Automatic Generation of Proof Tactics for Finite-Valued Logics [*]

João Marcos
LoLITA and DIMAp / UFRN, Brazil, and
Institut für Computersprachen (E1852), TU-Wien, Austria
jmarcos@dimap.ufrn.br

A number of flexible tactic-based logical frameworks are nowadays available that can implement a wide range of mathematical theories using a common higher-order metalanguage. Used as proof assistants, one of the advantages of such powerful systems resides in their responsiveness to extensibility of their reasoning capabilities, being designed over rule-based programming languages that allow the user to build her own 'programs to construct proofs' — the so-called proof tactics.

The present contribution discusses the implementation of an algorithm that generates sound and complete tableau systems for a very inclusive class of sufficiently expressive finite-valued propositional logics, and then illustrates some of the challenges and difficulties related to the algorithmic formation of automated theorem proving tactics for such logics. The procedure on whose implementation we will report is based on a generalized notion of analyticity of proof systems that is intended to guarantee termination of the corresponding automated tactics on what concerns theoremhood in our targeted logics.

Keywords: automated theorem proving, analyticity, rule-based programming, rewriting.

1 Introduction

The early history of the LCF family of theorem provers, first implemented as proof checkers by Robin Milner in the early 70s, based on Dana Scott's Logic for Computable Functions, can be said to be essentially an evolution of Alonzo Church's original proposal of a simple theory of types, developed three decades before (cf. [7]). Arguably, though, their great success as generic logical frameworks for the specification of a wide range of useful mathematical theories within a unified setting came in fact from later developments, namely: (1) the design of an accompanying powerful type-safe functional language that would allow for the needs of the theorem-proving community to be quite naturally expressed; (2) the decision to use a constructive higher-order logic as the underlying metalanguage and to use higher-order unification as the underlying mechanism in which to specify diverse genera of inference systems as theories written in a common framework. The programming language that was designed in that process, `ML`, was intended to give support to the expression of higher-order abstract syntax for the definition and manipulation of object-logics, as well as to advanced pattern-matching capabilities for the definition and manipulation of abstract high-level datatypes. From the point of view of theorem-proving, such flexible datatypes were to allow for the representation of useful objects such as *formulas*, *theorems* or even *proofs*, as well as some strategical operations over those objects, called *tactics*, that represented subgoaling strategies used in the construction of proofs. Higher-order operations for combining tactics and taking stricter control of the result of proof-search procedures were also to be made available as

[*]The author acknowledges financial support by CNPq and CAPES. Dalmo Mendonça and Lucas de Lima, both also financially supported by CNPq, wrote substantial chunks of the implementation code and carefully proof-read the paper, at different stages.

the so-called *tacticals*. A modern heir of the LCF-style family of proof assistants and tactical provers, allowing for both interactive and automated reasoning, is the system `Isabelle` (cf. [6]), which will be utilized in what follows.

A simple and elegant deductive formalism for the specification of proof procedures for both classical and non-classical logics is provided by the refutation-oriented method of *tableaux* (cf. [8]). In the classical bivalent propositional case, the inference rules of (signed or unsigned) tableau systems are based on adequate versions of a *subformula principle* that guarantees that the overall complexity of the involved formulas decreases as tableau rules are applied in the construction of a tableau derivation. The resulting collection of rules, in that case, is said to be *analytic*, and decidability, in general, follows from that. Indeed, analytical proof procedures eliminate in particular the use of the so-called 'cut rule' (which often presupposes some ingenuity from the proof designer) and are very useful for automation as they greatly facilitate the finding of proofs. On the other hand, exactly because they eliminate cut, such procedures render the expression of proof lemmas more difficult, if not outright impossible. However, this limitation can often be negotiated with an additional gain in the speed-up of the corresponding derivations if one considers systems allowing for the so-called 'analytic cuts' (cf. [4]). In one way or another, the objective is to define a rule-based framework for propositional logics in which the termination, with more or less efficiency, of a given theorem-proving task is guaranteed at the outset.

