More on Truth Trees for Predicate Logic

8

8–1. CONTRADICTIONS, LOGICAL TRUTH, LOGICAL EQUIVALENCE, AND CONSISTENCY

In this section we are going to see how to apply the truth tree method to test predicate logic sentences for some familiar properties. This will be little more than a review of what you learned for sentence logic. The ideas are all the same. All we have to do is to switch to talking about interpretations where before we talked about lines of a truth table.

Let's start with logical contradiction. In sentence logic we say that a sentence is a contradiction if and only if it is false in all possible cases, where by "possible cases" we mean assignments of truth values to sentence letters—in other words, lines of the sentence's truth table. Recharacterizing the idea of a possible case as an interpretation, we have

> A closed predicate logic sentence is a *Contradiction* if and only if it is false in all of its interpretations.

The truth tree test for being a contradiction also carries over directly from sentence logic. The truth tree method is guaranteed to find an interpretation in which the initial sentence or sentences on the tree are true, if there is such an interpretation. Consequently

> To test a sentence, **X**, for being a contradiction make **X** the first line of a truth tree. If there is an interpretation which makes **X** true, the tree method

will find such an interpretation, which will provide a counterexample to **X** being a contradiction. If all branches close, there is no interpretation in which **X** is true. In this case, **X** is false in all of its interpretations; that is, **X** is a contradiction.

Here is a very simple example. We test '(∃x)(Bx & ~Bx)' to see whether it is a contradiction:

√1	(∃x)(Bx & ~Bx)	*S* (The sentence being tested)
√2	(Ba & ~Ba)	1, ∃, **New name**
3	Ba	2, &
4	~Ba	2, &
	×	

The sentence is a contradiction.

The idea of a logical truth carries over from sentence logic in exactly the same way. In sentence logic a sentence is a logical truth if it is true for all possible cases, understood as all truth value assignments. Now, taking possible cases to be interpretations, we say

> A closed predicate logic sentence is a *Logical Truth* if and only if it is true in all of its interpretations.

To determine whether a sentence is a logical truth, we must, just as we do in sentence logic, look for a counterexample—that is, a case in which the sentence is false. Consequently

> To test a predicate logic sentence, **X**, for being a logical truth, make ~**X** the first line of a tree. If there is an interpretation which makes ~**X** true, the tree method will find such an interpretation. In such an interpretation, **X** is false, so that such an interpretation provides a counterexample to **X** being a logical truth. If all branches close, there is no interpretation in which ~**X** is true, and so no interpretation in which **X** is false. In this event, **X** is true in all of its interpretations; that is, **X** is a logical truth.

Again, let's illustrate with a simple example: Test '(∃x)Bx ∨ (∃x)~Bx' to see if it is a logical truth:

√1	~[(∃x)Bx ∨ (∃x)~Bx]	~*S* (The negation of the sentence being tested)
√2	~(∃x)Bx	1, ~∨
√3	~(∃x)~Bx	1, ~∨
a4	(∀x)~Bx	2, ~∃
a5	(∀x)~~Bx	3, ~∃
6	~Ba	4, ∀
7	~~Ba	5, ∀
	×	

The sentence is a logical truth.

The tree shows that there are no interpretations in which line 1 is true. Consequently, there are no interpretations in which the original sentence (the one which we negated to get line 1) is false. So this original sentence is a logical truth.

Notice that I had to introduce a name when I worked on line 4. Line 4 is a universally quantified sentence, and having no name at that point I had to introduce one to start my try at an interpretation. Line 5 is another universally quantified sentence, and when I worked on it, I already had the name 'a'. So I instantiated line 5 with 'a'. At no place on this tree did the **new name** requirement of the rule ∃ apply. This is because at no place on the tree is the entire sentence an existentially quantified sentence. In particular, the sentences of lines 2 and 3 are **negated** existentially quantified sentences, not existentially quantified sentences, so the rule ∃ and the new name requirement do not apply to them.

