Lecture notes 6.0

Semantic tableaux

COMP 2411, session 1, 2004

Introduction

To check whether a formula φ is valid, we can build a truth table for φ.

This is however a very inefficient method since the number of rows is exponential in the number of atomic formulas in φ.

Moreover, the truth table method has no counterpart in the predicate calculus.

The method of semantic tableaux is a proof procedure that is:

- easy to understand;
- efficient in many cases;
- can be generalized to the predicate calculus.

Principle (1)

The basic idea behind the method is: try and find a model of φ in a systematic way, by breaking down formulas into smaller formulas, until we are left with nothing but atomic formulas and/or negations of atomic formulas.

Definition: A literal is an atomic formula or the negation of an atomic formula.

We can easily infer whether a set S of literals has a model or not:

- either S contains an atom and its negation (*e.g.*, $S = \{p, q, \neg r, \neg q\}$), in which case S has no model,
- or S never contains an atom and its negation (*e.g.*, $S = \{p, q, \neg r, \neg s\}$), in which case S has a model.

Principle (2)

When we break down a formula φ into smaller formulas, two cases can happen.

- Case 1: to make φ true, we have to make two smaller formulas φ_1 and φ_2 true. For instance, to make $(p \vee \neg q) \wedge (q \rightarrow s)$ true, we have to make $p \vee \neg q$ true and we have to make $q \rightarrow s$ true.
- Case 2: to make φ true, we have to make at least one of two smaller formulas φ_1 and φ_2 true. For instance, to make $(p \vee \neg q) \vee (q \rightarrow s)$ true, we can make $p \vee \neg q$ true or we can make $q \rightarrow s$ true.

The algorithm has two kinds of rules: α-rules for the first case, and β-rules for the second case.

Example 1

Suppose that we want to make $\varphi = (p \vee q) \wedge (\neg p \wedge \neg q)$ true. Then we have to make both $p \vee q$ and $\neg p \wedge \neg q$ true.

To make $p \vee q$ true, we can either make p true or make q true. Hence to make φ true, we can either:

- make p and $\neg p \wedge \neg q$ true, or
- make q and $\neg p \wedge \neg q$ true.

Now to make $\neg p \wedge \neg q$ true, we have to make both $\neg p$ and $\neg q$ true. Hence to make φ true, we can either:

- make p, $\neg p$ and $\neg q$ true, or
- make q, $\neg p$ and $\neg q$ true.

Since neither $\{p, \neg p, \neg q\}$ nor $\{q, \neg p, \neg q\}$ has a model, we conclude that φ has no model.

Example 2

Suppose that we want to make $\varphi = (p \vee q) \wedge (\neg p \wedge \neg s)$ true. Then we have to make both $p \vee q$ and $\neg p \wedge \neg s$ true.

To make $p \vee q$ true, we can either make p true or make q true. Hence to make φ true, we can either:

- make p and $\neg p \wedge \neg s$ true, or
- make q and $\neg p \wedge \neg s$ true.

Now to make $\neg p \wedge \neg s$ true, we have to make both $\neg p$ and $\neg s$ true. Hence to make φ true, we can either:

- make p, $\neg p$ and $\neg s$ true, or
- make q, $\neg p$ and $\neg s$ true.

Since $\{q, \neg p, \neg s\}$ does have a model, we conclude that φ has a model.

Tree representation

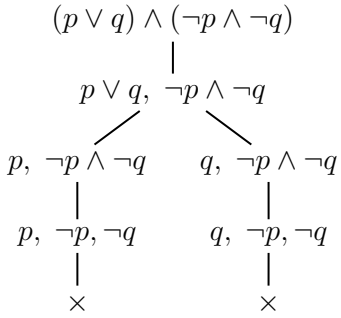
The tableau construction can be represented by a labeled tree where:

- the root is labeled with the initial formula φ;
- each internal node has one child or two children, depending on whether an α-rule or a β-rule is applied;
- each node different from the root is labeled with a sequence of formulas;
- each formula in the sequence of formulas that label the leaf is a literal.

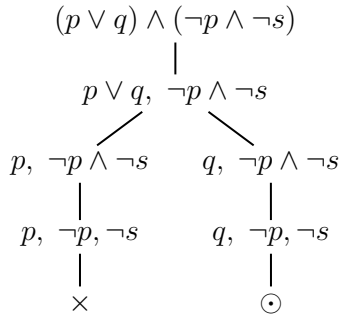
A branch is marked as $\odot$ (open) if the literals that label its leaf have a model; it is marked as $\times$ (closed) otherwise.

φ has a model iff at least one branch is marked as open.

Example 1 again



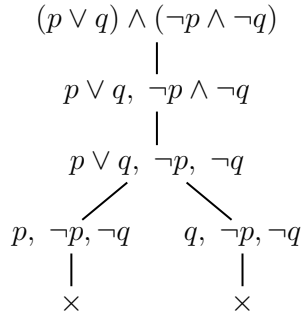
Example 2 again



Nonuniqueness

It is usually possible to build different semantic tableaux for the same formula, by picking up different formulas in the sequences of formulas that label a node.

