St. Francis Xavier University
Department of Computer Science

CSCI 544: Computational Logic
Lecture 3: Propositional Logic II—Semantic Tableaux
Winter 2023

1 Decision Procedures

To begin this lecture, let's take a brief detour to discuss a core concept of theoretical computer science. A *decision problem* is a problem for which each input instance corresponds to either a "yes" or a "no" answer. If there exists a procedure that produces the correct "yes" or "no" answer for any instance of a given decision problem, and that procedure halts on all inputs, then we say that the decision problem is *decidable*. If no such procedure exists, then we say that the decision problem is *undecidable*.

Decidability plays a key role in computational logic, as certain theories have this desirable property of decidability while other theories lack the property. Thus, if we want to use a computer to perform any meaningful tasks surrounding a particular theory, we'd better hope the problems we wish to solve are indeed decidable.

While you may be familiar with the definition of decidability and decision procedures from a previous course on theoretical computer science, we will use the following formal logic-based definition to frame the notion of decidability in this course.

Definition 1 (Decision procedure)**.** Let S be a set of formulas. We say that an algorithm is a decision procedure for S if, given an arbitrary formula A, the algorithm halts and returns "yes" if $A \in S$ or halts and returns "no" if $A \notin S$.

Recall that the two fundamental semantic properties we care about in a logical system are satisfiability and validity. It's straightforward for us to develop a decision procedure to decide whether satisfiability holds: we only need to construct a truth table for a given formula. If any entry of the truth table corresponding to the formula's column takes a true value, then we can conclude that the formula is satisfiable.

Remark. Those of you who are well-versed in theory may recall that the Boolean satisfiability problem is NP-complete, implying that there is no efficient (deterministic polynomial-time) algorithm for deciding the problem. The decision procedure we just introduced may shed some light as to why this is the case: if our formula has n propositions, then we must evaluate 2^n possible interpretations with our truth table.

To develop a decision procedure for validity, on the other hand, we need look no further than the decision procedure we just described for satisfiability. Since we know that a formula A is valid if and only if $\neg A$ is unsatisfiable, all we need to do is construct a truth table for $\neg A$ and check whether $\neg A$ is satisfiable. If it isn't, then we can conclude that A is valid. This style of procedure is known as a *refutation procedure*, since we're proving the validity of A by refuting the negation of A.

After defining the syntax and semantics of a logical system, our focus should naturally turn to what we can do with that system. Thus, in this and the following lectures, we will study some decision procedures for propositional logic (and, later, predicate logic).

2 The Semantic Tableau Method

The first decision procedure for satisfiability that we will study for propositional logic is the *method of semantic tableaux*.[1] A semantic tableau is essentially a tree where each vertex contains a subformula of the original formula to be proved. The original formula is located at the root of the tree, and the leaves contain sets of propositions and their negations. As we progress through the tree from top to bottom, we decompose the formula into smaller subformulas until we arrive at an aforementioned set of propositions. If there exists one such set that is satisfiable (i.e., if the set does not contain both a proposition p and its negation $\neg p$), then we conclude that the original formula is satisfiable.

We can formalize some of these notions as follows.

Definition 2 (Literals)**.** A literal is either a proposition or the negation of a proposition. More precisely, a positive literal is a proposition, and a negative literal is the negation of a proposition.

Definition 3 (Complementary pair of literals)**.** For any literal p, the set $\{p, \neg p\}$ is called a complementary pair of literals.

Example 4. Consider the set $L = \{p, q, \neg q, r\}$. In this set, p, q, and r are positive literals, and $\neg q$ is a negative literal. The set also contains the complementary pair of literals $\{q, \neg q\}$.

Thus, the goal of the semantic tableau method is to check, for all sets of literals obtainable from a given formula, whether that set contains a complementary pair of literals.

