An implementation of a tableau theorem prover for modal logics

Zhen Li, University of Manchester, liz@cs.man.ac.uk

1 Introduction

Modal logics are well known as a formal way of handling notions of knowledge, believe and time. Many efforts have been made to develop deduction systems for modal logics [13, 14, 2, 4]. Tableau algorithms are capable of dealing with both classical and non-classical logics which gives them a wide scope of possible applications. Semantics-based tableau algorithms are intuitive and can generate more understandable proofs compared with resolution and connection-based methods. Recent research has shown that certain optimization techniques help increase the performance of tableau algorithms significantly.

Tableau algorithms have been implemented in many theorem provers for modal logics or its variants, description logics. Well-known tableau provers include TABLEAUX [5], the Logics Workbench [10], Lotrec [6], the Tableaux Work Bench [1], FaCT [11], DLP [15] and RACER [9]. TABLEAUX, Lotrec, LWB and the Tableaux Work Bench are general theorem provers which can handle a wide range of modal logics. Essentially they are experimental tools with few optimizations. FaCT, DLP and RACER are highly optimized theorem provers. They utilize heuristics and search strategies to speed up the derivations in description logics and the corresponding modal logics.

In this paper, we will discuss the implementation issues of a modal logic theorem prover called MLTP (short for Modal Logic Tableaux Prover). We intended to build a prover that has fairly good performance comparable with state-of-art provers and also has the ability to deal with various of modal logics. MLTP is based on a labelled tableau algorithm and uses structural rules to deal with extensions of the modal logic K. Currently MLTP can handle modal logics with the axioms K, T, D, B, 4 and 5. A naive loop checking mechanism is incorporated for logics that could cause non-termination. Optimization techniques have been incorporated into MLTP. Effects of these techniques are analyzed based on performance data from empirical tests. In MLTP it is possible for the user to select a subset of the expansion rules that have been implemented in the system. The order of the application of the rules is specifiable as well.

2 Modal logics

Modal logics extend propositional logic with two operators, the necessity ($\Box$) and the possibility ($\Diamond$) operators. These two operators are not truth-functional, therefore the well-known Kripke semantics of modal logics is based on relational structure over possible worlds,also called frames.

An axiom is a formula that is valid in a given modal logic. Different modal logics are obtained from modal logic K by addition of a combination of extra axioms. Well-known axioms include the axioms T, 4, 5, B, and D. All these axioms are *Sahlqvist formulae* [3]. That means, given an axiom, there exists a first-order correspondence property which describes a class of frames in which the axiom is valid. Table 1 gives the axioms, their relational correspondence properties and the first-order logic definitions.

Axiom	Property	FOL definition
T: $\Box p \to p$	reflexivity	$\forall x.(x,x) \in R$
D: $\Box p \to \Diamond p$	seriality	$\forall x \exists y.(x,y) \in R$
B: $p \to \Box\Diamond p$	symmetry	$\forall x,y.(x,y) \in R \to (y,x) \in R$
4: $\Box p \to \Box\Box p$	transitivity	$\forall x,y,z.((x,y) \in R \land (y,z) \in R) \to (x,z) \in R$
5: $\Diamond p \to \Box\Diamond p$	euclideanness	$\forall x,y,z.((x,y) \in R \land (x,z) \in R) \to (y,z) \in R$

Table 1: Axioms and their relational correspondence properties

α-rule	β-rule	ν-rule	π-rule
$\dfrac{w:\alpha}{\begin{array}{c} w:\alpha_1 \\ w:\alpha_2 \end{array}}$	$\dfrac{w:\beta}{w:\beta_1 \mid w:\beta_2}$	$\dfrac{\begin{array}{c} w:\nu \\ (w,w') \in R \end{array}}{w':\nu_0}$	$\dfrac{w:\pi}{\begin{array}{c} w^*:\pi_0 \\ (w,w^*) \in R \end{array}}$

Table 2: Tableau expansion rules

3 Tableau Algorithm

Labelled tableau systems [7, 8] are widely used and can be viewed as procedures for testing the satisfiability of modal logic formulae. A labelled tableau system consists of labelled formulae. A labelled formula has the form $w : \phi$ where w denotes a possible world and ϕ is a modal formula. The semantic meaning is that ϕ is true in the world w. A labelled tableau is a tree that is constructed by applying a set of expansion rules. A branch of a labelled tableau is *closed* if it contains some labelled formula $w : \phi$ and also contains $w : \neg\phi$. If such pair is not found, then the branch is *open*. A label w is *new* on a branch if there is no labelled formula $w : \phi$ on this branch so far.

In [7], Fitting gave a classification of the expansion rules. According to his definition, α denotes conjunction formulae, β denotes disjunction formulae, ν denotes necessity formulae and π possibility formulae. The expansion rules are listed in Table 2, where w^* in the π-rule is new on the branch. These rules handle the four kinds of formulae (α, β, ν and π) in different ways in accordance with their semantics.

