Semantic Tableaux
for Propositional Logic

There are many techniques which may be used to automate logical deduction. Each has its advantages and disadvantages.

The first technique to be studied is that of semantic tableaux.

- It differs from the other techniques which we will study in that it does not generate a sequence of conclusions from a set of hypotheses.

- Rather, it conducts a direct search for models.

- Thus, it is termed a *semantic* technique.

Problem solving using propositional logic:

Example: The following word problem is taken from Example 2.4 of the textbook. We start by assigning symbols to the various assertions.

Assertion	Symbolic Representation
John will go to the party.	J
Joyce will go to the party.	Y
Clare will go to the party.	C
Stephen will go to the party.	S

Here is the argument in English, together with the logical interpretations.

- John or Joyce or both will go to the party.
  $(J \vee Y)$

- If Joyce goes to the party then Clare will go unless Stephen goes.
  $(Y \rightarrow (\neg S \rightarrow C))$

- Stephen will go if John does not go.
  $(\neg J \rightarrow S)$

- Therefore, Clare will go to the party.
  C

Note that the English is somewhat ambiguous. The second and third assertions could also be interpreted as

$(Y \rightarrow (\neg S \equiv C))$

$(\neg J \equiv S)$

This is always a problem when writing natural-language descriptions of formal problems, as natural language is often ambiguous.

The premises of this problem thus consist of three statements:

$\Phi = \{(J \vee Y), (Y \rightarrow (\neg S \rightarrow C)), (\neg J \rightarrow S)\}.$

This may be represented as a single statement by conjoining the formulas:

$\Phi_{\wedge} = (J \vee Y) \wedge (Y \rightarrow (\neg S \rightarrow C)) \wedge (\neg J \rightarrow S)$

The conclusion consists of a single statement:

$\varphi = C.$

It is desired to establish that

$\Phi \models \varphi.$

To do so, it suffices to show that

$\Phi_{\wedge} \wedge \neg\varphi$

is unsatisfiable.

- By definition,

  $\Phi \models \varphi$ holds iff $\mathrm{Mod}(\Phi) \subseteq \mathrm{Mod}(\varphi)$ does.

- This inclusion can hold iff

  $\mathrm{Mod}(\Phi) \cap \mathrm{Mod}(\neg\varphi) = \varnothing.$

  (This is just set theory – draw a Venn diagram.)

The naïve approach is to construct the truth table, and to see if the formula is true for any assignments.

$\Phi_{\wedge} \wedge (\neg\varphi) =$

$(J \vee Y) \wedge (Y \rightarrow (\neg S \rightarrow C)) \wedge (\neg J \rightarrow S) \wedge \neg C$

J	Y	S	C	$\Phi_{\wedge} \wedge (\neg\varphi)$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Thus, out of 16 possible worlds, four are models.

If we are only interested in whether or not the conclusion follows from the premises, then if this table is constructed row-by-row, we could stop at after generating the seventh.

This approach requires 2^n steps for n proposition letters, in the worst case.

However, it is easy to see that the search could be performed more intelligently. For example, since $\neg C$ is a conjunct of the formula, only interpretations for which C is false need be considered. This immediately cuts the number of cases to be tested in half.

The method of *semantic tableaux* performs this basic sort of search, but employs some selection heuristics which reduce the size of the search in many cases.

The ideas of this approach will now be developed.

Disjunctive Normal form:

Definitions:

- A *literal* is either a proposition name or its negation.
- The *complement* of a literal is its logical negation. Thus, the complement of A is $\neg A$, and the complement of $\neg A$ is A.
- A pair of literals $\{\ell_1, \ell_2\}$ is complementary if each literal is the complement of the other. Thus, a complementary pair is of the form $\{A, \neg A\}$.

Observation: Let $\varphi = \ell_1 \wedge \ell_2 \wedge .. \wedge \ell_n$ be a wff which is a conjunction of literals. Then φ is satisfiable iff it does not contain a complementary pair of literals. □

Definition: A wff φ is said to be in *disjunctive normal form* (DNF) if it is of the form $\varphi_1 \vee \varphi_2 \vee .. \vee \varphi_k$, with each φ_i a conjunction of literals.

Example: The formula

$$\varphi = (A_1 \wedge \neg A_2 \wedge A_3 \wedge \neg A_4) \vee (\neg A_1 \wedge \neg A_3 \wedge A_3 \wedge \neg A_4)$$

is in DNF.

Proposition. Let $\varphi = \varphi_1 \vee \varphi_2 \vee .. \vee \varphi_k$ be a wff in DNF. Then φ is satisfiable iff at least one of its disjuncts is satisfiable. □

Example: The formula

$$\varphi = (A_1 \wedge \neg A_2 \wedge A_3 \wedge \neg A_4) \vee (\neg A_1 \wedge \neg A_3 \wedge A_3 \wedge \neg A_4)$$

is satisfiable since the first disjunct is.

Example: The formula

$$\varphi = (A_1 \wedge \neg A_2 \wedge A_3 \wedge \neg A_4 \wedge A_2) \vee (\neg A_1 \wedge \neg A_3 \wedge A_3 \wedge \neg A_4).$$

is not satisfiable, since both of its conjuncts contain complementary pairs of literals.

