Simplification and backjumping in modal tableau

Ullrich Hustadt and Renate A. Schmidt

Department of Computing, Manchester Metropolitan University,
Chester Street, Manchester M1 5GD, United Kingdom
{U.Hustadt, R.A.Schmidt}@doc.mmu.ac.uk

Abstract. This paper is concerned with various schemes for enhancing the performance of modal tableau procedures. It discusses techniques and strategies for dealing with the nondeterminism in tableau calculi, as well as simplification and backjumping. Benchmark results obtained with randomly generated modal formulae show the effect of combinations of different schemes.

1 Introduction

Usually the literature on theorem provers for modal logic confines itself to a description of the underlying calculus and methodology. Sometimes the description is accompanied with a consideration of the worst-case complexity of an algorithm based on the presented calculus or a small collection of benchmark results. Problems arising when implementing modal theorem provers and also considerations concerning optimisations towards increased efficiency have received much less attention, which, of course, is typical in a field under development. Sometimes the description of the theorem prover mentions some simplification rules [2] or discusses the use of structure sharing and use-checking [9]. Less attention has been paid to an empirical evaluation of the influence of such optimisations. But, recent work by Giunchiglia and Sebastiani [7], Horrocks [10], and Hustadt and Schmidt [11,12] has put increased emphasis on optimisation techniques for modal decision procedures.

In this paper we discuss various known techniques and strategies [7,9,10] and study their usefulness by experiments. We focus on two techniques which we think are instrumental for increased efficiency, namely *simplification* and *backjumping*. These techniques are well-known from other areas of computer science, like automated theorem proving in propositional logic, constraint solving and search. Our exposition concentrates on a modal **KE** tableau [3,4], but applies equally to standard tableau [6,8].

The paper is organised as follows. Section 2 recalls some basic notions and describes a standard tableau calculus for the basic modal logic **K**. Section 3 discusses the problems of dealing with the nondeterminism in tableau calculi. In Sections 4 and 5 we describe a simplification technique for modal tableau procedure and discuss the importance of backjumping and dependency-directed backtracking. Finally, Section 6 describes experiments which illustrate the effects in practice of the different optimisation techniques.

2 Basic notions

By definition, a *formula* of the basic modal logic **K** is a boolean combination of propositional and modal atoms. A *modal atom* is an expression of the form $\Box\psi$ where ψ is a formula of **K**. A *modal literal* is either a modal atom or its negation. We assume that $\phi \vee \psi$ is the abbreviation for $\neg(\neg\phi \wedge \neg\psi)$ and $\Diamond\psi$ for $\neg\Box\neg\psi$. $\top$ denotes a constant true proposition and $\bot$ a constant false proposition. $\overline{\phi}$ denotes the complementary formula of ϕ, for example $\overline{\neg p} = p$ and $\overline{\Box p} = \Diamond\neg p$. The following notation will be used: ϕ and ψ denote modal formulae, C and D denote multisets of modal formulae, $C; D$ denotes the multiset-union $C \cup D$, $C; \phi$ denotes $C \cup \{\phi\}$, and $\Box C$ denotes the multiset $\{\Box\phi \mid \phi \in C\}$.

$$(\wedge)\frac{C; \phi \wedge \psi}{C; \phi; \psi} \qquad (\vee)\frac{C; \phi \vee \psi}{C; \phi \mid C; \psi} \qquad (\neg)\frac{C; \neg\neg\phi}{C; \phi}$$

$$(\bot)\frac{C; \neg\phi; \phi}{\{\bot\}} \qquad (\theta)\frac{C; D}{C} \qquad (K)\frac{\Box C; \Diamond\phi}{C; \phi}$$

Fig. 1. Tableau rules for basic modal logic

Figure 1 describes the rules of a standard tableau system for basic modal logic as given by Goré [8] with a slight modification of the $(\bot)$ rule. The *numerator* of any rule is a set of formulae of which one or two are distinguished. For example, the distinguished formula of the numerator of the rule $(\wedge)$ is $\phi \wedge \psi$. Distinguished formulae are called *principal formulae*. The *denominator* of any rule is a list of sets of formulae. The rules $(\wedge)$, $(\vee)$, $(\neg)$, and (K) are the *elimination rules*, (θ) is the *thinning rule*, and $(\bot)$ is the *closure rule*. A *tableau* for a set C of formulae is a finite tree labelled with finite sets of modal formulae whose root is labelled with C. A tableau rule with numerator C is *applicable* to a node labelled with C. The steps for extending a tableau are the following.

1. Choose a leaf node N labelled with C, a rule R which is applicable to C, and a set of principal formulae D.
2. If D are the principal formulae of R, having k denominators $D_1, \ldots, D_k$, then create k successor nodes for N labelled with $D_1, \ldots, D_k$, respectively.

A branch in a tableau is *closed* if its end node contains $\bot$, otherwise it is *open*. A tableau is *closed* if all its branches are closed. The tableau calculus for basic modal logic by Fitting [6] uses the following refinement of the thinning rule (θ), called the *branch modification rule*:

$$(BM)\frac{D; \Box C; \Diamond\phi}{\Box C; \Diamond\phi} \quad \text{where } D \text{ contains no } \Box\text{-formula.}$$

3 Nondeterminism in tableau calculi

As usual when implementing a set of proof rules in a deterministic algorithm an important issue is how to deal with the nondeterminism present in the calculus. Several choices need to be made in extending a tableau, namely, which leaf node to continue with, which rule to apply, and to which principal formulae. The choices are don't care nondeterministic and don't know nondeterministic. A *don't care nondeterministic* choice is an arbitrary choice of one among multiple possible continuations for a computation which render the same result, for example, the nondeterminism of the $(\wedge)$ and $(\neg)$ rules, while a *don't know nondeterministic* choice is a choice among multiple possible continuations which do not necessarily render the same result, for example, the nondeterminism of the (K) rule. Techniques and strategies are required for dealing with nondeterministic choices, in a way that ensures soundness and completeness while delivering good performance.

