1 Introduction

Remark 25.1.1 We now describe a system for building proofs of sentences in predicate logic. As with propositional logic proofs will be labeled binary branching trees, whose nodes are labeled by first-order formulae. We will also take over the form of the rules of tableaux construction for the propositional connectives, and once again it will be true that if a formulae at a node is to be true, then at least one of the immediate successors *must be true* as well. We will add four additional rules for the two quantifiers, although come in two types: one type of rule is "universal" in character and the other type of rule is "existential" in character. The motivation behind the rules is that if a formula on the tableau is true then it is possible to also make one of its immediate successors true as well. I did not say that an immediate successor *must be true*, and this is one of the subtleties with the quantifier rules.

The tableau method is based on an attempt to construct a counterexample to a formula. So, the tableau begins with $\neg\,\alpha$ for some sentence α in a language $\mathcal{L}$. A counterexample is now a structure $\mathcal{A}$, which means we must specify (i) a domain, (ii) an interpretation of each constant in $\mathcal{L}$ and (iii) an interpretation of each predicate in $\mathcal{L}$. (By Lemma 22.2.3 it is sufficient to provide an interpretation for the symbols actually occurring in α.) Lets start with the domain. We begin the construction with a countably infinite set of *new constants* $A = \{a_0, a_1, \ldots\}$ to the language $\mathcal{L}$, and so will be working in the expanded language $\mathcal{L} \cup A$, denoted by $\mathcal{L}_A$. We call these new constants *parameters* since we start with no committments about them.

Suppose at some point in our construction we come to an existential formula $(\exists x)\phi(x)$ on a node in the tableau. We want to try to make this true, so we need some element a in our domain such that $\phi(a)$ will be true. Here we must be careful: we must make no commitment about the properties of a, *except* that $\phi(a)$ will be true of a. But we will have no commitments about any parameter not yet introduced into the construction, so our rule will allow us to introduce $\phi(a)$ on the path – provided that the parameter a has not yet appeared on the path. (We have an infinite supply of parameters, and a tableaux under construction is finite, so there are plenty of unused parameters lying around.) Now that we are committed to $\phi(a)$ being true, we will ensure $(\exists x)\phi(x)$ is true, so there is nothing further to do and we mark it off as finished.

Suppose at some point in our construction we come to a universal formula $(\forall x)\phi(x)$ on a node of the tree. Since $\phi(a)$ must be true for any element of our domain, there is never a problem with substituting *any term*. So, our rule allows us to extend any path on which $(\forall x)\phi(x)$ with $\mathbf{T}\phi(t)$ for any constant or parameter t. However, we are not yet through with $\mathbf{T}(\forall x)\phi(x)$ because if we are to really construct a counterexample then we must instantiate $(\forall x)\phi(x)$ for every constant that is in $\mathcal{L}$ or appears as a parameter at some point along the path containing $(\forall x)\phi(x)$. So, we can never truly be sure we are finished with a true universally quantified sentence, until a contradiction appear. The way we handle this in the rules of semantic tableaux is that we always add the instance $\phi(a)$ to a branch *as well as* $(\forall x)\phi(x)$ again to the path.

Aside. $(\forall x)\phi(x)$ and $\neg\,(\exists x)\phi(x)$ are the source of difficulties for any attempt to automate proofs using tableau (or any other system). The other rules are basically one-and-done, you could prematurely branch, but eventually you will reach the end. There are very sophisticated heuristics for choosing how to choose terms to instantiate in these two cases, but there will always be limitations to these methods.

2 The Tableau Method for Predicate Logic

Remark 25.2.1 Below are the quantifier truth conditions:

$$v_{\mathcal{A}}((\exists x)\phi) = \begin{cases} \mathbf{T} & \text{if } v_{\mathcal{A}}(\phi_a^x) = \mathbf{T} \text{ for at least one } a \in A, \\ \mathbf{F} & \text{if } v_{\mathcal{A}}(\phi_a^x) = \mathbf{F} \text{ for all } a \in \mathcal{A}; \end{cases}$$

$$v_{\mathcal{A}}((\forall x)\phi) = \begin{cases} \mathbf{T} & \text{if } v_{\mathcal{A}}(\phi_a^x) = \mathbf{T} \text{ for all } a \in \mathcal{A}, \\ \mathbf{F} & \text{if } v_{\mathcal{A}}(\phi_a^x) = \mathbf{F} \text{ for at least one } a \in A; \end{cases}$$

