

Automatic extraction of axiomatizations in terms of two-signed tableaux for finite-valued logics

Dalmo Mendonça

Scientific Undergraduate Fellow, UFRN

`dalmo3@gmail.com`

João Marcos

DIMAp / CCET, UFRN

`jmarcos@dimap.ufrn.br`

Abstract

Classical Logic is bivalent in that it admits exactly two truth-values: the *true* and the *false*. Many-valued logics, in contrast, allow for the consideration of arbitrarily large classes of truth-values. To export the canonical notion of entailment to the realm of many-valuedness, the trick is to characterize any such class of truth-values by saying that some of these values are ‘designated’. One might remark then that a shade of bivalence lurks in the distinction between the values that are designated and those that are not. It is known (cf. [2]) that this residual bivalence allows in fact for an alternative, and in many cases even constructively obtained (cf. [1]), representation of many-valued logics in terms of appropriate bivalent semantics. In this paper we will present a first concrete implementation of the method devised in [1] in order to obtain sound and complete classic-like tableaux systems for a very comprehensive class of finite-valued logics. The method is implemented in the functional programming language ML, and our program outputs a text file containing the corresponding theory to be processed by *Isabelle*, a flexible theorem-proving environment in which it is possible to check meta-results and theorems about the logics under scrutiny. The formulation of different many-valued logics under a common ground—two-signed tableaux systems—makes it easier to compare properties of these logics and to appreciate the relations between them.

Keywords. Many-valued logics, bivalence, axiomatization, tableaux, automated theorem proving.

Reducing logics to a bivalent common ground

In the 1970s, the Polish logician Roman Suszko used to insist that *there are but two logical values, true and false*, and complained about the *mad multiplication*

of *logical values* that plagued the literature on many-valuedness (cf. [5]). He insisted that the many different *algebraic truth-values* of many-valued logics, belonged in the end to two classes: the ‘true’ values (those we nowadays call *designated*) and the ‘false’ ones (those we call *undesignated*). To illustrate the significance of those classes, Suszko showed (cf. [4]) that it is possible to represent Łukasiewicz’s 3-valued logic L_3 in terms of a bivalent semantics.

Caleiro et al. presented a constructive method to reduce a large class of logics characterized by truth-functional finite-valued semantics into semantics containing exactly two truth-values (cf. [1]). To fit as input for that method, a given logic should prove sufficiently expressible, that is, it should be possible to distinguish any given truth-value from all the remaining ones using exclusively the linguistic resources of that very logic. The main output of the method is a set of clauses written in the language of First-Order Logic (**FOL**) that impose restrictions over the set of bivaluation functions in such a way that this set should provide a sound and complete two-valued semantics for the many-valued logic considered from the start. Those same clauses can then be used, as shown in [1], to provide a sound and complete two-signed tableaux system for the given logic, which looks and tastes very much like the well-known signed tableaux systems for Classical Logic.

An illustration of the reductive algorithm

Let’s consider Łukasiewicz’s logic L_3 , which has three truth-values (0 , $\frac{1}{2}$ and 1) and four connectives (\neg , \rightarrow , \wedge and \vee) defined over this 3-valued domain. The value 1 is the only designated value in this logic. Let’s also define the function b that maps T for designated values and F for undesignated ones:

$$L_3 = \langle \{0, \frac{1}{2}, 1\}, \{\neg, \rightarrow, \wedge, \vee\}, \{1\} \rangle$$

$$\neg v_1 = 1 - v_1 \qquad (v_1 \rightarrow v_2) = \text{Min}(1, 1 - v_1 + v_2)$$

$$(v_1 \vee v_2) = \text{Max}(v_1, v_2) \qquad (v_1 \wedge v_2) = \text{Min}(v_1, v_2)$$

$$b : \{0, \frac{1}{2}, 1\} \longrightarrow \{T, F\}$$

x	0	$\frac{1}{2}$	1
$b(x)$	F	F	T

Notice that, in the bivalent semantics obtained after composing b with the truth-functional semantics of L_3 , the truth-value 1 is ‘separated’ from both $\frac{1}{2}$ and 0 , but how can we now separate the latter two? The constructive answer goes by finding appropriate *separating connectives*. In this case, the primitive connective \neg , as defined below, solves the problem:

x	0	$\frac{1}{2}$	1
$b(x)$	F	F	T
$\neg x$	1	$\frac{1}{2}$	0
$b(\neg x)$	T	F	F

Indeed, we can now distinguish the truth-values by taking a series of statements about them that, together, provide a unique identification to each value:

$$\begin{aligned}
x = 0 & \quad \text{iff} \quad b(x) = F \text{ and } b(\neg x) = T \\
x = \frac{1}{2} & \quad \text{iff} \quad b(x) = F \text{ and } b(\neg x) = F \\
x = 1 & \quad \text{iff} \quad b(x) = T
\end{aligned} \tag{I}$$