In [1] an algorithm was devised to extract bivalent (in general, non-truth-functional) characterizations for an extensive class of finite-valued propositional logics and then turn those characterizations into classic-like adequate tableau systems for those logics. By a 'bivalent' characterization of a logic, here and in that paper, we mean a collection of interpretation mappings that takes only *two* 'logical' values into consideration, in spite of the many 'algebraic' values that might be used by the logic's original multi-valued truth-functional semantics — the role of the extraction algorithm is to guarantee that both the bivalent and the finite-valued characterization end up determining the same entailment relation. We have used `ML` to implement the mentioned algorithm in [5],[1] and the output of our program is an `Isabelle` theory which can be used for computer-assisted proofs of theorems and derived rules of the corresponding finite-valued logics. Such proof systems, automatically extracted from the sets of truth-tables taken as input by our program, contained a non-eliminable version of the cut rule, and in fact no detailed proof was presented then that analytic cuts, for instance, would suffice for every proof system generated by the above mentioned algorithm. An improved axiom extraction algorithm has recently been proposed in [2], though, for the same class of logics, in which cut *is* eliminable. The latter algorithm has some remarkable features, being based on non-standard complexity measures that are intended to guarantee the analyticity of its output, once one uses such measures to formulate convenient proof strategies. The paper [3] shows in detail how that same axiom extraction mechanism can be extended for *any* finite-valued logic, irrespective of the expressiveness of its original language. The present paper employs an illustration of this procedure to briefly report on the challenges and difficulties related to the implementation of the mentioned novel algorithms, having again as output `Isabelle` theories, but this time enhanced with the automatic formation of cut-free proof tactics for the complete automation of the corresponding theorem-proving tasks.

2 Tableaux

A tableau system is both a proof and a counter-model building procedure based on the construction of refutation trees. A tableau rule is a schematic tree modifier, and its application allows us, given a

[1]Check also `http://tinyurl.com/5cakro`.

branch in which we find instances of the rule's heads, to extend the leaf of this branch by considering all the possibilities provided by the corresponding instances of the rules's daughters. For an example, the classical tableau rules for negation and implication can be represented as:

$$\begin{array}{c} F{:}(\neg\alpha) \\ | \\ T{:}\alpha \end{array} \qquad \begin{array}{c} T{:}(\neg\alpha) \\ | \\ F{:}\alpha \end{array} \qquad \begin{array}{c} F{:}(\alpha\to\beta) \\ | \\ T{:}\alpha \\ F{:}\beta \end{array} \qquad \begin{array}{cc} \multicolumn{2}{c}{T{:}(\alpha\to\beta)} \\ \diagup & \diagdown \\ F{:}\alpha & T{:}\beta \end{array} \tag{1}$$

This means, for instance, that a branch containing a signed formula of the form $F{:}(\alpha \to \beta)$ may be extended by adding in sequence new nodes of the form $T{:}\alpha$ and $F{:}\beta$. Similarly, a branch containing a signed formula of the form $T{:}(\alpha \to \beta)$ may be extended in two different ways, both by adding a new node of the form $F{:}\alpha$ and by adding a new node of the form $T{:}\beta$. The semantic reading of such rules is obvious. The following *closure rule*, syntactically expressing an unobtainable semantic situation, completes the characterization of classical logic:

$$\begin{array}{c} T{:}\alpha \\ F{:}\alpha \\ | \\ * \end{array} \tag{2}$$

The rule is intended to say that a branch that contains an occurrence of the formula α labelled with the sign T and an occurrence of the same formula labelled with the sign F may be said to be *closed*. A whole tree is said to be closed if all of its branches are closed. Now, in case we want to verify the inference of a formula α from a set of premises $\gamma_1, \gamma_2, \ldots, \gamma_n$, using such 2-signed tableau rules for classical logic, what we do is to try and find a closed tableau tree starting from the linear sequence of labelled nodes $T{:}\gamma_1, T{:}\gamma_2, \ldots, T{:}\gamma_n, F{:}\alpha$.

