Optimizing a BDD-based Modal Solver*

Guoqiang Pan, Moshe Y. Vardi

Department of Computer Science, Rice University, Houston, TX
gqpan,vardi@cs.rice.edu

Abstract. In an earlier work we showed how a competitive satisfiability solver for the modal logic $\mathcal{K}$ can be built on top of a BDD package. In this work we study optimization issues for such solvers. We focus on two types of optimizations. First we study variable ordering, which is known to be of critical importance to BDD-based algorithms. Second, we study modal extensions of the pure-literal rule. Our results show that the payoff of the variable-ordering optimization is rather modest, while the payoff of the pure-literal optimization is quite significant. We benchmark our optimized solver against both native solvers (DLP) and translation-based solvers (MSPASS and SEMPROP). Our results indicate that the BDD-based approach dominates for modally heavy formulas, while search-based approaches dominate for propositionally-heavy formulas.

1 Introduction

In the last 20 years, modal logic has been applied to numerous areas of computer science, including artificial intelligence, program verification, hardware verification, database theory, and distributed computing. In this paper, we restrict our attention to the smallest normal modal logic $\mathcal{K}$ [3]. Since modal logic extends propositional logic, the study in modal satisfiability is deeply connected with that of propositional satisfiability. In the past, a variety of approaches to propositional satisfiability have been combined with various approaches to handle modal connectives and implemented successfully. For example, a tableau-based decision procedure for $\mathcal{K}$ is presented in [16, 12]. It is built on top of the propositional tableau construction procedure by forming a fully expanded propositional tableau and generating successor nodes "on demand". A similar method uses the Davis-Putnam-Logemann-Loveland (DPLL) method as the propositional engine by treating all modal subformulas as propositions and, when a satisfying assignment is found, checking modal subformulas for the legality of this assignment [10]. Another approach to modal satisfiability, the inverse calculus for $\mathcal{K}$ [37] can be seen as a modalized version of propositional resolution. Non-propositional based methods take a different approach to the problem. It is well known that formulas in $\mathcal{K}$ can be translated to first order formulas via standard translation [35, 19]. Recently, it has been shown that, by encoding the modal depth information into the encoding, a first-order theorem prover can be used efficiently for deciding modal satisfiability [1]. The latter approach works nicely with resolution-based first-order theorem prover, which can be

* Authors supported in part by NSF grants CCR-9988322, CCR-0124077, IIS-9908435, IIS-9978135, and EIA-0086264, by BSF grant 9800096, and by a grant from the Intel Corporation.

used as a decision procedure for modal satisfiability by using appropriate resolution strategies [14].

In [21] we described a new approach to decide satisfiability of $\mathcal{K}$ formulas, inspired by the automata-theoretic approach for logics with the tree-model property [36]. In that approach one proceeds in two steps. First, an input formula is translated to a tree automaton that accepts all tree models of the formula. Second, the automaton is tested for non-emptiness, i.e., does it accept some tree. The algorithms described in [21] combine the two steps and apply the non-emptiness test without explicitly constructing the automaton. (As was pointed out in [28, 2], the inverse method described in [37] can also be viewed as an implementation of the automata-theoretic approach that avoids an explicit automata construction.)

The logic $\mathcal{K}$ is simple enough for the automaton non-emptiness test to consist of a single fixpoint computation. This computation starts with a set of states and then repeatedly applies a monotone operator until a fixpoint is reached. In the automata that correspond to formulas, each state is a *type*, i.e., a set of formulas satisfying some consistency conditions. The algorithms in [21] all start from some set of types, and then repeatedly apply a monotone operator until a fixpoint is reached: either they start with the set of *all* types and remove those types with "possibilities" $\diamond\varphi$ for which no "witness" can be found, or they start with the set of types having no possibilities $\diamond\varphi$, and add those types whose possibilities are witnessed by a type in the set. The two approaches, top-down and bottom-up, corresponds to the two ways in which non-emptiness can be tested for automata for $\mathcal{K}$: via a greatest fixpoint computation for automata on infinite trees or via a least fixpoint computation for automata on finite trees. The bottom-up approach is closely related to the inverse method described in [37], while the top-down approach is reminiscent of the "type-elimination" method developed for Propositional Dynamic Logic in [24].

The key idea underlying the implementation in [21] is that of representing sets of types and operating on them symbolically, using Binary Decision Diagrams (BDDs) [4]. BDDs provide a compact representation of propositional formulas, and are commonly used as a compact representation of states. One of their advantages is that they enable efficient operations for certain manipulations on BDDs. The work reported in [21] consisted of a viability study for the BDD-based approach, using existing benchmarks of modal formulas, TANCS 98 [13] and TANCS 2000 [18], and comparing performance with with that of the DPLL-based solver *SAT [33] and the tableau-based solver DLP [22]. A straightforward implementation of the BDD-based approach did not yield a competitive algorithm, but an implementation using a careful representation of types and taking advantage of the finite-tree-model property for $\mathcal{K}$ turned out to be competitive.

In this work we study optimization issues for BDD-based $\mathcal{K}$ solvers. We focus on two types of optimizations. First we focus on improving the performance the the BDD operations used by the algorithm. We study the issue of variable order, which is known to be of critical importance to BDD-based algorithms. The performance of BDD-based algorithms depend crucially on the size of the BDDs and variable order is a major factor in determining BDD size, as a "bad" order may cause an exponential blow-up. While finding an optimal variable order is known to be intractable [34], heuristics often work quite well in practice [26]. We focus here on finding a good initial variable order (for large problem instances we have no choice but to invoke dynamic variable ordering, provided by the BDD package), tailored to the application at hand. Our finding is that choosing a good initial variable order does improve performance, but the improvement is rather modest.

We then turn to a preprocessing optimization. The idea is to apply some light-weight reasoning to simplify the input formula before starting to apply heavy BDD operations. In the propositional case, a well-known preprocessing rule is the *pure-literal* rule [7]. Preprocessing has also been shown to be useful for linear-time formulas [30, 8], but has not been explored for $\mathcal{K}$. Our preprocessing is based on a modal pure-literal simplification, which takes advantage of the layered-model property of $\mathcal{K}$. We show that adding preprocessing yields a fairly significant performance improvements, enabling us to handle the hard formulas of TANCS 2000.

To assess the competitiveness of our optimized solver, called $\mathcal{KBDD}$, we benchmark it against both a native solver and against translation-based solver. As mentioned earlier, DLP is a tableau-based solver. MSPASS [14] is a resolution-based solver, applied to a translation of modal formulas to first-order formulas. Finally, we developed also a translation of $\mathcal{K}$ to QBF (which is of independent interest), and applied SEMPROP, which is a highly optimized QBF solver [17]. Our results indicate that the BDD-based approach dominates for modally heavy formulas: $\mathcal{KBDD}$'s performance is superior for TANCS 2000 formulas. In contrast, search-based approaches dominate for propositionally-heavy formulas: for random formulas, generated according to the distribution advocated in [23], DLP's performance is superior to that of $\mathcal{KBDD}$.

