CHAPTER 6
TRUTH TREES

THE PURPOSE OF TRUTH TREES

Although the truth tree technique we will be exploring in this chapter can ultimately be used to obtain answers to the same questions truth tables answer, the tree method is primarily designed to find out whether a set of statements is consistent or inconsistent. Once we have learned to construct trees to find out whether a set of statements is consistent or inconsistent we will see how to use them to answer other kinds of questions we might have.

One great advantage the truth tree method has over truth tables is that trees frequently involve much less work. When we were learning to build tables, we saw that the more atomic statement letters there are in a problem the longer the table needs to be. With trees, as we will discover, this is not necessarily so.

SETTING THE PROBLEM UP

A truth tree always begins with a set of statements. We start by listing each member of the set, one beneath the other. We then number the lines to the far left and justify them "SM," which is an abbreviation for "set member," to the far right. Suppose, for example, we want to test: {(Pv-(Q.-R));----T;-(Av--B);(S>(U.W));---T}. Our tree will begin as follows:

1	(P v - (Q . - R))	SM
2	- - - - T	SM
3	- (A v - - B)	SM
4	(S > (U . W))	SM
5	- - - T	SM

THE ANATOMY OF A TREE

Although this may not look much like a tree, we should view it as one. We should imagine it as upside down, with the trunk beginning on line 1, starting with the first set member, and extending until we have listed all the set members. As we list other formulas beneath the initial set members the tree will gradually grow. Though all of our early trees will only look like trunks coming out of the ground, later we will learn how to build branches on them until they finally really do begin resembling trees.

The trees we will be building won't ever look like Christmas trees. Instead, they will consist in connected two-pronged branching affairs, with or without flowers on the ends of their branches. To see how these trees will look, just imagine the following structure containing formulae along its branches:

```
 |
/\
|  /\
|   | |
*  * /\
     |  /\
     * * |
         /\
         | *
```

This tree contains six branches. Five of these end in flowers. Only the fifth branch from the left has no flower.

The flowering process is extremely important, for it decides whether a set of statements is consistent or inconsistent. What, however, causes a flower?

LINTELS AND FLOWERS

Let's call any atomic statement letter that either is not negated, or is singly negated, "a lintel" (so, P, Q, R, -P, -Q, and -R, are all examples of lintels). And let's refer to any pair of lintels, one of which is the negation of the other (e.g., P and -P), as "a conflicting pair of lintels." Whenever a conflicting pair of lintels occurs on a branch we will place a flower at the end of that branch. Thus, for example, in the following tree every branch has a flower on the end of it.

```
1                 ((P v - Q) v (R v ( - - T . U)))          √      SM
2                 ( - P . Q)                                √      SM
3                 - (R v T)                                 √      SM
4                   - P                                            2,  . D
5                    Q                                             2,  . D
6                   - R                                            3, - vD
7                   - T                                            3, - vD
                     /\
8            (P v - Q) √   (R v ( - - T . U))               √      1,  vD
               /\
9             P   - Q          /\                                  8,  vD
10           *    *          R   (- - T . U)                √      8,  vD
11                           *      - - T                   √      10,  . D
12                                    U                            10,  . D
13                                    T                            11,  - -D
                                      *
```

The farthest branch to the left has a flower because of the conflicting pair of lintels P (on line 9) and -P (on line 4). While the second branch from the left has a flower because of the conflicting pair of lintels -Q (on line 9) and Q (on line 5). The third branch from the left has a flower because of the conflicting pair of lintels R (on line 10) and -R (on line 6). Finally, the farthest branch to the right has a flower because of the conflicting pair of lintels T (on line 13) and -T (on line 7).

If all of the branches of a tree end in flowers we will call that tree "a completely flowering tree." A completely flowering tree tells us the set we are testing is inconsistent.

