Calculational Presentation of Propositional Tableaux

Juan Michelini Álvaro Tasistro
Universidad ORT Uruguay
tasistro@ort.edu.uy, juan@juan.com.uy

We claim that program design techniques, like equational derivation of programs from specifications and separation of concerns, can be applied to obtain results in Mathematics. This means that Mathematics and Programming are not at all alien in their methods, which is a positive and relevant result in connection to teaching Mathematics in Computing, and especially in Software Engineering. We illustrate the point with a detailed case study, namely the introduction of the tableaux method for Propositional Logic. The whole idea derives from Dijkstra's Mathematical Methodology and, according to it, we try to disclose the motivation underlying every step taken.

1 Introduction

We teach Logic in the context of an undergraduate Software Engineering program and are especially interested in presenting proofs that are entirely natural —in the sense that its design is plainly justified- and simple —in the sense that its formal justification is as compact as possible. Therefore we have been led to experimenting with Dijkstra's Mathematical Methodology and Calculational Mathematics.

As a result of our experience, we have observed that Calculational Mathematics is in some of its manifestations quite akin to methodic program derivation, particularly within functional programming. The basic idea behind this latter activity is that one starts with a definition of a function of interest that is in general directly justified but not obviously computable, and by means of convenient transformations ends up obtaining a program computing the function.

As can readily be observed, functional program derivation is indeed a very appropriate field of application of Calculational Mathematics and Mathematical Methodology, since we are of course interested in obtaining proofs of program correctness as natural and compact as possible, as well as in disclosing heuristics as general as they can get. But we claim that, at least in several interesting cases, the converse is also true, i.e. we say that certain mathematical results or methods can be naturally arrived at and proven by employing program derivation principles and techniques.

Our purpose in this work is to give a detailed example illustrating this claim. It consists in a classical subject of (propositional) logic, namely the introduction of the method of analytic tableaux [**?**] including its fundamental property of correctness (i.e. soundness and completeness). We start with a direct, general specification —which in this case consists in the problem of computing the set of all models of a given set of formulæ— and derive from it a functional procedure which can be seen quite naturally to embody the method of tableaux. The proof of correctness of the method that we obtain is significantly simpler than the ones in the literature, e.g. that in [**?**].

Actually the presentation of the method at which we arrive consists of a list of equations, whereas tableaux are normally presented as trees of formulæNow, this gives rise to yet another point in which the principles of methodic program derivation can advantageously be applied. Specifically, we can see the method with the trees as a further refinement of the derived procedure, viz. one employing the trees as a data structure for tracing its execution. Therefore we end up emphasizing another important general

Submitted to:
TFPIE 2014

principle of (program) design, i.e. the *separation of concerns*: we first treat the fundamental logical ideas that make up the method of tableaux and only afterwards consider a (lower-level) representation. In this way we avoid complications that arise in the standard presentations due to the conflation of issues diverse in nature.

We believe that pursuing all these goals is important, particularly in the context of our educational work in programs of Software Engineering. Indeed, general principles of design are fundamental constituents in the education of those students, as must also be methods of mathematical proof. In both these respects, approaches like the one to be employed in this paper can be indeed quite fruitful and illuminating, especially if they show, as we expect, that Mathematics and Programming are, in structure and methods, not at all alien to each other.

The structure of the paper is quite simple: we start with basic definitions in section 2, introduce the main idea and the whole development in section 3, and end up with a discussion in section 4.

2 Basic Notions

The method of tableaux is a proof procedure for both propositional and predicate logic dating back to [1] and [2], and whose ultimate variant (termed *analytic* tableaux) has been introduced in [3]. It is primarily presented as directed towards deriving logically valid formulæ of the language in question, but it can more generally be used to establish that given sets of formulæ are unsatisfiable. Indeed, in the particular case in which such set consists of only one formula, the opposite to this is derived as logically valid. We shall presently develop the method for the propositional case. We start with basic definitions:

Preliminaries Function application is denoted by infix $\cdot$. At one point we consider a function defined along a family of sets, i.e. given a base set A and a collection of sets $\{B\cdot\ a \mid\ a\in\ A\}$, we consider the function space $(x\in\ A)\to\ B\cdot\ x$ whose elements f map $a\in\ A$ to $f\cdot\ a\in\ B\cdot\ a$.

