BDD-Based Decision Procedures for the Modal Logic K [1]

Guoqiang Pan* — **Ulrike Sattler**** — **Moshe Y. Vardi*****

* *Department of Computer Science, Rice University, Houston, Texas, 77005, USA.*
gqpan@cs.rice.edu

** *School of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK*
sattler@cs.man.ac.uk

*** *Department of Computer Science, Rice University, Houston, Texas, 77005, USA.*
vardi@cs.rice.edu

ABSTRACT. *We describe BDD-based decision procedures for the modal logic* **K**. *Our approach is inspired by the automata-theoretic approach, but we avoid explicit automata construction. Instead, we compute certain fixpoints of a set of types—which can be viewed as an on-the-fly emptiness of the automaton. We use BDDs to represent and manipulate such type sets, and investigate different kinds of representations as well as a "level-based" representation scheme. The latter turns out to speed up construction and reduce memory consumption considerably. We also study the effect of formula simplification on our decision procedures. To proof the viability of our approach, we compare our approach with a representative selection of other approaches, including a translation of* **K** *to QBF. Our results indicate that the BDD-based approach dominates for modally heavy formulae, while search-based approaches dominate for propositionally heavy formulae.*

KEYWORDS: *Modal Logic, Binary Decision Diagram*

---

1. Introduction

In the last 20 years, modal logic has been applied to numerous areas of computer science, including artificial intelligence [BRA 94, MCC 69], program verification [CLA 86, PRA 76, PNU 77], hardware verification [BOC 82, REI 83], database theory [CAS 82, LIP 77], and distributed computing [BUR 88, HAL 90]. In these applications, deciding satisfiability of a modal formula is one of the most basic rea-

---

[1] Portions of this paper have been presented at CADE-18 and CADE-19.

Journal of Applied Non-Classical Logics — June 28, 2005, Special Issue on Implementation of Logics 1th submission.

soning problem, and various techniques have been developed and optimized to decide it. Since satisfiability of even the smallest normal modal logic, $\mathbf{K}$, is PSPACE-complete [LAD 77, STO 77, HAL 92], it is clear that different techniques are useful for inputs of different characteristics, and that it is unlikely that one technique would be able to always outperform others. As modal logic extends propositional logic, the study in modal satisfiability is deeply connected with that of propositional satisfiability. For example, tableau-based decision procedures for $\mathbf{K}$ are presented in [LAD 77, HAL 92, PAT 99]. Such methods are 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-Logemann-Loveland method [DAV 62] as the propositional engine by treating all modal subformulae as propositions and, when a satisfying assignment is found, checking modal subformulae for the legality of this assignment [GIU 00, TAC 99]. In the last years, we have seen efforts to combine the optimizations used in tableau and DPLL based approaches. For example, using semantic branching and Boolean constraint propagation in a tableau-based solver made DLP and FaCT some of the fastest $\mathbf{K}$ solvers [PAT 99].

Non-propositional methods take a different approach to the problem. It has been shown recently that, by embedding $\mathbf{K}$ into first order logic, a first-order theorem prover can be used for deciding modal satisfiability [HUS 00, ARE 00]. 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 [HUS 00]. Other approaches for modal satisfiability such as mosaics, type elimination, or automata-theoretic approaches are well-suited for proving exact upper complexity bounds, but are rarely used in actual implementations [BLA 01, HAL 92, VAR 97].

In this paper, we restrict our attention to the smallest normal modal logic $\mathbf{K}$, and describe a novel approach to decide the satisfiability of formulae in this logic. The basic algorithms presented here are inspired by the automata-theoretic approach for logics with the tree-model property [VAR 97]. 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., whether it accepts some tree. In our approach, we combine, in essence, the two steps, and we carry out the non-emptiness test without explicitly constructing the automaton. As pointed out in [BAA 01], the inverse method described in [VOR 01] can also be viewed as an application of the automata-theoretic approach that avoids an explicit automata construction.

The logic $\mathbf{K}$ is simple enough that the automaton's non-emptiness test consists of a single fixpoint computation, which starts with a set of states and then repeatedly applies a monotone operator until a fixpoint is reached.[1] In the automaton that correspond to a formula, each state is a *type*, i.e., a set of formulae satisfying some consistency conditions. The algorithms presented here all start from some set of types,

---

[1] This approach can be easily extended to $\mathbf{K}$ (m).