THE TYPES OF FORMULAS THAT ARE NOT LINTELS

All of this may seem clear enough, but what if a formula is not a lintel and our tree is not completely flowering? It turns out that any formula that is not a lintel must be one among the following nine types of formulas:

1	--♥		
2	(♥. ♦)	6	-(♥. ♦)
3	(♥v♦)	7	-(♥v♦)
4	(♥> ♦)	8	-(♥> ♦)
5	(♥= ♦)	9	-(♥= ♦)

Thus, -(---(P. -Q) v - (R = (T > B))), is of type 7, since its main connective is a tilde and its next main connective is a wedge. While - - - (P. -Q) is of type 1, since it is a doubly negated formula. And the formula, - (R = (T > B)) is of type 9, because its main connective is - and its next main connective is =.

Corresponding to each of these nine types of formulas is a decomposition rule. Each decomposition rule tells us how to break a particular kind of formula up into simpler formulas, formulas that are equivalent to the original. When we are constructing a tree, and we find a formula that isn't a lintel, we simply identify and apply the rule that is applicable to that type of formula. In this way, as our tree develops, we will get to simpler and simpler formulas, until we finally obtain lintels.

In what follows we are going to be presenting the nine decomposition rules, one after the other. After we formulate a new rule, we will present an example that uses that rule, with the earlier ones. We will introduce the easier rules first, and you will find that it is a good idea to try to use these rules before using the more difficult ones. This creates trees that are not as complex as they would otherwise be.

One other thing is worth mentioning. When a new rule is presented, you might want to write it down for future reference.

TILDE TILDE DECOMPOSITION

n.	--♥	√
p.	♥	n, --D

This rule tells us that if a formula is a doubly negated formula, we may write down that formula, without the two tildes, on any later line. The earlier formula is then checked off (n and p are just line numbers).

AN EXAMPLE

1	(P > - (Q = - R))		SM
2	- - - - R		SM
3	- - - R		SM

Let's begin by decomposing line 2. When we apply the present rule on this line, we get. . . .

1	(P > - (Q = - R))		SM
2	- - - - R	√	SM
3	- - - R		SM
4	- - R		2, - -D

In effect, we have now replaced the formula on line 2 with one simpler than it, namely, the one on line 4. Now, however, we can decompose line 4.

1	(P > - (Q = - R))		SM
2	- - - - R	√	SM
3	- - - R		SM
4	- - R	√	2, - -D
5	R		4, - -D

Our first lintel! Unfortunately, we can't flower yet, however, because we don't have a -R to go with R. It's now time for us to decompose line 3.

1	(P > - (Q = - R))		SM
2	- - - - R	√	SM
3	- - - R	√	SM
4	- - R	√	2, - -D
5	R		4, - -D
6	- R		3, - -D
	*		

The last application of the rule causes a flower. Moreover, since all of the branches now have flowers on them, our tree is completely flowering. Therefore, we have shown that the set we are testing is inconsistent.

DOT DECOMPOSITION

n.	(♥.♦)	√
p.	♥	n, .D
p+1.	♦	n, .D

This rule tells us that if a formula is a dot claim, we may write its left side down, followed immediately, on the very next line, by its right side.

AN EXAMPLE

1	(P . - - (Q . R))		SM
2	- - - Q		SM

In this problem we should begin with line 2, because it involves an earlier rule than line 1 does. When we decompose this line we obtain the following:

1	(P . - - (Q . R))		SM
2	- - - Q	√	SM
3	- Q		2, - -D

Next, we have to do line 1. Note that we can't use the Tilde Tilde rule on it because these are not its main connectives. Whenever we select it, the only rule we can use on it is Dot Decomposition.

1	(P . - - (Q . R))	√	SM
2	- - - Q	√	SM
3	- Q		2, - -D
4	P		1, . D
5	- - (Q . R)		1, . D

Notice that the Dot Decomposition rule requires that we write down two later lines. (Many rules resemble Dot Decomposition in this respect.) We can now use our Tilde Tilde rule on line 5.

1	(P . - - (Q . R))	√	SM
2	- - - Q	√	SM
3	- Q		2, - -D
4	P		1, . D
5	- - (Q . R)	√	1, . D
6	(Q . R)		5, - -D

We can now finish the problem by decomposing line 6. (If you haven't already done so, note how we justify the steps to the far right.)