Syntax It is enough to consider the set of connectives $\{\neg\ ,\wedge\}$. Then the set $\mathscr{F}$ of formulæ is defined as usual, starting out from a denumerable set $\mathscr{V}$ of propositional letters $\mathtt{p}$.

Interpretations, Models, Satisfiability Let Γ be a set of formulæ. Let $\mathscr{V}_\Gamma$ be the set of propositional letters occurring in Γ. An *interpretation* of Γ is a mapping $i:\mathscr{V}_\Gamma\to\ \mathsf{Bool}$. We write $\mathscr{I}_\Gamma$ the set of interpretations of Γ [1]. *Truth* (and falsity) of formulæ under an interpretation is defined recursively in the obvious way. A *model* i of a set Γ of formulæ is an interpretation of Γ that makes all its formulæ true. This is written $i\models\Gamma$ and the same notation is used for an interpretation making a formula true. A set of formulæ is *satisfiable* (or *consistent*) iff it has a model. Otherwise it is *unsatisfiable* (*inconsistent*).

3 The Set of All Models

As already said, we intend to set up a proof procedure for establishing if given sets of formulæ are unsatisfiable. It is important to make it clear what is meant by *proof procedure*. In a general sense, this is a set of rules of syntactic manipulation that allows to arrive at certain conclusions of the desired nature. It is of course desirable that the obtainable (we say: *derivable*) conclusions are valid according to the semantics of the language, in which case the proof procedure is termed *sound*. And, in addition, one could also expect that every valid conclusion is derivable in which case the proof procedure is called *complete*. It is a different matter whether the problem of validity, or even derivability, is algorithmically decidable.

[1] On some occasions we shall consider members of $\mathscr{I}_\Gamma$ which are actually defined on supersets of $\mathscr{V}_\Gamma$. We will just not bother to point the fact out.

Now, our approach for providing rules for establishing the unsatisfiability of sets of propositional formulæ consists in solving a more general problem, namely providing rules for "computing" the set of all models of the given set of formulæ. Clearly this solves the original problem in a sound and complete manner in case our method is capable of computing the empty set exactly when the given set of formulæ is unsatisfiable. The reason for the quotes around *computing* is that we will allow infinite sets of formulæ to be manipulated, and this makes it necessary in turn to allow for "computations" of infinite objects, since the sets of models and the models themselves could be infinite. In the sequel we shall speak of the *finite* or the *infinite* case according to the cardinality of the set of formulæ involved. We now start outright with the necessary considerations:

We seem to be interested in a function $\mathscr{M} : (\Gamma \in \mathscr{P}\mathscr{F}) \to \mathscr{I}_\Gamma$.

Now, in virtue of the definitions in section 2, this is given directly by: $\mathscr{M} \cdot \Gamma = \{i \in \mathscr{I}_\Gamma \mid i \models \Gamma\}$.

But this presents the inconvenience that, as a method of computation, it obliges to construct all the interpretations of the set Γ and check each of them against the formulæ in Γ. As already said, we would rather provide rules of syntactic manipulation for arriving at the desired set of models of Γ. Now, these rules are obviously to be applied to Γ itself, which motivates to examine the form of the latter.

First of all, Γ could be empty, which is indeed a plainly uninteresting case. A minute's reflection leads us to the convenience of assuming $\Gamma \neq \emptyset$. Hence, Γ can be written as $\Delta \cup \{\alpha\}$ for some formula α. Now, if we are to manipulate the formulæ in Γ, we had better do that without re-visiting any formula. So we would like to take Δ above in a way such that $\alpha \notin \Delta$. We choose to reflect this in the notation $\Delta \sqcup \alpha = \Delta \cup \{\alpha\}$ with $\alpha \notin \Delta$.