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 **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 [VOR 01], while the top-down approach is reminiscent of the type-elimination method developed for propositional dynamic logic in [PRA 80].

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) [BRY 86]: BDDs are compact representations of propositional formulae, and are commonly used as a compact representation of states. One of their advantages is that they come with efficient operations for certain manipulations. By representing a set of types by a BDD, we are able to symbolically construct fixpoint type sets efficiently.

We then study optimization issues for BDD-based **K** solvers. First, we focus on alternative representations that can be used for a set of states. Types exert a strict consistency requirement on the assignment to related subformulae, which is a major factor in the size of the BDD used to represent the type sets. For example, if a type contains a conjunction, then it must also contain both conjuncts. For the type-based approach, we have employed the *box normal form*: negation can only be applied to atoms or box formulae (and no diamonds are available). In contrast, in the *particle*-based approach, we employ the standard negation normal form, and thus deal with both diamond and box formulae.

Secondly, for both the type- and the particle-approach, we investigate a *lean* approach: intuitively, our lean sets only capture the minimal, atomic information. E.g., conjunctions are only implicitly represented by the presence of both conjuncts. This clearly reduces BDD size, but also makes the manipulation of BDDs more complex.

Thirdly, we take advantage of the properties of **K**, namely the finite-tree-model property. A set of types/particles can be seen to encode a model for the formula. By considering a layered model instead of a general model, we modify the bottom-up procedure so that each step only checks witness for diamond operators occurring at a specific depth. This approach yields further performance improvements.

Fourthly, we turn to a pre-processing optimization. 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 [DAV 62]. Preprocessing has also been shown to be useful for linear-time formulae [SOM 00, ETE 00]. Our preprocessing is based on a modal pure-literal simplification which takes advantage of the tree-model property of **K**. We show that adding preprocessing yields a fairly significant performance improvements, enabling us to handle the hard formulae of TANCS 2000.

Finally, we also focus on BDD-specific optimizations on our implementation of the algorithm. Besides using optimized image finding techniques like conjunctive clustering with early quantification [BUR 91, GEI 94, RAN 95, CIM 00], we also study the issue of variable order, which is known to be of critical importance to BDD-based algorithms. The performance of BDD-based depends 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 [BRY 86]. While finding an optimal variable order is known to be intractable [TAN 93], heuristics often work quite well in practice [RUD 93]. We focus here on finding a good initial variable order tailored to the application at hand, but for large problem instances we have no choice but to invoke dynamic variable ordering, provided by the BDD package. Our finding is that choosing a good initial variable order does improve performance, but the improvement is rather modest.

This paper describes a viability study for our approach. To compare the different optimizations of BDD-based approaches, we use existing benchmarks of modal formulae, TANCS 98 [HEU 96] and TANCS 2000 [MAS 00], and we used *SAT [TAC 99] as a reference. A straightforward implementation of our approach did not yield a competitive implementation, but an optimized implementation did yield a competitive implementation, called $\mathcal{KBDD}$, indicating the viability of our approach.

To assess the competitiveness of our optimized solver, we compare it with the native solvers *SAT and DLP as well as the translation-based solver MSPASS. Additionally, we also developed a translation from $\mathbf{K}$ to QBF (which is of independent interest), and apply semprop, which is a highly optimized QBF solver [LET 02]. Our results indicate that the BDD-based approach dominates for modally heavy formulae while search-based approaches dominate for propositionally-heavy formulae.

The paper is organized as follows. After introducing the modal logic $\mathbf{K}$ in Section 2, we present our algorithms and show them to be sound and complete in Section 3. In Section 4, we discuss four optimizations that we applied. In Section 5, we present a BDD-based implementation. An embedding of $\mathbf{K}$ into QBF is presented in Section 6. Finally, we present the empirical evaluation, both between different optimizations in the BDD-based framework and with other solvers, in Section 7.

2. Preliminaries

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

The set of $\mathbf{K}$ formulae 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 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 **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 $L : V \rightarrow 2^W$ is 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 $K, w \models \varphi$) is inductively defined as follows:

- $K, w \models q$ for $q \in \Phi$ iff $w \in L(q)$
- $K, w \models \varphi \wedge \psi$ 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. Two formulae φ and ψ are said to the equivalent if, for all structures K and all worlds $w \in W$, $K, w \models \varphi$ if and only of $K, w \models \psi$.

