St. Francis Xavier University
Department of Computer Science

CSCI 544: Computational Logic
Lecture 7: Predicate Logic II—Semantic Tableaux
Winter 2023

1 Revisiting the Semantic Tableau Method

In this lecture, we will revisit the semantic tableau method that we first introduced in our study of propositional logic, and we will extend it to work for predicate logic as well. Recall that, when we used semantic tableaux in propositional logic, we referred to the method as a *decision procedure*, as it gave us a way to determine whether or not a given formula is satisfiable. In the predicate logic case, unfortunately, the method of semantic tableaux is not a decision procedure. This is because, as we will see, we lose the guarantee that each branch of a semantic tableau will eventually close: it's possible for some branch to extend infinitely in a predicate logic semantic tableau.

Notwithstanding this fact, we can extend the method of semantic tableaux to work for predicate logic formulas with the appropriate modifications. Here, we will specify a method for constructing semantic tableaux, study some potential issues we must be aware of when using semantic tableau with predicate logic formulas, and establish both the soundness and completeness of the method.

1.1 Constructing a Semantic Tableau

Recall that a predicate can take as arguments either variables or constants; for example, the predicate $P(x, a)$ takes as arguments one variable x and one constant a. Each of the symbols for predicates, variables, and constants come from a countably infinite set, though individual predicates use only a finite number of symbols from these sets. When we build a semantic tableau, though, we must have full access to these sets; thus, if some formula $P(x, a)$ uses a constant a, we will assume the constant comes from the countably infinite set of constants $\{a_1, a_2, \dots\}$.

Likewise, we must have a way to handle the variables of a formula appropriately when they're quantified, and for this we also draw from our set of constant symbols. If we have a quantified formula A of the form $\exists x\ P(x)$ or $\forall x\ P(x)$ and a constant symbol a, then we say that the *instantiation* of A by a is the formula $A(a)$, where all free occurrences of x are replaced by a. In essence, we are performing substitution and taking a to be an example value that satisfies the quantifier $\exists$ or $\forall$.

Just as in propositional logic, we will make use of the definition of a *literal*, but we must first adapt this definition to work for predicates.

Definition 1 (Literals)**.** A literal is a closed atomic formula $P(a_1, \dots, a_k)$ all of whose arguments are constants, or the negation of such a formula $\neg P(a_1, \dots, a_k)$.

At the root of our semantic tableau, as well as at every other vertex of the tree, we place not only the current formula (or set of formulas), but also the set of constants a_i appearing in the formula. If no constants appear in a formula, then we just include the first constant a_0 from our countably infinite set. Then, each time we go down one level in the tree, we decompose the formula in some way depending on its structure.

As we did in our previous treatment of the method of semantic tableaux, we categorize formulas into α-formulas and β-formulas, depending on whether the formula involves conjunction or disjunction, respectively. We can copy our tables of α- and β-formulas and use them exactly as they are, so we will omit repeating them here. Now that we've introduced quantifiers, though, we require two additional categories of formula:

- The third category of γ-*formulas* corresponds to formulas that are universally quantified.

- The fourth category of *δ-formulas* corresponds to formulas that are existentially quantified.

The rules for both γ-formulas and δ-formulas are summarized in their own tables, where the rules essentially amount to the process of instantiation by the constant a:

γ	$\gamma(a)$
$\forall x\ A(x)$	$A(a)$
$\neg\exists x\ A(x)$	$\neg A(a)$

Table 1: Rules for γ-formulas

δ	$\delta(a)$
$\exists x\ A(x)$	$A(a)$
$\neg\forall x\ A(x)$	$\neg A(a)$

Table 2: Rules for δ-formulas

Having established everything necessary, we can specify our modified semantic tableau algorithm. The procedure to construct 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 $(\{A\}, \{a_{0_1}, \ldots, a_{0_k}\})$
▷ a_{0_1} through a_{0_k} are constants appearing in A
while some unmarked leaf is in T **do**
  $l \leftarrow$ an unmarked leaf labelled by a set of formulas $U(l)$ and constants $C(l)$
  if $U(l)$ is a set of literals **then**
    if $U(l)$ contains a complementary pair of literals **then**
      mark l with $\times$
  if $U(l)$ is not a set of literals **then**
    $A' \leftarrow$ a formula in $U(l)$ that is an α-, β-, or δ-formula
    if A' is an α-formula **then**
      $l' \leftarrow$ a new child vertex of l labelled by $((U(l) \setminus \{A'\}) \cup \{\alpha_1, \alpha_2\},\ C(l))$
    if A' is a β-formula **then**
      $l' \leftarrow$ a new child vertex of l labelled by $((U(l) \setminus \{A'\}) \cup \{\beta_1\},\ C(l))$
      $l'' \leftarrow$ a new child vertex of l labelled by $((U(l) \setminus \{A'\}) \cup \{\beta_2\},\ C(l))$
    if A' is a δ-formula **then**
      $l' \leftarrow$ a new child vertex of l labelled by $((U(l) \setminus \{A'\}) \cup \{\delta(a')\},\ C(l) \cup \{a'\})$
      ▷ a' is a constant not appearing in $U(l)$
  $\{\gamma_{l_1}, \ldots, \gamma_{l_m}\} \leftarrow$ the set of γ-formulas appearing in $U(l)$
  $C(l) \leftarrow \{c_{l_1}, \ldots, c_{l_k}\}$
  $l' \leftarrow$ a new child vertex of l labelled by $\left(U(l) \cup \left\{\cup_{i=1}^{m} \cup_{j=1}^{k} \gamma_{l_i}(c_{l_j})\right\},\ C(l)\right)$
  if $U(l)$ contains only literals and γ-formulas and $U(l') = U(l)$ **then**
    mark l with $\odot$
return T

Comparing this algorithm to our previous semantic tableau algorithm from an earlier lecture, we see that there are some major changes to take into account. First, as we mentioned earlier, vertices of the tree are now labelled by both sets of formulas and sets of constants. Second, the control flow of the algorithm is modified. If $U(l)$ is a set of literals, then we can only potentially mark a leaf with $\times$, but not with $\odot$. If $U(l)$ is not a set of literals, then we decompose the set of formulas depending on the category of the currently chosen formula. Then, we handle all γ-formulas separately, and we only mark a leaf with $\odot$ if the γ-formula step makes no change to the previous set of formulas.

We can now see why the predicate logic version of our method of semantic tableaux is not a decision procedure: our algorithm may choose to expand one branch of the tree infinitely, in which case it will never halt and produce a definite answer.

Branches of the tree thus must be handled differently, in terms of when a leaf is marked as either closed or open. We say that a branch is closed if and only if the branch ends in a leaf marked as closed; otherwise, the branch is open, regardless of whether the branch ends in a leaf marked as open or the branch is infinite.

Similarly, a tableau whose branches are all closed is a closed tableau; the tableau is open otherwise, regardless of whether it has a finite or infinite open branch.

We can thus conclude that the predicate logic version of the method of semantic tableaux can only be used to prove the validity of a formula A by showing that the semantic tableau for $\neg A$ is closed and, therefore, $\neg A$ is not satisfiable. Only in this case will the algorithm halt and produce a definite answer.

Since the predicate logic form of the method of semantic tableaux is more complex, drawing trees to represent a semantic tableau can understandably become a bit unwieldy. Thus, we can use an alternative approach to drawing a semantic tableau that makes use of a linear list format. Levels of the semantic tableau tree correspond to indentations of list entries, so that the leaves of the tree correspond to the most-deeply nested entries of the list. Moreover, for each list entry, we can label the entry by the formula rule used on that line.