Theorem 5. *A set of literals is satisfiable if and only if the set does not contain a complementary pair of literals.*

Proof. Suppose L is a set of literals that does not contain a complementary pair of literals. Define the interpretation $\mathscr{I}$ to be as follows:

$$v_{\mathscr{I}}(p) = \begin{cases} \mathrm{T} & \text{if } p \in L; \text{ and} \\ \mathrm{F} & \text{if } \neg p \in L. \end{cases}$$

Since there is no complementary pair of literals in L by our assumption, only one truth value is assigned to each literal p in L. Moreover, by this interpretation, every literal in L evaluates to true, and so the set L is satisfiable.

Conversely, suppose that L does contain a complementary pair of literals; say, $\{p, \neg p\} \in L$. Then, for any interpretation $\mathscr{I}$, either $v_{\mathscr{I}}(p) = \mathrm{F}$ or $v_{\mathscr{I}}(\neg p) = \mathrm{F}$, so L cannot be satisfiable. □

2.1 Constructing a Semantic Tableau

To determine whether a formula is satisfiable, we can simply use our set of inductive truth value rules from the previous lecture and consider the truth values of each subformula of the formula.

Example 6. Consider the formula $A = q \wedge (\neg p \vee \neg q)$. Let $\mathscr{I}$ be an arbitrary interpretation. We know that

- $v_{\mathscr{I}}(A) = \mathrm{T}$ if and only if both $v_{\mathscr{I}}(q) = \mathrm{T}$ and $v_{\mathscr{I}}(\neg p \vee \neg q) = \mathrm{T}$; and
- $v_{\mathscr{I}}(\neg p \vee \neg q) = \mathrm{T}$ if and only if either $v_{\mathscr{I}}(\neg p) = \mathrm{T}$ or $v_{\mathscr{I}}(\neg q) = \mathrm{T}$.

Thus, $v_{\mathscr{I}}(A) = \mathrm{T}$ if and only if $\mathscr{I}$ is such that

1. $v_{\mathscr{I}}(q) = \mathrm{T}$ and $v_{\mathscr{I}}(\neg p) = \mathrm{T}$; or
2. $v_{\mathscr{I}}(q) = \mathrm{T}$ and $v_{\mathscr{I}}(\neg q) = \mathrm{T}$.

[1] *Tableaux* is the plural form of the word *tableau*.

The second scenario is a contradiction, as that set of literals contains a complementary pair of literals. Thus, A is satisfied by the first interpretation.

Example 7. Consider the formula $B = (\neg p \wedge \neg q) \wedge (p \vee q)$. Let $\mathscr{I}$ be an arbitrary interpretation. We know that

- $v_{\mathscr{I}}(B) = \mathrm{T}$ if and only if both $v_{\mathscr{I}}(\neg p \wedge \neg q) = \mathrm{T}$ and $v_{\mathscr{I}}(p \vee q) = \mathrm{T}$;
- $v_{\mathscr{I}}(\neg p \wedge \neg q) = \mathrm{T}$ if and only if both $v_{\mathscr{I}}(\neg p) = \mathrm{T}$ and $v_{\mathscr{I}}(\neg q) = \mathrm{T}$; and
- $v_{\mathscr{I}}(p \vee q) = \mathrm{T}$ if and only if either $v_{\mathscr{I}}(p) = \mathrm{T}$ or $v_{\mathscr{I}}(q) = \mathrm{T}$.

Thus, $v_{\mathscr{I}}(B) = \mathrm{T}$ if and only if $\mathscr{I}$ is such that

1. $v_{\mathscr{I}}(\neg p) = \mathrm{T}$, $v_{\mathscr{I}}(\neg q) = \mathrm{T}$, and $v_{\mathscr{I}}(p) = \mathrm{T}$; or
2. $v_{\mathscr{I}}(\neg p) = \mathrm{T}$, $v_{\mathscr{I}}(\neg q) = \mathrm{T}$, and $v_{\mathscr{I}}(q) = \mathrm{T}$.

However, both scenarios are contradictions, as both sets of literals contain a complementary pair of literals. Therefore, B is unsatisfiable.

As you may have noticed, even for small formulas like A and B in the previous examples, this "list-style" formatting can become hard to follow. Hence, we introduce the "tree-style" formatting of semantic tableaux to make clearer each step of the process.