Testing for satisfiability of wff's in DNF can be performed very efficiently: Suppose that the truth values of propositions are stored in a manner such that a lookup of a single value takes constant time. Then satisfiability for propositional formulas in DNF can be determined in time O(m), where m is the length of the formula (as a string). □

Unfortunately, the best known algorithm for converting an arbitrary wff to DNF has complexity $\Theta(2^m)$, so this method is not particularly efficient in general.

Conversion to DNF:

Clearly, not every wff is in DNF. However, every formula may be converted to one which is in DNF.

Example: $\varphi = \neg((A_1 \rightarrow A_2) \wedge (A_3 \vee \neg\ (A_3 \vee \neg A_1)))$
Here is a step by step conversion:

1. Original formula:
   $\neg((A_1 \rightarrow A_2) \wedge (A_3 \vee \neg\ (A_3 \vee \neg A_1)))$

2. Convert the implication to a disjunction:
   $\neg((\neg A_1 \vee A_2) \wedge (A_3 \vee \neg\ (A_3 \vee \neg A_1)))$

3. Apply de Morgan's identity to the whole formula.
   $\neg(\neg A_1 \vee A_2) \vee \neg(A_3 \vee \neg(A_3 \vee \neg A_1))$

4. Apply de Morgan's identity to each of the disjuncts.
   $(A_1 \wedge \neg A_2) \vee (\neg A_3 \wedge (A_3 \vee \neg A_1))$

5. Apply the distributive identity to the second disjunct.
   $(A_1 \wedge \neg A_2) \vee ((\neg A_3 \wedge A_3) \vee (\neg A_3 \wedge \neg A_1))$

6. Remove excess parentheses:
   $(A_1 \wedge \neg A_2) \vee (\neg A_3 \wedge A_3) \vee (\neg A_3 \wedge \neg A_1)$

This formula is thus satisfiable, since at least one disjunct (*e.g.,* the first) is.

- Note that we always drop double negatives ($\neg\neg\psi \equiv \psi$) without an explicit step.

The general algorithm for conversion to DNF:

1. Remove all occurrences of $\leftrightarrow$ (and $\equiv$) using the definition in terms of $\rightarrow$.

2. Remove all occurrences of $\rightarrow$ using the definition

$$(\varphi_1 \rightarrow \varphi_2) \equiv (\neg\varphi_1 \vee \varphi_2)$$

3. Use de Morgan's identities to move the negation signs in to the atoms. Eliminate double negations in the process.

4. Use the distributive identity to create a disjunction of conjunctions of literals.

Semantic Tableaux:

Stripped of all of its fluff, the method of semantic tableaux is basically one which tests an arbitrary wff for satisfiability by converting it to DNF. It adds some useful computational features:

- The expansion structure is represented using a tree, rather than a sequence of formulas.

- The expansion may be halted upon finding a satisfiable disjunct.

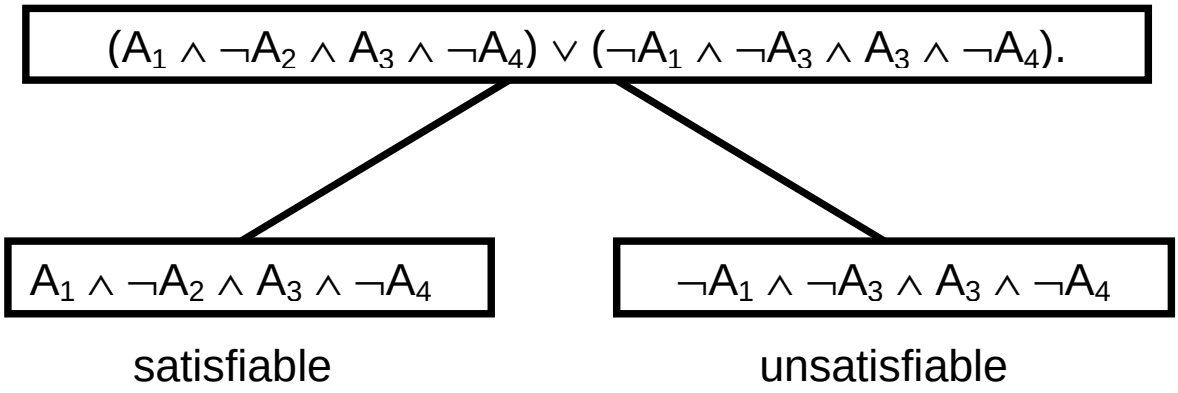
Regarding the second point, the expansion on the previous slide could have been stopped at step 4, since a satisfiable disjunct was found. Satisfiability of the other disjuncts became irrelevant. (It would not even have been necessary to simplify the second disjunct using de Morgan's identity.)

Here are some examples of trees for semantic tableaux.

