Uniform Notation of Tableau Rules for Multiple-Valued Logics*

Reiner Hähnle
University of Karlsruhe, Dept. of Computer Science
Institute for Logic, Complexity and Deduction Systems
7500 Karlsruhe, FRG
haehne@ira.uka.de

Abstract
Starting from the general tableau-based framework for multiple-valued logics presented in [5] we show that the advantages of Smullyan's uniform notation [7], for classical logic can be made available to multiple-valued logic as well. The result is a system that serves as theoretical basis for automated theorem proving in multiple-valued logics.

1 Introduction
We assume that the reader is familiar with the standard reference on tableau calculi [7]. Here, Smullyan introduces so-called uniform notation for tableau rules, which plays a key role in the concise presentation of his calculus as well as in the proofs of the results. Uniform notation in the propositional case exploits the fact that the semantic definitions of some propositional operators can be written as a conjunction (resp. disjunction) of its subformulas. The tableau rules for each propositional operator in Table 1 can be represented as instances of only two rule schemes, which are said to be of type α (conjunctive) and type β (disjunctive) resp. (see Table 3). Unified Rule schemes for the quantifier rules of first-order logic can be devised in a similar way. Uniform notation makes it possible to have logics with a wide variety of built-in operators while at the same time the proofs of soundness and completeness are essentially the same as if one had only one built-in operator (besides negation). On the other hand the possibility of adding certain provisos to a rule scheme provides considerable flexibility, so that uniform notation has been successfully used as a tool e.g. in tableau systems for modal and intuitionistic logics [3] (and many others) as well as for specifically tuned calculi in automated theorem proving [4]. The purpose of this paper is to show how uniform notation may be also used for multiple-valued logics to give a tableau system for a wide class of operators in arbitrary logics with finitely many truth values that can serve as well as a theoretical basis for automated theorem proving.

In Section 1 we present a tableau system for arbitrary propositional logics with finitely many truth values, which will be a signed version of tableau calculus with a generalized notion of signs. This system was presented in [5] and will serve as a framework for the following sections. In Section 2 we will give a more or less direct adaption of Smullyan's tableau system to multiple-valued logics, while in Section 3 a slightly more general system for a wide class of multiple-valued operators is presented, but still in uniform notation style. In Section 4 we take a look at functional completeness of the respective logics, in Section 5 we sketch a possible extension of our ideas to quantified multiple-valued logic.

2 A Tableau System for arbitrary Finitely-Valued Propositional Logics
Let \( \mathcal{F} = \{ F_1, \ldots, F_r \} \) be a set of logical connectives and \( L_0 := \{ p_i \mid i \in \text{Nat} \} \) the set of propositional variables or atomic formulas, which has to be disjoint from \( \mathcal{F} \). By \( L \) we denote the abstract algebra that is freely generated over \( L_0 \) in the class of algebras with type \( \mathcal{F} \).

\( L_i \) denotes the formulas of depth \( i \). We call \( L \) (propositional) language, the members of \( L \) are called (propositional) \((L-)\)formulas.

Let \( N = \{ 0, 1, \ldots, (n - 1) \} \) be the set of truth values and \( D \subseteq N \) the set of designated truth values\(^1\). Furthermore let us denote with \( n = |N| \) and \( d = |D| \) the number of elements in \( N \) and \( D \) resp. Though all nonnegative values are possible for \( n \) and \( d \), we are only interested in the nontrivial cases where \( n \geq 2 \) and \( d \geq 1 \).

Let \( A = \langle N, \{ f_i \mid 1 \leq i \leq r \} \rangle \) be an algebra of the same type as \( L \). Then we call the pair \( A = \langle A, D \rangle \) a structure for \( L \) and the \( f_i \) interpretations of the \( F_i \). A defines the semantics of the logical operators. We say that \( A = \langle L, A \rangle \) is an \( n\)-valued propositional logic with \( d \) designated truth-values.

A propositional \((A-)\)valuation of \( L \) is a homomorphism \( v \) from \( L \) to \( A \). A set \( M \) of \( L \)-formulas is called \((A-)\)satisfiable, if there is a valuation \( v \) from \( L \) to \( A \) such that for any \( X \in M \) \( v(X) \in D \) holds. In this case \( v \) is called \((A-)\)model for \( M \), \( X \) is called \((A-)\)formula, if any \( A \)-valuation \( v \) is also a model for \( \{ X \} \).

