Chapter 1

Sentential Logic: Truth Trees

Truth-tables provide a mechanical means of determining the logical status of sentences, sets, and arguments. You simply follow the rules, and the table will give you the answer. As I mentioned before, though, they have a significant drawback. With every additional propositional constant, the truth-table doubles in size. Truth-trees provide another means for determining consistency, validity, etc., and in many cases truth-trees are far more efficient than truth-tables.

Strictly speaking, truth-trees are methods for determining consistency of sets. Since the concepts of tautology, contradiction, consistency, logical equivalence, and validity can all be defined in terms of consistency, truth trees can be very powerful tools for logical analysis.

Remember that a set is consistent if and only if its members are all true in at least one model. A set is inconsistent if and only if there is no model in which all of its members are true. Now, let's define our other logical concepts in terms of consistency.

Logical Falsehood : α is logically false if and only if $\{\alpha\}$ is inconsistent.

Logical Truth : α is logically true if and only if $\{\neg\alpha\}$ is inconsistent.

Logical Indeterminacy : α is logically indeterminate if and both $\{\alpha\}$ and $\{\neg\alpha\}$ are consistent.

Logical Equivalence : α and β are logically equivalent if and only if $\{\neg(\alpha \equiv \beta)\}$ is inconsistent.

Logical Entailment : $\Gamma \vDash \alpha$ if and only if $\Gamma \cup \{\neg\alpha\}$ is inconsistent.

Logical Validity : An argument is logically valid if and only the set containing the premises and the negation of the conclusion is inconsistent.

The trees work by breaking down complex sentences into increasingly simpler sentences, until nothing is left except for simple sentences and the negations of simple sentences. Complex sentences are broken down by using the following decomposition rules.

1.1 Decomposition Rules

Conjunctions

Conjunction Decomposition

$$\begin{array}{c} \alpha \,\&\, \beta \checkmark \\ \alpha \\ \beta \end{array}$$

Negated Conjunction Decomposition

$$\begin{tikzcd}[column sep=small] & \neg(\alpha \,\&\, \beta) \checkmark \arrow[dl, dash] \arrow[dr, dash] & \\ \neg\alpha & & \neg\beta \end{tikzcd}$$

Note that the tree on the right splits, or has branches. A tree branches when there is more than one way that a sentence could be true. In order for a conjunction to be true, both conjuncts must be true. So, the tree for a conjunction does not branch. There are two ways, however, that conjunctions could be false — either the first conjunct is false or the second conjunct is false.

Knowing the truth conditions for disjunctions, conditionals, and biconditionals, you should be able to see what the trees for those sentences and their negations will look like. Disjunctions and conditionals will both branch, but their negations will not. The trees for biconditionals and their negations will both branch.

Disjunctions

Disjunction Decomposition

$$\begin{tikzcd} & \alpha \lor \beta \arrow[dl, no head] \arrow[dr, no head] & \\ \alpha & & \beta \end{tikzcd}$$

Negated Disjunction Decomposition

$\neg(\alpha \lor \beta)$
$\neg\alpha$
$\neg\beta$

Conditionals

Conditional Decomposition

$$\begin{tikzcd} & \alpha \supset \beta \checkmark \arrow[dl, no head] \arrow[dr, no head] & \\ \neg\alpha & & \beta \end{tikzcd}$$

Negated Conditional Decomposition

$\neg(\alpha \supset \beta) \checkmark$
α
$\neg\beta$

Biconditionals

Biconditional Decomposition

$$\begin{tikzcd} & \alpha \equiv \beta \checkmark \arrow[dl, no head] \arrow[dr, no head] & \\ \alpha \,\&\, \beta & & \neg\alpha \,\&\, \neg\beta \end{tikzcd}$$

Negated Biconditional Decomposition

$$\begin{tikzcd} & \neg(\alpha \equiv \beta) \checkmark \arrow[dl, no head] \arrow[dr, no head] & \\ \alpha \,\&\, \neg\beta & & \neg\alpha \,\&\, \beta \end{tikzcd}$$

Double Negations

Finally, we need a rule for double negations. As you can guess, it's rather simple:

Double Negation Decomposition

$\neg\neg\alpha \checkmark$
α

1.2 Logical Analysis with Trees

A branch includes all of the sentences acquired by reading from the bottom of the tree upwards to the top. If a branch contains both a simple sentence and its negation, then that branch is closed. If all of the complex sentences on a branch have been completely decomposed, and that branch does not contain a simple sentence and the negation of that same sentence, then it is a completed open branch. A tree is completed when every branch is either a closed branch or a completed open branch. A tree is closed if all of its branches are closed. It is open if it contains at least one completed open branch. Sentences that have been decomposed are marked with a checkmark; branches that are closed are marked with an "x."