It's time to talk about logical equivalence. We already discussed this subject in section 3–4, which you may want to review at this point. For completeness, let's restate the definition:

> Two closed predicate logic sentences are *Logically Equivalent* if and only if in each of their interpretations the two sentences are either both true or both false.

Do you remember how we tested for logical equivalence of sentence logic sentences? Once again, everything works the same way in predicate logic. Two closed predicate logic sentences have the same truth value in one of their interpretations if and only if their biconditional is true in the interpretation. So the two sentences will agree in truth value in all of their interpretations if and only if their biconditional is true in all of their interpretations—that is, if and only if their biconditional is a logical truth. So to test for logical equivalence we just test for the logical truth of the biconditional:

> To determine whether the closed predicate logic sentences, **X** and **Y**, are logically equivalent, test their biconditional, **X**≡**Y**, for logical truth. That is, make ~(**X**≡**Y**) the first line of a tree. If all branches close, ~(**X**≡**Y**) is a logical truth, so that **X** and **Y** are logically equivalent. If there is an open branch, **X** and **Y** are not logically equivalent. An open branch will be an interpretation in which one of the two sentences is true and the other false, so that such an open branch provides a counterexample to **X** and **Y** being logically equivalent.

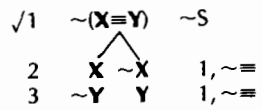
Here is another way in which you can test two sentences, **X** and **Y**, for logical equivalence. Consider the argument "**X**. Therefore **Y**." with **X** as premise and **Y** as conclusion. If this argument is invalid, there is a counterexample, an interpretation in which **X** is true and **Y** is false. Thus if "**X**. Therefore **Y**." is invalid, **X** and **Y** are not logically equivalent, and a

counterexample to the argument is also a counterexample which shows **X** and **Y** not to be logically equivalent. The same goes for the argument "**Y**. Therefore **X**.", this time taking the second sentence, **Y**, as premise and the first sentence, **X**, as conclusion. If this argument is invalid there is a counterexample, that is, an interpretation in which **Y** is true and X is false, and hence again a counterexample to **X** and **Y** being logically equivalent.

Now, what happens if both the arguments "**X**. Therefore **Y**." and "**Y**. Therefore **X**." are valid? In this event every interpretation in which **X** is true is an interpretation in which **Y** is true (the validity of "**X**. Therefore **Y**."), and every interpretation in which **Y** is true is an interpretation in which **X** is true (the validity of "**Y**. Therefore **X**."). But that is just another way of saying that in each interpretation **X** and **Y** have the same truth value. If whenever **X** is true **Y** is true and whenever **Y** is true **X** is true, we can't have a situation (an interpretation) in which one is true and the other is false. Thus, if "**X**. Therefore **Y**." and "**Y**. Therefore **X**." are both valid, **X** and **Y** are logically equivalent:

> To determine whether the closed predicate logic sentences, **X** and **Y**, are logically equivalent, test the two arguments "**X**. Therefore **Y**." and "**Y**. Therefore **X**." for validity. If either argument is invalid, **X** and **Y** are not logically equivalent. A counterexample to either argument is a counterexample to the logical equivalence of **X** and **Y**. If both arguments are valid, **X** and **Y** are logically equivalent.

In fact, the two tests for logical equivalence really come to the same thing. To see this, suppose we start out to determine whether **X** and **Y** are logically equivalent by using the first test. We begin a tree with ~(**X**≡**Y**) and apply the rule ~≡:



Now notice that the left-hand branch, with **X** followed by ~**Y**, is just the way we start a tree which tests the validity of the argument "**X**. Therefore **Y**.". And, except for the order of ~**X** and **Y**, the right-hand branch looks just like the tree which we would use to test the validity of the argument "**Y**. Therefore **X**.". So far as the right-hand branch goes, this order makes no difference. Because we are free to work on the lines in any order, what follows on the right-hand branch is going to look the same whether we start it with ~**X** followed by **Y** or **Y** follow by ~**X**.