For the completeness of the calculus it is sufficient to assume that all choices are don't know nondeterministic. However, a deterministic algorithm based on the assumption that all choices are don't know nondeterministic will be hopelessly inefficient, since it has to consider all possible continuations of all the don't care nondeterministic choices where the consideration of only one would suffice. Therefore, a clear distinction between don't care and don't know nondeterministic choices is necessary.

Even if we know which choices are don't care and which are don't know nondeterministic, the algorithm has to use a particular strategy to make these choices. It is well known for propositional decision procedures that different strategies lead to vastly different computational performance. For modal decision procedures such a body of knowledge does not seem to exist.

One of the problems is that applications of the (BM) and (K) rules have to be done don't know nondeterministically. Consider the tableau $\mathbf{T}_1$ given by the single node $\{\Box p, \Diamond p, \Diamond\neg p\}$. If we apply (BM) and (K) to the formula $\Diamond p$, we obtain $\mathbf{T}_2$ which cannot be closed. However, if we apply (BM) and (K) to the formula $\Diamond\neg p$ we obtain $\mathbf{T}_3$ which can be closed by an application of the closure rule. Within the framework of Fitting's tableau calculus it is impossible to avoid this don't know nondeterminism. Whenever there is more than one $\Diamond$-formula on a branch we systematically have to consider the application of the (K) rule to each of them (which can be done using backtracking).

$$\mathbf{T}_2\text{:}\quad \begin{array}{c} \{\Box p, \Diamond p, \Diamond\neg p\} \\ {\scriptstyle (BM),(K)}\downarrow \\ \{p, p\} \end{array} \qquad\qquad \mathbf{T}_3\text{:}\quad \begin{array}{c} \{\Box p, \Diamond p, \Diamond\neg p\} \\ {\scriptstyle (BM),(K)}\downarrow \\ \{p, \neg p\} \end{array}$$

It is not only relevant to which formula we apply the diamond elimination rule, but also at which state of our computation we do so, since the preceding application of the branch modification rule might delete information which is relevant for finding a closed tableau. Consider the tableau $\mathbf{T}_4$ given by the single node $\{\Box p \vee \Box q, \Diamond(\neg p \wedge \neg q)\}$. We can apply (BM) and (K) to the second formula in the tableau and obtain tableau $\mathbf{T}_5$.

$$\begin{array}{c}
\{\Box p \vee \Box q, \Diamond(\neg p \wedge \neg q)\} \\
{\scriptstyle (BM),(K)}\big\downarrow \\
\{\neg p \wedge \neg q\}
\end{array}$$

It is not possible to obtain a closed tableau from $\mathbf{T}_5$. However, if we start by applying $(\vee)$ to the first formula in $\mathbf{T}_4$ and then (BM) and (K) to the second formula, we obtain the closed tableau $\mathbf{T}_6$.

$$\begin{tikzcd}[column sep=small]
 & \{\Box p \vee \Box q, \Diamond(\neg p \wedge \neg q)\} \arrow[dl, "(\vee)"] \arrow[dr, "(\vee)"'] & \\
\{\Box p, \Diamond(\neg p \wedge \neg q)\} \arrow[d, "{(BM),(K)}"'] & & \{\Box q, \Diamond(\neg p \wedge \neg q)\} \arrow[d, "{(BM),(K)}"'] \\
\{p, \neg p \wedge \neg q\} \arrow[d, "(\wedge)"'] & & \{q, \neg p \wedge \neg q\} \arrow[d, "(\wedge)"'] \\
\{p, \neg p, \neg q\} \arrow[d, "(\bot)"'] & & \{q, \neg p, \neg q\} \arrow[d, "(\bot)"'] \\
\{\bot\} & & \{\bot\}
\end{tikzcd}$$

There are several solutions how a deterministic algorithm can deal with this problem. The most obvious solution is to use backtracking. The states when branch modification is applicable are remembered, and we can apply the rule at the current state of our computation or we can delay the application. We explore one possibility and restore the current state of the computation unless a closed tableau was found.

The nondeterminism of the branch modification rule can be avoided altogether. We can delay the application of the rule until we are sure that we can find a closed tableau whenever it exists for the formula under consideration. This is the case, for example, if we delay the application of the branch modification rule until no further elimination rules for the boolean connectives are applicable on the current branch. The knowledge representation system $\mathcal{KRIS}$ uses this solution [1].

There is a trade-off between these two solutions, and we are not aware of any theoretical or empirical analysis of this trade-off. The performance of a tableau-based theorem prover depends on our ability to generate as few branches in a tableau as possible, and to close branches as soon as possible. So, it is desirable to apply the diamond elimination and branch modification rules as early as possible. However, if we fail to close the tableau and we have to go back to an earlier state of the computation, then a lot of computational effort has been wasted.

There are also solutions between the two extremes. For example, the modal theorem prover KSAT [7] proceeds as follows. Instead of delaying the application of the diamond elimination rule until no further applications of the elimination rules for boolean connectives are possible, it systematically applies all possible applications of the diamond elimination rule before a possible application of Davis-Putnam's propositional split rule. If none of the applications of the diamond elimination rule close the branch, it will continue with the intended application of the split rule.