Strategically, the goal is to close branches as quickly as possible. That will keep the tree manageable. Sentences may be decomposed in any order. The best strategy is to first decompose any sentence that does not branch, then decompose sentences that will result in a closed branch.

Consistency

To use a tree to determine the consistency of a set, start by listing the members of the set at the top of the tree on separate, numbered lines.

1. $P \supset Q$
2. $\neg(Q \vee R)$
3. $R \vee \neg P$

The first thing to do is to decompose line 2, since it doesn't branch. Just as in derivations, we'll list what line these came from and what rule was used. We'll also check off line 2, so that we'll know that it has been decomposed already.

1.	$P \supset Q$	
2.	$\neg(Q \vee R)$ ✓	
3.	$R \vee \neg P$	
4.	$\neg Q$	$2 \neg \vee D$
5.	$\neg R$	$2 \neg \vee D$

Lines 1 and 3 will both branch, so we should see if either will result in a closed branch. In this case, both will, so it doesn't matter which we do first.

```
1.   P ⊃ Q ✓
2. ¬(Q ∨ R) ✓
3.   R ∨ ¬P
4.     ¬Q          2 ¬ ∨ D
5.     ¬R          2 ¬ ∨ D
       / \
6.   ¬P   Q        1 ⊃ D
          ✗
```

Note that the branch on the right now contains Q and ¬Q, so it is closed. We will mark it as closed with some symbol, an “X” is common, especially when writing trees out by hand. If a branch is closed, then nothing more needs to be done on that branch.

Now, we just have line 3 remaining to finish.

```
1.      P ⊃ Q ✓
2.    ¬(Q ∨ R) ✓
3.     R ∨ ¬P ✓
4.        ¬Q          2 ¬ ∨ D
5.        ¬R          2 ¬ ∨ D
         /   \
6.    ¬P       Q      1 ⊃ D
      / \
7.  R    ¬P    ✗
    ✗
```

An open tree contains at least one open branch; a closed tree contains no open branches.

So, we finished decomposing all of the complex sentences, but we still have one open branch. That means our original set is consistent. The tree also gives us a model on which all of the set’s members are true. We just read up an open branch and assign F to any negated simple sentence and T to any non-negated simple sentence. In this case, the members of the set are all true when P, Q, and R are all false.

Trees for single sentences can be used to determine if a sentence is a tautology, a contradiction, or a contingency. If the tree for a sentence is closed, then it is a contradiction. If the tree for the negation of a sentence is closed, then the sentence is a tautology. If neither the tree for the sentence nor the tree for its negation is closed, then the sentence is contingent.

1.3 Contradictions and Tautologies

To determine whether a sentence is a contradiction, we do a tree for that sentence. If the tree is closed, then there is no model in which the sentence is true, and it is therefore a contradiction. Here's an example:

P & (¬(Q ⊃ P))

We start by writing the sentence on line 1.

1.	P & ¬(Q ⊃ P)	

Then, we will decompose it using the conjunction decomposition rule, and checking it off showing that we're finished with it.

1.	P & ¬(Q ⊃ P) ✓	
2.	P	1 & D
3.	¬(Q ⊃ P) ✓	1 & D

Line 2 is already a simple sentence, so there's nothing we need to do with it. Decomposing the negated conditional on line 3 produces '¬P', which closes the branch.

1.	P & ¬(Q ⊃ P) ✓	
2.	P	1 & D
3.	¬(Q ⊃ P) ✓	1 & D
4.	Q	3 ¬ ⊃ D
5.	¬P	3 ¬ ⊃ D
	✗	

Since the tree has only one branch and it is closed, there is no model on which the original sentence is true. So, it is a contradiction.