\(^1\)In multiple-valued logics the designated truth values are those that support the validity of a statement. We do not assume any specific structure on the set of truth values nor do we associate any philosophical interpretation with the truth values.
Example 2.1 As the set of logical operators we take \( \mathcal{F} = \{ \vee, \wedge, \neg, \forall, \exists \} \) with arities \( m(\vee) = 2, m(\neg) = 2, m(\wedge) = 1, m(\forall) = 1, m(\exists) = 1 \) and as truth values \( N = \{ 0, 1, 2 \} \), \( D = \{ 2 \} \). Their semantics is given by the following truth tables:

<table>
<thead>
<tr>
<th>v</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \vee )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \neg )</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Let us refer to this logic with the symbol \( \mathcal{L}_3 \).

Given any logic of this kind, we would like to have a sound and complete tableau proof system for it. This task was begun by Surma [8] and carried further by Carnielli [l], who provided a generic signed tableau proof system for multiple-valued first-order logics with arbitrary logical connectives and generalized quantifiers. Carnielli’s system is not particularly well suited for automating theorem proving (which was not his aim), because it works by simply introducing a sign for each truth value and thus extending the analysis from two to multiple truth values. One consequence is that the exclusion of all interpretations of a formula \( X \) assigns a non-designated truth value to \( X \) (which proves the validity of \( \neg \neg X \)), requires the construction of \( n - d \) separate proof trees. The rules generate many more branches than in classical logic and the simple \( \alpha \beta \) schemes are lost due to the case analysis by single truth values. This is a kind of inefficiency that is not inherent in the problem and in [5] we presented a method, how it can be overcome. Our idea is to increase the expressivity of signs in order to be able to express conditions like \( \neg \neg v(X) \neq 2 \), or equivalently \( "v(X) \neq 2" \), with a single signed formula by admitting sets of truth values as signs. To establish the validity of \( X \) it is then no longer necessary to build \( n - d \) proof trees. Instead, it suffices to build a single proof tree with root \( S_{N \cdot D} X \), where the sign \( S_{N \cdot D} \) corresponds to the set of non-designated truth values.

Definition 2.1 (Sign, Signed Formula, Satisfiable Formula)

Let \( L \) be any language and \( D \) and \( N \) defined as above. Then we define the set of signs as \( S = \{ S_i \mid i \in 2^N \} \). For any logic \( \mathcal{L} \) we fix a certain set of signs \( S_{\mathcal{L}} \subseteq S \) which satisfies \( S_{\{0\}}, S_{\{1\}}, \ldots, S_{\{N-1\}} \subseteq S_{\mathcal{L}} \). From now on a logic will be a triple \( \mathcal{L} = (L, A, S_{\mathcal{L}}) \). With \( I_L = \{ i \mid S_i \in S_{\mathcal{L}} \} \) we denote the set of allowed indices of signs. With the same symbol we identify the abstract algebra generated by \( I_L \) that has the same type as \( A \) and whose fundamental operations are defined by

\[
F_L(i_1, \ldots, i_m) = \{ f^A(j_1, \ldots, j_m) \mid j_k \in i_k, 1 \leq k \leq m \}
\]

If \( X \) is an \( L \)-formula and \( S_i = S_{\{i_1, \ldots, i_k\}} \) a sign, then we call the string \( S_i(X) \) signed \( (L-)\)formula. \( L^* \) is the set of signed formulas in a logic \( \mathcal{L} \), i.e. all signed \( L \)-formulas with signs from \( S_{\mathcal{L}} \). The members of \( L^* \) will be called \( L_{\mathcal{L}} \)-signed formulas.

Example 2.2 For \( \mathcal{L}_3 \) define \( \{ S_{\{0\}}, S_{\{1\}}, S_{\{2\}}, S_{\{0,1\}} \} \) as the set of signs, which for convenience we rewrite as \( \{ F, U, T, \{ F, U \} \} \). The intended interpretation of a signed formula \( \{ F, U \}(X) \) is \( "v(X) = 0 \) or \( v(X) = 1" \).}

Definition 2.2 (Tableau Rule)

Let \( X = F(X_1, \ldots, X_m) \) be an \( L \)-formula in the logic \( \mathcal{L} = (L, A, S_{\mathcal{L}}) \). An \( \mathcal{L} \)-tableau rule is a function \( \pi : \mathcal{P} \rightarrow \mathcal{S} \) which assigns to a signed formula \( S_i(X) \in L^* \) a tree with root \( S_i(F(X_1, \ldots, X_m)), \) called premise, and linear subtrees (denote with \( \ldots \) a linear tree)

\[
S_{j_1}(X_{i_1}) \circ \ldots \circ S_{j_t}(X_{i_t})
\]

for which \( j_1, \ldots, j_t \in I_L, t \leq m \) and \( H_i(F; j_1, \ldots, j_t) \) (see below) holds.

called extensions.

