2.24. Truth Tree Restrictions

Truth trees combine two good results: (i) a general test of validity for arguments in our formal language, and (ii) a big savings in labor compared to the truth table test. Here we review two restrictions on truth trees needed to preserve those nice features. Without the first restriction, the truth tree test would fail as a generally reliable test of validity – yielding the wrong verdict on some arguments. Without the second restriction truth trees would needlessly multiply the work involved.

1. Multiple Open Lines. This simple English argument seems intuitively valid.

1. We're having ice cream or cake, but not both.
2. We're having ice cream

∴ We're not having cake.

And truth tables confirm its validity: the one valuation satisfying both the premises also satisfies the conclusion.

P: We're having ice cream
Q: We're having cake

1. $((P \vee Q) \wedge \sim(P \wedge Q))$
2. P

∴ ~Q

					(2)	**(1)**	**∴**
	P	Q	$(P \vee Q)$	$(P \wedge Q)$	$\sim(P \wedge Q)$	$\mathbf{((P \vee Q) \wedge \sim(P \wedge Q))}$	**~Q**
	1	1	1	1	0	0	0
⇨	**1**	0	1	0	1	**1**	**1** ⇦
	0	1	1	0	1	1	0
	0	0	0	0	1	0	1

A truth tree test of the argument’s validity begins with both premises on the left, and conclusion on the right.

$$\begin{array}{r|l} ((P \vee Q) \wedge \sim(P \wedge Q)) & \\ P & \\ & \sim Q \end{array}$$

“~Q” on the right follows the **False Negation** rule.

False Negation

$$\begin{array}{r|l} \bullet & \sim\bullet \checkmark \end{array}$$

$$\begin{array}{r|l} ((P \vee Q) \wedge \sim(P \wedge Q)) & \\ P & \\ & \mathbf{\sim Q} \checkmark \\ \mathbf{Q} & \end{array}$$

With both “P” and “Q” true, this line doesn’t yet close. It obeys Bivalence so far.

“((P ∨ Q) ∧ ~(P ∧ Q))” on the left is a large conjunction, following the **True Conjunction** rule: if the whole conjunction is true, both its parts are true.

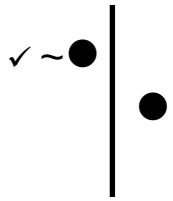
True Conjunction

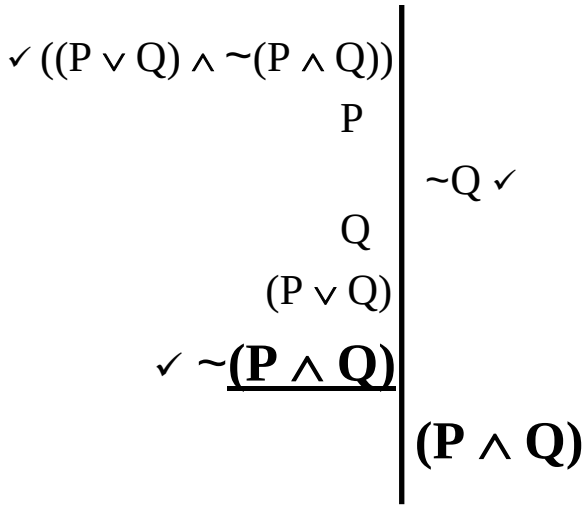
$$\begin{array}{r|l} \checkmark (\bullet \wedge \blacktriangle) & \\ \bullet & \\ \blacktriangle & \end{array}$$

$$\begin{array}{r|l} \checkmark \underline{\mathbf{(P \vee Q)}} \wedge \underline{\mathbf{\sim(P \wedge Q)}} & \\ P & \\ & \sim Q \checkmark \\ Q & \\ \mathbf{(P \vee Q)} & \\ \mathbf{\sim(P \wedge Q)} & \end{array}$$

"~(P ∧ Q)" on the left follows the **True Negation** rule: since "~(P ∧ Q)" is true, "(P ∧ Q)" is false.

