

Part I

TFL Truth Trees

Chapter 1

Truth Trees

1.1 Truth Tables aren't Enough

We've learned two ways of using truth tables to test for consistency, equivalence, validity, etc. Both of them have their uses and their drawbacks. A complete truth table always gives the right answer by following a mechanical process. But it can be long and tedious; and the complexity and length increases exponentially with the number of variables. A 2 variable table only has 4 rows, but a 20 variable table has over 1 million.

A partial truth table, on the other hand, requires a little bit of cunning to see which step should be taken next, and to spot contradictions. And it can be difficult to tell which steps were done in which order, unless you make many copies of your truth table.

So our first method takes too long for complex formulas, while our second method is too unstructured both when creating and reviewing. In this part, we will introduce a third method that is intrinsically more complex than either, but does not grow as fast as complete truth tables, and is more structured, rule-driven and checkable than partial truth tables. It will also work for the logic we introduce in the second half of this book, unlike truth tables.

The structures produced by this new method are called TRUTH TREES, because of the way the diagram of formulas can 'branch' out to represent possibilities.

1.2 Mutual Consistency

The method of trees is most directly a way to test for the consistency of a set of TFL formulas. Since all our other logical notions can be defined in terms

of consistency (or lack of consistency), it indirectly gives a way to test for equivalence, validity, and so forth. For example, you may recall that one way to test if an argument is valid, is to test if the premises and the negation of the conclusion are mutually consistent – that is, if there is a counter-example, and thus the argument is invalid.

The basic idea of the tree method is that we'll write down the formulas we are testing for consistency, and then we'll break these formulas down into their component subformulas, like with did with our partial truth tables, in order to see whether it's possible to find a valuation where they are all true.

Our first Truth Tree We will begin by working through an example that illustrates the general method, then come back to give precise rules for the tree method. We want to know whether the following formulas are all mutually consistent in TFL:

$$A \wedge B, \neg(C \vee D), (\neg B \vee C) \vee E, \neg E$$

We begin by writing down the formulas we will test for consistency:

1.	$A \wedge B$	Root
2.	$\neg(C \vee D)$	Root
3.	$(\neg B \vee C) \vee E$	Root
4.	$\neg E$	Root

The formulas we write down at the beginning of the tree are called the **ROOT**. Trees are designed to show whether the root is consistent, and if so, provide a valuation. We will assume the formulas in the root are all true. We then write down formulas which our previous formulas tell us must also be true, and repeat this process until we have found the information we need.

Consider line (1), $A \wedge B$. For this conjunction to be true, both conjuncts must be true. The truth of each conjunct follows from what is already written down. We add them to the tree:

1.	$A \wedge B$ ✓	Root
2.	$\neg(C \vee D)$	Root
3.	$(\neg B \vee C) \vee E$	Root
4.	$\neg E$	Root
5.	A	1 \wedge
6.	B	1 \wedge

When we add lines (5) and (6), we note that they came from line (1), and that we considered conjunction (\wedge) to produce them. This is useful for you

when revising your work, and to help us to understand what you thought you were doing when we see some unexpected formulas in the truth tree. We also add a check mark on (1) once we've decomposed it, to avoid repeating our actions.

Now consider line (2) $\neg(C \vee D)$. This is a negated disjunction. A negated disjunction is true when a disjunction is false. Disjunctions are false iff *both* disjuncts are false. We only write down formulas that are true, so if a formula is false, we write down its negation, which will be true. So we include new lines for $\neg C$ and $\neg D$, adding a check mark on line (2):

1.	$A \wedge B$	✓	Root
2.	$\neg(C \vee D)$	✓	Root
3.	$(\neg B \vee C) \vee E$		Root
4.	$\neg E$		Root
5.	A		1 \wedge
6.	B		1 \wedge
7.	$\neg C$		2 $\neg\vee$
8.	$\neg D$		2 $\neg\vee$

Line (3) is a disjunction. Unlike the previous two cases, it doesn't tell us what *must* be the case; it says that (at least) one of the disjuncts must be true. We represent this by *branching* our tree :

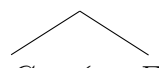
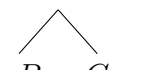
1.	$A \wedge B$	✓	Root
2.	$\neg(C \vee D)$	✓	Root
3.	$(\neg B \vee C) \vee E$	✓	Root
4.	$\neg E$		Root
5.	A		1 \wedge
6.	B		1 \wedge
7.	$\neg C$		2 $\neg\vee$
8.	$\neg D$		2 $\neg\vee$
$\swarrow \quad \searrow$			
9.	$\neg B \vee C$	E	3 \vee

These two branches represent different potential valuations that we are considering. We always read a branch from the bottom up. So both branches contain all the root formulas. Another way of thinking about branches in a truth tree is that we have several lists, and we've been too lazy to write the formulas in common more than once.