This paper is organized as follows. After a review of modal logic $\mathcal{K}$ and our previous BDD-based decision procedure in Section 2, we present optimizations based on BDD variable order and formula simplification in Section 3. In Section 4, we present a new way to benchmark problem suites in $\mathcal{K}$ by translating them into QBF based on a automata-theoretical algorithm, and we outline a class of random formulas that is hard for our solver. We also present a performance comparison between solvers which used different approaches and discussed the strength and shortcomings of our implicit-state solver vs. explicit-state solvers.

2 Background

In this section, we introduce the syntax and semantics of the modal logic $\mathcal{K}$, and describe the BDD-based algorithm of [21].

The Modal Logic $\mathcal{K}$

The set of $\mathcal{K}$ formulas is constructed from a set of propositional variables $\Phi = \{q_1, q_2, \ldots\}$, and is the least set containing Φ and being closed under the Boolean connectives $\wedge$ and $\neg$ and the unary modality $\Box$. As usual, we use other Boolean connectives as abbreviations, and $\Diamond\varphi$ as an abbreviation for $\neg\Box\neg\varphi$. The set of propositional variables used in a formula φ is denoted $AP(\varphi)$.

A formula in $\mathcal{K}$ is interpreted in a *Kripke structure* $K = \langle \Phi, W, R, L \rangle$, where W is a set of possible *worlds*, $R \subseteq W^2$ is the accessibility relation on worlds, and $L : W \to 2^{\Phi}$ a labeling function. The notion of a formula φ being *satisfied* in a world w of the Kripke structure K (written as $K, w \models q$) is that of propositional logic extended with:

– $K, w \models \Box\varphi$ if, for all $w' \in W$, if $(w, w') \in R$, then $K, w' \models \varphi$

A formula ψ is *satisfiable* if there exist K, w such that $K, w \models \psi$. In this case, K is called a *model* of ψ.

Given a formula ψ, let $\mathsf{sub}(\psi)$ be the set of subformulas of ψ. Given $\varphi \in \mathsf{sub}(\psi)$, we define $dist(\psi, \varphi)$ as follows: if $\psi = \varphi$, then $dist(\psi, \varphi) = 0$; if $\varphi = \varphi' \wedge \varphi''$, $\varphi' \vee \varphi''$, or $\neg\varphi'$, then $dist(\psi, \varphi') = dist(\psi, \varphi'') = dist(\psi, \varphi)$; if $\varphi = \Box\varphi'$ or $\Diamond\varphi'$, then $dist(\psi, \varphi') = dist(\psi, \varphi) + 1$. The modal depth $md(\psi)$ is defined as $max_{\varphi \in \mathsf{sub}(\psi)}(dist(\psi, \varphi))$.

A key property of $\mathcal{K}$ is the *tree-model property*, which allows the automata-theoretic approach to be applied [36]. In fact, it has the stronger *finite-tree-model property*. For our algorithms, we rely on a weaker property.

Theorem 1. *$\mathcal{K}$ has the* layered model property. *That is, if a $\mathcal{K}$ formula φ is satisfiable, then there is Kripke structure $K = \langle V, W, R, L\rangle$ such that $K, w_0 \models \varphi$ and for all $w \in W$, the distance between w and w_0 is uniquely defined.*

Symbolic satisfiability solving

Three approaches are described in [21]. The top-down approach corresponds to checking emptiness of automata on infinite tree (with trivial acceptance condition). The bottom-up approach corresponds to checking emptiness of automata on finite-trees. It is shown that best performance is obtained by using a level-based bottom-up approach, which relies on the layered-model property of Theorem 1.

We assume that we are dealing with $\mathcal{K}$ formulas in *negation normal form* (NNF), where all subformulas are of the form $\varphi \wedge \varphi'$, $\varphi \vee \varphi'$, $\Box\varphi$, $\Diamond\varphi$, q, or $\neg q$ where $q \in AP(\psi)$. All $\mathcal{K}$ formulas can be converted (with linear blow-up) into negation normal form by pushing negation inwards.

A set $p \subseteq \mathsf{sub}(\psi)$ is a *full ψ-particle* if it satisfies the following conditions:

– If $\varphi = \neg\varphi'$, then $\varphi \in p$ implies $\varphi' \notin p$.
– If $\varphi = \varphi' \wedge \varphi''$, then $\varphi \in p$ implies $\varphi' \in p$ and $\varphi'' \in p$.
– If $\varphi = \varphi' \vee \varphi''$, then $\varphi \in p$ implies $\varphi' \in p$ or $\varphi'' \in p$.

It is known that a model for ψ can be constructed from a set of ψ-particles [3]. We take advantage here of the layered-model property, and instead of representing a model using a set of particles, we represent each layer of the model using a separate set of particles. Since not all subformulas are relevant in a single layer, the representation can be more compact. For $0 \le i \le md(\psi)$, let $\mathsf{sub}_i(\psi) := \{\varphi \in \mathsf{sub}(\psi) \mid dist(\psi, \varphi) = i\}$ and $\mathsf{part}_i(\psi) \subset 2^{\mathsf{sub}_i(\psi)}$ be the set of ψ-particles that are contained in $\mathsf{sub}_i(\psi)$. We define a layered accessibility relation R_i:

$$R_i(p, p') \text{ iff } p \subseteq \mathsf{sub}_i(\psi),\, p' \subseteq \mathsf{sub}_{i+1}(\psi), \text{ and } \varphi \in p' \text{ for all } \Box\varphi \in p.$$

A sequence $\overline{P} = \langle P_0, P_1, \ldots, P_d\rangle$ of sets of particles with $P_i \subseteq \mathsf{part}_i(\psi)$ can be converted into a layered Kripke structure $K_P = \langle AP(\psi), \biguplus_{i=0}^{d} P_i, \bigcup_{i=0}^{d} R_i, L\rangle$ (here $\biguplus$ denote the disjoint union), where for a particle $p \in P_i$, we defined $L(w) = P_i \cap AP(\psi)$.

A layered Kripke structure for ψ can be described by such a sequence $\overline{P}$ such that if $p \in P_i$ and $\Diamond\varphi \in p$, then $\Diamond\varphi$ is *witnessed* by a particle in P_{i+1}, that is, there exists $p' \in P_{i+1}$, where $\varphi \in p'$ and for all $\Box\varphi' \in p$, we have $\varphi' \in p'$.