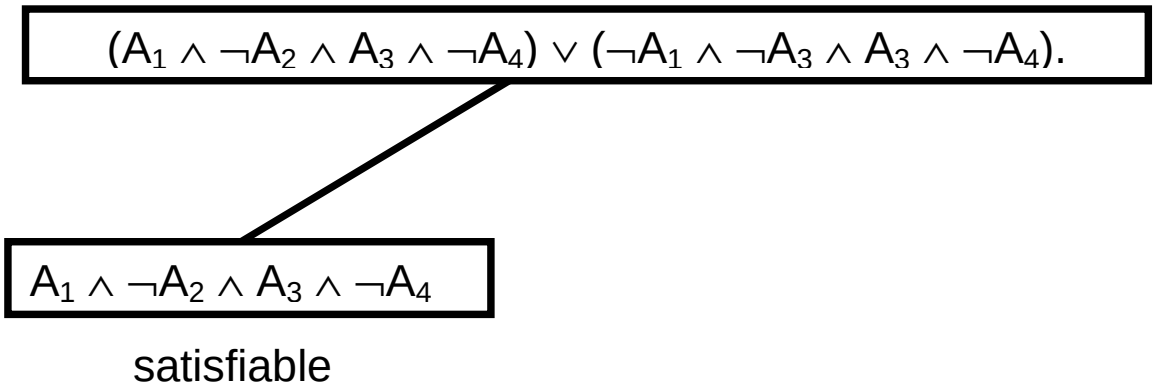
Example: Let

$$\varphi = (A_1 \wedge \neg A_2 \wedge A_3 \wedge \neg A_4) \vee (\neg A_1 \wedge \neg A_3 \wedge A_3 \wedge \neg A_4).$$

The root of the tree is the original formula. Each child of the root node is a disjunct. Since one of the branches is satisfiable, the whole formula is.



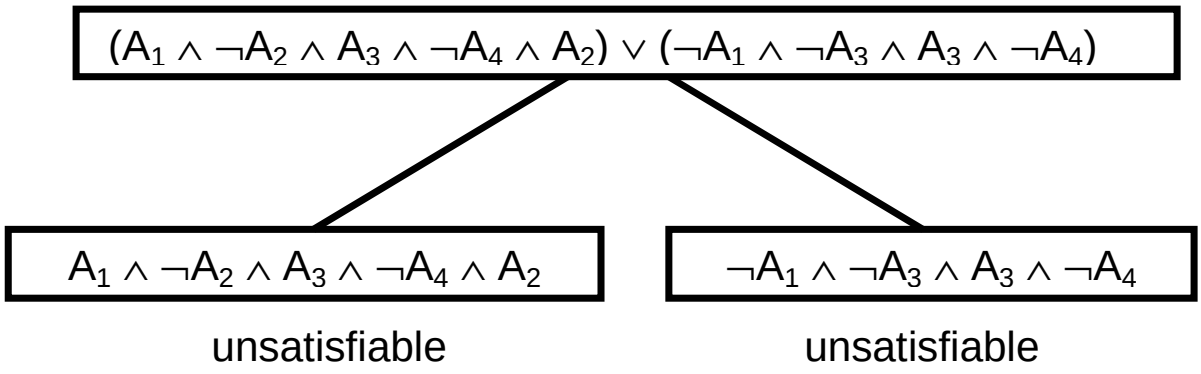
Notice that expansion need not proceed beyond discovery of the first satisfiable node. Thus, a computational process could be halted once the following partial tree is discovered.,



Example: Let

$$\varphi = (A_1 \wedge \neg A_2 \wedge A_3 \wedge \neg A_4 \wedge A_2) \vee (\neg A_1 \wedge \neg A_3 \wedge A_3 \wedge \neg A_4).$$

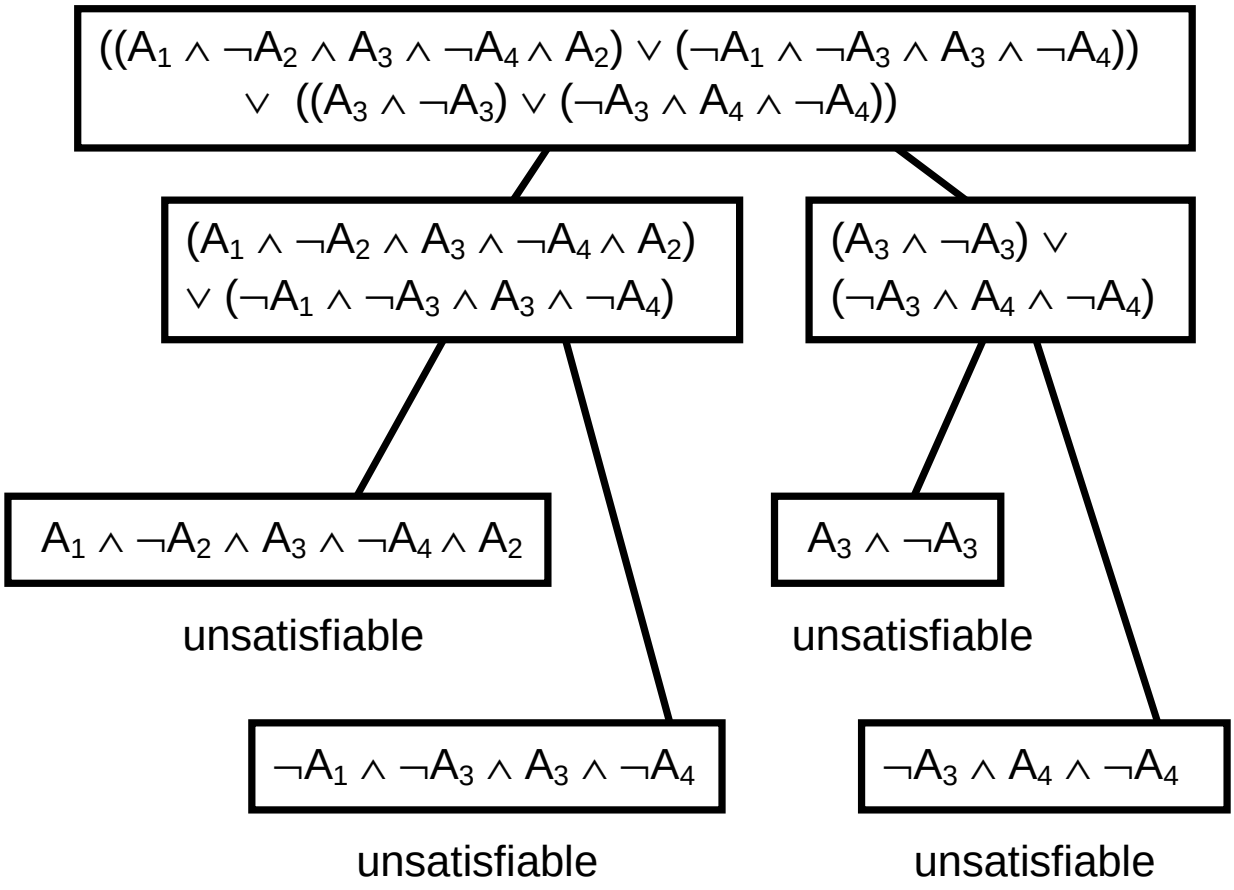
This formula is unsatisfiable, as is determined once the two children of the root are generated.