For our concern here, the most important property of **K** is the *tree-model property*, which allows automata-theoretic approaches to be applied. In fact, it has the stronger *finite-tree-model property*, which will allow both top-down and bottom-up construction of such automata.

THEOREM 1 ([BLA 01]). — **K** *has the finite-tree-model property, i.e., every satisfiable formula* φ *has a model* K, w *such that* R *is a finite tree with root* w_0 *and* $K, w_0 \models \varphi$.

In fact, a formula ψ has a finite tree model that is only as deep as its *modal depth*, which we define next as usual; and we will use this “small tree model” property for the “level” optimizations in our algorithm.

Given a formula ψ, call its set of subformulae $\mathsf{sub}(\psi)$. For $\varphi \in \mathsf{sub}(\psi)$, we define $\mathsf{depth}(\varphi)$ as follows:

- if $\varphi \in \Phi$, then $\mathsf{depth}(\varphi) = 0$;
- if $\varphi = \neg\varphi'$, then $\mathsf{depth}(\varphi) = \mathsf{depth}(\varphi')$;
- If $\varphi = \varphi' \wedge \varphi''$ or $\varphi = \varphi' \vee \varphi''$, then $\mathsf{depth}(\varphi) = \max\{\mathsf{depth}(\varphi'), \mathsf{depth}(\varphi'')\}$,
- If $\varphi = \Box\varphi'$ or $\varphi = \Diamond\varphi'$, then $\mathsf{depth}(\varphi) = \mathsf{depth}(\varphi') + 1$.

We restrict our attention to formulae in a certain normal form. A formula ψ is said to be in *box normal form* (BNF) if all its subformulae are of the form $\varphi \wedge \varphi'$, $\varphi \vee \varphi'$, $\Box\varphi$, $\neg\Box\varphi$, q, or $\neg q$ where $q \in AP(\psi)$. Each **K** formulae can be obviously converted into an equivalent one in BNF that is of linear size. If not stated otherwise, we assume all formulae 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 $\{\varphi, \neg\varphi\} \subseteq \mathsf{cl}(\psi)$. Please note that $\mathsf{cl}(\psi)$ may contain negated conjunctions and negations, and thus formulae that are not in BNF.

The first algorithms we present work on *types*, i.e., maximal sets of formulae that are consistent w.r.t. the Boolean operators, and where (negated) box formulae are treated as atoms. A set of formulae $a \subseteq \mathsf{cl}(\psi)$ 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 of types A, we define a maximal accessibility 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 of types $A \subseteq 2^{\mathsf{cl}(\psi)}$, we can construct a Kripke structure K_A using Δ as follows: $K_A = \langle AP(\psi), A, \Delta, L\rangle$ with $a \in L(q)$ iff $q \in a$. Such a Kripke structure K_A is *almost* a canonical model [BLA 01]—the only difference can be seen when trying to prove that K_A satisfies, for all $\varphi \in \mathsf{cl}(\psi)$:

CLAIM 2. — $K_A, a \models \varphi$ iff $\varphi \in a$.

This statement is clearly true for atomic and propositional φ by definition of types, and it is also true for $\varphi = \Box\varphi'$ by construction of Δ. The only case that fails is the case $\varphi = \neg\Box\varphi' \in a$: it might be the case that $\varphi' \in b$ for all b with $\Delta(a, b)$. If this is the case, then we say that the negated box formula $\neg\Box\varphi'$ in a is *not witnessed* by any b in A. In the following section, we will describe operators on type sets whose fixpoint A then indeed satisfies Claim 2.

3. Our algorithms

The two algorithms presented here take a certain initial set of types and repeatedly apply a monotone operator to it. If this application reaches a fixpoint A, we can show that the above construction of K_A indeed a satisfies Claim 2, i.e., all negated box formulae 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 valid types, and the monotone operator removes those types containing negated box formulae 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 formulae, and then adds those types whose negated box formulae are witnessed in the current set of types.

In the following, we will call our class of algorithms $\mathcal{KBDD}$ since we intend to use BDD as the state set representation.

Both algorithms follow the following scheme:

```
$X \Leftarrow \texttt{Init}(\psi)$
repeat
   $X' \Leftarrow X$
   $X \Leftarrow \texttt{Update}(X')$
until $X = X'$
if exists $x \in X$ such that $\psi \in x$ then
   return "$\psi$ is satisfiable"
else
   return "$\psi$ is not satisfiable"
end if
```

If this algorithm is started with a finite set $\texttt{Init}(\psi)$ and uses a monotone $\texttt{Update}(\cdot)$ operator, it obviously terminates. In fact, after defining these two operators, we will show that it will terminate in $\mathsf{depth}(\psi) + 1$ iterations.