1	(P . - - (Q . R))	√	SM
2	- - - Q	√	SM
3	- Q		2, - -D
4	P		1, . D
5	- - (Q . R)	√	1, . D
6	(Q . R)	√	5, - -D
7	Q		6, . D
8	R		6, . D
	*		

The conflicting lintels on lines 7 and 3 cause a flower. The set is, therefore, inconsistent. Note that we finish using the rule (on line 8) before flowering.

TILDE HORSESHOE DECOMPOSITION

n.	- (♥ > ♦)	√
p.	♥	n, - >D
p+1.	- ♦	n, - >D

This rule tells us that if a formula is a tilde horseshoe claim, we may write its left side down, followed immediately, on the next line, by tilde and then its right side

AN EXAMPLE

1	- (P > - (Q . R))		SM
2	- (R > S)		SM

Although we can choose to decompose either line 1 or line 2, whichever one we decide to do we have to use the Tilde Horseshoe rule on it. Let's begin with line 2.

1	- (P > - (Q . R))		SM
2	- (R > S)	√	SM
3	R		2, - >D
4.	- S		2, - >D

This rule works as it does because -(R>S) is equivalent to (R.-S). Shall we decompose line 1 now?

1	- (P > - (Q . R))	√	SM
2	- (R > S)	√	SM
3	R		2, - >D
4	- S		2, - >D
5	P		1, - >D
6	- - (Q . R)		1, - >D

Notice that on line 6 all we did was to bring down the right side of line 1 and place a tilde in front of it. We can now use our Tilde Tilde rule on 6.

1	- (P > - (Q . R))	√	SM
2	- (R > S)	√	SM
3	R		2, - >D
4	- S		2, - >D
5	P		1, - >D
6	- - (Q . R)	√	1, - >D
7	(Q . R)		6, - - D

That was easy, wasn't it? Now all we can do is decompose line 7. Can you see what is going to happen after we have finished decomposing it?

1	- (P > - (Q . R))	√	SM
2	- (R > S)	√	SM
3	R		2, - >D
4	- S		2, - >D
5	P		1, - >D
6	- - (Q . R)	√	1, - >D
7	(Q . R)	√	6, - -D

8	Q	7, . D
9	R	7, . D

At this point we are finished, because we have decomposed every formula that isn't a lintel, yet the tree has no flowers. Does this mean that the set is consistent? Yes! Whenever any branch on a tree contains only formulas that are lintels or have already been checked off, if that branch is not flowering we have a result. The tree is not completely flowering and the set we are testing is consistent.

TILDE WEDGE DECOMPOSITION

n.	- (♥v♦)	√
p.	- ♥	n, - vD
p+1.	- ♦	n, - vD

This rule tells us that if a formula is a tilde wedge claim, we may write down a tilde, then the formula on its left side, followed on the very next line, by a tilde, and then its right side.

AN EXAMPLE

1	P	SM
2	- ((P . Q) > (R v - S))	SM

In this problem, the only thing we can do is line 2, and the only rule we can use on it is Tilde Horseshoe Decomposition. Do you remember how that rule works?

1	P		SM
2	- ((P . Q) > (R v - S))	√	SM
3	(P. Q)		2, - >D
4	- (R v – S)		2, - >D

Now we need to do either line 3 or line 4. Since the rule for decomposing line 3 is an earlier rule than the one for decomposing line 4, let's do it next.

1	P		SM
2	- ((P . Q) > (R v - S))	√	SM
3	(P . Q)	√	2, - >D
4	- (R v - S)		2, - >D
5	P		3, . D
6	Q		3, . D

It's now time for us to use our new rule and decompose line 4. (The fact that the formulas, -(Rv-S) and (-R.--S), are logically equivalent, explains why this rule works the way it does, if you really wish to know.)

1	P		SM
2	- ((P . Q) > (R v - S))	√	SM
3	(P . Q)	√	2, - >D
4	- (R v - S)	√	2, - >D
5	P		3, . D
6	Q		3, . D
7	- R		4, - vD
8	- - S		4, - vD

The only formula we haven't yet decomposed is the one on line 8. Though you may already see the result, we need to finish the branch anyway.