The following is an alternative (and better) tableau for the formula of Example 1.



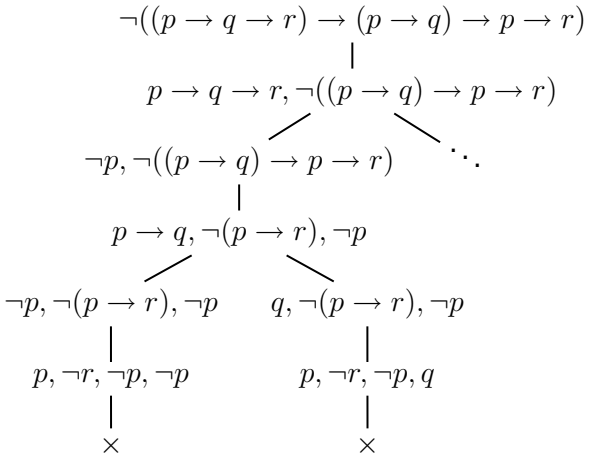
α-rules

α	α_1	α_2
$\neg\neg\varphi$	φ	
$\varphi_1 \wedge \varphi_2$	φ_1	φ_2
$\neg(\varphi_1 \vee \varphi_2)$	$\neg\varphi_1$	$\neg\varphi_2$
$\neg(\varphi_1 \rightarrow \varphi_2)$	φ_1	$\neg\varphi_2$
$\neg(\varphi \uparrow \varphi_2)$	φ_1	φ_2
$\varphi \downarrow \varphi_2$	$\neg\varphi_1$	$\neg\varphi_2$
$\varphi_1 \leftrightarrow \varphi_2$	$\varphi_1 \rightarrow \varphi_2$	$\varphi_2 \rightarrow \varphi_1$
$\neg(\varphi_1 \odot \varphi_2)$	$\varphi_1 \rightarrow \varphi_2$	$\varphi_2 \rightarrow \varphi_1$

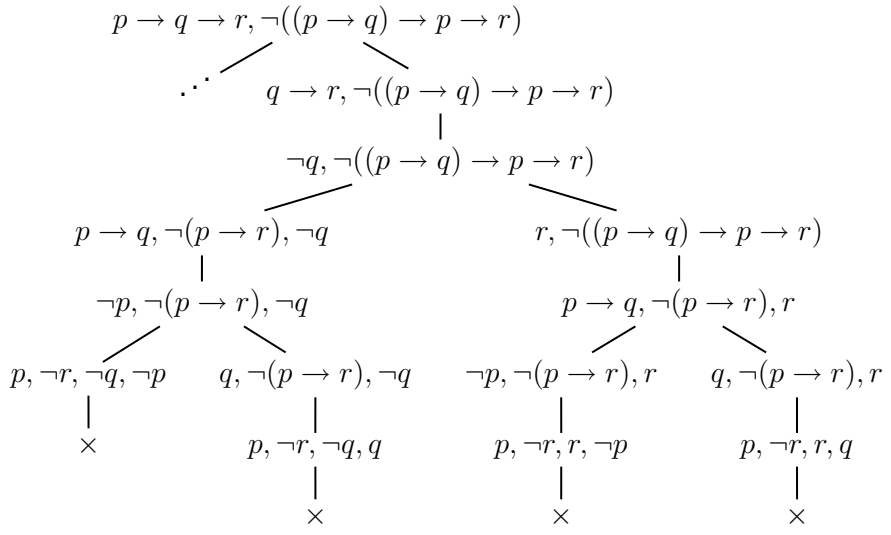
β-rules

β	β_1	β_2
$\neg(\varphi_1 \wedge \varphi_2)$	$\neg\varphi_1$	$\neg\varphi_2$
$\varphi_1 \vee \varphi_2$	φ_1	φ_2
$\varphi_1 \rightarrow \varphi_2$	$\neg\varphi_1$	φ_2
$\varphi \uparrow \varphi_2$	$\neg\varphi_1$	$\neg\varphi_2$
$\neg(\varphi \downarrow \varphi_2)$	φ_1	φ_2
$\neg(\varphi_1 \leftrightarrow \varphi_2)$	$\neg(\varphi_1 \rightarrow \varphi_2)$	$\neg(\varphi_2 \rightarrow \varphi_1)$
$\neg\varphi \odot \varphi_2$	$\neg(\varphi_1 \rightarrow \varphi_2)$	$\neg(\varphi_2 \rightarrow \varphi_1)$

A longer example...



...A longer example



General construction (1)

To build a semantic tableau $\mathcal{T}$ for a propositional formula φ, we start with a tree having just a root labeled with φ.

At each stage in the construction, we choose an unmarked leaf L labeled with a sequence of formulas $\varphi_1, \ldots, \varphi_n$.

- If $\{\varphi_1, \ldots, \varphi_n\}$ is a set of literals that contains some atom and its negation, we extend $\mathcal{T}$, giving a child to L (L becomes an internal node) labeled with $\times$.
- If $\{\varphi_1, \ldots, \varphi_n\}$ is a set of literals that contains no atom together with its negation, we extend $\mathcal{T}$, giving a child to L (L becomes an internal node) labeled with $\odot$.