Unlike the diamond elimination rule and branch modification rule, the application of the elimination rules for the boolean connectives can be performed don't care nondeterministically. However, this does not mean, as far as the efficiency of a modal theorem prover is concerned, all possible continuations are equally preferable. Already Smullyan [16, p. 28] noted that it is more efficient to give priority to applications of the conjunction elimination rule, that is, disjunction elimination should be delayed until no further application of conjunction elimination is possible. Even if we follow Smullyan's guideline it remains to decide in which order we apply the elimination rules to the formulae in a tableau. Three possible solutions can be found in existing systems:

(1) Select the first formula.
(2) Select the formula which contains the least number of occurrences of the disjunction operator (assuming all formulae are in negation normal form).
(3) Select the formula which has the smallest symbol weight.

For example, for $\{(p \vee r) \vee (q \vee r), \Diamond(p \wedge r) \vee \Diamond q, p \vee (q \vee r)\}$ strategy (1) will select the first formula, strategy (2) will select the second formula, and strategy (3) will select the last formula. While the literature on heuristics for selecting split variables in Davis-Putnam algorithms is extensive, very little is known about appropriate strategies for selecting formulae in tableau algorithms.

4 Simplification

This section describes a known simplification technique which helps reducing both the lengths of branches and the number of nonredundant branches.

D'Agostino [3] has shown that an algorithm for testing the satisfiability of propositional formulae based on the tableau rules of Figure 1 for the propositional connectives is not able to simulate the truth table method for propositional logic in polynomial time. He proposes the replacement of the disjunction elimination rule by the rules of Figure 2. The resulting calculus, restricted to the rules for propositional connectives, is called **KE**. The connectives $\wedge$ and $\vee$ are assumed to be commutative. The rule $(\vee S)$ is usually not explicit in presentations of **KE**, but implicit in the definition of *subsumed formulae* on a branch [15]. The calculus obtained by adding (θ) and (K) will be referred to as **MKE**. (Note that **MKE** is not related to the calculus $\Box$**KE** presented by Pitt and Cunningham [15] which is a free variable tableau method for modal logics.)

The truth table method *always* succeeds to prove that ϕ is a tautology using $\mathcal{O}(n \cdot 2^k)$ computation steps, where n is the length of ϕ and k is the number of

$$(PB)\frac{}{C;\phi \mid C;\overline{\phi}} \qquad (\vee E)\frac{C;\phi \vee \psi;\overline{\phi}}{C;\psi;\overline{\phi}} \qquad (\vee S)\frac{C;\phi \vee \psi;\phi}{C;\phi}$$

Fig. 2. Rules for disjunction elimination in **KE**

distinct variables in ϕ. D'Agostino and Mondadori [4, p. 303] show that there is a **KE**-refutation $\mathbf{T}$ of $\overline{\phi}$ with $\mathbf{T}$ having $\mathcal{O}(n \cdot 2^k)$ many nodes. However, their argument requires (i) applications of the (PB) rule which are not strongly analytic, and requires that (ii) after an application of the $(\wedge)$ rule to a conjunction $\varphi \wedge \psi$ one of φ or ψ has to be chosen *don't know* nondeterministically for further analysis, while the other subformula is prohibited from further analysis. For systems which are based on the **KE** calculus but do not adhere to restriction (ii) there is no guarantee that we find a **KE**-refutation of $\overline{\phi}$ using only $\mathcal{O}(n \cdot 2^k)$ computation steps. Consider the formula ϕ_1

$$\neg p \vee \neg q \vee (\neg\phi_1^1 \wedge \neg\phi_1^2) \vee \ldots \vee (\neg\phi_m^1 \wedge \neg\phi_m^2) \vee ((p \vee \neg\phi_{m+1}^1) \wedge (q \vee \neg\phi_{m+1}^2)),$$

where the ϕ_j^i, $1 \leq i \leq 2$, $1 \leq j \leq m{+}1$, are pairwise distinct boolean combinations of p and q, not identical to either p or q. A truth table method will consider the four possible truth assignments to p and q and will show that ϕ_1 is a tautology using $\mathcal{O}(|\phi_1|)$ computation steps. Note that the truth value of ϕ_1 under a truth assignment to p and q is actually independent of the truth value of the ϕ_j^i.

The tableau method based on **KE** will try to find a closed tableau for $\overline{\phi_1}$, that is

$$p \wedge q \wedge (\phi_1^1 \vee \phi_1^2) \wedge \ldots \wedge (\phi_m^1 \vee \phi_m^2) \wedge ((\neg p \wedge \phi_{m+1}^1) \vee (\neg q \wedge \phi_{m+1}^2)).$$

Following D'Agostino and Mondadori [4] we start by applying (PB) repeatedly with respect to the propositional variables p and q, to get the tableau of Figure 3. If we continue by applying conjunction elimination to $\overline{\phi_1}$ we are able to close all branches but the leftmost branch. On the leftmost branch we can not apply the $(\bot)$ rule, but have to proceed by further applications of the (PB) rule. Figure 4 shows that we can close the leftmost branch with only one application of the (PB) rule to the formula $\neg p \wedge \phi_{m+1}^1$ followed by applications of the $(\wedge)$, $(\vee E)$ and $(\bot)$ rules. The size of the tableau is $\mathcal{O}(m)$ which is within the upper bound given by D'Agostino and Mondadori. However, if we proceed by applications of the (PB) rule to the formulae $\phi_1^1, \ldots, \phi_m^1$, and finally to $\neg p \wedge \phi_{m+1}^1$, then we obtain a tableau of size $\mathcal{O}(m \cdot 2^m)$ which exceeds the upper bound. Figure 5 shows a part of the tableau we obtain in this case.