1	P		SM
2	- ((P . Q) > (R v - S))	√	SM
3	(P . Q)	√	2, - >D
4	- (R v - S)	√	2, - >D
5	P		3, . D
6	Q		3, . D
7	- R		4, - vD
8	- - S	√	4, - vD
9	S		8, - -D

We hope you evaluated the set as consistent, because it is. Let's turn now to a new kind of rule, and to one that is more complicated than the rules we have been examining.

WEDGE DECOMPOSITION

n.	(♥v♦)	√	
	/\		
p.	♥ ♦		n, vD

This rule tells us that if a formula is a wedge claim, to decompose it, we need to draw a fork (viz., /\), and place the left side of the wedge claim under the left branch, and the right side of the formula under the right branch.

FIRST EXAMPLE

1	(P . - Q)		SM
2	(Q v - P)		SM

Since this rule introduces some new twists, let's start with some very simple examples. Given the principle that we should use the earlier rules first, we should begin the decomposition process by selecting line 1 to decompose. When we do this we get. . . .

1	(P . - Q)	√	SM
2	(Q v - P)		SM
3	P		1, . D
4	- Q		1, . D

We now need to decompose line 2. The fork, you should note, is part of the rule. It needs to be drawn in, and it goes at the end of the branch. Notice also that this rule adds only one later line (viz., line 5). Under the leftmost branch of our fork we place the formula that occurred on the left side of the wedge claim; while under the rightmost branch we place the right side of that formula.

1	(P . - Q)	√	SM
2	(Q v - P)	√	SM
3	P		1, . D
4	-Q		1, . D
	/\		
5	Q - P		2, vD
	* *		

Clearly, the leftmost branch of the tree now flowers because of the conflicting lintels Q and -Q. While the rightmost branch flowers because of -P and P. Our tree is now completely flowering and this establishes that the set is inconsistent.

SECOND EXAMPLE

```
1          (P v Q)                                   SM
2          ( - Q v R)                                SM
```

Since both these lines involve using our wedge rule, it really doesn't make a difference which one we start with. Let's do line 2 first. When we decompose this line, we get. . . .

```
1          (P v Q)                                   SM
2          ( - Q v R)                        √       SM
                /\
3.           - Q    R                                2,  vD
```

Now we have a puzzle, however. We need to do line 1, but to decompose it we need to fork. Where should we fork, however? The answer is, we need to fork under every branch that doesn't already have a flower on it and on which we find the formula we want to decompose. In the present case this means under both -Q and under R. The two forks will look absolutely identical.

```
1          (P v Q)                           √       SM
2          (- Q v R)                         √       SM
              /\
3          - Q     R                                 2,  vD
            /\     /\
4          P Q   P  Q                                1,  vD
             *
```

Obviously the resulting tree has three branches that are finished but do not have flowers on the ends of them. All it takes is one such branch to establish that the set we are testing is consistent, however. So the set is consistent. Before we turn to the next rule let's try one more problem -- a bit more complex one this time, hopefully.

THIRD EXAMPLE

```
1          ( - (P > Q) v (T v ( - Q v R)))           SM
2                (Q .  - T)                          SM
```

We'll decompose line 2 first, because the rule for breaking it up is much simpler than the rule for breaking up line 1. (As a general principle, do not use forking rules until you have to.)

```
1          ( - (P > Q) v (T v ( - Q v R)))           SM
2                 (Q . - T)                  √       SM
3                     Q                              2, . D
4                   - T                              2, . D
```

Now we have no choice. We have to break up the formula on line 1. The main connective in this formula is the wedge between Q and T, so we must use the Wedge Decomposition rule.

```
1          ( - (P > Q) v (T v ( - Q v R)))   √       SM
2                 (Q . - T)                  √       SM
3                     Q                              2, . D
4                   - T                              2, . D
                       /\
5           - (P > Q)     (T v ( - Q v R))           1,  vD
```