To determine if a sentence is a tautology, we do a tree for the negation of the sentence. Since the negation of a tautology is false on every model, the tree will be closed. Here's an example:

P ⊃ (Q ⊃ P)

We start by putting the negation of the sentence on line 1.

1. ¬[P ⊃ (Q ⊃ P)]

After that, decompose line 1 with negated conditional decomposition.

1.	¬[P ⊃ (Q ⊃ P)] ✓
2.	P
3.	¬(Q ⊃ P)

Now, decompose line 3, again using negated conditional decomposition

1.	¬[P ⊃ (Q ⊃ P)] ✓	
2.	P	1 ¬ ⊃ D
3.	¬(Q ⊃ P) ✓	1 ¬ ⊃ D
4.	Q	3 ¬ ⊃ D
5.	¬P	3 ¬ ⊃ D
	✗	

This tree tells us that '¬[P ⊃ (Q ⊃ P)]' is a contradiction, which means that our original sentence, 'P ⊃ (Q ⊃ P)' is tautology.

If a tree for a sentence is open, we only know that it is not a contradiction, *even if every branch is open.* Likewise, if tree for a negated sentence is open, we only know that the original sentence is not a tautology. Let's see a simple example:

P ∨ (Q & R)

1. P ∨ (Q & R) ✓

2.	P	Q & R ✓	1 ∨ D
3.		Q	2 & D
4.		R	2 & D

Note that both branches are open. Now, we can easily state a model on which 'P ∨ (Q & R)' is true. The left-hand branch tells us that any model in which 'P' is true will be a model in which 'P ∨ (Q & R)' is true. We can fill in any values for 'Q' and 'R' that we like. Here's one such model:

P = **T**
Q = **T**
R = **T**

Note, though, that even though every branch is open, our sentence is definitely not a tautology. 'P ∨ (Q & R)' is false whenever both P and at least one of Q or R are false. We can show this by doing a tree for the negation of the sentence. Here's the completed tree:

```
1.  ¬[P ∨ (Q & R)] ✓
2.          ¬P               1¬ ∨ D
3.     ¬(Q & R) ✓            1¬ ∨ D
          /  \
4.     ¬Q     ¬R             3¬ & R
```

Now, since we have an open tree for both 'P ∨ (Q & R)' and its negation, we have shown that the sentence is logically indeterminate, or contingent.

1.4 Equivalence

Remember that a biconditional is true whenever the two sentences joined by the biconditional operator have the same truth value. So, if the two sentences have the same truth value in every model, then the biconditional will be a tautology. This means that we can test whether two sentences are logically equivalent by joining them with a triple bar, negating the resulting sentence, then working a truth-tree for it. If the tree is closed, then the two original sentences are logically equivalent.

Let's test it with a simple example:

P ⊃ Q ¬P ∨ Q

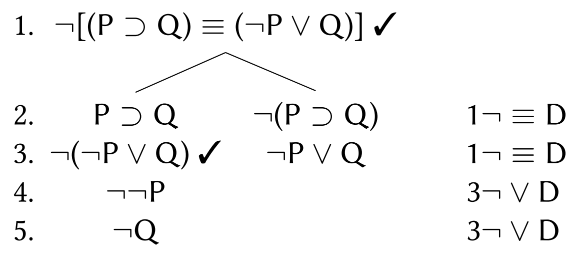
First, we will make a negated biconditional using the two sentences.

```
1.  ¬[(P ⊃ Q) ≡ (¬P ∨ Q)]
```

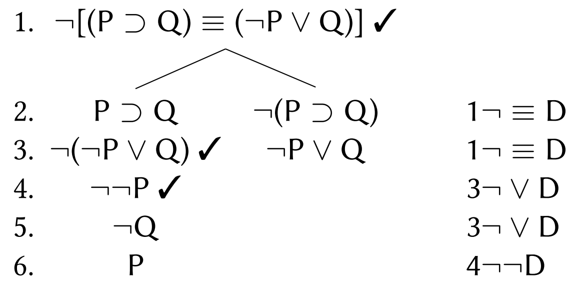
Now, we can decompose it using the negated biconditional rule.

```
1.  ¬[(P ⊃ Q) ≡ (¬P ∨ Q)] ✓
             /     \
2.      P ⊃ Q      ¬(P ⊃ Q)        1¬ ≡ D
3.    ¬(¬P ∨ Q)     ¬P ∨ Q         1¬ ≡ D
```

