Appendix: Chapter 6, §7, second printing

7 Tableaux for monadic predicate logic

In this section, we extend the semantic tableau test for validity in sentential logic (see §5 of Chapter Three) to monadic predicate logic. Since we now have '⋏' in the language, we must first adjust the definition of when a tableau-path closes:

> A path Π in a semantic tableau is *closed* if and only if (i) '𝕋: ⋏' occurs at some node on Π, or (ii) there is a formula s and nodes n_1 and n_2 of Π such that ⌜𝕋: *s*⌝ occurs at n_1 and ⌜𝔽: *s*⌝ occurs at n_2. Π is *open* if and only if it is not closed.

In view of the decidability of monadic predicate calculus, as discussed in the previous section, the reader might expect the extended tableau method to constitute a decision procedure. However, it is rather complicated to give such a procedure for full monadic predicate calculus, so we will content ourselves with an algorithm that tests the validity of those sequents *which do not contain nested quantifiers*, that is, quantifiers that occur within the scope of other quantifiers. With this restriction in mind, we add to the sentential rules four new rules, 𝕋∀, 𝔽∀, 𝕋∃ and 𝔽∃. These rules extend a path containing a quantified formula ⌜(Qv)ϕv⌝, signed 𝕋 or 𝔽, by adding a new node, or new nodes, that contain formulae signed in the same way as ⌜(Qv)ϕv⌝. Unfortunately, there are some complications.

Suppose '𝕋: (∀x)Fx' occurs at a node. '(∀x)Fx' requires for its truth on an interpretation $\mathcal{I}$ that every instance ⌜F*t*⌝ be true, where t names an element of the domain of $\mathcal{I}$. But in constructing a tree, a domain is not given at the outset, so how do we know what instances to use? The answer is that we can use any instance we please, and any number of instances we please, since each time we write down an instance we can think of ourselves as adding a new object to the domain. On the other hand, it would be pointless just to add instances for the sake of doing so. We can certainly add instances that use names which already occur in the formulae on the paths in question, but we will want to be as parsimonious as possible in adding instances with new names. The same remarks apply to the rule 𝔽∃, since a false existential has universal import; for instance, '𝔽: (∃x)Fx' requires for its truth on an interpretation 𝓘 that *every*thing in 𝓘's domain fail to satisfy 'F', that is, that every formula ⌜~F*t*⌝ be true, where t names an element of the domain of 𝓘. For the purposes of stating the tableau quantifier rules succinctly, we shall use the term *witness* to mean a formula with the signature required by its associated signed quantified formula. For example, for any t, the signed formula ⌜𝔽: F*t*⌝ is a witness for '𝔽: (∃x)Fx', and the signed formula ⌜𝕋: F*t*⌝ is a witness for '𝕋: (∀x)Fx'.

The rules 𝔽∀ and 𝕋∃ require special restrictions. A tableaux with, say, only '𝕋: (∃x)Fx', '𝕋: (∃x)~Fx' and '𝔽: ⅄' at its root node should not close, since (∃x)Fx, (∃x)~Fx ⊭ ⅄. But we could close the tableau by using the *same* name in witnesses for '𝕋: (∃x)Fx' and '𝕋: (∃x)~Fx', since we could get '𝕋: Fa' and '𝕋: ~Fa', hence '𝔽: Fa', in three steps. Evidently, the fallacy is in using 'a' twice. To be true on 𝓘, '(∃x)~Fx' requires the truth on 𝓘 of *some* instance, but there is no justification for choosing one that mentions the *same* object as figured in '(∃x)Fx's witness. To avoid such fallacious steps, we stipulate that when we apply 𝕋∃ to ⌜(∃v)ϕv⌝ and extend a path Π by adding a node to Π with a witness ϕt, we never replace v with a name t that already occurs in a formula on Π. The same applies to the rule 𝔽∀, since a false universal has existential import; for instance, '𝔽: (∀x)Fx' requires for its truth on an interpretation 𝓘 that *some*thing in 𝓘's domain fail to satisfy 'F', that is, that some formula ⌜~F*t*⌝ be true, where t names some element of the domain of 𝓘.