The above tableau system for classical logic respects an obvious *subformula principle* according to which each of the daughters of a non-closure rule are proper subformulas of some of the rule heads, disregarding the corresponding labels. It is easy to see that the following canonical *complexity measure* decreases with rule application:

$$\begin{array}{ll} (\ell 1) & \ell(p) = 0, \text{ where } p \text{ is an atom} \\ (\ell 2) & \ell(\neg\varphi_1) = \ell(\varphi_1) + 1 \\ (\ell 3) & \ell(\varphi_2 \to \varphi_3) = \ell(\varphi_2) + \ell(\varphi_3) + 1 \end{array} \tag{3}$$

Obviously, the closure rule is the only rule applicable to nodes with complexity zero. We say that a proof system is *analytical* if it only allows you to apply a rule when its daughters have smaller complexity than at least one of the corresponding heads. In other words, an analytical proof system is one to which a convenient *proof strategy* has been conveniently associated in such a way that complexity always decreases with rule application. This is obviously the case, without restriction, for the above collection of rules for classical logic, applied in any particular order.

Analyticity guarantees *termination* of a proof procedure, as soon as the application of rules has a completely deterministic result, and becomes otherwise redundant. We say that a tableau tree is terminated when: (T1) all of its branches are closed; (T2) there are open branches and no further rule is applicable without introducing redundancies. In case (T1) we may say the the initial inference has been successfully verified; in case (T2), the open branches allow us to extract all the counter-models to the initial inference.

3 Many-Valued Logics

Many-valued logics deviate from classical logic in allowing larger classes of truth-values, the so-called *designated* and *undesignated* values, to represent, respectively, ‘degrees of truth’ and ‘degrees of falsity’. The rest remains pretty much the same, from the semantical point of view, so that for each assignment of truth-values to the atoms of a given m-ary formula φ there is a unique way of extending that into an interpretation $\widetilde{\varphi}$ of that formula as an m-ary operator over the extended algebra of truth-values.

An algorithm for obtaining analytic 2-signed tableau systems for finite-valued logics was described in [2], and we will illustrate it in what follows, for the instructive case of Łukasiewicz’s four-valued logic Ł$_4$. This logic has 1 as its only designated value and $\frac{2}{3}$, $\frac{1}{3}$ and 0 as its undesignated values. Its connectives $\neg$ and $\to$ are interpreted as operators over $\mathscr{V} = \{1, \frac{2}{3}, \frac{1}{3}, 0\}$ by way of the following definitions and their corresponding truth-tables:

$$\begin{array}{ll} (\text{Ł}_4\neg) & \widetilde{\neg}v = 1 - v \\ (\text{Ł}_4\to) & v_1 \widetilde{\to} v_2 = \mathsf{Min}(1, 1 - v_1 + v_2) \end{array} \tag{4}$$

Now, to produce a classic-like 2-signed tableau system for Ł$_4$ the idea is to associate, in terms of the signs T and F, to each truth-value of this logic a unique *binary print* that distinguishes this truth-value from any other truth-value. Given a collection of truth-values $\mathscr{V}$, its characteristic function $t : \mathscr{V} \to \{T, F\}$ is a mapping that associates T to designated values and F to undesignated values. Binary prints are sequences of unary formulas, called *separating formulas*, that use the latter characteristic functions to distinguish in between truth-values. In the case of Ł$_4$, the following choice of separating formulas can be seen to do the job: $\theta_1(\varphi) = \neg\varphi$ and $\theta_2(\varphi) = \neg\neg(\varphi \to \neg\varphi)$. Consider indeed the table:

v	$t(v)$	$\widetilde{\theta}_1(v)$	$t(\widetilde{\theta}_1(v))$	$\widetilde{\theta}_2(v)$	$t(\widetilde{\theta}_2(v))$
0	F	1	T	1	T
$\frac{1}{3}$	F	$\frac{2}{3}$	F	1	T
$\frac{2}{3}$	F	$\frac{1}{3}$	F	$\frac{2}{3}$	F
1	T	0	F	0	F

(5)

Notice how each truth-value v is associated to a unique triple $\left\langle t(v), t(\widetilde{\theta}_1(v)), t(\widetilde{\theta}_2(v))\right\rangle$.