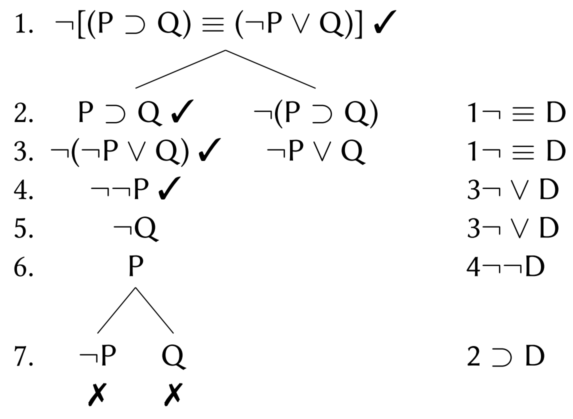
Now, we have to finish each branch; let's work on the left side first. Too keep things as simple as possible, remember to always decompose non-branching sentences first. In this case, we should decompose the negated disjunction.



The next step is to take care of the double negation.



Now, we can decompose the conditional on line 2, which will result in two closed branches.


1. ¬[(P ⊃ Q) ≡ (¬P ∨ Q)] ✓
2. P ⊃ Q ✓ ¬(P ⊃ Q) 1¬ ≡ D
3. ¬(¬P ∨ Q) ✓ ¬P ∨ Q 1¬ ≡ D
4. ¬¬P ✓ 3¬ ∨ D
5. ¬Q 3¬ ∨ D
6. P 4¬¬D
7. ¬P Q 2 ⊃ D
✗ ✗

Now, we'll decompose the negated conditional on the main right hand branch, since it doesn't branch.

```
1.  ¬[(P ⊃ Q) ≡ (¬P ∨ Q)] ✓
           /          \
2.    P ⊃ Q ✓      ¬(P ⊃ Q) ✓      1¬ ≡ D
3.  ¬(¬P ∨ Q) ✓     ¬P ∨ Q         1¬ ≡ D
4.     ¬¬P ✓          P            3¬ ∨ D; 2¬ ⊃ D
5.      ¬Q           ¬Q            3¬ ∨ D; 2¬ ⊃ D
6.       P                         4¬¬D
        / \
7.    ¬P   Q                       2 ⊃ D
      ✗    ✗
```

Finally, we decompose the disjunction on line 3, which gives us two new closed branches.

```
1.  ¬[(P ⊃ Q) ≡ (¬P ∨ Q)]
           /          \
2.    P ⊃ Q       ¬(P ⊃ Q)        1¬ ≡ D
3.  ¬(¬P ∨ Q)      ¬P ∨ Q         1¬ ≡ D
4.     ¬¬P           P            3¬ ∨ D; 2¬ ⊃ D
5.      ¬Q          ¬Q            3¬ ∨ D; 2¬ ⊃ D
                    / \
6.       P        ¬P   Q          4¬¬D; 3 ∨ D
        / \
7.    ¬P   Q      ✗    ✗          2 ⊃ D
      ✗    ✗
```

1.5 Validity

Since validity can be defined in terms of consistency, truth trees can be used to determine if an argument if valid. List the premises on numbered lines at the top, followed by the negation of the conclusion. Then, complete the tree using the same rules and methods as above. If the tree is closed, then there is no way for the premises to be true and the conclusion false, and the argument is valid. If the tree is open, then there is at least one model on which the premises are true and the conclusion false, and the argument is invalid. A truth-value assignment showing invalidity can be read from any open branch.

Here's a simple tree on a version of DeMorgan's Law, P & Q ⊢ ¬(¬P ∨ ¬Q). First, we list the premises and the negated conclusion.

1.	P & Q
2.	¬¬(¬P ∨ ¬Q)

Then, we decompose any non-branching sentences.

1.	P & Q ✓	Premise
2.	¬¬(¬P ∨ ¬Q) ✓	Negated Conclusion
3.	P	1 &D
4.	Q	1 &D
5.	¬P ∨ ¬Q	2 ¬¬ D

We then continue by decomposing any branching sentences, beginning with those that will result in closed branches. In this case, we only have line 5 remaining.