If the formula contains conjunctions and disjunctions which are nested more deeply, these rules may be applied recursively.

Example:

$\varphi = ((A_1 \wedge \neg A_2 \wedge A_3 \wedge \neg A_4 \wedge A_2)$
$\vee (\neg A_1 \wedge \neg A_3 \wedge A_3 \wedge \neg A_4))$
$\vee ((A_3 \wedge \neg A_3) \vee (\neg A_3 \wedge A_4 \wedge \neg A_4))$

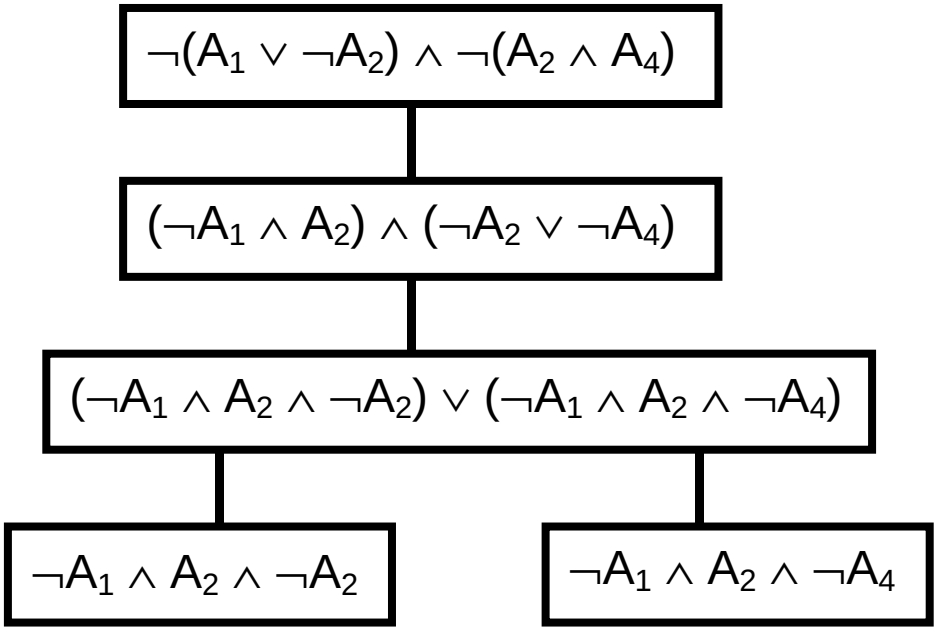


The technique just sketched only works under the following circumstances:

- All negations ($\neg$) are applied to atoms.
- The only connectives are $\wedge$ and $\vee$.

To relax these restrictions, we make use of various identities to place the formula in DNF.

Example: Let $\varphi = \neg(A_1 \vee \neg A_2) \wedge \neg(A_2 \wedge A_4)$.



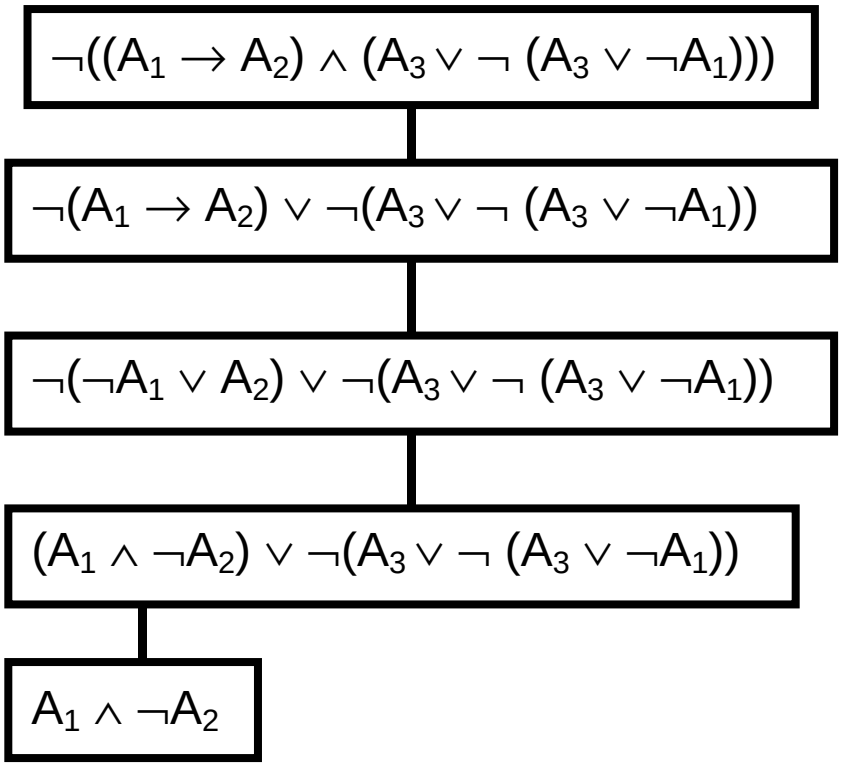
In the first step, de Morgan's identity was applied to move the negations inwards to the atoms.

