier scope, we *have* to use different variable letters in our quantifiers, or it will not be clear which variables are bound by which quantifiers.

More difficult cases of multiple generality

“Everyone loves someone”:

We will need to use two quantifiers, an existential and a universal, to translate this sentence. But it is ambiguous. There are two possible readings:

1. Everybody loves somebody or other: $\forall x\exists yLxy$
2. There is some one person whom everybody loves: $\exists x\forall yLyx$

Using one-place and two-place predicates together:

Some boy loves some girl: $\exists x(Bx\wedge\exists y(Gy\wedge Lxy))$

Some boy loves every girl: $\exists x(Bx\wedge\forall y(Gy\rightarrow Lxy))$

Every boy loves some girl: $\forall x(Bx\rightarrow\exists y(Gy\wedge Lxy))$

Every boy loves every girl: $\forall x(Bx\rightarrow\forall y(Gy\rightarrow Lxy))$

Using two two-place predicates:

There is someone who is admired by everyone (s)he meets

$\exists x\forall y(Myx\rightarrow Ayx)$

A few more examples

Nobody loves anybody: $\neg\exists x\exists yLxy$

Everyone is loved by themselves if they are loved by anyone: $\forall x(\exists yLyx\rightarrow Lxx)$

Every logician's hat is a philosopher's hat: $\forall x[\exists y(Ly\wedge Hxy)\rightarrow\exists z(Pz\wedge Hxz)]$

(Hxy: x is a hat belonging to y)

Every place lies between Paris and some place: $\forall x\exists yBxpy$

(Bxyz: x lies between y and z)

Applying tree rules to sentences involving multiple generality

This is just like any other application of the rules. The first thing to do is identify the correct rule to use. Check to see if you have a quantifier at the start of the sentence whose scope is the whole sentence. (NB: if the quantifier is inside a bracket it will not have the whole sentence as its scope.) If so, apply the rule for that quantifier. Or apply the appropriate negated quantifier rule if there is a negation in front of the quantifier.

Examples:

$\forall x\exists yRxy$ Apply the rule for \forall
Do not apply the rule for \exists

$\neg\forall x\exists yRxy$ Apply the rule for $\neg\forall$
Do not apply the rule for \exists

$\exists x\forall y(Fx\rightarrow Fy)$ Apply the rule for \exists
Do not apply the rule for \forall

$\neg\exists x\exists y(Fx\rightarrow Fy)$ Apply the rule for $\neg\exists$
Do not apply the rule for \exists

And of course:

$\forall x \exists y Rxy \rightarrow Fa$ Apply the rule for \rightarrow
 Do not apply the rule for \forall or the rule for \exists

$\neg(\forall x \exists y Rxy \wedge \exists z Fx)$ Apply the rule for \wedge
 Do not apply the rule for $\neg\forall$, the rule for \forall or the rule for \exists

Negated quantifier rules and multiple generality:

The negated quantifier rules are applied just as before. We exchange the quantifier for one of the other kind, and push the negation sign through the quantifier, thus:

$$\begin{array}{c} \neg \exists x \exists y (Fx \rightarrow Fy) \quad \checkmark \\ | \\ \forall x \neg \exists y (Fx \rightarrow Fy) \end{array}$$

Note that although there is another quantifier around, nothing happens to it or to the variables it binds. Similarly:

$$\begin{array}{c} \neg \forall x \exists y Rxy \quad \checkmark \\ | \\ \exists x \neg \exists y Rxy \end{array}$$

Instantiation rules and multiple generality

The rules for \forall and \exists (instantiation rules) are also applied just as before.

Applying the rule for \forall :

Remove the initial quantifier and uniformly replace all occurrences of the variable which were previously bound by that quantifier with an individual letter (using an old individual letter if possible).

$$\begin{array}{c} Fa \\ \forall x \exists y Rxy \quad \checkmark^a \\ | \\ \exists y Ray \end{array}$$

Again, just ignore the fact that there are other quantifiers and variables around.

Applying the rule for \exists :

Remove the initial quantifier and uniformly replace all occurrences of the variable which were previously bound by that quantifier with a new individual letter.

$$\begin{array}{c} Fa \\ \exists x \exists y (Fx \rightarrow Fy) \quad \checkmark \\ | \\ \exists y (Fc \rightarrow Fy) \end{array}$$

Note that we never instantiate two quantifiers at once. This kind of mistake is 'safe' – it won't let you 'prove' anything false (provided you obey the restrictions on the letters used to instantiate that you would have obeyed had you applied the rules in turn) – but nevertheless it is still a mistake.