So now we have to consider our branches in turn. We start by examining the right branch, just because it's simpler – it just contains an atomic formula,

E . Notice, however, that line (4) was $\neg E$. So this branch's valuation requires both E and $\neg E$ to be true; that's impossible. If a branch contains any formula and its negation, we know that branch represents an impossible valuation. We'll call this branch *closed*, and mark it with an ' \times '.

The left branch on (9) is another disjunction $\neg B \vee C$. It too branches out into its two disjuncts:

1.	$A \wedge B$	✓	Root
2.	$\neg(C \vee D)$	✓	Root
3.	$(\neg B \vee C) \vee E$	✓	Root
4.	$\neg E$		Root
5.	A		1 \wedge
6.	B		1 \wedge
7.	$\neg C$		2 $\neg\vee$
8.	$\neg D$		2 $\neg\vee$
			
9.	$\neg B \vee C$	✓	3 \vee
		E	\times
			
10.	$\neg B$	C	9 \vee
	\times	\times	

Both of these disjuncts also result in closed branches. The left branch at (10) is the negation of (6), and the right branch is the negation of (7). Now every branch in this tree is closed. This corresponds to the idea that every potential way to make the formulas consistent ended up requiring a contradiction, so are impossible. There is no possible valuation that makes (1)-(4) all true. In other words, the formulas are mutually inconsistent.


Our second Truth Tree Let's work through another consistency example. We've seen most of the elements of a truth tree, so we can make this one simpler. we want to test whether the following formulas are mutually consistent in TFL:

$$D \vee G, \neg D, \neg G \vee S$$


We begin by writing down the root:

1.	$D \vee G$	Root
2.	$\neg D$	Root
3.	$\neg G \vee S$	Root



Line (1) is a disjunction, so it will produce a branch:

1.	$D \vee G$	✓	Root
2.	$\neg D$		Root
3.	$\neg G \vee S$		Root
			
4.	D	G	$1 \vee$

Line (2) is already atomic, and can't be decomposed. But it does help us to close one of the branches:

1.	$D \vee G$	✓	Root
2.	$\neg D$	✓	Root
3.	$\neg G \vee S$		Root
			
4.	D	G	$1 \vee$
			×

Line (3) is another disjunction, and will branch again. Note that if we hadn't closed one of the branches, we would have to add branches to both of them, leading to four open branches! It is important to try to close branches as soon as you can, to stop the tree from getting too complex.

1.	$D \vee G$	✓	Root
2.	$\neg D$	✓	Root
3.	$\neg G \vee S$	✓	Root
			
4.	D	G	$1 \vee$
	×		
			
5.	$\neg G$	S	$3 \vee$
	×	↑	

We've closed one of these branches, but we can see that the last branch is still open, and that all the formulas above it either have a check mark, or are atomic. There is nothing more we can do; our tree is complete. We mark the open branch with an \uparrow . This represents the valuation we were seeking.

To read off the valuation (or truth table row, if you prefer), start at the arrow and read all the atomic and negated atomic symbols from the tree. In this case, we get: $S, G, \neg D$. So when $G = 1$, $S = 1$ and $D = 0$, all the formulas are true; they are mutually consistent.

Practice Exercises

★ **Exercise A:** How well do you understand the idea of a truth tree? Try to explain each of the following concepts:

1. What is the Root of a truth tree?
2. When should you tick (✓) a formula, and why?
3. What does a branch of a truth tree represent?
4. When should you create new branches of your tree?
5. When should you close a branch, and why?
6. When is an open branch complete? Why is this important?
7. What does the collection of atomic formulas in an open branch mean?

Chapter 2

Planting your Tree

2.1 Selecting the Right Root

Truth trees are used to indicate whether the root set of formulas is consistent. To use a truth tree to answer a question, the first step is always to convert the question into a question about whether some formula, or set of formulas, is mutually consistent.

Consider how this would apply to validity. An argument in TFL is invalid iff the set of its premises and the negation of its conclusion are mutually consistent. It is valid iff this set of formulas is mutually inconsistent. So we can test for the validity of a TFL argument using a truth tree. If at least one branch of the tree is open, then the set is consistent, and we've found a valuation that makes the premises true and the conclusion false. The argument is invalid, and the valuation is a counter-example. If the tree closes, the argument is valid.

