# Truth Trees for Predicate Logic: Fundamentals

7

## 7-1. THE RULE FOR UNIVERSAL QUANTIFICATION

You have already learned the truth tree method for sentence logic. And now that you have a basic understanding of predicate logic sentences, you are ready to extend the truth tree method to predicate logic.

Let's go back to the basics of testing arguments for validity: To say that an argument is valid is to say that in every possible case in which the premises are true, the conclusion is true also. We reexpress this by saying that an argument is valid if and only if it has no counterexamples, that is, no possible cases in which the premises are true and the conclusion false. When we were doing sentence logic, our possible cases were the lines of a truth table, and in any one problem there were only finitely many lines. In principle, we could always check all the truth table lines to see if any of them were counterexamples. Often the truth tree method shortened our work. But the trees were really just a labor-saving device. We could always go back and check through all the truth table lines.

Predicate logic changes everything. In predicate logic our cases are interpretations, and there are always infinitely many of these. Thus we could never check through them all to be sure that there are no counterexamples. Now truth trees become much more than a convenience. They provide the only systematic means we have for searching for counterexamples.

Everything we learned about truth trees in sentence logic carries over

to predicate logic. Someone gives us an argument and asks us whether it is valid. We proceed by searching for a counterexample. We begin by listing the premises and the denial of the conclusion as the beginning of a tree. Just as before, if we can make these true we will have a case in which the premises are true and the conclusion false, which is a counterexample and which shows the argument to be invalid. If we can establish that the method does not turn up a counterexample, we conclude that there is none and that the argument is valid.

We have boiled our job down to the task of systematically looking for a case which will make true the initial sentence on a tree. In sentence logic we did this by applying the rules for the connectives '&', 'v', '~', '⊃', and '≡'. These rules broke down longer sentences into shorter ones in all the minimally sufficient possible ways which would make the longer sentences true by making the shorter ones true. Since, in sentence logic, this process terminates in sentence letters and negated sentence letters, we got branches which (if they do not close) make everything true by making the sentence letters and negated sentence letters along them true. In this way you should think of each branch as a systematic way of developing a line of a truth table which will make all the sentences along the branch true.

The tree method for predicate logic works in exactly the same way, with just one change: Each branch is no longer a way of developing a line of a truth table which will make all the sentences along the branch true. Instead, a branch is a way of developing an **interpretation** which will make all the sentences along the branch true. All you have to do is to stop thinking in terms of building a line of a truth table (an assignment of truth values to sentence letters). Instead, start thinking in terms of building an interpretation.

Let's see this strategy in action. Consider the example that got us started on predicate logic, way back in chapter 1:

| | |
|---|---|
| Everybody loves Eve. | (∀x)Lxe |
| Adam loves Eve. | Lae |

We begin our search for an interpretation in which the premise is true and the conclusion is false by listing the premise and the denial of the conclusion as the initial lines of a tree:

```
1   (∀x)Lxe   P
2   ~Lae      ~C
```

We already know quite a bit about any interpretation of these two sentences which makes them both true. The interpretation will have to have something called 'a' and something called 'e', and '~Lae' will have to be true in the interpretation. '~Lae' is already a negated atomic sentence. We cannot make it true by making some shorter sentence true.

But we can make '(∀x)Lxe' true by making some shorter sentences true. Intuitively, '(∀x)Lxe' says that everybody loves Eve. In our interpretation we have a (Adam) and e (Eve). In this interpretation, in this little novel or story of the way the world might be, we can make it true that everybody loves Eve by making it true that Adam loves Eve and making it true that Eve loves Eve. So we extend the branch representing our interpretation with the sentences 'Lae' and 'Lee':

```
a, e 1   (∀x)Lxe   P
     2   ~Lae      ~C
     3   Lae       1, ∀
     4   Lee       1, ∀
           ×
```

And the branch closes! The branch includes both '~Lae' and 'Lae', where the first is the negation of the second. They cannot both be true in an interpretation. We had to include '~Lae' to get an interpretation which makes the conclusion of the argument false. We had to include 'Lae' to get an interpretation which makes '(∀x)Lxe' true. But no interpretation can make the same sentence both true and false. So there is no interpretation which makes lines 1 and 2 true—there is no counterexample to the argument. And so the argument is valid.