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 [HAL 92]. For the top-down algorithm, the functions $\texttt{Init}(\psi)$ and $\texttt{Update}(\cdot)$ are defined as follows:

– $\texttt{Init}(\psi)$ is the set of *all* ψ-types.

– $\texttt{Update}(A) := A \setminus \mathsf{bad}(A)$, where $\mathsf{bad}(A)$ are the types in A that contain unwitnessed negated box formulae. More precisely,

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

THEOREM 3. — *The top-down algorithm decides satisfiability of* **K** *formulae.*

PROOF. — Let A be the set of types that is the fixpoint of the top-down algorithm, i.e., $\texttt{Update}(A) = A$. We use A^0 for $\texttt{Init}(\psi)$ and A^i for the set of types after i iterations. Since $\texttt{Update}(\cdot)$ is monotone and each A^i is a subset of the finite $\mathsf{cl}(\psi)$, the top-down algorithm terminates. More precisely, A^{i+1} is obtained from A^i by removing types containing a formula φ with $\mathsf{depth}(\varphi) > i$. As a consequence, the algorithm stops after at most $\mathsf{depth}(\psi) + 1$ iterations. To finish the proof, it thus suffices to prove soundness and completeness.

LEMMA 4 (**SOUNDNESS**). — *For each type $a \in A$ and formula $\varphi \in \mathsf{cl}(\psi)$, if $\varphi \in a$, then $K_A, a \models \varphi$.*

PROOF. — By induction on the structure of formulae:

– if $\varphi \in AP(\psi)$, then $K_A, a \models \varphi$ iff $\varphi \in a$ by construction of L.

– if $\varphi = \neg q$, $\varphi = \varphi' \wedge \varphi''$, or $\varphi = \varphi' \vee \varphi''$, the claim follows immediately by induction and the definition of types.

– if $\varphi = \neg(\varphi' \wedge \varphi'')$, then $\varphi \in a$ implies that $\varphi' \notin a$ or $\varphi'' \notin a$ since, otherwise, $\varphi' \wedge \varphi''$ would be in a. By maximality of a, this implies that $\neg\varphi' \in a$ or $\neg\varphi'' \in a$, and thus we have $K_A, a \models \neg\varphi'$ or $K_A, a \models \neg\varphi''$ by induction. Hence $K_A, a \models \neg\varphi$.

– the case $\varphi = \neg(\varphi' \vee \varphi'')$ is completely analogous.

– let $\varphi = \Box\varphi' \in a$. The definition of Δ implies that $\varphi' \in a'$ for all a' with $\Delta(a, a')$. By induction, $K_A, a' \models \varphi'$, for all a' with $\varphi' \in a'$, and thus $K_A, a \models \Box\varphi'$.

– if $\varphi = \neg\Box\varphi' \in a$, then $a \notin \mathsf{bad}(A)$ because $\mathtt{Update}(A) = A$, and thus there exists $b \in A$ with $\Delta(a, b)$ and $\varphi' \notin b$. By definition of types, $\neg\varphi' \in b$, and thus we have $K_A, b \models \neg\varphi'$ by induction. Hence $K_A, a \models \neg\Box\varphi'$.

■

LEMMA 5 (**COMPLETENESS**). — *For all φ in $\mathsf{cl}(\psi)$, if φ is satisfiable, then there exists some $a \in A$ with $\varphi \in a$.*

PROOF. — Given a satisfiable formula φ, take a model $K = \langle AP(\psi), W, R, L\rangle$ with $K, w_\varphi \models \varphi$. For a world $w \in W$, we define its type $a(w) = \{\varrho \in \mathsf{cl}(\psi) \mid K, w \models \varrho\}$, and we define $A(W) = \{a(w) \mid w \in W\}$. Obviously, due to the semantics of the box modality, $R(v, w)$ implies $\Delta(a(v), a(w))$. Then we show, by induction on i, that $A(W) \subseteq A^i$. Since $\varphi \in a(w_\varphi)$ by construction, this proves the lemma.

– $A(W) \subseteq A^0$ since A^0 contains *all* types $a \subseteq \mathsf{cl}(\psi)$.