The formula selection strategies of the previous section do not ensure the tableau of Figure 4 is constructed instead of the one of Figure 5. Our proposed solution to this problem is the introduction of *simplification techniques* into tableau calculi.

$$\begin{tikzcd}
 & & & \{\overline{\phi_1}\} \arrow[dll, "(PB)"'] \arrow[drr, "(PB)"] & & & \\
 & \{\overline{\phi_1}, p\} \arrow[dl, "(PB)"'] \arrow[dr, "(PB)"] & & & & \{\overline{\phi_1}, \neg p\} \arrow[dl, "(PB)"'] \arrow[dr, "(PB)"] & \\
\{\overline{\phi_1}, p, q\} & & \{\overline{\phi_1}, p, \neg q\} & & \{\overline{\phi_1}, \neg p, q\} & & \{\overline{\phi_1}, \neg p, \neg q\}
\end{tikzcd}$$

Fig. 3. First part of a tableau $\mathbf{T}_{10}$ for ϕ_1

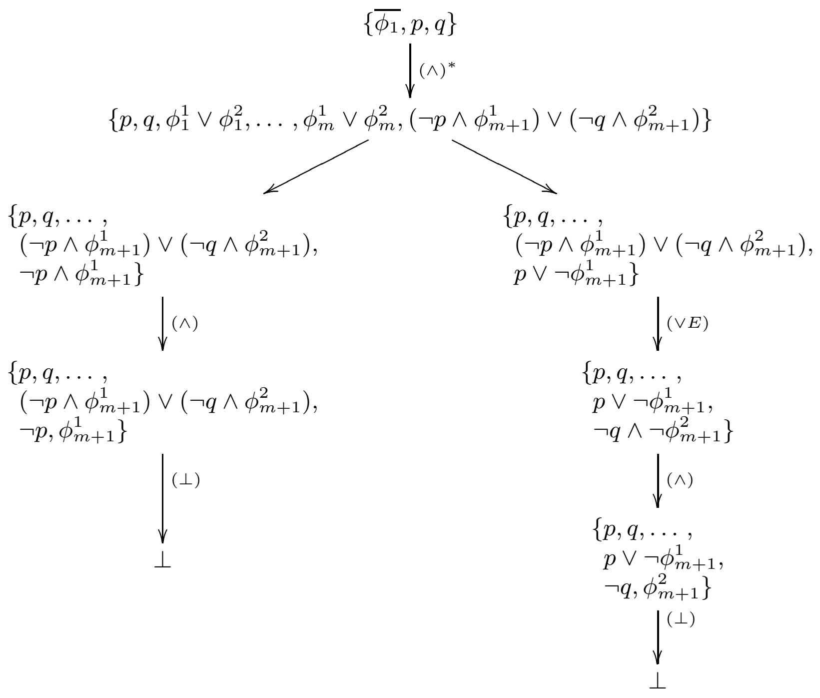


Fig. 4. Application of (PB) with respect to $p \vee \neg\phi_{m+1}^1$ in the leftmost branch of $\mathbf{T}_{10}$

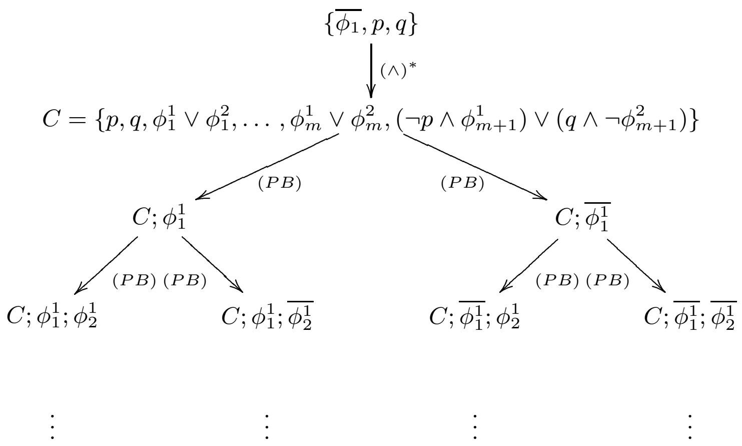


Fig. 5. Applications of (PB) with respect to $\phi_1^1, \dots, p \vee \neg\phi_{m+1}^1$ in the leftmost branch of $\mathbf{T}_{10}$

$\neg\phi \vee \phi \to \top$	$\phi \vee \top \to \top$	$\phi \vee \bot \to \phi$	$\phi \vee \phi \to \phi$
$\neg\phi \wedge \phi \to \bot$	$\phi \wedge \top \to \phi$	$\phi \wedge \bot \to \bot$	$\phi \wedge \phi \to \phi$
$\Box\top \to \top$	$\Diamond\bot \to \bot$	$\neg\top \to \bot$	$\neg\bot \to \top$