Let's talk more generally about how I got lines 3 and 4 out of line 1. Already, when we have just lines 1 and 2, we know that our branch will represent an interpretation with something called 'a' and something called 'e'. We know this because our interpretation must be an interpretation of all the sentences already appearing, and these sentences include the names 'a' and 'e'. Our immediate objective is to make '(∀x)Lxe' true in this interpretation. But we know that a universally quantified sentence is true in an interpretation just in case all its substitution instances are true in the interpretation. So to make '(∀x)Lxe' true in the interpretation we must make 'Lae' and 'Lee' true in the interpretation. This is because 'Lae' and 'Lee' are the substitution instances of '(∀x)Lxe' formed with the interpretation's names, 'a' and 'e'.

Notice that I did something more complicated than simply checking line 1 after working on it and putting the annotation '1,∀' after lines 3 and 4. The rule for the universal quantifier differs in this respect from all the other rules. The other rules, when applied to a "target" sentence, tell us to write something at the bottom of every open branch on which the target sentence appears. When this is done, we have guaranteed that we have made the target sentence true in all possible minimally sufficient ways. We thus will never have to worry about the target sentence again. To note the fact that we are done with the sentence, we check it.

But the rule for the universal quantifier is not like this. First, in applying the rule to a universally quantified sentence, we have to search the branch on which the target sentence appears for names. Then, at the bottom of every open branch on which the target sentence appears, we must instantiate the target sentence with each name which occurs along that branch. To help keep track of which names have already been used to instantiate the target sentence, we list them as we use them.

You might think that when we have thus accounted for all the names on the branch we are done with the target sentence and can check it. But you will see that **new names** can arise after first working on a universally quantified target sentence. In such a case we must come **back** and work on the universally quantified sentence again. Because we must recognize the possibility of having to return to a universally quantified sentence, we never check the sentence as a whole. Instead, we list the names which we have thus far used in the sentence, because once a universally quantified sentence has been instantiated with a given name, we never have to instantiate it with the same name again.

Here is a summary statement of our rule:

> *Rule* ∀*:* If a universally quantified sentence (∀**u**)(. . . **u** . . .) appears as the entire sentence at a point on a tree, do the following to each open branch on which (∀**u**)(. . . **u**. . .) appears. First, collect all the names $\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{s}_3$, . . . that appear along the branch. (If no name appears on the branch, introduce a name so that you have at least one name.) Then write the substitution instances (. . . $\mathbf{s}_1$ . . .), (. . . $\mathbf{s}_2$ . . .), (. . . $\mathbf{s}_3$ . . .), . . . at the bottom of the branch, and write the names $\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{s}_3$, . . . to the left of (∀**u**)(. . . **u** . . .). **Do not** put a check by (∀**u**)(. . . **u** . . .).

Several facets of this rule bear further comment. First, in working on a universally quantified sentence on a given branch, you only need to instantiate it with the names along that branch. If the same universally quantified sentence occurs along a second branch, that second branch calls for use of the names that occur along that second branch. This is because each branch is going to represent its own interpretation. Also, when instructed to write a substitution instance, (. . . **s** . . .), at the bottom of an open branch, you do not need to write it a second time if it already appears.

Next, the rule instructs you to write all of the substitution instances, (. . . $\mathbf{s}_1$ . . .), (. . . $\mathbf{s}_2$ . . .), (. . . $\mathbf{s}_3$ . . .), . . . at the bottom of every open path. But if the path closes before you are done, of course you can stop. Once a path closes, it cannot represent a counterexample, and further additions will make no difference. Thus, in the last example, I could have correctly marked the path as closed after line 3, omitting line 4. You can make good use of this fact to shorten your work by choosing to write down first the substitution instances which will get a branch to close. But don't forget that if the branch does **not** close, you must list **all** the substitution instances.

Finally, listing the names used to the left of (∀**u**)(. . . **u** . . .) is a practical reminder of which names you have already used to instantiate (∀**u**)(. . . **u** . . .). But this reminder is not foolproof because it does not contain the information about which branch the substitution instance appears on. In practice, this isn't a difficulty because in almost every problem your substitution instance will appear on all branches. Indeed, when a universally quantified sentence appears on a branch it never hurts to introduce a substitution instance formed with a name which had not otherwise appeared on that branch.

