The Tableau Prover Generator MetTeL2

Dmitry Tishkovsky, Renate A. Schmidt, and Mohammad Khodadadi*

School of Computer Science, The University of Manchester, UK

Abstract. This paper introduces MetTeL^2, a tableau prover generator producing Java code from the specification of a tableau calculus for a logical language. MetTeL^2 is intended to provide an easy to use system for non-technical users and allow technical users to extend the generated implementations.

1 Introduction

MetTeL^2 is a tool for generating tableau provers from specifications of a syntax and a tableau calculus for a logical theory. The syntax and tableau rule specification languages of MetTeL^2 are designed to be as simple as possible for the user and to be as close as possible to the traditional notation used in logic and automated reasoning textbooks. At the moment the syntax specification language is limited to (possibly multi-sorted) propositional languages with finitary connectives and uses a simplified BNF. The tableau calculus specification language covers different types of tableau calculi that fit the traditional representation of tableau rules of the form $X_0/X_1 \mid \cdots \mid X_m$ creating a branching point with m successors in tableau derivations. The X_i denote finite sets of expressions of the given logical theory. X_0 is the set of premises and $X_1, \ldots, X_m$ are the sets of conclusions of the rule. Many labelled semantic tableau calculi for modal, description, hybrid and superintuitionistic logics belong to this paradigm.

MetTeL^2 is complementary to the tableau synthesis framework introduced in [4]. The framework effectively describes a class of logics for which tableau calculi can be automatically generated. This class includes many modal, description, intuitionistic and hybrid logics. The framework provides a theoretical foundation for sound, complete and terminating implementations of tableau procedures for logics from the mentioned class and, in particular, for many logics which can be specified in MetTeL^2. The scope of MetTeL^2 extends however that of tableau calculi derived in the framework and is not limited to semantic or labelled tableau calculi.

The tableau reasoning core of MetTeL^2 is considerably based on the generic prover MetTeL [6], but has been completely reimplemented and several new features have been added. Notable new features are dynamic backtracking and conflict-directed backjumping, as well as ordered forward and backward rewriting, which can be used to perform equality reasoning. There is support for different search strategies. The tableau rule specification language in MetTeL^2 now

* This research is supported by UK EPSRC research grant EP/H043748/1.

allows the specification of rule application priorities thus providing a flexible and simple tool for defining rule selection strategies. To our knowledge, MetTeL^2 is the first system with full support of these techniques for *arbitrary* logical syntax.

The aim of the current implementation is to provide an easy to use prover generator with basic specification languages without sophisticated meta-programming features that might overwhelm non-technical users. For technical users, the generated code consists of a thoroughly designed hierarchy of public Java classes and interfaces that can be extended and integrated with other systems.

2 Application areas and experiences so far

Software to generate code for provers is useful anywhere where automated reasoning is needed. The provers generated by MetTeL^2 output models for satisfiable problems on termination, so can be used for model generation purposes.

With MetTeL^2 a quick implementation of a tableau prover can be obtained and changes can be made without programming a single line of code. Prover generation is useful for obtaining provers for newly defined logics or new combinations of logics. This is particularly pertinent to an area such as multi-agent systems where the models are staggeringly complex. In ongoing work we are using MetTeL^2 in combination with the tableau synthesis framework to develop provers for multi-agent interrogative-epistemic logics [3]. For these logics and related dynamic epistemic logics there are almost no implemented reasoning tools. Therefore being able to generate tableau provers is very useful especially to researchers without the resources or expertise to implement automated reasoning tools themselves.

We have found MetTeL^2 useful for analysing tableau calculi under development whose properties are not known yet. For example, in research conducted for [2] we used MetTeL^2 to determine the refinability or unrefinability of tableau rules for a modal logic with global counting operators. MetTeL^2 can also be used to compare the effectiveness of different sets of tableau rules for the same logic. For example, with minimal effort it is possible to compare the effectiveness of standard tableau calculi with calculi following the KE approach where disjunction is handled by an analytic cut rule and a unit propagation rule (e.g., in terms of proof length, size of produced model, or other derivation statistics which are not tied to particular implementation details).

Concrete case studies we have undertaken with MetTeL^2 include implementing unlabelled tableau calculi for Boolean logic and three-valued Łukasiewicz logic, labelled tableau calculi for standard modal logics and description logics, and internalised tableau calculi for hybrid and description logics. We used MetTeL^2 to implement the first tableau decision procedure for $\mathcal{ALBO}^{\mathsf{id}}$, a description logic with the same expressive power as the two-variable fragment of first-order logic. Some of the specifications are available from the MetTeL^2 website http://www.mettel-prover.org.