At the root of our semantic tableau, we place the original formula. Then, each time we go down one level in the tree, we find either one or two children depending on how the parent vertex is decomposed. Each leaf of the tree contains a set of literals, and we additionally label each leaf in one of two ways:

- If a leaf corresponds to a set of literals having no complementary pair of literals, then we label it with the symbol $\odot$.
- If a leaf corresponds to a set of literals having at least one complementary pair of literals, then we label it with the symbol $\times$.

We say that a tableau whose leaves are all marked by $\times$ is a *closed tableau*, while a tableau with at least one leaf marked by $\odot$ is an *open tableau*.

Now that we've handled leaves, how do we decompose formulas into subformulas? The heuristic we will use categorizes formulas according to the primary logical connective employed in the formula.

- The first category of *α-formulas* corresponds to formulas using conjunction, where both subformulas α_1 and α_2 must be satisfied in order to satisfy the original formula α.
- The second category of *β-formulas* corresponds to formulas using disjunction, where either subformula β_1 or β_2 may be satisfied in order to satisfy the original formula β.

The rules for both α-formulas and β-formulas are summarized in the following tables:

α	α_1	α_2
$A_1 \wedge A_2$	A_1	A_2
$\neg(A_1 \vee A_2)$	$\neg A_1$	$\neg A_2$
$\neg(A_1 \Rightarrow A_2)$	A_1	$\neg A_2$
$A_1 \Leftrightarrow A_2$	$A_1 \Rightarrow A_2$	$A_2 \Rightarrow A_1$

Table 1: Rules for α-formulas

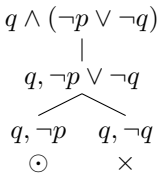
β	β_1	β_2
$\neg(B_1 \wedge B_2)$	$\neg B_1$	$\neg B_2$
$B_1 \vee B_2$	B_1	B_2
$B_1 \Rightarrow B_2$	$\neg B_1$	B_2
$\neg(B_1 \Leftrightarrow B_2)$	$\neg(B_1 \Rightarrow B_2)$	$\neg(B_2 \Rightarrow B_1)$

Table 2: Rules for β-formulas

Thus, if some vertex of our semantic tableau is of the form $\{S, \alpha\}$ where S is some other set of subformulas, then the child of that vertex will be of the form $\{S, \alpha_1, \alpha_2\}$. Similarly, if some vertex of our semantic tableau is of the form $\{S, \beta\}$, then the two children of that vertex will be of the form $\{S, \beta_1\}$ and $\{S, \beta_2\}$, respectively.

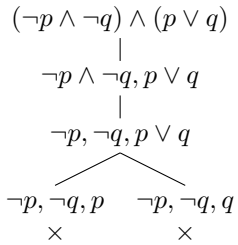
Additionally, there is one special rule for double negation: if $\alpha = \neg\neg A_1$, then $\alpha_1 = A_1$ and there is no corresponding α_2.

Example 8. The following semantic tableau corresponds to our formula $A = q \wedge (\neg p \vee \neg q)$ from an earlier example:



This is an open tableau, since at least one leaf is marked by $\odot$. Thus, the formula is satisfiable.

Example 9. The following semantic tableau corresponds to our formula $B = (\neg p \wedge \neg q) \wedge (p \vee q)$ from an earlier example:



This is a closed tableau, since all leaves are marked by $\times$. Thus, the formula is unsatisfiable.

Note that this particular semantic tableau was produced by decomposing the conjunctive term first at level 1 of the tree. A different semantic tableau results by decomposing the disjunctive term first instead, but the outcome remains the same.[2]

Our technique for constructing a semantic tableau, given an input formula A, is summarized in the following algorithm.