Theorem 2. *If $\overline{P}$ is a layered Kripke structure and there is $p \in P_0$ where $\psi \in P$, then $K_P, p \models \psi$.*

Proof. It can be shown for all $p \in K_P$, if $\varphi \in p$, then $K_P, p \models \varphi$ by induction from P_d to P_0. The witnessing requirement ensures the existence of successors. For a more detailed proof, see [20].

The level-based bottom-up algorithm constructs such a maximal $\overline{P}$.

$d = md(\psi)$
$P_d = Initial_d(\psi)$
for $i = d - 1$ downto 0 **do**
  $P_i \Leftarrow Iterate(P_{i+1}, i)$
if exists $p \in P_0$ where $\psi \in p$ **then** ψ is satisfiable.
else ψ is not satisfiable.

The auxiliary functions are defined as follows:

- $Initial_i(\psi) = \mathsf{part}_i(\psi)$.
- $Iterate(P, i) = \{p \in Initial_i(\psi) \mid$ for all $\Diamond\varphi \in p$ there exists $q \in P$ where $\varphi \in q$ and $R_i(p, q)\}$.

Corollary 1. *The level-based bottom-up algorithm is sound and complete.*

Note the soundness is a direct result of Theorem 2. Maximality of the generated $\overline{P}$ ensures completeness. This algorithm works bottom-up in the sense that it starts with the leaves of a layered model *at the deepest level* and then move up the layered model toward the root, adding particles that are "witnessed". A standard bottom-up approach for finite-tree automata emptiness would start with *all* leaves of a tree model.

For a BDD-based implementation, we found that using variables to represent only *lean particles* increases performance significantly. For formulas in $\mathsf{sub}(\psi)$, we represent each formula of type q, $\neg q$, $\Diamond\varphi$, and $\Box\varphi$ with a BDD variable. The other subformulas are not represented explicitly, but are logically implied. The witness check (for every $\Diamond$ check in $Iterate$) can be implemented using a symbolic pre-image operation. We refer to the level-based, bottom-up, lean-particle implementation of the BDD-based algorithm as $\mathcal{KBDD}$.

3 Optimizations

The investigation reported in [21] constituted a viability study, investigating the basic implementation strategies for a BDD-based $\mathcal{K}$-solver and comparing to other solvers such as *SAT and DLP. The paper ended with the conclusion that $\mathcal{KBDD}$, with its level-based, bottom-up, lean-particle implementation, is a viable solver. In this paper our focus is on further optimization of $\mathcal{KBDD}$. We focus on two types of optimizations. First we study low-level optimization, by focusing on BDD variable order, which is known to be of critical importance to BDD-based algorithms. Second, we study high-level optimization by focusing on modal extensions of the pure-literal rule.

Variable order

Performance of BDD-based algorithms is very sensitive to BDD variable order, since it is the primary factor that influences the BDD size. Space blowups of of BDDs for the state sets P_i, as well as intermediate BDDs during pre-image operation, is observed in our experiments to be a major factor in performance degradation. Since every step in the iteration process uses BDDs with variables from different modal depth, dynamic variable ordering is of limited benefit (though necessary when dealing with intermediate BDD blowups) is not always the answer for $\mathcal{KBDD}$, because there may not be sufficient reuse to make it worthwhile. Thus, we focused here on constructing heuristically a good initial variable order. Our heuristic attempts to find a variable order that is appropriate for $\mathcal{KBDD}$. In this we follows the work of Kamhi and Fix, who argued in favor of application-dependent variable orders [15]. As we will show, choosing a good initial variable order does improve performance, but the improvement is rather modest.

A naive method for assigning initial variable order to a set of subformulas would be to traverse the DAG for the formula in some order. We used a depth-first, pre-order traversal. This order, however, does not meet the basic principle of BDD variable ordering, which is to keep related variables in close proximity. Our heuristic is aimed at identifying such variables. Note that in our lean representation variables correspond to modal subformulas or atomic subformulas. We found that related variables correspond to subformulas that are related via the sibling or niece relationships. We say that v_x is the *child* of v_y if for the corresponding subformulas we have that $\varphi_x \in \mathsf{sub}_i(\psi)$, $\varphi_y \in \mathsf{sub}_{i+1}(\psi)$, and $\varphi_y \in \mathsf{sub}_1(\varphi_x)$, for some $0 \leq i < md(\psi)$. We say that v_x and v_y are siblings if either both φ_x and φ_y are in $\mathsf{sub}_i(\psi)$ or they are both children of another variable v_z. We say that v_y is a *niece* of v_x if there is a variable v_z such that v_z is a sibling of v_x and v_y is a child of v_x. We say that v_x and v_y are *dependent* if they are related via the sibling or the niece relationship. The rationale is that we want to optimize state-set representation for pre-image operations. Keeping siblings close helps in keeping state-set representation compact. Keeping nieces close to their "aunts", helps in keeping intermediate BDDs compact.

We build variable order from the top of the formula down. We start with left-to-right traversal order of top variables in the parse tree of ψ as order for variables corresponding to subformulas in $\mathsf{sub}_0(\psi)$. Given an order of the variables of modal depth $< i$, a greedy approach is used to determine the placement of variables at modal depth i. When we insert a new variable v we measure the cumulative distance of v from all variables already in the order that are dependent on v. We find a location for v that minimizes the cumulative distance from other dependent variables.

Formula simplification

We now turn to a high-level optimization, in which we apply some preprocessing to the formula before submitting it to $\mathcal{KBDD}$. The idea is to apply some light-weight reasoning to simplify the input formula before starting to apply heavy-weight BDD operations. In the propositional case, a well-known preprocessing rule is the *pure-literal* rule [7], which can be applied both in a preprocessing step as well as dynamically, following the unit-propagation step. Preprocessing has also been shown to be useful for linear-time formulas [30, 8], but has not been systematically explored for $\mathcal{K}$. Our preprocessing is based on a modal pure-literal simplification, which takes advantage of the layered-model property of $\mathcal{K}$.

When studying preprocessing for satisfiability solvers, two types of transformation should be considered:

1. Equivalence preserving: Unit propagation is an example of an equivalence-preserving transformation. Such transformations are used in model checking [30, 8], where the semantics of the formula needs to be preserved. An equivalence-preserving rule can be applied to subformulas.
2. Satisfiability preserving: Pure-literal simplification is an example of a satisfiability-preserving transformation. Such transformations allows for more aggressive simplification, but cannot be applied to subformulas. Note that such a transformation can be used for satisfiability solving but not for model checking.