In sum, lines 2 and 3 in our tree are just the beginning of trees which test the validity of "**X**. Therefore **Y**." and "**Y**. Therefore **X**.". Thus the completed tree will contain the trees which test the arguments "**X**. Therefore **Y**." and "**Y**. Therefore **X**.". And, conversely, if we do the two trees which test the arguments "**X**. Therefore **Y**." and "**Y**. Therefore **X**." we will have done all the work which appears in the tree we started above, the tree which tests **X**≡**Y** for logical truth. So the two ways of determining whether **X** and **Y** are logically equivalent really involve the same work.

If you did all of exercise 7–4 you have already tested 11 pairs of sentences for logical equivalence! In each of these pairs you tested two arguments, of the form "**X**. Therefore **Y**." and "**Y**. Therefore **X**.". Using our new test for logical equivalence, you can use your work to determine in each of these problems whether or not the pair of sentences is logically equivalent.

Truth trees also apply to test sets of sentences for consistency. Recall from section 9–2 in volume I that a set of sentence logic sentences is consistent if and only if there is at least one case which makes all of the sentences in the set true. Interpreting cases as interpretations, we have

> A *Model* of a set of one or more predicate logic sentences is an interpretation in which all of the sentences in the set are true.
>
> A set of one or more predicate logic sentences is consistent just in case it has at least one model, that is, an interpretation in which all of the sentences in the set are true.
>
> To test a finite set of predicate logic sentences for consistency, make the sentence or sentences in the set the initial sentences of a tree. If the tree closes, there is no interpretation which makes all of the sentences true together (no model) and the set is inconsistent. An open branch gives a model and shows the set to be consistent.

Every truth tree test of an argument is also a test of the consistency of the argument's premises with the negation of the argument's conclusion. An argument is valid if and only if its premises are inconsistent with the negation of the argument's conclusion. In other words, an argument is invalid if and only if its premises are consistent with the negation of its conclusion. Thus one can view the truth tree test for argument validity as a special application of the truth tree test for consistency of sets of sentences. (If you have any trouble understanding this paragraph, review exercise 9–7 in volume I. Everything in that exercise applies to predicate logic in exactly the same way as it does to sentence logic.)

EXERCISES

8–1. Test the following sentences to determine which are logical truths, which are contradictions, and which are neither. Show your work and state your conclusion about the sentence. Whenever you find a counterexample to a sentence being a logical truth or a con-

tradiction, give the counterexample and state explicitly what it is a counterexample to.

a) (∀x)Dx ∨ (∃x)~Dx b)(∀x)Kx & (∃x)~Kx
c) (∀x)Nx ∨ (∀x)~Nx d)(∀x)Jx & (∀x)~Jx
e) (∃x)Bx ∨ (∃x)~Bx f)(∃x)Px & (∃x)~Px
g) [(∀x)Gx ∨ (∀x)Hx] & ~(∀x)(Gx ∨ Hx)
h) (∀x)(Kx ∨ Jx) ⊃ [(∃x)~Kx ⊃ (∃x)Jx]
i) [(∀x)Mx ⊃ (∀x)~Nx] & (∃x)(~Mx & Nx)
j) [(∃x)Hx ⊃ (∀x)(Ox ⊃ Nx)] ⊃ [(∃x)(Hx & Ox) ⊃ (∀x)Nx]
k) (∃x)[~Sx & (Gx ∨ Kx)] ∨ [(∀x)Gx ⊃ (∀x)(Sx ∨ Kx)]
l) [(∀x)Fx ∨ (∀x)Gx] ≡ [(∃x)~Fx & ~(∀x)Gx]

8–2. Use the truth tree method to test the following sets of sentences for consistency. In each case, state your conclusion about the set of sentences, and if the set of sentences is consistent, give a model.

a) (∃x)Px, (∃x)~Px
b) (∀x)Px, (∀x)~Px
c) (∀x)Px, (∃x)~Px
d) (∀x)~Fx, (∀x)Sx, (∃x)[(~Fx ⊃ Sx) ⊃ Fx]
e) (∃x)Gx & (∃x)Qx, ~(∃x)(Gx & Qx)
f) (∀x)(Gx ∨ Qx), ~[(∀x)Gx ∨ (∀x)Qx]
g) (∃x)(Jx ∨ Dx), (∀x)(Jx ⊃ ~Hx), (∀x)(Dx ⊃ Hx),
(∀x)[Jx ≡ (Dx ∨ Hx)]

8–3. Explain the connections among consistency, logical truth, and logical contradiction.