Algorithm 1: Semantic tableau construction

$T \leftarrow$ a tree consisting of a root labelled by $\{A\}$
while some unmarked leaf is in T **do**
  $l \leftarrow$ an unmarked leaf labelled by a set of formulas $U(l)$
  if $U(l)$ is a set of literals **then**
    if $U(l)$ contains a complementary pair of literals **then**
      mark l with $\times$
    else
      mark l with $\odot$
  if $U(l)$ is not a set of literals **then**
    $A' \leftarrow$ a formula in $U(l)$ that is not a literal
    if A' is an α-formula **then**
      $l' \leftarrow$ a new child vertex of l labelled by $U(l') = (U(l) \setminus \{A'\}) \cup \{\alpha_1, \alpha_2\}$
    if A' is a β-formula **then**
      $l' \leftarrow$ a new child vertex of l labelled by $U(l') = (U(l) \setminus \{A'\}) \cup \{\beta_1\}$
      $l'' \leftarrow$ a new child vertex of l labelled by $U(l'') = (U(l) \setminus \{A'\}) \cup \{\beta_2\}$
return T

Since our algorithm progressively decomposes formulas into smaller subformulas and marks every set of literals with either $\times$ or $\odot$, it is clear that we will eventually exit the while loop and our algorithm will terminate on any input.

[2]Generally, it's preferable to decompose conjunctive terms before disjunctive terms, since this results in the semantic tableau having fewer vertices.

2.2 Soundness and Completeness

Having presented our algorithm for constructing a semantic tableau, all that remains is for us to prove the algorithm is correct. The process of constructing a semantic tableau is very mechanical, since we're following a clearly-defined set of rules and applying those rules in the same manner and sequence at each step. Thus, it seems natural for us to assume that the algorithm must be correct! However, "correctness" in this context goes beyond whether or not the algorithm simply produces a semantic tableau. What we must do is connect the syntactic notion of the tableau itself to the semantic notion of a truth value; in other words, we must show that our algorithm constructs an open tableau if and only if there exists a model for the given input formula.

The big result we will be proving in this section is the following:

Theorem 10 (Soundness and completeness of semantic tableaux)**.** *Let T be a semantic tableau for a formula A. Then T is a closed tableau if and only if A is unsatisfiable.*

This theorem asserts two things: the soundness of the method of semantic tableaux, and the completeness of the method of semantic tableaux. The method being *sound* means that, if the method produces a closed tableau for a formula $\neg A$, then the formula A is valid. On the other hand, the method being *complete* means that, if A is a valid formula, then we can prove this by constructing a closed tableau for $\neg A$.

The proof of this theorem is lengthy and quite involved, so we will prove it in parts. However, for the time being, let's make a few observations about consequences of this theorem.

Naturally, the following corollary is a direct consequence of our theorem:

Corollary 11. *Let T be a semantic tableau for a formula A. Then T is an open tableau if and only if A is satisfiable.*

Proof. T is an open tableau if and only if T is not a closed tableau, and A is satisfiable if and only if A is not unsatisfiable. □

Though the method of semantic tableaux is used to check satisfiability, we can easily adapt it to check validity by following the observations we made in the previous lecture.

Corollary 12. *Let A be a formula. Then A is valid if and only if the semantic tableau for $\neg A$ is a closed tableau.*

Proof. By our definitions of validity and satisfiability, the formula A is valid if and only if $\neg A$ is unsatisfiable. By Theorem 10, $\neg A$ is unsatisfiable if and only if its semantic tableau is a closed tableau. □

Altogether then, we can conclude that the method of semantic tableaux is a decision procedure for validity.

2.2.1 Proving Soundness

Following the wording of Theorem 10, in order for us to prove soundness, we must prove that if the semantic tableau T for a formula A is a closed tableau, then A is unsatisfiable. This is one direction of the biconditional statement given in the theorem.

Our proof will tackle a more general claim. Denote by T_n the subtree rooted at some vertex n in the tableau T. We will prove that if T_n is itself a closed tableau, then the set of formulas $U(n)$ labelling the vertex n is unsatisfiable. We can then apply our proof to establish soundness by taking n to be the root of the overall tree.

Proof of Soundness. We prove by induction on the height of the vertex n in T_n, denoted h_n. Recall that a set of formulas is unsatisfiable if and only if there exists no interpretation where all formulas in the set evaluate to true.