Example 2. Consider the formula

$$(\exists x\ P(x) \Rightarrow \forall x\ Q(x)) \Rightarrow \forall x\ (P(x) \Rightarrow Q(x)).$$

In order to prove this formula is valid, we can use the method of semantic tableaux to show that the negation of this formula is unsatisfiable. We will express our semantic tableau in list format as follows:

$\neg((\exists x\ P(x) \Rightarrow \forall x\ Q(x)) \Rightarrow \forall x\ (P(x) \Rightarrow Q(x)))$

- $\exists x\ P(x) \Rightarrow \forall x\ Q(x),\ \neg\forall x\ (P(x) \Rightarrow Q(x))$ $(\alpha \Rightarrow)$
  - $\neg\exists x\ P(x),\ \neg\forall x\ (P(x) \Rightarrow Q(x))$ $(\beta \Rightarrow)$
    - $\neg\exists x\ P(x),\ \neg(P(a) \Rightarrow Q(a))$ (δ)
      - $\neg\exists x\ P(x),\ P(a),\ \neg Q(a)$ $(\alpha \Rightarrow)$
        - $\neg P(a),\ P(a),\ \neg Q(a)$ $(\gamma,\ \times)$
  - $\forall x\ Q(x),\ \neg\forall x\ (P(x) \Rightarrow Q(x))$ $(\beta \Rightarrow)$
    - $\forall x\ Q(x),\ \neg(P(a) \Rightarrow Q(a))$ (δ)
      - $\forall x\ Q(x),\ P(a),\ \neg Q(a)$ $(\alpha \Rightarrow)$
        - $Q(a),\ P(a),\ \neg Q(a)$ $(\gamma,\ \times)$

In all cases, the leaves (i.e., the most-deeply nested entries) of the semantic tableau are closed. Thus, the negated formula is unsatisfiable, and so the original formula is valid.

1.2 Tips and Pitfalls

Given the added complexity involved in constructing a semantic tableau for a predicate logic formula, we will review in this section a few helpful tips to keep in mind as well as a few common pitfalls to try and avoid. We will also see a number of other, smaller examples of semantic tableaux for each tip and pitfall.

Instantiating Quantified Formulas. Consider the formula

$$\forall x\ (P(x) \Rightarrow Q(x)) \Rightarrow (\forall x\ P(x) \Rightarrow \forall x\ Q(x)).$$

This is a valid formula, and we can prove this by constructing a semantic tableau for the negation of this formula. The first few lines of the semantic tableau are as follows:

$\neg(\forall x\ (P(x) \Rightarrow Q(x)) \Rightarrow (\forall x\ P(x) \Rightarrow \forall x\ Q(x)))$

- $\forall x\ (P(x) \Rightarrow Q(x)),\ \neg(\forall x\ P(x) \Rightarrow \forall x\ Q(x))$ $(\alpha \Rightarrow)$
  - $\forall x\ (P(x) \Rightarrow Q(x)),\ \forall x\ P(x),\ \neg\forall x\ Q(x)$ $(\alpha \Rightarrow)$
    - $\forall x\ (P(x) \Rightarrow Q(x)),\ \forall x\ P(x),\ \exists\neg x\ Q(x)$ (duality)

Focusing on the third formula in the set on the fourth line of the tableau, we know that this existentially quantified formula will be true if there exists an element $c \in D$ such that $c \in R_Q$, where R_Q is the relation assigned to the predicate Q by some interpretation. Suppose that a_1 is the element in question. We instantiate the formula with a_1 to get

$$\forall x\ (P(x) \Rightarrow Q(x)),\ \forall x\ P(x),\ \exists \neg x\ Q(x) \qquad \text{(duality)}$$
$$\forall x\ (P(x) \Rightarrow Q(x)),\ \forall x\ P(x),\ \neg Q(a_1) \qquad (\gamma)$$

Now, since the other two formulas in the set are each universally quantified, they will be true if they hold for all elements in the interpretation's domain. Notably, the formulas must hold for that element a_1 we chose earlier. Thus, we can instantiate the universally quantified formulas with the same element to get

$$\forall x\ (P(x) \Rightarrow Q(x)),\ \forall x\ P(x),\ \neg Q(a_1) \qquad (\gamma)$$
$$\forall x\ (P(x) \Rightarrow Q(x)),\ P(a_1),\ \neg Q(a_1) \qquad (\gamma)$$
$$(P(a_1) \Rightarrow Q(a_1)),\ P(a_1),\ \neg Q(a_1) \qquad (\gamma)$$

We can then apply the β-formula rule for $\Rightarrow$ to the first formula in this set to obtain a closed tableau.