In the second step, the distributive identity was applied to create a disjunction of conjunctions.

Now let us tackle the formula

$$\varphi = \neg((A_1 \rightarrow A_2) \wedge (A_3 \vee \neg\ (A_3 \vee \neg A_1)))$$

from the DNF example.



satisfiable

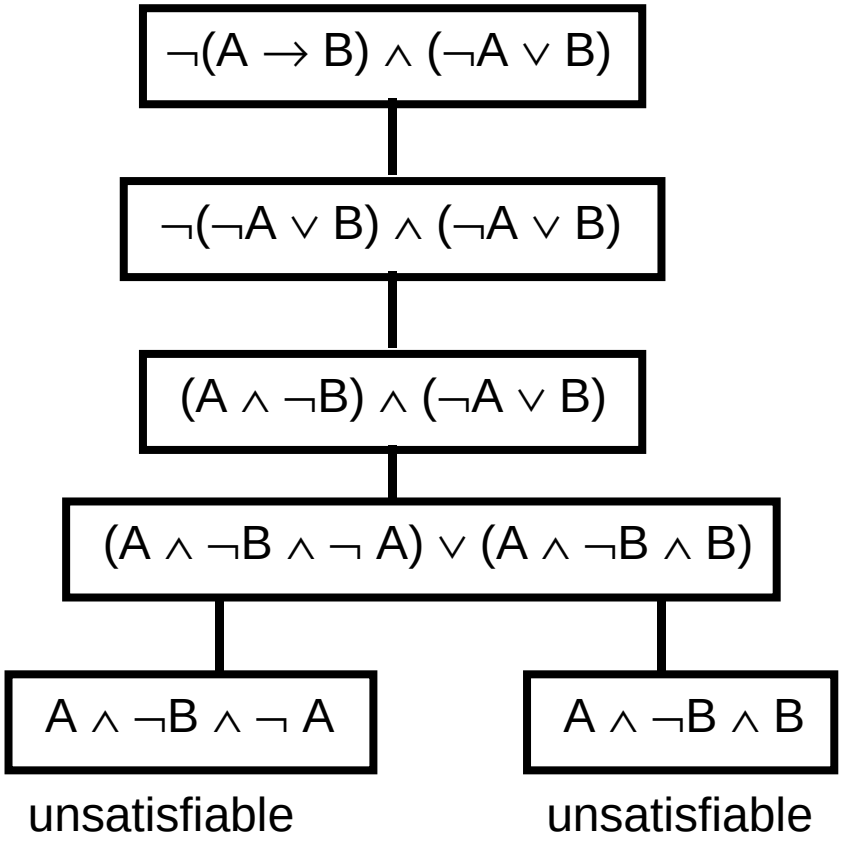
Note that the expansion terminates once a satisfiable node is found.

The rest of the formula need not be converted to DNF.

