# 18 Introducing PL/PLC truth trees

In earlier chapters we defined what it is for PL and PLC inferences to be tautologically valid. We then introduced one way of testing whether an inference is tautological valid, namely the brute-force *truth table* method, where we search through every possible assignment of values to the relevant atoms to determine whether there is a 'bad' assignment which makes the premisses true and the conclusion false.

We now want to explore another way of testing for tautological validity – one that is usually faster in practice, often a *lot* faster. So, in this chapter, we introduce the *truth tree* method by talking through a number of examples quite slowly and informally. We will then give a brisker official story about truth trees in the next chapter.

In our preferred 'unsigned' version, truth trees are rather elegant as well as speedy. However, you *can* easily live without them; so do feel free to skip forward to the chapters on natural deduction, starting with Chapter [**ref**], or even to the initial chapters on the language of quantificational logic, starting with Chapter [**ref**].

## 18.1 Four quick preliminaries

(1) In this chapter and the following one, take wffs in all our examples to belong to some interpreted PL/PLC language or other. And take 'valid'/'invalid' always to mean *tautologically* valid/invalid.

(2) It will be useful to have a word for those wffs which are either atoms or negations of atoms: we will here use 'simple'. Other wffs are therefore 'complex'.

(3) To avoid ugly rashes of quotation marks, we will start allowing ourselves to write the likes of

$$\mathsf{P} \Rightarrow \mathrm{T},\ \mathsf{Q} \Rightarrow \mathrm{F},\ \neg(\mathsf{Q} \vee \mathsf{R}) \Rightarrow \mathrm{T},\ (\mathsf{P} \rightarrow (\mathsf{Q} \rightarrow \mathsf{R})) \Rightarrow \mathrm{T}, \ldots$$

without quotation marks around the wffs (compare §§10.3, 10.4). Just as we treat '$\vDash$' as generating its own quotation marks (see §15.1), we will now treat '$\Rightarrow$' as generating quotation marks on the left.

(4) Most significantly, in this chapter we will usually prefer to say of a wff, not that it is *false* (on a given valuation of its atoms), but rather that its *negation is true*. For example, we will now prefer to say that an inference is (tautologically) valid if and only if there is no relevant valuation which makes the premisses and the negation of the conclusion all true. Obviously nothing can turn on this way of putting things; in our classical two-valued logical framework, a

wff is false if and only if its negation is true. But there is a reason for this apparently quirky preference for talking of truth (of negations) rather than of falsity, which will become clear in §18.4.

## 18.2 'Working backwards'

(a) Now down to work. At the end of §14.4, we considered the following inference,

**A** $\neg P, \neg P', \neg P'', \neg P''', \ldots, \neg P''\cdots'' \therefore Q$

where there are fifty similar premisses, each the negation of an atom. The argument is patently invalid – and it would be quite crazy to try to write down a full truth table to confirm this (which would take millennia). It is enough to remark that **A** is invalid if and only if there is a valuation of the fifty-one atoms such that

$$\neg P \Rightarrow T, \neg P' \Rightarrow T, \neg P'' \Rightarrow T, \neg P''' \Rightarrow T, \ldots, \neg P''\cdots'' \Rightarrow T, \neg Q \Rightarrow T.$$

Then we just note that we can immediately read off such a valuation, namely

$$P \Rightarrow F,\ P' \Rightarrow F,\ P'' \Rightarrow F,\ P''' \Rightarrow F,\ \ldots, P''\cdots'' \Rightarrow F,\ Q \Rightarrow F.$$

That settles it (in rather less than a millennium): **A** is invalid.

(b) Take next the argument

**B** $(P \wedge \neg Q), (R \wedge \neg S) \therefore (Q \vee S)$

This too is obviously invalid. We could run up a manageable sixteen-line truth table to show that. But we could also argue as follows, setting out our reasoning step by step.

The inference **B** is invalid if (and only if) there is a valuation of the four relevant atoms which makes the premisses true and the negation of the conclusion true too, i.e. a valuation such that

(1) $(P \wedge \neg Q) \Rightarrow T$
(2) $(R \wedge \neg S) \Rightarrow T$
(3) $\neg(Q \vee S) \Rightarrow T$

Now consider what it takes for (1) to (3) to obtain. First, note that by the truth table for conjunction, (1) holds if and only if these claims hold too:

(4) $P \Rightarrow T$
(5) $\neg Q \Rightarrow T$

Similarly, (2) holds if and only if these claims do too:

(6) $R \Rightarrow T$
(7) $\neg S \Rightarrow T$

Finally, (3) holds if and only if the unnegated disjunction '$(Q \vee S)$' is false, which requires that both disjuncts are false. Avoiding talk of falsehood, this means that the negation of each disjunct has to be true:

(8) $\neg Q \Rightarrow T$
(9) $\neg S \Rightarrow T$

So, putting everything together, we can ensure that (1) to (3) hold – i.e. we can make the premisses and negated conclusion of **B** all true – by taking the valuation where $P \Rightarrow T$, $\neg Q \Rightarrow T$ (i.e. $Q \Rightarrow F$), $R \Rightarrow T$, and $\neg S \Rightarrow T$ (i.e. $S \Rightarrow F$). Hence **B** is invalid.

(c) For a contrasting third example, consider

**C** $(P \wedge \neg Q) \therefore \neg(Q \wedge R)$

Again, the inference is invalid just if there is a valuation of the relevant three atoms which makes the premiss and negated conclusion both true:

(1) $(P \wedge \neg Q) \Rightarrow T$
(2) $\neg\neg(Q \wedge R) \Rightarrow T$

For (1) to obtain, we need

(3) $P \Rightarrow T$
(4) $\neg Q \Rightarrow T$

By the truth table for negation applied twice, for (2) to obtain, we need

(5) $(Q \wedge R) \Rightarrow T$

and hence we need both

(6) $Q \Rightarrow T$
(7) $R \Rightarrow T$

But hold on! We have now hit a contradiction. *No valuation can make* $\neg Q$ *and* $Q$ *simultaneously true together*. So there can not possibly be a valuation satisfying both (1) and (2) and making the premiss of **C** true and conclusion false.

Which shows that **C** is valid.

(d) In our three examples we have been – so to speak – 'working backwards'.

On the brute-force truth-table method, in order to decide whether a given argument is tautologically valid, we start by laying out all the possible valuations for the relevant atoms. Then we plod forwards through each valuation, calculating the truth values of the premisses and conclusion as we go, looking for a 'bad' valuation. Now we are starting on the same decision task from the other end. In other words, we suppose that there *is* a bad valuation, one where the premisses are true and the negation of the conclusion is true too. Then we try to unravel the implications of this supposition, working backwards towards uncovering a valuation of the relevant atoms that is indeed bad.

If we *can* uncover a bad valuation, then obviously the argument being tested is not valid. On the other hand, if we get inextricably entangled in contradiction as we work backwards, then (by an informal reductio ad absurdum inference) that shows that there is no bad valuation after all, and the argument in question is valid.

This is, in headline terms, the appealing strategy that we are exploring in this chapter. Though as we will see in the next section, things don't in general go *quite* as simply as our first easy examples might suggest.

(e) Before moving on, one quick comment about our example **C**. Note that at line (2) we gave the negation of the conclusion '$\neg(Q \wedge R)$' as '$\neg\neg(Q \wedge R)$' with its extra prefixed

negation sign. Then at line (5) we stripped away the double negation again by using the obvious rule that tells us that $\neg\neg\alpha \Rightarrow$ T holds just when $\alpha \Rightarrow$ T. Why go through this little two-step dance? Why not immediately jump to treating the negation of something of the form $\neg\alpha$ as plain $\alpha$?

Because, in more advanced work (beyond the scope of this book), you will eventually encounter important logical systems with a non-classical understanding of negation. In particular, you will meet so-called intuitionistic logic where $\neg\neg\alpha$ is in general not equivalent to $\alpha$. It is therefore not a good idea to get into the habit of simply ignoring double negations (even if that is classically permitted). For us, then, negating a PL or PLC wff is *always* a matter of adding a prefixed negation sign, as already noted in §9.1(d).

## 18.3 Why trees?

(a) Here is an obviously invalid little argument:

**D** $(\mathsf{P} \vee \mathsf{Q}) \therefore \mathsf{P}$

What happens in this case if we try working backwards to expose a valuation which makes the premiss true and negated conclusion true too?

We start, as before, with the assumption that there *is* a bad valuation, one which makes the following hold:

(1) $(\mathsf{P} \vee \mathsf{Q}) \Rightarrow$ T
(2) $\neg\mathsf{P} \Rightarrow$ T

Given the truth table for disjunction, (1) tells us that either (3a) $\mathsf{P} \Rightarrow$ T or (3b) $\mathsf{Q} \Rightarrow$ T or perhaps both – and either alternative suffices to verify the disjunction. *But (1) doesn't tell us which alternative holds*. We will have to consider the two cases in turn.

In fact, the alternative (3a) where $\mathsf{P} \Rightarrow$ T need not detail us long, for it is immediately inconsistent with (2), i.e. with $\neg\mathsf{P} \Rightarrow$ T.

The alternative (3b) where $\mathsf{Q} \Rightarrow$ T fares better; there is no contradiction to be found this time. The resulting valuation on which $\neg\mathsf{P} \Rightarrow$ T (i.e. $\mathsf{P} \Rightarrow$ F), $\mathsf{Q} \Rightarrow$ T makes the premiss of **D** true and the negated conclusion true too. As we knew all along, then, the inference **D** is indeed invalid.