A collection of extensions satisfying \( (T0) \) is called conclusion of a tableau rule.

\( (T0) \) for any \( (z_1, \ldots, z_m) \in f^{-1}(i) \) there is an extension \( S_{j_1}(X_{i_1}) \circ \ldots \circ S_{j_t}(X_{i_t}) \) with \( z_k \in j_k, 1 \leq k \leq t \) and the set of extensions is minimal with respect to this condition.\(^4\)

The condition \( H_i(F; j_1, \ldots, j_t) \) means, there exists a homomorphism \( h : L \rightarrow I_L \), satisfying \( (T1)-(T4) \) below:

\( (T1) \) \( h(X_{i_k}) = j_k \) for \( 1 \leq k \leq t \).

\( (T2) \) If \( f \) interprets \( F \), then \( f(v_1, \ldots, v_m) \in \mathcal{P} \), \( v_i \in h(X_{i_k}) \), for \( 1 \leq k \leq t \) and all other arguments arbitrary.

\( (T3) \) There is no \( j_k \) with \( |j_k| > |j_k| \) for \( 1 \leq k \leq t \) that satisfies \( (T1) \) and \( (T2) \).

\( (T4) \) There is no \( t' \) with \( t' < t \) that satisfies \( (T1) \) and \( (T2) \).

If no such homomorphism exists, no rule for the specific combination of formula and sign is defined.

Extensions are treated like sets. Thus of all subtrees that differ only in the ordering of their signed formulas one appears as an extension of the rule.

\(^2\)Otherwise it is not guaranteed that all rules can be properly stated.

\(^4\)Already in the two-valued case there may be more than one minimal (in our sense) set of extensions for a signed formula, so we need the minimality condition; see [2, p. 12f] for an example.
The conclusion of a tableau rule for a sign \( i \) and connective \( F \) can be thought of as a minimal generalized sum-of-products representation of the \( \text{two-valued function} \) that holds the entry true in its truth table on each place where the truth table of \( F \) holds a member of \( i \) and holds false otherwise.

Each extension corresponds to a product term in this representation. A geometrical interpretation would associate a partial cover of entries (namely the ones occurring in \( i \)) in the hypercube that constitutes the truth table of \( F \) with an extension. All extensions taken together are a total cover.

Example 2.3 The rules for implication and \((F\cup)\), implication and \( U, \nabla \) and \((F\cup)\) in \( L_3 \) are

<table>
<thead>
<tr>
<th>( (F\cup)(A \rightarrow B) )</th>
<th>( U(A \rightarrow B) )</th>
<th>( U(A \rightarrow B) )</th>
<th>( (F\cup)\nabla A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( UA )</td>
<td>( TA )</td>
<td>( TA )</td>
<td>( UB )</td>
</tr>
</tbody>
</table>

For \( U \) and \( \sim \) (weak negation) exists no corresponding rule.

A complete tableau for a finite set of \( L_3 \)-signed formulas is now created in the usual manner. Only a minor modification has to be made to the conditions for branch closure:

**Definition 2.3 (Open, Closed)**

A tableau branch is called closed if one of the following conditions is satisfied:

1. It contains a complementary atom set, i.e. signed atomic formulas \( S_i(p), \ldots, S_n(p) \) satisfying
   \[ \bigcap_{j=1}^n i_j = \emptyset \]
2. It contains a non-atomic signed formula for which no rule is defined.

Example 2.4 We prove that the formula \((\neg A \vee A) \vee \neg A \) is a \( \sim \)-tautology by constructing a closed tableau (Figure 1) with root \((F\cup)(\neg A \vee A) \vee \neg A \).

**Theorem 2.1 (Soundness, Completeness; \[5\])**

For any logic \( L \) an \( L \)-formula \( A \) is a tautology iff there exists a closed tableau with root \( S \neg \neg \neg (A) \).

Although this system yields quite nice tableau rules for most logics that can be found in the literature, it has still some disadvantages. For example, no algorithm is given to determine the tableau rules, and it is very tedious to check out the required homomorphisms by hand. Also, the common structure of the rules is merely

**Table 1: Primary propositional connectives**

<table>
<thead>
<tr>
<th>( \neg )</th>
<th>( A )</th>
<th>( \sim )</th>
<th>( \vee )</th>
<th>( \neg )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Let \( N = \{0, \ldots, n-1\} \) be a set of truth values, \( D = \{n-d, \ldots, n-1\} \) the set of designated truth values and \( \bar{D} = \{0, \ldots, n-d-1\} \) the set of non-designated truth values, i.e. \( N = D \cup \bar{D} \). For any truth value \( i \in N \) we define its conjugate \( \bar{i} = n-1-i \). As usual, the conjugate of a set is the set of its conjugates, i.e. if \( S \subseteq N \) then \( S^* = \{\bar{i} : i \in S\} \). For reasons to be seen later we include the conjugation operator also in our logic as another name for negation.