– Let $A(W) \subseteq A^i$ and assume that $A(W) \not\subseteq A^{i+1}$. Then there is some $w \in K$ such that $a(w) \in \mathsf{bad}(A^i)$. So there is some $\neg\Box\varrho \in a(w)$ and, for all $b \in A^i$ with $\Delta(a(w), b)$, we have $\varrho \in b$. Hence there is no $v \in W$ with $R(w, v)$ and $K, v \models \neg\varrho$, in contradiction to $K, w \models \neg\Box\varrho$.

■

■

3.2. *Bottom-up approach*

As mentioned above, the top-down algorithm starts with all valid types, and repeatedly removes types with unwitnessed formulae. In contrast, the bottom-up algorithm starts with a small set of types (i.e., those without negated box formulae), and repeatedly adds those types whose negated box formulae are witnessed in the current set. For the bottom-up approach, the functions $\mathtt{Init}(\psi)$ and $\mathtt{Update}(\cdot)$ are defined as follows:

– $\mathtt{Init}(\psi)$ is the set of all those types that do not require any witnesses, which means that they do not contain any negated box formula or, equivalently, that they contain all positive box formulae in $\mathsf{cl}(\psi)$. More precisely,

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

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

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

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

THEOREM 6. — *The bottom-up algorithm decides satisfiability of* **K** *formulae.*

PROOF. — As in the proof of Theorem 3, we use A for the fixpoint of the bottom-up algorithm, A^0 for $\texttt{Init}(\psi)$, and A^i for the set of types after i iterations. Again, $\texttt{Update}(\cdot)$ is monotone and A^0 is finite, and thus the bottom-up algorithm terminates. More precisely, A^{i+1} is obtained from A^i by adding types containing a formula φ with $\mathsf{depth}(\varphi) > i$. As a consequence, the algorithm stops after at most $\mathsf{depth}(\psi) + 1$ iterations. To finish the proof, we prove soundness and completeness.

LEMMA 7 (**SOUNDNESS**). — *For each type $a \in A$ and formula $\varphi \in \mathsf{cl}(\psi)$, if $\varphi \in a$, then $K_A, a \models \varphi$.*

PROOF. — Again, soundness can be proved by induction on the structure of formulae. We restrict our attention to the only interesting case, namely $\varphi = \neg\Box\varphi' \in a$. Let $\varphi \in a$. By construction of A, there is some $b \in A$ with $\neg\varphi' \in b$ and $\Delta(a,b)$. Thus, by induction, $K_A, b \models \neg\varphi'$, and thus $K_A, a \models \varphi$. .

■

LEMMA 8 (**COMPLETENESS**). — *For all $\varphi \in \mathsf{cl}(\psi)$, if φ is satisfiable, then there exists some $a \in A$ with $\varphi \in a$.*

PROOF. — It is well-known that **K** has the finite-tree-model property (see, e.g. [HAL 92]), i.e., each satisfiable **K** formula ψ has a model whose relational structure forms a finite tree. Take such a model $K = \langle AP(\psi), W, R, L\rangle$ with $K, w_\varphi \models \varphi$, and define the mappings $a(\cdot)$ and $A(\cdot)$ from worlds in K to types as in the proof of Lemma 5. We show by induction on i that, if i is the maximal distance between a node $w \in W$ and the leaves of K's subtree rooted at w, then $a(w) \in A^i$. Since $A^j \subseteq A^{j+1}$ for all j and K forms a finite tree model of φ, this proves the lemma.

– If $i = 0$, then w is a leaf in K (i.e., there is no $w' \in W$ with $R(w, w')$), and thus $K, w \not\models \neg\Box\varphi'$ holds for all $\neg\Box\varphi' \in \mathsf{cl}(\psi)$. Hence $a(w) \in A^0$.

– Let $i > 0$ and w a node with i the maximal distance between w and the leaves of K's subtree rooted at w. Then, by induction, for each child w' of w, we have $a(w') \in A^{i-1}$. Now $R(w, w')$ implies $\Delta(a(v), a(w))$. Thus, for each $\neg\Box\varphi' \in a(w)$, there is some $w' \in W$ with $a(w') \in A^{i-1}$ and $\neg\varphi' \in a(w')$. Thus $a(w) \in \mathsf{supp}(A^{i-1}) \subseteq A^i$.