(b) We want a perspicuous way of setting out argumentation like this when our path forks, and we have to explore alternatives. The obvious device is to draw forking branches in our chains of reasoning. We then get (upside down, trunk-at-the-top) *truth trees*, and can display our reasoning about **D** like this:

| | | |
|---|---|---|
| (1) | $(\mathsf{P} \vee \mathsf{Q}) \Rightarrow$ T | (premiss assumed true) |
| (2) | $\neg\mathsf{P} \Rightarrow$ T | (negated conclusion assumed true) |
| (3) | $\mathsf{P} \Rightarrow$ T  $\mathsf{Q} \Rightarrow$ T | (alternatives from 1) |
| | ✻ | (contradiction on left branch!) |

This mode of presentation should be self-explanatory. The main novelty compared with our informal reasoning a moment ago is that, instead of considering options sequentially,

we consider them side-by-side; we split our reasoning into left and right branches as we consider the alternative ways of making (1) come out true.

The other novelty is that we now begin to use '✻' as an *absurdity sign*, signalling that we have hit a contradiction on a path down the truth tree from the trunk-at-the-top. In other words, '✻' at the end of a path indicates that it features a pair of wffs $\alpha$ and $\neg\alpha$ which are both assigned T (note that the relevant $\alpha$ doesn't have to be an atom).

Our miniature downward branching tree, then, has two paths through it from top to bottom, as we explore alternative potential ways of making true the initial claims at the top of its trunk. One path is, as we will say, *closed* with an absurdity marker; but the other path remains *open*. And from the open path we can read off a perfectly consistent assignment of values to the relevant atoms which satisfies (1) and (2), making **D** invalid.

(c) Compare next the following argument:

**E** $\mathsf{P}, \neg(\mathsf{P} \wedge \neg\mathsf{Q}) \therefore \mathsf{Q}$

We again want to see whether we can consistently make the premisses and negated conclusion all true:

| | | |
|---|---|---|
| (1) | $\mathsf{P} \Rightarrow \mathrm{T}$ | (premiss assumed true) |
| (2) | $\neg(\mathsf{P} \wedge \neg\mathsf{Q}) \Rightarrow \mathrm{T}$ | (premiss assumed true) |
| (3) | $\neg\mathsf{Q} \Rightarrow \mathrm{T}$ | (negated conclusion assumed true) |

Now, a conjunction is false when at least one conjunct is false. So a negated conjunction is true just when at least one negated conjunct is true. In other words, given $\neg(\alpha \wedge \beta) \Rightarrow \mathrm{T}$ we must have at least one of $\neg\alpha \Rightarrow \mathrm{T}$ or $\neg\beta \Rightarrow \mathrm{T}$. This is the principle we can apply in moving from (2) to the alternatives at (4):

(4) $\neg\mathsf{P} \Rightarrow \mathrm{T}$ ✻ $\quad$ $\neg\neg\mathsf{Q} \Rightarrow \mathrm{T}$ ✻ $\quad$ (alternatives from 2)

This time *both* alternatives immediately produce contradictory assignments of truth values ($\mathsf{P} \Rightarrow \mathrm{T}$ and $\neg\mathsf{P} \Rightarrow \mathrm{T}$ on the left-branching path; $\neg\mathsf{Q} \Rightarrow \mathrm{T}$ and $\neg\neg\mathsf{Q} \Rightarrow \mathrm{T}$ on the right-branching path). So both paths get closed off with the absurdity sign. We see that neither of the potential ways of satisfying conditions (1) to (3) is consistent. Therefore the inference **E** has to be valid.

(d) For another example, take the argument

**F** $(\mathsf{P} \to \mathsf{Q}), (\mathsf{Q} \to \mathsf{R}) \therefore (\mathsf{P} \to \mathsf{R})$

We showed that this is valid by a brute-force truth-table test in §16.4. Let's now prove it valid by 'working backwards'. So assume the argument *isn't* valid, and aim for absurdities. We start, then, by assuming that there is a valuation satisfying

| | | |
|---|---|---|
| (1) | $(\mathsf{P} \to \mathsf{Q}) \Rightarrow \mathrm{T}$ | (premiss assumed true) |
| (2) | $(\mathsf{Q} \to \mathsf{R}) \Rightarrow \mathrm{T}$ | (premiss assumed true) |
| (3) | $\neg(\mathsf{P} \to \mathsf{R}) \Rightarrow \mathrm{T}$ | (negated conclusion assumed true) |

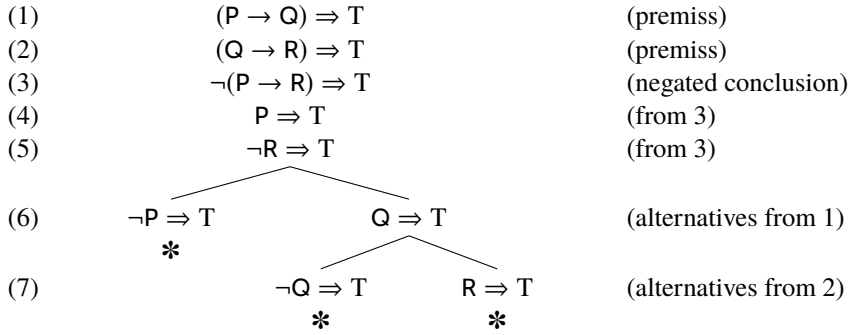
How do we 'unpack' the implications of assuming a material conditional is true? Such a conditional is true, remember, if either its antecedent is false or consequent is true. So,

avoiding talk of falsity, the unpacking rule is as follows: given $(\alpha \rightarrow \beta) \Rightarrow \mathrm{T}$, we have to consider the alternatives $\neg\alpha \Rightarrow \mathrm{T}$ or $\beta \Rightarrow \mathrm{T}$.

And what are the implications of assuming a negated material conditional is true, i.e. that the conditional is false? This requires its antecedent to be true and conclusion false. So, again avoiding talk of falsity, the unpacking rule is as follows: given $\neg(\alpha \rightarrow \beta) \Rightarrow \mathrm{T}$, then we must have both $\alpha \Rightarrow \mathrm{T}$ and $\neg\beta \Rightarrow \mathrm{T}$.

Which unpacking rule – to continue with that vivid metaphor – shall we apply first in constructing our tree? Let's start by unpacking the implications of (3), because doing things in this order delays having to consider branching possibilities and keeps our tree tidier. As we will confirm later, the order in which we tackle complex wffs can't make any difference to the eventual verdict on the argument which we are testing with a tree.

So, following that advice, here's our completed truth tree:




What has gone on here? Having drawn out the implications of (3), we consider the implications of (1), exploring alternatives. The left-hand path, though, can be immediately closed – that doesn't give us a consistent way of ensuring (1) and (3) hold.

The right-hand path stays open at (6). But we haven't yet drawn out the implications of (2). This requires us to consider alternatives again, leading to our second fork in the tree. And this time, we immediately hit a contradiction on each new branch.

Which all goes to show that there is no consistent way of working backwards to make (1), (2) and (3) all correct. Hence, as we want, **F** is shown to be valid.

(e) Note, it would evidently be equally correct to use the following rule for setting out the implications of a claim of the form $(\alpha \rightarrow \beta) \Rightarrow \mathrm{T}$. We could say that one of these *three* options must hold: $\alpha \Rightarrow \mathrm{T}, \beta \Rightarrow \mathrm{T}$ or else $\neg\alpha \Rightarrow \mathrm{T}, \beta \Rightarrow \mathrm{T}$ or else $\neg\alpha \Rightarrow \mathrm{T}, \neg\beta \Rightarrow \mathrm{T}$ (corresponding to the three lines on the truth table where the material conditional comes out true). So why not use this rule which would lead to three-way branching? Aesthetics! It is just prettier to keep our trees binary if we can, with only two branches starting at any point where we have to explore alternatives.

(f) We take just one more example in this section, in order to highlight two crucial points about constructing branching trees like these. So consider:

**G** $((\mathsf{P} \wedge \mathsf{Q}) \vee \mathsf{R}) \therefore \neg(\neg\mathsf{P} \vee \neg\mathsf{R})$

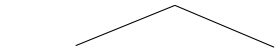
We are looking as before for a valuation which makes the premiss and the negated conclusion both true, i.e. a valuation which gives us

(1) $((P \wedge Q) \vee R) \Rightarrow T$ (premiss)
(2) $\neg\neg(\neg P \vee \neg R) \Rightarrow T$ (negated conclusion)

Trivially, (2) holds just when we have

(3) $(\neg P \vee \neg R) \Rightarrow T$ (from 2)

To unpack the implications of (1) we now need to consider alternative scenarios:



(4) $(P \wedge Q) \Rightarrow T$ $R \Rightarrow T$ (alternatives from 1)

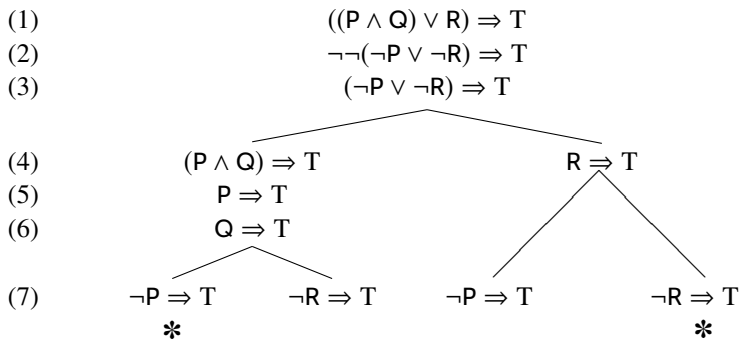
For '$(P \wedge Q)$' to be true, both the conjuncts have to be true. So we can continue the left branch as follows:

(5) $P \Rightarrow T$ (from 4)
(6) $Q \Rightarrow T$ (from 4)

But note – and this illustrates the first obvious point we need to emphasize – these additional conditions '$P \Rightarrow T$' and '$Q \Rightarrow T$' *are added to the left branch only*. For it is only on the scenario described on the path going left at (4) that we are assuming '$(P \wedge Q) \Rightarrow T$' holds; and so it is only while further elaborating this scenario that we are committed to the implications of that assumption. Generalizing:

> Given a statement $\alpha \Rightarrow T$ appearing on a tree, with $\alpha$ complex, then when we unpack the (perhaps alternative) implications of this statement, we add the results *only* to the open paths through the tree which actually contain that statement!

