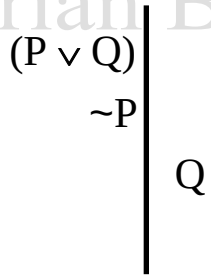
3.23. Truth Trees

The indirect test of validity, run in the vertical line notation, provides upside-down tree diagrams called **truth trees**.[1] Some examples illustrate truth tree mechanics.

We turn first to a familiar formal argument.

$$\begin{array}{l} (P \vee Q) \\ \quad \sim P \\ \hline \therefore\ Q \end{array}$$

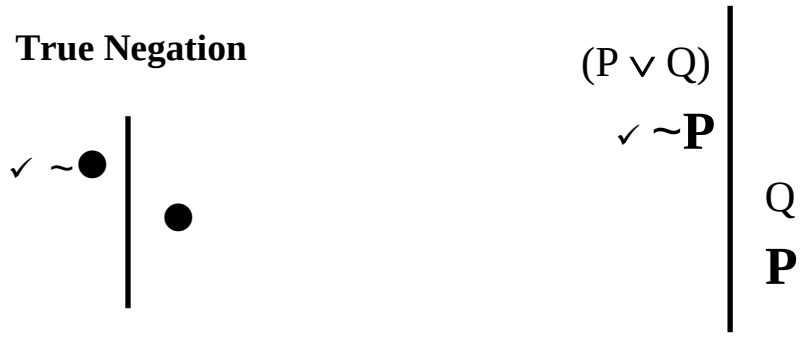
A truth tree test of validity always starts the same way: picturing a validity counterexample for the argument, by placing the premises on the left of the line (the *true* side) and conclusion on the right (the *false* side).



We then proceed to **break all molecular sentences down to atoms** (sentence letters), using the semantic rules.

[1] Some logic texts call them "semantic tableaux".

"~P" on the left is broken down with the **True Negation** rule: if "~P" is true, then "P" is false. (And as part of truth tree bookkeeping, we **check a sentence after breaking it down**.)



After breaking a sentence down, we pause to diagnose each open path (here, only one) by **tracing the path from bottom to top**. (As a visual aid: imagine the line is a pipe filled with water, and an air bubble rises in it, from bottom to top.)

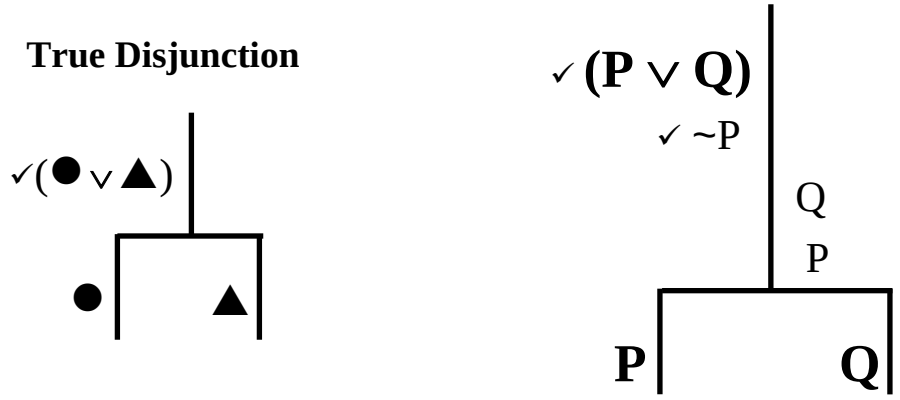
Following this path up, our diagnosis of it looks **only at sentence letters**; for if there is a violation of Bivalence, it will show up on the level of sentence letters.

Here we have "P" on the right (false) and "Q" on the right (false). Since each sentence has only one value, we have *no violation of Bivalence* so far.

P: 0
Q: 0

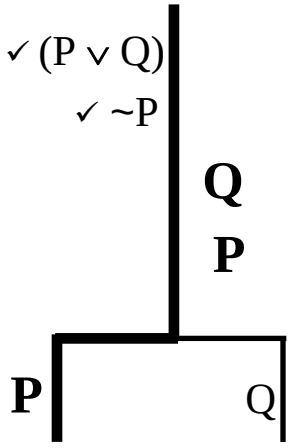
The only molecular sentence still unchecked (not yet broken down) is the first premise, "(P ∨ Q)".