For the base case ($h_n = 0$), we know that n is a leaf. If T_n is a closed tableau, then this implies that $U(n)$ contains a complementary set of literals, and so the set is unsatisfiable.

For the inductive case, suppose that n is a vertex such that $h_n > 0$ in T_n. By our inductive hypothesis, we know that for any vertex m with height $h_m < h_n$, the subtree T_m being a closed tableau implies that $U(m)$ is unsatisfiable.

Since $h_n > 0$, some rule for either an α-formula or a β-formula must have been applied to produce the children of n. Without loss of generality, suppose that $A = A_1 \wedge A_2$ in the α-formula case, and $B = B_1 \vee B_2$ in the β-formula case.

- **Case 1: α-formulas.** In this case, n is labelled by $U_0 \cup \{A_1 \wedge A_2\}$ for some (possibly empty) set of formulas U_0. Assuming that T_n is a closed tableau and that n' is the single child vertex of n, we know that $T_{n'}$ must also be a closed tableau. Moreover, by our inductive hypothesis, $U(n')$ is unsatisfiable.

  Let $\mathscr{I}$ be an arbitrary interpretation. We consider three subcases:

  - If $v_{\mathscr{I}}(A') = \mathrm{F}$ for some formula $A' \in U_0$, then by the fact that $U_0 \subseteq U(n)$, we have that $U(n)$ is unsatisfiable.
  - If $v_{\mathscr{I}}(A_1) = \mathrm{F}$, then we know by our truth value rules that $v_{\mathscr{I}}(A_1 \wedge A_2) = \mathrm{F}$ as well. Since $\{A_1 \wedge A_2\} \in U(n)$, we have that $U(n)$ is unsatisfiable.
  - If $v_{\mathscr{I}}(A_2) = \mathrm{F}$, then the outcome is analogous to the previous subcase.

- **Case 2: β-formulas.** In this case, n is labelled by $U_0 \cup \{B_1 \vee B_2\}$ for some (possibly empty) set of formulas U_0. Assuming that T_n is a closed tableau and that n' and n'' are the two child vertices of n, we know that both $T_{n'}$ and $T_{n''}$ must also be closed tableaux. Moreover, by our inductive hypothesis, both $U(n')$ and $U(n'')$ are unsatisfiable.

  Let $\mathscr{I}$ be an arbitrary interpretation. We consider two cases:

  - If $v_{\mathscr{I}}(B') = \mathrm{F}$ for some formula $B' \in U_0$, then by the fact that $U_0 \subseteq U(n)$, we have that $U(n)$ is unsatisfiable.
  - If both $v_{\mathscr{I}}(B_1) = \mathrm{F}$ and $v_{\mathscr{I}}(B_2) = \mathrm{F}$, then we know by our truth value rules that $v_{\mathscr{I}}(B_1 \vee B_2) = \mathrm{F}$ as well. Since $\{B_1 \vee B_2\} \in U(n)$, we have that $U(n)$ is unsatisfiable.

In either case, we obtain the desired result. Therefore, the claim holds by the principle of mathematical induction. □

2.2.2 Proving Completeness

Again, following the wording of Theorem 10, proving completeness requires us to show that if a formula A is unsatisfiable, then any semantic tableau T is a closed tableau. This is the other direction of the theorem's biconditional statement.

In general, proving completeness is more difficult than proving soundness. Intuitively, this is because the logical systems we build only consist of rules that are sound, or else our systems wouldn't make any sense. On the other hand, verifying that we've included all of the rules that are necessary to prove what we want is a more involved process.

In our proof of soundness, we considered one semantic tableau that was closed, and we showed that the corresponding formula was unsatisfiable by induction. Here, we must show that for any unsatisfiable formula A, we can construct the semantic tableau in *any* way according to our rules and it will always be closed.