Consistency and validity are the two main notions we are interested in. But we've already looked at several other TFL properties, including tautologies (logical truths), contradictions (logical falsehoods), and equivalences. These can all be tested with truth trees too.

For example, suppose you want to test if a formula is a tautology. To test this with a truth tree, we'll check if the formula can be false – that is whether it is consistent for its *negation* to be true. So we will put the negation of the formula in the root. If this truth tree remains open, it shows us how to make the negation true, so the formula is *not* a tautology. If it closes, the negation is inconsistent (can't be true), so the formula *is* a tautology.

You should be able to devise methods for testing for contradictions and equivalences from our complete and partial truth table methods, and the definitions of these notions. Give this a go, and then test your trees by trying some of the Exercises from previous chapters.

2.2 Growing your Tree

If you have been wondering how to decompose each formula in the tree, we'll introduce the formal rules for TFL truth trees in §3. In the meantime, we are just reading our truth tables backwards, as we did for partial truth tables. We figure out what subformulas have to be true for the current formula to be true, and write those sub-formulas down. If either of two subformulas being true is enough to make the current formula true, then we have a choice. Instead of making that choice, we branch the tree and put one option on each branch. This allows us to delay making our choice until we have more information.

Let's now test an argument for validity. Consider this argument form:

$$(D \vee A) \wedge \neg N, N \vee \neg A \therefore \neg N \wedge A$$

We are looking to see if a counter-example is possible; that is, if the premises are consistent with the conclusion being false. So the root of our tree will have the premises and the negation of the conclusion. We'll be looking for a valuation that makes all these formulas true. It's also useful to note what each part of the root comes from. We'll label the premises 'Premise' and the negated conclusion 'Neg Conc'. This also serves as a reminder to negate the conclusion!

- | | | |
|----|----------------------------|----------|
| 1. | $(D \vee A) \wedge \neg N$ | Premise |
| 2. | $N \vee \neg A$ | Premise |
| 3. | $\neg(\neg N \wedge A)$ | Neg Conc |

Line (1) is a conjunction; it decomposes into its conjuncts on (4) and (5). Line (2) is a disjunction; it branches into its disjuncts on line (6), and then the left branch closes as N and $\neg N$ cannot both be true.

- | | | |
|----|---|------------|
| 1. | $(D \vee A) \wedge \neg N \checkmark$ | Premise |
| 2. | $N \vee \neg A \checkmark$ | Premise |
| 3. | $\neg(\neg N \wedge A)$ | Neg Conc |
| 4. | $D \vee A$ | 1 \wedge |
| 5. | $\neg N$ | 1 \wedge |
| | $\swarrow \quad \searrow$
$N \quad \neg A$ | |
| 6. | | 2 \vee |
| | \times | |

Line (3) is a negated conjunction. A conjunction is true iff both conjuncts are true, so it is false if at least one conjunct is false. As we don't know which conjunct has to be false, the tree branches on line (7), with one negated conjunct on each branch: $\neg\neg N$ and $\neg A$. The first of these branches then closes because it is the negation of line (5). The final formula to decompose is line (4)'s disjunction; it branches on line (8), and one branch closes as it is the negation of line (7).

The remaining formulas are all atomic, so cannot be decomposed.

1.	$(D \vee A) \wedge \neg N$	✓	Premise
2.	$N \vee \neg A$	✓	Premise
3.	$\neg(\neg N \wedge A)$	✓	Neg Conc
4.	$D \vee A$	✓	1 \wedge
5.	$\neg N$		1 \wedge
<div style="text-align: center;">└──┬──</div>			
6.	N		2 \vee
	×		
<div style="text-align: center;">└──┬──</div>			
7.	$\neg\neg N$		3 $\neg\wedge$
	×		
<div style="text-align: center;">└──┬──</div>			
8.	D	A	4 \vee
	↑	×	

The \uparrow indicates that the open branch ending in D is *completed*. (We'll define 'complete' in §3.) This branch represents the valuation we were seeking. This valuation is a counter-example, and so this argument is *not* valid in TFL. We can read the counter-example off the tree by starting at the arrow, reading upwards, and listing the atomic symbols: $\neg A$, D , and $\neg N$.