“(P ∨ Q)” on the left follows the **True Disjunction** rule. Here the tree branches to cover the two distinct ways “(P ∨ Q)” might be true: because “P” is true, or because “Q” is true (or both).



Again we diagnose each path, following it from bottom to top like a bubble, and *looking only at sentence letters*.

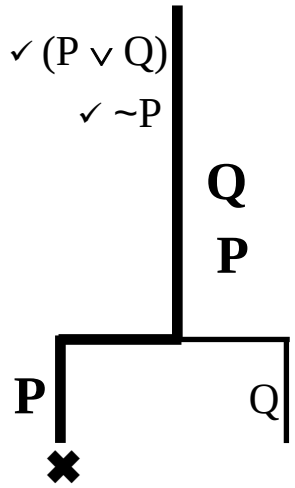
The left path has “P” on the left (*true*); “P” on the right (*false*); and “Q” on the right (*false*).



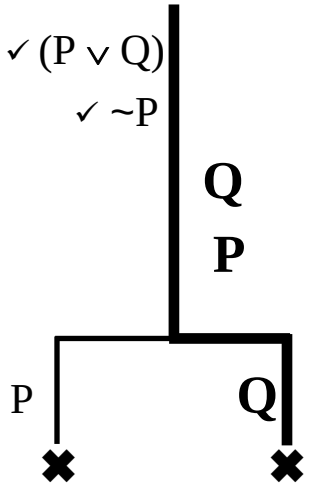
P: 0, 1
Q: 0

Since “P” is both true and false here, the path violates Bivalence. Now, a validity counterexample is a **possible** situation where premises are true and conclusion false. But by violating Bivalence, this path depicts an *impossible* situation. An impossible situation can’t qualify as a counterexample.

We mark the line as impossible by **closing** it, with an "✖".



Following from bottom to top, and looking only at sentence letters, the **right path** has "Q" on the left (*true*), "P" on the right (*false*), and "Q" on the right (*false*).



P: 1
Q: 0, 1

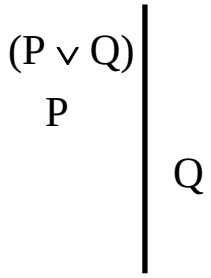
Making "Q" both true and false, this path violates Bivalence. So the path closes.

Though we sought a validity counterexample for this argument, we find that it's *impossible* for the premises to be true while the conclusion is false. This argument is **valid**.

By contrast, we know already from truth tables that the following formal argument is **invalid**.

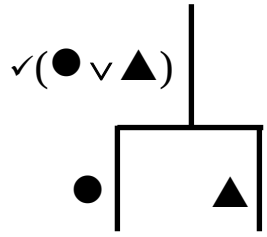
$$\begin{array}{l} (P \vee Q) \\ \quad P \\ \hline \therefore\ Q \end{array}$$

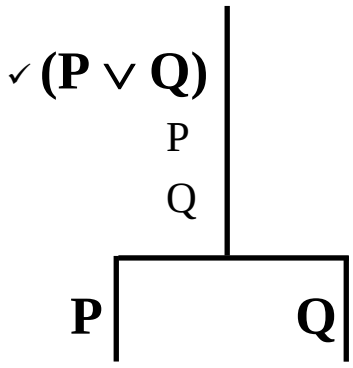
The truth tree test of validity begins as usual: premises on the *left*, conclusion on the *right*.



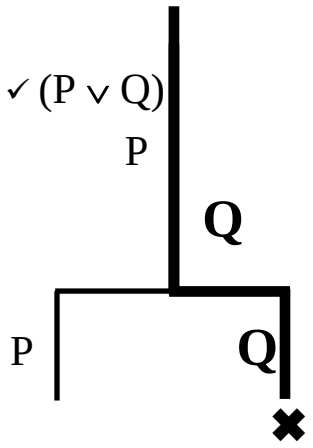
Here the only molecular sentence is the first premise, “(P ∨ Q),” which follows the True Disjunction Rule.

True Disjunction



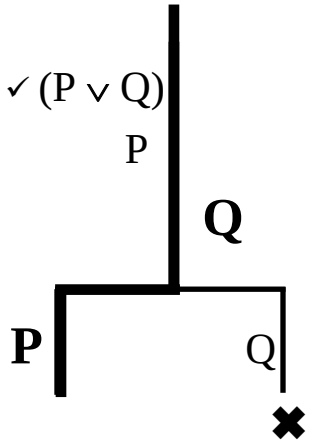


We diagnose each of the two open paths, bubble-style, from bottom to top. The right path makes "Q" both true and false, and closes.