8–4. By examining your results from exercise 7–4(a) through (k), determine which pairs of sentences are logically equivalent and which are not. This is more than an exercise in mechanically applying the test for logical equivalence. For each pair of sentences, see if you can understand intuitively why the pair is or is not logically equivalent. See if you can spot any regularities.

8–2. TRUTH TREES WITH MULTIPLE QUANTIFIERS

In the last chapter I tried to keep the basics in the limelight by avoiding the complication of multiple quantifiers. Multiple quantifiers involve no new rules. But they do illustrate some circumstances which you have not yet seen.

Suppose I asked you to determine whether '(∃x)[Lxa ⊃ (∀y)Lya]' is a

logical truth. To determine this, we must look for a counterexample to its being a logical truth, that is, an interpretation in which it is false. So we make the negation of the sentence we are testing the first line of a tree. Here are the first six lines:

√1	~(∃x)[Lxa ⊃ (∀y)Lya]	~*S*
a 2	(∀x)~[Lxa ⊃ (∀y)Lya]	1, ~∃
√3	~[Laa ⊃ (∀y)Lya]	2, ∀
4	Laa	3, ~⊃
√5	~(∀y)Lya	3, ~⊃
6	(∃y)~Lya	5, ~∀

We begin with the negation of the sentence to be tested. Line 2 applies the rule for a negated quantifier, and line 3 instantiates the resulting universally quantified sentence with the one name on the branch. Lines 4, 5, and 6 are straightforward, first applying the rule ~⊃ to line 3 and then the rule ~∀ to line 5.

But now the rules we have been using all along are going to force on us something we have not seen before. Applying the rule ∃ to line 6 forces us to introduce a new name, say, 'b', giving '~Lba' as line 7. This has repercussions for line 2. When we worked on line 2 we instantiated it for all the names we had on that branch **at that time.** But when we worked on line 6 we got a new name, 'b'. For the universally quantified line 2 to be true in the interpretation we are building, it must be true for all the names in the interpretation, and we now have a name which we did not have when we worked on line 2 the first time. So we must **return** to line 2 and instantiate it again with the new name, 'b'. This gives line 8. Here, with the final two easy steps, is the way the whole tree looks:

√1	~(∃x)[Lxa ⊃ (∀y)Lya]	~*S*
b, a 2	(∀x)~[Lxa ⊃ (∀y)Lya]	1, ~∃
√3	~[Laa ⊃ (∀y)Lya]	2, ∀
4	Laa	3, ~⊃
√5	~(∀y)Lya	3, ~⊃
√6	(∃y)~Lya	5, ~∀
7	~Lba	6, ∃, **New name**
a 8	~[Lba ⊃ (∀y)Lya]	2, ∀
9	Lba	8, ~⊃
√10	~(∀y)Lya	8, ~⊃
	×	

The sentence is a logical truth.

We do not need to work on line 10 because line 7 is the negation of line 9, and the branch thus closes. Indeed, I could have omitted line 10.

There was no way for us to avoid going back and working on line 2 a second time. There was no way in which we could have worked on the

existentially quantified sentence of line 6 before working on line 2 the first time. The sentence of line 6 came from **inside** line 2. Thus we could get line 6 only by instantiating line 2 first. You should always be on the watch for this circumstance. In multiple quantified sentences it is always possible that an existentially quantified sentence will turn up from inside some larger sentence. The existential quantifier will then produce a new name which will force us to go back and instantiate all our universally quantified sentences with the new name. This is why we never check a universally quantified sentence.