All rules of the corresponding tableau system will have labelled binary prints as branches. For example, the rules corresponding to (Ł$_4\neg$) are:

$$\begin{tikzcd}[arrows=-]
 & F{:}\neg\alpha \arrow[dl] \arrow[d] \arrow[dr] & & T{:}\neg\alpha \arrow[d] \\
\begin{array}{c} F{:}\alpha \\ F{:}\theta_1(\alpha) \\ T{:}\theta_2(\alpha) \end{array} & \begin{array}{c} F{:}\alpha \\ F{:}\theta_1(\alpha) \\ F{:}\theta_2(\alpha) \end{array} & \begin{array}{c} T{:}\alpha \\ F{:}\theta_1(\alpha) \\ F{:}\theta_2(\alpha) \end{array} & \begin{array}{c} F{:}\alpha \\ T{:}\theta_1(\alpha) \\ T{:}\theta_2(\alpha) \end{array}
\end{tikzcd} \tag{6}$$

An additional set of rules, with heads of the form $S{:}\theta_1(\varphi)$ and $S{:}\theta_2(\varphi)$, with $\varphi = \neg\alpha$ and $\varphi = \alpha \to \beta$, and $S \in \{T, F\}$, is needed to guarantee soundness and completeness of the 2-signed tableau system with respect to the initial finite-valued truth-tabular characterization of the current target logic, Ł$_4$. Here are, by way of an illustration, the rules for $T{:}\theta_2(\alpha \to \beta)$ and $T{:}\theta_1(\neg\alpha)$:

$$\begin{array}{ccc}
\multicolumn{3}{c}{T{:}\theta_2(\alpha \to \beta)} \\
/ & \mid & \backslash \\
F{:}\alpha & T{:}\alpha & T{:}\alpha \\
F{:}\theta_1(\alpha) & F{:}\theta_1(\alpha) & F{:}\theta_1(\alpha) \\
F{:}\theta_2(\alpha) & F{:}\theta_2(\alpha) & F{:}\theta_2(\alpha) \\
F{:}\beta & F{:}\beta & F{:}\beta \\
T{:}\theta_1(\beta) & T{:}\theta_1(\beta) & F{:}\theta_1(\beta) \\
T{:}\theta_2(\beta) & T{:}\theta_2(\beta) & T{:}\theta_2(\beta)
\end{array}
\qquad\qquad
\begin{array}{c}
T{:}\theta_1(\neg\alpha) \\
\mid \\
T{:}\alpha \\
F{:}\theta_1(\alpha) \\
F{:}\theta_2(\alpha)
\end{array} \tag{7}$$

Finally, the set of closure rules contains not only the classical rule (2), but also all other combinations of labelled binary prints that do *not* correspond to possible valuations, according to the truth-tables of Ł$_4$. In the case of this logic, the extra closure rules will then be:

$$\begin{array}{c}
F{:}\alpha \\
T{:}\theta_1(\alpha) \\
F{:}\theta_2(\alpha) \\
\mid \\
\divideontimes
\end{array}
\qquad
\begin{array}{c}
T{:}\alpha \\
F{:}\theta_1(\alpha) \\
T{:}\theta_2(\alpha) \\
\mid \\
\divideontimes
\end{array}
\qquad
\begin{array}{c}
T{:}\alpha \\
T{:}\theta_1(\alpha) \\
F{:}\theta_2(\alpha) \\
\mid \\
\divideontimes
\end{array}
\qquad
\begin{array}{c}
T{:}\alpha \\
T{:}\theta_1(\alpha) \\
T{:}\theta_2(\alpha) \\
\mid \\
\divideontimes
\end{array} \tag{8}$$