Definition 25.2.2 (quantifier rules for tableaux) Here are the new rules for treating quantified formulae and negations of quantified formulae. Recall, that a ground term is a constant in the original language $\mathcal{L}$ together with the new parameters $a_1, a_2, \ldots$ we are using in constructing the domain.

$$\begin{array}{c} (\forall x)\phi(x) \\ | \\ \phi(t) \\ (\forall x)\phi(x) \end{array}$$
where t is a ground term

$$\begin{array}{c} \neg\,(\forall x)\phi(x) \\ | \\ \neg\,\phi(a) \end{array}$$
where a is a new parameter

$$\begin{array}{c} (\exists x)\phi(x) \\ | \\ \phi(a) \end{array}$$
where a is a new parameter

$$\begin{array}{c} \neg\,(\exists x)\phi(x) \\ | \\ \neg\,\phi(t) \\ \neg\,(\exists x)\phi(x) \end{array}$$
where t is ground term

We can unify the rules and simplify notation. There are four types of tableaux rules based on the type of formulae. Type-A and type-B are the same as before. We add the two new types of for first-order formulae which act existentially (existential and negated universal) and those which act universally (universal and negated existential).

type-C (universal)	
C	$C(a)$
$(\forall x)\phi(x)$	$\phi(t)$
$\neg\,(\exists x)\phi(x)$	$\neg\,\phi(t)$

type-D (existential)	
D	$D(a)$
$(\exists x)\phi(x)$	$\phi(a)$
$\neg\,(\forall x)\phi(x)$	$\neg\,\phi(a)$

$$\begin{array}{c} A \\ | \\ A_1 \\ A_2 \end{array} \qquad \begin{array}{c} B \\ \swarrow\ \searrow \\ B_1 \quad B_2 \end{array} \qquad \begin{array}{c} C \\ | \\ C(t) \\ C \end{array}$$
where t is ground term

$$\begin{array}{c} D \\ | \\ D(a) \end{array}$$
where a is a new parameter

In the last two rules we are replacing a quantified variable by a term, which may be of any of the forms $(\forall x), (\exists x), \neg\,(\forall x), \neg\,(\exists x)$.

Remark 25.2.3 In most situations in predicate logic we do not aim to prove a sentence outright. Instead, one normally proves a sentence based on various assumptions or axioms. The semantics embodied this in the notion of *logical consequence*. To capture the corresponding proof theoretic notion of a deduction from a set Σ of assumptions or axioms, we need to modify the rules of proof to allow that we are assuming the sentences of Σ are true. For the remainder of this section, Σ will be a set of sentences from the language $\mathcal{L}$

(not the expanded language $\mathcal{L}_A$ of the tableaux.)

From now on we are going to define all notions for tableaux from a set of premises Σ. If we are proving a sentence with no assumptions, Σ will be empty and the additional clause (f) below to the construction of tableaux will not be required.

Definition 25.2.4 (Tableaux for predicate logic) Let Σ be a set of sentences from $\mathcal{L}$. A *finite tableau from* Σ is a binary tree labeled with formulae (of $\mathcal{L}_A$), which satisfies the following inductive definition:

(a) All one-node trees labeled with a formula are finite tableaux from Σ.

(b) If τ is a finite tableau from Σ, π a path through τ, and A on π, then the extension placing the components A_1 and the A_2 on π is also a finite tableau from Σ.

(c) If τ is a finite tableau from Σ, π a path through τ, and $\mathcal{B}$ on π, then the extension of π placing the component B_1 on the left branch and the B_2 on the right branch is also a finite tableau from Σ.

(d) If τ is a finite tableau from Σ, π a path through τ, and C on π, then the extension placing the component C_1 and C on π is also a finite tableaux from Σ.

(e) If τ is a finite tableau, π a path through τ, and D on π, then the extension placing the component D_1 on π is also a finite tableaux. from Σ

(f) If τ is a finite tableaux from Σ, ϕ a sentence from Σ, then the extension of τ where ϕ is placed on each path π through τ (or any collection of paths through τ) is also a finite tableaux from Σ.

(g) If $\tau_0, \tau_1, \ldots, \tau_n, \ldots$ is a (finite or infinite) sequence of finite tableaux from Σ such that for each $n \geq 0$, τ_{n+1} is constructed from τ_n by an application of either (b), (c), (d), (e) or (f), then $\tau = \cup_n \tau$ is a *tableau from* Σ

Warning. It is crucial that the application of (d) repeats the type-C formulae in the application of the rule. You may simplify your tableaux constructions by not repeating the type-C formulae, provided you are aware that an application of clause (d) never finishes the formula.