Let me illustrate how the rule applies when no name appears on a path. At the same time, I will show you how we will write down counterexamples:

```
     A                 1      A        P
  ───────             √2   ~~(∀x)Bx   ~C
  ~(∀x)Bx             a 3    (∀x)Bx   2, ~~
                       4      Ba       3, ∀

Invalid. Counterexample: D = {a}; Ba & A
```

'A' is an atomic sentence letter which we make true in the counterexample. 'A' is **not** a name. So when we get to line 3 and need to instantiate '(∀x)Bx' with all the names in the interpretation we are building, we find that we don't have any names. What do we do? Every interpretation must have at least one thing in it. So when applying the rule ∀ to a universally quantified sentence on a branch which has no names, we have to introduce a name to use. This is the **only** circumstance in which the ∀ rule tells us to introduce a new name. Any name will do. In this example I used 'a'.

Notice how I indicated the counterexample to the argument provided by the open branch. The counterexample is an interpretation which makes everything along the branch true. You read the counterexample off the open branch by listing the names which occur on the branch and the atomic and negated atomic sentences which occur along the branch. The rules have been designed so that these shortest sentences make true the longer sentences from which they came, which in turn make true the still longer sentences from which they came, and so on, until finally everything along the open branch is true.

*EXERCISES*

7–1. Use the truth tree method to test the following arguments for validity. In each problem, state whether or not the argument is valid; if invalid, give a counterexample.

```
a) (∀x)(Kx & Jx)     b) (∀x)(Fx ⊃ Gx)     c) (∀x)(Cx ⊃ Ix)
   ─────────────           ~Ga                ─────────────
        Ka              ─────────────            Ch v Ih
                             Fa

d) A ⊃ (∀x)Mx        e) (∀x)(Bx ⊃ Cx)     f) (∀x)(Ne ≡ Px)
        A                 (∀x)Bx              ─────────────
   ─────────────        ─────────────              Pg
     Mg & Mi              Ca & Cb

g) (∀x)(Kx v Ax)     h) (∀x)(Dx v Gx)     i) (∀x)(Sx ≡ Tx)
       ~Kj              (∀x)(Dx ⊃ Jx)         ─────────────
   ─────────────        (∀x)(Gx ⊃ Jx)           Sb v ~Ta
        Ad              ─────────────
                             Ja

j) ~Tfg v (∀x)Px
   Ph ⊃ (∀x)Qx
   ─────────────
   Tfg ⊃ Qh
```

## 7–2. THE RULE FOR EXISTENTIAL QUANTIFICATION

Consider the argument

```
Somebody is blond.     (∃x)Bx
──────────────────     ──────
Adam is blond.         Ba
```

As we noted in chapter 2, this argument is obviously invalid. If somebody is blond, it does not follow that Adam is blond. The blond might well be somebody else. We will have to keep the clear invalidity of this argument in mind while formulating the rule for existentially quantified sentences to make sure we get the rule right.

Begin by listing the premise and the negation of the conclusion:

```
1  (∃x)Bx    P
2  ~Ba      ~C
```

As in the last example, we already know that we have an interpretation started, this time with one object named 'a'. We also know that '~Ba' will have to be true in this interpretation. Can we extend the interpretation so as also to make '(∃x)Bx' true?

We have to be very careful here. We may be tempted to think along the following lines: An existentially quantified sentence is true in an interpretation just in case at least one of its substitution instances is true in the interpretation. We have one name, 'a', in the interpretation, so we could make '(∃x)Bx' true by making its substitution instance, 'Ba', true. But if we do that, we add 'Ba' to a branch which already has '~Ba' on it, so that the branch would close. This would tell us that there are no counterexamples, indicating that the inference is valid. But we **know** the inference is invalid. Something has gone wrong.

As I pointed out in introducing the example, the key to the problem is that the blond might well be someone other than Adam. How do we reflect this fact in our rules? Remember that in extending a branch downward we are **building** an interpretation. In so doing, we are always free to add new objects to the interpretation's domain, which we do by bringing in **new names** in sentences on the tree. Since there is a possibility that the blond might be somebody else, we indicate this by instantiating our existentially quantified sentence with a **new name**. That is, we make '(∃x)Bx' true by writing 'Bb' at the bottom of the branch, with 'b' a **new name**. We bring the **new name**, 'b', into the interpretation to make sure that there is no conflict with things that are true of the objects which were in the interpretation beforehand.

The completed tree looks like this:

```
√1   (∃x)Bx    P
 2   ~Ba       ~C
 3    Bb       1, ∃, New name
```

Invalid. Counterexample: D = {a,b}; ~Ba & Bb

The open branch represents a counterexample. The counterexample is an interpretation with domain D = {a,b}, formed with the names which appear on the open branch. The open branch tells us what is true about a and b in this interpretation, namely, that ~Ba & Bb.

