Lecture notes 16.0

Semantic tableaux

COMP 2411, session 1, 2004

Introduction

Here we work in the predicate calculus without equality.

We want to generalize the construction of semantic tableaux from propositional logic to first-order logic, applying the same basic idea:

> In order to show that φ is satisfiable, 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 that do not clash.

We know what to do with the boolean operators. The only issue is what to do with the quantifiers.

Which rules for the quantifiers? (1)

Intuitively:

- if we want to make $\exists x\varphi$ true, then we have to make sure that some individual has property φ;
- if we want to make $\forall x\varphi$ true, then we have to make sure that all individuals have property φ;
- if we want to make $\neg\exists x\varphi$ true, then we have to make sure that all individuals have property $\neg\varphi$;
- if we want to make $\neg\forall x\varphi$ true, then we have to make sure that some individual has property φ.

The first and fourth rules remind of the β-rules, but with more than two branches (infinitely many?)

The second and third rules remind of the α-rules, but adding more than two formulas (infinitely many?)

Which rules for the quantifiers? (2)

Moreover, how shall we denote the potential individuals?

If we want to make φ true, φ will be true in a structure $\mathfrak{M}$ and in our language (given by the chosen vocabulary V):

- some individuals can have many names;
- some individuals can have more than one name.

We do not care about individuals having many names because we have excluded equality. Hence we cannot differentiate between:

- an individual have two names a and b;
- two individuals having respective names a and b and the same properties expressible in the formal language.

In other words, we can assume that different names denote different individuals.

Which rules for the quantifiers? (3)

When we want to make sure that some individual has a given property, we choose a new constant meant to represent a new individual.

When we want to make sure that all individuals have a given property, we have to consider all constants introduced now or to be introduced later.

This means that we will not apply the rules for a universal quantifier or for the negation of an existential quantifier once and for all, but again and again, to take care of the new constants that are being introduced as the tableau construction proceeds.

Before we define the method in full generality, we give a few examples.

$\exists x \forall y p(x, y) \rightarrow \forall y \exists x p(x, y)$ is valid

$$\neg(\exists x \forall y p(x, y) \rightarrow \forall y \exists x p(x, y))$$
$$\exists x \forall y p(x, y), \neg \forall y \exists x p(x, y)$$
$$\forall y p(a1, y), \neg \forall y \exists x p(x, y)$$
$$\neg \exists x p(x, a2), \forall y p(a1, y)$$
$$\neg p(a2, a2), \neg p(a1, a2), \forall y p(a1, y), \neg \exists x p(x, a2)$$
$$p(a1, a2), p(a1, a1), \neg p(a2, a2), \neg p(a1, a2), \neg \exists x p(x, a2), \forall y p(a1, y)$$
$$\times$$

$\exists x \exists y p(x, y) \rightarrow \exists y \exists x p(y, x)$ is valid

$$\neg(\exists x \exists y p(x, y) \rightarrow \exists y \exists x p(y, x))$$
$$\exists x \exists y p(x, y), \neg \exists y \exists x p(y, x)$$
$$\exists y p(a1, y), \neg \exists y \exists x p(y, x)$$
$$p(a1, a2), \neg \exists y \exists x p(y, x)$$
$$\neg \exists x p(a2, x), \neg \exists x p(a1, x), p(a1, a2), \neg \exists y \exists x p(y, x)$$
$$\neg p(a2, a2), \neg p(a2, a1), \neg \exists x p(a1, x), p(a1, a2), \neg \exists y \exists x p(y, x), \neg \exists x p(a2, x)$$
$$\neg p(a1, a2), \neg p(a1, a1), \neg p(a2, a2), \neg p(a2, a1), p(a1, a2), \neg \exists y \exists x p(y, x), \neg \exists x p(a2, x), \neg \exists x p(a1, x)$$
$$\times$$