Here is another example. We will test the following argument for validity:

Everyone loves someone.	(∀x)(∃y)Lxy
Anyone who loves someone loves themself.	(∀x)[(∃y)Lxy ⊃ Lxx]
Everyone loves themselves.	(∀x)Lxx

a 1	(∀x)(∃y)Lxy	*P*
a 2	(∀x)[(∃y)Lxy ⊃ Lxx]	*P*
√3	~(∀x)Lxx	~C
√4	(∃x)~Lxx	3, ~∀
√5	~Laa	4, ∃
√6	(∃y)Lay	1, ∀
√7	Lab	6, ∃, **New name**
√8	(∃y)Lay ⊃ Laa	2, ∀
√9	~(∃y)Lay | Laa	8, ⊃
b 10	(∀y)~Lay | ×	9, ~∃
11	~Lab	10, ∀
	×	
	Valid	

Getting this tree to come out as short as I did requires some care in choosing at each stage what to do next. For example, I got line 8 by instantiating the universally quantified line 2 with just the name 'a'. The rule ∀ requires me to instantiate a universally quantified sentence with all the names on the branch. But it does not tell me when I have to do this. I am free to do my instantiating of a universally quantified sentence in any order I like, and if I can get all branches to close before I have used all available names, so much the better. In the same way, the rule ∀ requires that I return to instantiate lines 1 and 2 with the new name 'b', which arose on line 7. But the rule doesn't tell me when I have to do that. With a combination of luck and skill in deciding what to do first, I can get all branches to close before returning to line 1 or line 2 to instantiate with 'b'. In this circumstance I can get away without using 'b' in these sentences.

However, in any problem, if I instantiate with fewer than all the names and I have failed to close all the branches, then I **must** return and put the names not yet used into all universally quantified sentences which appear on open branches.

8–3. THREE SHORTCUTS

In general, it is very dangerous to do two or more steps at the same time, omitting explicitly to write down one or more steps which the rules require. When you fail to write down all the steps, it becomes too easy to make mistakes and too hard to find mistakes once you do make them. Also, omitting steps makes it extremely hard for anyone to correct your papers. However, there are three step-skipping shortcuts which are sufficiently clear-cut that, once you are proficient, you may safely use. You should begin to use these shortcuts only if and when your instructor says it is alright to do so.

Suppose you encounter the sentence ~(∀x)(∀y)Lxy on a tree. The rules as I have given them require you to proceed as follows:

√1	~(∀x)(∀y)Lxy	
√2	(∃x)~(∀y)Lxy	1, ~∀
√3	~(∀y)Lay	2, ∃
√4	(∃y)~Lay	3, ~∀
5	~Lab	4, ∃, **New name**

Now look at line 2. You may be tempted to apply the rule ~∀ **inside** the sentence of line 2. In most cases, applying a rule inside a sentence is disastrous. For example, if you should try to instantiate '(∀x)Bx' inside '~[(∀x)Bx ∨ A]', you will make hash of your answer. But in the special case of the rule for negated quantifiers, one can justify such internal application of the rule.

In the example we have started, an internal application of the rule ~∀ gives the following first three lines of the tree:

√1	~(∀x)(∀y)Lxy	
√2	(∃x)~(∀y)Lxy	1, ~∀
3	(∃x)(∃y)~Lxy	2, ~∀

In fact, we can sensibly skip line 2 and simply "push" the negation sign through both quantifiers, changing them both. Our tree now looks like this:

√1	~(∀x)(∀y)Lxy	
2	(∃x)(∃y)~Lxy	1, ~∀, ~∀

We can do the same with two consecutive existential quantifiers or a mixture of quantifiers:

√1	~(∃x)(∃y)Lxy	
2	(∀x)(∀y)~Lxy	1, ~∃, ~∃

√1	~(∃x)(∀y)Lxy	
√2	(∀x)(∃y)~Lxy	1, ~∃, ~∀

√1	~(∀x)(∃y)Lxy	
2	(∃x)(∀y)~Lxy	1, ~∀, ~∃

Indeed, if a sentence is the negation of a triply quantified sentence, you could push the negation sign through all three quantifiers, changing each quantifier as you go.

