CHAPTER 11
QUANTIFICATION THEORY
TRANSLATION AND TREES

INTRODUCTION

In our earlier chapters on Translation, Tables, Trees, and Proofs, we were primarily concerned with building a symbolic system that would allow us to decide the validity of a certain, simple kind of argument. We then noted that this system was not sophisticated enough to adequately capture the structure of many arguments. It could not, for example, be used to establish the validity of the argument: "Since all priests are men and all men are rational, all priests are rational." In the chapters on Syllogisms and Venn Diagrams, we developed some techniques for testing these more complex arguments.

We are now going to begin constructing a symbolic system that will not only allow us to test these arguments, it will also allow us to test arguments that are vastly more sophisticated then they are. In the present chapter we will learn how to translate them into symbols and how to construct trees on them to determine their validity. Then, in the next chapter, we will learn how to construct proofs of the ones that are valid.

The basic idea here is to build a system that is an extension of the one we constructed earlier. As a result, it presupposes an understanding of the material contained in the earlier chapters.

CONSTANTS, PROPERTIES, AND RELATIONS

Consider the claim that Bill is tall. This claim ascribes a property, viz., the property of being tall, to an individual. In our earlier chapter on Translation we might have assigned the letter "T" to this claim, and we could still do so, but, for reasons that will become clear shortly, we may now want to represent a bit more of the claims' structure than we did before. Instead of using capital letters to represent entire statements we can also use them to represent only a property. We will use "T," for example, to stand for the property of being tall; and we will use lower case letters from "a" to "u" to represent individuals. (The letters from "a" to "u" are called "constants.") In the present case, we might select the letter "b" to represent Bill. We can then express the claim that Bill is tall by writing: Tb.

Of course, for the symbols "Tb" to be used to represent the claim that Bill is tall, in our translation key we need to specify that "T" represents the property of being tall, and that "b" represents Bill. We do this as follows:

Tx: x is tall.

b: Bill

Notice the use of "x" here. It functions as a placeholder. Unlike "Tb," the expression "Tx" is not a claim. Nor, for that matter, is "x is tall." What the key tells us, in effect, is that we can construct a statement by placing a constant after "T." More specifically, it tells us that we get the claim that Bill is tall by writing: Tb.

Besides ascribing properties to individuals we also sometimes assert that certain relations obtain between individuals. For example, we say that Bill loves Carol. Here we are talking about two individuals, Bill and Carol, and we are asserting that a certain relation, the loving relation, obtains between them. In the system we are constructing, besides using capital letters to represent properties, we also use them to represent relations. Thus, we might use "L" to represent the loving relation. In our key, we set this up as follows:

Lxy: x loves y.
b: Bill
c: Carol

Once we have constructed the key in this way, we can then say that Bill loves Carol by writing: Lbc. On the other hand, we can make the different claim that Carol loves Bill, by writing: Lcb. (Note that we always place the property or relation first, and then put the constants after it.)

With our five connectives and these symbols we can make some complex claims. We can say, for example, that if Bill loves Carol, she loves him, by writing: (Lbc>Lcb). Or, we might say that if Bill doesn't love himself, Carol doesn't love him either, by writing: (-Lbb>-Lcb).

DOMAINS, VARIABLES, AND QUANTIFIERS

If this were all the system we are building could do it wouldn't be very useful. However, it can also represent claims like: "Everybody is tall," and "Someone loves Carol." To be able to represent claims of this sort we need to include a domain in our translation key. The domain tells us what kinds of things are to be included in an "all" or "some" claim. In the sorts of cases we have been discussing our domain would probably be the set of people. However, if we also wanted to say things like, "Fido loves Carol," we would need to widen it to include dogs as well. (If we were to widen the domain in this way, we would then need to include two more letters in our key, one for the property of being a person and another for the property of being a dog. Let's assume, however, that we are only talking about people for now.)

In the translation key, we specify the domain before we identify any properties, relations, or individuals. Thus, our translation key might read:

DOMAIN: {People}

Lxy: x loves y.
Tx: x is tall.
b: Bill
c: Carol

Suppose we want to say that everybody loves Carol. In symbols we represent this as (x)Lxc. The initial part of this formula, namely, (x), is called a "universal quantifier." It really says, "Everything in the domain is such that." However, since our domain has been identified as people, it says that "All people are such that." The second occurrence of "x" -- its occurrence in "Lxc" -- functions just like a pronoun. "Lxc" can be read "he/she/it loves Carol." So the entire formula reads: "Every person is such that he/she loves Carol." In other words, it says that everybody loves Carol.