1.	P & Q ✓		Premise
2.	¬¬(¬P ∨ ¬Q) ✓		Negated Conclusion
3.	P		1 &D
4.	Q		1 &D
5.	¬P ∨ ¬Q ✓		2 ¬¬ D
6.	¬P	¬Q	5 ∨ D
	✗	✗	

Since all branches are closed, the argument is valid. Note that a tree for logical entailment works exactly the same way, simply put all of the members of the set and the negation of the entailed sentence on the tree.

Here is a more complex example:

(G & H) ⊃ ¬I, I ∨ G, H ⊃ ¬G, H ≡ I ⊢ ¬H ∨ G

As before, we begin by list the premises and the negation of the conclusion.

1.	(G & H) ⊃ ¬I
2.	I ∨ G
3.	H ⊃ ¬G
4.	H ≡ I
5.	¬(¬H ∨ G)

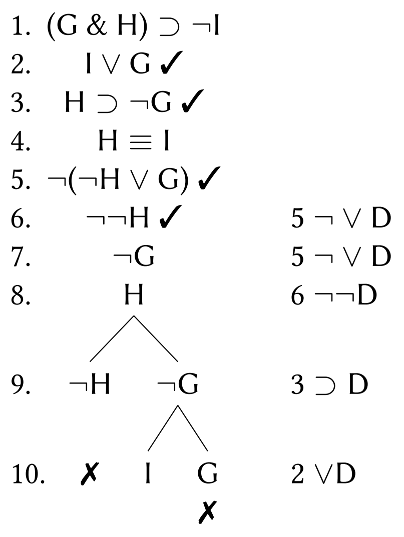
Unfortunately, there is only line that doesn't branch. So, we'll decompose line 5, followed by decomposing the double negation that will result.

1.	(G & H) ⊃ ¬I	
2.	I ∨ G	
3.	H ⊃ ¬G	
4.	H ≡ I	
5.	¬(¬H ∨ G) ✓	
6.	¬¬H ✓	5 ¬ ∨ D
7.	¬G	5 ¬ ∨ D
8.	H	6 ¬¬D

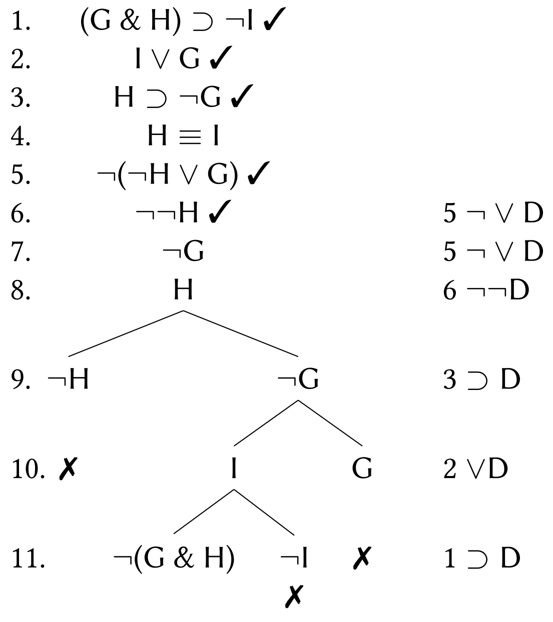
Now, we have a choice. Both line 2 and line 3 will branch, but they will both also result in an immediately closed branch. Let's start with line 3.

```
1.  (G & H) ⊃ ¬I
2.       I ∨ G
3.    H ⊃ ¬G ✓
4.       H ≡ I
5.  ¬(¬H ∨ G) ✓
6.      ¬¬H ✓          5 ¬ ∨ D
7.        ¬G           5 ¬ ∨ D
8.         H           6 ¬¬D
          / \
9.     ¬H     ¬G       3 ⊃ D
       ✗
```

We'll next do line 2.



That just leaves two lines. I usually try to avoid biconditionals until the end, so I'll decompose line 1.