Our counter-example is: $A = 0, D = 1, N = 0$. A partial truth table can prove the mutual consistency of the formulas in the root:

A	D	N	$(D \vee A)$	\wedge	\neg	N	$N \vee \neg A$	\neg	$(\neg N \wedge A)$
1	1	0	1	1	0	0	0	1	0

All three formulas are true – the counter-example is possible.

On line (7) we wrote $\neg\neg N$. Some students want to write N instead. But that's not what our rules say. We also wrote $\neg A$ on both lines (6) and (7). That's absolutely fine. There's more than one piece of information that led us to the same formula. Again, don't try to out-think the rules and leave out seemingly redundant steps. It will just cause you confusion and error.

2.3 Completing your Tree

A tree is complete when there are no more formulas to decompose on any of its branches. This occurs when each branch has either been closed, or all its complex formulas have already been decomposed. By a complex formula, we mean any formula which is not just an atomic symbol or negated atomic symbol.

Closing a Branch

We *close* a branch when it contains a formula along with its negation:

A branch is CLOSED iff it contains both some formula \mathcal{A} and its negation $\neg\mathcal{A}$. We mark closed branches with the ‘ \times ’ symbol.
 A branch is OPEN iff it is not closed.
 A tree is CLOSED iff every branch in that tree is closed.
 A tree is OPEN iff it is not closed.

If a tree is closed, we have shown that its root is inconsistent.

Completing an Open Branch

An open branch is only complete when we’ve decomposed all its complex formulas. We then write an ‘ \uparrow ’ at the bottom of that branch.

An open branch is COMPLETE iff all its complex formulas have been decomposed, as indicated by a check mark.
 A tree is COMPLETE iff all its branches are closed or complete.

If there is at least one open branch in a completed tree, then each open branch represents a valuation that makes all the formulas in the root true.

Reading the Tree Leaves

To read off the valuation from an open branch on a completed tree, start at the ‘ \uparrow ’ arrow and follow the tree upwards, writing down all the atomic and negated atomic symbols from the tree. An atomic symbol means that atom is true; a negated atomic symbol means that atom is false. If you have both an atomic symbol and its negation in your list, close the branch.

This valuation will correspond to a case where all the formulas in the root are true. The mutual consistency of these formulas will mean different things depending on whether you are testing for consistency, validity, logical truth, logical falsehood, or equivalence, as we’ve already discussed.

Practice Exercises

★ **Exercise A:** List the formulas in the root of your tree when testing whether:

1. Are $(p \rightarrow q), (p \wedge r), (\neg r \vee \neg q)$ consistent?
2. Are $(p \vee \neg q), (q \vee \neg r), (r \vee \neg p)$ consistent?
3. Is $(\neg p \rightarrow p)$ a logical falsehood?
4. Is $((p \rightarrow q) \rightarrow p) \wedge \neg q$ a logical falsehood?
5. Is $((p \rightarrow q) \rightarrow p) \rightarrow p$ a logical truth?
6. Is $((q \vee \neg q) \rightarrow p)$ a logical truth?
7. Are $(p \leftrightarrow q), (\neg(p \wedge q) \leftrightarrow (\neg p \wedge \neg q))$ equivalent?
8. Are $((p \vee q) \wedge (q \vee r) \wedge (r \vee p)), ((p \wedge q) \vee (q \wedge r) \vee (r \wedge p))$ equivalent?
9. Are $(p \leftrightarrow q), (p \leftrightarrow \neg q)$ contradictory?
10. Are $(p \wedge q), (\neg p \wedge \neg q)$ contradictory?
11. Is $(p \rightarrow r), (q \rightarrow r) \therefore (p \rightarrow q) \rightarrow r$ valid?
12. Is $(p \rightarrow r), (q \rightarrow r) \therefore (p \vee q) \rightarrow r$ valid?

Exercise B: Complete the trees from Exercise A. For each closed tree, state what you have proved.

Exercise C: For each open tree from Exercise B, complete the partial truth table, and state what you have proved.

Chapter 3

TFL Tree Rules

The examples we've worked through should have given you an overview of the tree method for TFL. But we haven't precisely defined how to decompose each type of main connective. We are now ready to give formal rules for trees. You should be able to recognize the following rules as a generalization of the steps of the proofs given above.

Our DECOMPOSITION rules describe how the tree can be extended by decomposing formulas. The rules depend on the main connective of the formula. (If the main connective is a negation, then they also depend on the main connective inside that negation.)