A closer look at the above four closure rules will reveal, for instance, that the second and fourth rules, from left to right, only differ in signs for $\theta_1(\alpha)$. Clearly, however, $T{:}\theta_1(\alpha)$ and $F{:}\theta_1(\alpha)$ are the only two possible ways of labelling the formula $\theta_1(\alpha)$. Accordingly, those two rules should give origin to a simpler rule:

$$\begin{array}{c}
T{:}\alpha \\
T{:}\theta_2(\alpha) \\
\mid \\
*
\end{array} \tag{9}$$

A similar approach can in fact be used to simplify other rules of the system, reducing the number of resulting branches and formulas (cf. [5]). Using that idea, for instance, the three branches of the rules $[F{:}\neg]$ and $[T{:}\theta_2 \to]$, in the left halves of (6) and (7), could be simplified into just two branches, each with one node less.

Analyticity for the above system is ensured by enforcing a particular proof strategy that regulates rule applications based on an adequate non-canonical measure of complexity. To implement that strategy, a convenient first step would be to precede definition (3) by a further clause:

$$(\ell 0) \quad \ell(\theta(\varphi)) = \ell(\varphi), \text{ for every separating formula } \theta \tag{10}$$

Observe how now different clauses of the upgraded definition of complexity may potentially apply to the same formula φ, if we look at it as a θ-formula or not. Notice moreover that the new complexity measure is still well-defined as a function, once it is read from (ℓ0) to (ℓ3), in this order. On the other hand, even if we identify a given formula as a θ-formula, there might be, for instance, formulas φ_1 and φ_2 and separating formulas θ_1 and θ_2 such that $\theta_1(\varphi_1) = \varphi = \theta_2(\varphi_2)$. In that case, the rule to be applied should be the one that decreases the complexity the most, and this 'minimality requirement' should also be conveniently internalized in the above definition of the complexity measure (check the details in [2]). For example, the signed formula $T{:}\neg\neg((\alpha\to\beta)\to\neg(\alpha\to\beta))$ might equally well be read as an instance of $T{:}\theta_1(\neg((\alpha\to\beta)\to\neg(\alpha\to\beta)))$ or as an instance of $T{:}\theta_2(\alpha\to\beta)$. The three choices of reading would result in three different extensions of a tableau branch having the initial signed formula as one of its nodes. The first two choices are, according to the right halves of (6) and (7):

Rule $[T{:}\neg]$ is applied:

$$\begin{array}{c} T{:}\neg\neg((\alpha\to\beta)\to\neg(\alpha\to\beta)) \\ | \\ F{:}\neg((\alpha\to\beta)\to\neg(\alpha\to\beta)) \\ T{:}\theta_1(\neg((\alpha\to\beta)\to\neg(\alpha\to\beta))) \\ T{:}\theta_2(\neg((\alpha\to\beta)\to\neg(\alpha\to\beta))) \end{array}$$

Rule $[T{:}\theta_1\neg]$ is applied:

$$\begin{array}{c} T{:}\theta_1(\neg((\alpha\to\beta)\to\neg(\alpha\to\beta))) \\ | \\ T{:}((\alpha\to\beta)\to\neg(\alpha\to\beta)) \\ F{:}\theta_1(((\alpha\to\beta)\to\neg(\alpha\to\beta))) \\ F{:}\theta_2(((\alpha\to\beta)\to\neg(\alpha\to\beta))) \end{array}$$

The third choice corresponds exactly to the rule pictured at the left half of (7). Clearly, it is in this last and more ‘concrete’ choice that the rule application results in less complex formulas. Our tableau strategy should take that into consideration. To guarantee in fact that the new complexity measure given in (3) and (10) continues to be well-defined as a complexity *function*, one also has to guarantee that (10) chooses, for a non-atomic formula φ, the separating formula θ that results in ‘minimally’ complex output branches, when the corresponding rule is applied. Details of this can be found in [2] and [3]. The final tableau strategy of choice is then to be strictly based on such upgraded complexity measure, in order to guarantee analyticity.