It is now best to decompose -(P>Q). Notice that when we decompose it we only break it up under the left branch, and we check off only the formula on that branch. We will turn to the formula on the right branch of line 5 soon.

```
1      ( - (P > Q) v (T v ( - Q v R)))              √      SM
2              (Q . - T)                            √      SM
3                  Q                                       2, . D
4                  -T
                   /\                                      2, . D
5       - (P > Q) √   (T v ( - Q v R))                     1,  vD
6         P                                                5, - >D
7        - Q                                               5, - >D
          *
```

The leftmost branch is now flowering because of the conflicting lintels -Q and Q. We still need to break up the formula on the right branch on line 5, however. It causes a fork. Watch how we do it.

```
1      ( - (P > Q) v (T v ( - Q v R)))              √      SM
2               (Q .  - T)                          √      SM
3                  Q                                       2, . D
4                 - T                                      2, . D
                   /\
5       - (P > Q) √   (T v ( - Q v R))              √      1,  vD
6         P                                                5, - >D
7        - Q                                               5, - >D
          *
                              /\
8                       T         ( - Q v R)        √      5,  vD
                        *
```

The leftmost branch on line 8 flowers because of the conflicting pair of lintels, T and -T. However there is still something else that needs to be done, namely, (-QvR). Can you see how the tree is going to end?

```
1      ( - (P > Q) v (T v ( - Q v R)))              √      SM
2              (Q . - T)                            √      SM
3                  Q                                       2,  . D
4                 - T                                      2,  . D
                   /\
5       - (P > Q) √   (T v ( - Q v R))              √      1,  vD
6         P                                                5, - >D
7        - Q                                               5, - >D
          *
                              /\
8                       T         ( - Q v R)        √      5,  vD
                        *
                                     /\
9                                 - Q  R                   8,  vD
                                    *
```

Although there is only one branch that hasn't flowered, our tree is finished. The set is consistent. You might want to study this example before continuing.

HORSESHOE DECOMPOSITION

```
n.        (♥>♦)    √
            /\
p.         -♥  ♦     n, >D
```

This rule tells us that if a formula is a horseshoe claim, to decompose it we need to draw a fork (viz., /\), and put a tilde and the left side of the claim under the leftmost branch and the right side of the formula under the rightmost branch.

AN EXAMPLE

```
1      ((P v Q) > (R v (S > T)))                        SM
2             P                                         SM
```

Because the main connective in the formula on line 1 is a horseshoe, we have to use the new rule first. (Note: To understand why the new rule works the way it does, it is necessary to realize that (P>Q) is equivalent to (-PvQ).)

```
1      ((P v Q) > (R v (S > T)))              √         SM
2              P                                        SM
              /\
3      - (P v Q)  (R v (S > T))                         1, >D
```

The normal thing to do next would be to decompose -(PvQ), but we are going to do the right side of line 3 instead. (The reason for this will become clear shortly.)

```
1      ((P v Q) > (R v (S > T)))              √         SM
2              P                                        SM
              /\
3      - (P v Q)    (R v (S > T))             √         1, >D
                         /\
4                      R  (S > T)                       3, vD
```

What are we supposed to do now? Oddly enough, we can stop because the middle branch is finished and does not have a flower, and so, we know the set we are testing is consistent. We could go on, but we don't have to.

TILDE DOT DECOMPOSITION

```
n.        -(♥.♦)    √
            /\
p.       -♥ -♦    n, - . D
```

This rule tells us that if a formula is a tilde dot claim, to decompose it we need to draw a fork, and put a tilde and then the formula on the left side of the claim we are decomposing under the leftmost branch, and a tilde and the right side of that formula under the right branch.

The fact that the formulas, -(P.Q) and (-Pv-Q), are logically equivalent, explains why the rule works the way it does.

AN EXAMPLE

```
1        (P . - (Q .  (R > P)))                    SM
2           Q                                      SM
```

We obviously have to start by using dot decomposition on line 1.

```
1        (P . - (Q .  (R > P)))            √       SM
2           Q                                      SM
3           P                                      1, .D
4        - (Q .  (R > P))                          1, .D
```

We can now use the new rule on line 4.

```
1        (P . - (Q .  (R > P)))            √       SM
2           Q                                      SM
3           P                                      1, .D
4        - (Q .  (R > P))                  √       1, .D
            /\
5         - Q   - (R > P)                  √       4,-.D
          *
6                  R                               5,->D
7                 - P                              5,->D
                  *
```

The set is inconsistent.