3.1 Decomposition Rules

The decomposition rule for conjunction is that if you have a conjunction in a branch, you may make a linear extension of the branch that includes each conjunct, adding a check mark next to the conjunction. So, any time you have a conjunction $\mathcal{A} \wedge \mathcal{B}$ on line (i) you may extend that branch of the tree by writing:

$$\begin{array}{ll} \mathcal{A} & i \wedge \\ \mathcal{B} & i \wedge \end{array}$$

It is important to remember once again that \mathcal{A} and \mathcal{B} here can stand for *any* formula of TFL, including complex ones. The formulas must be copied exactly as written, including any negation signs. Also, write the line # of the decomposed formula, and which decomposition rule you used, on the new line(s). This will help you when you check your work. You do check your work, right?

Conjunction

Our conjunction decomposition rule is:

$$\begin{array}{c} \mathcal{A} \wedge \mathcal{B} \checkmark \\ \mathcal{A} \quad i, \wedge \\ \mathcal{B} \quad i, \wedge \end{array}$$

Negated conjunction

A negated conjunction, branches into the negation of each conjunct. This is because there are two ways for a negated conjunct to be true – either conjunct can be false:

$$\begin{array}{c} \neg(\mathcal{A} \wedge \mathcal{B}) \checkmark \\ \swarrow \quad \searrow \\ \neg\mathcal{A} \quad \neg\mathcal{B} \quad i, \neg\wedge \end{array}$$

Disjunction

Disjunctions branch into each disjunct:

$$\begin{array}{c} \mathcal{A} \vee \mathcal{B} \checkmark \\ \swarrow \quad \searrow \\ \mathcal{A} \quad \mathcal{B} \quad i, \vee \end{array}$$

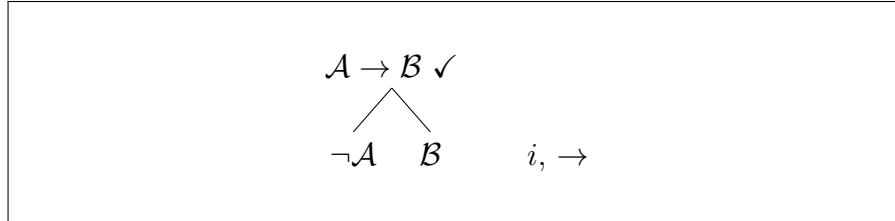
Negated disjunction

Since disjunctions are true any time either disjunct is true, they are only false if both disjuncts are false; that is, their negations are both true:

$$\begin{array}{c} \neg(\mathcal{A} \vee \mathcal{B}) \checkmark \\ \neg\mathcal{A} \quad i, \neg\vee \\ \neg\mathcal{B} \quad i, \neg\vee \end{array}$$

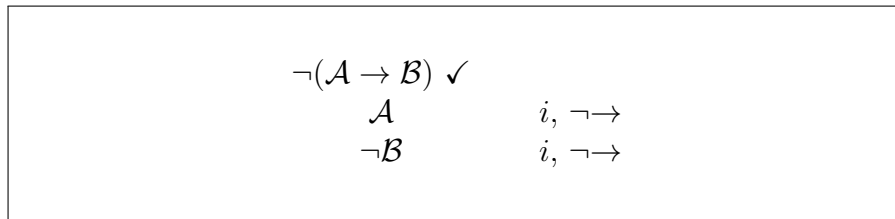
Conditional

Conditionals are true when the antecedent is false *or* the consequent is true. This means that conditionals in TFL are treated similarly to disjunctions:



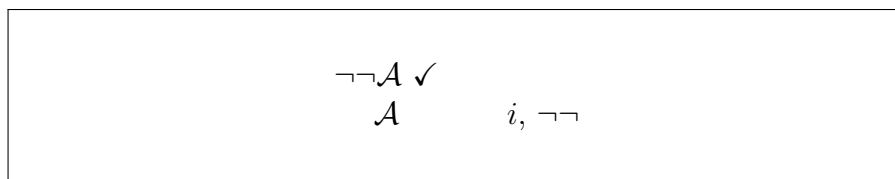
Negated conditional

The negation of a conditional is true when the conditional is false. That is, when its antecedent is true *and* its consequent is false:



Double negation

A doubly-negated formula is true iff the singly-negated formula is false iff the original formula is true. So we can simply remove a double negation. However, this has to be done as a separate step:

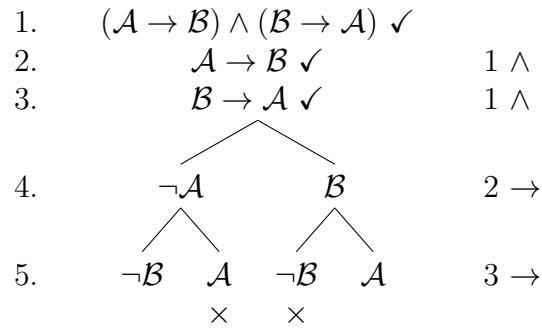


Logic Corner

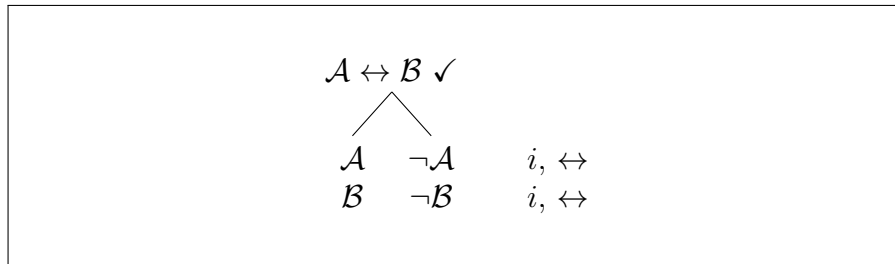
You may have noticed that the decomposition rules for the standard connectives and their negations are duals of each other – if one branches, the other doesn't, if one negates \mathcal{A} , the other does not, and so forth. This means that if you know the decomposition rule for a connective, you also know the rule for its negation. Just do everything the opposite way!

Biconditional

Biconditionals are really just a combination of two conditionals. That is, $\mathcal{A} \leftrightarrow \mathcal{B}$ iff $(\mathcal{A} \rightarrow \mathcal{B}) \wedge (\mathcal{B} \rightarrow \mathcal{A})$. If we decompose this formula, we get:

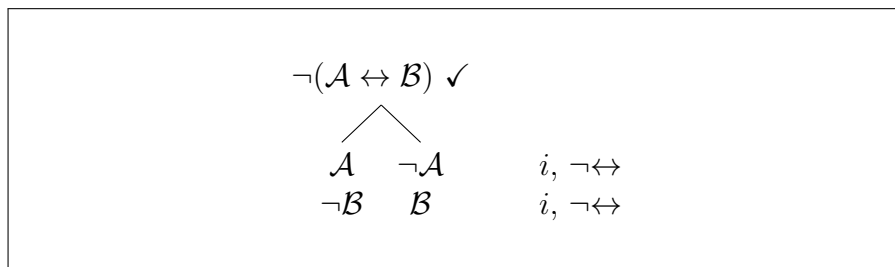


One open branch has \mathcal{A} and \mathcal{B} ; the other $\neg \mathcal{A}$ and $\neg \mathcal{B}$. This might seem odd, but recall that a biconditional is true when its sub formulas have the same truth value: either both are true, or both are false. And that's exactly what this rule states:



Negated biconditional

Negated biconditionals are true when the subformulas have different truth values. The tree rule looks remarkably similar to the standard biconditional:



You can derive this rule too, by creating a tree for the negation of a conjunction of conditionals: $\neg((\mathcal{A} \rightarrow \mathcal{B}) \wedge (\mathcal{B} \rightarrow \mathcal{A}))$.

3.2 Truth Tree Complexities

Incomplete Valuations

Sometimes our open branch will not contain every atom in our set of formulas. That is, our valuation does not specify the truth value of some atomic symbols. When this occurs, the truth value of the remaining atomic symbols does not affect the consistency of the formulas. However, a valuation must specify the truth value of every symbol, so you can arbitrarily pick some truth value for each, or indicate its truth value is unknown by a ‘?’.

For example, suppose an open branch for a set of formulas using the symbols $\{p, q, r\}$ contains only p and $\neg q$. This valuation would be: p is true, q is false, r is unknown; or in symbols: $p = 1, q = 0, r = ?$.