Instead of saying that everybody loves Carol we might want to say that someone loves her. We do this by using an existential quantifier instead of a universal one. (Ex) is an existential quantifier. It should be read, "there is at least one thing in the domain that is such that," or, since our domain here is {people}, in the present case it says: "Someone is such that." So the formula, (Ex)Lxc, says, "Someone is such that he/she loves Carol" or, more pleasantly expressed, it says that somebody loves Carol.

However, now suppose we wanted to say that Carol loves someone who is tall. We would express this in symbols by writing: (Ex)(Lcx.Tx), i.e., there is someone who is such that Carol loves him/her and he/she is tall. While we would represent the claim that Carol loves everyone who is tall as (x)(Tx>Lcx), or, in other words, every person is such that, if they are tall then Carol loves them.

We could represent the claim that Carol doesn't love anyone who is tall in either of two ways. We might express it in symbols by writing: -(Ex)(Lcx.Tx), i.e., it is not true that there is someone Carol loves and who is tall. Or, we might express it by writing: (x)(Tx>-Lcx), i.e., everyone is such that, if they are tall then Carol doesn't love them.

From our discussion, it should be clear that variables function in three different ways: They occur in the translation key, where they are used as place-holders; they occur as parts of both universal and existential quantifiers; and they function as pronouns in expressions like "Lxc."

THE SCOPE OF QUANTIFIERS

It isn't important that we selected "x" as our variable. We could have chosen any variable. (The letters "w," "x," "y," and "z" are all variables.) The claims (w)Lwc and (x)Lxc, say the same thing. What we cannot do, however, is to write (x)Lwc. It violates a fundamental principle about constructing well-formed formulas in our system. Roughly, the problem here is that the "w" that occurs after "L" is supposed to be functioning like a pronoun, and as such, it must refer back to an antecedent noun or noun phrase. It doesn't refer back to the noun phrase, (x), however, because that expression contains a different variable.

There is a simple principle to use here. Every occurrence of a variable that is not a part of a quantifier must lie within the scope of a quantifier that contains that variable. In the example above, the variable "w," in "Lwc," doesn't lie within the scope of a quantifier which contains "w."

To identify the scope of a quantifier begin looking, immediately after that quantifier, for a binary connective or a left parenthesis. If a binary connective occurs before you see a left parenthesis, all and only those variables in that portion of the expression from the quantifier up to the connective, which are identical with the variable in that quantifier, are within its scope. If the left parenthesis is a part of another quantifier, skip it. Otherwise, continue until you find the right parenthesis that is the mate to the first left one you hit. All and only those variables that are the same as the one inside the quantifier are within that quantifier's scope.

Consider the following examples:

(1) (x)(y)(Pxy>Qzy)
(2) (Ex)-((y)Pxy>Pyx)
(3) ((Ex)(y)Pxy>(z)Paz)

In (1) after the two quantifiers, (x) and (y), we notice that there are several variables that occur, viz., the "x" and "y" after "P," and the "z" and "y" after "Q." Now the scope of the (x) quantifier begins immediately after that quantifier. The first thing we see after this quantifier is another quantifier, (y). Skip it. After this, we see a left parenthesis. The mate to this is the right parenthesis at the end of the expression. Since the only "x" in this portion of the expression occurs between this set of parentheses, it is in the scope of the (x) quantifier. Clearly also, after the (y) quantifier, both of the occurrences of "y" are within the scope of the (y) quantifier. What about the "z" immediately following "Q" however? There is no (z) or (Ez) quantifier whose scope it is within. So (1) is not a legal well-formed formula.

In (2) the first thing we see is an existential quantifier, (Ex). Its scope starts after it. Immediately after it, we see "-." Skip it. We then see a left parenthesis. The right mate to this occurs at the far right end of the expression, after "Pyx." The scope of the (Ex) quantifier goes all the way to this right parenthesis. So both occurrences of x after P are within the scope of this quantifier. What about y? The scope of the (y) quantifier begins right after it and extends only until we hit the ">." So the "y" in "Pyx" is not within the scope of (y) and therefore (2) is also not a well-formed formula.

In (3) all of the occurrences of "x," "y," and "z" that are not parts of any quantifiers are within the scope of the appropriate quantifiers. So (3) is a well-formed formula.