You may be a little annoyed that I keep stressing '**new name**'. I do this because the **new name** requirement is a very important aspect of the rule for existentially quantified sentences—an aspect which students have a very hard time remembering. When I don't make such a big fuss about it, at least 50 percent of a class forgets to use a new name on the next test. By making this fuss I can sometimes get the percentage down to 25 percent.

Here is the reason for the new name requirement. Suppose we are working on a sentence of the form (∃**u**)(. . . **u** . . .) such as '(∃x)Bx' in our example. And suppose we try to make it true along each open branch on which it appears by writing a substitution instance, (. . . **t** . . .), at the bottom of each of these branches. Now imagine, as happened in our example, that ~(. . . **t** . . .)—or something which logically implies ~(. . . **t** . . .)—already appears along one of these branches. In the example we already had '~Ba'. This would lead to the branch closing when in fact we can make a consistent interpretation out of the branch.

We can always do this by instantiating (∃**u**)(. . . **u** . . .) with a **new name**, say, **s**, a name which does not appear anywhere along the branch. We use this new name in the instantiation (. . . **s** . . .). Then (. . . **s** . . .) can't conflict with a sentence already on the branch, and we are guaranteed not to have the kind of trouble we have been considering.

Not infrequently you get the right answer to a problem even if you don't use a **new name** when instantiating an existentially quantified sentence. But this is just luck, or perhaps insight into the particular problem, but insight which cannot be guaranteed to work with every problem. We want the rules to be guaranteed to find a counterexample if there is one. The only way to guarantee this is to write the rule for existentially quantified sentences with the **new name** requirement. This guarantees that we will not get into the kind of difficulty which we illustrated with our example:

> *Rule* ∃: If an existentially quantified sentence (∃**u**)(. . . **u**. . .) appears as the entire sentence at some point on a tree, do the following to each open branch on which (∃**u**)(. . . **u** . . .) appears: First pick a **new name, s,** that is, a name which does not appear anywhere on the branch. Then write the one substitution instance (. . . **s** . . .) at the bottom of the branch. Put a check by (∃**u**)(. . . **u** . . .).

Why do we always need a new name for an existentially quantified sentence but no new name for a universally quantified sentence (unless there happens to be no names)? In making a universally quantified sentence true, we must make it true for all things (all substitution instances) in the interpretation we are building. To make it true for more things, to add to the interpretation, does no harm. But it also does no good. If a conflict is going to come up with what is already true in the interpretation, we cannot avoid the conflict by bringing in new objects. This is because the universally quantified sentence has to be true for **all** the objects in the interpretation anyway.

We have an entirely different situation with existentially quantified sentences. They don't have to be true for all things in the interpretation. So they present the possibility of avoiding conflict with what is already true in the interpretation by extending the interpretation, by making each existentially quantified sentence true of something new. Finally, since the rules have the job of finding a consistent interpretation if there is one, the rule for existentially quantified sentences must incorporate this conflict-avoiding device.

## EXERCISES

7–2. Test the following arguments for validity. State whether each argument is valid or invalid; when invalid, give the counterexamples shown by the open paths.

```
a)  A ⊃ (∃x)Gx    b)  (∃x)Dx        c)  (∃x)(Px & Qx)
         A            (∃x)~Dx           ____________
    __________        _______             Pa v Qb
        Gb               A

d)    (∃x)Px      e)    A v B
      (∃x)Qx          A ⊃ (∃x)Nx
    __________        B ⊃ (∃x)Nx
     Pm ≡ Qm          __________
                          Ng
```

## 7–3. APPLYING THE RULES

Now let's apply our rules to some more involved examples. Let's try the argument

```
(∀x)Lxe v (∀x)~Lxa
~Lae
__________________
~(∃x)Lxa
```

I am going to write out the completed tree so that you can follow it as I explain each step. Don't try to understand the tree before I explain it. Skip over it, and start reading the explanation, referring back to the tree in following the explanation of each step.

```
√1          (∀x)Lxe v (∀x)~Lxa            P
 2                ~Lae                     P
√3             ~~(∃x)Lxa                  ~C
√4              (∃x)Lxa                    3, ~~
                 /     \
 5  a, e, c (∀x)Lxe   a, e, c (∀x)~Lxa    1, v
 6       Lca               Lca            4, ∃ New Name
 7       Lae              ~Laa            5, ∀
 8       Lee              ~Lea            5, ∀
 9       Lce              ~Lca            5, ∀
          ×                 ×
               Valid
```

