BDD-Based Decision Procedures for $\mathcal{K}$

Guoqiang Pan[1]*, Ulrike Sattler[2], Moshe Y. Vardi[1]*

[1] Department of Computer Science, Rice University, Houston, TX
gqpan,vardi@cs.rice.edu
[2] Institut für Theoretische Informatik, TU Dresden, Germany
sattler@tcs.inf.tu-dresden.de

Abstract. We describe BDD-based decision procedures for $\mathcal{K}$. Our approach is inspired by the automata-theoretic approach, but we avoid explicit automata construction. Our algorithms compute the fixpoint of a set of types, which are sets of formulas satisfying some consistency conditions. We use BDDs to represent and manipulate such sets. Experimental results show that our algorithms are competitive with contemporary methods using benchmarks from TANCS 98 and TANCS 2000.

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 $\mathcal{K}$ [14] and describe a new approach to decide the satisfiability of formulas in this logic. 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 [18, 14]. 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-Longemann-Loveland 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 [13, 27]. Another approach to modal satisfiability, the inverse calculus for $\mathcal{K}$ [30] 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 [28, 21]. Recently, it has been shown that, by encoding the modal depth infomation into the translation, a first-order theorem prover can be used efficiently for deciding modal satisfiability [2]. The latter approach works nicely with a resolution-based first-order theorem prover, which can be used as a decision procedure for modal satisfiability by using appropriate resolution strategies [16]. Other approaches for modal satisfiability such as mosaics, type elimination,

* Supported in part by NSF grants CCR-9700061, CCR-9988322, IIS-9908435, IIS-9978135, and EIA-0086264, by BSF grant 9800096, and by a grant from the Intel Corporation.

or automata-theoretic approaches are well-suited for proving exact upper complexity bounds, but are rarely used in actual implementations [5, 14, 29].

The algorithms presented here are inspired by the automata-theoretic approach for logics with the tree-model property [29]. 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. In our approach, we combine the two steps and apply the non-emptiness test without explicitly constructing the automaton. As was pointed out in [3], the inverse method described in [30] 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 presented here 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 [30], while the top-down approach is reminiscent of the "type-elimination" method developed for Propositional Dynamic Logic in [23].

The key idea underlying our implementation is that of representing sets of types and operating on them symbolically. Our implementation uses Binary Decision Diagrams (BDDs) [6]: BDDs are a compact representation of propositional formulas, and commonly used as a compact representation of states. One of their advantages is that they come with efficient operations for certain manipulations on BDDs. This paper consists of a viability study for our approach. To see whether it yields competitive algorithms, we used existing benchmarks of modal formulas, TANCS 98 [15] and TANCS 2000 [19], and we compared our algorithms with *SAT [27] and DLP [22]. A straightforward implementation of our approach did not yield a competitive algorithm, but an optimized implementation did yield a competitive algorithm (see Fig. 1) indicating the viability of our approach.

The paper is organized as follows. After introducing the modal logic $\mathcal{K}$ in Section 2, we present our algorithms and discuss how they can be implemented using BDD packages in Section 3. In Section 4, we discuss three optimizations that we applied, and compare in Section 5 the performance of our implementations with *SAT on formulas from TANCS 98 and TANCS 2000.

2 Preliminaries

In this section, we introduce the syntax and semantics of the modal logic $\mathcal{K}$, as well as types and how they can be used to encode a Kripke structure.

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 Boolean operators $\wedge$ and $\neg$ and the unary modality $\Box$. As usual, we use other Boolean operators 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 $K = \langle V, W, R, L\rangle$, where V is a set (containing Φ) of propositions, W is a set of possible worlds, $R \subseteq W \times W$ is the accessibility relation on worlds, and $L : W \to V \to \{0, 1\}$ a labeling function for each state. The notion of a formula φ being *satisfied* in a world w of a Kripke structure K (written as $K, w \models q$) is inductively defined as follows:

- $K, w \models q$ for $q \in \Phi$ iff $L(w)(q) = 1$
- $K, w \models \varphi \wedge \psi$ iff $K, w \models \varphi$ and $K, w \models \psi$
- $K, w \models \neg\varphi$ iff $K, w \not\models \varphi$
- $K, w \models \Box\varphi$ iff, for all w', if $(w, w') \in R$, then $K, w' \models \varphi$

The abbreviated operators can be defined as follows:

- $K, w \models \varphi \vee \psi$ iff $K, w \models \varphi$ or $K, w \models \psi$
- $K, w \models \Diamond\varphi$ iff there exists w' with $(w, w') \in R$ and $K, w' \models \varphi$.

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

To simplify the following considerations, we restrict our attention to formulas in a certain normal form. A formula ψ of $\mathcal{K}$ is said to be in *box normal form* (BNF) if all its subformulas are of the form $\varphi \wedge \varphi'$, $\varphi \vee \varphi'$, $\Box\varphi$, $\neg\Box\varphi$, q, or $\neg q$ where $q \in AP(\psi)$. All $\mathcal{K}$ formulas can be obviously converted into BNF by pushing negation inwards and, if not stated otherwise, we assume all formulas to be in BNF. The *closure* of a formula $\mathsf{cl}(\psi)$ is defined as the smallest set such that, for all subformula φ of ψ, if φ is not $\neg\varphi'$, then $\{\varphi, \neg\varphi\} \subseteq \mathsf{cl}(\psi)$.