The tableaux rules for monadic predicate calculus are these:

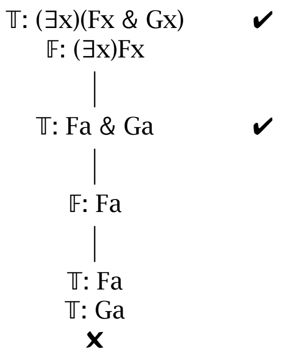
$$\begin{tikzcd}
\mathbb{T}\forall◗ & & \mathbb{F}\forall◗ & & \mathbb{T}\exists◗ & & \mathbb{F}\exists◗ & \\
& \mathbb{T}: (\forall v)\phi v \arrow[d] & & \mathbb{F}: (\forall v)\phi v \arrow[d] & & \mathbb{T}: (\exists v)\phi v \arrow[d] & & \mathbb{F}: (\exists v)\phi v \arrow[d] \\
& \{\mathbb{T}: \phi t\} & & \{\mathbb{F}: \phi t\} & & \{\mathbb{T}: \phi t\} & & \{\mathbb{F}: \phi t\}
\end{tikzcd}$$

The rules say that if the tail formula is on a node n in a tree, then we may extend each path Π on which n lies by adding a new node to the bottom of Π which contains as many witnesses as we please (different t's). The braces indicate that we may label the new node with a *set* of formulae of the relevant form, not just a single formula. And we may use different sets for different nodes. However, we must observe the restriction that when applying 𝔽∀ and 𝕋∃, *we use only names which do not occur in formulae at earlier nodes of* Π.

If we simply add these rules to the sentential system, there would be nothing to prevent us from failing to close a tableau which can be closed, since we could pointlessly continue applying the rules without ever making the *right* application. To avoid this, we adopt the following rules of order (see Jeffrey):

(1) Apply all possible sentential rules, checking off a formula whenever a rule is applied to it.
(2) If there are still open paths after (1), apply 𝔽∀ and 𝕋∃ once to each appropriate formula, using only one witness for each application and checking off a formula whenever a rule is applied to it.
(3) If there are still open paths after (2), apply 𝕋∀ and 𝔽∃ as often as possible. Either use only names in formulae already on the path, but do not *repeat* any formula already on the path; or, if there are no names on the path, write down one witness using any name. Do not check off the formulae.
(4) If there are still open paths after (3), go back to (1) and repeat the process until a pass through (1), (2) and (3) causes no changes in the tableau.

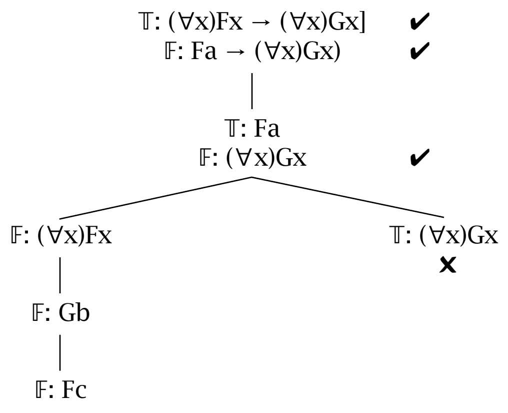
Example 1: Determine whether (∃x)(Fx & Gx) ⊨ (∃x)Fx.



Note the absence of nested quantifiers. The tableau closes, hence (∃x)(Fx & Gx) ⊨ (∃x)Fx. We follow our rules of order, though it does not matter here whether we apply 𝕋& before or after 𝔽∃. Also, if it is indifferent which of two rules is applied first and one of the rules would cause branching, it is sometimes better to apply it second. Following the rules of order guarantees that if a tableau can be closed, it will be, and that if it cannot be closed, this will be established in finitely many steps. That is why the rules of order constitute a decision procedure for validity when applied to sequents without nested quantification.

Our next example is of a tableau which determines whether (∀x)Fx → (∀x)Gx ⊨ Fa → (∀x)Gx. The tableau is displayed on the next page. The left path in this tableau remains open, and therefore we have shown (∀x)Fx → (∀x)Gx ⊭ Fa → (∀x)Gx. As in the sentential case, we can read off a counterexample to the sequent from the signed atomic formulae on an open branch once we are finished. However, we will not go into the details of this procedure, since our primary method for demonstrating invalidity is the method of §2 of this chapter. (The interested reader who wishes to explore tableaux further might try applying our method to an invalid sequent containing nested quantifiers such as (∀x)[(∀y)Gy → Fx] ⊨ (∀x)(∀y)(Gy → Fx), to see exactly what goes wrong.)

Example 2: Determine whether (∀x)Fx → (∀x)Gx ⊨ Fa → (∀x)Gx.



❑ Exercises

Repeat problems 1–19, 22 and 23 of Exercise I, §2, demonstrating invalidity by constructing open tableaux. Then for each of these sequents in which there is only one premise formula, test the converse semantic sequent for validity. For example, for problem 19, determine whether *(∃x)(Fx ↔ Gx) ⊨ (∀x)Fx ↔ (∀x)Gx.