Altogether, this example illustrates the need for us to instantiate existentially quantified formulas with some constant representing the element of the domain that satisfies that formula. We can then reuse that constant in all universally quantified formulas.

Remark. Be careful not to reuse constants when instantiating existentially quantified formulas! Even though an existentially quantified formula must hold for *some* element of the domain, there is no guarantee that two existentially quantified formulas hold for the *same* element. Whenever you instantiate an existentially quantified formula, you must use a new constant.

Preserving Universally Quantified Formulas. Recall that a universally quantified formula must hold for all elements of the domain. Consider the following intermediate steps of some proof tableau:

$$\ddots$$
$$\forall x\ (P(x) \vee Q(x)),\ \neg P(a_2),\ \neg Q(a_1) \qquad (\delta)$$
$$P(a_1) \vee Q(a_1),\ \neg P(a_2),\ \neg Q(a_1) \qquad (\gamma)$$

Prior to and on the first line, we instantiated two existentially quantified formulas with the constants a_1 and a_2, respectively. Then, on the second line, we instantiated the universally quantified formula with the constant a_1. However, this will not work, since we've now removed the ability to check whether the universally quantified formula holds for a_2 as well!

To fix this, we will simply never remove a universally quantified formula from a given line of the tableau. Each time we need to instantiate such a formula, we will make a "copy" of the formula that uses whatever constant we desire:

$$\ddots$$
$$\forall x\ (P(x) \vee Q(x)),\ \neg P(a_2),\ \neg Q(a_1) \qquad (\delta)$$
$$\forall x\ (P(x) \vee Q(x)),\ P(a_1) \vee Q(a_1),\ \neg P(a_2),\ \neg Q(a_1) \qquad (\gamma)$$
$$\forall x\ (P(x) \vee Q(x)),\ P(a_2) \vee Q(a_2),\ P(a_1) \vee Q(a_1),\ \neg P(a_2),\ \neg Q(a_1) \qquad (\gamma)$$

Identifying Non-Terminating Branches. Consider the formula $\forall x \exists y\ P(x, y)$. This is a universally quantified formula, since the universal quantifier $\forall x$ is the "outermost" quantifier. As we observed earlier, we only need to instantiate universally quantified formulas with constants that already appeared in formulas. Since we have no existing constants yet, how do we begin a semantic tableau for this formula?

Recall from our definition of an interpretation that we require the domain of an interpretation to be nonempty. Thus, we can choose an arbitrary constant to use in this first step; say, the constant a_1. Our semantic tableau then begins as follows:

$$\forall x \exists y\ P(x,y)$$

$$\forall x \exists y\ P(x,y),\ \exists y\ P(a_1,y) \quad (\gamma)$$

$$\forall x \exists y\ P(x,y),\ P(a_1,a_2) \quad (\delta)$$

Now, since we have introduced a new constant a_2 into our semantic tableau, we must instantiate the universally quantified formula with this constant:

$$\forall x \exists y\ P(x,y),\ P(a_1,a_2) \quad (\delta)$$

$$\forall x \exists y\ P(x,y),\ \exists y\ P(a_2,y),\ P(a_1,a_2) \quad (\gamma)$$

$$\forall x \exists y\ P(x,y),\ P(a_2,a_3),\ P(a_1,a_2) \quad (\delta)$$

Now, we've introduced another new constant a_3 into our semantic tableau, so we must instantiate the universally quantified formula again! We can see that this process will create another constant that we must then instantiate, and this results in us getting trapped within an infinite loop of instantiations. Thus, this branch of the semantic tableau will never terminate.

This example solidifies our earlier observation that the method of semantic tableaux for predicate logic is not a decision procedure, since we are no longer guaranteed to halt by marking each branch as either open or closed. Unfortunately, there's no easy way for us to determine whether a given branch will or will not terminate, as this essentially amounts to the classic halting problem. Thus, we simply need to be cautious as we construct our semantic tableaux.

1.3 Soundness and Completeness

Just as we did before, we conclude our study of the method of semantic tableaux by establishing both the soundness and the completeness of this method.