Our algorithms will work on *types*, i.e., sets of (sub)formulas that are consistent w.r.t. the Boolean operators, and where (negated) box formulas are treated as atoms. A set $a \subseteq \mathsf{cl}(\psi)$ of formulas is called a ψ-*type* (or simply a type if ψ is clear from the context) if it satisfies the following conditions:

- If $\varphi = \neg\varphi'$, then $\varphi \in a$ iff $\varphi' \notin a$.
- If $\varphi = \varphi' \wedge \varphi''$, then $\varphi \in a$ iff $\varphi' \in a$ and $\varphi'' \in a$.
- If $\varphi = \varphi' \vee \varphi''$, then $\varphi \in a$ iff $\varphi' \in a$ or $\varphi'' \in a$.

For a set A of types, we define the relation $\Delta \subseteq A \times A$ as follows:

$$\Delta(a, a') \text{ iff for all } \Box\varphi' \in a, \text{ we have } \varphi' \in a'.$$

Given a set $A \subseteq 2^{\mathsf{cl}(\psi)}$ of types, we can construct a Kripke structure K_A using the relation Δ as follows: $K_A = \langle AP(\psi), A, \Delta, L\rangle$ with $L(a)(q) = 1$ iff $q \in a$. Then we almost have that, for all $\varphi \in \mathsf{cl}(\psi)$:

$$K, a \models \varphi \text{ iff } \varphi \in a. \qquad (*)$$

The only reason why $(*)$ might be false is due to formulas of the form $\neg\Box\varphi \in a$: it might be the case that $\varphi \in b$ for all b with $\Delta(a, b)$, i.e., a negated box formula might not be "witnessed".

3 Our Algorithms

The two algorithms presented here take a certain "initial" set of types and apply repeatedly a monotone operator to it. If this application reaches a fixpoint, we can show that it yields a set of types where the above construction yields indeed a Kripke structure that satisfies the condition $(*)$, i.e., all negated box formulas are indeed "witnessed" by some $b \in A$. This Kripke structure is then a model of ψ iff $\psi \in a$ for some $a \in A$.

The first algorithm follows a "top-down" approach, i.e., it starts with the set $A \subseteq 2^{\mathsf{cl}(\psi)}$ of all types, and the monotone operator removes those types containing negated box formulas which are not witnessed in the current set of types. Dually, the second, "bottom-up", approach starts with the set of types that do not contain negated box formulas, and then adds those types whose negated box formulas are witnessed in the current set of types.

Both algorithms follow the following scheme, in which X, X' are sets of types:

$X = Initial(\psi)$
repeat
  $X' \Leftarrow X$
  $X \Leftarrow Iterate(X')$
until $X = X'$
if exists $x \in X$ such that $\psi \in x$ **then return** "ψ is satisfiable"
else return "ψ is not satisfiable"
endif

Since this algorithm works on a fixed set of types and uses a monotone operator $Iterate(\cdot)$, it obviously terminates. In fact, we can show that it will terminate in $d + 1$ iterations, where d is the modal nesting depth of the input formula ψ. It remains to define $Initial(\psi)$ and $Iterate(\cdot)$.

3.1 Top-Down Approach

The top-down approach is closely related to the type elimination approach which is, in general, used for more complex modal logics, see, e.g., Section 6 of [14]. For the algorithm pursuing the top-down approach, the functions $Initial(\psi)$ and $Iterate(\cdot)$ are defined as follows:

- $Initial(\psi)$ is the set of *all* ψ-types.
- $Iterate(A) := A \setminus \mathsf{bad}(A)$, where $\mathsf{bad}(A)$ are the types in A that contain unwitnessed negated box formulas. More precisely,

$$\mathsf{bad}(A) := \{a \in A \mid \text{there exists } \neg\Box\varphi \in a \text{ and, for all } b \in A \text{ with } \Delta(a, b), \text{ we have } \varphi \in b\}.$$

3.2 Bottom-Up Approach

As mentioned above, the algorithm pursuing the bottom-up approach starts with a small set of types (i.e., those without negated box formulas), and repeatedly adds those types whose negated box formulas are witnessed in the current set. More precisely, for the bottom-up approach, the functions $Initial(\psi)$ and $Iterate(\cdot)$ are defined as follows:

- $Initial(\psi)$ is the set of all those types that do not require any witnesses, i.e., they do not contain any negated box formula or, equivalently, they contain all positive box formulas in $\mathsf{cl}(\psi)$:

$$Initial(\psi) := \{a \subseteq \mathsf{cl}(\psi) \mid a \text{ is a type and } \Box\varphi \in a \text{ for each } \Box\varphi \in \mathsf{cl}(\psi)\}.$$

- $Iterate(A) := A \cup \mathsf{supp}(A)$, where $\mathsf{supp}(A)$ is the set of those types whose negated box formulas are witnessed by types in A. More precisely,

$$\mathsf{supp}(A) := \{a \subseteq \mathsf{cl}(\psi) \mid a \text{ is a type and for all } \neg\Box\varphi \in a, \text{ there exists } b \in A \text{ with } \neg\varphi \in b \text{ and } \Delta(a,b)\}.$$

  We say that a type in $\mathsf{supp}(A)$ is *witnessed* by a type in A.

3.3 Implementations

We use Binary Decision Diagrams (BDDs) [6, 1] to represent sets of types. BDDs, or more precisely, Reduced Ordered Binary Decision Diagrams (ROBDDs), are obtained from binary decision trees by following a fixed variable splitting order and by merging nodes that have identical child-diagrams. BDDs provide a canonical form of representation for Boolean functions. Experience has shown that BDDs provide a very compact representation for very large Boolean functions. Consequently, over the last decade, BDDs have had a dramatic impact in the areas of synthesis, testing, and verification of digital systems [4, 7]

In this section, we describe how our two algorithms are implemented using BDDs. First, we define a *bit-vector representation* of types. Since types are complete in the sense that either a subformula or its negation must belong to a type, it is possible for a formula and its negation to be represented using a single BDD variable.