Our preprocessing was designed to reduce the number of BDD operations called by $\mathcal{KBDD}$, though its correctness is algorithm independent. The focus of the simplification is on both reducing the size of the formula and reducing the number of model operators. A smaller formula leads to a reduction in BDD size as well as a reduction in the number of BDD operations and dynamic variable re-orderings. Less model operators would give a smaller transition relation, since we have a constraint for each $\Box$ subformula, as well as a smaller number of BDD operations involved in witnessing $\Diamond$ subformulas.

Rewrite rules Our preprocessing includes rewriting according to a collection of rewrite rules (see Table 1). Although the rules can be applied in both directions, we apply only the direction that reduces the size of the formula. It is easy to see that the rules are equivalence or satisfiability preserving. These rules by themselves are only modestly effective for $\mathcal{K}$ formulas; they do become quite effective, however, when implemented in combination with pure-literal simplification, described below. These rules allows us to propagate the effects of pure-literal simplification by removing redundant portions of the formula after pure-literal simplification. This usually allows more pure literals to be found and can greatly reduce the size of the formula.

Propositional rules		
Equivalence	$f \wedge \mathsf{true} \to f$	$f \wedge \mathsf{false} \to \mathsf{false}$
	$f \vee \mathsf{true} \to \mathsf{true}$	$f \vee \mathsf{false} \to f$
	$f \wedge f \to f$	$f \vee f \to f$
	$f \wedge \neg f \to \mathsf{false}$	$f \vee \neg f \to \mathsf{true}$
Modal rules		
Equivalence	$\Diamond\, \mathsf{false} \to \mathsf{false}$	$\Box\, \mathsf{true} \to \mathsf{true}$
	$\Diamond f \vee \Diamond g \to \Diamond(f \vee g)$	$\Box f \wedge \Box g \to \Box(f \wedge g)$
Satisfiability preserving	$\Diamond f \wedge \Box g \wedge h \to \Diamond(f \wedge g) \wedge h$ where h is a propositional formula.	$\Diamond f \to f$

Table 1. Simplification rewriting rules for $\mathcal{K}$

Pure-literal simplification To apply pure-literal simplification to $\mathcal{K}$ satisfiability solving, we first need to extend it to the modal setting.

Definition 1. *Given a set S of (propositional or modal) formulas in NNF, We define* $\mathsf{lit}(S) = \{l \mid l \in \mathsf{sub}(S)$ *and* l *is* q *or* $\neg q\}$ *as the set of literals of* S*. The set* $\mathsf{pure}(S)$

of defined as the set of literals which have a pure-polarity occurrence in S, i.e., $l \in \mathsf{pure}(S)$ *iff* $l \in \mathsf{lit}(S)$ *and* $\neg l \notin \mathsf{lit}(S)$.

It is well known that pure-literal simplification preserves propositional satisfiability; that is, given a propositional formula φ, for any literal $l \in \mathsf{pure}(\varphi)$, φ is satisfiable iff $\varphi[l/\,\mathsf{true}]$ is satisfiable. There are a number of ways to extend the definition of pure literals to modal logics. A naive definition can be as follows:

Definition 2. *For a formula ψ in NNF, we define* $\mathsf{pure}(\psi) = \mathsf{pure}(\mathsf{sub}(\psi))$ *as the set of* globally pure literals *of ψ, and define the corresponding formula after pure literal simplification as* $\psi'_G = \psi[\mathsf{pure}(\psi)/\,\mathsf{true}]$.

Given that $\mathcal{K}$ has the layered-model property, assignments to literals at different modal depth are in different worlds and should not interfere with each other. A stronger definition of pure literals can be as follows:

Definition 3. *For ψ in NNF, we define* level-pure literals *by* $\mathsf{pure}_i(\psi) = \mathsf{pure}(\mathsf{sub}_i(\psi))$, *for* $0 \leq i \leq md(\psi)$. *The substitution used for level-pure literals needs to take into consideration that* $l \in \mathsf{pure}_i(\psi)$ *is only pure at modal depth* i, *so we let* $\psi[\mathsf{pure}_i(\psi)/\,\mathsf{true}]_i$ *be the substitution with* true *of all literals* l *that occur distance* i *from* ψ. *The result of the pure-literal simplification is* $\psi'_L = \psi[\mathsf{pure}_0(\psi)/\,\mathsf{true}]_0 \ldots [\mathsf{pure}_{md(\psi)}(\psi)/\,\mathsf{true}]_{md(\psi)}$.

Remark 1. It is possible to push this idea of "separation" further. Because each world in the model only needs to satisfy a subset of $\mathsf{sub}(\psi)$, the possible subsets can be constructed to determine which of the literals can be pre-assigned true. For example, it is possible to construct sets of subformulas that can occur together in a tableau and define pure literals based on such sets. We did not find that the performance benefit justified the implementation overhead for this extension.

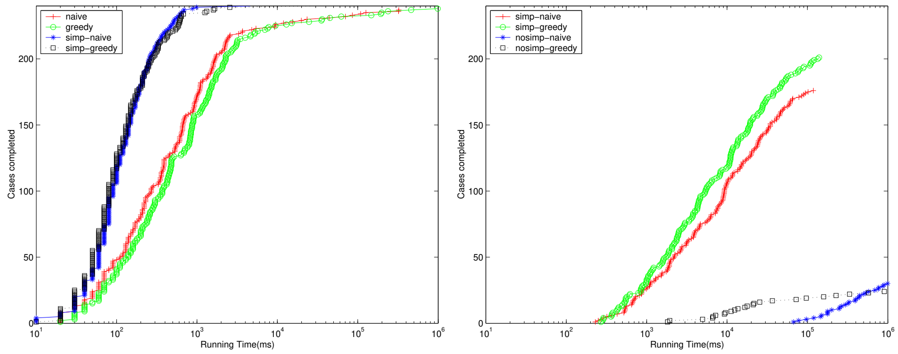
We now prove the sound and completeness of pure-literal simplification. That is, we show that pure-literal simplification preserves satisfiability for both globally pure literals and level-pure literals.

Theorem 3. *Both global and level pure-literal simplifications are satisfiability preserving. That is, for a formula ψ, we have that ψ is satisfiable iff ψ'_G (or ψ'_L) is satisfiable.*

Proof. We write ψ' instead of $\psi'_{G(L)}$ when the formula used is clear from the context. Without loss of generality, we assume that only one literal l is substituted. Since other pure literals for ψ are still pure with respect to ψ' under both definitions, the general case can be shown by induction on the number of literals.