$\forall x(p(x) \rightarrow q(x)) \rightarrow \forall x p(x) \rightarrow \forall x q(x)$ is valid

$$\neg(\forall x(p(x) \rightarrow q(x)) \rightarrow \forall x p(x) \rightarrow \forall x q(x))$$
$$\forall x(p(x) \rightarrow q(x)), \neg(\forall x p(x) \rightarrow \forall x q(x))$$
$$\forall x p(x), \neg \forall x q(x), \forall x(p(x) \rightarrow q(x))$$
$$\neg q(a1), \forall x p(x), \forall x(p(x) \rightarrow q(x))$$
$$p(a1), \neg q(a1), \forall x(p(x) \rightarrow q(x)), \forall x p(x)$$
$$p(a1) \rightarrow q(a1), p(a1), \neg q(a1), \forall x p(x), \forall x(p(x) \rightarrow q(x))$$
$$\neg p(a1), p(a1), \neg q(a1), \forall x p(x), \forall x(p(x) \rightarrow q(x)) \qquad q(a1), p(a1), \neg q(a1), \forall x p(x), \forall x(p(x) \rightarrow q(x))$$
$$\times \qquad\qquad \times$$

$\forall x p(x) \rightarrow \exists x p(x)$ is valid

$$\begin{array}{c}
\neg(\forall x p(x) \rightarrow \exists x p(x)) \\
| \\
\forall x p(x), \neg\exists x p(x) \\
| \\
p(a1), \neg\exists x p(x), \forall x p(x) \\
| \\
\neg p(a1), p(a1), \forall x p(x), \neg\exists x p(x) \\
| \\
\times
\end{array}$$

γ and δ rules

To the α and β rules of propositional rules, we add two new kinds of rules for the quantifiers. The rules are determined by instances of the formula to which the rule is applied, removing the quantifier and replacing the quantified variable by a constant.

γ	$\gamma(a)$
$\forall x\varphi$	$\varphi[a/x]$
$\neg\exists x\varphi$	$\neg\varphi[a/x]$

δ	$\delta(a)$
$\exists x\varphi$	$\varphi[a/x]$
$\neg\forall x\varphi$	$\neg\varphi[a/x]$

General construction (1)

To build a semantic tableau $\mathcal{T}$ for a closed first-order 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 formulas that contains some formula 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, i.e., atomic formulas (of the form $p(t_1, \ldots, t_n)$ for some n-ary predicate symbol p) or negation of atomic formula, we extend $\mathcal{T}$, giving a child to L (L becomes an internal node) labeled with $\odot$.

General construction (2)

- Otherwise, we select a $\varphi_i \in \{\varphi_1, \ldots, \varphi_n\}$.
  - 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)

- Continuing the previous case analysis on φ_i:
  - If φ_i is formula of the δ kind, we extend $\mathcal{T}$, giving a child to L labeled with

    $$\varphi_1, \ldots, \varphi_{i-1}, \delta(a), \varphi_{i+1}, \ldots, \varphi_n$$

    where a is a new contant.
  - If φ_i is formula of the γ kind, we extend $\mathcal{T}$, giving a child to L labeled with

    $$\varphi_1, \ldots, \varphi_{i-1}, \varphi_i, \gamma(a), \varphi_{i+1}, \ldots, \varphi_n$$

    where a is a constant that has already been introduced by a δ rule before, unless no such rule has been used before in which case a is a new constant. Note that φ_i is kept.

Termination

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

A completed tableau is closed if all leaves are marked as closed ($\times$). It is open if some leaf is marked as open ($\odot$).

In contrast to what happens in propositional logic, the tableau construction is not guaranteed to terminate.

- If the formula φ that labels the root is unsatisfiable, in which case the construction is guaranteed to terminate and the tableau is closed.
- If the formula φ that labels the root is satisfiable then
  - either the construction is guaranteed to terminate and the tableau is open, or
  - the construction does not terminate.

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.

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

But since the construction does not always terminates, it is an undecidable proof procedure.

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 and complete 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.

Optimizations

The program `First_order/Evaluations/tableau.pl` implements the following strategy:

- if possible, apply an α rule before a β rule.
- If possible, apply a β rule before a δ rule.
- If possible, apply a δ rule before a γ rule.
- When applying a γ rule, add all instances of the formula being dealt with (using all constants introduced so far, or a new one in case no constant has been introduced, like in the proof of the validity of $\forall x p(x) \rightarrow \exists x p(x)$ above).