Let \( L_3^\text{sym} \) be the \( n \)-valued logic whose language consists of the primary connectives plus negation, and whose semantic is given by the following evaluation rules:

\[
\begin{align*}
&v(\neg x) = v(x^*) = v^*(x) \\
&v(x \lor y) = \max\{v(x), v(y)\} \\
&v(x \land y) = \min\{v(x), v(y)\} \\
&v(x \rightarrow y) = \max\{v^*(x), v(y)\} \\
&v(x \leftrightarrow y) = \min\{v(x), v(y)\} \\
&v(x \sim y) = \max\{v(x), v(y)\} \\
&v(x \sim y) = \min\{v(x), v(y)\}
\end{align*}
\]

The primary connectives are exactly those, that can be characterized by an \( \alpha \) or \( \beta \) rule, see Table 1.

Note, that in the case when \( N = \{0,1\} \), \( D = 1 \) the classical primary connectives with the truth tables as shown above will result.

3 Primary Connectives in Multiple-Valued Logics

Consider multiple-valued logics that consist solely of generalized versions of classical primary connectives in the sense of Smullyan [7]. For these connectives tableau rules can be supplied that resemble the classical rules for \( \alpha \) and \( \beta \) formula components very closely. First we have to state what we mean by generalized versions of classical primary connectives. For convenience in Table 1 we repeat Smullyan's primary connectives.

<table>
<thead>
<tr>
<th>( v )</th>
<th>( A )</th>
<th>( \sim )</th>
<th>( \vee )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

This case corresponds to closure of branches that contain e.g. \( T \) in classical logic.

As pointed out above, one could use existing multiple-valued minimization techniques, but usually these yield only near minimal solutions. The problem of finding a minimal sum-of-product representation for a given function is known to be NP-complete.
This definition provides a natural generalization of classical primary connectives to the multiple-valued case. Let the set of signs of $L^m$ be $\{S_D, S_D^+, S_D^-, S_D^0\}$ which for convenience we abbreviate with $\{D, D^+, D^-, D^0\}$. Table 2 lists the components of $\alpha$ and $\beta$ type formulas, which together with the usual tableau rule schemes and the obvious rules for negation (see Table 3) constitute a sound and complete tableau system for each of the logics $L^m$. Note that the number of signs does not depend on $N$ or $D$.

So we have constructed a uniform notation style tableau system for testing validity of formulas in any logic that contains at most negation plus the generalized versions of the classical primary operators. To test validity of a formula, say $A$, all we have to do is to construct a single closed tableau with root $\hat{D} A$.

Automated tableau construction is possible with any theorem prover that can be modified to handle four instead of two signs.

The reason why the conjugates of $D$ and $\hat{D}$ must be present is, that in general the equality $\hat{D} = D^*$ does not hold, as our example $L_3$ shows. $\hat{D}$ represents non-designated truth values, whereas $D^* A$ is equivalent to $D A$. In classical logic negation and non-designatedness coincide.

Table 3: Tableau rules for $\alpha$ and $\beta$ formulas and for negation

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_1$</td>
<td>$\beta_1$</td>
</tr>
<tr>
<td>$\alpha_2$</td>
<td>$\beta_2$</td>
</tr>
</tbody>
</table>

$D \neg \neg x \quad D^* \neg \neg x \quad D \neg \neg x \quad D^* \neg \neg x$

4 A Uniform Notation Style Tableau System

In the following we will define a larger class of logical operators than in the last Section with a certain kind of regularity resulting in a particularly simple tableau system. The price will be an increase in the number of involved signs.

We observe that the shape of the required coverings in the truth tables (if represented as a two-dimensional array) for the primary connectives is extremely simple, namely one of:
The pattern may start from an arbitrary corner. Combining this with the fact that in multiple-valued deduction systems one usually asks whether the truth value of a formula is in $D$ or $\overline{D}$, we are led to the following definitions.