The representation of types $a \subseteq \mathsf{cl}(\psi)$ as bit vectors is defined as follows: Since both formulas and their negations are in $\mathsf{cl}(\psi)$, we define

$$\mathsf{cl}_+(\psi) = \{\varphi_i \in \mathsf{cl}(\psi) \mid \varphi_i \text{ is not of the form } \neg\varphi'\},$$
$$\mathsf{cl}_-(\psi) = \{\neg\varphi \mid \varphi \in \mathsf{cl}_+(\psi)\},$$

and use m for $|\mathsf{cl}_+(\psi)| = |\mathsf{cl}(\psi)|/2$. For $\mathsf{cl}_+(\psi) = \{\varphi_1, \ldots \varphi_m\}$, a vector $\boldsymbol{a} = \langle a_1, \ldots, a_m\rangle \in \{0,1\}^m$ represents a set[1] $a \subseteq \mathsf{cl}(\psi)$ with of $\varphi_i \in a$ iff $a_i = 1$. A set of such bit vectors can obviously be represented using a BDD with m variables. It remains to "filter out" those bit vectors that represent types.

We define $Consistent_\psi$ as the characteristic predicate for types: $Consistent_\psi(\boldsymbol{a}) = \bigwedge_{1\le i\le m} Cons_i(\boldsymbol{a})$, where $Cons_i(\boldsymbol{a})$ is defined as follows:

[1] Please note that this set is not necessarily a type.

– if φ_i is neither of the form $\varphi' \wedge \varphi''$ nor $\varphi' \vee \varphi''$, then $Cons_i(\boldsymbol{a}) = \mathbf{1}$,
– if $\varphi_i = \varphi' \wedge \varphi''$, then $Cons_i(\boldsymbol{a}) = (a_i \wedge a' \wedge a'') \vee (\neg a_i \wedge (\neg a' \vee \neg a''))$,
– if $\varphi_i = \varphi' \vee \varphi''$, then $Cons_i(\boldsymbol{a}) = (a_i \wedge (a' \vee a'')) \vee (\neg a_i \wedge \neg a' \wedge \neg a'')$,

where $a' = a_\ell$ if $\varphi' = \varphi_\ell \in \mathsf{cl}_+(\psi)$, and $a' = \neg a_\ell$ if $\varphi' = \neg\varphi_\ell$ for $\varphi_\ell \in \mathsf{cl}_+(\psi)$ (and analogously for a''). From this, the implementation of $Initial$ is fairly straight forward: For the top-down algorithm,

$$Initial(\psi) := \{\boldsymbol{a} \in \{0,1\}^m \mid Consistent_\psi(\boldsymbol{a})\},$$

and for the bottom-up algorithm,

$$Initial(\psi) := \{\boldsymbol{a} \in \{0,1\}^m \mid Consistent_\psi(\boldsymbol{a}) \wedge \bigwedge_{\varphi_i = \Box\varphi'} a_i = 1\}.$$

In the following, we do not distinguish between a type and its representation as a bit vector $\boldsymbol{a}$. Next, to specify $\mathsf{bad}(\cdot)$ and $\mathsf{supp}(\cdot)$, we define auxiliary predicates:

– $\Diamond_{1,i}(\boldsymbol{x})$ is read as "$\boldsymbol{x}$ needs a witness for a diamond operator at position i" and is true iff $x_i = 0$ and $\varphi_i = \Box\varphi'$.
– $\Diamond_{2,i}(\boldsymbol{y})$ is read as "$\boldsymbol{y}$ is a witness for a negated box formula at position i" and is true iff $\varphi_i = \Box\varphi_j$ and $y_j = 0$ or $\varphi_i = \Box\neg\varphi_j$ and $y_j = 1$.
– $\Box_{1,i}(\boldsymbol{x})$ is read as "$\boldsymbol{x}$ requires support for a box operator at position i" and is true iff $x_i = 1$ and $\varphi_i = \Box\varphi'$.
– $\Box_{2,i}(\boldsymbol{y})$ is read as "$\boldsymbol{y}$ provides support for a box operator at position i" and is true iff $\varphi_i = \Box\varphi_j$ and $y_j = 1$ or $\varphi_i = \Box\neg\varphi_j$ and $y_j = 0$.

For a set A of types, we construct the BDD that represents the "maximal" accessibility relation Δ, i.e., a relation that includes all those pairs $(\boldsymbol{x}, \boldsymbol{y})$ such that $\boldsymbol{y}$ supports all of $\boldsymbol{x}$'s box formulas. For types $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^m$, we define

$$\Delta(\boldsymbol{x}, \boldsymbol{y}) = \bigwedge_{1 \le i \le m} (\Box_{1,i}(\boldsymbol{x}) \rightarrow \Box_{2,i}(\boldsymbol{y})).$$

Given a set A of types, we write the corresponding characteristic function as χ_A. Both the top-down and the bottom-up algorithm can be defined using the predicates χ_A, Δ, $\Diamond_{j,i}$, and $\Box_{j,i}$.

The predicate bad is true on those types that contain a negated box formula $\varphi_i = \neg\Box\varphi_j$ that is not witnessed in the current set of types. The corresponding predicate for bit vectors χ_{bad_i} can then be written as follows:

$$\chi_{\mathsf{bad}_i(X)}(\boldsymbol{x}) = \Diamond_{1,i}(\boldsymbol{x}) \wedge \forall \boldsymbol{y} : ((\chi_X(\boldsymbol{y}) \wedge \Delta(\boldsymbol{x}, \boldsymbol{y})) \rightarrow \neg\Diamond_{2,i}(\boldsymbol{y})),$$

and thus $\mathsf{bad}(X)$ can be written as $\chi_{\mathsf{bad}(X)}(\boldsymbol{x}) = \bigvee_{1 \le i \le m} \chi_{\mathsf{bad}_i(X)}(\boldsymbol{x})$.