Table 1. Rewrite rules for modal formulae

Let $\phi[\psi/\omega]$ be the formula obtained from ϕ by replacing all occurrences of ψ which do not occur in the scope of a modality by the formula ω (either $\top$ or $\bot$). More precisely, $\phi[\psi/\omega]$ is defined by

$$\phi[\psi/\omega] = \begin{cases} \omega, & \text{if } \phi =_{AC} \psi \\ \overline{\omega} & \text{if } \phi =_{AC} \overline{\psi} \\ \neg\phi_1[\psi/\omega], & \text{else if } \phi = \neg\phi_1 \\ \phi_1[\psi/\omega] \wedge \phi_2[\psi/\omega], & \text{else if } \phi = \phi_1 \wedge \phi_2 \\ \phi_1[\psi/\omega] \vee \phi_2[\psi/\omega], & \text{else if } \phi = \phi_1 \vee \phi_2 \\ \phi, & \text{else.} \end{cases}$$

Here $=_{AC}$ denotes equality modulo associativity and commutativity of the connectives $\wedge$ and $\vee$, while $=$ denotes syntactic equality. Let $\phi{\downarrow}$ be the normal form of the formula ϕ obtained with the rewrite rules of Table 1. By $C[\psi/\omega]$ we denote the set $\{\phi[\psi/\omega] \mid \phi \in C\}$ and by $C{\downarrow}$ we denote the set $\{\phi{\downarrow} \mid \phi \in C\}$.

The definition $\phi[\psi/\omega]$ differs from the one given by Massacci [14] in the use of equality modulo associativity and commutativity. Massacci achieves this by replacing the binary connectives $\wedge$ and $\vee$ with set-oriented versions which are commutative, associative and idempotent. Furthermore, Massacci makes the implicit assumption that the replacement operation $\phi[\psi/\omega]$ is followed by a form of elimination of the $\top$ and $\bot$ symbols introduced, but does not specify how.

Let **SKE** and **MSKE** be the calculi of **KE** and **MKE** endowed with the following rule.

$$(S)\frac{C;\phi}{C[\phi/\top]{\downarrow};\phi}$$

The rule can be applied don't care nondeterministically. One possible strategy is to apply (S) after an application of (PB) as follows.

$$\begin{tikzcd} & C \arrow[dl, "(PB)"'] \arrow[dr, "(PB)"] & \\ C;\phi \arrow[d, "(S)"] & & C;\overline{\phi} \arrow[d, "(S)"] \\ C[\phi/\top]{\downarrow};\phi & & C[\overline{\phi}/\top]{\downarrow};\overline{\phi} \end{tikzcd}$$

In this case, the $(\vee E)$ and $(\vee S)$ rules are obsolete.

Figure 6 shows an **SKE**-tableau for the formula ϕ_1. Since each of the final sets contains $\bot$, the tableau is closed. A procedure following the strategy exemplified in Figure 6, and restricting (PB) to propositional variables, corresponds exactly

$$
\begin{tikzcd}
 & & \{\overline{\phi_1}\} \arrow[dl, "(PB)"'] \arrow[dr, "(PB)"] & \\
 & \{\overline{\phi_1}, p\} \arrow[d, "(S)"'] & & \{\overline{\phi_1}, \neg p\} \arrow[d, "(S)"'] \\
 & \{\overline{\phi_1}[p/\top]\downarrow, p\} \arrow[dl, "(PB)"] \arrow[dr, "(PB)"'] & & \{\overline{\phi_1}[\neg p/\top]\downarrow, \neg p\} \\
\{\overline{\phi_1}[p/\top], p, q\} \arrow[d, "(S)"'] & & \{\overline{\phi_1}[p/\top], p, \neg q\} \arrow[d, "(S)"'] & \\
\{\overline{\phi_1}[p/\top][q/\top]\downarrow, p, q\} & & \{\overline{\phi_1}[p/\top][\neg q/\top]\downarrow, p, \neg q\} &
\end{tikzcd}
$$

Fig. 6. A closed tableau for ϕ_1 using **SKE**

to a Davis-Putnam procedure for formulae which are not in clausal form. If we delay the application of the rule (S) until we have applied (PB) to all propositional variables occurring in the formula ϕ under consideration, we obtain a procedure which corresponds exactly to the truth table method.

We do not claim that simplification is a solution to the problem of choosing the next formula to be expanded, but adopting the case distinction mechanism of the truth table method together with simplification, a procedure can be devised which is no worse than the truth table method. Note that the rule (S) can also be added to standard tableau calculi. Simplification by (S) is a means of closing a branch as soon as possible. As a side effect the number of branches is also reduced, since superfluous applications of the (PB) rule can be avoided. In addition, branches are shorter and the counterexample obtainable from an open branch is simpler.

5 Backjumping

This section addresses how a tableau procedure can deal with the don't know nondeterminism inherent in the alternatives of the disjunction elimination rule by forms of backtracking.

Many procedures utilize chronological backtracking, that is, they go back to the most recent state before an application of disjunction elimination. The next example illustrates the drawbacks of this form of backtracking. Let ϕ_2 be a modal formula of the form

$$
\begin{aligned}
\neg\Box s \wedge \Box(p \vee r) \wedge (\Box\neg r \vee \Box q) &\wedge (\neg\Box p \vee \Box r) \\
&\wedge (\phi_1^1 \vee \phi_1^2 \vee \phi_1^3) \\
&\dots \\
&\wedge (\phi_n^1 \vee \phi_n^2 \vee \phi_n^3)
\end{aligned}
$$

where the ϕ_j^i, with $1{\leq}i{\leq}3$, $1{\leq}j{\leq}n$, are modal literals different from the modal literals in the first four conjuncts of ϕ_2. Assume that ϕ_2 is satisfiable. Then:

1. $\Box\neg r$ is false in any model of ϕ_2, since $\Box\neg r$ and $\neg\Box s \wedge (\neg\Box p \vee \Box r)$ imply $\neg\Box p$ and $\Box(p \vee r) \wedge \Box\neg r \wedge \neg\Box p$ is not satisfiable.
2. A simplification step replacing $\Box\neg r$ by $\top$ in ϕ_2 does not affect the literal $\Box r$.