Early Closing

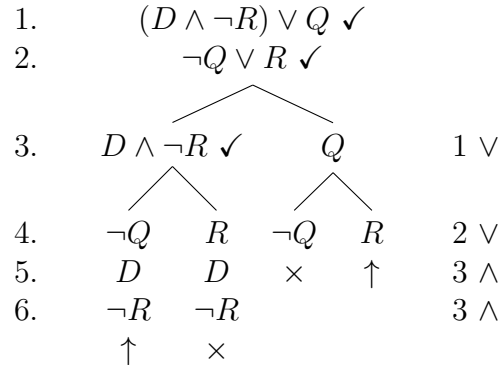
Note that we close a branch when it contains *any* formula and its negation. We don’t need an atomic formula and its negation. We saw this in an early example when we closed a branch containing $\neg\neg N$ and $\neg A$. Here is another example:

$$\begin{array}{ll}
 1. & (\neg S \wedge T) \rightarrow (\neg P \leftrightarrow (Q \vee R)) \checkmark \\
 2. & \neg S \wedge T \\
 3. & \neg(\neg P \leftrightarrow (Q \vee R)) \\
 & \swarrow \quad \searrow \\
 4. & \neg(\neg S \wedge T) \quad \neg P \leftrightarrow (Q \vee R) \\
 & \times \qquad \qquad \times
 \end{array}$$

In this tree, we only decomposed the first formula – using the conditional rule – and then the tree immediately closed. The left branch of (4) is the negation of (2), and the right branch is the negation of (3). Notice that we *could* have ignored (or missed) this, and decomposed lines (2) and (3) using the conjunction or negated biconditional rules, respectively. However, this would have resulted in a more complicated tree that would also have eventually closed. Missing an early closing opportunity will never affect the final result of a truth tree, just create more work and a higher chance of error. You can occasionally save yourself a lot of effort by noticing when branches are ready to mark as closed. However, if you don’t notice this early closing, no real harm will be done.

Multiple Open Branches

If you are decomposing a formula after the tree has already branched, you must add the new formulas under *each* open branch that descends from that formula. Here is a tree illustrating how to do this:



This tree has two disjunctions in the root, so it will branch multiple times. We decomposed line (1) first, branching into the two disjuncts at line (3). When we decomposed (2), both branches descending from (2) were still open, so we added line (4) to both these branches. That's why our two branches split into four at line (4). The third branch contains Q and $\neg Q$, and so closes. At line (5), the conjunction $D \wedge \neg R$ at line (3) is decomposed. Only the left-most 2 of the 3 open branches descend from this conjunction. So the conjuncts need to be added to both branches that descend from that conjunction, but not the right-most branch, which does not.

Multiple Valuations

A tree can have more than one completed open branch. The tree above is an example of this. These completed open branches can have different valuations. This is because there can be more than one counter-example, or row that doesn't satisfy the property you are testing. It is fine to use any completed open branch. However, be careful.

First, don't combine your open branches. In the tree above, one valuation has $Q = 1, R = 1$, and the other $Q = 0, R = 0$. Any combination of these would fail to be a valuation where all the formulas in the root are true.

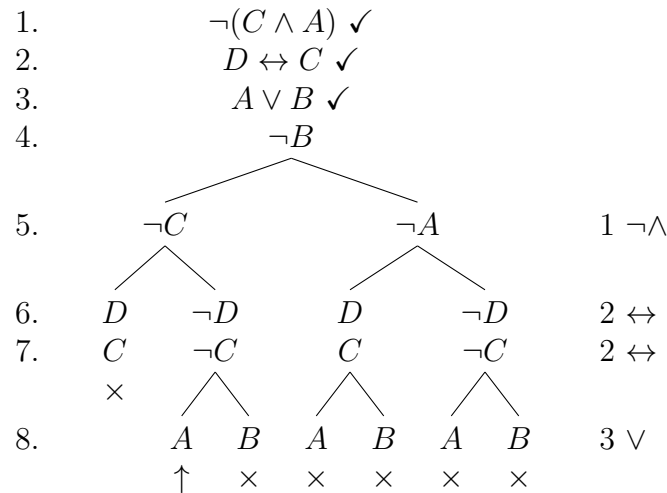
Second, indicate clearly which open branch your counter-example comes from. We suggest that you only use the symbol \uparrow on the completed branch which you are reading your valuation from. Leave the other open branches blank.

3.3 Order of Decomposition

You do *not* need to decompose formulas in strict order from the top of the tree. You can decompose formulas in whichever order you find convenient. Some ways of decomposing trees are more efficient than others. The most basic principle is to decompose formulas that won't branch before those that will. When all the remaining complex formulas will branch, try to look a step ahead, and select a formula that will have a branch that closes immediately. This reduces the likelihood of having multiple open branches. Next, decompose simple formulas before complex ones, as you are less likely to muck this up. Finally, I suggest you decompose biconditionals last, as they have the most complex rules of all.

1. Non-branching rules before branching.
2. Close branches quickly.
3. Simple formulas before complex.
4. Biconditionals last.