We begin by listing the premises and the negation of the conclusion. Our first move is to apply the rule for double negation to line 3, giving line 4. Next we work on line 1. Notice that even though '(∀x)' is the first symbol to appear on line 1, the sentence is **not** a universally quantified sentence. Ask yourself (As in chapters 8 and 9 in volume I): What is the last thing I do in building this sentence up from its parts? You take '(∀x)Lxe' and '(∀x)~Lxa' and form a disjunction out of them. So the main connective is a disjunction, and to make this sentence true in the interpretation we are building, we must apply the rule for disjunction, just as we used it in sentence logic. This gives line 5.

In lines 1 through 4 our tree has one path. Line 5 splits this path into two branches. Each branch has its own universally quantified sentence which we must make true along the branch. Each branch also has '(∃x)Lxa', which is common to both branches and so must be made true along both branches. What should we do first?

When we work on '(∃x)Lxa' we will have to introduce a new name. It is usually better to get out all the new names which we will have to introduce **before** working on universally quantified sentences. To see why, look at what would have happened if I had worked on line 5 before line 4. Looking at the right branch I would have instantiated '(∀x)~Lxa' with 'a' and 'e'. Then I would have returned to work on line 4, which would have introduced the new name 'c'. But now with a new name 'c' on the branch I must go back and instantiate (∀x)~Lxa' with 'c'. To make this sentence true, I must make it true for **all** instances. If a new name comes up in midstream, I must be sure to include its instance. Your work on a tree is more clearly organized if you don't have to return in this way to work on a universally quantified sentence a second time.

We will see in the next chapter that in some problems we cannot avoid returning to work on a universally quantified sentence a second time. (It is because sometimes we cannot avoid this situation that we must never check a universally quantified sentence.) But in the present problem we keep things much better organized by following this practical guide:

> Practical Guide: Whenever possible, work on existentially quantified sentences before universally quantified sentences.

Now we can complete the problem. I work on line 4 before line 5. Line 4 is an existentially quantified sentence. The rule ∃ tells me to pick a **new name**, to use this new name in forming a substitution instance for the existentially quantified sentence, and to write this instance at the bottom of every open path on which the existentially quantified sentence appears. Accordingly, I pick 'c' as my new name and write the instance 'Lca' on each branch as line 6. Having done this, I check line 4, since I have now ensured that it will be made true along each open path on which it appears.

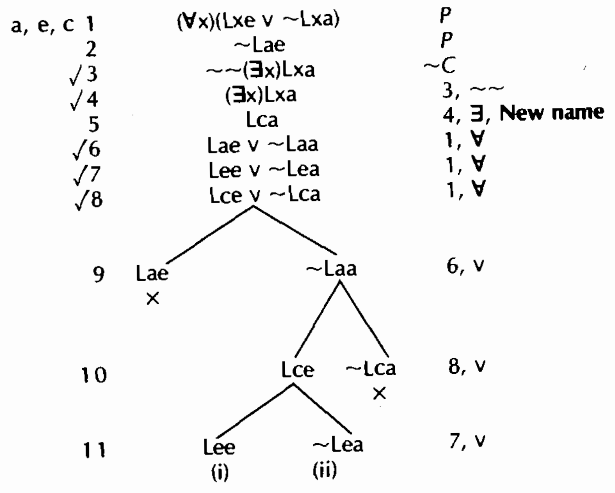
Finally, I can work on line 5. On the left branch I must write substitution instances for '(∀x)Lxe' for all the names that appear along that branch. So below '(∀x)Lxe' I write 'Lae', 'Lee', and 'Lce', and I write the names 'a', 'e', and 'c' to the left of the target sentence '(∀x)Lxe' to note the fact that this sentence has been instantiated with these three names. The branch closes because 'Lae' of line 7 conflicts with '~Lae' on line 2. On the right branch I have '(∀x)~Lxa'. At the bottom of the branch I write its substitution instances for all names on the branch, giving '~Laa', '~Lea', and '~Lca'. Again, I write the names used to the left of the target sentence. '~Lca' is the negation of 'Lca' on line 6. So the right branch closes also, and the argument is valid.