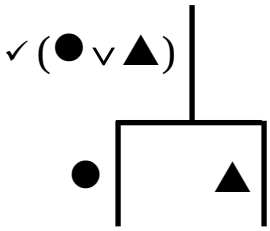
True Negation

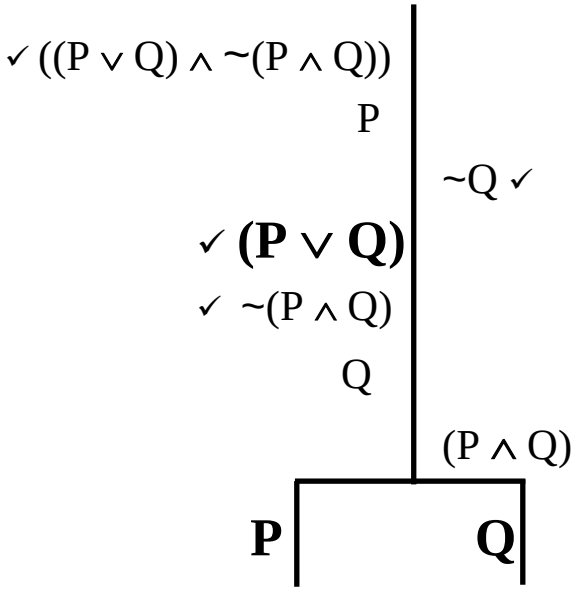




"(P ∨ Q)" on the left calls for the **True Disjunction** rule: if "(P ∨ Q)" is true, then either "P" or "Q" must be true.