Moving on, we haven't yet unpacked (3) '$(\neg P \vee \neg R) \Rightarrow T$'. But note that this assumption sits both on the path going left from (1) to (6) and also on the path going right from (1) to (4). Since (3) is in play whichever route down the tree we follow, we can go on to consider its implications either way. Now, to keep track of when we have fully unpacked the implications of assigning truth to some wff, the simplest policy is to add these implications to all relevant paths at the same time. So the next step in continuing our tree produces this:




As required, then, at line (7) we have added the alternatives generated by (3) to both open paths containing (3). And generalizing gives us the second point we need to emphasize:

Given a statement $\alpha \Rightarrow$ T appearing on a tree, with $\alpha$ complex, then when we unpack the (perhaps alternative) implications of this statement, we add the results to *all* the open paths through the tree which actually contain that statement.

Two of the four extended paths generated by unpacking (3) can be immediately closed off. And there remain no more complex wffs to unpack, so there are no more lurking contradictions to be exposed on the remaining two open paths – i.e. those two paths cannot be made to close.

The left-hand open path contains the assignment of values $\mathsf{P} \Rightarrow$ T, $\mathsf{Q} \Rightarrow$ T, $\neg\mathsf{R} \Rightarrow$ T (i.e. $\mathsf{R} \Rightarrow$ F). This valuation gives us one way of making (1) and (2) correct, i.e. one way of making **G**'s premiss true and its negated conclusion true too (why? – we will say more about this shortly, but for now check that this claim is correct).

The other open path contains the assignment $\neg\mathsf{P} \Rightarrow$ T (i.e. $\mathsf{P} \Rightarrow$ F), and $\mathsf{R} \Rightarrow$ T. Those values also make (1) and (2) correct, however we settle the value of '$\mathsf{Q}$' (why?). Hence, depending on the choice we then make for '$\mathsf{Q}$', this gives us two more ways of making **G**'s premiss true and its negated conclusion true too.

So our tree reveals three different 'bad' valuations of the relevant atoms making the argument's premiss true and conclusion false. Check that this agrees with the verdict of a brute-force truth table.

(g) Let's pause for a brisk interim summary about the tree test for tautological validity. (To be sure, more work needs to be done, and will be done, to elaborate and really nail down the following claims – though hopefully these headlines should already look plausible.)

We can test an inference for tautological validity by assuming that it is invalid, i.e. by assuming there is a valuation which makes its premisses true and negated conclusion true too, and then working backwards, trying to uncover a consistent valuation which warrants that assumption.

In working backwards, we move at each stage from the assignment of truth to some complex wff to corresponding assignments of truth to some simpler subformula(s) – though we may have to consider alternatives. And when we have to consider alternatives, our reasoning is naturally set out in the form of a downward branching tree.

If working backwards, perhaps through alternatives, always leads to contradictions, then we know that the original assumption that the inference is invalid has to be incorrect. In other words, if all paths through the tree setting out our working can be closed off with absurdity signs, then the argument is *valid*.

Conversely, if at least one alternative path through the tree involves no contradiction (even we have unpacked all the implications of assignments of truth to complex wffs on the path), then the assumption that the argument is *invalid* is vindicated. That's because, as we will check, a valuation which makes the simple wffs on this open path true will make the original premisses and negated conclusion true too.

## 18.4 Unsigned trees

(a) The 'working backwards' arguments which we have been illustrating have so far been regimented as *signed* trees (we will speak of 'trees' here even when, as in examples **B** and **C**, we actually have only a bare trunk without any branching). The trees are 'signed' in the sense that the wffs that appear on the trees are all *explicitly* assigned a truth value – though, in our version of signed trees, the wffs are all assigned T.

But now, in the interests of neatness and economy of effort, we are going to move to using so-called *unsigned* truth trees. *We will simply delete '⇒ T' from every expression that might occur on one of our signed trees*. We have seen a similar trick before. In §9.2 we moved from construction trees decorated with expressions of the form *'α' is a wff* to construction trees which have a plain wff $\alpha$ at each location (at each 'node', to use the standard term). Now we are moving from truth trees decorated by expressions of the form $\alpha \Rightarrow T$ to trees which have a plain wff $\alpha$ at each node.

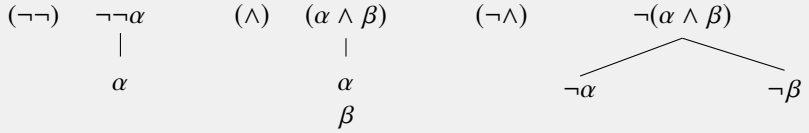
(b) Note, we *could* have initially introduced a (rather more natural?) version of working-backwards trees where some wffs are assigned T and others are assigned F. So, for example, we start the tree for an argument with its premisses assigned T and conclusion assigned F, and then we proceed from there (with rules for unpacking the implications of wffs assigned T and more rules for unpacking wffs assigned F). But you can now see the pay-off from using T-only signed trees from the start. The move from our T-only trees to unsigned trees is as trivial as trivial can be.

(c) The principles that we have informally be using for unpacking the implications of complex PL/PLC wffs assigned T on a signed tree will therefore now become rules for dealing with 'naked' wffs on an unsigned tree. We have in effect already met all the needed rules in their signed form. In the rest of this chapter, however, we will (re)present the rules for constructing unsigned trees in a more systematic order.

Sticking still with the 'unpacking' metaphor, let's call the wff that we are unpacking at any stage the *parent* wff, and call the new wffs we add when we unpack its implications its *children*. As before, we require our unpacking rules to ensure the following:

> *Parents-to-children* When we apply a non-branching rule, if the parent wff is true then every added child is true too. When we apply a branching rule, if the parent wff is true then, on at least one branch of every added fork, the added child is true.

Let's start then with the rules for unpacking complex wffs of one of the following three forms: $\neg\neg\alpha$, $(\alpha \wedge \beta)$, $\neg(\alpha \wedge \beta)$. We can vividly display the required rules in diagrammatic form:



How do we read these diagrams? The rule (¬¬) tells us that given a particular wff of the form $\neg\neg\alpha$ on the tree (now invisibly assigned T), then we unpack its implications by adding $\alpha$ (invisibly assigned T) to the tree. Add where? We add a child $\alpha$ at the foot of every open path containing the parent wff we are unpacking.

The rule (∧) tells us that given a parent wff of the form $(\alpha \wedge \beta)$, then we unpack its implications by adding both $\alpha$ and $\beta$ as children to the foot of every open path containing the parent wff.

The rule (¬∧) tells us that given a parent wff of the form $\neg(\alpha \wedge \beta)$, then we unpack its implications by adding a fork with a branch to $\neg\alpha$ and a branch to $\neg\beta$ to the foot of every open path containing the parent wff.

Each rule obviously satisfies Parents-to-children.

(d) Let's have a couple of examples of these three rules in play. First, consider the argument

**H** $((P \wedge Q) \wedge R) \therefore (Q \wedge (R \wedge P))$

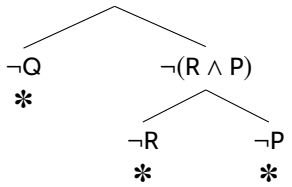
To show this is valid using an unsigned tree, we start its trunk with the argument's premiss (quietly assumed true on some valuation) plus its negated conclusion (quietly assumed true on that same valuation). So we write simply

| | | |
|---|---|---|
| (1) | $((P \wedge Q) \wedge R)$ | (premiss) |
| (2) | $\neg(Q \wedge (R \wedge P))$ | (negated conclusion) |

We now follow the tactic of keeping our tree tidy by applying one of the non-branching rules before the branching rule when we have a choice. So we continue by adding more wffs which must also be true (on the same supposed valuation):

| | | |
|---|---|---|
| (3) | $(P \wedge Q)$ | (from 1 by ∧) |
| (4) | $R$ | (from 1 by ∧) |
| (5) | $P$ | (from 3 by ∧) |
| (6) | $Q$ | (from 3 by ∧) |

Next, we apply our branching rule twice to extract the implications of (2), but again suppressing the explicit assignments of the truth value T:

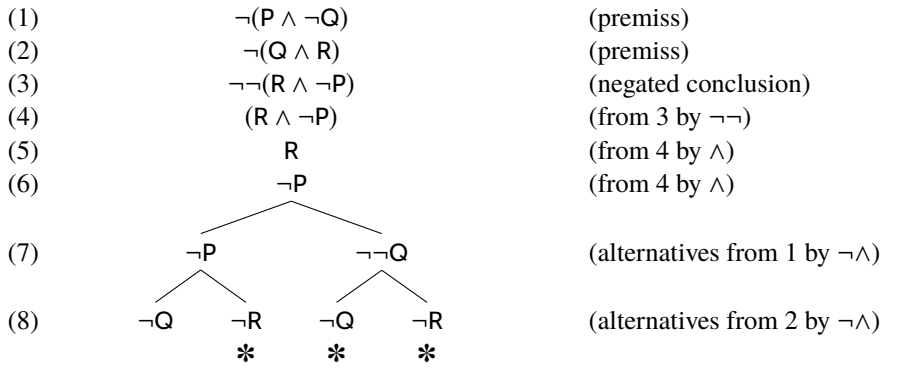



Each path leads to contradiction, since it contains a wff and its negation both invisibly assigned T. Hence the assumption that **H**'s premiss and negated conclusion are both true leads to absurdity. Hence **H** is valid.