A procedure based on **MSKE** will start by applying conjunction elimination to ϕ_2. It is reasonable to continue with an application of the (S) rule to the units $\neg\Box s$ and $\Box(p \vee r)$. This will not close the tableau. Suppose we proceed by a sequence of applications of (PB) and (S). Let us assume that one of the first formulae to which we apply (PB) is $\Box\neg r$, followed by n modal formulae $\psi_1, \ldots, \psi_n$ chosen from $\phi_1^1, \ldots, \phi_n^3$, and finally $\neg\Box p$. Let us further assume that we construct the tableau in a depth-first way, considering the branch where $\Box\neg r$ is true first. As $\Box\neg r$ is false in any model, traversing the tree below this node, which has 2^n branches, is wasted. After closing the first and second branch we know that whenever $\neg\Box s$, $\Box(p \vee r)$, and $\Box\neg r$ are true on a branch we will be able to close it. However, this insight is not used by the procedure in order to skip the consideration of the branches generated by the n applications of the rule (PB) to the formulae $\psi_1, \ldots, \psi_n$.

This phenomenon is called *thrashing* [13]. Thrashing is the exploration of subtrees of the search tree which differ only in inessential features. Due to the rather complex constraints imposed by the modal formulae on a branch and due to strategies which delay the evaluation of these constraints, modal theorem provers are extremely vulnerable to thrashing.

Techniques that can be used to improve the backtracking behaviour of an algorithm are *backjumping* [5] and *dependency-directed backtracking* [17]. Backjumping backtracks to the last branching point of the search tree which is *relevant* to the failure on the current branch. For example, in the tableau

$$
\begin{tikzcd}
\{\Box p \vee q, \Diamond\neg p \vee q, r \vee s\} \arrow[d, "(\vee)"] \arrow[dr, "(\vee)"] & \\
\{\Box p, \Diamond\neg p \vee q, r \vee s\} \arrow[d, "(\vee)"] \arrow[dr, "(\vee)"] & \mathbf{B}_3 \\
\{\Box p, \Diamond\neg p, r \vee s\} \arrow[d, "(\vee)"] \arrow[dr, "(\vee)"] & \mathbf{B}_2 \\
\{\Box p, \Diamond\neg p, r\} \arrow[d, "(K)"] & \mathbf{B}_1 \\
\{p, \neg p\} &
\end{tikzcd}
$$

after closing the left-most branch backjumping enables us to backtrack past the last branching point, thus skipping $\mathbf{B}_1$, since the application of the $(\vee)$ rule to $r \vee s$ has not introduced a formula to the leftmost branch that has contributed to the derivation of p and $\neg p$.

Dependency-directed backtracking requires, in addition, the maintenance of assumption sets which contain all formulae which have contributed to the closure of a branch. Assumption sets are used to avoid the investigation of any branch which contains the formulae in one of the assumption sets. The assumption sets need to be transfered from one branch of the tableau to another. Furthermore, their number can grow exponentially. For this reason, the additional benefit of using dependency-directed backtracking instead of backjumping is often out of relation to the additional overhead.

Existing systems with backjumping are FaCT [10] and the Logics Workbench [9].

6 Empirical evaluation

This section describes an empirical analysis of implementations of the calculus **MKE**, and its extension **MSKE** with simplification as well as **MSKE** with backjumping. The procedures were implemented using SICStus Prolog Version 3.5. Common features of the procedures are:

1. Input formulae are transformed into negation normal form, and then into a normal form with respect to the rewrite rules of Table 1.
2. The connectives $\wedge$ and $\vee$ are considered to be n-ary operators.
3. The rules $(\wedge)$, $(\neg)$, and $(\bot)$ are preferred over (PB) and $(\vee E)$. These in turn are preferred over (θ) and (K).
4. Our realisation of the (PB) rule is strongly analytic. It is applied only to the smallest disjunct of the selected formula, where the ordering on formulae is defined by a weighted symbol count with $\Box$ and $\Diamond$ assigned 7, $\wedge$ and $\vee$ assigned 0, and $\neg$ and every propositional variable assigned 1. The intention is that propositional literal have smaller weight than modal literals of depth one which in turn have smaller weight than modal literals of depth two (on the assumption that clauses have maximal length 3).
5. The elimination rules are applied to formulae with smallest weight (the weights are those as specified in 4), that is, selection strategy (3) is used.

The procedure **MSKE** applies the (S) rule to any formula introduced by the (PB) rule.

The evaluation was done on a large set of formulae randomly generated as proposed by Giunchiglia and Sebastiani [7] and adopted by Hustadt and Schmidt [11]. The generated formulae are determined by five parameters: the number of propositional variables N, the number of modalities M, the number of modal subformulae per disjunction K, the number of modal subformulae per conjunction L, the modal degree D, and the probability P. Based on a given choice of parameters random modal KCNF formulae are defined inductively according to:

1. A *random (modal) atom* of degree 0 is a variable randomly chosen from the set of N propositional variables. A *random modal atom* of degree D, $D{>}0$, is with probability P a random modal atom of degree 0 or an expression of the form $\Box_i\phi$, otherwise, where $\Box_i$ is a modality randomly chosen from the set of M modalities and ϕ is a random modal KCNF clause of modal degree $D{-}1$.