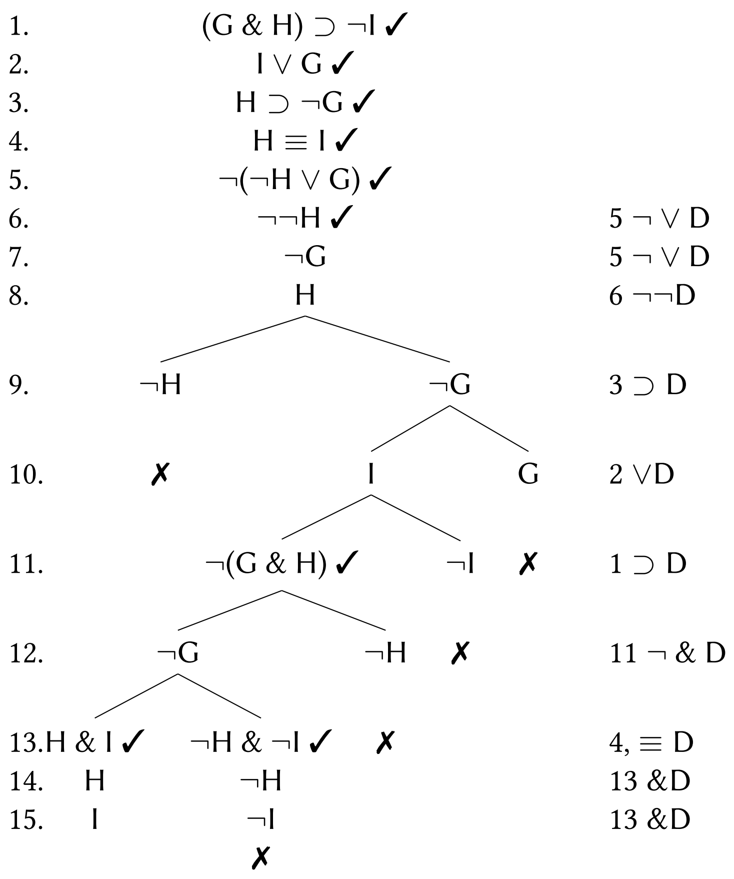
Now, I'll decompose the negated conjunction in line 11.

```
1.        (G & H) ⊃ ¬I ✓
2.             I ∨ G ✓
3.           H ⊃ ¬G ✓
4.             H ≡ I
5.         ¬(¬H ∨ G) ✓
6.             ¬¬H ✓                    5 ¬ ∨ D
7.              ¬G                      5 ¬ ∨ D
8.               H                      6 ¬¬D
            /         \
9.  ¬H                  ¬G              3 ⊃ D
                      /     \
10. ✗                I       G          2 ∨D
                   /   \
11.        ¬(G & H) ✓   ¬I    ✗         1 ⊃ D
             /   \
12.        ¬G     ¬H     ✗              11 ¬ & D
                  ✗
```

The right branch closes with H and ¬H. Now I'm forced to decompose the biconditional in line 4.

1.		$(G \,\&\, H) \supset \neg I$ ✓				
2.		$I \vee G$ ✓				
3.		$H \supset \neg G$ ✓				
4.		$H \equiv I$ ✓				
5.		$\neg(\neg H \vee G)$ ✓				
6.		$\neg\neg H$ ✓				5 $\neg \vee$ D
7.		$\neg G$				5 $\neg \vee$ D
8.		H				6 $\neg\neg$D
9.	$\neg H$			$\neg G$		3 $\supset$ D
10.	✗		I		G	2 $\vee$D
11.		$\neg(G \,\&\, H)$ ✓		$\neg I$	✗	1 $\supset$ D
12.	$\neg G$		$\neg H$	✗		11 $\neg$ & D
13.	$H \,\&\, I$	$\neg H \,\&\, \neg I$	✗			4, $\equiv$ D

All that remains now is to decompose the two conjunctions in line 13.



The tree is finally finished. The right branch is closed, but the left branch is not. It tells us that the premises of the original argument will be true and the conclusion will be false whenever G is false, H is true, and I is true. That would be the fifth line of the truth table, and would look like this:

G	H	I	(G	&	H)	⊃	¬I	/	I	∨	G	/	H	⊃	¬G	/	H	≡	I	//	¬H	∨	G
F	T	T	F	F	T	**T**	F		T	**T**	F		T	**T**	T		T	**T**	T		F	**F**	F