P: 1
Q: 0, 1

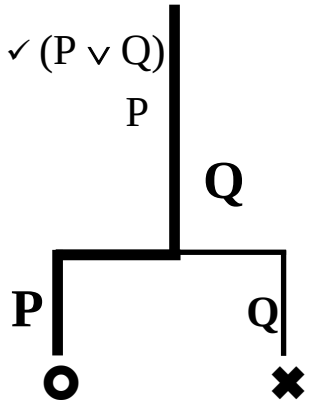
But the left path stays open, since Bivalence is obeyed: no sentence letter on the path has more than one value.



P: 1
Q: 0

And with all molecular sentences now checked, there are no further sentences yet to come that might close the path. This path will *never* close.

When a path will never close, we mark it with a “⭘”.



With all molecular sentences checked and every path diagnosed, the tree is complete. Since one path stayed open to the end, there is indeed a validity counterexample. This argument is **invalid**.

In closing, we note some further appealing features of truth trees. **First**, where there *is* a validity counterexample, the truth tree tells us exactly *which* valuation that is – just as truth tables do. With that last argument, for instance, the tree tells us that the validity counterexample is the case **where “P” is true, and “Q” is false**. That’s just what truth tables report.

P	Q	**(P ∨ Q)**	**∴ Q**
1	1	1	1
1	0	**1**	**0**
0	1	1	1
0	0	0	0

So these **details of the validity counterexample** are not lost when we trade truth tables for truth trees.

A **second** bit of information preserved by truth trees is the complete **list of sentences used** in the argument: the premises, conclusion, and all their parts, down to sentence letters. With truth tables that list can be read off the top.

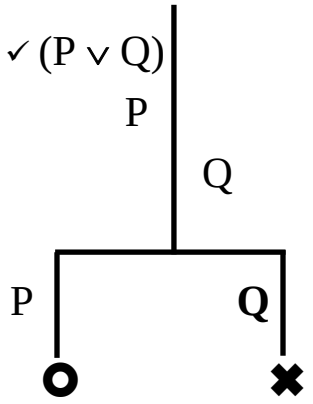
P	**Q**	**(P ∨ Q)**	∴ Q
1	1	1	1
1	0	1	0
0	1	1	1
0	0	0	0

Sentences Used:

P
Q
(P ∨ Q)

We know why truth tables work that way: they just mirror the steps of the *construction trees* for these sentences. Indeed, for complicated arguments our practice was to first build construction trees for premises and conclusion, and model the truth tables after them.

But truth trees also provide this list, since in the process of breaking sentences down to sentence letters, the truth tree semantically mirrors the construction steps for those sentences. So the same list of sentences appears in the truth tree for this argument (some sentences appearing more than once in the tree).



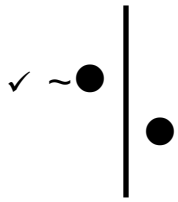
Sentences Used:

P
Q
(P ∨ Q)

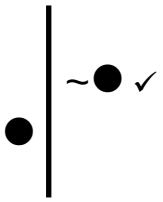
And truth trees offer this advantage: whereas we needed to first build a construction tree, as a guide to complicated truth tables, truth trees don't call for that preliminary step. That's one more *savings in labor*.

Truth Tree Rules

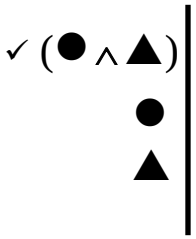
True Negation



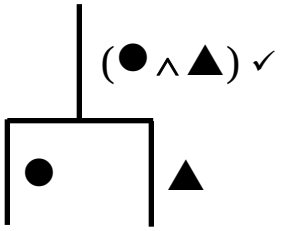
False Negation



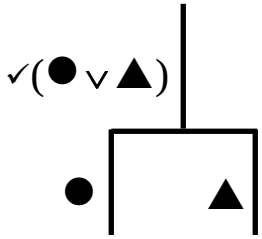
True Conjunction



False Conjunction



True Disjunction



False Disjunction