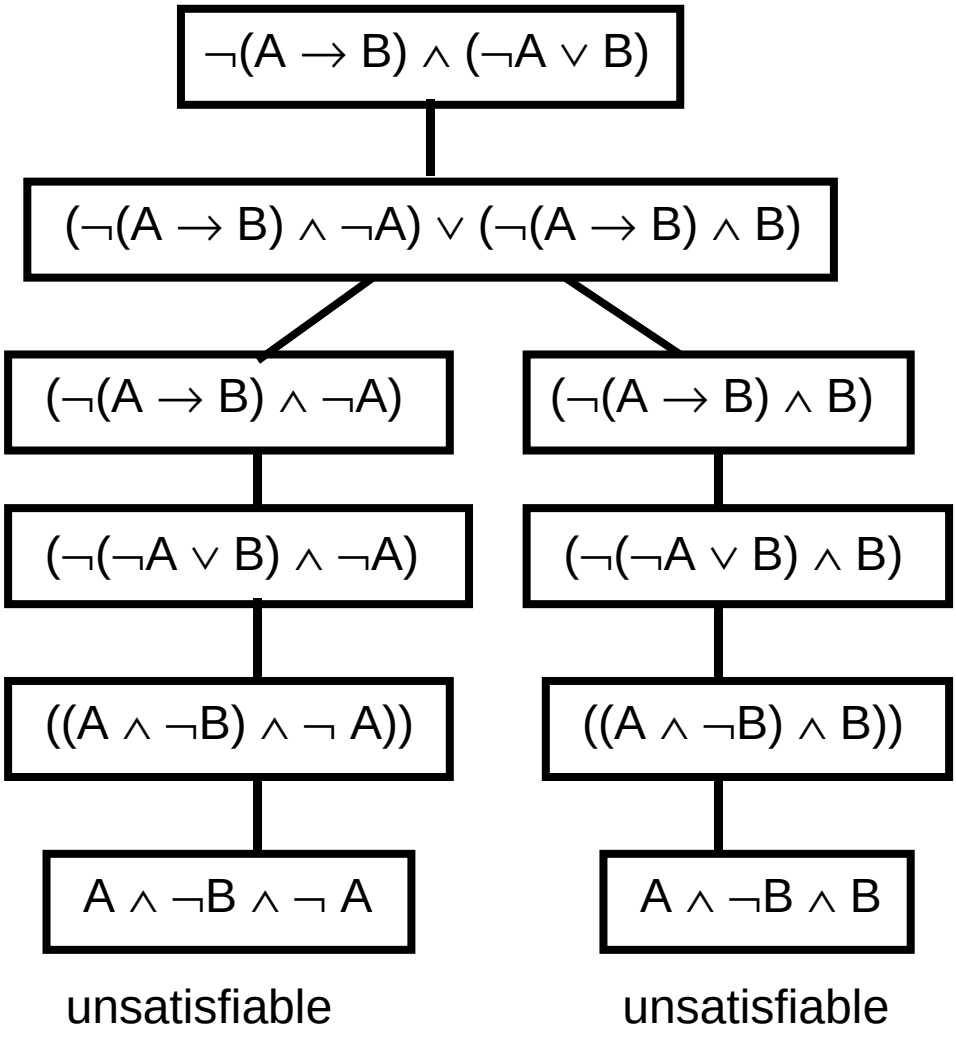
Example: Here is a solution of Example 2.1 from the textbook.

$\varphi = \neg(A \rightarrow B) \wedge (\neg A \vee B)$

Here is our version of Figure 2.2 of the textbook.



Here is the same example with the first expansion from the textbook (Figure 2.1).



Example: This is a restatement of Example 2.2 from the textbook. Suppose that it is desired to prove $\varphi = (\neg A \vee D)$ from the following set of axioms:

$$\Phi = \{ (\neg A \vee B), \neg(B \wedge \neg C), (C \rightarrow D) \}.$$

In other words, it is desired to establish that $\Phi \models \varphi$ holds. As noted earlier, if we define

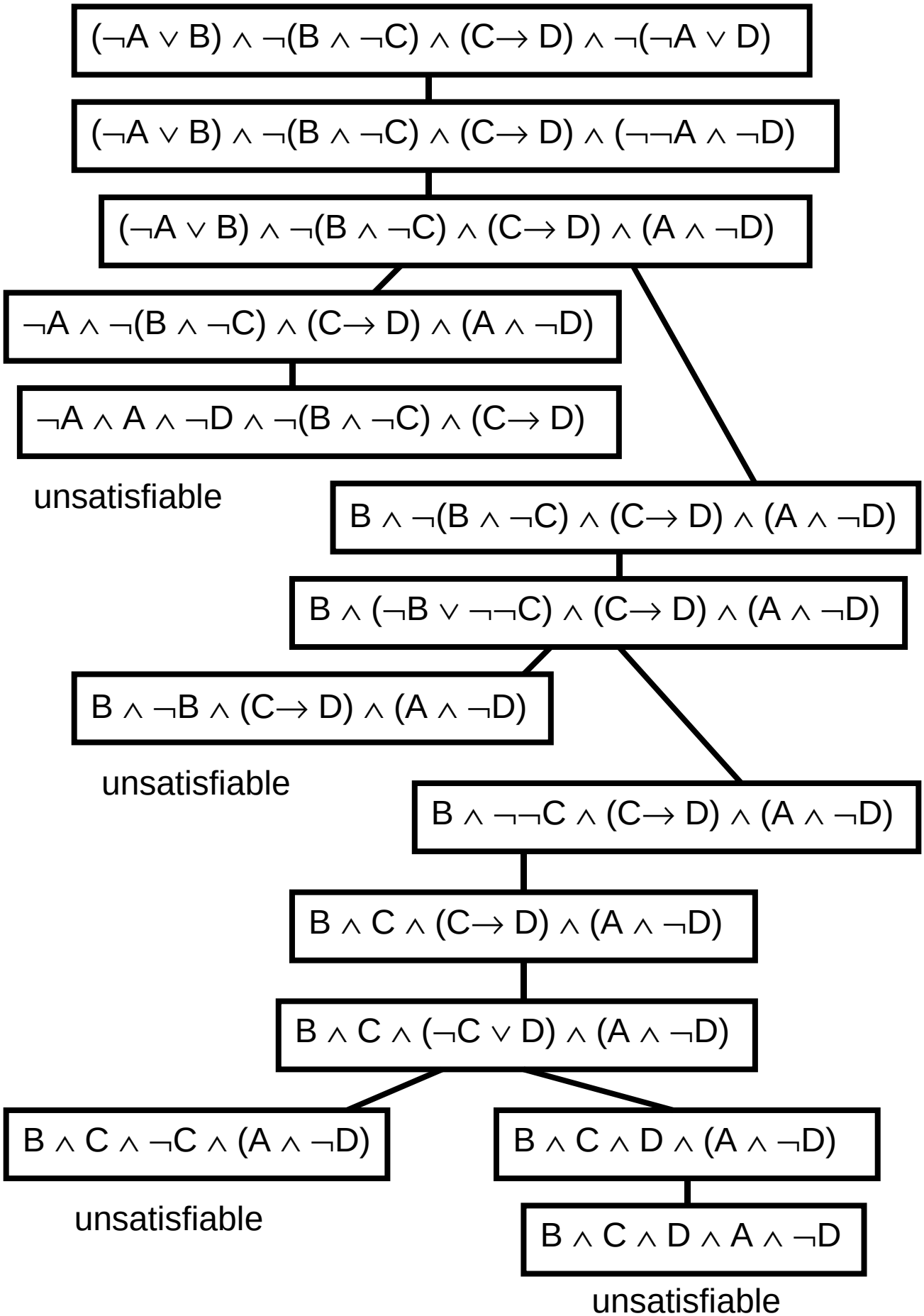
$$\Phi_{\wedge} = (\neg A \vee B) \wedge \neg(B \wedge \neg C) \wedge (C \rightarrow D)$$