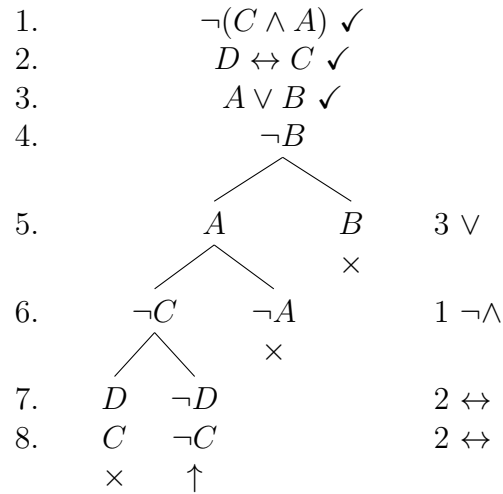
Suppose we want to test whether $\neg(C \wedge A), D \leftrightarrow C, A \vee B, \neg B$ are mutually consistent. These formulas would then form the root of our tree. Here's the tree formed if we decompose the formulas in the order in which they're listed:



This completed tree is open. We can read off the valuation from its one open branch. The atomic and negated atomic symbols on this open branch are: $A, \neg C, \neg D, \neg C, \neg B$. This tells us that a valuation where all the formulas are true is $A = 1, B = 0, C = 0, D = 0$.

This tree suffers from the problem of having multiple open branches during its construction. That's why, for example, decomposing line (3) at line (8) requires three different new branchings. So when there are more open branches, decomposing formulas requires more work on the tree, more repetition, and more chance of errors.

The tree above is a perfectly fine completed tree, and it gives a correct answer. However, it's possible to get there with less work, by choosing to decompose formulas that will close off branches right away. Here is another tree with the same root as in the previous example, but which decomposes formulas in a more efficient order:



We can use a partial truth table to confirm our answer:

A	B	C	D	\neg	$(C \wedge A)$	$(D \leftrightarrow C)$	$A \vee B$	$\neg B$
1	0	0	0	1	0	0	1	0

This tree gets us to the same result much more quickly, and generates the same valuation. Here's a description of the reasoning for deciding which formula to decompose at each step:

5. Line (4) contains a $\neg B$. Line (3) would generate a branch with B .
6. Line (5) contains an A . Line (1) would generate a branch with $\neg A$.
7. Line (6) contains a C . Line (2) would generate a branch with $\neg C$.

You usually don't need to think more than one line ahead to select branches that create a much simpler tree.

Practice Exercises

★ **Exercise A:** To evaluate each of the following claims with a tree, (a) what would you put in the root of the tree?, and (b) if the tree closes, does that show that the claim is true or false?

1. $P, P \rightarrow Q, Q \rightarrow \neg P$ is mutually consistent
2. $(P \rightarrow Q) \leftrightarrow (Q \rightarrow P)$ is a tautology.
3. The following argument is valid: $P \wedge Q, \neg R \rightarrow \neg Q \therefore P \wedge R$
4. Every valuation making $P, P \rightarrow Q$, and $\neg Q$ true also makes A true.
5. There is no valuation that makes $A \vee B, B \rightarrow C$, and $A \leftrightarrow C$ all true but C false.
6. $A \leftrightarrow \neg A$ is a contradiction.
7. There is at least one valuation that makes $P \rightarrow Q, \neg P \vee \neg Q$, and $Q \rightarrow P$ true.

★ **Exercise B:** Evaluate each claim from Exercise A by constructing a tree. If applicable, give the valuation that demonstrates the claim true or false, and use a partial truth table to confirm this.

★ **Exercise C:** Evaluate the argument

$$A \leftrightarrow B, \neg B \rightarrow (C \vee D), E \rightarrow \neg C, (\neg D \wedge F) \vee G, \neg A \wedge E, \therefore H \vee G$$

by constructing a tree. If the tree is open, prove your counter-example is correct using a partial truth table.

Exercise D: The Sheffer stroke $\mathcal{A} \mid \mathcal{B}$ is a rarely-used connective of TFL. It has the following characteristic truth table:

\mathcal{A}	\mathcal{B}	$\mathcal{A} \mid \mathcal{B}$
1	1	0
1	0	1
0	1	1
0	0	1

Create appropriate tree decomposition rules for the Sheffer stroke, and for the negated Sheffer stroke.

Exercise E: Construct trees for any of the exercises from the complete and partial truth tables chapters. Check the valuations from any open branches are correct using partial truth tables.