Other kinds of 'safe' mistake include applying a rule for a negated quantifier, when it does not appear at the beginning of the sentence. The following is incorrect although safe:

$$\begin{array}{l} \forall x(Fx \rightarrow \neg \exists y Gy) \quad \checkmark \\ | \\ \forall x(Fx \rightarrow \forall y \neg Gy) \end{array}$$

What's wrong with safe mistakes?

The tree rules are designed to ensure clarity, and mistakes of these kinds render a tree proof less clear. Remember that trees are not rough workings but proofs, and that we must follow the construction rules strictly if they are to serve this purpose.

Also, allowing oneself to make safe mistakes can sometimes lead to unsafe mistakes. If we forget that rules for negated quantified sentences can only be applied when the negated quantifier appears at the beginning of a sentence, we may allow ourselves to apply a rule to a *non*-negated quantifier which is not at the beginning of its sentence. This is unsafe. Consider the (invalid) argument:

Everybody loves somebody or other

Therefore there is somebody whom everybody loves.

Translating, we get: $\forall x \exists y Lxy \vdash \exists x \forall y Lyx$

And we draw our tree:

$$\begin{array}{l} \forall x \exists y Lxy \quad \checkmark \\ \neg \exists x \forall y Lyx \quad \checkmark \\ | \\ \forall x \neg \forall y Lyx \quad \checkmark^c \\ | \\ \forall x Lxc \quad \checkmark^d \quad \text{(a mistake: instantiating the existential in the first} \\ \quad \text{line, using c)} \\ | \\ \neg \forall y Lyc \quad \checkmark \\ | \\ \exists y \neg Lyc \quad \checkmark \\ | \\ \neg Ldc \\ | \\ \underline{Ldc} \end{array}$$

We've 'proved' that the argument is valid!

Predicate Trees as a Partial Decision Procedure

Once we introduce multiple generality, there will be some consistent sets of sentences such that you cannot use trees to *prove* that they are consistent (and correspondingly some invalid arguments such that you cannot use trees to *prove* that they are invalid).

Are these sentences consistent?

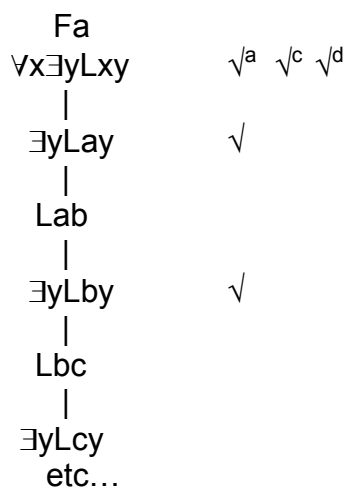
{Adam is friendly. Everyone loves someone or other.}

Translating: $\{Fa, \forall x\exists yLxy\}$

Fx : x is friendly

Lxy : x loves y

a : Adam



To prove that our starting set is consistent, we would need to show that we've *applied every rule that we can*. (Exception: the rule for \forall . Remember that with regard to a universally quantified sentence we need only show that we have instantiated it with every old name on its branch or with a new name if there is no old name available.)

The tree we just looked at will *never* be finished in this sense. Infinitely many new names will be introduced onto the branch, and to complete the tree we would have to instantiate the universally quantified sentence on the second line with all of these names. But it is impossible to perform infinitely many instantiations!

A **decision procedure** (or 'effective procedure') for deciding whether P is true (e.g. whether an argument is valid) is a procedure that a machine could execute, i.e. a procedure requiring no intuition or insight, which is guaranteed to give you a definite verdict as to whether or not P is true within a finite amount of time.

In Sentential Logic, trees provide a decision procedure for deciding whether or not an argument is valid and for deciding whether or not a set of sentences is consistent.