TRIPLE BAR DECOMPOSITION

```
n.        (♥=♦)      √
            /\
p.         ♥  -♥     n, = D
p+1.       ♦  -♦     n, = D
```

This rule tells us that if a formula is a triple bar claim, to decompose it we need to draw a fork and put first the left, and then underneath that, the right sides of the formula we are decomposing, under the left branch. Then we need to put the left and right sides of this formula, but negated, under the right branch of the fork.

AN EXAMPLE

```
1                 - P                              SM
2                 - Q                              SM
3                 (P = - Q)                        SM
```

Notice that this rule requires not only a fork, but also two later lines. It, and the next rule, are the most complex ones of all.

```
1            - P                                SM
2            - Q                                SM
3         (P = - Q)                     √       SM
              /\
4          P    - P                             3, =D
5         - Q   - - Q                           3, =D
           *
```

Now all we need to do is decompose the formula, --Q, on the right side of line 5 and we're finished. Once we do this we get. . . .

```
1            - P                                SM
2            - Q                                SM
3         (P = - Q)                     √       SM
                /\
4             P    - P                          3, =D
5            - Q   - - Q                √       3, =D
              *
6                   Q                           5,--D
                    *
```

The set is inconsistent.

TILDE TRIPLE BAR DECOMPOSITION

```
n.        -(♥=♦ )     √
             /\
p.         ♥  -♥      n, - = D
p+1.      -♦   ♦      n, - = D
```

This rule tells us that if a formula is a tilde triple bar claim, to decompose it we need to draw a fork and put first the left side of the formula we are decomposing, and then, underneath that, a tilde and then the right side of this formula, under the leftmost branch. Then, we need to put a tilde and the left side of this formula, and, below that, the right side, under the rightmost branch.

AN EXAMPLE

```
1            (P = Q)                            SM
2            - (P = Q)                          SM
```

Let's start with line 1.

```
1            (P = Q)                    √       SM
2            - (P = Q)                          SM
                 /\
3              P      - P                       1,  =D
4              Q      - Q                       1,  =D
```

Now it's time for us to decompose line 2 using our new rule.

```
1          (P = Q)                 √     SM
2          - (P = Q)               √     SM
              /\
3           P       - P                  1,  =D
4           Q       - Q                  1,  =D
           /\        /\
5          P  - P    P   - P             2, - =D
6          -Q  Q   - Q   Q               2, - =D
           *   *    *    *
```

The set we are testing is inconsistent.

The fact that (P=Q) is equivalent to ((P.Q)v(-P.-Q)) explains why the Triple Bar Decomposition rule works the way it does; while the fact that -(P=Q) is equivalent to ((P.-Q)v(-P.Q)) explains why the Tilde Triple Bar rule works as it does.

Now that we have examined all of the decomposition rules, all we need to see is how to use the truth tree method to find answers to the other kinds of questions we might have. More specifically, we need to know how to use the tree method to: (1) determine whether an argument is valid or invalid; (2) determine whether a single statement is logically true, logically false, or logically indeterminate; and (3) determine whether a pair of statements is, or is not, equivalent. Let's turn briefly to these topics.

USING THE TREE METHOD TO DETERMINE VALIDITY

Really, the tree method is only capable of determining whether a set of statements is consistent or inconsistent. To use this method to find out whether an argument is valid or invalid we must first transform the argument into a set of statements. In the chapter on Basic Concepts we said that an argument is valid just in case the set of statements consisting in that argument's premises and the negation of its conclusion is inconsistent. This definition provides the clue we need to be able to use the tree method on arguments. It suggests that we can find out whether an argument is valid or invalid in the following way:

INSTRUCTIONS FOR DETERMINING VALIDITY

1. Construct a set of statements consisting in the original argument's premises, together with the negation of its conclusion.
2. Use the tree method on this set of statements.
3. If the result is that the set is inconsistent, this will tell us that the original argument is valid; while if the result is that the set is consistent, this will tell us that the original argument is invalid.