Using this we can rewrite the definition as: $\mathscr{M} \cdot (\Delta \sqcup \alpha) = \{i \in \mathscr{I}_{\Delta \sqcup \alpha} \mid i \models (\Delta \sqcup \alpha)\}$. which immediately rewrites to: $\mathscr{M} \cdot (\Delta \sqcup \alpha) = \{i \in \mathscr{I}_{\Delta \sqcup \alpha} \mid i \models \Delta \wedge i \models \alpha\}$ Now we have no other way out than to deal with the condition $i \models \alpha$. This depends on the form of α, and so we are led to an examination of cases:

$\mathscr{M} \cdot (\Delta \sqcup \alpha \wedge \beta) = \{i \in \mathscr{I}_{\Delta \sqcup \alpha \wedge \beta} \mid i \models \Delta \wedge i \models \alpha \wedge \beta\}$
$\mathscr{M} \cdot (\Delta \sqcup \neg\alpha) = \{i \in \mathscr{I}_{\Delta \sqcup \neg\alpha} \mid i \models \Delta \wedge i \models \neg\alpha\}$
$\mathscr{M} \cdot (\Delta \sqcup \mathtt{p}) = \{i \in \mathscr{I}_{\Delta \sqcup \mathtt{p}} \mid i \models \Delta \wedge i \models \mathtt{p}\}$

Let us begin with the first case:

$\mathscr{M} \cdot (\Delta \sqcup \alpha \wedge \beta)$
$=$ (Definition of $\mathscr{M}$)
$\{i \in \mathscr{I}_{\Delta \sqcup \alpha \wedge \beta} \mid i \models \Delta \wedge i \models \alpha \wedge \beta\}$
$=$ (Truth of $\wedge$)
$\{i \in \mathscr{I}_{\Delta \sqcup \alpha \wedge \beta} \mid i \models \Delta \wedge i \models \alpha \wedge i \models \beta\}$
$=$ (Satisfaction of set of formulæ)
$\{i \in \mathscr{I}_{\Delta \sqcup \alpha \wedge \beta} \mid i \models \Delta \cup \{\alpha, \beta\}\}$
$=$ (Definition of $\mathscr{M}$)
$\mathscr{M} \cdot (\Delta \cup \{\alpha, \beta\})$

Now, what we have got, namely:

$\mathscr{M} \cdot (\Delta \sqcup \alpha \wedge \beta) = \mathscr{M} \cdot (\Delta \cup \{\alpha, \beta\})$

is a good result. In the finite case, it can be interpreted as an equation in a recursive definition of $\mathscr{M}$ where, although we do not necessarily reduce the size of the set, we replace a formula by others of lower complexity, which is just fine. Of course, this does not extend itself to the infinite case, but the equations hold indeed for this case too. And a computational interpretation can also be construed if we think of $\mathscr{M}$ as a process having an effect on an (infinite) data structure. In this case, the effect is produced by the deletion of $\alpha \wedge \beta$ and its replacement by α and β. We will consider this again later.

Let us now turn to the second case: $\mathscr{M} \cdot (\Delta \sqcup \neg\alpha) = \{i \in \mathscr{I}_{\Delta \sqcup \neg\alpha} \mid i \models \Delta \wedge i \models \neg\alpha\}$

If we now apply the definition of $i \models \neg\alpha$ (i.e. what, in accordance to the corresponding step above, we could call Truth of $\neg$) we get: $i \not\models \alpha$ which looks unfortunate because it introduces the negation of the satisfaction relation, instead of just the latter, which is the one that interests us. In face of this situation, we can try pursuing the analysis, i.e. examining the form of α, which leads us to the usual three cases. We examine here the ones corresponding to composite formulæ.

$\mathscr{M} \cdot (\Delta \sqcup \neg(\alpha \wedge \beta))$
$=$ (Definition of $\mathscr{M}$)
$\{i \in \mathscr{I}_{\Delta \sqcup \neg(\alpha \wedge \beta)} \mid i \models \Delta \wedge i \models \neg(\alpha \wedge \beta)\}$
$=$ (de Morgan)
$\{i \in \mathscr{I}_{\Delta \sqcup \neg(\alpha \wedge \beta)} \mid i \models \Delta \wedge (i \models \neg\alpha \vee i \models \neg\beta)\}$
$=$ (Distributing $\wedge$ over $\vee$ and splitting $\vee$ in set comprehension as set union)
$\{i \in \mathscr{I}_{\Delta \sqcup \neg(\alpha \wedge \beta)} \mid i \models \Delta \wedge i \models \neg\alpha\} \cup \{i \in \mathscr{I}_{\Delta \sqcup \neg(\alpha \wedge \beta)} \mid i \models \Delta \wedge i \models \neg\beta\}\}$
$=$ (Satisfaction of set of formulæ)
$\{i \in \mathscr{I}_{\Delta \sqcup \neg(\alpha \wedge \beta)} \mid i \models \Delta \cup \{\neg\alpha\}\} \cup \{i \in \mathscr{I}_{\Delta \sqcup \neg(\alpha \wedge \beta)} \mid i \models \Delta \cup \{\neg\beta\}\}$
$=$ (Definition of $\mathscr{M}$)
$\mathscr{M} \cdot (\Delta \cup \{\neg\alpha\}) \cup \mathscr{M} \cdot (\Delta \cup \{\neg\beta\})$.