Next, consider the argument

**I** $\neg(P \wedge \neg Q), \neg(Q \wedge R) \therefore \neg(R \wedge \neg P)$

You know how a tree will start. And here's one finished version:




We have again applied non-branching rules first, starting with (¬¬) to get line (4), and then applying (∧) to get the next two lines. Then we applied our branching rule (¬∧) to extract the implications of (1) and (2). (See what happens if we unpack (1), (2) and (3) in different orders: explore which order leads to the neatest tree, and why.)

This tree is now finished in the sense that no complex wff on the open path remains to be dealt with. We have already drawn out the implications of each of (1) to (4), so there can be no lurking contradictions still to be exposed. We are left with an open path containing the simple wffs '¬P', '¬Q', and 'R'. What we can learn from this?

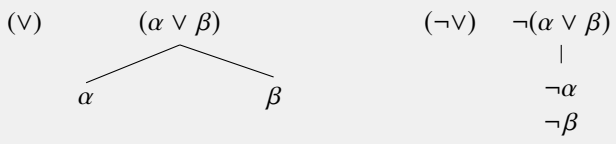
Well, note that our three rules not only satisfy the Parents-to-children principle, but also a kind of converse:

> *Children-to-parents* For each of our rules, branching or non-branching, if a parent's children on some path through the tree are true, so too is the parent wff.

Therefore, as we go up some path on a tree, if children we encounter are true, so too are their parents (and their parents' parents, etc.). In the present case, if we take the valuation making the simple wffs '¬P', '¬Q', and 'R' on the open path all true, then this will make the parents of these wffs true, and those parents will make *their* parents true, thereby making (1), (2) and (3) all true. Which establishes that **I** is invalid.

## 18.5 More rules for unsigned trees

(a) Next – in summary display form – here are the two rules we need for unpacking disjunctions, both naked and negated:

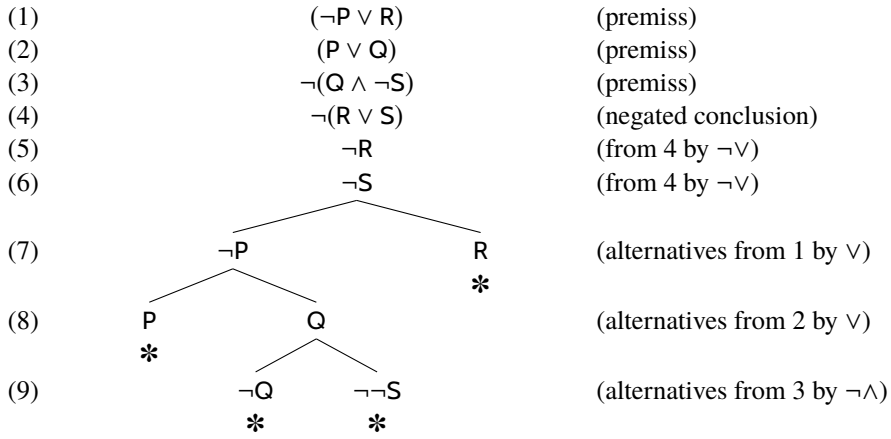



The branching rule (∨) is of course to be interpreted like the branching rule (¬∧); and the non-branching rule (¬∨) is to be interpreted like the non-branching rule (∧). And our rules obviously continue to satisfy Parents-to-children and Children-to-parents.

We can now apply these disjunction rules to construct an unsigned truth tree to test the following for validity:

**J** $(\neg P \vee R),\ (P \vee Q),\ \neg(Q \wedge \neg S) \therefore (R \vee S)$

And here is one possible result (if we follow the usual tactic of applying non-branching rules first when we can, to keep the tree tidy):

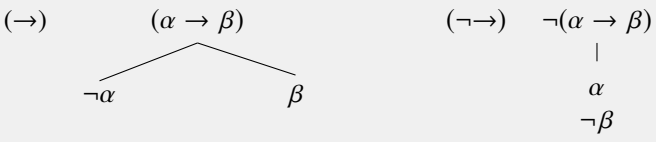



Need we spell things out again? We have assumed that the premisses and negated conclusion of **J** are all true, and tried to worked backwards to uncover a valuation of simple wffs which makes this the case. But we hit contradiction every way we turn. So there is no such valuation, and hence the argument **J** is valid.

(b) We have continued to put line numbers on the left of our trees and put some minimal commentary on the right. But these are optional extras. Strip them away, and the uncommented tree which shows argument **J** is valid is very quick and neat.

We have in fact seen this very same argument before in §14.4(b). It was there labelled **E** and we ran a truth-table test on it. So we are now in a position to compare the amount of work involved in the two tests for tautological validity applied to this same argument. And we see that in the time it takes to write down the frame of the truth-table test (writing the premisses and conclusion across the top, and the catalogue of sixteen possible valuations of the atoms down the left side), i.e. before we have done *any* work filling in the table by evaluating premisses and conclusions, our tree test for validity will be finished. That's not at all untypical.

(c) If we are working in a PLC language, we will need two additional rules for dealing with material conditionals, both naked and negated:



We interpret these diagrams for rules just like the previous cases, and note that the two Parent/Children principles hold again.

(d) Let's work through one more example which exploits this last pair of rules, but now speeding up by dropping the running commentary on the right. The inference up for evaluation is this:

**K** $((P \to Q) \to (R \to \neg S), (\neg R \to (Q \wedge P')), (\neg P \to P') \therefore (S \to P')$

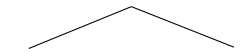
As usual, we are looking to see if there is a 'bad' valuation which makes the premisses all true and makes the negation of the conclusion too. So we suppose we have the following (all invisibly assigned T on this supposed bad valuation):

(1) $((P \to Q) \to (R \to S)$
(2) $(\neg R \to (Q \wedge P'))$
(3) $(\neg P \to P')$
(4) $\neg(S \to P')$

Applying the tactic of applying non-branching rules first where possible, we start by using the tree-building rule $(\neg\to)$ on (4):

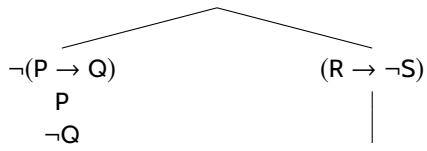
(5) $S$
(6) $\neg P'$

Where shall we go from here? Let's deal with (3) first; for a moment's thought tells us that, although applying the $(\to)$ rule leads to a fork in the tree, one resulting branch will immediately close, like this:



(7) $\neg\neg P$ $P'$
(8) $P$ ✻

We look next at the implications of (1), and then apply our non-branching rule $(\neg\to)$:



(9) $\neg(P \to Q)$ $(R \to \neg S)$
(10) $P$
(11) $\neg Q$

Which leaves just the wff at (2) and the wff on the right-hand branch at (9) to unpack. Let's deal with (2) first. We need to add to the results of applying the rule $(\to)$ to both open paths on the tree. So, we add

(12) $\neg\neg R$ $(Q \wedge P')$ $\neg\neg R$ $(Q \wedge P')$

We now have four open paths through the tree, and on each path there remain complex wffs still to be unpacked. Take first, though, the left-most path. We can continue it thus:

(13) $R$

And now we see that there are no more complex wffs *on the left-most path* to be unpacked, so there can be no lurking contradiction to be found, at least on *that* path. If we make the simple wffs on that path all true – by putting $\mathsf{P} \Rightarrow \mathrm{T}$, $\mathsf{Q} \Rightarrow \mathrm{F}$, $\mathsf{R} \Rightarrow \mathrm{T}$, $\mathsf{S} \Rightarrow \mathrm{T}$, $\mathsf{P}' \Rightarrow \mathrm{F}$ – that will make the more complex wffs on that branch true too (by repeated applications of the Children-to-Parents principle). In particular, that valuation makes (1) to (4) at the top of the path all true, showing that argument **K** is invalid.

If – as is usually the case – all we care about is a verdict on validity, we can now stop work on the tree. Recall that finding one bad line on a truth table is enough to show the argument being tested in invalid. Exactly similarly here: finding one open branch on which every complex wff has been unpacked is enough to show that the argument being tested is invalid (we needn't complete the other branches unless we want more information about different ways of making the premisses true and conclusion false).

(e) We now have seven unpacking rules for dealing with complex PLC wffs on a tree. Do we need any more? No. Any complex wff – i.e. any wff other than an atom or negated atom – falls into one of two classes. Either (i) it has one of the three binary connectives as its main connective. Or else (ii) it begins with a negation sign and then the main connective of the rest of the wff is either a binary connective or negation again. So (i) covers three kinds of wff, and (ii) covers another four. We need seven rules for dealing with complex wffs of these seven kinds. Which is exactly what we now have.