AN EXAMPLE

ORIGINAL ARGUMENT TESTED		SET OF STATEMENTS TO BE TESTED	
Premise:	(P > (Q v R))	Set Member:	(P > (Q v R))
Premise:	(P . - Q)	Set Member:	(P . - Q)

Conclusion:	R	Set Member:	- R

```
        TREE                                        EXPLANATION

1       (P > (Q v R))        SM
2       (P .  - Q)           SM
3       - R                  SM
4         P                  2, . D
5       - Q                  2, . D
          /\
6     P    (Q v R)           1, >D
      *        /\
7            Q    R          6, vD
             *    *
```

The tree on the left tells us that the set we have tested is INCONSISTENT. This means that the argument we started with is VALID. If the tree had instead shown that the set was CONSISTENT, we would have concluded that the argument was INVALID.

USING THE TREE METHOD TO DETERMINE WHETHER A SINGLE STATEMENT IS LOGICALLY TRUE, LOGICALLY FALSE, OR LOGICALLY INDETERMINATE

If we want to understand how to use the tree method to test a single statement we need to recognize two things. First, a single statement is logically false if and only if the set consisting in that statement, and nothing else, is inconsistent. Second, a statement is logically true if and only if its negation is logically false.

To test a single statement all we have to do is transform that statement into the appropriate set of statements and construct our tree on that set. The following principles tell us how to do this.

INSTRUCTIONS FOR DETERMINING IF A STATEMENT IS LOGICALLY FALSE

Where S is any statement, to find out whether or not S is logically false test {S}. If the tree on this set is completely flowering, this means that S is logically false. On the other hand, if the tree on this set is not completely flowering, this means that S is not logically false.

```
        EXAMPLE                                EXPLANATION

        1         (P . - P)  √         SM
        2           P                  1, .D
        3          - P                 1, .D
                    *
```

The tree on the left shows that the single statement, (P.-P), is LOGICALLY FALSE. If the tree had not been completely flowering, this would have shown that the statement is NOT LOGICALLY FALSE.

INSTRUCTIONS FOR DETERMINING IF A STATEMENT IS LOGICALLY TRUE

Where S is any statement, to find out if S is logically true, test {-S}. If the tree on this set is completely flowering it means that S is logically true. On the other hand, if the tree on this set is not completely flowering it means that S is not logically true.

```
        EXAMPLE                                EXPLANATION

        1      - (P v - P)√            SM
        2           - P                1,-vD
        3          - - P      √        1,-vD
        4            P                 3,--D
                     *
```

The tree on the left shows that the single statement (Pv-P), is LOGICALLY TRUE. If the tree had not been a completely flowering tree, this would have shown that the statement is NOT LOGICALLY TRUE.

To use the tree method to find out that a single statement is logically indeterminate, we would have to construct a tree on both the statement and its negation and find both trees not completely flowering.

USING THE TREE METHOD TO DETERMINE WHETHER A PAIR OF STATEMENTS IS EQUIVALENT

The key to understanding how to use the tree method to discover if two statements are, or are not, equivalent, lies in recognizing that they will be equivalent if and only if the claim that they are equivalent is logically true. Put a little more clearly, if S1 and S2 are equivalent the statement, (S1=S2), will be logically true; and if S1 and S2 are not equivalent the statement, (S1=S2), will not be logically true.

INSTRUCTIONS FOR DETERMINING WHETHER A PAIR OF STATEMENTS IS EQUIVALENT

Where S1 and S2 are the pair of statements in question, test {-(S1=S2)}. If the tree on this set is completely flowering the two statements will be equivalent. On the other hand, if the tree is not completely flowering the two statements will not be equivalent.

EXAMPLE

```
1          - (P = - - P)√       SM
                /\
2             P      - P        1,-=D
3          - - - P√  - - P√     1,-=D
4            - P      P         4,--D
              *       *
```

EXPLANATION

The tree on the left shows that the single statements, P and - - P are EQUIVALENT. Had the tree not been a completely flowering tree this would have told us that the statements are NOT EQUIVALENT.