Why is this shortcut justified? To give the reason in a very sketchy way, the subsentences to which we apply the negated quantifier rule are logically equivalent to the sentences which result from applying the rule. In short, we are appealing to the substitution of logical equivalents. To make all this rigorous actually takes a little bit of work (for the reasons explained in exercise 3–6), and I will leave such niceties to your instructor or to your work on logic in a future class.

Here is another shortcut: Suppose you have a multiple universally quantified sentence, such as '(∀x)(∀y)Lxy', on a tree that already has several names, say, 'a' and 'b'. Following the rules explicitly and instantiating with all the names is going to take a lot of writing:

a, b, 1	(∀x)(∀y)Lxy	
a, b, 2	(∀y)Lay	1, ∀
3	Laa	2, ∀
4	Lab	2, ∀
a, b, 5	(∀y)Lby	1, ∀
6	Lba	5, ∀
7	Lbb	5, ∀

In general, it is not a good idea to skip steps, because if we need to look for mistakes it is often hard to reconstruct what steps we skipped. But we won't get into trouble if we skip steps 2 and 5 in the above tree:

1	(∀x)(∀y)Lxy	(a, a), (a, b), (b, a), (a, b)
2	Laa	1, ∀, ∀
3	Lab	1, ∀, ∀
4	Lba	1, ∀, ∀
5	Lbb	1, ∀, ∀

(In noting on line 1 what names I have used in instantiating the doubly universally quantified sentence '(∀x)(∀y)Lxy', I have written down the **pairs** of names I have used, being careful to distinguish the order in which they occurred, and I wrote them to the right of the line simply because I did not have room on the left.)

In fact, if you think you can get all branches to close by writing down just some of the lines 2 to 5, write down only what you think you will need. But if in doing so you do not get all branches to close, **you must be sure** to come back and write down the instances you did not include on all open branches on which line 1 occurs.

What about using the same shortcut for a doubly existentially quantified sentence? That is, is it all right to proceed as in this mini-tree?

1	(∃x)(∃y)Lxy	
2	Lab	1, ∃, ∃, **New names**

This is acceptable **if** you are very sure that the names you use to instantiate both existential quantifiers are both **new names,** that is, names that have not yet appeared anywhere on the branch.

Our last shortcut does not really save much work, but everyone is tempted by it, and it is perfectly legitimate: You may drop double negations anywhere they occur, as main connectives or within sentences. This step is fully justified by the law of substitution of logical equivalents from sentence logic.

A final reminder: You may use these shortcuts only if and when your instructor judges that your proficiency is sufficient to allow you to use them safely. Also, **do not** try to omit other steps. Other shortcuts are too likely to lead you into errors.

8–4. INFINITE TREES

So far, truth trees have provided a mechanical means for testing arguments for validity and sentences for consistency, logical truth, logical contradiction, or logical equivalence. But if logic were a purely mechanical procedure it could not have enough interest to absorb the attention of hundreds of logicians, mathematicians, and philosophers. However, in one way, the truth tree method is not purely mechanical.

Let's test the sentence '(∀x)(∃y)Lxy' for consistency. That is, let's look for an interpretation which makes it true:

d, c, b, a, 1	(∀x)(∃y)Lxy	*S*
√2	(∃y)Lay	1, ∀
3	Lab	2, ∃, **New name**
√4	(∃y)Lby	1, ∀
5	Lbc	4, ∃, **New name**
√6	(∃x)Lcy	1, ∀
7	Lcd	6, ∃, **New name**
.	.	
.	.	
.	.	