In our implementation, we compute the characteristic function $\chi_{\overline{\mathsf{bad}_i(X)}}$ of the complement of each $\mathsf{bad}_i(X)$ and use it in the implementation of the top-down and the bottom-up algorithm. It is easy to see that $\chi_{\overline{\mathsf{bad}_i(X)}}$ is equivalent to

$$\Diamond_{1,i}(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y} : (\chi_X(\boldsymbol{y}) \wedge \Delta(x, y) \wedge \Diamond_{2,i}(\boldsymbol{y})).$$

For the top-down algorithm, the $Iterate$ function can be written as:

$$\chi_{X \setminus \mathsf{bad}(X)} := \chi_X(\boldsymbol{x}) \wedge \bigwedge_{1 \leq i \leq m} (\chi_{\overline{\mathsf{bad}_i(X)}}(\boldsymbol{x}))$$

For the bottom-up algorithm, additionally, we must take care of only adding those bit vectors representing types, and so the $Iterate$ function can be implemented as:

$$\chi_{X \cup \mathsf{supp}(X)} := \chi_X(\boldsymbol{x}) \vee (\chi_{Consistent_\psi}(\boldsymbol{x}) \wedge \bigwedge_{1 \leq i \leq m} (\chi_{\overline{\mathsf{bad}_i(X)}}(\boldsymbol{x}))$$

These functions can be written more succinctly using the pre-image function for the relation Δ:

$$\mathsf{preim}_\Delta(\chi_N)(\boldsymbol{x}) = \exists \boldsymbol{y} : \chi_N(\boldsymbol{y}) \wedge \Delta(\boldsymbol{x}, \boldsymbol{y}).$$

Using pre-images, we can rewrite $\chi_{\overline{\mathsf{bad}_i(X)}}$ as follows:

$$\chi_{\overline{\mathsf{bad}_i(X)}}(\boldsymbol{x}) = \Diamond_{1,i}(\boldsymbol{x}) \rightarrow \mathsf{preim}_\Delta(\chi_X(\boldsymbol{y}) \wedge \Diamond_{2,i}(\boldsymbol{y})).$$

Finally, the bottom-up algorithm is implemented as an iteration over the sets $\chi_{X \cup \mathsf{supp}(X)}$, and the top-down algorithm is implemented as iterations over $\chi_{X \setminus \mathsf{bad}(X)}$. Both stop when a fixpoint is reached. Then checking whether ψ is present in a type of this fixpoint is trivial.

The pre-image operation is a key operation in both the bottom-up and the top-down approaches. It is also known to be a key operation in symbolic model checking [7] and it has been the subject of extensive research (cf. [8, 11, 24, 9]) since it can be a quite time and space consuming operation. Various optimizations can be applied to the pre-image computation to reduce the time and space requirements. A method of choice is that of *conjunctive partitioning* combined with *early quantification.* The idea is to avoid building a monolithic BDD for the relation Δ since this BDD can be quite large. Rather, we take advantage of the fact that Δ is defined as a conjunction of simple conditions. Thus, to compute the pre-image we have to evaluate a quantified Boolean formula of the form $(\exists y_1) \ldots (\exists y_n)(c_1 \wedge \ldots \wedge c_m)$, where the c_i's are Boolean formulas. Suppose, however, that a variable y_j does not occur in the clauses $c_{i+1}, \ldots, c_m$. Then the formula can be rewritten as

$$(\exists y_1) \ldots (\exists y_{j-1})(\exists y_{j+1}) \ldots (\exists y_n)((\exists y_j)(c_1 \wedge \ldots \wedge c_i) \wedge (c_{i+1} \wedge \ldots \wedge c_m)).$$

This enables us to apply existential quantification to smaller BDDs. Of course, there are many ways in which one can cluster and re-order the c_i's. We used the methodology developed in [24], called the "IWLS 95" methodology, to compute pre-images.

4 Optimizations

The decision procedures described above handles a formula in four steps. Firstly, the formula is converted into box normal form. Secondly, a set of bit vectors representing types is generated. Thirdly, this set is updated through a fixpoint process. Finally, the answer of the algorithm depends on a simple syntactic check of this fixpoint. In this section, we will describe three different optimization techniques, each related to a different step of the procedure.

4.1 Particles

In the approaches presented so far, we memorize and take care of redundant information: for example, a bit vector represents both a conjunction and the corresponding conjuncts, whereas the truth value of the former is determined by the truth value of the latter. Now we propose a representation where we only keep track of the "non-redundant" subformulas, which possibly reduces the size of the corresponding BDDs. To do so, it is convenient to work on formulas in a different normal form.

A $\mathcal{K}$ formula ψ is said to be in *negation normal form* (NNF) if all its subformulas are of the form $\varphi \wedge \varphi'$, $\varphi \vee \varphi'$, $\Box\varphi$, $\Diamond\varphi$, q, or $\neg q$ where $q \in AP(\psi)$. We write $NNF(\psi)$ for the NNF of ψ and $\mathsf{sub}(\psi)$ for the set of subformulas of $NNF(\psi)$. All $\mathcal{K}$ formulas can be converted 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 $\varphi'' \in p$.