PROBLEMS

A. Use the truth tree method to decide whether the following sets of statements are consistent or inconsistent.

1. {(P v (- Q > R)); - ((R = Q) v (P v S))}
2. {(P = (Q . R)); (P v - R); - (P = Q)}
3. {(P > (Q . - R)); (- R > (Q . P)); - (P = R)}
4. {((A > B) > (C > D)); (- (C > A) . (D > E)); - E}
5. {((A > - D) v (A > C)); (D = - C); (C > (- A . - D))}
6. {((F > G) . (H > I)); (- (J . H) > - (-G > I)); (F v H); - H}
7. {(D v (E . - F)); (D = (G . - H)); - (F > - H)}

B. Use the truth tree method to decide whether the following arguments are valid or invalid.

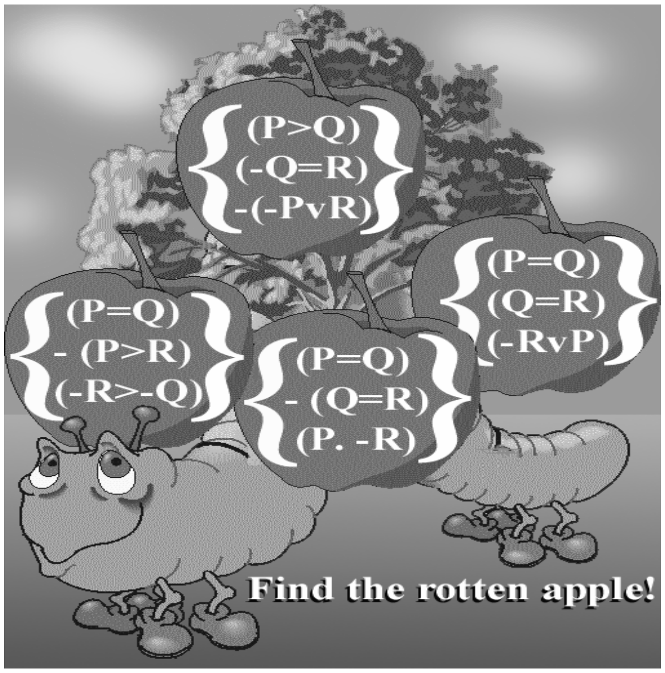
1. {(P > Q); (Q > (R . S))} / (- R > - P)
2. {(P = Q); - (Q = R)} / (P > -R)
3. {(P > (Q . (R v S))); (R > - - S)} / (- S > - R)
4. {(P > (Q v R)); - (Q . R); - (P = S)} / (P > (Q v - S))
5. {((- Q v - R) > - P); (R > - S)} / (P > (- S . T))
6. {(- A v - (- B . C)); - (A > D); (D = B)} / (- C v - B)
7. {(A = (B > F)); (F > (B v C))} / (B > (A > F))

C. Use the truth tree method to decide whether the following single statements are logically true, logically false, or logically indeterminate.

1. (P > (- P = (Q . - Q)))
2. ((P . Q) = (- P v - Q))
3. ((P > (P > Q)) > (P > (P . Q)))
4. (((P v Q) > R) . - (- R > - Q))
5. ((T > S) = (S . T))
6. ((P v (Q v R)) > (- P > (- Q > R)))
7. - ((((P > Q) . (R > S)) . (P v R)) > (Q v S))

D. Use the truth tree method to decide whether the following pairs of statements are equivalent or are not equivalent.

1.	(P > (Q > R))	((Q . P) > R)
2.	(P . Q)	(P v Q)
3.	(P > (- Q v R))	((Q . P) > R)
4.	(P = Q)	(- Q > - P)
5.	(- A v - (B . C))	- ((A v B) . (C v A))
6.	(P > (- Q > - R))	((R . P) > Q)
7.	(P > (Q > P))	(R v - R)


{(P>Q) (-Q=R) -(-PvR)}
{(P=Q) (Q=R) (-RvP)}
{(P=Q) - (P>R) (-R>-Q)}
{(P=Q) - (Q=R) (P. -R)}
Find the rotten apple!