**Definition 4.1 (Minterm, Circle)**

Consider the $k$-dimensional hypercube of length $n$ that represents the truth table of a $k$-ary operator $o$ in an $n$-valued logic $L$. A minterm\(^{10}\) of $L$ is $I = (x_1, \ldots, x_k)$ such that $x_i = 1$ for $1 \leq i \leq k$ corresponds to exactly one entry of the cube, namely the one that contains the truth value of $o(x_1, \ldots, x_k)$. The circle with radius $r$ and center $I$ is defined as

$$c_{r, I} = \{(y_1, \ldots, y_k) \mid (i) \text{ there is a } j \leq k \text{ such that } |y_i - x_i| = r \text{ for } i \neq j \text{ and } |y_i - x_i| \leq j \text{ for all } i : y_i \in N\}$$

We define the set of truth values that occur in the entries of a truth table for $o$ on a circle $c_{r, I}$ with radius $r$ around a minterm $I$ as $C_{o, r, I} = \{o(y) \mid y \in c_{r, I}\}$.

**Definition 4.2 (Regular logical operator)**

A logical operator $o$ is called regular iff there is a corner minterm $I$ such that:

1. for any $r \in N$, $C_{o, r, I}$ is a singleton $\{c_r\}$ and for these
2. $c_0 \leq \ldots \leq c_{n-1}$ or $c_0 \geq \ldots \geq c_{n-1}$ holds.

Call $I$ the starting point of $o$.

The entries in truth tables of regular operators are ordered starting from some corner with the lowest (resp. highest) truth value the operator takes on. On each circle with the starting point as its center all entries contain the same truth value. These truth values are monotonically increasing (resp. decreasing) with the radius of the circles. Table 4 shows an example of a regular operator. Also all operators of our earlier example $L_3$ are regular.

Just to simplify notation from now on we concentrate on binary operators.

\(^{10}\)We adopt here a terminology that is frequently used in minimization of logical functions, see e.g. [2].

---

**Table 4: Truth table for operator $\odot$. The marked entries constitute a circle with radius 2 and center $(3, 0)$.**

<table>
<thead>
<tr>
<th>$\odot$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Definition 4.3 (Direction, Orientation, Threshold)**

Let $o$ be a regular operator and $I$ its starting point. Then $\delta(o) = (b_1(o), b_2(o))$, where $b_i(o) = *$ iff $x_i = n - 1$ and $b_i(o) = I$ otherwise, is called the direction of $o$. $e(o) = *$ iff the values on circles around $I$ are monotonically increasing with the radius and $I$ otherwise is called the orientation of $o$.

Finally, define thresholds for an operator $o$ and a truth value $s \in N$ as:

\[
\begin{align*}
t_{o, s} &= \min\{z \mid o(z \circ x) \geq s\} \\
t_{o, s} &= \max\{z \mid o(z \circ x) \leq s\}
\end{align*}
\]

**Figure 2: Determining the tableau rules for $E^* \odot y$**

Permuting all parameters in the two-valued case determines exactly the eight classical primary connectives as the non-trivial regular operators associated with each combination. In the multiple-valued case a regular operator is completely determined by its parameters and the values on the diagonal through its starting point.

Our goal is a tableau proof system with a sufficient number of rules for completely handling queries of the form $D A$ and $\overline{D} A$. Assume for the moment that $\delta(o) = (I, I)$, $e(o) = I$. From Figure 2 it is obvious that we need only consider signs of the form $S_{(0, \ldots, 1, \ldots, -1)}$, where $s_1 = t_{o, d}$ for $\overline{D} A$ (since $\overline{D} = E^{n-4}$). The same reasoning applies to $S_{(1, \ldots, n-1, \ldots, -1)}$ and $s_2 = t_{o, d}$ for $D A$. As

\(^{11}\)Here $*$ stands for the conjugation defined in the previous section and $I$ stands for the identity function.
a shorthand for the corresponding signs we write \( E' \) for \( S_{0,\ldots,n-1} \) for \( s \in N \)
\[ s' = S_{s+1,\ldots,n-1} \text{ for } s \in N \]

Provided, in the premises of our rules occur only signs of the form \( E' \) and \( s'E \), we have also in the conclusion only signs of this form and obviously the following holds:
\[ x \odot y \in E' \text{ iff } x \in E^\odot_0 \text{ and } y \in E^\odot_0 \]
\[ x \odot y \in s'E \text{ iff } x \in (s^\odot_0E) \text{ or } y \in (s^\odot_0E) \]

**Example 4.1** Let \( \odot \) be defined as in Table 4. Then 
\[ \delta_1(\odot) = ^*, \delta_2(\odot) = I, \delta(\odot) = ^* \]
Consider \( \wedge \) from Example 2.1:
\[ t_{\wedge,2} = 2, \bar{t}_{\wedge,2} = 2. \]