REPRESENTING CLAIMS INVOLVING MULTIPLE QUANTIFIERS

By now it should be clear that we could represent all the different kinds of categorical statements discussed in the chapters on Syllogisms and Venn Diagrams. The statement, "All P are Q" can be represented as, (x)(Px>Qx). While the statement, "No P are Q" can be translated as either -(Ex)(Px.Qx), or (x)(Px>-Qx). Our old friend, "Some P are Q" can be expressed, (Ex)(Px.Qx); while "Some P are not Q" is either, (Ex)(Px.-Qx) or -(x)(Px>Qx). However, we can also use our symbols to represent claims that involve multiple quantifiers. We can say things like, "Everybody loves everybody," (x)(y)Lxy; "Nobody loves anybody," (x)(y)-Lxy; "There is someone who loves everybody," (Ex)(y)Lxy; and "Everybody loves someone," (x)(Ey)Lxy. Using the translation key provided earlier how do you think we would say, "Everybody, who loves anyone, is loved by him or her?"

If you said, (x)(y)(Lxy>Lyx), you were right; but now compare this claim with the claim, "Everyone who loves anybody is loved by somebody." This we can represent as either:

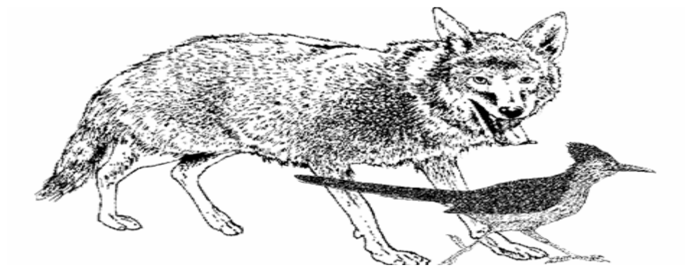
(x)((Ey)Lxy>(Ez)Lzx)   or   (x)(y)(Lxy>(Ez)Lzx)

Notice the difference between these last two cases. The statement, (x)(y)(Lxy>Lyx) tells us that anyone who loves is loved by whomever they love. The claim, (x)((Ey)Lxy>(Ez)Lzx), on the other hand, commits us only to their being loved by someone or other, and not necessarily by the person whom they love. (So the second claim is weaker than the first.)

The trouble with all of this is that it gets very difficult very quickly. We need to understand exactly what is being said in English and then construct a formula that makes the same claim. Typically there are many such formulas and any one of them will do. Suppose, for example, we wanted to say, "Only those who are tall, are loved." We could express this by writing either: (x)((Ey)Lyx>Tx), or (y)((Ex)Lxy>Ty), or (x)(y)(Lxy>Ty), or (x)(-Tx>(y)-Lyx), or (x)(-Tx>-(Ey)Lyx). Just as there are many ways of translating it correctly, however, there are also many ways of expressing it incorrectly. It would be wrong, for example, to translate it: (x)(Tx>(Ey)Lyx). Although it makes no difference which variables we use, as the first two cases hopefully illustrate, where they occur in the formula and how those that aren't part of a quantifier are related to those that are, frequently matters a lot. Thus, (x)((Ey)Lxy>Tx) is a very different claim than, (x)((Ey)Lyx>Tx). The former asserts that all of those who love someone are tall, while the latter makes a different claim. It asserts that all of those who are loved by someone are tall. When we were dealing with translation in the earlier chapter, we pointed out that practice is essential. The same is true here to an even greater extent. Unfortunately, because there are so many different ways of expressing the same claim in symbols, and space is severely limited, we cannot give you a lot of examples. On the computer disks, the practice exercises in this chapter contain only six arguments, each of which has three premises. Each time through, you will be asked to translate two of these arguments. You should try going through the exercises until you have seen all of the problems. While a correct translation is provided, however, even when you know what that answer is you might try formulating the same claim in other ways. Although copying the answer you are given down, and then entering it when you see that problem again will result in a high score, it will be of only limited value when it comes to understanding how to translate.

EXERCISES

Instructions: Using the translation key provided, translate each of the following statements into symbolic notation.



DOMAIN: {Roadrunners & Coyotes}
Cx: x is a coyote.
Rx: x is a roadrunner.
Fxy: x runs faster than y.
c: Wily Coyote

1. Every roadrunner runs faster than some coyote.
2. No coyote runs faster than any roadrunner.
3. If Wily Coyote runs faster than any roadrunner, then he runs faster than any coyote.

USING TREES IN QUANTIFICATION THEORY

In this section we will expand on the tree method developed in Chapter 6 so we can use it to determine whether single statements are quantificationally true, false, or indeterminate, whether a pair of statements is quantificationally equivalent, whether sets of statements are quantificationally consistent or inconsistent, and whether arguments are quantificationally valid or invalid.