To deal with the extra modal axioms, we use the *structural rules* shown in Table 3, where w^* in the rule for seriality is new on the branch. Table 1 gives the relational properties corresponding to each of the well-known axioms T, D, 4, 5 and B. Essentially the corresponding structural rules complete the partial frame constructed thus far in the derivation so that the frame satisfies the relational correspondence properties. For instance, to apply the structural rule of euclideanness we need to go through the frame, and if there are two worlds w, w' with a common predecessor then an arrow from w to w' is added to the frame.

Derivation involving some of the axioms (e.g. 5 and B) require exponential space, because all the worlds that have been explored need to be preserved.

4 Implementation

System description. Existing tableau theorems are implemented in various languages (TABLEAUX is implemented in Prolog, LWB in C++, FACT and

Reflexivity	Transitivity	Euclideanness	Symmetry	Seriality
$\dfrac{\quad}{(w,w) \in R}$	$\dfrac{\begin{array}{c} (w,w') \in R \\ (w',w'') \in R \end{array}}{(w,w'') \in R}$	$\dfrac{\begin{array}{c} (w,w') \in R \\ (w,w'') \in R \end{array}}{(w',w'') \in R}$	$\dfrac{(w,w') \in R}{(w',w) \in R}$	$\dfrac{\quad}{(w,w^*) \in R}$

Table 3: Structural rules

RACER in LISP, DLP in ML, Lotrec in Java and TWB in OCaml). MLTP is implemented in the C programming language because C outperforms other languages like Prolog, LISP and Java in terms of efficiency.

MLTP consists of four modules: the parser, the preprocessor, the inference engine and the output module. The parser takes a plain text file as input. The input file includes general information about the problem, the specification of the axioms, and the input problem to be tested. The parser sets up the data structures holding this information. The preprocessor then simplifies and transforms the input formula into a negation normal form. In the inference engine, expansion rules and structural rules are repeatedly applied until the deduction terminates. Loop checking is performed to prevent infinite computations. Finally the recorded derivations performed during the search can be retrieved and output in an understandable format according to the demands of the user.

Data structures. In MLTP a formula is stored in a tree structure. Hence MLTP can deal with extremely long formulae limited only by the memory space available. The tree structure also makes it easy to extract subformulae, which is in fact the most common operation during derivations.

To have flexibility, the expansion rules are formalized into a general data structure. A number of fields are incorporated to denote the numerators and the denominators of the rules, which makes it possible to let the user define their own expansion rules. Stored in an array, these rules are accessed by the inference engine in an order which is also modifiable by the users.

A branch in the derivation tree is implemented by a *derivation queue*. The labelled formulae and the worlds and arrows in the frame are stored in a data structure called *inference node*. An inference node also holds the information about its premises and the rule applied. A FILO stack is used to implement backtracking over disjunctive branches.

Inference engine. The inference engine employs a depth-first, left to right search strategy. To reduce the loss of efficiency caused by function calls, MLTP is implemented iteratively rather than recursively. Expansion rules are used one by one in a specified order and are applied repeatedly to the applicable inference nodes in the derivation queue. In the inference loop the structural rules for T, D, 4, 5 and B are applied after a new world is generated. Clash checking is performed after each inference step. Loop checking is done before each application of the π-rule in order to detect two identical worlds and thereby prevent the generation of a potentially infinite branch in the model. It is important that when dealing with modal logics which could cause infinite branches, the π-rule is applied after the other expansion rules, because otherwise the prover could judge a branch as a loop by mistake (e.g. $\Box\Diamond p \wedge \Box(\neg p \wedge q)$ in $KT4$).

Optimization. Many techniques and strategies have been proposed to improve the efficiency of inference for modal logic. The optimization techniques implemented in MLTP are *unit propagation*, *simplification* and *backjumping*.

Unit propagation, also called boolean constraint propagation, is a technique used to minimize non-deterministic expansion steps (branching). Branching exponentially increases the search space. Essentially unit propagation performs clash detection before branching and prunes the branch that is found to be inconsistent.

Simplification aims to minimize the size of the problem formulae and therefore reduce the size of the search space. For this purpose, a set of simplification rules are applied to the formulae. Most of the rules correspond to tautologies of propositional logic, for example $A \vee A$ is replaced by A. Others are related to modal operators, for example $\Box\top$ is reduced to $\top$.

Backjumping is a clever form of backtracking, also known as dependency-directed backtracking. The derivation for a complex problem usually needs to explore a number of branches. It often happens that the closures of several branches are caused by the same unsatisfiability. Backjumping works by detecting this situation after expanding one of the branches and bypassing the others.

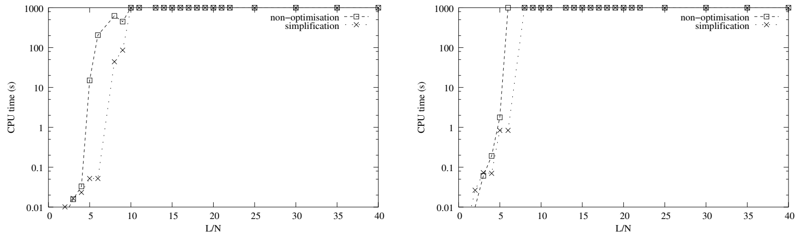
5 Evaluation

Empirical tests have been done to evaluate the performance of the MLTP system and the effectiveness of the optimization techniques. The tests were performed on a desktop with Pentium 650MHz CPU and 128MB main memory. Random problem sets from [12] were used in these tests. The results of the tests are shown as percentile graphs. The 50th percentile is equivalent to the median and the 100th percentile is the worst case.