Definition 25.2.5 (tableau proofs from Σ) Let τ be a tableau and π a path through τ.

- π is *contradictory* if, for some sentence ϕ, both ϕ and $\neg\,\phi$ appear on π.
- τ is *contradictory* if every path on τ is contradictory.
- τ is a proof of α from Σ if τ is a (finite) contradictory tableau from Σ with its root node labeled $\neg\,\alpha$. If there is a proof τ of α from Σ, the we say α is *provable* from Σ, denoted by $\Sigma \vdash \alpha$.
- Σ is *inconsistent* if there is a proof of $\bot$ from Σ.

Note that we are not *requiring* proofs to be finite tableaux; however, since a tableau is finite branching we know from König's Lemma that if τ is an infinite contradictory tableau, then some finite subset τ' is a contradictory finite tableau. (There is nothing in the rules that prevents a contradictory path from being extended.)

Example 25.2.6 Lets check the validity of $((\forall x)\phi(x) \to (\exists x)\phi(x))$.

$$\begin{array}{c}
\neg\,((\forall x)\phi(x) \to (\exists x)\phi(x)) \\
(\forall x)\phi(x) \\
\neg\,(\exists x)\phi(x) \\
\phi(a) \\
(\forall x)\phi(x) \\
\neg\,\phi(a) \\
\neg\,(\exists x)\phi(x) \\
\otimes
\end{array}$$

$$\neg\big(((\forall x)P(x) \to Q(x)) \to ((\forall x)P(x) \to (\forall x)Q(x))\big)$$
$$((\forall x)P(x) \to Q(x))$$
$$\neg\,((\forall x)P(x) \to (\forall x)Q(x))$$
$$(\forall x)P(x)$$
$$\neg\,(\forall x)Q(x)\ \neg Q(a)\ \ {\scriptstyle \text{new } a}$$
$$P(a)$$
$$(\forall x)P(x)$$
$$(P(c) \to Q(a))$$
$$((\forall x)P(x) \to Q(x))$$
$$\neg\,P(a) \qquad Q(a)$$
$$\otimes \qquad\quad \otimes$$

The reason the sentence is valid is that we have assumed that every structure has a *nonempty* domain. So, we are justified in bringing into the proof a parameter, even through an application of a type-C rule.

Example 25.2.7 In practice it will generally be more efficient to extend a tableau by first applying type-D rules which require introducing new parameters, before applying type-C rules on those parameters.

Example 25.2.8 Show $(\exists y)((\exists x)P(x) \to P(y))$ is valid.

$$(1)\ \neg\,(\exists y)((\exists x)P(x) \to P(y))$$
$$(2)\ \neg\,((\exists x)P(x) \to P(a))\ (1)$$
$$(3)\ \neg\,(\exists y)((\exists x)P(x) \to P(y))\ (1)$$
$$(4)\ (\exists x)P(x)\ (2)$$
$$(5)\ \neg\,P(a)\ (2)$$
$$(6)\ P(b)\ \ (4)\ \ {\scriptstyle \text{new } b}$$
$$(7)\ \neg\,((\exists x)P(x) \to P(b))\ (3)$$
$$(8)\ \neg\,(\exists y)((\exists x)P(x) \to P(y))\ \ (3)$$
$$(9)\ (\exists x)P(x)\ \ (7)$$
$$(10)\ \neg\,P(b)\ (7)$$
$$\otimes\ \ (6{,}10)$$

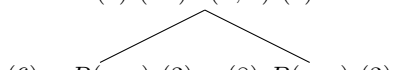
This example is a little circumspect: we introduced the parameter a by a type-C rule on (2) and then had to introduce a *new* parameter b by a type-D rule on (6), only to have to repeat our type-C rule for b on (7). It is possible to *liberalize* the type-D and require

- (Type-D rule, liberalized) From D we may infer $D(a)$, provided a is a parameter which has not been introduced by another type-D application.

The reason this is legitimate is this: if a enters the proof by a type-C application, say from $(\forall x)P(x)$ we infered $P(a)$ earlier in the proof, then we are still not assuming that a has any *special properties*, since $P(a)$ must be the case regardless of how a enters the proof. Later on we come across a type-D application, say $(\exists x)Q(x)$, we could use the old rule and introduce a new parameter b and infer $Q(b)$ and $P(b)$, but this is really no different then simply allowing $Q(a)$.