The tree starts with the universally quantified sentence '(∀x)(∃y)Lxy'. At this point the tree has no names, so I pick a name, 'a', and use it to instantiate 1, giving 2. Line 2 is an existentially quantified sentence, so I must instantiate it with a new name, 'b', giving line 3. But having the new name, 'b', I must go back and make 1 true for 'b'. This produces 4, again an existentially quantified sentence, which calls up the new name, 'c'. Now I must go back once more to 1 and instantiate it with 'c', producing the existentially quantified 6 and the new name, 'd', in 7. I am going to have to return to 1 with 'd'. By this time you can see the handwriting on the wall. This procedure is never going to end! The tree is just going to keep on growing. What does this mean? What has gone wrong?

Your immediate reaction may be that the troublesome new name requirement has clearly gummed up the works. The tree keeps on growing without end only because we keep needing to use a new name each time the existentially quantified sentence comes up. It's the new name from the existentially quantified sentence which has to be used to instantiate the universally quantified sentence which produces a new existentially quantified sentence which . . . and so on.

On the other hand, we know that without the new name requirement, the method will not always do its job. So what should we make of this situation?

First, let us understand what this infinite tree represents. It represents an interpretation with infinitely many names. The tree goes on forever, and corresponding to it we have a domain, D = {a,b,c,d, . . .}, and a specification that Lab & Lbc & Lcd & . . . . In other words, each object bears the relation L to the next.

Perhaps you have noticed that we can simplify the interpretation by supposing that there really is only one object to which all of the infinitely many names refer. This gives an interpretation in which there is only one thing, a, such that 'Laa' is true. In this interpretation it is true that for each thing (there is only one, namely a) there is something (namely a itself) such that Laa.

This is the last straw! you may well be saying to yourself. The new name requirement horribly complicates things, in this case by unnecessarily making the tree infinite. In this case the requirement prevents the method from finding the simplest interpretation imaginable which makes the original sentence true!

In fact we could rewrite the rules so that they would find this simple interpretation in the present case. But then the new rules would have some analogue of the new name requirement, an analogue which would provide a similar difficulty in some other problem. Let us say a bit more specifically what the difficulty comes to. In the infinite tree we have seen, it is very easy to tell that the tree will go on forever. And it is easy to figure out what infinite interpretation the infinite tree will provide. But in more complicated problems it will not be so easy. The rub is that there

can be no mechanical way which will apply to all cases to tell us, in some limited number of steps, whether or not the tree will eventually close. There will always be cases in which, even after thousands of pages, we will still not know whether the tree will close in just a few more steps or whether it will go on forever.

One can show that this problem, or some analogue of it, will come up no matter how we write the rules of logic. Indeed, this is one of the many exciting things about logic. The rules can be mechanically applied. But logic will always leave room for insight and ingenuity. For in difficult problems the mechanical rules may never tell you whether the tree will eventually close. In these cases you can find out only by being clever.

Unfortunately, we must stop at this point. But I hope that all of you have been able to get a glimpse of one of the ways in which logic can be exciting. If you continue your study of logic beyond this course, you will come to understand why and how the problem of infinite trees is really a very general fact about all formulations of predicate logic, and you will understand the essential limitations of predicate logic in a much more thorough way.

EXERCISES

8–5. Test the following sentences to determine which are logical truths, which are contradictions, and which are neither. Show your work and state your conclusion about the sentence. Whenever you find a counterexample to a sentence being a logical truth or a contradiction, give the counterexample and state explicitly what it is a counterexample to.

a) (∀x)[(∀y)Py ⊃ Px]
b) (∀x)[(∃y)By ⊃ Bx]
c) (∀x)[(∃y)Cy & ~Cx]
d) (∃x)(∃y)(Lxy & ~Lyx)
e) (∃y)[(∀x)Rxy ⊃ (∀x)Ryx]
f) (∀x)[(∀y)Txy & (∃x)~Tyx]
g) (∃x)(∀y)(Fx⊃Fy)
h) (∀x)(∃y)(Rxy & ~Ryx)

8-6. Use the truth tree method to test the following sets of sentences for consistency. In each case, state your conclusion about the sets of sentences, and if the set of sentences is consistent, give a model.