Just to illustrate the fundamental relevance of such strategy, if one did not strictly follow it in the above example, one could have opted for the first choice of reading, that of rule $[T{:}\neg]$, and then it could be observed that from the sequence of three resulting daughters, the second would be just the head of the rule reiterated, and the third would be the more complex formula $T{:}\neg\neg((\neg((\alpha\to\beta)\to\neg(\alpha\to\beta)))\to\neg(\neg((\alpha\to\beta)\to\neg(\alpha\to\beta))))$. The tableau building procedure, in such a situation, would not necessarily be terminating.

4 Tactics

Our axiom extraction program takes as input the definition of a many-valued logic and generates a file with a theory ready to use with `Isabelle`. The theory includes the set of all tableau rules for the object logic. In addition, taking advantage of the analytical character of the system defined by the new algorithm, rewrite rules and tactics for automated theorem proving are constructed.

In the output file for the logic Ł$_4$, the rules for $F{:}\neg\alpha$, $T{:}\neg\alpha$, $T{:}\theta_2(\alpha\to\beta)$ and $T{:}\theta_1(\neg\alpha)$ exhibited at the previous section are represented in `Isabelle`’s syntax[2] by:

```
FNeg:    "[| [ $H, F:A0, F:t1(A0), T:t2(A0), $G ] ;
             [ $H, F:A0, F:t1(A0), F:t2(A0), $G ] ;
             [ $H, T:A0, F:t1(A0), F:t2(A0), $G ] |]
               ==> [ $H, F:~(A0), $G ]"

TNeg:    "[| [ $H, F:A0, T:t1(A0), T:t2(A0), $G ] |]
               ==> [ $H, T:~(A0), $G ]"

Tt1Neg:  "[| [ $H, T:A0, F:t1(A0), F:t2(A0), $G ] |]
               ==> [ $H, T:t1(~(A0)), $G ]"
```

[2]The syntax employed here is that of `Isabelle` 2005, and the assisted proofs are done in the command line interface.

```
Tt2Imp:  "[| [ $H, F:A0, F:t1(A0), F:t2(A0), F:A1, T:t1(A1), T:t2(A1), $G ] ;
             [ $H, T:A0, F:t1(A0), F:t2(A0), F:A1, T:t1(A1), T:t2(A1), $G ] ;
             [ $H, T:A0, F:t1(A0), F:t2(A0), F:A1, F:t1(A1), T:t2(A1), $G ] |]
               ==> [ $H, T:t2(A0 --> A1), $G ]"
```

In the above higher-order sequent-style syntax, the symbol $ marks a context, and the meta-implication ==> separates the branch representing the current goal at the right from its subgoals at the left. A closure rule such as the first one from (8), is represented as an axiom of the form:

```
CR1:    "[ $C1, F:A, $C2, T:t1(A), $C3, F:t2(A), $C4]"
```

We further add to the theory some convenient rewrite rules to allow the system to recognize given formulas as instances of separating formulas whenever possible. Only the outermost formulas may be instantiated as θ-formulas, as this rewrite is intended to be followed by a rule application, and there are no rules for formulas with nested θs.

```
t1_def:    "S:~A0          == S:t1(A0)"
t2_def:    "S:~~(A0-->~A0) == S:t2(A0)"
```

Again, to guarantee termination of proofs we must follow a convenient order of instantiation, starting with the rewrite rule that reduce the most the complexity of the formula, namely the one that takes θ_2 into consideration. A tactic for ordered instantiation, in the case of Ł$_4$, may be defined by:

```
val auto_rw = (rewrite_goals_tac [t2_def]) THEN
              (rewrite_goals_tac [t1_def]);
```