The completeness part of the claim is easy. It is known that the $\Box$ and $\Diamond$ operators are *monotone* [3]. More formally, if ψ is a formula in NNF, α is a subformula occurrence of ψ and β is another formula that is logically implied by α, then $\psi[\alpha/\beta]$ is logically implied by ψ. It follows that ψ' is logically implied by ψ. In particular, if ψ is satisfiable, then ψ' is satisfiable.

We only need to show soundness for level-pure literals since globally pure literals can be seen as a special case. In the following, we take $K = \langle \Phi, W, R, L\rangle$ and $K' = \langle \Phi, W, R, L'\rangle$ to be tree Kripke structures of depth $md(\psi)$ with the same underlying frame, and $w_0 \in W$ to be the root of the tree, where we want ψ and ψ' to hold.



TANCS 2000-easy (cnfSSS) TANCS 2000-medium (cnfLadn)

Fig. 1. Optimizations on TANCS 2000

Assume $K', w_0 \models \psi'$. Let $dist(\psi, l) = d$. For $0 \leq i \leq md(\psi)$, define $W_i = \{w \mid$ distance between w and $w_0 = i\}$. We construct K from K' by defining L as follows: (1) $L(w) = L'(w)$ for $w \notin W_d$, (2) $L(w)(l) = \mathsf{true}$ for $w \in W_d$, and (3) $L(w)$ agree with $L'(w)$ for $p \in \Phi - AP(l)$ and $w \in W_d$. Intuitively, we modify L' by making l true in all worlds $w \in W_d$.

We claim that for a formula $\varphi \in \mathsf{sub}_i(\psi)$, and a world $w \in W_i$ we have that $K', w \models \varphi[l/\mathsf{true}]_{d-i}$ implies $K, w \models \varphi$. It follows that $K, w_0 \models \psi[l/\mathsf{true}]_d$. For $d < i \leq md(\psi)$, note that $\varphi[l/\mathsf{true}]_{d-i} = \varphi$ and L agrees with L' on all worlds in $\cup_{j=i}^{md(\psi)} W_j$. Since, truth of formulas in worlds of W_j depends only on worlds in $\cup_{j=i}^{md(\psi)} W_j$, the claim holds trivially. For $i \leq d$, we use induction on the structure of φ. If φ is a propositional literal, the property holds because either $\varphi = l$ and $dist(\psi, \varphi) = d$, in which case $K, w \models l$ by construction, or either φ is a literal l' such that $AP(l') \neq AP(l)$ or $dist(\psi, \varphi) \neq d$, in which case $L(w)$ and $L'(w)$ agree on l', so $K', w \models l'$ implies $K, w \models l'$. For the induction, we show only the case when $\varphi = \Box\varphi'$. Given $K', w \models \varphi[l/\mathsf{true}]_{d-i}$, we have that $K', w' \models \varphi'[l/\mathsf{true}]_{d-i-1}$ holds for all w' such that $R(w, w')$. Note that if $R(w, w')$ holds and $w \in W_i$, then $w' \in W_{i+1}$. By the inductive hypothesis, $K, w' \models \varphi'$ for all such w' as well. So $K, w \models \varphi$ holds.

Results

To demonstrate the effects of variable ordering and formula simplification, we tested $\mathcal{KBDD}$ with both naive and greedy variable ordering with and without formula simplification, using TANCS 2000 easy and medium formulas. The results are in Figure 1.

To avoid getting into overwhelming details in the comparison of solvers and to present a global view of performance, we used the presentation technique suggested in [32], where we plot the number of cases solved against the running time used.

We see in Figure 1 that formula simplification yields a significant performance improvement. This improvements was observed for different types of formulas and different variable-ordering algorithms. In particular, $\mathcal{KBDD}$ was able to avoid space outs in many cases. We can also see greedy variable ordering is useful in conjunction with formulas simplification, improving the number of completed cases and sometimes running time as well. Without formula simplification, the results for greedy variable ordering are not consistent, as overhead of finding the variable order may offset any advantages of applying it. The combination of formula simplification and greedy variable ordering clearly improves the performance of $\mathcal{KBDD}$ in a significant way. In the next section, we compare the performance of optimized $\mathcal{KBDD}$ against three other solvers.

4 Benchmarking

To assess the effectiveness of BDD-based decision procedures for $\mathcal{K}$, we compared the optimized $\mathcal{KBDD}$ against three solvers: (1) DLP is a tableau-based solver [22], (2) MSPASS is a resolution-based solver, applied to a translation of modal formulas to first-order formulas [14][1] (3) We developed also a translation of $\mathcal{K}$ to QBF (which is of independent interest), and applied SEMPROP, which is a highly optimized QBF solver [17]. For a fair comparison, we checked first whether our formula-simplification optimization is useful for these solvers, and used it when it was (DLP and SEMPROP).

We ran the comparison on benchmark formulas from TANCS 98 and TANCS 2000. The latter suite is divided into easy, medium, and hard portion. In addition we used randomly generated formulas, as suggested in [23]. This scheme generates random CNF formulas parameterized with the number of propositions N, the number of literals in each clause C, the fraction of modal literals in each clause p, a depth bound d, and the number of top level clauses L. L clauses are generated with C literals each, where $p \times C$ literals are modal and the rest are propositional. The maximum modal depth of the formula is bounded by d. With fixed density, depth, modal fraction, and number of atomic propositions, the complexity of the resulting formula can be varied by adjusting the density L/N. We used $d = 1, 2$, $C = 3$ and $p = 0.5$ in our experiments.

A reduction of $\mathcal{K}$ to QBF

Both $\mathcal{K}$ and QBF have PSPACE-complete decision problems [16, 31]. This implies that the two problems are polynomially reducible to each other. A natural reduction from QBF to $\mathcal{K}$ is described in [12]. In the last few years extensive effort was carried out into the development of highly-optimized QBF solvers [17, 5]. One motivation for this effort is the hope of using QBF solvers as generic search engines [25], much in the same way that SAT solvers are being used as generic search engines. This suggests that another approach to $\mathcal{K}$ satisfiability is to find a natural reduction of $\mathcal{K}$ to QBF, and then apply a highly optimized QBF solver. We describe now such a reduction. (A similar approach is suggested in [5] without providing either details or results.)

QBF is an extension of propositional logic with quantifiers. The set of QBF formulas is constructed from a set $\Phi = \{x_1, \ldots x_n\}$ of Boolean variables, and closed under the Boolean connectives $\wedge$ and $\neg$, as well as the quantifier $\forall x_i$. As usual, we use other Boolean operators as abbreviations, and $\exists x_i : \varphi$ as shorthand for $\neg\forall x_i : \neg\varphi$. Like propositional formulas, QBF formulas are interpreted over truth assignments. The semantics of quantifiers is defined by: $\tau \models \forall p : \varphi$ iff $\tau[p/1] \models \varphi$ and $\tau[p/0] \models \varphi$.