2. A *random modal literal* (of degree D) is with probability 0.5 a random modal atom (of degree D) or its negation, otherwise.
3. A *random modal KCNF clause* (of degree D) is a disjunction of K random modal literals (of degree D).
4. Now, a *random modal KCNF formula* (of degree D) is a conjunction of L random modal KCNF clauses (of degree D).

It is important to note that in contrast to generating random 3SAT formulae, typically used for the evaluation of propositional decision procedures, generating a random modal 3CNF clause of degree 0 means randomly generating a *multiset* of three propositional variables and negating each member of the multiset with probability 0.5. This means the scheme we have just described allows for tautologous subformulae, like $p \vee \neg p \vee q$, and contradictory subformulae, like $\neg\Box(p \vee \neg p \vee p)$. Furthermore, most of the unsatisfiable random modal 3CNF formulae are trivially unsatisfiable. By definition, a random modal 3CNF formula ϕ is *trivially unsatisfiable* if the conjunction of the purely propositional clauses of ϕ is unsatisfiable. It turns out, these formulae are well suited to show the advantages and disadvantages of backjumping.

We use the parameter settings **PS0** (N=5, M=1, K=3, D=2, P=0.5), **PS1** (N=10, M=1, K=3, D=2, P=0.5), and **PS2** (N=4, M=1, K=3, D=1, P=0.0). **PS2** was generated in accordance with the guidelines of [11], which means, for all occurrences of $\Box\phi$ in a random modal 3CNF formula of degree 1, ϕ has to be a nontautologous clause containing exactly three differing literals.

The tests were conducted by proceeding as follows. We take one of the parameter settings which fixes all parameters except L, the number of clauses. The parameter L ranges from N to $40N$ for **PS0** and **PS1** and from N to $30N$ for **PS2**. For each value of the ratio L/N a set of 100 random modal KCNF formulae of degree D is generated. For small L the generated formulae are more likely to be satisfiable and for larger L the generated formulae are more likely to be unsatisfiable. For **PS2**, already for L/N=30 all formulae are unsatisfiable, so there is no need to increase the ratio beyond 30. For each generated formula ϕ we measure the time needed by one of the tableau procedures to determine the satisfiability of ϕ. There is a upper limit for the CPU time consumed. As soon as this limit is reached, the computation for ϕ is stopped. Our tests were run on a Sun Ultra 1/170E with 196MB main memory using a time-limit of 1000 CPU seconds.

Figures 7–10 depict the outcome of our experiments in the form of percentile graphs. Formally, the *Q%-percentile* of a set of values is the value V such that Q% of the values is smaller or equal to V and $(100-Q)$% of the values is greater than V. The median of a set coincides with its 50%-percentile.

Figure 7 illustrates the importance of design decision 2. Without accounting for associativity of $\vee$ and $\wedge$, the computational behaviour of **MKE** is drastically worse. This can be attributed to the phenomenon that, whenever the (PB) rule introduces a formula ϕ on a branch, modal clauses like $\psi_1 \vee (\phi \vee \psi_2)$ on the branch become true and need not be considered. However, $(\vee S)$ is not applicable when not regarding $\vee$ as being associative. Instead, **MKE** will apply the (PB)

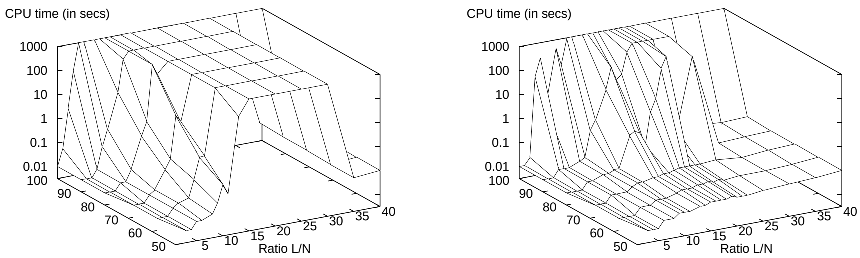
rule to ψ_1. This leads to increased thrashing. Interestingly, the associativity assumption is not important for **MSKE**.

When assuming $\vee$ is an n-ary operator inside **MKE**, then the rules $(\vee E)$ and $(\vee S)$ have essentially the same effect as simplification in **MSKE** on KCNF formulae. This is illustrated by the graphs of Figures 8 and 9, for **MKE** and **MSKE** on **PS1**. Only for the parameter setting **PS2** we see the advantage of using simplification.

The effect of enhancing **MSKE** with backjumping is apparent in Figure 10. Even though a slow down is evident for the 50%–70%-percentiles on **PS1**, the 80%–90%-percentiles show a significant improvement. The slow down is due to the additional computational overhead of managing the information required for performing backjumping. Nevertheless, there are almost no samples in our benchmark suite, for which **MSKE** with backjumping fails to terminate within the given time limit.

It should be stressed that the advantages of the optimisation techniques become most apparent on hard test formulae which are difficult to come by. The usefulness of the optimisation techniques discussed in this paper on other problem sets, like the benchmark collection of the Logics Workbench, requires further investigation.

Our discussion was limited to optimisations of modal **KE** procedures. Similar results can obtained for standard tableau-based procedures with the same enhancements. It is open, however, how enhanced **KE** and standard tableau procedures compare.