One more comment about this example: The new name requirement did not actually avoid any trouble in this particular case. If I had used either of the old names 'a' or 'e', I would in this case have gotten the right answer. Moreover, the tree would have been shorter. You may be thinking: What a bother this new name requirement is! Why should I waste my time with it in a case like this? But you must follow the new name requirement scrupulously if you want to be sure that the tree method works. When problems get more complicated, it is, for all practical purposes, impossible to tell whether you can safely get away without using it. The only way to be sure of always getting the right answer is to use the new name requirement every time you instantiate an existentially quantified sentence.

Now let's try an example which results by slightly changing the first premises of the last case:

(∀x)(Lxe v ~Lxa)
~Lae
―――――――――
~(∃x)Lxa

Instead of starting with a disjunction of two universally quantified sentences, we start with a universal quantification of a disjunction:



Lines 1, 2, and 3 list the premises and the negation of the conclusion. Line 4 gives the result of applying ~~ to line 3. Looking at line 1, we ask, What was the very last step performed in writing this sentence? The answer: applying a universal quantifier. So it is a universally quantified sentence. But line 4 is an existentially quantified sentence. Our practical guide tells us to work on the existentially before the universally quantified sentence. Accordingly, I pick a **new name**, 'c', and use it to instantiate '(∃x)Lxa', giving me line 5. Now I can return to line 1 and apply the rule ∀. At this point, the names on the branch are 'a', 'e', and 'c'. So I get the three instances of 1 written on lines 6, 7, and 8, and I record the names used to the left of line 1. Lines 9, 10, and 11 apply the v rules to lines 6, 7, and 8. Notice that I chose to work on line 8 before line 7. I am free to do this, and I chose to do it, because I noticed that the disjunction of line 8 would close on one branch, while the disjunction of line 7 would not close on any branches.

We have applied the rules as far as they can be applied. No sentence can be made true by making shorter sentences true. We are left with two open branches, each of which represents a counterexample to the original argument. Let's write these counterexamples down.

The branch labeled (i) at the bottom has the names 'e', 'c', and 'a'. (In principle, the order in which you list information on a branch makes no difference. But it's easiest to read off the information from the bottom of the branch up.) So I indicate the domain of branch (i)'s interpretation by writing D = {e,c,a}. What is true of e, c, and a in this interpretation? The branch tells us that e bears L to itself, that c bears L to e, that a does not bear L to itself, that c bears L to a and that a does not bear L to e. In short, the interpretation is

D = {e,c,a}; Lee & Lce & ~Laa & Lca & ~Lae

To read an interpretation of an open branch, you need only make a list of the branch's names and the atomic and negated atomic sentences which appear along the branch. We use the format I have just indicated to make clear that the names are names of the objects of the domain, and the atomic and negated atomic sentences describe what is true of these objects. Check your understanding by reading the counterexample off the branch labeled (ii). You should get

D = {e,a,c}; ~Lea & Lce & ~Laa & Lca & ~Lae

Notice that neither of these counterexamples as read off the branches constitutes complete interpretations. The branches fail to specify some of the atomic facts that can be expressed with 'L', 'a', 'c', and 'e'. For example, neither branch tells us whether 'Lcc' is true or false. We have seen the same situation in sentence logic when sometimes we had a relevant sentence letter and an open branch on which neither the sentence letter nor its negation appeared as the entire sentence at a point along the branch. Here, as in sentence logic, this happens when a branch succeeds in making everything along it true before completing a selection of truth values for all relevant atomic sentences. In effect, the branch represents a whole group of interpretations, one for each way of completing the specification of truth values for atomic sentences which the branch does not mention. But for our purposes it will be sufficient to read off the interpretation as the branch presents it and call it our counterexample even though it may not yet be a complete interpretation.

*EXERCISES*

7–3. Test the following arguments for validity. State whether each argument is valid or invalid, when invalid, give the counterexamples shown by the open paths.

a) (∃x)(Px ⊃ Qx)
———
~(∀x)(Px & ~Qx)

b) (∃x)Cx
———
~(∃x)~Cx

c) (∃x)Jx v (∃x)Kx
(∀x)~Jx
———
~(∀x)~Kx

d) (∀x)(Px ⊃ Qx)
(∃x)Px
———
~(∀x)~Qx

e) (∀x)(Gx v Hx)
———
~(∃x)~Gx v ~(Ex)~Hx

f) (∃x)Px & (∃x)~Px
———
~(∀x)Px & ~(∀x)~Px

g) ~(∀x)Px ⊃ (∃x)Qx
———
(∃x)~Qx ⊃ ~(∀x)~Px

h) Jq ⊃ (∃x)Kx
(∀x)(Kx ⊃ Lx)
———
Jq ⊃ ~(∀x)~Lx

i) (∃x)Hx & (∃x)Gx
———
~(∀x)~(Hx & Gx)

j) (∃x)~Fx
~(∀x)Fx ⊃ (∀x)~Px
~(∀x)Fx ⊃ (∀x)~Qx
———
~(∀x)(Px v Qx)