The second case is derived readily, and thus we get two more equations:

$\mathscr{M} \cdot (\Delta \sqcup \neg(\alpha \wedge \beta)) = \mathscr{M} \cdot (\Delta \cup \{\neg\alpha\}) \cup \mathscr{M} \cdot (\Delta \cup \{\neg\beta\})$
$\mathscr{M} \cdot (\Delta \sqcup \neg\neg\alpha)) = \mathscr{M} \cdot (\Delta \cup \{\alpha\})$,

which again express $\mathscr{M}$ in terms of itself, acting on sets with less complex formulæ.

Now there remain two cases to consider. First there is the case left out in the local case analysis carried out for negations. This corresponds to formulæ of the form $\neg$p. And, in the course of the main case analysis, we are still short of considering the case of atomic formula p. There seems to be little room in these cases for the kind of rewriting of the function $\mathscr{M}$ that we have performed so far, for we cannot reduce further the complexity of propositional letters. So we are led to think of letters and their negations (i.e. literals) as terminal cases. And indeed they can be so taken, or rather sets of literals can, since for any such set $\mathscr{L}$, its set of models is directly definable. Indeed, if $\mathscr{L}$ contains a contradiction then the set of its models is of course empty and, otherwise, $\mathscr{L}$ is just an interpretation of itself under a different guise. In this way then we conclude our formal manipulations arriving at the equations:

$\mathscr{M} \cdot (\Delta \sqcup \alpha \wedge \beta) = \mathscr{M} \cdot (\Delta \cup \{\alpha, \beta\})$
$\mathscr{M} \cdot (\Delta \sqcup \neg(\alpha \wedge \beta)) = \mathscr{M} \cdot (\Delta \cup \{\neg\alpha\}) \cup \mathscr{M} \cdot (\Delta \cup \{\neg\beta\})$

$$\mathscr{M} \cdot (\Delta \sqcup \neg\neg\alpha)) = \mathscr{M} \cdot (\Delta \cup \{\alpha\})$$
$$\mathscr{M} \cdot \mathscr{L} = \mathscr{L}^*$$

where $\mathscr{L}$ is a set of literals and $\mathscr{L}^*$ is either empty or the trivial rewriting of $\mathscr{L}$ as an interpretation, according to whether $\mathscr{L}$ contains or not a contradiction.

We propose these equations as the proof procedure we were seeking. The manner in which they can actually be used depends on whether we face the finite or the infinite case.

In the finite case, the equations define an algorithm. Then this can simply be executed so as to arrive at the desired result (be it the actual calculation of some or all models or the evidence that the given set of formulæ is unsatisfiable). Such execution can of course be actually implemented in more than one way. One, rather standard, implementation is based upon the observation that the algorithm is (essentially) tail-recursive and that, therefore, its execution can be traced by just maintaining the data structures (sets of formulæ) successively generated. The second equation forces to use a (binary) tree to do this. This representation is straightforward and should require no further clarification, but we will give a detailed description as follows:

Initially there is only one node consisting in the originally given set of formulæ Γ. In the general case, we have a binary tree, each of whose leaves is a set of formulæ. For reasons to be clear immediately, we will assume that some of the leaves are marked. Then choose a unmarked leaf and apply to it one of the equations (there is always at least one equation applicable). In case the equation was one of the three recursive ones, one or two children to the former leaf are generated, containing the sets of formulæ described in the recursive calls. If the equation was the fourth one, the leaf is to be marked. There should be two kinds of mark, in order to distinguish whether the leaf contains or not a contradiction. Let us call these *mark as closed* and *mark as open*, respectively. When all leaves are marked, the models of the original set Γ are those leafs marked as open.