where the command `rewrite_goals_tac [t2_def]` rewrites all formulas of the subgoal using the definition of `t2_def`, and similarly for `t1_def`. The tactical `THEN` makes sure that the second line of the above tactic will be executed only after the first one, and this strategy will guarantee the correct order of instantiation in the case where different θ-rules are applicable, in view of the minimality requirement mentioned in the previous section, necessary to guarantee analyticity. Here is an illustration of the use of `auto_rw`:

```
 1. [F:~~(A-->~A), T:~~(A-->~B)]              (* Current state of proof *)
 2. [T:~~((A-->B)-->~(A-->B)), T:~A, F:~~A]

ML> by auto_rw;                               (* Using the tactic *)
 1. [F:t2(A), T:t1(~(A-->~B))]                (* New state of proof *)
 2. [T:t2(A-->B), T:t1(A), F:t1(~A)]
```

We may now use again the native `Isabelle`'s tacticals and construct a tactic for fully automatic theorem proving, by describing a procedure to exhaustively repeat, for every branch of the proof tree, the following steps:

1. instantiate formulas by rewriting (`auto_rw`), then
2. close the branch by applying one of the closure rules or
3. apply another rule of the system, in some suitable order.

The first step will ensure that the right choice will be made when multiple rules are applicable to a formula. Next, the tactic tries to close the branch as soon as possible, to speed-up the process. If closure is not possible at that stage, the next step will try to apply another rule of the system, in the most convenient application order (for instance, postponing branching as much as possible), and start again.

The procedure terminates, due to the analyticity of the system, and at the end either `Isabelle` will deliver a message that says '`No subgoals!`', meaning that the proof has been successfully concluded, or else there will be a list of subgoals — open branches — which are impossible to close and such that all their formulas have complexity zero, so that no further rule is applicable. From those open branches, as usual, counter-models can be assembled.

Extra details will be at hand to be surveyed by the interested reader as the full system is made available on-line, in open source.

References

[1] Carlos Caleiro, Walter Carnielli, Marcelo E. Coniglio & João Marcos (2005): *Two's company: "The humbug of many logical values"*. In: J.-Y. Béziau, editor: *Logica Universalis*. Birkhäuser Verlag, Basel, Switzerland, pp. 169–189. Preprint available at: `http://sqig.math.ist.utl.pt/pub/CaleiroC/05-CCCM-dyadic.pdf`.

[2] Carlos Caleiro & João Marcos (2009): *Classic-like analytic tableaux for finite-valued logics*. In: H. Ono, M. Kanazawa & R. de Queiroz, editors: *Proceedings of the* XVI *Workshop on Logic, Language, Information and Computation* (WoLLIC 2009), held in Tokyo, JP, June 2009, *Lecture Notes in Artificial Intelligence* 5514. Springer, pp. 268–280. Preprint available at: `http://sqig.math.ist.utl.pt/pub/CaleiroC/09-CM-ClATab4FVL.pdf`.

[3] Carlos Caleiro & João Marcos (2009). *A uniform classic-like analytic deductive formalism for finite-valued logics*. Submitted to publication. Preprint available at: `http://sqig.math.ist.utl.pt/pub/CaleiroC/09-CM-UclADF4fvL.pdf`.

[4] Marcello D'Agostino & Marco Mondadori (1994): *The taming of the cut: classical refutations with analytic cut*. *Journal of Logic and Computation* 4(3), pp. 285–319.

[5] João Marcos & Dalmo Mendonça (2009): *Towards fully automated axiom extraction for finite-valued logics*. In: W. Carnielli, M. E. Coniglio & I. M. L. D'Ottaviano, editors: *The Many Sides of Logic*, Studies in Logic, pp. 425–440. College Publications, London. Preprint available at: `http://sqig.math.ist.utl.pt/pub/MarcosJ/08-MM-towards.pdf`.

[6] Tobias Nipkow, Lawrence C. Paulson & Markus Wenzel (2002): *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, *LNCS* 2283. Springer.

[7] Lawrence C. Paulson (1987): *Logic and Computation: Interactive proof with Cambridge LCF*. Cambridge University Press.

[8] Raymond M. Smullyan (1995): *First-Order Logic*. Dover.