[1] We used MSPASS 1.0.0t1.3 with options -EMLTranslations=1 -EMLFuncNary=1 -Select=2 -PProblem=0 -PGiven=0 -Sorts=0 -CNFOptSkolem=0 -CNFStrSkolem=0 -CNFRenOps=1 -Split=-1 -Ordering=0 -CNFRenMatch=0 -TimeLimit=1000. Compiled with gcc-3.1.1. Allowing different parameter on different cases for MSPASS would give much better results, but good parameter for one subset of formulas causes another subset to timeout or crash.

Based on the layered model $\overline{P} = \langle P_0, \ldots, P_d \rangle$ generated by $\mathcal{KBDD}$, we construct QBF formulas $f_0, f_1, \ldots f_d$ so each f_i encodes the Kripke structure defined by $\langle P_i, P_{i+1}, \ldots P_d \rangle$. The construction is by backward induction for $i = d \ldots 0$.

For every $\varphi \in \mathsf{sub}_i(\psi)$, we have a corresponding variable $x_{\varphi,i}$ as a free variable in f_i. The intuition is that f_i describes the set P_i. That is, for each $p \subseteq \mathsf{sub}_i(\psi)$, define the truth assignment τ_p^i as follows: $\tau_p^i(x_{\varphi,i}) = 1$ iff $\varphi \in p$. The intention is to have $P_i = \{p \subset \mathsf{sub}_i(\psi) | \tau_p^i \models f_i\}$. We then say that f_i *characterizes* P_i.

We start by constructing a propositional formula lc_i such that for each $p \subseteq \mathsf{sub}_i(\psi)$ we have that $p \in \mathsf{part}_i(\psi)$ iff $\tau_p^i \models lc_i$. The formulas lc_i is a conjunction of clauses as follows:

- For $\varphi = \neg\varphi' \in \mathsf{sub}_i(\psi)$, we have the clause $x_{\varphi,i} \to \neg x_{\varphi',i}$.
- For $\varphi = \varphi' \wedge \varphi'' \in \mathsf{sub}_i(\psi)$, we have the clauses $x_{\varphi,i} \to x_{\varphi',i}$ and $x_{\varphi,i} \to x_{\varphi'',i}$.
- For $\varphi = \varphi' \vee \varphi'' \in \mathsf{sub}_i(\psi)$, we have the clause $x_{\varphi,i} \to (x_{\varphi'',i} \vee x_{\varphi'',i})$.

For $i = d$ we simply take f_d to be lc_d. Indeed, we have $P_d = \{p | p \in \mathsf{part}_d(\psi)\} = \{p \subseteq \mathsf{sub}_d(\psi) | \tau_p^d \models f_d\}$. Thus, f_d characterizes $Initial_d(\psi)$.

For $i < d$, suppose we already constructed a QBF formula f_{i+1} that characterizes P_{i+1}. We start by constructing f_i', which also characterizes P_i. We let $f_d' = f_d$. The propositional part of f_i' is lc_i, which describes the particles in $\mathsf{part}_i(\psi)$. In addition, for each $\Diamond\varphi \in \mathsf{sub}_i(\psi)$, we need a conjunct $mc_{\Diamond\varphi}$ that says that if $\Diamond\varphi$ is in a particle $p \in P_i$, then $\Diamond\varphi$ in p is witnessed by a particle in P_{i+1}. That is, we define $mc_{\Diamond\varphi}$ as $x_{\Diamond\varphi,i} \to \exists x_{\theta,i+1:\{\theta \in \mathsf{sub}_{i+1}(\psi)\}}(f_{i+1} \wedge x_{\varphi,i+1} \wedge tr_i)$, where tr_i is the formula $\bigwedge_{\Box\eta \in \mathsf{sub}_i(\psi)}[x_{\Box\eta,i} \to x_{\eta,i+1}]$. (Here the existential quantifier is a sequence $\exists x_i \exists \ldots \exists x_j$ of quantifiers,one for each of the formulas in $\mathsf{sub}_{i+1}(\psi)$.)

Lemma 1. *If f_{i+1}' characterizes P_{i+1}, then f_i' characterizes $P_i = Iterate(P_{i+1}, i)$.*

Proof. By construction, lc_i characterizes $Initial_i(\psi)$. For the witnessing requirement, we can see if $\tau_p^i \models mc_{\Diamond\varphi}$ and $x_{\Diamond\varphi,i}$, then there is assignment $\tau_{p'}^{i+1}$ where $\tau_p^i, \tau_{p'}^{i+1} \models f_{i+1}' \wedge x_{\varphi,i+1} \wedge tr_i$. This is equivalent to $p' \in P_{i+1}$, $\varphi \in p'$ and $R_i(p, p')$. So the lemma holds.

Corollary 2. *ψ is satisfiable iff $\exists x_{\theta,0:\{\theta \in \mathsf{sub}_0(\psi)\}} x_{\psi,0} \wedge f_0'$ is satisfiable.*

Proof. Immediate from the soundness and completeness of $\mathcal{KBDD}$.

This reduction of $\mathcal{K}$ to QBF is correct; unfortunately, it is not polynomial. The problem is that f_i' requires a distinct copy of f_{i+1} for each formula $\Diamond\varphi$ in $\mathsf{sub}_i(\psi)$. This may cause an exponential blow-up for f_0'. We would like f_i to use only one copy of f_{i+1}. We do this by replacing the conjunction over all $\Diamond\varphi$ formulas in $\mathsf{sub}_i(\psi)$ by a universal quantification. Let k be an upper bound on the number of $\Diamond\varphi$ formulas in $\mathsf{sub}_i(\psi)$. We associate an index $j \in \{0, \ldots, k-1\}$ with each such subformula; thus, we let ξ_j^i the j-th $\Diamond\varphi$ subformula in $\mathsf{sub}_i(\psi)$, in which case we denote φ by $\mathsf{strip}(\xi_j^i)$. Let $m = \lceil lg\ k \rceil$. We introduce m new Boolean variables $y_1, \ldots, y_m$. Each truth assignment to these variables induce a number between 0 and $k-1$. We refer to this number is $val(\mathbf{y})$ and we use it to point to $\Diamond$ subformulas. Let $witness_i$ be the formula $\bigvee_{j=0}^{k-1} x_{\xi_j^i}$, which asserts that some witnesses are required.