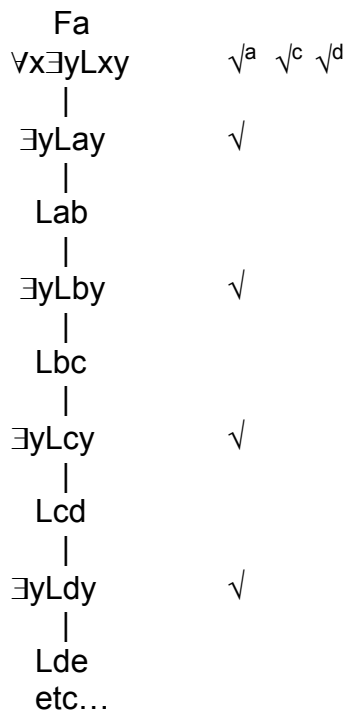
But in Predicate Logic, trees only provide a **partial decision procedure** for validity and consistency. A partial decision procedure is like a decision procedure, except that it is not guaranteed to give you a verdict within a finite time as to whether P is true. But it does guarantee that *if* P is true, then it will tell you so in a finite time. It's just that it doesn't guarantee you an answer within a finite time if P is false. Using trees for Predicate Logic, we know that *if* the verdict is 'valid' or 'inconsistent', the tree test will deliver that result in a finite time. But if the verdict is 'invalid' or 'consistent', the tree method is not guaranteed to give you an answer in a finite amount of time.

If an argument is valid, then when you construct the tree for it, all the branches will close (giving you a proof of validity). However, if the argument is invalid, it is not guaranteed that when you construct a tree for it will eventually get a finished tree with open branches (i.e. you are not guaranteed to get a proof of invalidity).

Consider again the (uncompletable) tree above. Looking at this can help you to see that the set is consistent. You can see that however many new names get introduced and however many times we instantiate the second line using these new names, the branch is never going to close. You can use insight to see that the branch never closes.

Infinite trees and models

The uncompletable tree that we've considered can still be used to provide a model for the starting set. Recall that a model in Predicate Logic consists of a *domain* and an *interpretation*. Things will be clearer if we extend our tree a little further:



In this case, we will have a model with an infinite domain and an infinitely long interpretation. This makes it too long to write down in full, but we can abbreviate! We begin with the domain:

$$D = \{a, b, c, d, e, \dots\}$$

Then, starting with the sentences which actually appear on the branch, we assign values to atomic sentences:

$$I(Fa) = T$$

$$I(Lab) = T$$

$$I(Lbc) = T$$

$$I(Lcd) = T$$

$$I(Lde) = T$$

...

Although we haven't written down all the sentences which appear on our (infinite) branch, we've done enough to make it obvious how to continue.

Finally, we arbitrarily assign values to the relevant atomic sentences which don't appear on the branch:

$$I(Fb) = F$$

$$I(Fc) = F$$

$$I(Fd) = F$$

$$I(Fe) = F$$

...

$$I(Lba) = F$$

$$I(Laa) = F$$

$$I(Lbb) = F$$

$$I(Lcb) = F$$

$$I(Lcc) = F$$

$$I(Ldc) = F$$

$$I(Ldd) = F$$

$$I(Led) = F$$

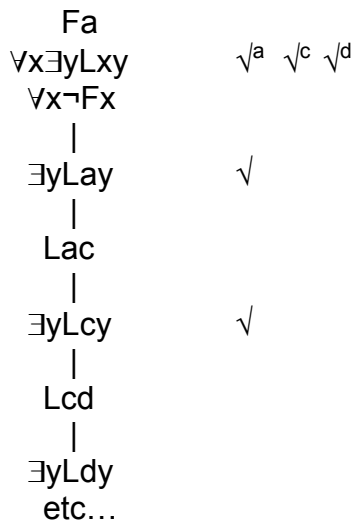
$$I(Lee) = F$$

...

Again, although we haven't mentioned every name which appears on the branch, we have done enough to make it obvious how to continue.

Is the tree method even a partial decision procedure?

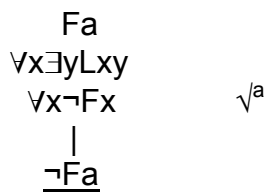
Consider the following:



The starting set is clearly inconsistent, yet the tree will never finish, so we will never have a proof of inconsistency. So how can we claim that whenever a set is inconsistent the tree method will show that it is inconsistent within a finite time?

The answer is that there is one more rule to remember which deals with such cases. If there is a sentence on a branch such that applying the rule for that sentence will close the branch, we must apply it immediately. (If there are several sentences like this, it doesn't matter which you choose.)

So what we should have done was:



Further reading

On trees with infinite branches: Howson, chapter 6 section 2.

On models and counterexamples: Howson, chapter 8, section 2.