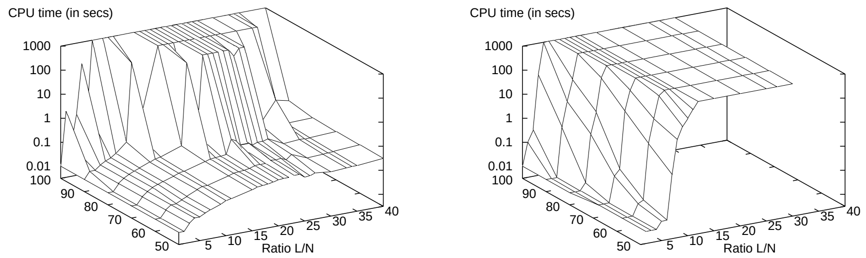
(a) Without associativity

(b) With associativity

Fig. 7. Graphs for **MKE** on **PS0**

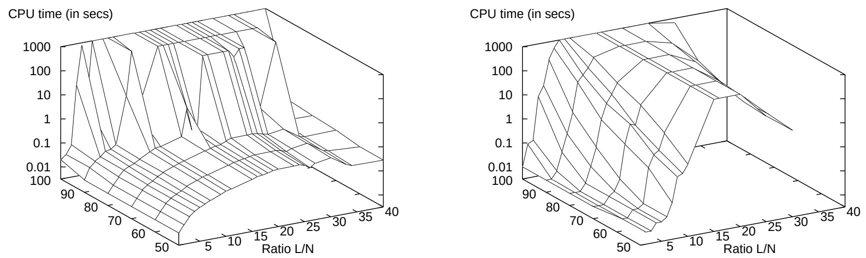
Acknowledgements

We thank the anonymous referees for their comments. This research was conducted while the authors were employed at the Max-Planck-Institut für Infor-



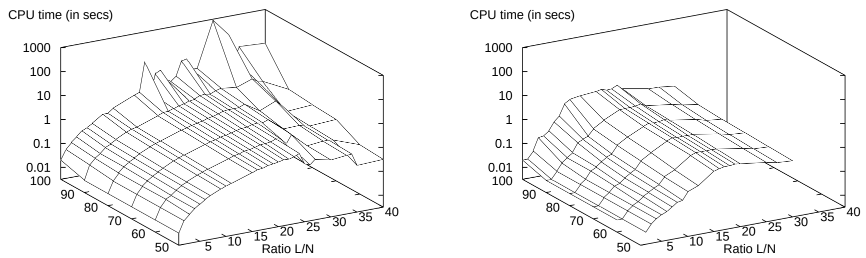
(a) **PS1** (b) **PS2**

Fig. 8. Graphs for **MKE**



(a) **PS1** (b) **PS2**

Fig. 9. Graphs for **MSKE**



(a) **PS1** (b) **PS2**

Fig. 10. Graphs for **MSKE** with backjumping

matik in Saarbrücken, Germany. The work was funded by the MPG and the DFG through grant Ga 261/8-1 and the TRALOS-Project.

References

1. F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithms. In H. Boley and M. M. Richter, editors, *Proc. of the Intern. Workshop on Processing Declarative Knowledge (PDK '91)*, LNAI 567, pages 67–86. Springer, 1991.
2. L. Catach. Tableaux: A general theorem prover for modal logics. *Journal of Automated Reasoning*, 7(4):489–510, 1991.
3. M. D'Agostino. Are tableaux an improvement on truth-tables? *Journal of Logic, Language, and Information*, 1:235–252, 1992.
4. M. D'Agostino and M. Mondadori. The taming of the cut. Classical refutations with analytic cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.
5. R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artifical Intelligence*, 41(3):273–312, 1989/90.
6. M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 169 of *Synthese Library, Studies in Epistemology, Logic, Methodology, and Philosophy of Science*. D. Reidel Publishing Company, 1983.
7. F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures: Case study of modal K. In M. A. McRobbie and J. K. Slaney, editors, *Proc. of the 13th Intern. Conf. on Automated Deduction (CADE-13)*, LNAI 1104, pages 583–597. Springer, 1996.
8. R. Goré. Tableau methods for modal and temporal logics. Technical Report TR-ARP-15-95, Automated Reasoning Project, Australian National University, Canberra, Australia, November 1995.
9. A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfried. The Logics Workbench LWB: A snapshot. *Euromath Bulletin*, 2(1):177–186, 1996.
10. I. Horrocks. Optimisation techniques for expressive description logics. Technical Report UMCS-97-2-1, Department of Computer Science, University of Manchester, Manchester, UK, February 1997.
11. U. Hustadt and R. A. Schmidt. On evaluating decision procedures for modal logic. Research report MPI-I-97-2-003, Max-Planck-Institut für Informatik, Saarbrücken, Germany, February 1997.
12. U. Hustadt and R. A. Schmidt. On evaluating decision procedures for modal logic. In M. E. Pollack, editor, *Proc. of the Fifteenth Intern. Joint Conf. on Artificial Intelligence (IJCAI'97)*, pages 202–207. Morgan Kaufmann, 1997.
13. A. K. Mackworth. Constraint satisfaction. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 205–211. John Wiley & Sons, 1987.
14. F. Massacci. Simplification with renaming: A general proof technique for tableau and sequent-based provers. Technical Report 424, Computer Laboratory, University of Cambridge, Cambridge, UK, May 1997.
15. J. Pitt and J. Cunningham. Theorem proving and model building with the calculus ke. *Journal of the IGPL*, 4(1):129–150, February 1996.
16. R. M. Smullyan. *First-order logic.* Springer, New York, 1968.
17. R. M. Stallman and G. J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artifical Intelligence*, 9:135–196, 1977.