Theorem 3 (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.*

Our soundness and completeness proofs in the predicate logic case will be quite similar to those we produced in the propositional logic case, modulo the necessary changes we made earlier in this lecture.

1.3.1 Proving Soundness

To prove soundness, we must show that if the semantic tableau T for a formula A is closed, then A is unsatisfiable.

The bulk of our proof can be copied verbatim from our proof of the soundness of semantic tableaux for propositional logic. Here, we simply need to add two additional cases to the inductive case to account for the handling of γ-formulas and δ-formulas.

As before, we 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.

Proof of Soundness. We prove by induction on the height of the vertex n in T_n, denoted h_n. The proofs of the base case and of the inductive case for α-formulas and β-formulas are identical to those given in the propositional logic version. We therefore focus on the remaining formula types for the predicate logic version.

- **Case 3: γ-formulas.** In this case, n is labelled by $U(n) = U_0 \cup \{\forall x\ P(x)\}$ for some set of formulas U_0, and another vertex m with height $h_m < h_n$ is labelled by $U(m) = U_0 \cup \{\forall x\ P(x), P(a)\}$. By our inductive hypothesis, $U(m)$ is unsatisfiable.

We will prove that $U(n)$ is also unsatisfiable. To do this, assume by way of contradiction that $U(n)$ is satisfiable, and let $\mathscr{I}$ be an interpretation satisfying these formulas. Then we have that $v_{\mathscr{I}}(P_i) = \mathrm{T}$ for all $P_i \in U_0$ and $v_{\mathscr{I}}(\forall x\ P(x)) = \mathrm{T}$. However, $U(m) = U(n) \cup \{P(a)\}$, so showing that $v_{\mathscr{I}}(P(a)) = \mathrm{T}$ will contradict the inductive hypothesis claiming that $U(m)$ is unsatisfiable.

We know that $v_{\mathscr{I}}(\forall x\ P(x)) = \mathrm{T}$ if and only if $v_{\sigma_{\mathscr{I}}}(P(x)) = \mathrm{T}$ for all assignments $\sigma_{\mathscr{I}}$. Thus, this property must also hold for the assignment that assigns the same element of the domain to x that the interpretation $\mathscr{I}$ assigns to a. In this case, $v_{\mathscr{I}}(P(a)) = \mathrm{T}$, which leads to a contradiction.

- **Case 4: δ-formulas.** In this case, n is labelled by $U(n) = U_0 \cup \{\exists x\ P(x)\}$ for some set of formulas U_0, and another vertex m with height $h_m < h_n$ is labelled by $U(m) = U_0 \cup \{P(a)\}$. By our inductive hypothesis, $U(m)$ is unsatisfiable.

  We will prove that $U(n)$ is also unsatisfiable. To do this, assume by way of contradiction that $U(n)$ is satisfiable, and let $\mathscr{I} = (D, \{R_1, \ldots, R_n\}, \{d_1, \ldots, d_k\})$ be an interpretation satisfying these formulas.

  We know that $v_{\mathscr{I}}(\exists x\ P(x)) = \mathrm{T}$ if and only if $v_{\sigma_{\mathscr{I}}}(P(x)) = \mathrm{T}$ for some assignment $\sigma_{\mathscr{I}}$; that is, if and only if $\sigma_{\mathscr{I}}(x) = d$ for some domain element $d \in D$. Suppose that we extend our interpretation $\mathscr{I}$ to assign this domain element d to the constant a. This produces the interpretation $\mathscr{I}' = (D, \{R_1, \ldots, R_n\}, \{d_1, \ldots, d_k, d\})$. We can do this, since a does not occur in $U(n)$ and it is therefore not among the constants $\{a_1, \ldots, a_k\}$ that have already been assigned elements $\{d_1, \ldots, d_k\}$.

  By doing this, and from our assumption that $v_{\mathscr{I}'}(U_0) = v_{\mathscr{I}}(U_0) = \mathrm{T}$, we get that $v_{\mathscr{I}'}(P(a)) = \mathrm{T}$, which leads to a contradiction.