Let $v : \mathbf{For}_{\mathbb{L}_3} \rightarrow \{0, \frac{1}{2}, 1\}$ be an assignment function from the set $\mathbf{For}_{\mathbb{L}_3}$ of well-formed formulas of \mathbb{L}_3 to the set of truth-values of the same logic. Now consider the truth-table of implication, and its negation:

\rightarrow	0	$\frac{1}{2}$	1
0	1	1	1
$\frac{1}{2}$	$\frac{1}{2}$	1	1
1	0	$\frac{1}{2}$	1

$\neg \rightarrow$	0	$\frac{1}{2}$	1
0	0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	0	0
1	1	$\frac{1}{2}$	0

To obtain a sound and complete two-valued semantics for a many-valued logic, it's necessary to write restrictive clauses on bivaluations that describe the behavior of all connectives of the language. By the $\neg \rightarrow$ table above, it's correct to say for instance that:

$$v(\neg(\alpha \rightarrow \beta)) = 1 \text{ iff } v(\alpha) = 1 \text{ and } v(\beta) = 0 \tag{II}$$

Adding the following notations:

$$\begin{aligned}
T : \alpha & \text{ iff } b(v(\alpha)) = T \\
F : \alpha & \text{ iff } b(v(\alpha)) = F
\end{aligned}$$

and combining them with the statements in (I), it follows that:

$$\begin{aligned}
v(\alpha) = 0 & \quad \text{iff} \quad F : \alpha \text{ and } T : \neg\alpha \\
v(\alpha) = \frac{1}{2} & \quad \text{iff} \quad F : \alpha \text{ and } F : \neg\alpha \\
v(\alpha) = 1 & \quad \text{iff} \quad T : \alpha
\end{aligned}$$

In that case we may rewrite sentence (II) as:

$$T : \neg(\alpha \rightarrow \beta) \text{ iff } T : \alpha \text{ and } F : \beta \text{ and } T : \neg\beta$$

Using **FOL** as our meta-language this can alternatively be written as:

$$T : \neg(\alpha \rightarrow \beta) \Leftrightarrow T : \alpha \ \& \ F : \beta \ \& \ T : \neg\beta$$

where \Leftrightarrow and $\&$ represent, respectively, the bi-implication and conjunction from **FOL**. According to the method proposed in [1], this clause originates the following tableau rule:

$$\begin{array}{c} T: \neg(\alpha \rightarrow \beta) \\ | \\ T: \alpha, F: \beta, T: \neg\beta \end{array}$$

For another instructive example, let's consider the truth-table of disjunction:

\vee	0	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
1	1	1	1

Repeating the above process of reducing this into a bivalent version and translating the corresponding clauses using **FOL**, it follows for instance that:

$$F: (\alpha \vee \beta) \Leftrightarrow F: \alpha \& T: \neg\alpha \& F: \beta \& T: \neg\beta \mid F: \alpha \& T: \neg\alpha \& F: \beta \& F: \neg\beta \mid F: \alpha \& F: \neg\alpha \& F: \beta \& T: \neg\beta \mid F: \alpha \& F: \neg\alpha \& F: \beta \& F: \neg\beta$$

where the vertical bar, \mid , represents the disjunction from **FOL**. Again we may extract from that the corresponding tableau rule:

$$\begin{array}{c} F: (\alpha \vee \beta) \\ \swarrow \quad \downarrow \quad \searrow \\ \begin{array}{cccc} F: \alpha, & F: \alpha, & F: \alpha, & F: \alpha, \\ T: \neg\alpha, & T: \neg\alpha, & F: \neg\alpha, & F: \neg\alpha, \\ F: \beta, & F: \beta, & F: \beta, & F: \beta, \\ T: \neg\beta & F: \neg\beta & T: \neg\beta & F: \neg\beta \end{array} \end{array}$$

Now, while this is indeed the clause obtained from the blind application of the method proposed in [1], one can use **FOL** to show that the following is an equivalent formulation of the latter clause, in the presence of the other clauses on the bivaluations:

$$F: (\alpha \vee \beta) \Leftrightarrow F: \alpha \mid F: \beta$$

This of course gives rise to a much simpler (classic-like) tableau rule, namely:

$$\begin{array}{c} F: (\alpha \vee \beta) \\ \swarrow \quad \searrow \\ F: \alpha \quad F: \beta \end{array}$$

In fact, the main clause on bivaluations that guarantees this equivalence between the latter two tableau rules is the one that asserts that:

$$T: \alpha \mid F: \alpha$$

for any formula α . This, again, clearly reveals a shade of bivalence. The tableau rule that corresponds to that clause is the following *branching rule*:

$$\begin{array}{c} \wedge \\ T:\alpha \quad F:\alpha \end{array}$$

that may turn the corresponding tableaux non-analytical.