3 The implemented system

The parser for the specification of the user-defined logical language is implemented using the ANTLR parser generator. The specification is parsed and internally represented as an abstract syntax tree (AST). The internal ANTLR format for the AST is avoided for performance purposes. The created AST is passed to the generator class which processes the AST and produces the following files: (i) a hierarchy of JAVA classes representing the user-defined logical language, (ii) an object factory class managing the creation of the language classes, (iii) classes representing substitution, replacement, and rewrite orderings, (iv) an ANTLR grammar file for generating a parser of the user-specified language and the tableau language, (v) a main class for the prover parsing command line options and initiating the tableau derivation process, and (vi) JUNIT test classes for testing the parsers and testing the correctness of tableau derivations. The generated JAVA classes for syntax representation and algorithms for rule application follow the same paradigm as the generic prover METTEL [6].

METTEL^2 implements two general techniques for reducing the search space in tableau derivations: dynamic backtracking and conflict directed backjumping. Dynamic backtracking avoids repeating the same rule applications in parallel branches by keeping track of rule applications common to the branches. Conflict-directed backjumping derives conflict sets of expressions from a derivation. This causes branches with the same conflict sets to be discarded. Since METTEL^2 is a prover generator, dynamic backtracking and backjumping needed to be represented and implemented in a generic way, completely independent of any specific logical language and tableau rules.

The provers generated by METTEL^2 come with support for ordered backward and forward rewriting with respect to equality expressions appearing in the current branch. In the language specification, equality expressions can be identified with built-in keywords. Each JAVA class representing a tableau node keeps a rewrite relation completed with respect to all equality expressions appearing in a branch. Once an equality expression is added within a tableau node, backward rewriting is applied. This means the rewrite relation is rebuilt with respect to the newly added equality, and all expressions of the node are rewritten with respect to the rewrite relation. Forward rewriting (with respect to the current rewrite relation) is applied to all new expressions added to the branch during the derivation.

The core tableau engine of METTEL^2 provides various ways for controlling derivations. The default search strategy is depth-first left-to-right search, which is implemented as a MettelSimpleLIFOBranchSelectionStrategy request to the MettelSimpleTableauManager. The MettelSimpleTableauManager object manages tableau branches at the very top level: it stores branches for expansion and selects them in accordance with the specified branch selection strategy. Breadth-first search is implemented as a MettelSimpleFIFOBranchSelectionStrategy request and can be used after a small modification in the generated JAVA code. In future this will be configurable at the generation stage. A user can also implement their own search strategy and pass it to the MettelSimpleTableauManager.

The rule selection strategy can be controlled by specifying priority values for the rules in the tableau calculus specification. The rule selection algorithm checks the applicability of rules and returns a rule that can be applied to some expressions on the current branch according to rule priority values. If several rules are applicable preference is given to those which have smaller priority values. Rules within each priority group are checked for applicability sequentially. To ensure fairness for rules within the same priority group all rules in the group are checked for applicability an equal number of times. Again the user could implement their own rule selection strategy and modify the generated code.

To achieve termination for semantic tableau approaches some form of blocking is usually necessary. To generate a prover with blocking the user can specify a blocking rule similar to the unrestricted blocking rule from [5] as one of the rules of the tableau calculus. If the definition of the rule involves equality operators then rewriting is triggered (see above). Based on the results in [4, 5], the blocking rule can be used to achieve termination for logics with the finite model property. The first of the two termination conditions in [4, 5] is automatically true because the generated provers are equipped with ordered rewriting. The second termination condition can be satisfied by using appropriate priority values for tableau rules of the tableau calculus. By varying the specification of the blocking rule it is possible to perform blocking more selectively [1].

4 Conclusion

MetTeL^2 can be downloaded from http://www.mettel-prover.org. A web-interface is provided, where a user can input their specifications in syntax aware textareas and generate provers. The user can either download the generated prover or directly run it in the web-interface.

References

1. M. Khodadadi, R. A. Schmidt, and D. Tishkovsky. An abstract tableau calculus for the description logic $\mathcal{SHOI}$ using unrestricted blocking and rewriting. In *Proc. DL'12*, vol. 846 of *CEUR Workshop Proceedings*, pp. 224–234, 2012.
2. M. Khodadadi, R. A. Schmidt, D. Tishkovsky, and M. Zawidzki. Terminating tableau calculi for modal logic K with global counting operators. Manuscript, http://www.mettel-prover.org/papers/KEn12.pdf, 2012.
3. S. Minica, M. Khodadadi, R. A. Schmidt, and D. Tishkovsky. Synthesising and implementing tableau calculi for interrogative epistemic logics. In *Proc. PAAR'12*, pp. 109–123, 2012.
4. R. A. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi. *Log. Methods Comput. Sci.*, 7(2:6):1–32, 2011.
5. R. A. Schmidt and D. Tishkovsky. Using tableau to decide description logics with full role negation and identity. Manuscript, http://www.mettel-prover.org/papers/ALBOid.pdf, 2011.
6. D. Tishkovsky, R. A. Schmidt, and M. Khodadadi. MetTeL: A tableau prover with logic-independent inference engine. In *Proc. TABLEAUX'11*, vol. 6793 of *LNCS*, pp. 242–247. Springer, 2011.