Thus, in contrast to a type, a full particle may contain both φ' and φ'', but neither $\varphi' \wedge \varphi''$ nor $\varphi' \vee \varphi''$.

For particles, $\Delta(\cdot,\cdot)$ is defined as for types. From a set of particles P and the corresponding $\Delta(\cdot,\cdot)$, we can construct a Kripke structure K_P in the same way as from a set of types.

For the top-down approach, the auxiliary functions $Initial(\cdot)$ and $Iterate(\cdot)$ for full particles are defined as follows:

- $Initial(\psi)$ is the set of all full ψ-particles.
- $Iterate(P) = P - \mathsf{bad}(P)$, where $\mathsf{bad}(P)$ is the particles in P that contain unwitnessed diamond formulas, i.e.

$$\mathsf{bad}(P) = \{p \in P \mid \text{there exists } \Diamond\varphi \in p \text{ such that, for all } q \in P \text{ with } \Delta(p,q), \text{ we have } \varphi \notin q\}.$$

Analogously, these functions are defined for the bottom-up approach as follows:

- $Initial(\psi)$ is the set of full ψ-particle p that do not contain diamond formulas, i.e., $\Diamond\varphi \notin p$ for all $\Diamond\varphi \in \mathsf{sub}(\psi)$.
- $Iterate(P) = P \cup \mathsf{supp}(P)$, where $\mathsf{supp}(P)$ is the set of witnessed particles, i.e., .

$$\mathsf{supp}(P) = \{p \subseteq \mathsf{sub}(\psi) \mid p \text{ is a } \psi\text{-particle and, for all } \Diamond\varphi \in p, \text{ there exists } q \in P \text{ with } \varphi \in q \text{ and } \Delta(p,q)\}.$$

While encoding particle sets by BDDs may require more BDD variables, we still might see a reduction in BDD size because particles requires fewer constraints than types. [2] Besides a possible reduction in the size required to encode a bit-vector representation of particle sets, the particle-based approaches also can improve running time.

[2] Of course, BDD size is always formula dependent. In our experiments, we observed that particle approaches gives BDD sizes between a small constant factor (i.e., 2-3) larger to orders of magnitudes smaller compared to type approaches.

We can see that for each iteration, the number of pre-image operations a type based approach will need to do is equal to the total number of modal operators, while the corresponding number for particle based approaches is only equal to the number of diamond operators in the NNF form.

4.2 Lean Approaches

This optimization is also motivated by the idea to compress the size of the bit vector representing a type by omitting redundant information. To this purpose, we first define a set of "non-redundant" subformulas $\mathsf{atom}(\psi)$ as the set of those formulas in $\mathsf{cl}(\psi)$ that are neither conjunctions nor disjunctions, i.e., each φ is of the form $\Box\varphi'$, q, $\neg\Box\varphi'$, or $\neg q$. By the definition of types, each type $a \subseteq \mathsf{cl}(\psi)$ corresponds one-to-one to a *lean type* $a' := a \cap \mathsf{atom}(\psi)$. So storing types in lean form is equivalent to storing them in full form.

Analogously, we can define a lean representation for particles. First, we define the relevant subformulas $\mathsf{part}(\psi)$ as follows: For $\varphi \in \mathsf{sub}(\psi)$, if φ is $\Diamond\varphi'$, $\Box\varphi'$, q, or $\neg q$, then φ is in $\mathsf{part}(\psi)$. For a full particle $p \subseteq \mathsf{sub}(\psi)$, we define the corresponding *lean particle* p' as follows: $p' = p \cap \mathsf{part}(\psi)$. Because the (first) condition on particles is more relaxed than that of atoms, a lean particle does not correspond to a single full particle, but can represent several full particles. Although lean approaches can possibly reduce the size required for representing worlds, we have to pay for these savings since computing bad and supp using lean types and particles can be more complicated.

4.3 Level-based evaluation

As already mentioned, $\mathcal{K}$ has the finite-tree-model property, i.e., each satisfiable formula ψ of $\mathcal{K}$ has a finite tree model of depth bounded by the depth $md(\psi)$ of nested modal operators in ψ. Here, we take advantage of this property and, instead of representing a complete model using a set of particles or types, we represent each layer (i.e., all worlds being at the same distance from the root node) in the model using a separate set (for a level-based approach in the context of the first-order approach to $\mathcal{K}$, see [2]). Since only a subset of all subformulas appears in one layer, the representation can be more compact. We only present the optimization for the approach using (full) types. The particle approach and the lean approaches can be constructed analogously. For $0 \leq i \leq md(\psi)$, we write

$$\mathsf{cl}_i(\psi) := \{\varphi \in \mathsf{cl}(\psi) \mid \varphi \text{ occurs at modal depth } i \text{ in } \psi\},$$

and we adapt the definition of the possible accessibility relation Δ accordingly:

$$\Delta_i(a, a') \text{ iff } a \subseteq \mathsf{cl}_i,\ a' \subseteq \mathsf{cl}_{i+1}, \text{ and } \varphi' \in a' \text{ for all } \Box\varphi' \in a.$$

A sequence of sets of types $A = \langle A_0, A_1, \ldots, A_d\rangle$ with $A_i \subseteq 2^{\mathsf{cl}_i(\psi)}$ can be converted into a tree Kripke structure

$$K_A = \langle AP(\psi), \dot{\bigcup}_{0\leq i\leq d} A_i, R, L\rangle$$

(where the worlds are the disjoint union of the A_i) as follows:

– For a world $a \in A_i$ and $q \in AP(\psi)$, we define $L(a)(q) = 1$ if $q \in a$, and $L(a)(q) = 0$ if $q \notin a$.
– For a pair of states a, a', $R(w, w') = 1$ iff, for some i, $a \in A_i$ and $a' \in A_{i+1}$ and $\Delta_i(a, a')$.