then $\Phi \models \varphi$ holds iff $(\Phi_{\wedge} \wedge \neg\varphi)$ is unsatisfiable.

We may use the method of semantic tableaux to show satisfiability and unsatisfiability. The formula which we will start with is thus

$(\Phi_{\wedge} \wedge \neg\varphi) =$
$(\neg A \vee B) \wedge \neg(B \wedge \neg C) \wedge (C \rightarrow D) \wedge \neg(\neg A \vee D).$

The graph on the next page follows the general plan of Figures 2.2 – 2.9 of the textbook.


(¬A ∨ B) ∧ ¬(B ∧ ¬C) ∧ (C→ D) ∧ ¬(¬A ∨ D)
(¬A ∨ B) ∧ ¬(B ∧ ¬C) ∧ (C→ D) ∧ (¬¬A ∧ ¬D)
(¬A ∨ B) ∧ ¬(B ∧ ¬C) ∧ (C→ D) ∧ (A ∧ ¬D)
¬A ∧ ¬(B ∧ ¬C) ∧ (C→ D) ∧ (A ∧ ¬D)
¬A ∧ A ∧ ¬D ∧ ¬(B ∧ ¬C) ∧ (C→ D)
unsatisfiable
B ∧ ¬(B ∧ ¬C) ∧ (C→ D) ∧ (A ∧ ¬D)
B ∧ (¬B ∨ ¬¬C) ∧ (C→ D) ∧ (A ∧ ¬D)
B ∧ ¬B ∧ (C→ D) ∧ (A ∧ ¬D)
unsatisfiable
B ∧ ¬¬C ∧ (C→ D) ∧ (A ∧ ¬D)
B ∧ C ∧ (C→ D) ∧ (A ∧ ¬D)
B ∧ C ∧ (¬C ∨ D) ∧ (A ∧ ¬D)
B ∧ C ∧ ¬C ∧ (A ∧ ¬D)
unsatisfiable
B ∧ C ∧ D ∧ (A ∧ ¬D)
B ∧ C ∧ D ∧ A ∧ ¬D
unsatisfiable

Example: We now solve Example 2.4 of the text (solved earlier using a simple truth-table construction.) using semantic tableaux.

The formula to be checked is

$$\Phi_{\wedge} \wedge \neg\varphi = (J \vee Y) \wedge (Y \rightarrow (\neg S \rightarrow C)) \wedge (\neg J \rightarrow S) \wedge \neg C$$

A semantic tableau for this problem is shown on the next slide. In this slide, it is attempted to follow the pattern of Figure 2.12 of the textbook as closely as possible.