The rules for decomposing statements that are negated existentially quantified and negated universally quantified formulas are straightforward. We simply replace the existential quantifier with a universal quantifier, or replace the universal quantifier with an existential quantifier, move the tilde to the other side of this quantifier, and then check off the original statement. This is obviously legitimate since a statement that asserts that it is not the case that something in the domain has a certain characteristic is equivalent to the statement that everything in the domain lacks that characteristic, while the statement that not everything in the domain has a certain characteristic amounts to the statement that something in the domain lacks that characteristic. Thus,

NEGATED EXISTENTIAL QUANTIFIER DECOMPOSITION (-ED**)**

```
n. - (E variable) ♥          √
        .
        .
        .
p. (variable)  -   ♥         n, -ED
```

NEGATED UNIVERSAL QUANTIFIER DECOMPOSITION (-UD)

```
n. - (variable) ♥            √
        .
        .
        .
p. (E variable) ♥            n, - UD
```

Once we have used all the old rules from Chapter 6 on any statements these rules can be used on, and used these two new rules on any statements they can be used on, we will be left with only existentially and/or universally quantified formulas. We can then begin decomposing these remaining statements using two additional new rules.

An existentially quantified statement can be decomposed by first eliminating the existential quantifier and then replacing every occurrence of the variable that occurred in the quantifier in the original statement with a new constant that has never occurred on any branch before. Once we have done this we can check the original statement off. Thus,

EXISTENTIAL QUANTIFIER DECOMPOSITION (ED)

```
n. (E variable) ♥      √
        .
        .
        .
p.      ♥              n, ED
```

where every occurrence of the variable in (E variable) has been replaced by a new constant.

The following example may help illustrate this.

```
1. (Pa > - (x)(y)Qxy)                  √      SM
2.        Pa                                  SM
          /\
3.    - Pa   - (x)(y)Qxy               √      1, >D
4.       *    (Ex) - (y)Qxy            √      3, -ED
5.            - (y) Qby                √      4, ED
6.             (Ey) - Qby              √      5, -ED
7.                - Qbc                       6, ED
```

Why do we require that a new constant be selected when using this rule? If we didn't require this the following tree would show that the set of statements consisting in the set of formulas on lines 1 and 2 below would be inconsistent.

```
1. (Ex)Px                                   √        SM
2. (Ex) - Px                                √        SM
3.  Pa                                               1, ED
4. - Pa             ILLEGAL                          2, ED
     *
```

But clearly it is possible for something in the domain to have a property which something else lacks. The requirement that we pick a new constant makes the move on line 4 above illegal.

UNIVERSAL QUANTIFIER DECOMPOSITION (UD)

Our final rule, Universal Quantifier Decomposition functions much like the rule Existential Quantifier Decomposition, with two very important exceptions. First, once we have eliminated the quantifier, instead of replacing the variable with a constant that has not occurred on that branch before we need to replace it with a constant that has occurred before, and we need to do this for each constant that has occurred on that branch before; and second, we cannot check the original statement off. The reason for this is simple. Our original statement asserts that every object in the domain has the characteristic in question. So the claim implies that this characteristic applies to each and every object in the domain. Thus, the rule can be formulated as follows:

```
n. (variable) ♥
    .
    .                          where the variable occurring in the
    .                          quantifier is replaced by a constant that
p.  ♥          n. UD           has occurred on that branch before for
                               each such constant.
```

Study the following example carefully.

```
1. (- Pa v - (Ex)(- Px &  (Ey) Qxy))                     √      SM
2,              (Pa & Qab)                               √      SM
3.                  Pb                                          SM
4.                  Pa                                          2, &D
5.                  Qab                                         2, &D
                    /\
6.             - Pa     -(Ex)( Px & (Ey) Qxy)            √      1, vD
7.               *        (x) - (Px & (Ey)Qay)           √      6, -ED
8.                          - ( Pa & (Ey)Qay)                   7, UD
9.                          - (Pb  &  (Ey)Qby)                  7, UD
                                  /\
10.                           - Pa      - (Ey)Qay        √      8, - &D
                                *           /\
11.                                  - Pb   - (Ey)Qby    √      9. - &D
12.                                    *       (y) - Qay        10, - ED
13.                                            (y) - Qby        11, - ED
14.                                            - Qaa            12, UD
15.                                            - Qab            12, UD
                                                 *
```

Though the formula on line 13 has not been decomposed, the set being tested is quantificationally inconsistent because every branch has been flowered.