The liberalized type-D rule shortens the proof:

$$
\begin{array}{c}
(1)\ (\exists y)(\neg\, R(y,y) \vee P(y,y)) \\
(2)\ (\forall x)R(x,x) \\
(3)\ (\neg\, R(a,a) \vee P(a,a))\ (1) \\
(4)\ R(a,a)\ (2) \\
(5)\ (\forall x)R(x,x)\ (2)
\end{array}
$$



$$
\begin{array}{ll}
(6)\ \neg\, R(a,a)\ (3) & (8)\ P(a,a)\ (3) \\
(7)\ \neg\, R(a,a)\ (6) & \vdots \\
\otimes\ (4{,}7) &
\end{array}
$$

$$
\begin{array}{c}
(1)\ \neg\, (\exists y)((\exists x)P(x) \rightarrow P(y)) \\
(2)\ \neg\, ((\exists x)P(x) \rightarrow P(a))\ (1) \\
(3)\ \neg\, (\exists y)((\exists x)P(x) \rightarrow P(y))\ (1) \\
(4)\ (\exists x)P(x)\ \ (2) \\
(5)\ \neg\, P(a)\ (2) \\
(6)\ P(a)\ (4) \\
\text{(6) is liberalized type-D rule from (4)} \\
\otimes\ (5{,}6)
\end{array}
$$

You are welcome to use the liberalized rule-D but *caveat emptor*!

Example 25.2.9 In this example we show how re-using a parameter in a type-D rule that was already introduced by another application of a type-D rule can lead to trouble. The following example is not a proof.

$$
\begin{array}{c}
(1)\ \neg\, ((\exists x)P(x) \rightarrow (\forall x)P(x)) \\
(2)\ (\exists x)P(x)\ (1) \\
(3)\ \neg\, (\forall x)P(x)\ (1) \\
(4)\ P(a)\ \ (2)\ \ \text{new } a\text{, ok} \\
(5)\ \neg\, P(a)\ (3) \\
\text{(5) illegitimately re-uses } a \text{ in type-D rule} \\
\otimes\ (4{,}5)
\end{array}
$$

In fact, $((\exists x)P(x) \rightarrow (\forall x)P(x))$ is not valid. Let $\mathcal{L}$ have only the unary predicate P.

$$
\begin{array}{c}
(1)\ \neg\, ((\exists x)P(x) \rightarrow (\forall x)P(x)) \\
(2)\ (\exists x)P(x)\ (1) \\
(3)\ \neg\, (\forall x)P(x)\ (1) \\
(4)\ P(a)\ \ (2)\ \ \text{new } a \\
(5)\ \neg\, P(b)\ (3)\ \ \text{new } b
\end{array}
$$

Since there is no further rule to apply, this tableau is finished. A structure can be read off the open branch: Let the domain of $\mathcal{A}$ be $A = \{a, b\}$ and $P^{\mathcal{A}} = \{a\}$. You can verify that $((\exists x)P(x) \rightarrow (\forall x)P(x))$ is false in the structure $\mathcal{A}$.

Example 25.2.10 By repeating a type-C formula in the rule, tableaux in predicate logic need never terminate if no contradiction comes up. The next example tries to make lines (1) and (2) true.