## 18.6 'Checking off'

(a) We should emphasize an important point. Suppose that we have an open path on a truth tree starting with the premisses and negation of the conclusion of an argument. Then we can read off a valuation showing the argument is invalid *only if the path is completed*, i.e. only if no complex wff on the branch still requires its implications to be unpacked.

After all, if a complex wff on a path remains to be examined, its implications could – for all we yet know – generate a contradiction (and then there would be no consistent valuation which makes every wff on the path true). Hence, before we can conclude anything about an argument from a currently open path on its tree, we do have to check that every complex wff on the path really has been taken count of.

To make this check easier, it is very useful to have a device for signalling quite explicitly when a complex wff has been unpacked. The standard convention is to *check off* a wff at the point when we applying an unpacking rule to it.

Suppose we have already applied the appropriate rule, adding children to every open path containing the parent wff we are unpacking. Then it would be pointlessly redundant to apply the rule to the very same wff again. It would be worse than redundant to get 'stuck in a loop', repeatedly re-applying the same rule to the same wff, time without end. Henceforth, then, we adopt the following rule:

> When the appropriate unpacking rule has been applied to a complex wff, we *check off* the wff (with a '✓'). A wff, once checked off, is then treated as 'used up'; it is then no longer available to have one of the unpacking rules (re)applied to it.

(b) For a small example of checkmarks in use, consider again

**G** $((P \wedge Q) \vee R) \therefore \neg(\neg P \vee \neg R)$

which we earlier tested for validity with a signed tree. We will now construct an unsigned tree, checking off complex wffs as we go. We start the tree with

$$((P \wedge Q) \vee R)$$
$$\neg\neg(\neg P \vee \neg R)$$

Applying the unpacking rule ($\neg\neg$), this becomes

$$((P \wedge Q) \vee R)$$
$$\neg\neg(\neg P \vee \neg R) \checkmark$$
$$(\neg P \vee \neg R)$$

And then applying the unpacking rule ($\vee$) to the first wff we get

$$((P \wedge Q) \vee R) \checkmark$$
$$\neg\neg(\neg P \vee \neg R) \checkmark$$
$$(\neg P \vee \neg R)$$
$$(P \wedge Q) \qquad R$$

Now using the unpacking rule ($\wedge$) on the left branch, the tree becomes

$$((P \wedge Q) \vee R) \checkmark$$
$$\neg\neg(\neg P \vee \neg R) \checkmark$$
$$(\neg P \vee \neg R)$$
$$(P \wedge Q) \checkmark \qquad R$$
$$P$$
$$Q$$

In this example – even if we weren't checking wffs off as we unpack their implications – we would hardly be likely to overlook the fact that we haven't yet dealt with the third wff! But the use of check marks makes this unmissable. So we need to continue:

$$((P \wedge Q) \vee R) \checkmark$$
$$\neg\neg(\neg P \vee \neg R) \checkmark$$
$$(\neg P \vee \neg R) \checkmark$$
$$(P \wedge Q) \checkmark \qquad R$$
$$P \qquad\qquad \neg P \qquad \neg R$$
$$Q \qquad\qquad\qquad\quad *$$
$$\neg P \qquad \neg R$$
$$*$$

And *now* we are done. There are two open paths through the tree on which every complex wff is checked off. Therefore, as we already know, **G** is invalid.

## 18.7 Trees for tautologies

(a) A tautology, we remarked in §15.1(c), can be thought of as the conclusion of a tautologically valid argument with no premisses. This immediately tells us how to use a truth tree to determine whether a wff is a tautology.

How do we test whether the inference $\alpha_1, \alpha_2, \ldots, \alpha_n \therefore \gamma$ is tautologically valid using a truth tree? We put all the $\alpha$s and the negated conclusion $\neg\gamma$ at the top of a truth tree, and then use the tree-building rules. The argument is tautologically valid if and only if every path through the tree closes.

Similarly, how do we test whether the wff $\gamma$ is a tautology using a truth tree? We put its negation $\neg\gamma$ at the top of a truth tree, and proceed to use the tree-building rules. Then $\gamma$ is a tautology if and only if every path through the tree closes.

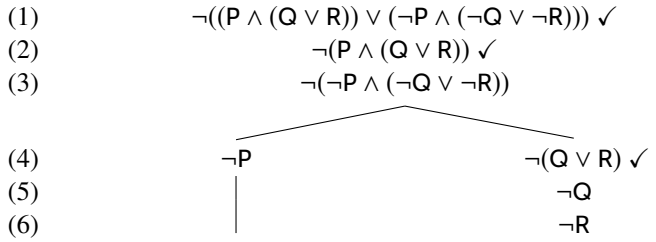
(b) For an example, take the wff

**L** $((P \wedge (Q \vee R)) \vee (\neg P \wedge (\neg Q \vee \neg R)))$

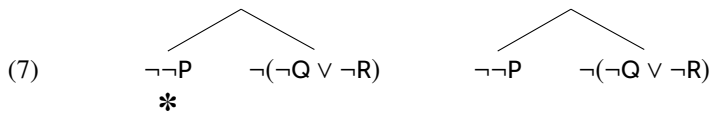
Is this a tautology? To apply the tree test, we start with the wff's negation (quietly assigned the value T):

(1) $\neg((P \wedge (Q \vee R)) \vee (\neg P \wedge (\neg Q \vee \neg R)))$

We then work backwards to determine whether this assumption in fact leads to contradiction. By bracket-counting, we see that our wff has the form $\neg(\alpha \vee \beta)$, where $\alpha$ is '$(P \wedge (Q \vee R))$ ' and $\beta$ is '$(\neg P \wedge (\neg Q \vee \neg R))$'. So we apply the non-branching rule $(\neg\vee)$ to (1) to get two derived wffs $\neg\alpha$ and $\neg\beta$ at lines (2) and (3). We then continue in the obvious way by unpacking (2) to get the following:




At this stage, the only complex wff to unpack is (3); so let's apply the rule $(\neg\wedge)$, adding the results to both open branches:




Which leaves us with three branches which so far remain open. Tackling the middle one first (hardly difficult!) we can add

(8) P

Remembering that (3) has indeed been checked off, there are no remaining complex wffs on this path, still waiting to be unpacked. Hence there can be no lurking contradictions

on this path – even if we continue work elsewhere on the tree, this path must stay open. Consider, then, the valuation which makes $\mathsf{P} \Rightarrow \mathrm{T}$, $\neg\mathsf{Q} \Rightarrow \mathrm{T}$ and $\neg\mathsf{R} \Rightarrow \mathrm{T}$; this will make the more complex wffs further up this path true, and hence make (1) true. Since (1) is the negation of **L**, this means that our valuation makes **L** false. Hence **L** is not a tautology.

(c) One last example in this chapter. Consider the string of symbols

**M** $\quad (\neg(\neg(\mathsf{P} \wedge \mathsf{Q}) \wedge \neg(\mathsf{P} \wedge \mathsf{R})) \vee \neg(\mathsf{P} \wedge (\mathsf{Q} \vee \mathsf{R})))$

Is this string even a wff? Well, that can be confirmed by the trick of changing the shape of some matching pairs of brackets, thus:

$$[\neg\{\neg(\mathsf{P} \wedge \mathsf{Q}) \wedge \neg(\mathsf{P} \wedge \mathsf{R})\} \vee \neg\{\mathsf{P} \wedge (\mathsf{Q} \vee \mathsf{R})\}]$$

Then we see this is a wff of the form $[\alpha \vee \beta]$ with $\alpha$ the wff '$\neg\{\neg(\mathsf{P} \wedge \mathsf{Q}) \wedge \neg(\mathsf{P} \wedge \mathsf{R})\}$', and $\beta$ the wff '$\neg\{\mathsf{P} \wedge (\mathsf{Q} \vee \mathsf{R})\}$'. So is it a tautology?

We start our tree test with the negation of **M**, which will therefore be of the form $\neg[\alpha \vee \beta]$ with the same $\alpha$ and $\beta$. So we can immediately apply the rule $(\neg\vee)$, to get

(1) $\neg[\neg\{\neg(\mathsf{P} \wedge \mathsf{Q}) \wedge \neg(\mathsf{P} \wedge \mathsf{R})\} \vee \neg\{\mathsf{P} \wedge (\mathsf{Q} \vee \mathsf{R})\}]$ ✓
(2) $\neg\neg\{\neg(\mathsf{P} \wedge \mathsf{Q}) \wedge \neg(\mathsf{P} \wedge \mathsf{R})\}$
(3) $\neg\neg\{\mathsf{P} \wedge (\mathsf{Q} \vee \mathsf{R})\}$

Continuing in the obvious way, by applying the rule $(\neg\neg)$ to both (2) and (3), and then applying the rule $(\wedge)$ twice we get

(2) $\neg\neg\{\neg(\mathsf{P} \wedge \mathsf{Q}) \wedge \neg(\mathsf{P} \wedge \mathsf{R})\}$ ✓
(3) $\neg\neg\{\mathsf{P} \wedge (\mathsf{Q} \vee \mathsf{R})\}$ ✓
(4) $\{\neg(\mathsf{P} \wedge \mathsf{Q}) \wedge \neg(\mathsf{P} \wedge \mathsf{R})\}$ ✓
(5) $\{\mathsf{P} \wedge (\mathsf{Q} \vee \mathsf{R})\}$ ✓
(6) $\neg(\mathsf{P} \wedge \mathsf{Q})$
(7) $\neg(\mathsf{P} \wedge \mathsf{R})$
(8) $\mathsf{P}$
(9) $(\mathsf{Q} \vee \mathsf{R})$

As this reminds us, one lovely feature of working with trees is that, as we apply the unpacking rules, the wffs that we add to the tree get simpler and simpler. We are now just left with the last four very much more tractable wffs to deal with (as everything earlier has been checked off). It is a very simple matter to confirm that the tree now quickly closes. Unpacking (6), (7) and (9) in turn gives us:

(11) $\neg\mathsf{P}$ ✻ $\quad \neg\mathsf{Q}$
(12) $\neg\mathsf{P}$ ✻ $\quad \neg\mathsf{R}$
(13) $\mathsf{Q}$ ✻ $\quad \mathsf{R}$ ✻

Hence the assumption that we can make the negation of **M** true leads to contradiction; **M** is indeed a tautology.