The algorithm for level-based evaluation works as follows, where X_i are sets of types/particles:

$d = md(\psi)$
$X_d = Initial_d(\psi)$
for $i = d - 1$ downto 0 **do**
  $X_i \Leftarrow Iterate(X_{i+1}, i)$
end for
if exists $x \in X_0$ such that $\psi \in x$ **then return** "ψ is satisfiable"
else return "ψ is not satisfiable"
endif

Please note that this algorithm works bottom-up in the sense that it starts with the leaves of a tree model *at the deepest level* and then move up the tree model towards the root, adding nodes that are "witnessed". In contrast, the bottom-up approach presented earlier can be said to start with *all* leaves of a tree model.

For the level based algorithm on types, the auxiliary functions are defined as follows:

– $Initial_i(\psi) = \{a \subseteq \mathsf{cl}_i(\psi) \mid a \text{ is a type}\}$.
– $Iterate(A, i) = \{a \in Initial_i(\psi) \mid \text{for all } \neg\Box\varphi \in a \text{ there exists } b \in A \text{ where } \neg\varphi \in b \text{ and } \Delta_i(a, b)\}$.

For A a set of types of formulas at level $i + 1$, $Iterate(A, i)$ represents all types of formulas at level i that are properly witnessed in A.

5 Results

We implemented the aforementioned algorithms in C++ using the CUDD 2.3.0 [25] package for BDDs. The parser for the languages used in the benchmark suites are taken with permission from *SAT [27]. In the following, we describe and compare the performance of the different algorithms. [3]

As benchmarks, we used both the $\mathcal{K}$ part of TANCS 98 [15] and portions of the MODAL PSPACE division of TANCS 2000 [19], and compared our implementation with *SAT [27] and portions with DLP [22]. [4] We compared the different algorithms with respect to the number of benchmark formulas whose satisfiability they can decide within a specific time frame. In contrast to compare solvers formula by formula, this approach gives a global view of their performance, and the risk of being drawn into

[3] All the tests run on a Pentium 4 1.7GHz with 512MB of RAM, running Linux kernel version 2.4.2. The solver is compiled with gcc 2.96.

[4] Our goal was to test the viability of our approach by comparing our algorithms to a known competitive modal satisfiability solver. Thus, we chose *SAT and DLP as a representative solvers. We return to this point in the conclusions.

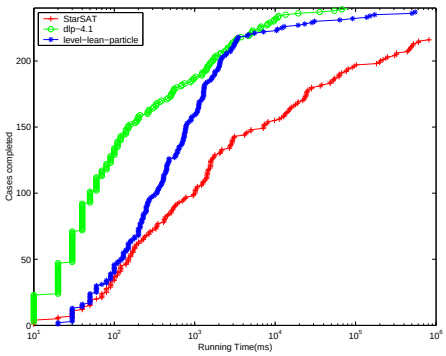

Fig. 1. Perf. on cnfSSS class of TANCS 2000

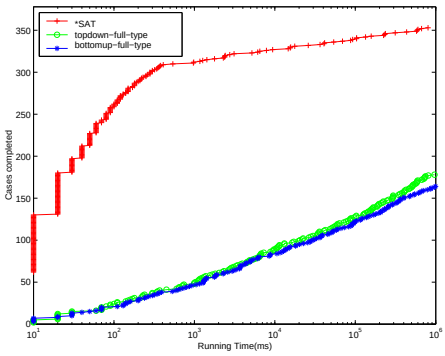

Fig. 2. Perf. on TANCS 98 (basic approaches)

too many details on how different solvers give preference to different formula classes is minimized [26, 12].

The time out is set at 1000s and the space limit for BDDs is set at 384MB. For each test case, we plot the number of formulas decided versus the time limit. Thus, the solver with a higher curve is faster than one with a lower curve. The chart is scaled so the full scale is the total number of cases in the benchmark.

5.1 TANCS 2000

The portion of TANCS 2000 we used is the easy-medium part of the Modal QBF formulas in the MODAL PSPACE division. The problems are generated as QBF formulas of different density, and encoded into $\mathcal{K}$ with the Schmidt-Schauss-Smolka translation (cnfSSS). TANCS 2000 also provides formulas from other translation schemes, namely the Ladner translation and the Halpern and Moses translation. However, we restricted our attention to cnfSSS since both *SAT and our algorithms could only handle very small and separate parts of the other formulas, which made a comparison meaningless. We also present results for DLP, a highly optimized tableaux engine.

With the optimizations described in this paper, we are able to achieve performance comparable to other provers. The results can be found in Fig. 1, where the performance of the level-based lean-particle approach represents our best BDD-based approach. This approach uses all the optimizations presented in this paper, and turned out to perform best of all our BDD-based approaches. We can see that, although the BDD-based approach has a higher overhead, it scales more gracefully than *SAT for the formula class used in this test. We are still slower than DLP.