We can make our job of proving this claim easier by rephrasing the wording. Recall that Corollary 11 tells us that a semantic tableau T is open if and only if the corresponding formula A is satisfiable. Thus, to prove completeness, it suffices for us to show that if *some* semantic tableau for A is open, then A is satisfiable.[3]

[3]Observe that this is the contrapositive of the original statement of completeness we considered.

If we have an open tableau T, then the model corresponding to the set of literals in an open leaf can be extended to form an interpretation for the overall formula A. For example, if an open leaf of our tableau contains the set of literals $\{p, \neg q\}$ and our formula A contains literals p, q, and r, then it's clear that our formula is satisfied by the truth value assignments $v(p) = \mathrm{T}$ and $v(q) = \mathrm{F}$, so we just need to assign either true or false to r in order to form an interpretation.

Thus, the basis of our proof of completeness will have us show that we can perform this extension from the truth value assignments obtained from an open leaf of the tableau to an overall model for the formula. We can do this in four steps.

The first step is to define a special property of sets of formulas that we will use. Essentially, this property ensures both that a set of formulas doesn't contain a complementary pair of literals and that the set is *downward saturated*; that is, as we decompose the set of formulas, we don't completely lose any literals or terms that we saw previously in the set. We say that any set of formulas having this property is a *Hintikka set*, named for the Finnish logician Jaakko Hintikka.

Definition 13 (Hintikka set)**.** Let U be a set of formulas. We say that U is a Hintikka set if and only if

1. for all literals p appearing in a formula in U, either $p \notin U$ or $\neg p \notin U$;
2. if $A \in U$ is an α-formula, then both $A_1 \in U$ and $A_2 \in U$; and
3. if $B \in U$ is a β-formula, then either $B_1 \in U$ or $B_2 \in U$.

The second step is to show that any set of formulas labelling the vertices in an open branch of our semantic tableau form a Hintikka set. Establishing this will allow us to traverse from the open leaf along the branch and up to the root of the semantic tableau, where the original formula is located, without affecting the special property. We show this by way of the following theorem.

Theorem 14. *Let l be an open leaf in a semantic tableau T, and let $U = \bigcup_i U(i)$, where i indexes the set of vertices along the branch from the root of T to l. Then U is a Hintikka set.*

Proof. We consider three cases, each corresponding to one of the three properties of a Hintikka set.

- **Case 1.** Suppose some literal p or $\neg p$ appears for the first time at a vertex n. Then, that literal will also appear at all vertices k along the branch from n to l. This is because literals are not decomposed. As a result, all literals that appear in U will also appear in $U(l)$, and since l is an open leaf, no complementary pair of literals appears in $U(l)$.
- **Case 2.** Suppose $A \in U$ is an α-formula. At some point in the semantic tableau, A must have been decomposed at some vertex n along the branch from the root to l. As a result, $\{A_1, A_2\} \subseteq U(n') \subseteq U$, where n' is the child vertex of n.
- **Case 3.** Suppose $B \in U$ is a β-formula. At some point in the semantic tableau, B must have been decomposed at some vertex n along the branch from the root to l. As a result, either $\{B_1\} \subseteq U(n') \subseteq U$ or $\{B_2\} \subseteq U(n') \subseteq U$, where n' is one of the two child vertices of n.

In all cases, the conditions hold as desired. □

The third step, which we will not discuss in any great detail, is to show that Hintikka sets are satisfiable. For this, we rely on the following lemma, whose proof we omit.

Lemma 15 (Hintikka's lemma)**.** *Every Hintikka set U is satisfiable.*

Proof. Omitted. □

Finally, in the fourth step, we combine all of the work we've done and we prove completeness.

Proof of Completeness. Let T be a semantic tableau for a given formula A, and suppose that T is an open tableau. Then $U = \bigcup_i U_i$, where i indexes the set of vertices along an open branch of T, is a Hintikka set by Theorem 14. Furthermore, by Lemma 15, we know that there exists an interpretation $\mathscr{I}$ that satisfies U. Since A is an element of U—being at the root of the semantic tableau T—we conclude that $\mathscr{I}$ is a model for A and, therefore, A is satisfiable. □