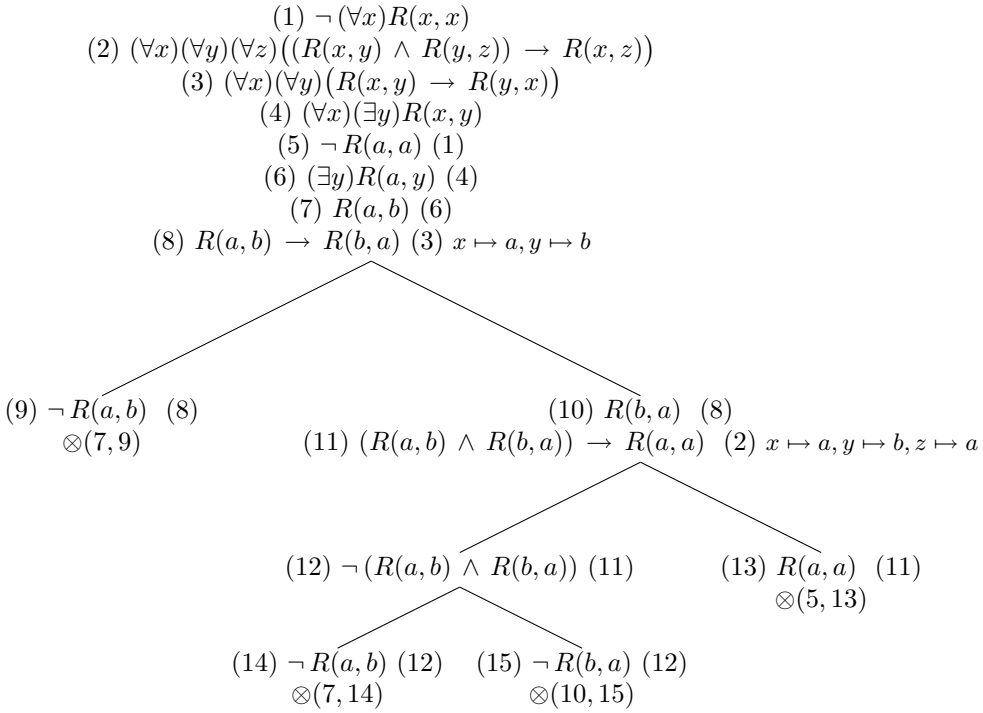
We still have not applied line (5), which is a type-C formulae. On the other hand, it is apparent that applying a type-C rule to (5) is unnecessary. The structure $\mathcal{A}$ given by domain $A = \{a\}$ and $P^{\mathcal{A}} = R^{\mathcal{A}} = \{\langle a, a\rangle\}$, from lines (4) and (8), satisfies the sentences on lines (1) and (2).

Remark 25.2.11 We can also use tableau to show $\Sigma \models \alpha$. In this case, we construct a tableau for $\Sigma \cup \{\neg\, \alpha\}$. The simplest way to do this (when Σ is finite) is to place $\neg\, \alpha$ at the root, followed by the sentences of Σ.

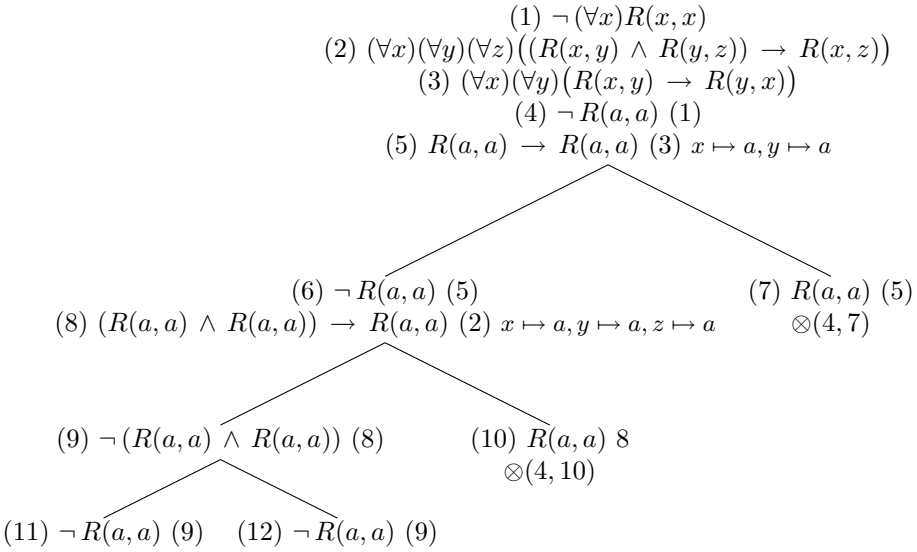
Example 25.2.12 Consider the following sentences:

$$\begin{array}{ll} (a) & (\forall x)(\forall y)(\forall z)\big((R(x,y) \wedge R(y,z)) \rightarrow R(x,z)\big) \\ (b) & (\forall x)(\forall y)\big(R(x,y) \rightarrow R(y,x)\big) \\ (c) & (\forall x)(\exists y)R(x,y) \\ (d) & (\forall x)R(x,x). \end{array}$$

We first show $(a), (b), (c) \models (d)$.



Now we show that $(a), (b) \not\models (d)$.



The structure suggested on each of the open branches is $A = \{a\}$ and $R^A = \emptyset$. You can verify (a) and (b) are true, but (d) is false. Of course (c) is false as well.