**Theorem 4.1** Let \( \mathcal{L} \) be any logic containing only regular connectives \( \mathcal{F} \), such that for each \( \alpha \in \mathcal{F} \) \( \delta_1 = \delta_2 = \epsilon = I \). Fix \( S_\mathcal{L} = \{E' \mid s \in N\} \cup \{s'E \mid s \in N\} \). Then the tableau system given by the following \( \alpha \) and \( \beta \) component rules is sound and complete.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( E^\alpha_0 x \odot y )</th>
<th>( E^\alpha_{0^*} x )</th>
<th>( E^\alpha_{0^*} y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \beta )</td>
<td>( s'E x \odot y )</td>
<td>( s^\odot_0E x )</td>
<td>( s^\odot_0E y )</td>
</tr>
</tbody>
</table>

**Proof:** Apply Theorem 2.1.

To adjust the rules for other values than \( I \) of \( \delta_1 \), observe that \( \delta_1 = ^* \) indicates that the truth table of \( f \) is flipped around its horizontal axis. To compute the threshold function we must count from upside down in the first argument of \( f \), in other words, we must conjugate the first argument before computing the threshold, however, the result has to be conjugated again. The same considerations also apply to the second argument of \( f \), thereby arriving at the following definitions for the threshold functions and component rules:
\[ t_{\alpha,2} = \min \{ x \mid v(\delta_2(\odot) o \delta_2(\odot)) \geq \delta(\odot) x \} \]
\[ \bar{t}_{\alpha,2} = \max \{ x \mid v(\delta_2(\odot) o \delta_2(\odot)) \leq \delta(\odot) x \} \]

**Example 4.2** Consider the three-valued implication as defined in Section 1. Assume we wanted to compute the tableau rule for the sign \( S_{1,2} \). First \( S_{0,1} = E' \), so our component rules tell us that we have a \( \beta \) type rule with extensions \( (s^\odot_0E)^{\delta_2(\odot)} x \) and \( (s^\odot_0E)^{\delta_2(\odot)} y \).

Now, since \( \delta(\odot) = ^* \), \( I \) we have \( t_{\odot,0} = \max \{ x \mid v(x^\odot_0y \leq 0 \} = 0 \), which yields extensions \( (s0E)^* x \) and \( s0E y \). \( (s0E)^* = (s0E)^0 = s0E = S_{1,2} \) finally yield the rule
\[ \frac{S_{1,2} (x \odot y)}{S_{0,1} (x)} \cdot \frac{S_{0,1} (y)}{S_{1,2} (y)} \]

We can treat negation as a special case of the above stated rules. If we note that \( \alpha_1(\neg) = \alpha_2(\neg), \beta_1(\neg) = \beta_2(\neg), \delta_1(\neg) = ^* \) and \( t_{\neg,2} = t_{\neg,2} = s \) for all \( s \), treat extensions as sets and omit identical extensions, we arrive at the negation rules stated in Table 5.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( \alpha_1 )</th>
<th>( \alpha_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E^\alpha_0 x \odot y )</td>
<td>( E^\alpha_{0^*} x )</td>
<td>( E^\alpha_{0^*} y )</td>
</tr>
<tr>
<td>( s'E x \odot y )</td>
<td>( s^\odot_0E x )</td>
<td>( s^\odot_0E y )</td>
</tr>
</tbody>
</table>

**Table 5: Negation Rules**

We still have to treat the case when \( \epsilon(\odot) = ^* \). Taking this into account in the component rules and threshold functions leads to the following definition and theorem:
\[ t_{\odot,2} = \min \{ x \mid v(x^\odot_0(\odot) o x^\odot_0(\odot)) \geq \epsilon(\odot) x \} \]
\[ \bar{t}_{\odot,2} = \max \{ x \mid v(x^\odot_0(\odot) o x^\odot_0(\odot)) \leq \epsilon(\odot) x \} \]

(5) (6)

(5) (6)

(5) (6)

**Example 4.3** Let \( \odot \) be defined as in Table 4.
\[ t_{\odot,2}, \bar{t}_{\odot,2} = 2. \]

**Theorem 4.2** Let \( \mathcal{L} \) be as in Theorem 4.1, but with no restrictions on \( \delta_1, \epsilon \). Then the tableau system given by the following \( \alpha \) and \( \beta \) component rules is sound and complete.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>( \alpha_1 )</th>
<th>( \alpha_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E^\alpha_{0^*} y \odot x )</td>
<td>( E^\alpha_{0^*} x )</td>
<td>( E^\alpha_{0^*} y )</td>
</tr>
<tr>
<td>( s^\odot_0E x \odot y )</td>
<td>( s^\odot_0E x )</td>
<td>( s^\odot_0E y )</td>
</tr>
</tbody>
</table>

**Proof:** By reduction to the case when \( \epsilon(\odot) = I \), using Lemma 4.1.
Lemma 4.1. For all regular operators $o$:

1. $(E^s)^* = i^s E$ and $i^s E = E^s$ for all $s \in \mathbb{N}$
2. $(i^s)^* = i$ for all $i \in \mathbb{N}$
3. $(X^s)^* = X$ for all $X \subseteq \mathbb{N}$
4. $t_{o^s}s^* = t_{o^s}$ and $t_{o^s}s^* = t_{o^s}$ for all $s \in \mathbb{N}$

Proof: Straightforward.