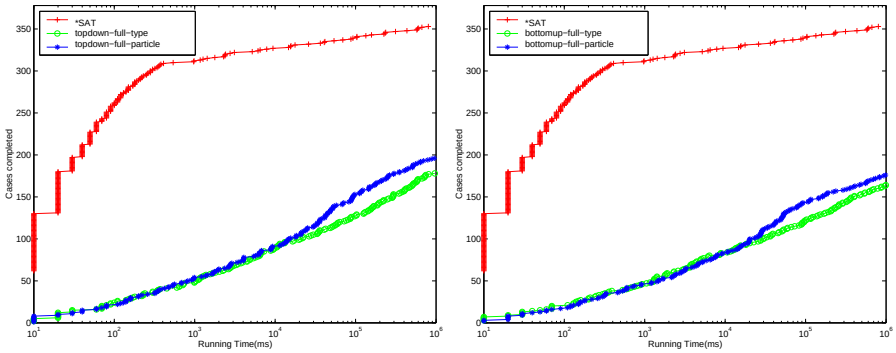

Fig. 3. Performance on TANCS 98 (particles vs. types)

5.2 TANCS 98

To analyze the usefulness of each optimization technique used, we run the algorithm with different optimization configurations on the $\mathcal{K}$ part of TANCS 98 benchmark suite [5] [15], a scalable benchmark which contains both provable and non-provable formulas.

The basic algorithms To compare our approaches, we first run the basic algorithms on TANCS 98. The results are presented in Fig. 2. We can see that *SAT clearly outperforms our two basic algorithms. An explanation of this "weak" behavior of our approaches is that the intermediate results of the preimage operation are so large that the BDDs space constraint is usually reached. Top-down slightly outperforms bottom-up since bottom-up requires an extra conjunction per iteration step for the $Consistent$ predicate.

Optimizations Now we compare the variants using types with their full particle-based variants. The results are presented in Fig. 3. We can see that, for TANCS 98, the particle approach slightly outperforms the type approach. Most of the improvements come from the use of negation normal form, which allows us to distinguish between diamonds and boxes, resulting in the reduction of the image operations needed.

Next, for types and particles, bottom-up and top-down, we compared the "full" approaches with their lean variants (see Fig. 4). Intuitively, the full variants trade a larger number of BDD variables in the representation of the transition relation for simpler consistency constraints. On TANCS 98, we can see that the lean approaches outperform in each combination their full variants. This shows that, as a general guideline, we should always attempt to reduce the number of BDD variables, since this results in smaller BDDs. Indeed, experience in symbolic model checking suggests that BDD size is typically the dominant factor when evaluating the performance of BDD-based algorithms [17].

[5] We did not use TANCS 2000 because unoptimized approaches time out on most of TANCS 2000 formulas, giving very little comparison between approaches.

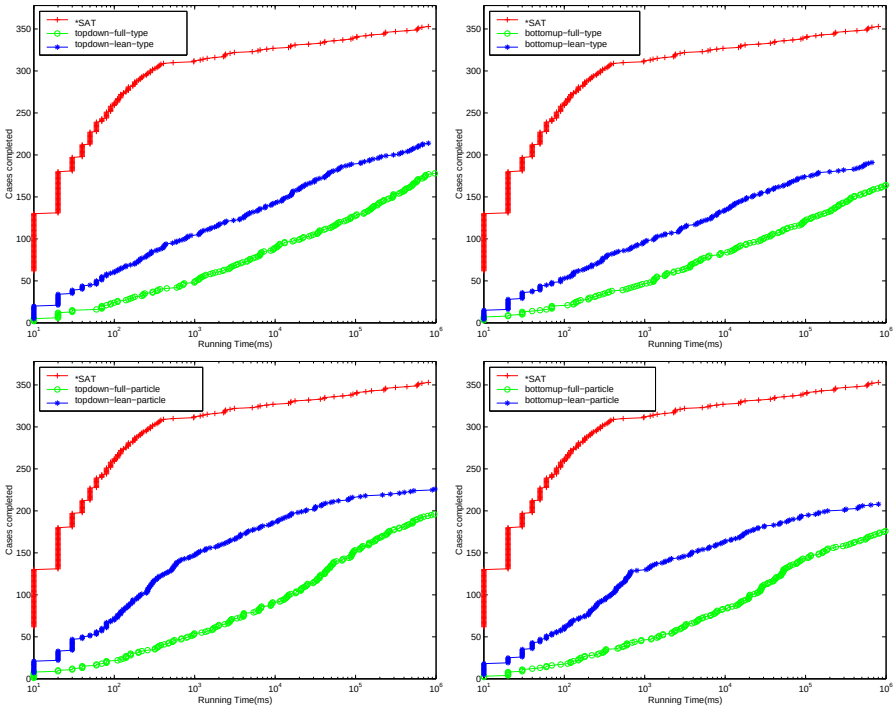

Fig. 4. Performance on TANCS 98 (lean vs. full)

Finally, we have compared the level-based approach with the top-down and the bottom-up approach. It turns out that the level-based approach outperforms both, and that, both for types and particles, the lean approach again outperforms the full one, see Fig. 5. The reason for this is that, by taking advantage of $\mathcal{K}$'s layered model property, we can split various space-consuming BDDs into smaller ones—depending on the modal depth of the corresponding subformulas. This minimizes space-outs and improves run time.

When compared with *SAT on the TANCS 98 benchmarks, all our approaches are still weaker than *SAT. Our implementation, however, is still open to a number of optimizations. In BDD-based symbolic model checking, it turned out that performance is extremely sensitive to the chosen order of BDD variables [7]. Moreover, there are different approaches to conjunctive partitioning [20]. So far, we did not optimize our implementation with respect to these aspects, and so we have yet to investigate the effect of problem-specific heuristics for variable ordering and conjunctive partitioning on the performance of our solver.

6 Conclusions

We have described various BDD-based decision procedures for $\mathcal{K}$. Our approach is inspired by the automata-theoretic approach, but we avoid explicit automata construction.