True Disjunction

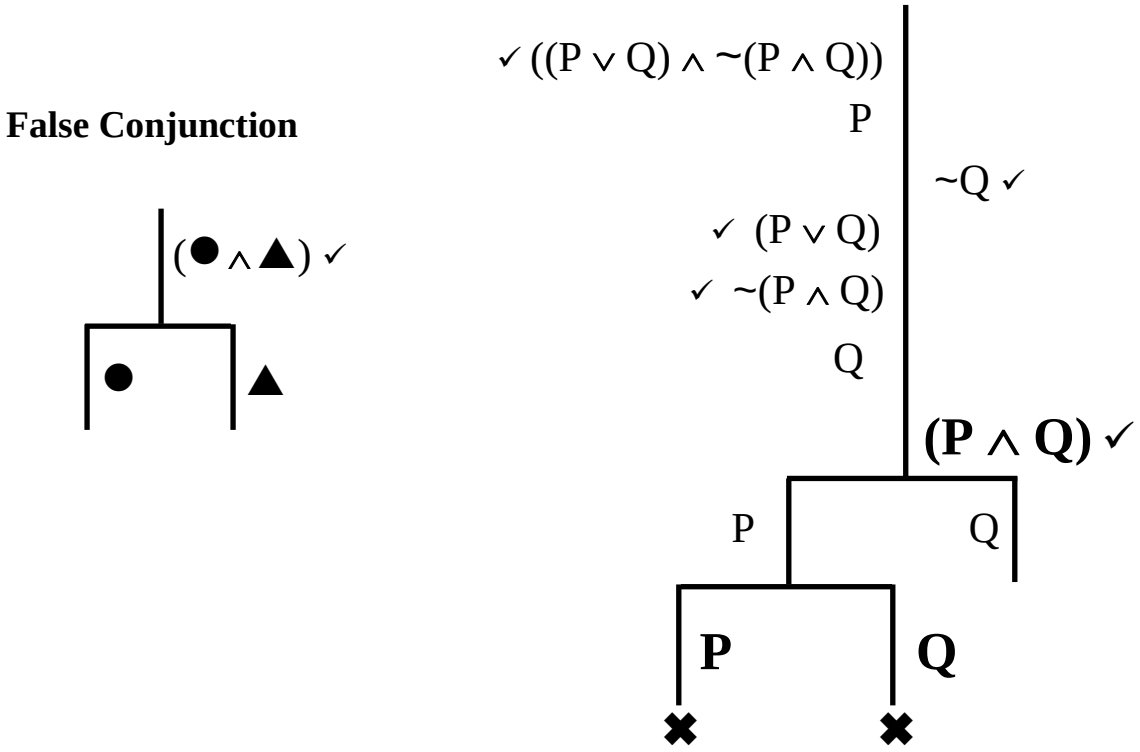




Both these paths stay open, since each obeys Bivalence. (Each path assigns "P" and "Q" only one value: true.)

Finally, "(P ∧ Q)" follows the **False Conjunction** rule: if "(P ∧ Q)" is false, then one or the other of its parts must be false. (*This is where the problem arises*.)

☠ A Suspicious Move ☠



Those last two paths each close: the left one because it makes "P" both true and false; the right one because it makes "Q" both true and false.

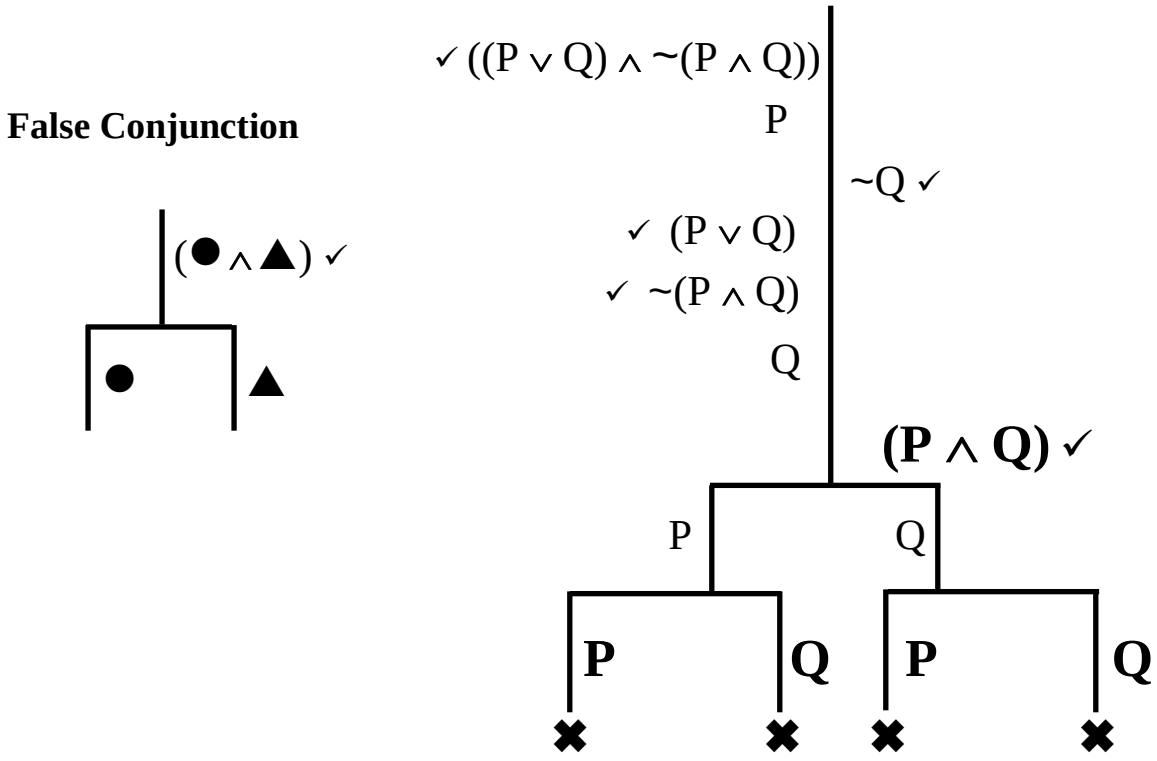
Now every molecular sentence is checked; but one of the paths remains open. The truth tree test of validity therefore judges the argument to be **invalid**.