Now, this representation (or notation) is no other than that of ordinary, standard tableaux [2]. Hence our deduction amounts to a proof that tableaux are devices to compute the set of all models of a given set of formulæ, and constitute therefore a sound and complete proof procedure. Actually, they are just structures used for representing the execution of the function $\mathscr{M}$.

The tree structure can of course be optimized [3] by not repeating unchanged elements when creating nodes (i.e. writing only the newly created formulæ at each node and labelling somehow the formula used in creating the new node).

In the infinite case, the equations are to be interpreted as manipulations on the given set which preserve its set of models. The tree structure is also useful for tracing the execution (although notice that all the nodes are necessarily infinite; we in this case conceive as possible the representation of actually arbitrary infinite sets of formulæ, which in general is admissible in the realm of classical mathematics, rather than of computing). At any time, the union of the models of the leaves is the set of models of the original set of formulæ. A leaf is to be considered marked as closed if it contains a contradiction (i.e. any pair of opposite formulæ). As a consequence, an infinite enlargement is possible only if the set admits a model. Therefore, the method is sound and complete.

4 Discussion

Our presentation should be compared with the one in [3]. We believe that there are three main differences.

The one and fundamental is that we make explicit that the method is directed towards the computation

of the set of all models of the given set of formulæ. That ordinary tableaux actually do this is of course absolutely neat in the finite case, and becomes certainly obvious to students when they put the method into practice. As a consequence, it in turn becomes rather frustrating that the standard proofs do not prove precisely this obvious and important fact, at least for the finite case. Now, probably the intent standing behind standard proofs is being able to treat uniformly also the infinite case. This is done quite smoothly for instance in [3] by using the Hintikka sets. But we have also interpreted uniformly our equations as the rules of syntactic manipulation of the tableaux, which enjoy the same fundamental property, namely the preservation of the set of all models, no matter whether we consider the finite or the infinite case.

A second difference is that we formulate the method as a process defined by a set of equations or rewrite rules, relegating the representation of tableaux as trees to a lower-level alternative for implementing or tracing the execution of the process. It could be said that our approach just dodges the complexity of proving the correct management of the actual lower-level structure, which is indeed a usual claim against intents of attacking problems in a way more abstract than the standard one. We claim in response that, as has been pointed out, this is, to a large extent, a routine matter belonging in rather basic computing science. And that, as a general principle, we prefer to separate concerns so that a greater clarification is directed towards the fundamental logical ideas underlying (we would like to say: constituting) the method. One consequence of this separation of concerns is the simplicity of logical principles employed in our derivation, i.e. we need only simple induction on formulæ for proving the function $\mathscr{M}$ to be well-defined. In contrast, the standard proofs of soundness and completeness of tableaux as trees require induction on the structure of the trees besides the induction of formulæ. Also our justification boils down to a single calculational argument instead of the usual soundness and completeness results obtained as separate direct and converse theorems.

Finally, a third, less important, difference is as to the motivation behind the tableaux rules dealing with negations. In [3] they arise from the considerations of *signed* formulæ, i.e. the formulæ used are not the usual ones of propositional logic, but instead these are preceded of a sign T or F which casts them as true or false. So the negation rules appear naturally –as the F rules. Now this way is perfectly natural if the method is to be employed for deriving tautologies or for investigating validity of arguments, but not if we take the more general view of investigating satisfiability of sets of (plain) formulæ. And then the negation rules appear (especially to students) as a bit contrived. What we have done above is to provide one possible justification of its introduction, which is, by the way, tied again to the intent of calculating the set of all models.

We believe that the present approach to tableaux can be indeed fruitful in our Logic courses and we intend to put it into practice in them. To have available a simple proof of completeness of tableaux makes it possible also to extend the result to other calculi by means of proofs of equivalence, and to carry this out in at least the most interesting cases is also of educational interest.

References

[1] Hintikka, K. J. J. *Form and content in quantification theory.* Acta Philosophica Fennica, 8, 7-55, 1955.

[2] Beth, E. W. *The Foundations of Mathematics.* North Holland 1959.

[3] Smullyan R. M. *First-Order Logic.* Dover, 1994. An unabridged, corrected republication of the work first published by Springer-Verlag, New York, 1968.