## 7–4. NEGATED QUANTIFIED SENTENCES

To complete the rules for quantification, we still need the rules for the negation of quantified sentences. As always, we illustrate with a simple example:

Lae
———
(∃x)Lxe

| | | |
|---|---|---|
| 1 | Lae | P |
| 2 | ~(∃x)Lxe | ~C |

What will make line 2 true? All we need do is to make use of the equivalence rule for a negated existential quantifier, which we proved in section 3–4. (Remember: "Not one is" comes to the same thing as "All are not." If you have forgotten that material, reread the first few paragraphs of section 3–4—you will very quickly remember how it works.) '~(∃x)Lxe' is true in an interpretation if and only if '(∀x)~Lxe' is true in the interpretation. So we can make '~(∃x)Lxe' true by making '(∀x)~Lxe' true. This strategy obviously will work for any negated existentially quantified sentence. So we reformulate the ~∃ rule for logical equivalence as a rule for working on a negated existentially quantified sentence on a tree:

*Rule* ~∃: If a sentence of the form ~(∃**u**)(. . . **u** . . .) appears as the full sentence at any point on a tree, write (∀**u**)~(. . . **u** . . .)at the bottom of every open branch on which ~(∃**u**)(. . . **u** . . .) appears. Put a check by ~(∃**u**)(. . . **u** . . .).

With this rule, we complete our problem as follows:

| | | |
|---|---|---|
| 1 | Lae | P |
| √2 | ~(∃x)Lxe | ~C |
| a, e, 3 | (∀x)~Lxe | 2, ~∃ |
| 4 | ~Lae | 3, ∀ |
| 5 | ~Lee | 3, ∀ |
| | × | |
| | Valid | |

Note that I did **not** use a new name when I worked on line 2. The new name rule does not enter anywhere in this problem because the problem has no existentially quantified sentence as the full sentence at some point on the tree. Consequently, the rule ∃ never applies to any sentence in this problem. Line 2 is the **negation** of an existentially quantified sentence, and we deal with such a sentence with our new rule, ~∃. Line 2 gives line 3 to which the rule ∀ applies.

The story for the negation of a universally quantified sentence goes the same way as for a negation of an existentially quantified sentence. Just as "not one is" comes to the same thing as "all are not," "not all are" comes to the same thing as "some are not." In other words, we appeal to the rule ~∀ for logical equivalence in exactly the same way as we appealed to the rule ~∃ for logical equivalence. Reformulating for application to a negated universally quantified sentence on a tree, we have

*Rule* ~∀: If a sentence of the form ~(∀**u**)(. . . **u** . . .) appears as the full sentence at any point on a tree, write (∃**u**)~(. . . **u** . . .) at the bottom of every open branch on which ~(∀**u**)(. . . **u** . . .) appears. Put a check by ~(∀**u**)(. . . **u** . . .).

Once again, this rule works because ~(∀**u**)(. . . **u** . . .) is equivalent to (∃**u**)~(. . . **u** . . .), as we noted in section 3–4 and as you proved in exercise 3–5.

Here is an example to illustrate the ~∀ rule:

$$\frac{(\exists x)Lxe}{(\forall x)Lxe}$$

| | | |
|---|---|---|
| √1 | (∃x)Lxe | *P* |
| √2 | ~(∀x)Lxe | ~C |
| √3 | (∃x)~Lxe | 2, ~∀ |
| 4 | Lce | 1, ∃, **New name** |
| 5 | ~Lde | 3, ∃, **New name** |

Invalid. Counterexample: D = {d,e,c}; ~Lde & Lce

In this example, note that failure to follow the **new name** rule at step 5 would have incorrectly closed the branch. Also note that we do **not** instantiate line 2 with any of the names. Line 2 is **not** a universally quantified sentence. Rather, it is the **negation** of a universally quantified sentence which we treat with the new rule ~∀.

Now you have all the rules for quantifiers. It's time to practice them.