Unfortunately there is a problem with all of this in cases where our domain contains infinitely many constants. Suppose we try to decompose a set that contains only the following very simple formula:

```
1. (x)(Ey)Pxy                                                   SM
2. (Ey)Pay                                               √      1, UD
3  Pab                                                          2, ED
```

Since a new constant ‘b’ has now occurred and we have not yet decomposed line 1 using that constant we need to go back and decompose it using this constant. We then get,

```
        4. (Ey)Pby                                        √       1, UD
```

followed by,

```
        5. Pbc                                                    4, ED
```

etc. This process will never end. Branches of this sort we will refer to as non-terminating branches. Unfortunately, whether a branch is a non-terminating branch or a branch that will eventually terminate is something that is not always easy to determine, and there is no method for constructing a tree that will always give us a result in a determinate number of steps because the system in question is incomplete. Nevertheless, if we proceed according to the following procedure we will at least have a result that will tell us in a great many cases whether the set we are dealing with is quantificationally consistent or inconsistent.

THE PROCEDURE

If at any point in the construction of the tree:

1. All the branches end in flowers then the set being tested is quantificationally inconsistent
2. Any branch is either:
   - (a) A non-terminating branch, or
   - (b) A branch on which every formula is either:
     - (i) A lintel, or
     - (ii) Has been checked off, or
     - (iii) Is a universally quantified formula in which,
       - (α) At least one instance of that formula occurs on that branch, and,
       - (β) For each constant occurring on that branch an instance of that universally quantified formula containing that constant occurs on that branch, then the set being tested is quantificationally consistent.

Step 1: Decompose all formulas that can be decomposed using the rules in Chapter 6, and any formula that results from using those rules which itself can be decomposed using them.

Step 2: Decompose all formulas that can be decomposed using -ED and -UD.

Step 3: Decompose all existentially quantified formulas, and then, if any new formulas are listed, return to step 1.

Step 4: Decompose all universally quantified formulas that have not been decomposed using every new constant that has appeared on that branch. If no constants have appeared on the branch in question select an instance of the universally quantified formula using the constant a.

Step 5: Return to Step 1.

EXAMPLE

```
1.                 (Ex)(y)Pxy                        √       SM
2.                 ((x)(Ey)Pxy > Qa)                 √       SM
3.                 - (Ex)Qx                          √       SM
                        /\
4.         - (x)(Ey)Pxy √     Qa                             2, >D
5.         (x)-Qx            (x)-Qx                          3, -ED
6.         (Ex) - (Ey)Pxy                            √       4, -UD
7.         (y)Pby            (y)Pby                          1, ED
8.         - (Ey)Pcy                                 √       6, ED
9.         (y)-Pcy                                           8, -ED
10.        - Qa               - Qa                           5, UD
11.        - Qb                  *                           5, UD
12.        - Qc                                              5, UD
13.        Pba                                               7, UD
14.        Pbb                                               7, UD
15.        Pbc                                               7, UD
16.        - Pca                                             9, UD
17.        -Pcb                                              9, UD
```

18. -Pcc 9, UD

The set being tested is quantificationally consistent.

Notice how we have proceeded. Step 1 told us to use the rules from Chapter 6 on any formulas we could. Since the formula on line 2 was the only one we could use one of these rules on, we did so. Then we proceeded to step 2 and applied the -ED rule on line 3. This created line 5. Next we used the -UD rule on the formula on the left branch of line 4. We then proceeded to step 3 and decomposed the formula on line 1 on both branches at line 7, and the formula on line 6 on the left branch only at line 8. Step 3 then instructed us to return to step 1. Nothing was relevant here, so we moved on to step 2. This caused line 9. Step 3 was now irrelevant, so we proceeded to step 4, which led to all the remaining formulas. Although the right branch flowered at line 10, the left branch continued to grow until it finally met condition 2(b) above.

EXERCISES

Instructions: Use the tree method to determine whether the following sets of statements are quantificationally consistent or inconsistent.

A. {(x)(y)Pxy, ((Ex) - Pxx v (x) - Qx), Qb}

B. {(x)(Px > Qx), (x)(- Qx v Rx), ((Ex)Px > - (Ex)Rx)}

Instructions: Use the tree method to determine whether the following arguments are quantificationally valid or invalid. Note: An argument is quantificationally valid just in case the set of statements consisting in its premises together with the negation of its conclusion is quantificationally inconsistent.

A. (x)(Px > Qx)
(x)(Qx > (Rx . Sx))
____
(x)(Rx v - Px)

B. (Ex)(y)Pxy
((Ex)Pxx > (y)Qyy)
____
(Ex)(Ey)Qxy

C. (x)(y)(- Pxy v Qxy)
- (Ex)(Ey)Qxy
____
(Ex)(Ey)Pxy