## 18.8 Two sorts of trees – a reality check

Let's finish this chapter by emphasizing a key difference between the construction trees we met before and our truth trees in this chapter

We can put it this way. The (upward) branchings in construction trees are *conjunctive* – reading upwards, an expression before a branch point is a wff if and only if *both* expressions after the branch point are wffs. The (downward) branchings on truth trees are *disjunctive* – reading downwards, a wff before a branch point is true if and only if *at least one* of the wffs after the branch point are true.

Which way up we draw a tree is, of course, a trivial matter of style (as we noted, linguists like their parse trees the other way up). What matters is whether the tree reflects a way of conjoining simpler claims into more complex ones or a way of exploring disjunctive options. Truth trees are of course of the second kind.

## 18.9 Summary

Testing an argument for tautological validity is a matter of searching for 'bad' valuations which make the premisses true and the conclusion false. We can perform such a search by a brute-force inspection of all possible valuations, as in a truth table. Or we can 'work backwards', i.e. assume we can make the premiss and negated conclusion all true, and try to determine what the assumed bad valuation would look like.

The working backwards method, in the general case, will necessitate considering alternative scenarios, e.g. when we have to look at alternative ways of making a disjunction true. The resulting exploration is naturally laid out in the form of a downward-branching tree.

By very simple tricks, we can ensure that every wff on such a tree is assigned the value T. But then, if every wff is assigned the same value, we need not bother to explicitly write down the value – resulting in our preferred *unsigned* trees.

We have rather informally explored the rules for constructing an unsigned tree using 'unpacking' rules taking us from more complex wffs on the tree (quietly assumed true) to simpler wffs (also true on the same imagined valuation).

An argument is tautologically valid if and only if every path through a properly constructed tree starting with the premisses and negated conclusion ends in contradiction.

We also noted that we can similarly use a tree to test a wff to determine whether it is a tautology. A wff is a tautology valid if and only if every path through a properly constructed tree starting with its negation ends in contradiction.

## Exercises 18

(To be added)

# 19 Truth trees further explored

The last chapter introduced truth trees in a relatively discursive way. This chapter starts by telling the same story again more briskly to fix ideas (and to provide a resource for future revision). We then explore just a little further the theory and practice of using truth trees to test arguments for tautological validity.

(We will not keep repeating the point made in §18.7, that trees can also be used to test wffs for being tautologies. We can leave it as an easy exercise to make any needed little tweaks to the general story about truth trees to cover that special application.)

## 19.1 Reminders about trees and our terminology

As is now familiar, truth trees will be downward-branching *binary* trees, meaning that a branch never forks more than two ways. We can take the general idea of a downward-branching binary tree to be intuitively clear (we could get mathematically precise here but nothing would be gained, given our elementary purposes).

However, we should perhaps gather together some relevant terminology, old and new. First, for binary trees in general:

(1) Positions on a tree are conventionally called *nodes*.

(2) Going down a binary tree, each node is followed by zero, one, or two *successor nodes*. A node is usually thought of as being joined to its successors by *edges*; but, for visual economy, in the case of truth trees, we have only drawn edges from a node to its successors when there is more than one successor – in other words, when the tree branches. (Compare our construction trees in §9.2 where we depicted all the edges.)

(3) The nodes from the topmost of the tree down to the first branch point, the first node with two successors, form the *trunk* of the tree.

(4) Start from the topmost node of the tree, and track down some route along edges through the tree to an *end-node* without successors, i.e. to the bottom tip of some branch. The nodes en route form a *path* through the tree.

(5) On a tree (as opposed to other kinds of 'graphs' made of nodes and edges) the paths downwards from the top node fan out without ever rejoining at a lower level: as in the case of a respectable family tree, incestuous reunions are banned. In an obvious sense, a tree is *loop-free*; this entails that there is only one path from the top node down to a given end-node.

Now for some initial terminology specific to our truth trees:

(6) On a truth tree, the nodes are occupied, labelled, or decorated by wffs (either from a PL or from a PLC language). If a wff $\alpha$ occupies a node on a path, we will say more briefly that the path *contains* $\alpha$, or that $\alpha$ is *on* the path.

(7) If a path contains, for some wff $\alpha$, both that wff and its negation $\neg\alpha$, then the path is *closed*. Otherwise it is *open*.

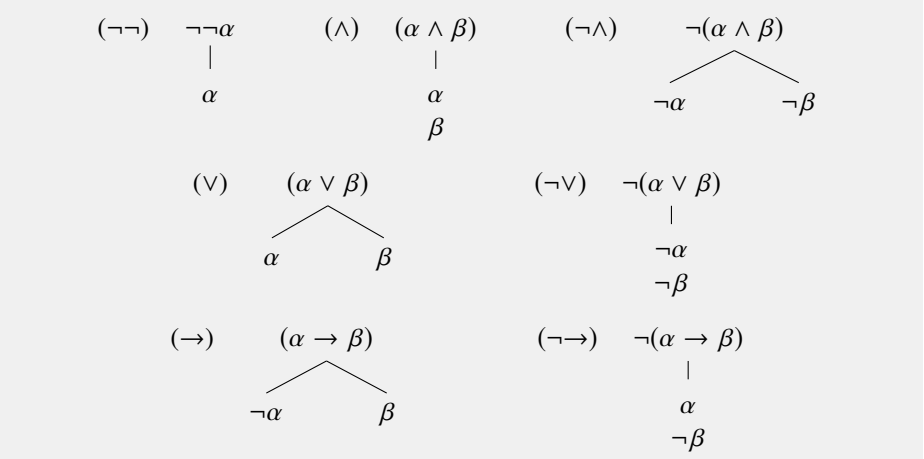
And recall,

(8) A wff is said to be *simple* if it is either an atomic wff or the negation of an atom. Otherwise it is *complex*.

## 19.2 Testing arguments using truth trees, in summary

We now summarize in one place the key ideas we introduced in the last chapter.

(a) Suppose, then, that we want to determine whether a given PL or PLC inference is tautologically valid. We begin a truth tree to test this inference by placing its premisses and its negated conclusion at nodes forming the initial trunk of a tree (we can imagine each of them accompanied by an invisible '$\Rightarrow$ T'). We then aim to systematically 'unpack' the implications of those initial assignments – and then the implications of these implications, etc. – exploring alternatives when necessary. As before, we call the wff that we are unpacking at any stage the *parent* wff, and call the new wffs we add when we unpack its implications its *children*. And we do the needed 'unpacking' of complex wffs using the following now-familiar rules (the last two being needed only if we are working with PLC wffs).




How do we interpret these diagrammatically presented rules? To repeat, the non-branching rule (¬¬) tells us that, given a parent wff of the form $\neg\neg\alpha$ (invisibly assigned T) on a tree, we unpack its implications by adding $\alpha$ (invisibly assigned T) as as a child to the foot of every open path containing that particular parent wff.

The non-branching rule ($\wedge$) tells us that, given a parent wff of the form ($\alpha \wedge \beta$), we unpack its implications by adding both $\alpha$ and $\beta$ to the foot of every open path containing that parent wff. Similarly for the other non-branching rules, ($\neg\vee$) and ($\neg\rightarrow$).

The branching rule ($\vee$) tells us that, given a parent wff of the form ($\alpha \vee \beta$), we unpack its implications by adding a binary fork with a branch to $\alpha$ and a branch to $\beta$ to the foot of every open path containing that parent wff. Similarly for the other branching rules, ($\neg\wedge$) and ($\rightarrow$).

And how do we deploy these unpacking rules to construct an unsigned truth tree to test an inference for tautological validity? Here, in summary form, is the procedure we have in effect being using all along (now officially adopting the policy of stopping work on a tree if and when we find an open path on which we have unpacked every complex wff).

To construct a tree for testing whether the inference $\alpha_1, \alpha_2, \ldots, \alpha_n, \therefore \gamma$ is tautologically valid, proceed as follows:

(P1) Start the trunk of the tree by placing $\alpha_1, \alpha_2, \ldots, \alpha_n, \neg\gamma$ down the top nodes.

(P2) Inspect every path through the tree which doesn't yet finish with an absurdity sign to see whether it involves a contradiction, i.e. to see whether it contains both $\beta$ and also the corresponding formula $\neg\beta$ (for some wff $\beta$). If it does, then we close that path by adding the absurdity sign '✻'.

(P3) If every path through the tree is closed with an absurdity sigh, then stop! Or if there is an open path on which every complex wff has been checked off, then also stop!

(P4) Otherwise, we choose some unchecked complex formula $\beta$ that occurs on an open path. We then apply the appropriate unpacking rule and add the results to the foot of every open path containing that wff $\beta$. We then check off $\beta$ (using a checkmark '✓').