Example 4.4. Recall $\circ$ defined in Table 4. We compute the tableau rule for sign $S_{[3]}$. From $S_{[3]} = \{E^2\}^*$ and Examples 4.1, 4.3 we see that our generic rule may be instantiated, yielding the extensions:

$$(E^2)^* \ x = 1 \ E \ x = S_{[2,3]} \ x$$
$$(E^2)^* \ y = S_{[0,1]} \ y$$

5. Functional Completeness

One may argue that the class of regular connectives is too small and many multiple-valued logics fall outside of it. In this Section we show that for each $n$ there is a functionally complete $n$-valued logic with only regular connectives. Thus nothing essential has been lost for one can always replace nonregular connectives by regular ones.

We start from a well-known family of many-valued logics that is known to be functionally complete, namely the logics $P_2$ of Post. Each $P_2$ contains exactly two connectives $V$ and $\sigma$ with arities 2 and 1 resp. The semantics of $V$ is as above, while $\sigma$ may be defined by

$$v(\sigma(X)) = (v(X) - 1) \ mod \ n$$

where "$\mod$" and "$\mod$" are interpreted as usual. Since $V$ is regular, we may be able to show that $\sigma$ can be composed by regular connectives alone for each $n$.

Consider the unary connectives $\sigma'$ and $J_0$ defined by

$$v(\sigma'(X)) = \max\{0, v(X) - 1\}$$

$$v(J_0(X)) = \begin{cases} n - 1 & \text{if } X = 0 \\ 0 & \text{otherwise} \end{cases}$$

Both are regular, and obviously $\sigma(X) \equiv \sigma'(X) \vee J_0(X)$.

6. First-Order Multiple-Valued Logic

In this Section we sketch how the idea of using sets of truth values as signs may be extended to First-Order Logic. Moreover, it will turn out that our restriction to signs of the form $E^s$ and $i^s E$ is sufficient to preserve the form of $\sigma$ and $\delta$ rule schemes entirely.

Let $L^1 = L^1(R, \text{Par})$ be a first-order language defined in the usual way, where $R$ is a nonempty, countable set of predicate symbols (each with an associated arity) and $\text{Par}$ an infinite, countable set of parameters.$^{14}$ Additionally, we have a set of individual variables $IV$. Formulas are constructed from those in the usual manner, using a set of connectives $\mathcal{F}$ and the quantifier symbols $V, \exists$.

We assume familiarity with the notions of substitution (denoted with $[X/p]$) and sentence from classical logic, which we use in the standard way. Denote the set of sentences from $L^1$ with $L^1$.

Let $L$ be the propositional sublanguage of $L^1$ consisting of its quantifier free sentences. Let $L'$ be the set of sentences from $L$ with no leading quantifier and let $L = \forall A, L >$ be a propositional logic based on $L$ as defined in Section 1.

Now we are able to define the first-order extension of a propositional valuation $v : L' \rightarrow N$ to $\vec{v} : L^1 \rightarrow N$ by

$$(V1) \ \vec{v}(Y) = v(Y) \text{ if } X \in L'$$
$$(V2) \ \vec{v}((\forall X)Y) = \min\{\vec{v}(Y{p/p}) | p \in \text{Par}\}$$
$$(V3) \ \vec{v}((\exists X)Y) = \max\{\vec{v}(Y{p/p}) | p \in \text{Par}\}$$

Analog to the propositional case it is sufficient to define $\vec{v}$ on $L$ and then extend it uniquely to $L^1$.

The semantical definitions of the quantifiers as given here, can be read as a generalized conjunction and disjunction resp. Another way to look at the quantifiers would be to ask which truth values are taken on by the quantified formula while the variable is running through the substitutions. If we denote this distribution of truth values by a subset of $N$, we obtain alternative characterizations of the quantifiers by giving a mapping from distributions to truth values. In the case of $n = 3$ e.g. $(\forall X)Y$ has truth value 1 for the distributions $\{1\}, \{1, 2\}$. This may also be stated as "$\vec{v}((\forall X)Y) = 1" \text{ if either } \vec{v}(Y{p/p}) = 1 \text{ for all } p \in \text{Par} \text{ or } \vec{v}(Y{p/p}) = 1 \text{ for some } p \text{ and } \vec{v}(Y{p/p}) = 2 \text{ for some other } p"$.