That's a **problem** – because both our English language intuitions and the truth table test agree that this argument is perfectly **valid**. The truth tree test is giving the wrong result.

The trouble was in the last step. Note that when we added that last branch, following the False Conjunction rule, we only hung a copy of the branch off the end of the left path. That's why the right path stayed open – leading to a mistaken verdict of invalidity.

If we instead make *two* copies of the False Conjunction branch – hanging one off *each open path* below "(P ∧ Q)" – we get very different results.

False Conjunction Step, *Performed Properly*



Now all four paths close: two because they make "P" false and also true, two because they make "Q" false and also true. With every path closed, the tree correctly judges the argument **valid**.

Our mistake the first time was that we had **two** open paths, and a new branch (from the False Conjunction rule) – but we only made **one** copy of that branch. The correct procedure, we now see, is: when a sentence causes a branch, we **hang a copy of that branch off *every* open path below that sentence**.[1] Since there were two paths open below the branching sentence – False "(P ∧ Q)" – we had to hang a copy of the False Conjunction branch off **both** of those paths.

[1] To see why we add "below that sentence," see Problem 2.24.1 B on what happens if we leave this out of the rule.

2. Branching vs. Non-Branching Sentences: A Shortcut. Strictly speaking this requirement – to hang a copy off every open path – applies to any breaking-down of molecules, not just to branchy ones. So consider the following **valid** argument.

1. We're either having ice cream or cake.
2. We're not having cake.

∴ We're having ice cream.

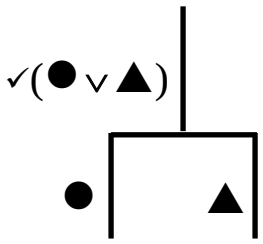
1. (P ∨ Q)
2. ~Q

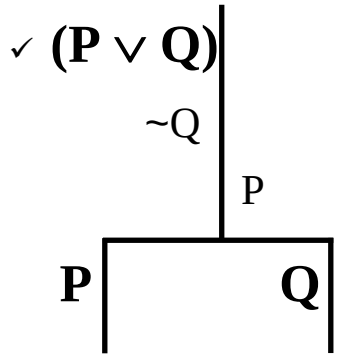
∴ P

			(1)	**(2)**	**∴**	
	P	Q	(P ∨ Q)	~Q	P	
	1	1	1	0	1	
⇨	1	0	**1**	**1**	**1**	⇦
	0	1	1	0	0	
	0	0	0	1	0	

Suppose in the truth tree test we start with the first premise,"(P ∨ Q)".

True Disjunction

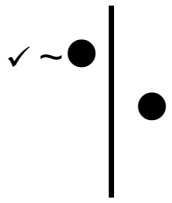


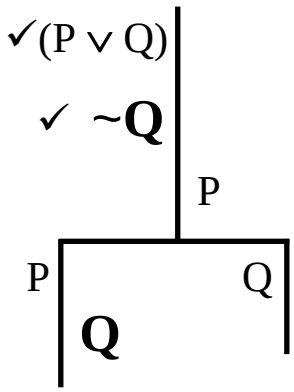


And suppose we (**incorrectly**) break down premise "~Q" **only** on the **left path**.