General construction (2)

- Otherwise, we select a nonliteral $\varphi_i \in \{\varphi_1, \ldots, \varphi_n\}$.
  - If φ_i is an α-formula, having φ_i^1 and φ_i^2 as corresponding α_1 and α_2 formulas in the α-table, we extend $\mathcal{T}$, giving a child to L labeled with

    $$\varphi_1, \ldots, \varphi_{i-1}, \varphi_i^1, \varphi_i^2, \varphi_{i+1}, \ldots, \varphi_n.$$

  - If φ_i is a β-formula, having φ_i^1 and φ_i^2 as corresponding β_1 and β_2 formulas in the β-table, we extend $\mathcal{T}$, giving two children to L, one labeled with

    $$\varphi_1, \ldots, \varphi_{i-1}, \varphi_i^1, \varphi_{i+1}, \ldots, \varphi_n,$$

    the other labeled with

    $$\varphi_1, \ldots, \varphi_{i-1}, \varphi_i^2, \varphi_{i+1}, \ldots, \varphi_n.$$

General construction (3)

Definition: A tableau whose construction has terminated is called a completed tableau.

A completed tableau is closed if all leaves are marked as closed ($\times$).

A completed tableau is open if some leaf is marked as open ($\odot$).

Proposition: The construction of a semantic tableau terminates.

Soundness and completeness (1)

The tableau construction is a sound and complete proof procedure for unsatisfiability:

- it is algorithmic (computable, effective)
- it is sound because it is always right when it tells us that a formula is unsatisfiable.
- it is complete because it always tells us when a formula is unsatisfiable.

Moreover, since it always terminates, it is a decidable proof procedure.

All these notions are captured by the following result.

Proposition $(\star)$: Let $\mathcal{T}$ be a completed tableau for a formula φ. Then φ is unsatisfiable iff $\mathcal{T}$ is closed.

Soundness and completeness (2)

Since a formula is unsatisfiable iff its negation is valid, we also have a sound, complete, and decidable proof procedure for validity.

This is captured by the next Corollary to Proposition $(\star)$:

Corollary: Let a formula φ be given. Let $\mathcal{T}$ be a completed tableau for $\neg\varphi$. Then φ is valid iff $\mathcal{T}$ is closed.

We also have a sound, complete, and decidable proof procedure for logical consequence:

Corollary: Let formulas $\psi_1, \ldots, \psi_n, \varphi$ be given. Let $\mathcal{T}$ be a completed tableau for $\psi_1 \wedge \ldots \wedge \psi_n \wedge \neg\varphi$. Then φ is a logical consequence of $\{\psi_1, \ldots, \psi_n\}$ iff $\mathcal{T}$ is closed.

Soundness

Soundness is the following half of Proposition $(\star)$.

Proposition: Let a formula φ and a completed semantic tableau $\mathcal{T}$ for φ be given. If $\mathcal{T}$ is closed then φ is unsatisfiable.

To prove soundness, it suffices to show that if a subtree of $\mathcal{T}$ rooted at a node labeled with $\varphi_1, \ldots, \varphi_n$ is closed, then $\{\varphi_1, \ldots, \varphi_n\}$ has no model.

The proof of the previous statement is easy by induction on the height of the subtrees of $\mathcal{T}$.

Completeness (1)

Completeness is the following half of Proposition $(\star)$.

Proposition: Let a formula φ and a completed semantic tableau $\mathcal{T}$ for φ be given. If φ is unsatisfiable then $\mathcal{T}$ is closed.

To prove completeness, we prove the contrapositive: if $\mathcal{T}$ is open then φ is satisfiable.

More precisely, we show that if $\mathcal{B}$ is an open branch in $\mathcal{T}$ (*e.g.*, a branch whose leaf is labeled with $\odot$)), then set of literals that label the parent of the leaf determines a model of φ.

Hintikka sets

The key notion for proving completeness is the following.

Definition: A Hintikka set is a set of formulas U such that:

- U does not contain an atom together with its negation.
- For all α-formulas in U, U contains the corresponding α_1 and α_2 formulas in the α-table.
- For all β-formulas in U, U contains one of the corresponding β_1 and β_2 formulas in the β-table.

Completeness (2)

It is then easy to show the following.

- Every Hintikka set has a model.
- The set S of formulas that label the nodes of an open branch $\mathcal{B}$ of $\mathcal{T}$ is a Hintikka set.

Since S contains φ (φ is the label of the root of $\mathcal{T}$, hence is also the label of the root of $\mathcal{B}$), we conclude that φ is satisfiable.

Good strategies

The construction of a semantic tableau is made more efficient if:

- α-rules are applied before β-rules;
- we close a branch as soon as a leaf is labeled with a formula and its negation, even if that formula is not an atom.

This strategy is implemented in the Prolog programs with the procedure `extend_systematic_tableau`, as opposed to the procedure `extend_tableau`.