Rephrasing this observation in the language of tableau rules leads Carnielli [1] to rules like:

$$\frac{S_1 \ (VX)Y}{S_1 \ Y[x/p]} \ | \ S_1 \ Y[x/p_1] \ | \ S_2 \ Y[x/p_2]$$

With the proviso that for $p$ in the left extension an arbitrary $p \in \text{Par}$ may be substituted and in the right extension $p_1 \neq p_2$ must be new on the tableau. Each extension corresponds to one possible distribution of truth values.

Obviously, we could adopt this approach to our system with generalized signs by simply taking into the conclusion all distributions that are associated with a truth value occurring in a sign. If we asked e.g. for the tableau rule for $S_{[0,1]} (VX)Y$ we would arrive at a rule that has as

$^{14}$Of course it is possible to include function symbols as well, but this requires some more work (and space) in the following definitions.
its conclusion all extensions from the rules for \( S_0 \) (\( \forall x \) \( Y \)) and for \( S_2 \) (\( \forall z \) \( Y \)). But then in the extensions only signs corresponding to singleton sets would occur and it is hard to see how one can take any advantage from the notion of generalized signs here.

The situation becomes quite different if we restrict as before the set of signs to be \( S_C = \{ E^s \mid s \in N \} \cup \{ E^i \mid s \in N \} \). In this case we arrive at the following, surprisingly simple uniform notation schemes for quantified formulas:

<table>
<thead>
<tr>
<th>( \gamma )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall x ) ( Y {z/p} )</td>
<td>( \exists x ) ( Y {z/p} )</td>
</tr>
<tr>
<td>( \forall z ) ( Y {z/p} )</td>
<td>( \exists z ) ( Y {z/p} )</td>
</tr>
</tbody>
</table>

The corresponding expansion rules in Table 6 are restricted by the usual provisos for \( \gamma \) and \( \delta \) rules, namely that \( p \in Par \) is arbitrary in the \( \gamma \) rules and new on the tableau in the \( \delta \) rules. Together with the propositional tableau rules from either Section 1–3 these rules result in a sound and complete system for the respective logics as long as the set of signs is restricted to \( S_C \).

To justify this let us concentrate on the \( \gamma \) rule for \( \forall \). The meaning of the premise is:

\((*)\) There is an \( i \in \{s + 1, \ldots, n - 1\} \) such that \( \min\{(i(Y \{z/p\}) \mid p \in Par\} = i \)

Obviously, this \( i \) must be unique, say it is \( i_0 \). Since \( i_0 \) is the minimum of all \( i(Y \{z/p\}) \), \( p \) ranging over \( Par \), all other values of \( i(Y \{z/p\}) \) lie between \( i_0 \) and \( n - 1 \) and we may conclude that

\((**)\) For each \( p \in Par \) there is an \( i_1 \in \{s + 1, \ldots, n - 1\} \) such that \( i(Y \{z/p\}) = i_1 \)

But this is exactly what the conclusion of the rule says. Note, that it is essential for the argument to hold that the set of truth values under consideration has no gaps between \( s + 1 \) and \( n - 1 \). If we substituted an arbitrary \( S_j \) for \( E \) we could not be sure whether the required \( i_1 \)’s are contained in it. For the other direction let \( i_0 \) be the minimum ranging over all \( i_1 \)’s. This proves the equivalence of \((*)\) and \((**)\), which is sufficient to prove tableau soundness and completeness.

By using the first part of the equivalence one can easily prove the \( \delta \) rule for \( \forall \). Justification of the rules for \( \exists \) is reduced to the \( \forall \) case by duality.

We remark that fairly nice rules may be also obtained for signs of the form \( S_{(j)} \), where \( j \in N \), if it is desired. A free variable version (cf. [4]) of the system is obtainable.

7 Summary and Conclusions

We have presented a framework for axiomatizing arbitrary finitely valued logics with minimal overhead if compared to the classical case. The main idea was to work with tableaux using generalized signs, which enabled us to express complex assertions regarding the possible truth values of a formula.

We have introduced the class of regular logical connectives which, together with a suitable restriction on queries (i.e. allowed signs) to the system, allow a uniform notation style presentation of multiple-valued propositional and first-order logics. It has been demonstrated that various systems differing in their allowed classes of connectives and complexity of rules may be formulated.

This allows the use of tools and methods that are close to the ones used in classical logic, both on the theoretical (uniform notation in definitions and proofs) and practical side (use of classical theorem provers with few modifications).

References


