Six Tips for Truth Trees

David W. Agler
2/17/14

Tip #1: Start every tree by asking yourself:

How should I set up the tree?

For example, suppose you are given the following propositions:

$P \wedge R$, $S \rightarrow T$, $Q \leftrightarrow \neg\neg L$

and are asked to see if they are **consistent** or **inconsistent**. In order to test for this, you must set up the tree in a specific way. In testing for consistency / inconsistency, you just look at all of the propositions and stack them on top of each other.

1	$P \wedge R$	P
2	$S \rightarrow T$	P
3	$Q \leftrightarrow \neg\neg L$	P

But this is not true if you are testing for a different property! How you set up a tree depends upon what property you are testing for!

Tip #2: In decomposing the tree, in order to get the right answer, it doesn't matter which proposition you start to decompose.

1	$P \wedge R$	P
2	$S \rightarrow T$	P
3	$Q \leftrightarrow \neg\neg L$	P

You could start by decomposing (1) or (2) or (3). It doesn't matter. Choosing one proposition over another might make decomposition *easier* but getting the *right* answer does not depend upon which proposition you decompose first.

Tip #3: When thinking about what decomposition rule to apply, you should always ask yourself the following question:

What type of proposition am I looking at?

A *conditional* $P \rightarrow R$ can only be decomposed using $\rightarrow$**D** (conditional decomposition)

A *negated conditional* ¬(P→R) can only be decomposed using **¬→D** (negated conditional decomposition). Here is an example:

1	P∧Z	P
2	P→R	P

I want to decompose line 1. It is a conjunction and so I have to apply ∧D.

1	P∧Z	P
2	P→R	P
3	P	1∧D
4	Z	1∧D

Now I want to decompose line 2. It is a *conditional* and so I have to apply →D

1	P∧Z	P
2	P→R	P
3	P	1∧D
4	Z	1∧D
2	¬P | R	2→D

Tip #4: Pay attention to whether the decomposition rule you are applying calls for *stacking* or *branching* or *stacking and branching*.

It is important to note that there is a difference between a disjunction 'P∨R', which is decomposed using ∨D, and which branches, and a negated disjunction '¬(P∨R)' which stacks.

Line #1 is a disjunction so it branches!

1	P∨R	P
2	P | R	1∨D

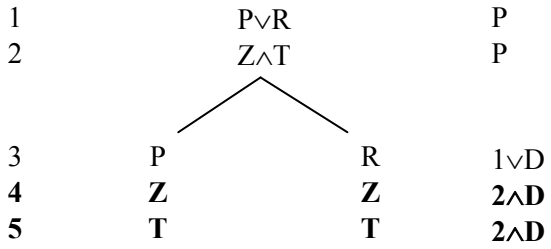
Line #1 is a negated disjunction so it stacks!

1	¬(P∨R)	P
2	¬P	1¬∨D
3	¬R	1¬∨D

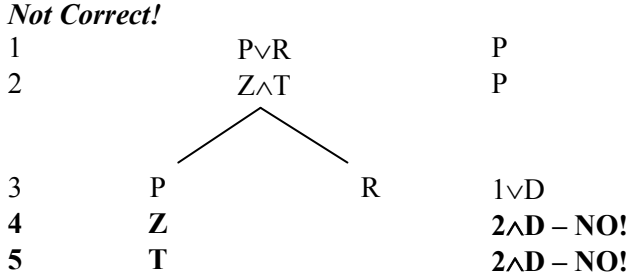
Tip #5: When you decompose a proposition P, you need to decompose it under every open branch that descends from P.

Here is an example of this rule being used correctly:

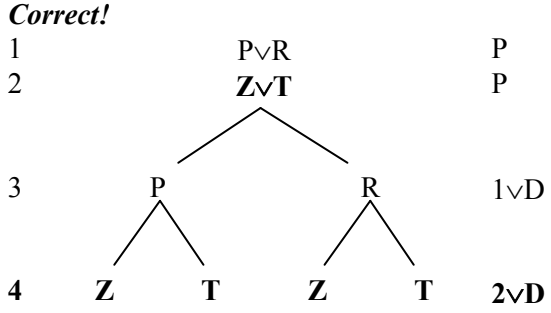
Correct!



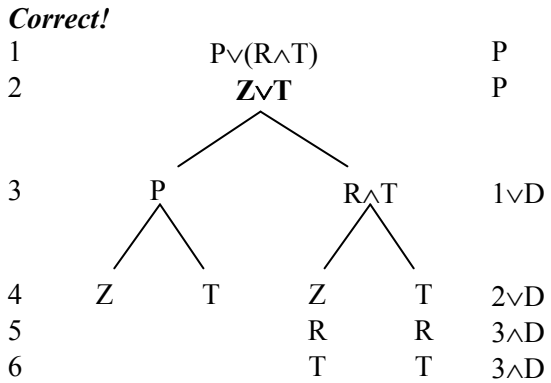
Here is an example of this rule being used incorrectly



This is the case even when you have to branch



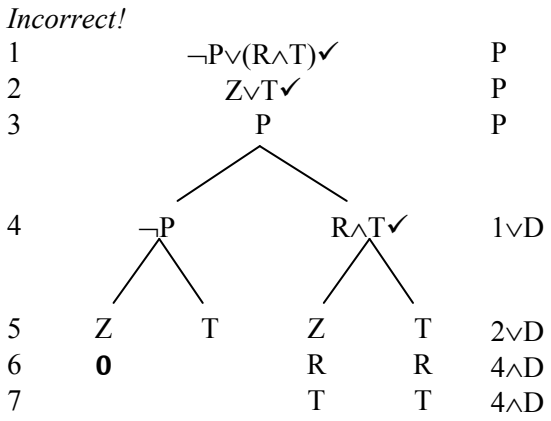
But notice that this rule says **"under every open branch that descends from P (the proposition you are decomposing)"**



Notice that R∧T only needs to be decomposed under the right hand side and not the left hand side.

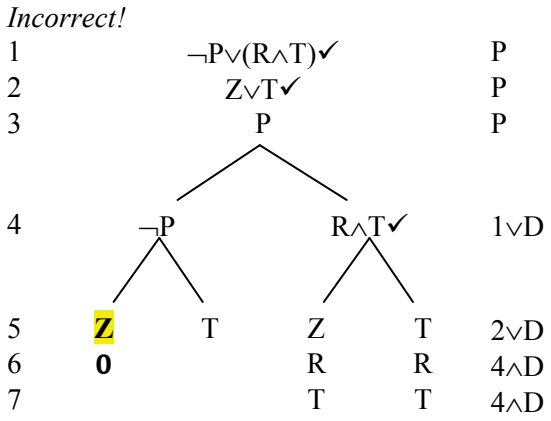
Tip #6: When determining whether a branch is closed, look at all of the propositions in the branch.

Consider the following completely decomposed tree:



You might look at the tree and reason as follows:

> ah! there are *four* branches in the above tree. I have completely decomposed the far left branch, and look, there is **Z** but there isn't **¬Z** and so the branch is a completed open branch.



But this is incorrect! In order to determine if a branch is open or closed, you need to look at *all* of the fully decomposed propositions in the branch and see if there is a proposition **P** and its literal negation ¬(**P**). Notice that leftmost branch consists of three fully decomposed propositions:

> Branch #1: Z, ¬P, P (closed!)

Correct!