(P5) Loop back to step (P2).

The 'checking off' rule ensures that the appropriate unpacking rule can only be applied once to any complex wff on the tree. Further, the new wffs added when we apply a rule are always shorter than the unpacked wff they result from. It follows that our unpacking-and-tree-extending procedure must eventually terminate as we unpack and check off shorter and shorter wffs. (We are assuming, as always, that an inference has only a finite number of premisses to start with!) Call any tree which results at the end of this process a *test tree* for the given inference under test.

(b) Two more bits of terminology applying to truth trees:

(1) A tree is *closed* if and only if every path through the tree is closed. Otherwise it is stays *open*.

(2) A path through a tree is *completed open* if and only it is open and every complex wff on that path has been duly unpacked and checked off.

It is easily seen that, on any particular run of our tree building procedure, the test tree we end up with is either closed, or else has at least one completed open path. And what do we learn from these alternative results? To finish our headline summary of the tree test, here is how we extract verdicts from a test tree for a given inference:

> Given a test tree for the inference $\alpha_1, \alpha_2, \ldots, \alpha_n, \therefore \gamma$, then it delivers verdicts as follows:
>
> (V1) If the tree is closed, the tested inference is tautologically valid.
>
> (V2) If the tree has a completed open path, the inference is tautologically invalid.

The claims (V1) and (V2) should look correct, given all our explanations in the previous chapter. Still, we should officially prove them to be true, and we do that in §19.5.

(c) Note that, as you would expect from our informal presentation in the last chapter, our procedural rules allow us a free choice of which unchecked complex wff to unpack next each time we circle round to step (P4) – at least until we get to deal with the last unchecked complex wff. So there is usually no unique test tree. However, we do have the following key result:

> Starting from the same initial wffs $\alpha_1, \alpha_2, \ldots, \alpha_n, \neg\gamma$ on the trunk of the tree, either any resulting test tree is closed, or alternatively any tree stays open. We can not get some closed trees and some with completed open paths.

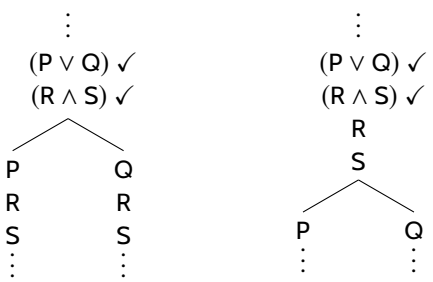
Why so? Because, given claims (V1) and (V2), if we could get both a closed tree and a (different!) tree with a completed open branch when testing a given argument, the argument would be both valid and invalid, which is impossible.

This confirms that it can't matter in which order we apply our tree-building rules. Test trees for a given inference always agree on their verdicts.

## 19.3 Trees in practice

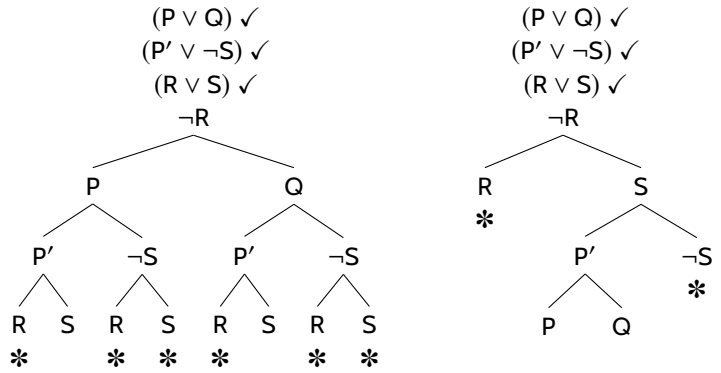
So much for a summary review of the theory. Now for three comments on using trees in practice. Again there is nothing new here. We are just gathering together points we touched on in passing in the last chapter.

(a) Compare first the following two fragments of truth trees:



The only difference between these two (parts of) trees is the order in which we have unpacked the pair of complex wffs: we end up, either way, with two paths with just the same wffs on them. But the second tree where we apply the non-branching rule first obviously avoids some duplication across branches and so needs a bit less time and ink to write down. That's a miniature illustration of our first maxim for getting neater trees: *generally, apply non-branching unpacking rules before branching rules, when we have the choice*.

(b) Compare next the following two completed truth trees. This time, the three complex wffs at the top of the tree all need to be unpacked using branching rules. But it still makes a difference which order we do the unpacking:




On the left, we unpack the initial wffs in the order they appear on the trunk of the tree; and on the right, we unpack them in the reverse order. We end up in each case with two open paths, with the same simple wffs on them. But obviously the right-hand tree is a lot neater, because we close off branches sooner. For instance, by choosing to unpack $(R \vee S)$ first, we immediately close off one alternative at the first fork, and don't have to explore it any further.

So that illustrates a second practical maxim: *wherever possible, choose to apply rules in an order which closes off branches as early as possible*.

(c) As we noted in §18.5(d), once we have found *one* completed open path on a tree for an argument, that is already enough to show that the argument in question is invalid (our proof of (V2) at the end of the chapter will confirm this). We have built this fact into our described official procedure for tree-building, but let's highlight it again: *assuming that all we care about is a yes/no verdict on validity, stop work on a tree once we have found one completed open path*.

(d) Let's work through a couple more annotated examples, keeping our practical maxims in mind. First, we will use a tree to test the following inference for validity:

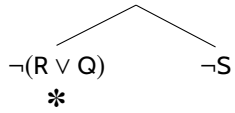
**A** $(P \wedge (R \vee Q))$, $((P \wedge \neg Q) \vee (\neg(P' \vee R) \vee S))$, $((P' \wedge S) \vee \neg Q)$
$\therefore\ ((R \vee Q) \wedge S)$

So we start in the familiar way with the premisses and negated conclusion at the top of the tree. We note that only one of these complex wffs calls for a non-branching rule to

be applied, so – following our first maxim – let's start by unpacking that one:

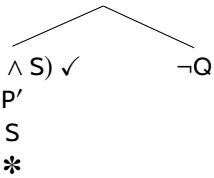
| | | |
|---|---|---|
| (1) | $(P \wedge (R \vee Q))$ ✓ | (premiss) |
| (2) | $((P \wedge \neg Q) \vee (\neg(P' \vee R) \vee S))$ | (premiss) |
| (3) | $((P' \wedge S) \vee \neg Q)$ | (premiss) |
| (4) | $\neg((R \vee Q) \wedge S)$ | (negated conclusion) |
| (5) | $P$ | (from 1 by $\wedge$) |
| (6) | $(R \vee Q)$ | (from 1 by $\wedge$) |

Now note that (6) and (4) each contains an occurrence of '$(R \vee Q)$', naked in the case of (6) and to-be-negated-on-one-fork when we unpack (4). So – following our second bit of tactical advice – let's continue the tree



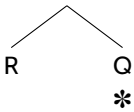
(7) $\neg(R \vee Q)$ $\neg S$ (alternatives from 4 by $\vee$)

With (4) now checked off, we have a choice of three complex wffs to unpack next, those at lines (2), (3) and (6). And although the last is the simplest, our second practical maxim suggests dealing with (3) first, because it quickly leads to another closed branch:



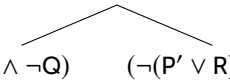
| | | | |
|---|---|---|---|
| (8) | $(P' \wedge S)$ ✓ | $\neg Q$ | (alternatives from 3 by $\vee$) |
| (9) | $P'$ | | (from 8 by $\wedge$) |
| (10) | $S$ | | (from 8 by $\wedge$) |

The sensible thing to do next, obviously enough, is *now* to unpack (6) which again leads a branch immediately closing off:



(11) $R$ $Q$ (alternatives from 6 by $\vee$)

If you are keeping track, then you will see that on our open path every complex wff has now been checked off except (2), so let's now unpack that at last:



(12) $(P \wedge \neg Q)$ $(\neg(P' \vee R) \vee S))$ (alternatives from 2 by $\vee$)

And now the end is sight! Let's work on the left-hand open path first. Every complex wff on that path has now been unpacked except the last one, '$(P \wedge \neg Q)$'. So let's deal with that. Check off (12) and add:

| | | |
|---|---|---|
| (13) | $P$ | (alternatives from 12 by $\wedge$) |
| (14) | $\neg Q$ | (alternatives from 12 by $\wedge$) |

There are no more complex wffs on *this* path to be unpacked; and there is no contradictory pair of wffs to be found along it either. It is a completed open path. So no more work is required: that's enough to show that **A** is indeed invalid. We are done.