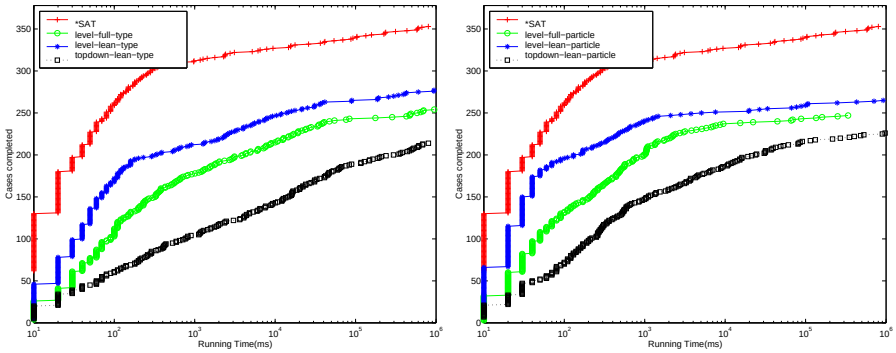

Fig. 5. Performance on TANCS 98 (level-based evaluation)

We explored a variety of optimization techniques and concluded that, in general, it is preferred to work with looser constraints; in general, we got the best performance with lean particles. We also showed that it is necessary to use a level-based approach to obtain a competitive implementation.

Our goal in this paper was not to develop the "fastest $\mathcal{K}$ solver", but rather to see whether the BDD-based approach is viable. From the competitiveness of our approach relative to *SAT on portions of the TANCS 2000 benchmark suite, we conclude that the BDD-based approach does deserve further study. In particular, we plan to study other optimizations strategies and also compare our approach to other modal satisfiability solvers.

References

1. H. R. Andersen. An introduction to binary decision diagrams. Technical report, Department of Information Technology, Technical University of Denmark, 1998.
2. C. Areces, R. Gennari, J. Heguiabehere, and M. de Rijke. Tree-based heuristics in modal theorem proving. In *Proceedings of the ECAI'2000*, 2000.
3. F. Baader and S. Tobies. The inverse method implements the automata approach for modal satisfiability. In *Proc. of IJCAR-01*, volume 2083 of *LNCS*. Springer Verlag, 2001.
4. I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In *Proc. of CAV-94*, volume 818 of *LNCS*, pages 182–193, 1994.
5. P. Blackburn, M. D. Rijke, Y. Venema, and M. D. Rijke. *Modal logic*. Cambridge University Press, 2001.
6. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, Vol. C-35(8):677–691, August 1986.
7. J. Burch, E. Clarke, K. McMillan, D. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Infomation and Computation*, 98(2):142–170, 1992.
8. J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. In *Int. Conf. on VLSI*, pages 49–58, 1991.
9. A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: A new symbolic model checker. *Int. Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.

10. R. Dyckhoff, editor. *Proceedings of TABLEAUX 2000*, volume 1847 of *LNAI*. Springer Verlag, 2000.
11. D. Geist and H. Beer. Efficient model checking by automated ordering of transition relation partitions. In *Proc. of the sixth Int. Conf. on CAV*, pages 299–310, 1994.
12. E. Giunchiglia, M. Maratea, A. Tacchella, and D. Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *IJCAR*, pages 347–363, 2001.
13. F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedure - the case study of modal K(m). *Infomation and Computation*, 162:158–178, 2000.
14. 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.
15. A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. Technical report, Universität Bern, Switzerland, 1996.
16. U. Hustadt and R. Schmidt. MSPASS: modal reasoning by translation and first order resolution. In Dyckhoff [10], pages 67–71.
17. G. Kamhi, L. Fix, and Z. Binyamini. Symbolic model checking visualization. In *Proc. of FMCAD'98*, volume 1522 of *LNCS*, pages 290–303. Springer Verlag, November 1998.
18. R. E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput.*, 6(3):467–480, 1977.
19. F. Massacci and F. M. Donini. Design and results of TANCS-00. In Dyckhoff [10], pages 52–56.
20. I.-H. Moon, G. D. Hachtel, and F. Somenzi. Border-block trianglular form and conjunction schedule in image computation. In W. H. Jr. and S. Johnson, editors, *FMCAD2000*, volume 1954 of *LNCS*, pages 73–90. Springer Verlag, 2000.
21. H. Ohlbach, A. Nonnengart, M. de Rijke, and D. Gabbay. Encoding two-valued non-classical logics in classical logic. In J. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier, 1999.
22. P. F. Patel-Schneider and I. Horrocks. DLP and FaCT. In *Proc. of TABLEAUX-99*, volume 1397 of *LNAI*, pages 19–23. Springer Verlag, 1999.
23. V. Pratt. A near-optimal method for reasoning about action. *Journal of Computer and System Sciences*, 20(2):231–254, 1980.
24. R. Ranjan, A. Aziz, R. Brayton, B. Plessier, and C. Pixley. Efficient BDD algorithms for FSM synthesis and verification. In *Proceedings of IEEE/ACM International Workshop on Logic Synthesis*, 1995.
25. F. Somenzi. CUDD: CU decision diagram package, 1998.
26. G. Sutcliffe and C. Suttner. Evaluating general purpose automated theorem proving systems. *Artificial intelligence*, 131:39–54, 2001.
27. A. Tacchella. *SAT system description. In *Collected Papers from the International Description Logics Workshop (DL'99). CEUR*, 1999.
28. J. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, 1983.
29. M. Vardi. What makes modal logic so robustly decidable? In N. Immerman and P. Kolaitis, editors, *Descriptive Complexity and Finite Models*, pages 149–183. American Mathematical Society, 1997.
30. A. Voronkov. How to optimize proof-search in modal logics: new methods of proving redundancy criteria for sequent calculi. *Computational Logic*, 2(2):182–215, 2001.