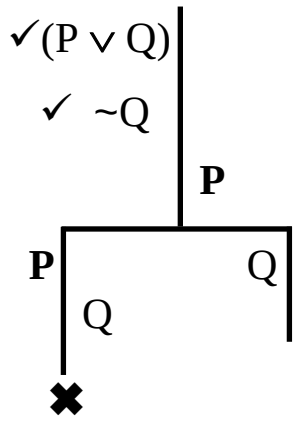
☠ True Negation (Done Incorrectly) ☠

True Negation





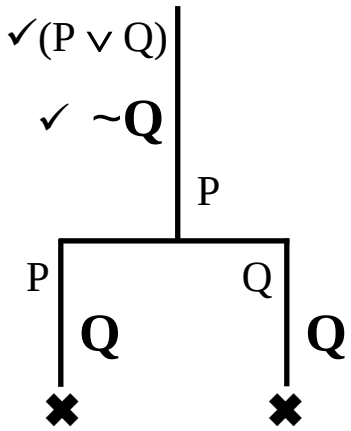
The left path closes (since "P" is both true and false), **but the right path stays open**.



With the right path open to the end, this tree wrongly judges the argument **invalid**.

We're familiar enough with truth trees now to spot the error here: with two paths open below it, True Negation "~Q" must be broken down on **both** those paths.

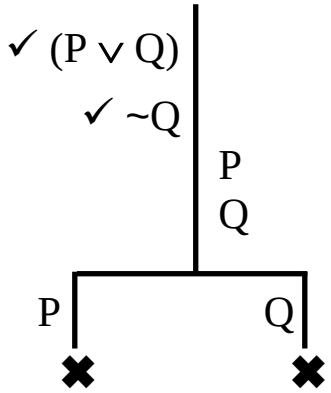
Putting a copy of False “Q” on **both** paths closes them both (the left path because it makes “P” both true and false, the right because it makes “Q” true and false). That’s the right result.



VALID!

But notice the **extra work** we made for ourselves here: because we broke down “~Q” only **after** we had two open paths, we needed to break it down **twice** (once on each path) to ensure the correct verdict.

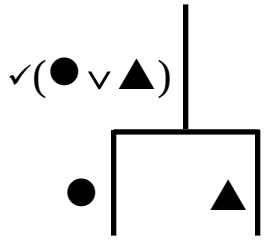
If instead we broke down “~Q” **before the branch**, we’d only need **one copy** of its part, False “Q” – while still (correctly) closing every path.



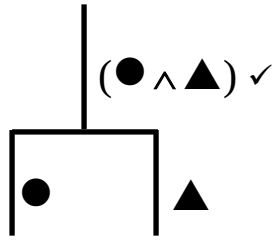
Since True “~Q” doesn’t branch, it saves us labor to break it down **before any branching sentences** – for that spares us making multiple copies of False “Q”.

Now, only two types of sentences introduce a **branch**: True Disjunctions and False Conjunctions.

True Disjunction



False Conjunction



To avoid unnecessary copying (and thereby reduce our workload) we will break down such **branching sentences** only **after** we've finished with the **non-branching sentences** (True Negations, False Negations, True Conjunctions, and False Disjunctions). Our earlier rule – "Multiple Copies for Multiple Open Paths" – then need only apply to the sentences remaining once we've taken care of all the non-branchers. And of course those will be just the branchers we'd saved for last.

So breaking down non-branchers first not only lightens our workload, but also permits a simplification of the "Multiple Copies" rule.

> When a sentence **branches**, hang a copy of the branch off the end of **every open path** below that sentence.

Summary: Testing Arguments for Validity Using Truth Trees

- Assume that there is a validity counterexample (a situation where the premises are all true, but the conclusion is false) by **putting all the premises on the left** (true) side of the tree trunk, and **the conclusion on the right** (false) side of the tree trunk.
- Break down **non-branching** sentences **before branching** sentences.
- **Check off each sentence** after that sentence is broken down.
- When a sentence branches, hang a **copy** of that branch off the end of **every open path** below that sentence.
- If a tree path has a **sentence letter on both** its **left and right sides**, **close** that path with an “✖”.
- When all the sentences have been broken down (so that only sentence letters remain unchecked), look at the end result:

  If **every path** has **closed**, the argument is **valid**.
  If **even one path is** still **open**, the argument is **invalid**.