We can now write f_i in a compact fashion:

$$lc_i \wedge \forall y_1, \ldots, \forall y_m : \exists x_{\theta,i+1:\{\theta \in \mathsf{sub}_{i+1}(\psi)\}} : witness_i \to$$

$$\left(f_{i+1} \wedge tr_i \wedge \bigwedge_{j=0}^{k-1} (val(\mathbf{y}) = j \wedge x_{\xi_j^i,i}) \to x_{\mathsf{strip}(\xi_j^i),i+1}) \right).$$

f_i first asserts the local consistency constraint lc_i. The quantification on $y_1, \ldots, y_m$ simulates the conjunction on all k $\diamond$ subformulas in $\mathsf{sub}_i(\psi)$. We then check if $witness_i$ holds, in which case we assert the existence of the witnessing particle. We use f_{i+1} to ensure that this particle is in P_{i+1} and tr_i to ensure satisfaction of $\Box$ subformulas. Finally, we let $val(\mathbf{y})$ point to the $\diamond$ subformulas that needs to be witnesses. Note that f_i contains only one copy of f_{i+1}.

Lemma 2. *If f_{i+1} characterizes P_{i+1}, then f_i characterizes $P_i = Iterate(P_{i+1}, i)$.*

Corollary 3. *ψ is satisfiable iff $\exists x_{\theta,0:\{\theta \in \mathsf{sub}_0(\psi)\}} x_{\psi,0} \wedge f_0$ is satisfiable.*

Proof. Both follows from the fact that $f_i \leftrightarrow f_i'$ by construction.

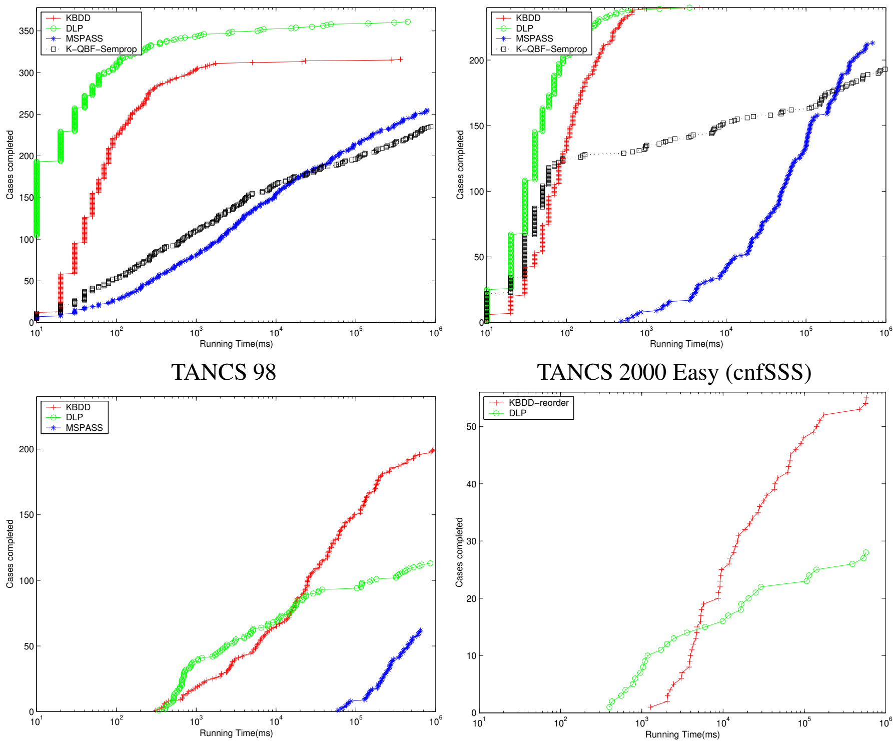
We implemented this approach by optimizing the translation further. As in the BDD-based implementation, we represent only Boolean literals, $\Box$ subformulas and $\diamond$ subformulas with Boolean variables. The other subformulas are not represented explicitly, but are logically implied.

Results

In Figure 2 we see that on the TANCS 98 benchmarks, DLP has the best performance, but on the more challenging TANCS 2000 benchmarks, $\mathcal{KBDD}$ outperformed the other solvers, especially on the harder portions of the suite. MSPASS was a distant third, especially on the harder formulas, and is omitted on the hard formulas of TANCS 2000. Reducing $\mathcal{K}$ to a search-based QBF solver completed the least number of cases, solving only a handful in the medium and hard difficulty classes of TANCS 2000 so we did not report the results. With comparatively more research going into QBF solvers, this approach might show promise in the future.

A different perspective on the comparison between DLP, a search-based solver, and $\mathcal{KBDD}$, a symbolic solver, is demonstrated on random modal-CNF formulas. We plot here median running time (16 samples per data point) as a function of density (L/N) to demonstrate the difference between the behavior of the two solvers.

As we can see in Figure 3, for $d = 1$, DLP demonstrates the bell-shaped "easy-hard-easy" pattern that is familiar from random propositional CNF formulas [29] and random QBF formulas [9]. In contrast, for $\mathcal{KBDD}$ we see an increase in running time as a function of the density; that is, the higher the density the harder the problem for $\mathcal{KBDD}$. This is consistent with known results on the performance of BDD-based algorithm for random propositional CNF formulas [6]. For each modal level, $\mathcal{KBDD}$ builds a BDD for the appropriate particle set. With increased density, the construction of these BDDs gets quite challenging, often resulting in space outs or requiring extensive variable reordering. (In the propositional case, one can develop algorithms that avoid the construction of a monolithic BDD, cf. [27]. It would be interesting to try to apply such



TANCS 98 | TANCS 2000 Easy (cnfSSS)

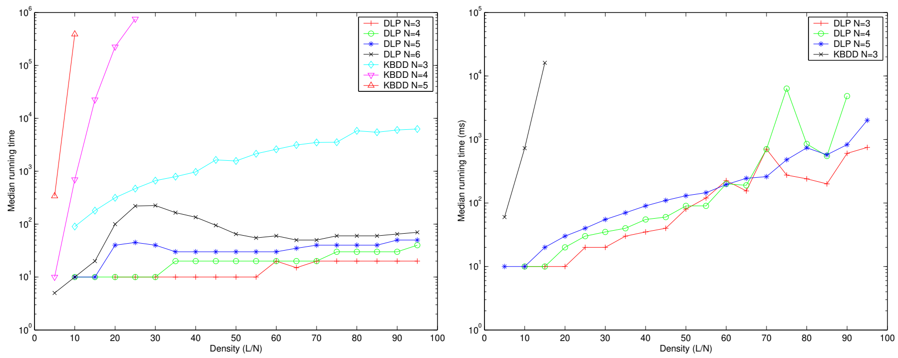
TANCS 2000 Medium (cnfLadn) | TANCS 2000 Hard (cnf)

Fig. 2. Comparison of $\mathcal{KBDD}$, DLP, SEMPROP/QBF and MSPASS on $\mathcal{K}$ formulas

ideas for $\mathcal{KBDD}$.) This explains why DLP performs much better than $\mathcal{KBDD}$ on random modal-CNF formulas. Unlike the benchmark formulas of TANCS 98 and TANCS 2000, the random modal-CNF formulas have a very high propositional complexity (low modal depth). In contrast, the formulas in TANCS 98 and TANCS 2000 have high modal complexity (high modal depth). Our conclusion is that DLP is better suited for formulas with high propositional complexity, while $\mathcal{KBDD}$ is better suited for formulas with high modal complexity.