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

1.3.2 Proving Completeness

By the statement of our theorem, completeness implies that if A is an unsatisfiable formula, then the semantic tableau for A is closed. Instead of proving this part of the theorem using this exact wording, we will modify our statement slightly to frame the given formula A in terms of validity. The modified (but equivalent) claim we will use states that if A is a valid formula, then the semantic tableau for $\neg A$ is closed.

First, we will state a rather technical lemma, which we will use later but not prove here.

Lemma 4. *Let b be an open branch of a semantic tableau T, let n be a vertex appearing on the branch b, and let A be some formula appearing in the set $U(n)$. Then some rule will be applied to A either at vertex n, or at some vertex m that is a descendent of n along b. Furthermore, if A is a γ-formula and c is a constant appearing at n, then $\gamma(c) \in U(m')$, where m' is the descendent vertex created by applying the appropriate rule at m.*

Proof. Omitted. □

Just as in the propositional logic proof, we require the notion of a *Hintikka set*. But, naturally, we must augment our previous definition to now take into account the existence of both γ- and δ-formulas.

Definition 5 (Hintikka set)**.** Let U be a set of formulas. We say that U is a Hintikka set if and only if the following conditions hold for all formulas $A \in U$:

1. if A is a literal, then either $A \in U$ or $\neg A \in U$;
2. if A is an α-formula, then both $\alpha_1 \in U$ and $\alpha_2 \in U$;
3. if A is a β-formula, then either $\beta_1 \in U$ or $\beta_2 \in U$;
4. if A is a γ-formula, then $\gamma(c) \in U$ for all constants c appearing in formulas in U; and
5. if A is a δ-formula, then $\delta(c) \in U$ for some constant c.

Now, using our technical lemma, we can characterize when sets of formulas in a semantic tableau constitute a Hintikka set.

Theorem 6. *Let b be an open branch, either finite or infinite, of a semantic tableau T, and let $U = \bigcup_{n \in b} U(n)$ be all sets of formulas at vertices along this branch. Then U is a Hintikka set.*

Proof. Let A be a formula in U. We will show by a case-based analysis that U meets the criteria to be a Hintikka set.

First, suppose that A is a literal. By the semantic tableau construction process, once a literal appears along the branch b, it is never removed. Therefore, if A appears at a vertex n and $\neg A$ appears at a vertex m that is a descendent of n, then A must also appear at m. However, by our assumption, the branch b is open. Thus, either $A \notin U$ or $\neg A \notin U$, and the first Hintikka set condition holds.

Next, suppose that A is neither an atomic formula nor a γ-formula. Then, by Lemma 4, some rule will eventually be applied to A, and the second/third/fifth Hintikka set conditions hold.

Lastly, suppose that A is a γ-formula that first appears at vertex n. Let c be a constant that first appears at some descendent vertex m, and take $k = \max\{n, m\}$. By the semantic tableau construction process, the set of γ-formulas and the set of constants do not decrease in size as we follow a branch, so both A and c appear at vertex k. By Lemma 4, we have that $\gamma(c) \in U(k')$ for some descendent vertex $k' > k$, and $U(k') \subseteq U$. □

Given a Hintikka set, we can then conclude that there must exist some satisfying interpretation for that set. Recall that this is essentially Hintikka's lemma, but restated for the predicate logic case. We will omit the proof here, but it amounts to explicitly defining such an interpretation based on the contents of the set U.

Lemma 7 (Hintikka's lemma—restated)**.** *Let U be a Hintikka set. Then there exists a satisfying interpretation for U that may be either finite or infinite.*

Proof. Omitted. □

Combining each of the previous results, we arrive at our proof of completeness.

Proof of Completeness. Let A be a valid formula, and suppose that the semantic tableau for $\neg A$ is not closed. By the definition of an open tableau, there must exist an open branch b within the tableau, which may be either finite or infinite. By Theorem 6, the set $U = \bigcup_{n \in b} U(n)$ is a Hintikka set, and so by Lemma 7, there exists a satisfying interpretation $\mathscr{I}$ for U. However, $\neg A \in U$ implies that $\mathscr{I} \vDash \neg A$, which contradicts the fact that we took A to be valid. □