The results of the tests in Figures 1, 2 and 3 have shown that the three optimization techniques implemented in MLTP all contribute to the improvement of the performance. The impact of unit propagation and simplification is greater on simple problems while backjumping improves the performance considerably on complex problems (50th percentile).

6 Future work

Our plans are to extend MLTP to deal with multi-modal logics and the common knowledge operator in order to reason about the knowledge and belief of multiple agents. In this application, it would be useful if MLTP could also reason about actions and plans of agents. We also intend to study and incorporate more optimization techniques and efficient data structures to speed up the derivation.



a) 50th percentile CPU time　　b) 100th percentile CPU time

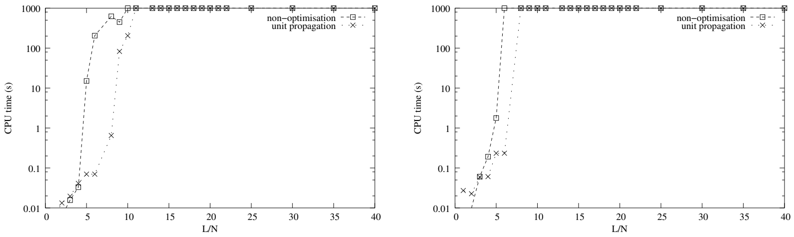
Figure 1: Results of testing simplification

References

[1] P. Abate and R. Goré. The tableaux work bench. In *Proc. TABLEAUX 2003*, volume 2796 of *LNAI*, pages 230–236. Springer, 2003.

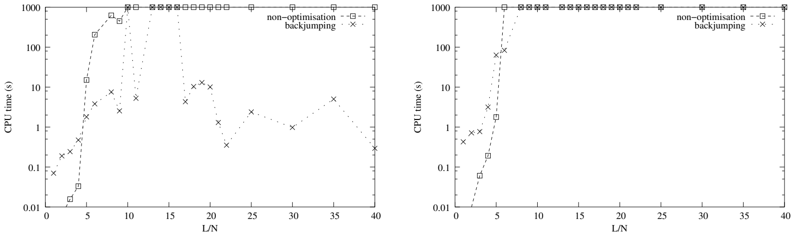
[2] M. Baldoni. *Normal Multimodal Logics: Automatic Deduction and Logic Programming Extension.* PhD thesis, Univ. degli Studi di Torino, Italy, 1998.

[3] P. Blackburn, M. Rijke, and Y. Venema. *Modal Logic.* Cambridge University Press, 2001.



a) 50th percentile CPU time b) 100th percentile CPU time

Figure 2: Results of testing unit propagation



a) 50th percentile CPU time b) 100th percentile CPU time

Figure 3: Results of testing backjumping

[4] M. A. Castilho, L. F. Cerro, O. Gasquet, and A. Herzig. Modal tableaux with propagation rules and structural rules. *Fundamenta Informaticae*, 32(3-4):281–297, 1997.

[5] L. Catach. TABLEAUX: A general theorem prover for modal logics. *Journal of Automated Reasoning*, 7:489–510, 1991.

[6] L. F. Cerro, D. Fauthoux, O. Gasquet, F. Massacci, A. Herzig, and D. Longin. Lotrec: Generic tableau prover for modal and description logic. In *Proc. IJCAR 2001*, volume 2083 of *LNAI*, pages 453–458. Springer, 2001.

[7] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel Publishing Co., 1983.

[8] R. Goré. Tableau methods for modal and temporal logics. In *Handbook of tableau methods*, pages 297–396. Kluwer, 1999.

[9] V. Haarslev and R. Möller. RACER system description. In *Proc. IJCAR 2001*, volume 2083 of *LNAI*, pages 701–706. Springer, 2001.

[10] A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfried. The Logics Workbench LWB: A snapshot. *Euronmath Bulletin*, 2(1):177–186, 1996.

[11] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, Univ. of Manchester, 1997.

[12] U. Hustadt and R. A. Schmidt. Simplification and backjumping in modal tableau. In *Proc. TABLEAUX 1998*, volume 1397 of *LNCS*, pages 187–201. Springer, 1998.

[13] U. Hustadt and R. A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In *Proc. TABLEAUX 2000*, volume 1847 of *LNAI*, pages 67–71. Springer, 2000.

[14] F. Massacci. Single step tableaux for modal logics. *Journal of Automated Reasoning*, 24:319–364, 2000.

[15] P. F. Patel-Schneider. DLP system description. In *Proc. Description Logics*, volume 98 of *CEUR Workshop Proceedings*, pages 87–89, 1998.