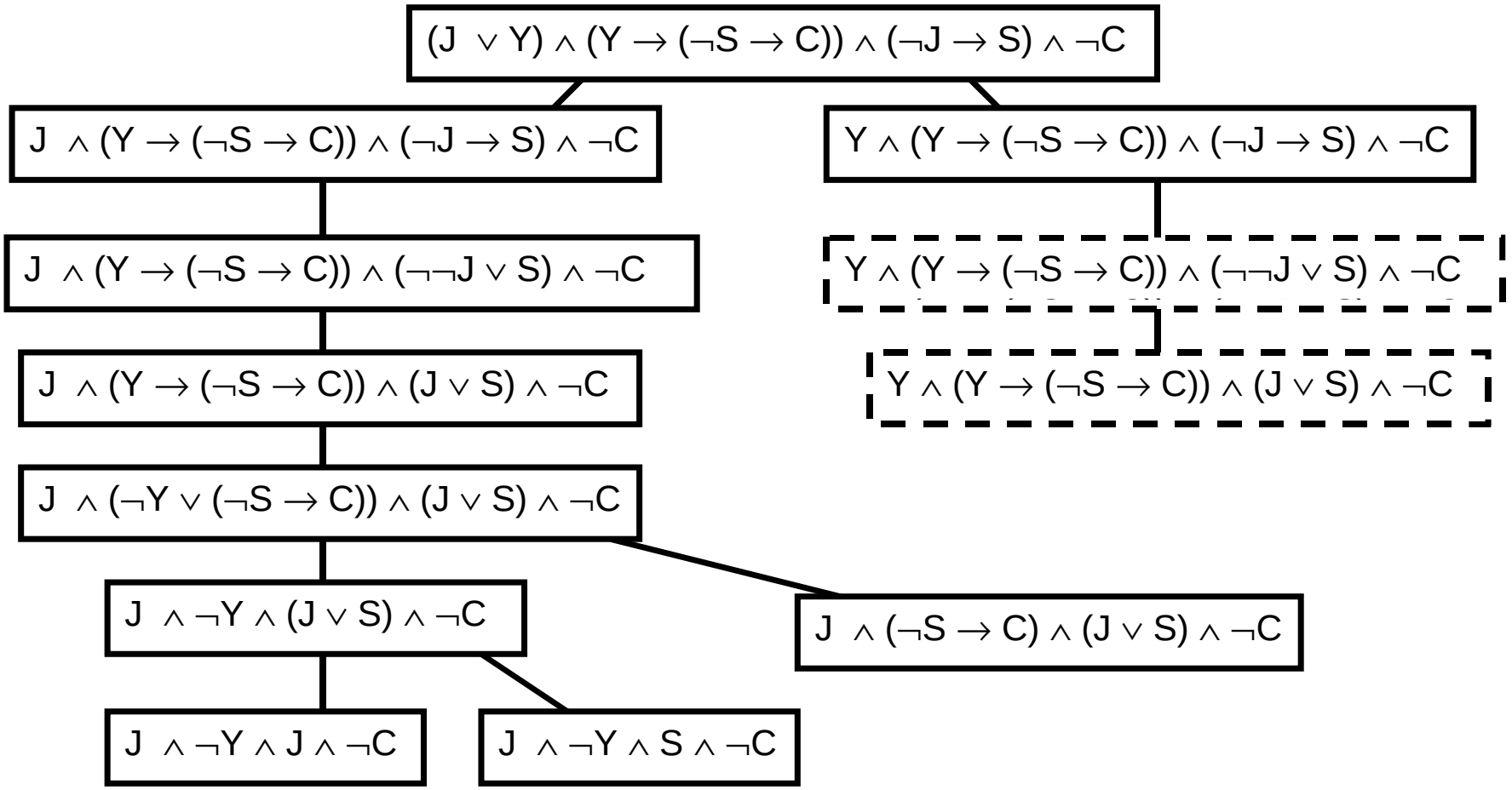
Notes:

- The author applies the conversion of $(\neg J \rightarrow S)$ to two branches at once. This is not necessary, and in this example, it turns out to be superfluous.

- Once a satisfiable branch is found, no further computation is necessary, since the question of whether or not other branches are satisfiable is irrelevant.

- The model defining the satisfiability may be read off from the satisfiable node.

- Nodes generated in the text example, but not needed, are shown in dashed lines.


(J ∨ Y) ∧ (Y → (¬S → C)) ∧ (¬J → S) ∧ ¬C
J ∧ (Y → (¬S → C)) ∧ (¬J → S) ∧ ¬C
Y ∧ (Y → (¬S → C)) ∧ (¬J → S) ∧ ¬C
J ∧ (Y → (¬S → C)) ∧ (¬¬J ∨ S) ∧ ¬C
Y ∧ (Y → (¬S → C)) ∧ (¬¬J ∨ S) ∧ ¬C
J ∧ (Y → (¬S → C)) ∧ (J ∨ S) ∧ ¬C
Y ∧ (Y → (¬S → C)) ∧ (J ∨ S) ∧ ¬C
J ∧ (¬Y ∨ (¬S → C)) ∧ (J ∨ S) ∧ ¬C
J ∧ ¬Y ∧ (J ∨ S) ∧ ¬C
J ∧ (¬S → C) ∧ (J ∨ S) ∧ ¬C
J ∧ ¬Y ∧ J ∧ ¬C
J ∧ ¬Y ∧ S ∧ ¬C
satisfiable

Summary of rules for semantic tableaux:

Leaf nodes:

- A wff may represent a terminal node iff it is of the form
$$\varphi_1 \wedge \varphi_2 \wedge .. \wedge \varphi_k,$$
with each φ_i a wff.

- A terminal node represented by the above formula is unsatisfiable iff there is some pair
$$\{ \varphi_p, \varphi_q \}$$
of conjuncts from that formula which forms a complementary pair of literals. The forms of the other φ_i's are irrelevant.

- A terminal node represented by the above formula is satisfiable iff each φ_i is a literal, and no two of those literals forms a complementary pair.

Rules for satisfiablity:

- To show that a formula is satisfiable, it suffices to find one leaf node which is satisfiable.

- To show that a formula is unsatisfiable, it must be shown that all leaf nodes are unsatisfiable.

Some general remarks on the method of semantic tableaux:

Let φ be a propositional formula. A graph with φ as the root, and constructed as shown in the examples on these slides, is called a *semantic tableau* for φ.

Theorem: Let φ be a propositional formula. Then φ is satisfiable iff every semantic tableau for φ may be extended to one which contains a path from the root to a node consisting of the conjunction of a satisfiable set of literals. □

This result will not be proven formally, although from the discussion about DNF, it should not be difficult to believe. Furthermore, the process is algorithmic, as stated below.

Observation: Every semantic tableau for φ may be extended to one for which every path leads to a leaf which is labelled with a conjunction of literals. □

Theorem: The process of constructing a semantic tableau for φ is a finite one. It will always terminate.