5 Conclusion

We studied optimization issues for BDD-based satisfiability solvers for the modal logic $\mathcal{K}$. We focused on two types of optimizations. First we studied variable ordering, which is known to be of critical importance to BDD-based algorithms. Second, we studied formulas simplification based on modal extensions of the pure-literal rule. Our results show that the payoff of the variable-ordering optimization is rather modest, while the payoff of the pure-literal optimization is quite significant. We benchmarked $\mathcal{KBDD}$, our optimized solver, against both native solvers (DLP) and translation-based solvers (MSPASS and SEMPROP). Our results indicate that the BDD-based approach dominates for modally heavy formulas, while search-based approaches dominate for propo-



$d = 1$ $\qquad$ $d = 2$

Fig. 3. Comparison of DLP and $\mathcal{KBDD}$ on Random formulas

sitionally heavy formulas. Further research is required to quantify the distinction between propositionally heavy and modally heavy formulas. This might enable the development of a combined solver, which invokes the appropriate engine for the formula under test. Another approach would be to develop a a hybrid solver, combining BDD-based and search-based techniques (cf. [11] for a hybrid approach in model checking), which would perform well on both modally heavy and propositionally heavy formulas. We leave this for future research.

References

1. C. Areces, R. Gennari, J. Heguiabehere, and M. de Rijke. Tree-based heuristics in modal theorem proving. In *Proc. of the ECAI'2000*, 2000.
2. F. Baader and S. Tobies. The inverse method implements the automata approach for modal satisfiability. In *Proc. of IJCAR'01*, volume 2083 of *LNAI*, pages 92–106.
3. P. Blackburn, M. de Rijke, and Y. Venema. *Modal logic*. Camb. Univ. Press, 2001.
4. R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, Vol. C-35(8):677–691, August 1986.
5. M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified Boolean formulae and its experimental evaluation. Technical report, Dipartmento di Imformatica e Sistemistica, Universita de Roma, 1999.
6. C. Coarfa, D.D. Demopoulos, A. San Miguel Aguirre, D. Subramanian, and M.Y. Vardi. Random 3-SAT: The plot thickens. In *Proc. of the Int. Conf. on Constraint Prog.*, 2000.
7. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5:394–397, 1962.
8. K. Etessami and G.J. Holzmann. Optimizing Büchi automata. In *CONCUR 2000*, pages 153–167, 2000.
9. I. Gent and T. Walsh. Beyond NP: The QSAT phase transition. In *AAAI: 16th National Conference on Artificial Intelligence*. AAAI / MIT Press, 1999.
10. F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedure - the case study of modal K(m). *Inf. and Comp.*, 162:158–178, 2000.
11. A. Gupta, Z. Yang, P. Ashar, L. Zhang, and S. Malik. Partition-based decision heuristics for image computation using SAT and BDDs. In *ICCAD*, pages 286–292, 2001.

12. J.Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.
13. A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. Technical report, Universität Bern, Switzerland, 1996.
14. U. Hustadt and R. Schmidt. MSPASS: modal reasoning by translation and first order resolution. In *Proc. of TABLEAUX 2000*, pages 67–71, 2000.
15. G. Kamhi and L. Fix. Adaptive variable reordering for symbolic model checking. In *ICCAD 1998*, pages 359–365, 1998.
16. R.E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput.*, 6(3):467–480, 1977.
17. R. Letz. Lemma and model caching in decision procedures for quantified Boolean formulas. In *TABLEAUX 2002*, volume 2381 of *LNAI*, pages 160–175, 2002.
18. F. Massacci and F.M. Donini. Design and results of TANCS-2000. In *Proc. of TABLEAUX 2000*, pages 52–56, 2000.
19. H.J. Ohlbach, A. Nonnengart, M. de Rijke, and D.M. Gabbay. Encoding two-valued non-classical logics in classical logic. In *Handbook of Automated Reasoning*. Elsevier, 1999.
20. G. Pan. BDD-based decision procedures for modal logic K, Master's Thesis, Rice University, 2002.
21. G. Pan, U. Sattler, and M.Y. Vardi. BDD-based decision procedures for K. In *Proc. of CADE 2002*, volume 2392 of *LNAI*, pages 16–30, 2002.
22. P.F. Patel-Schneider and I. Horrocks. DLP and FaCT. In *Analytic Tableaux and Related Methods*, pages 19–23, 1999.
23. P.F. Patel-Schneider and R. Sebastiani. A new system and methodology for generating random modal formulae. In *IJCAR 2001*, pages 464–468, 2001.
24. V.R. Pratt. A near-optimal method for reasoning about action. *Journal of Computer and System Sciences*, 20(2):231–254, 1980.
25. J. Rintanen. Constructing conditional plans by a theorem-prover. *J. of A. I. Res.*, 10:323–352, 1999.
26. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *ICCAD'93*, pages 42–47, 1993.
27. A. San Miguel Aguirre and M.Y. Vardi. Random 3-SAT and BDDs: The plot thickens further. In *CP01*, 2001.
28. R. A. Schmidt. *Optimised Modal Translation and Resolution*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 1997.
29. B. Selman, D.G. Mitchell, and H.J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.
30. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *CAV 2000*, pages 247–263, 2000.
31. L.J. Stockmeyer. The polynomial-time hierarchy. *Theo. Comp. Sci.*, 3:1–22, 1977.
32. G. Sutcliffe and C. Suttner. Evaluating general purpose automated theorem proving systems. *Artificial intelligence*, 131:39–54, 2001.
33. A. Tacchella. *SAT system description. In *Collected Papers from (DL'99). CEUR*, 1999.
34. S. Tani, K. Hamaguchi, and S. Yajima. The complexity of the optimal variable ordering problems of shared binary decision diagrams. In *ISAAC*, 1993.
35. J. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, 1983.
36. M.Y. Vardi. What makes modal logic so robustly decidable? In N. Immerman and Ph.G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, pages 149–183. AMS, 1997.
37. A. Voronkov. How to optimize proof-search in modal logics: new methods of proving redundancy criteria for sequent calculi. *Comp. Logic*, 2(2):182–215, 2001.