*EXERCISES*

Before you go to work, let me remind you of the three most common mistakes students make when working on trees. First of all, you must be sure you are applying the right rule to the sentence you are working on. The key is to determine the sentence's main connective. You then apply the rule for that connective (or for the first and second connectives in case of negated sentences). You should be especially careful with sentences which begin with a quantifier. Some are quantified sentences, some are not; it depends on the parentheses. '(∀x)(Px ⊃ A)' is a universally quantified sentence, so the rule ∀ applies to it. It is a universally quantified sentence because the initial universal quantifier applies to the whole following sentence as indicated by the parentheses. By way of contrast, '(∀x)(Lxa ⊃ Ba) & Lba' is not a universally quantified sentence. It is a conjunction, and the rule & is the rule to apply to it.

The second mistake to watch for especially carefully is failure to

instantiate a universally quantified sentence with all the names that appear on the sentence's branch. When a universally quantified sentence appears on a branch, you are not finally done with the sentence until either the branch closes or you have instantiated it with all the names that appear on the branch.

Finally, please don't forget the **new name** requirement when you work on an existentially quantified sentence. When instantiating an existentially quantified sentence, you use only one name, but that name must not yet appear anywhere on the branch.

7–4. Use the truth tree method to test the following arguments for validity. In each problem, state whether or not the argument is valid; if invalid give a counterexample.

| | | | |
|---|---|---|---|
| a1) | $\frac{(\forall x)Px \,\&\, (\forall x)Qx}{(\forall x)(Px \,\&\, Qx)}$ | a2) | $\frac{(\forall x)(Px \,\&\, Qx)}{(\forall x)Px \,\&\, (\forall x)Qx}$ |
| b1) | $\frac{(\exists x)Px \vee (\exists x)Qx}{(\exists x)(Px \vee Qx)}$ | b2) | $\frac{(\exists x)(Px \vee Qx)}{(\exists x)Px \vee (\exists x)Qx}$ |
| c1) | $\frac{(\forall x)Px \vee (\forall x)Qx}{(\forall x)(Px \vee Qx)}$ | c2) | $\frac{(\forall x)(Px \vee Qx)}{(\forall x)Px \vee (\forall x)Qx}$ |
| d1) | $\frac{(\exists x)Px \,\&\, (\exists x)Qx}{(\exists x)(Px \,\&\, Qx)}$ | d2) | $\frac{(\exists x)(Px \,\&\, Qx)}{(\exists x)Px \,\&\, (\exists x)Qx}$ |
| e1) | $\frac{A \supset (\forall x)Px}{(\forall x)(A \supset Px)}$ | e2) | $\frac{(\forall x)(A \supset Px)}{A \supset (\forall x)Px}$ |
| f1) | $\frac{A \supset (\exists x)Px}{(\exists x)(A \supset Px)}$ | f2) | $\frac{(\exists x)(A \supset Px)}{A \supset (\exists x)Px}$ |
| g1) | $\frac{(\forall x)Px \supset A}{(\exists x)(Px \supset A)}$ | g2) | $\frac{(\exists x)(Px \supset A)}{(\forall x)Px \supset A}$ |
| h1) | $\frac{(\exists x)Px \supset A}{(\forall x)(Px \supset A)}$ | h2) | $\frac{(\forall x)(Px \supset A)}{(\exists x)Px \supset A}$ |
| i1) | $\frac{(\forall x)Px}{(\exists x)Px}$ | i2) | $\frac{(\exists x)Px}{(\forall x)Px}$ |
| j1) | $\frac{(\forall x)Px \supset A}{(\forall x)(Px \supset A)}$ | j2) | $\frac{(\forall x)(Px \supset A)}{(\forall x)Px \supset A}$ |
| k1) | $\frac{(\exists x)(Px \supset A)}{(\exists x)Px \supset A}$ | k2) | $\frac{(\exists x)Px \supset A}{(\exists x)(Px \supset A)}$ |

l) $(\forall x)(Px \supset Qx)$
$(\exists x)Px$
$\overline{(\exists x)Qx}$

m) $(\exists x)(Lxa \equiv Lex)$
$(\forall x)Lxa$
$\overline{(\exists x)Lex}$

## *CHAPTER SUMMARY EXERCISES*

Here are the new ideas from this chapter. Make sure you understand them, and record your summaries in your notebook.

a) Rule $\forall$
b) Rule $\exists$
c) New Name Requirement
d) Rule $\sim\forall$
e) Rule $\sim\exists$
f) Reading an Interpretation Off an Open Branch