(e) For a final example, consider the following inference:

**B** $(P \to (\neg Q \to R)),\ (P' \to (\neg Q' \to R'))$
$\therefore\ ((P \wedge P') \to ((Q \wedge Q') \to (R \wedge R'))$

A little reflection should tell you that this ought to be valid (why?). There are six atoms in play, and you are very welcome to write down a sixty-four line truth-table to confirm that this inference is indeed valid. But a truth tree is a *lot* quicker if we remember our practical maxims. Begin the tree with the premisses and negated conclusion and, in accord with the first maxim, keep applying any possible non-branching rule and we get

| | | |
|---|---|---|
| (1) | $(P \to (\neg Q \to R))$ | (premiss) |
| (2) | $(P' \to (\neg Q' \to R'))$ | (premiss) |
| (3) | $\neg((P \wedge P') \to ((Q \wedge Q') \to (R \wedge R'))$ ✓ | (negated conclusion) |
| (4) | $(P \wedge P')$ ✓ | (from 3 by $\neg\to$) |
| (5) | $\neg((Q \wedge Q') \to (R \wedge R'))$ ✓ | (from 3 by $\neg\to$) |
| (6) | $P$ | (from 4 by $\wedge$) |
| (7) | $P'$ | (from 4 by $\wedge$) |
| (8) | $(Q \wedge Q')$ ✓ | (from 5 by $\neg\to$) |
| (9) | $\neg(R \wedge R')$ | (from 5 by $\neg\to$) |
| (10) | $Q$ | (from 8 by $\wedge$) |
| (11) | $Q'$ | (from 8 by $\wedge$) |

That leaves (1), (2) and (9) as complex wffs still to check off. If we unpack (1) or (2) the tree forks with one branch immediately closing; not so if we unpack (9). So let's sensibly follow our second maxim and unpack (1) and (2) first. Then our tree neatly concludes:

| | | |
|---|---|---|
| (12) | $\neg P$ ✻ $\quad (Q \to R)$ ✓ | (alternatives from 1 by $\to$) |
| (13) | $\neg Q$ ✻ $\quad R$ | (alternatives from 12 by $\to$) |
| (14) | $\neg P'$ ✻ $\quad (Q' \to R')$ ✓ | (alternatives from 2 by $\to$) |
| (15) | $\neg Q'$ ✻ $\quad R'$ | (alternatives from 14 by $\to$) |
| (16) | $\neg R$ ✻ $\quad \neg R'$ ✻ | (alternatives from 9 by $\neg\wedge$) |

So yes, **B** is valid, and can be shown to be so in a lot less than 64 lines of working.

It is worth pausing to think, though, what would have happened if we had ignored our maxims and started by fully unpacking (1) and then (2) using our branching rules first. Fully unpacking (1) splits the tree, and then splits it again (as in lines (12) and (13) above). So we get three open paths. Then fully unpacking (2) adds another three-way split at the foot of those paths, so we get nine open paths, before we go on to complete the tree. Which just reinforces the point that, if we are not careful, a tree for a given argument can sprawl horribly even there's another very tidy tree which gives the same result.

## 19.4 Comparative efficiency?

Taking up that last point, we have repeatedly seen that using a tree test on an argument can often be significantly faster than plodding through the corresponding truth table test. Sometimes the gain in speed is extraordinary: remember, for example, Example **A** in §18.2.

Yet, as also just remarked, things can become pretty messy too. The work needed on a truth *table* explodes exponentially with the number of atoms in play in the argument; but the work needed on a truth *tree* can *also* grow exponentially as the wffs in an argument get more numerous. Imagine dealing with a lot of unnegated disjunctions using the unpacking rule ($\vee$). The tree will keep forking. If things go badly, we get two branches, then four, then eight, then sixteen, then .... The work needed to complete the tree can explode again.

We have seen that adding rules of thumb – our practical maxims – can in good cases cut the work right down. But do they always tame trees and keep the needed amount of work under control? Or, perhaps casting our net beyond truth tables and truth trees, is there *any* mechanical test for tautological validity which doesn't – in the worst cases – lead to exponential explosion as the argument being tested gets more involved? No. Or at least, so it is very widely believed. But nobody really knows. The issue relates to a still unsolved deep problem in the foundations of the theory of computation, the so-called P vs NP problem. There is a million dollar prize on offer for the answer to *that*!

## 19.5 Trees *do* work as promised!

As we went along in the last chapter, we motivated our rules (V1) and (V2) for giving verdicts about validity. And those earlier informal explanations might well already be enough to convince you.

But strictly speaking we ought to confirm more carefully that trees really do give the right verdicts! So let's now give more careful official arguments for (V1) and (V2). The proofs are slightly abstract (and can certainly be skipped, which is why we have left them to the end of the chapter); but they are not difficult and indeed are pleasingly neat.

(a) First some more reminders and a new bit of terminology:

(1) Each of our unpacking rules ensures that *Parents-to-children* holds. In other words, when we apply a non-branching rule, if the parent wff is true so too are all its children. And when we apply a branching rule and have to consider forking alternatives, if the parent wff is true, so too is at least one of its children from each fork we add.

(2) Each of our unpacking rules ensures that *Children-to-parents* holds. In other words, for each of our rules, branching or non-branching, if the children of a parent wff on a path are true, so too is the parent wff.

(3) We will call a path through a tree *satisfiable* if there exists a valuation (of the relevant atoms that appear in wffs on the path) which makes all the wffs on the path true. So the wffs on a satisfiable path are consistent.

(b) *Proving (V1)* We need to show that a closed test tree for an argument does imply validity. We appeal to a simple observation: (S) if we extend a satisfiable path on a tree using one of our unpacking rules, then at least one of the resulting longer paths is still satisfiable. Why does (S) hold?

> Take a satisfiable path on a tree. By assumption, there exists a valuation which makes every wff on the path true, including the parent wff we are now unpacking.
>
> If we are applying a non-branching rule, this valuation which makes the parent wff true must also make true the newly added children of that wff (by Parents-to-children). So the extended path stays satisfiable.
>
> If we are applying a branching rule, the path now splits into two. But on at least one of the two extended paths, the newly added child of the unpacked parent wff must be true too, even if we don't know which path that is (by Parents-to-children again). So at least one extended path still has all true wffs on that valuation, i.e. it stays satisfiable.

Now to apply this result (S). Suppose the argument $\alpha_1, \alpha_2, \ldots, \alpha_n \therefore \gamma$ is tautologically *invalid*. Start constructing the tree for this argument. We put $\alpha_1, \alpha_2, \ldots, \alpha_n, \neg\gamma$ on its initial trunk. Since the argument is invalid by hypothesis, there is a valuation which makes those initial wffs all true: in other words, the initial trunk of the tree is satisfiable. And then (S) tells that, however far we extend the tree using our unpacking rules, at each stage there will always remain at least one satisfiable path through it.

But a satisfiable path through a tree cannot close. For if all the wffs on the path can be true together, they are consistent and cannot include a contradictory pair of wffs, $\beta$ and $\neg\beta$.

In short, our tree for the invalid argument $\alpha_1, \alpha_2, \ldots, \alpha_n \therefore \gamma$ can't close. Therefore, contraposing, (V1) if the tree for an argument *does* close, the argument under test is valid.

(c) *Proving (V2)* We need to show that a completed open path through a test tree for an argument implies invalidity. We again need a simple observation: (O) a completed open path on a tree is always satisfiable. Why? We have already seen the basic idea more than once, but let's spell it out again:

> Suppose a tree has a completed open path. Since the path is completed, every complex wff on the open path has been unpacked, until we get down to simple wffs, i.e. atoms and negated atoms. Since the path is open, no atom appears on the path both naked and negated. We can therefore read off a consistent valuation of atoms which makes the simple wffs on the path all true. (If there is any other atom which appears in a complex wff on the path, chose whichever value for it you like.) Call this resulting valuation of atoms read off from the path the *path* valuation for short.
>
> But true children on a path always make their parent wffs true (by Children-to-parents). Hence, as we work up our open path, every wff we pass will be true on our path valuation. Why? Because the path valuation is designed to make the simplest wffs true; and then these make their parents true, and those make *their* parents true, and so on all the way up. And every wff on the completed path is either a complex parent wff with children further down, or is simple.

Hence there is a valuation – the path valuation – which makes every wff on the completed open path true.

To make that less abstract, consider our tree for example **A** in §19.3. We found a completed open path, containing the simple wffs 'P', '¬Q', 'R', '¬S'. So consider the valuation P ⇒ T, Q ⇒ F, R ⇒ T, S ⇒ F. P′ ⇒ T which makes the simple wffs all true, and assigns an arbitrary value to the atom 'P′' which appears on the tree. As we go up the open path, it is easily checked that this valuation makes every wff we encounter also true, including the wffs at the top of the trunk.

Now we apply result (O). Suppose a test tree for the argument $\alpha_1, \alpha_2, \ldots, \alpha_n \therefore \gamma$ has an completed open path. Then by (O) this path is satisfiable. And so there is valuation which, in particular, makes the initial wffs $\alpha_1, \alpha_2, \ldots, \alpha_n, \neg\gamma$ at the top of the path all true. So the argument is not tautologically valid. Which proves (V2).

(d) Finally, we should briefly introduce for the first time some important terminology that you will meet again in other contexts. In showing that (V1) is correct, we have just proved that tree-testing using our procedure is a *sound* method – if it tells you that an argument is valid (on the basis of finding a closed tree), you can rely on that verdict.

The method is also *complete* – meaning that it picks up every validity: if an argument is valid, our procedure will find you a tree that tells you it is valid. Or contraposing, if the test doesn't tells you the argument is valid (i.e. it tells you it is invalid on the basis of a completed open path), the argument is indeed not valid. That's what we proved in showing that (V2) is correct.

## 19.6 Summary

We reviewed the procedures for constructing a (non-unique) truth tree for a PL/PLC inference, and the rules for extracting a verdict on whether the inference is tautologically valid – if the tree closes, the argument is valid; if the tree stays open, the argument is invalid.

Some practical maxims for working with trees: apply non-branching rules first; look for branchings that close quickly; stop once one completed open path is found.

Trees are typically faster than truth tables. But as we deal with trees for more and more complex inferences, in the worst cases the amount of work needed on a tree can exponentially explode.

We proved that such verdicts are reliable – testing by truth trees gets questions of tautological validity right.

## Exercises 19

(To be added)