

PY1003 Introduction to Logic
Lecture 5: Trees for Sentential Logic II

Last time we looked at five of the rules for constructing truth trees:

THE RULE FOR '∧', THE RULE FOR '∨', THE RULE FOR '¬¬', THE RULE FOR '→', and THE RULE FOR '↔'.

Consistency of a set of sentences:

A set Σ of sentences of Sentential Logic is consistent iff there is an interpretation (an assignment of truth-values to the atomic sentences) which makes them all true together. If there is no such interpretation, Σ is inconsistent.

$\{A, B \wedge C\}$ is a consistent set of sentences. If A is true and B is true and C is true, then all the sentences in the set $\{A, B \wedge C\}$ come out true. Thus the interpretation which makes all three atomic sentences true is a model for the set.

$\{A, \neg A\}$ is an inconsistent set. Any interpretation which makes A true makes $\neg A$ false. So there is no interpretation on which both members of this set are true.

A *model* of a set of sentences is an interpretation which makes them all true together. Consistent sets have a model; inconsistent sets do not.

Here is a model for the consistent set $\{A \wedge B, \neg C\}$:

$I(A) = T$
 $I(B) = T$
 $I(C) = F$

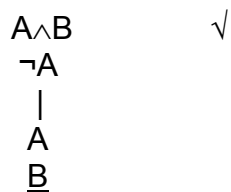
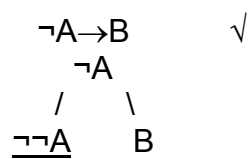
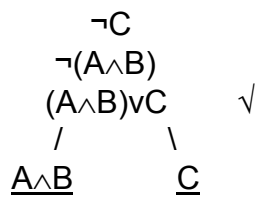
Remember that closed branches do not represent a way for the starting sentence(s) to be true, and that we display a branch closure with a horizontal line under the branch:

$$\begin{array}{l} A \wedge \neg A \quad \checkmark \\ | \\ A \\ \hline \neg A \end{array}$$

We do not write anything further on a branch once it has closed. A branch that has not closed is called an open branch.

Closures occur *when and only when* we have a sentence and its negation on the same branch. The sentence in question needn't be atomic. Close a branch as soon as it has a sentence and its negation on it. (But finish applying the rule you are currently applying!)

Examples:



If all branches have closed we have proved the starting sentence(s) inconsistent.

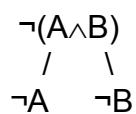
If there is no open branch then there is no way for the starting sentence(s) to be true.

If we have applied all possible rules and at least one branch has not closed then we have proved the starting sentence(s) consistent.

If there is at least one open branch then there is at least one way for the starting sentence(s) to be true.

Four New Tree Rules

THE RULE FOR NEGATED '∧'



THE RULE FOR NEGATED '∨'

$$\begin{array}{c} \neg(A \vee B) \\ | \\ \neg A \\ \neg B \end{array}$$

THE RULE FOR NEGATED '→'

$$\begin{array}{c} \neg(A \rightarrow B) \\ | \\ A \\ \neg B \end{array}$$

THE RULE FOR NEGATED '↔'

$$\begin{array}{cc} \neg(A \leftrightarrow B) & \\ / \quad \backslash & \\ A \quad \neg A & \\ \neg B \quad B & \end{array}$$

THERE IS NO RULE FOR AN ATOMIC SENTENCE OR FOR A NEGATED ATOMIC SENTENCE.

Examples:

1. Starting set: $\{\neg(A \wedge B), A \rightarrow B\}$

$$\begin{array}{cc} \neg(A \wedge B) & \checkmark \\ A \rightarrow B & \checkmark \\ / \quad \backslash & \\ \neg A \quad \neg B & \\ / \quad \backslash \quad / \quad \backslash & \\ \neg A \quad B \quad \neg A \quad \underline{B} & \end{array}$$

2. Starting set: $\{\neg(A \vee (B \vee C)), A \wedge B\}$

$$\begin{array}{l}
 \neg(A \vee (B \vee C)) \quad \checkmark \\
 A \wedge B \quad \checkmark \\
 | \\
 A \\
 B \\
 | \\
 \neg A \\
 \underline{\neg(B \vee C)}
 \end{array}$$

3. Starting set: $\{\neg(A \rightarrow B), \neg(A \vee \neg B)\}$

$$\begin{array}{l}
 \neg(A \rightarrow B) \quad \checkmark \\
 \neg(A \vee \neg B) \quad \checkmark \\
 | \\
 A \\
 \neg B \\
 | \\
 \neg A \\
 \underline{\neg \neg B}
 \end{array}$$

4. Starting set: $\{\neg(A \leftrightarrow B), \neg A \wedge \neg B\}$

$$\begin{array}{l}
 \neg(A \leftrightarrow B) \quad \checkmark \\
 \neg A \wedge \neg B \quad \checkmark \\
 | \\
 \neg A \\
 \neg B \\
 / \quad \backslash \\
 A \quad \neg A \\
 \underline{\neg B} \quad \underline{B}
 \end{array}$$

We sometimes justify a rule by thinking about logical equivalence. For example, the rule for negated ' \rightarrow ' is justified by noting that $\neg(A \rightarrow B)$ is logically equivalent to $A \wedge \neg B$. We therefore treat $\neg(A \rightarrow B)$ when it appears on a tree exactly as if it were $A \wedge \neg B$.

This might lead you to suspect that other facts about logical equivalence could be used in the same way, for example:

$(A \wedge A) \vee B$ is logically equivalent to $A \vee B$.

So when we see $(A \wedge A) \vee B$ on a tree, maybe we can just treat it as if it were $A \vee B$:

$$\begin{array}{c} (A \wedge A) \vee B \quad \checkmark \\ / \quad \backslash \\ A \quad B \end{array}$$

In fact we can't do this. Although it is a 'safe' mistake (won't lead you to mistake a valid argument for an invalid one, for example) it is still a mistake, because it is not a legitimate move according to the rules for the system we are using. (We'll look more at 'safe' and 'unsafe' mistakes later on.)

Although we use logical equivalence facts to justify our rules, that doesn't mean we can invent new rules whenever we discover logical equivalence facts.

The way to deal with $(A \wedge A) \vee B$ is:

$$\begin{array}{c} (A \wedge A) \vee B \quad \checkmark \\ / \quad \backslash \\ A \wedge A \quad \checkmark \quad B \\ | \\ A \\ A \end{array}$$

You might also think it would be safe to write something on a tree branch which is logically equivalent to something already on the branch, e.g.:

$$\begin{array}{c} \neg(A \rightarrow B) \quad \checkmark \\ | \\ A \wedge \neg B \end{array}$$

But again, this is not allowed by our rules. Even though it wouldn't make you go wrong in a serious way, because at the next stage you would simply write:

$$\begin{array}{c} \neg(A \rightarrow B) \quad \checkmark \\ | \\ A \wedge \neg B \quad \checkmark \\ | \\ A \\ \neg B \end{array}$$

which is exactly what you would have got had you performed the correct operation:

$$\begin{array}{c} \neg(A \rightarrow B) \quad \checkmark \\ | \\ A \\ \neg B \end{array}$$