a) (∃x)(∃y)Lxy, (∃x)(∃y)~Lxy
b) (∀x)(∀y)Lxy, (∀x)(∀y)~Lyx
c) (∀x)(∃y)Kxy, (∃x)(∀y)~Kxy

d) $(\forall x)Ax$, $\sim(\exists x)Bx$, $(\forall x)\{(\exists y)(Ax \,\&\, \sim By) \supset [(\exists y)Ay \supset (\forall y)\sim By]\}$
e) $(\forall x)(\exists y)Mxy$, $(\exists y)(\forall x)\sim Mxy$
f) $(\exists x)(\exists y)(Rxx \,\&\, \sim Ryy \,\&\, Rxy)$, $(\forall x)(\forall y)(Rxy \supset Ryx)$, $(\forall x)(\forall y)\forall z)[(Rxy \,\&\, Ryz) \supset Rxz]$

8–7. Use the truth tree method to determine which of the following are logically equivalent. Give counterexamples as appropriate.

a) $(\forall x)(\forall y)(Px \,\&\, Qy)$ and $(\forall x)Px \,\&\, (\forall x)Qx$
b) $(\exists x)(\exists y)(Px \,\&\, Qy)$ and $(\exists x)Px \,\&\, (\exists x)Qx$
c) $(\forall x)(\forall y)(Px \vee Qy)$ and $(\forall x)Px \vee (\forall x)Qx$
d) $(\exists x)(\exists y)(Px \vee Qy)$ and $(\exists x)Px \vee (\exists x)Qx$
e) $(\forall x)(\forall y)(Px \supset Qy)$ and $(\exists x)Px \supset (\forall x)Qx$
f) $(\exists x)(\exists y)(Px \supset Qy)$ and $(\forall x)Px \supset (\exists x)Qx$
g) $(\exists x)(\forall y)(Px \supset Qy)$ and $(\forall x)Px \supset (\forall x)Qx$
h) $(\forall x)(\exists y)(Px \supset Qy)$ and $(\exists x)Px \supset (\exists x)Qx$
i) $(\forall x)(\exists y)Lxy$ and $(\exists y)(\forall x)Lxy$
j) $(\forall y)[(\exists x)Bx \,\&\, Hy]$ and $(\exists x)Bx \,\&\, (\forall y)Hy$

8–8. Are the following arguments valid? If not, give a counterexample.

a) $\dfrac{(\exists x)(\exists y)[(\forall z)Lzx \supset Lxy]}{(\exists x)(\exists y)Lxy}$

b) $\dfrac{\begin{array}{l}(\forall x)(\exists y)Lxy \\ (\forall x)[(\exists y)Lxy \supset Lxx]\end{array}}{(\forall x)Lxx}$

c) $\dfrac{\begin{array}{l}(\forall x)(\exists y)Lyx \\ (\forall x)(\forall y)(Lxy \supset Txy)\end{array}}{(\forall x)(\exists y)Tyx}$

d) $\dfrac{\begin{array}{c}(\forall x)(\forall y)(\forall z)[(Jxy \,\&\, Jyz) \supset Jxz] \\ (\forall x)(\forall y)(Jxy \supset Jyx)\end{array}}{(\forall x)Jxx}$

e) $\dfrac{(\forall x)(\exists y)Lxy}{(\exists y)(\forall x)Lxy}$

f) $\dfrac{(\forall x)(Cx \supset Ax)}{(\forall x)[(\exists y)(Cy \,\&\, Txy) \supset (\exists y)(Ay \,\&\, Txy)]}$

(All cats are animals. Therefore all tails of cats are tails of animals.)

CHAPTER SUMMARY EXERCISES

Here are items from this chapter for you to review and record in summary:

a) Contradiction
b) Truth Tree Test for Contradictions
c) Logical Truth

d) Truth Tree Test for Logical Truth
e) Logical Equivalence
f) Truth Tree Test for Logical Equivalence
g) Consistency
h) Model
i) Truth Tree Test for Consistency
j) Three Permissible Truth Tree Shortcuts
k) Infinite Trees