A sound and complete set of tableaux rules can always be obtained by adding this branching rule to the set of all rules of the form $V : \mathbb{C}(\alpha_1, \dots, \alpha_n)$ and $V : \mathbb{S}(\mathbb{C}(\alpha_1, \dots, \alpha_n))$, where V is one of the two signs, T or F , \mathbb{C} is an arbitrary n -ary connective of the given logic, and \mathbb{S} an arbitrary separating connective among those appropriate for this given logic.

ML and Isabelle

We used the functional programming language ML to automate the axiom extraction process. ML provides us, among other advantages, with an elegant and suggestive syntax, and a very handy compile-time type checking and type inference that guarantees that we never run into unexpected run-time problems with our program, once it is proved correct with respect to the specification. **Isabelle** is a generic theorem-proving environment, also written in ML, in which it's quite simple to create theories with rules and axioms in various deductive systems, and easy to define functions and prove theorems about these systems.

Here's an illustration of part of **Isabelle**'s syntax, taking as example the first two tableau rules obtained in the last section:

```
TNegImp "[\$H, T:A, F:B, T:~B, \$E] ==> [\$H, T:~(A-->B), \$E]"

FDisj   "[| [ \$H, F:A, T:~A, F:B, T:~B, \$E ] ;
          [ \$H, F:A, T:~A, F:B, F:~B, \$E ] ;
          [ \$H, F:A, F:~A, F:B, T:~B, \$E ] ;
          [ \$H, F:A, F:~A, F:B, F:~B, \$E ]
         |] ==> [\$H, F:A|B, \$E]"
```

Here $T:X$ and $F:X$ are (labeled) formulas, $\$H$ and $\$E$ are sequences of such formulas (contexts) that are not involved in the rule, each sequence between square brackets represents a tableau branch, and a collection of branches is delimited by $[|$ and $|]$. The symbol $==>$ is the meta-implication. In **Isabelle**, the application of a rule means that is possible to achieve the goal (branch on the right of the meta-implication), once it's possible to prove the hypothesis (on the left of meta-implication), which is the new goal (or a collection of subgoals).

Now, the **Isabelle** clause corresponding to the third tableau rule from the last section can be written as:

```
FDisjA  "[| [ \$H, F:A, \$E ] ;
          [ \$H, F:B, \$E ]
         |] ==> [\$H, F:A|B, \$E]"
```

The proof that **FDisj** and **FDisjA** are indeed equivalent tableau rules, in the sense that one can prove the other in the presence of the remaining rules present at our theory, can now be done directly inside **Isabelle**.

Our ML logic-reductor program takes as input a definition of a finite-valued logic, such as the logic L_3 presented above, together with an appropriate set of separating connectives for that logic. Those separating connectives, when they exist, can also be found automatically through another procedure presented in this event (cf. [3]). Our program then extracts the axioms corresponding to the two-valued representation of that logic and generates a text file containing the full theory ready to use in **Isabelle**.

References

- [1] Carlos Caleiro, Walter Carnielli, Marcelo E. Coniglio, and João Marcos. Two's company: "The humbug of many logical values". In J.-Y. Béziau, editor, *Logica Universalis*, pages 169–189. Birkhäuser Verlag, Basel, Switzerland, 2005. Preprint available at:
<http://wslc.math.ist.utl.pt/ftp/pub/CaleiroC/05-CCCM-dyadic.pdf>.
- [2] Newton C. A. da Costa, Jean-Yves Béziau, and Otávio A. S. Bueno. Malinowski and Suszko on many-valued logics: on the reduction of many-valuedness to two-valuedness. *Modern Logic*, 3:272–299, 1996.
- [3] Talis Lincoln and João Marcos. Automating the calculation of the degree of expressiveness of finitary algebras. In *Proceedings of the XV EBL*, Brazil, 2008.
- [4] Roman Suszko. Remarks on Łukasiewicz's three-valued logic. *Bulletin of the Section of Logic*, 4:87–90, 1975.
- [5] Roman Suszko. The Fregean Axiom and Polish mathematical logic in the 1920's. *Studia Logica*, 36:373–380, 1977.