■

■

4. Optimizations

The decision procedures described above handles a formula in three steps. First, the formula is converted into box normal form. Then, the initial set of types is generated—we can think of this set as being represented by a set of bit vectors. Finally, this set is updated through a fixpoint process. The answer of the decision procedure depends on a simple syntactic check of this fixpoint. In this section, we will describe four orthogonal optimization techniques, working on different stages in the procedure.

4.1. *Particles*

The approaches presented so far strongly depend on the fact that we use the box normal form, and they can be said to be redundant: if a type contains two conjuncts of some subformula of the input, then it also contains the corresponding conjunction—although the truth value of the latter is determined by the truth values of the former. Now we propose a representation where we do not insist on such a redundancy, which possibly reduces the size of the representation of the corresponding sets. To do so, it is convenient to work on formulae in a different normal form.

A **K** formula ψ is said to be in *negation normal form* (NNF) if all its subformulae are of the form $\varphi \wedge \varphi'$, $\varphi \vee \varphi'$, $\Box\varphi$, $\Diamond\varphi$, q, or $\neg q$ where $q \in AP(\psi)$. It is well-known that every **K** formula can be converted into an equivalent on in NNF that is of linear size. When talking about "particles", we assume that all formulae are in NNF. As before, we use $\mathsf{sub}(\psi)$ to denote the set of subformulae of ψ.

A set $p \subseteq \mathsf{sub}(\psi)$ is a 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 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 $\mathtt{Init}(\cdot)$ and $\mathtt{Update}(\cdot)$ for particles are defined as follows:

– $\mathtt{Init}(\psi)$ is the set of all ψ-particles.

– $\mathtt{Update}(P) = P \setminus \mathsf{bad}(P)$, where $\mathsf{bad}(P)$ is the particles in P that contain unwitnessed diamond formulae, i.e.

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

THEOREM 9. — *The top-down algorithm for particles decides satisfiability of* **K** *formulae.*

PROOF. — Termination and the linear bound on the number of iterations are identical to the one of Theorem 3.

LEMMA 10. — ***(Soundness)*** *For each particle $p \in P$ and formula $\varphi \in \mathsf{sub}(\psi)$, if $\varphi \in p$, then $K_P, p \models \varphi$.*

PROOF. — The proof is analogous to one of Lemma 4, except for the fact that the $\neg\Box\varphi'$ case needs to be replaced with the $\Diamond\varphi'$ one.

– if $\varphi = \Diamond\varphi' \in p$, then $p \notin \mathsf{bad}(P)$ implies that there exists $q \in P$ with $\Delta(p,q)$ and $\varphi' \in q$. By induction, $K_P, q \models \varphi'$, and thus $K_P, p \models \Diamond\varphi'$.

■

LEMMA 11. — ***(Completeness)*** *For all $\varphi \in \mathsf{sub}(\psi)$, if φ is satisfiable, then there exists some $p \in P$ with $\varphi \in p$.*

PROOF. — The proof is analogous to the one of Lemma 5: we take a model K of φ, generate a particle set $P(W)$ from the states of K, and show that $P(W) \subseteq P$ by induction on the number of iterations i:

– $P(W) \subset P^0$ since P^0 contains *all* particles $p \subseteq \mathsf{sub}(\psi)$.

– Let $P(W) \subseteq P^i$ and assume that $P(W) \nsubseteq P^{i+1}$. Then there is some $w \in K$ such that $p(w) \in \mathsf{bad}(P^i)$. So there is some $\Diamond\varphi \in p(w)$ and, for all $q \in A^i$ with $\Delta(p(w), q)$, we have $\varphi \notin q$. Hence there is no $v \in W$ with $R(w,v)$ and $K, v \models \neg\varphi$, in contradiction to $K, w \models \Diamond\varphi$.

■

■

As for types, we also define a bottom-up algorithm for particles, and we do this by simply setting our two auxiliary functions accordingly:

– $\mathtt{Init}(\psi) := \{p \subseteq \mathsf{sub}(\psi) \mid p$ is a particle and $\Diamond\varphi \notin p$ for all $\Diamond\varphi \in \mathsf{sub}(\psi)\}$ is the set of ψ-particles p that do not contain diamond formulae.

– $\mathtt{Update}(P) := P \cup \mathsf{supp}(P)$, where $\mathsf{supp}(P)$ is the set of witnessed particles defined as follows:

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

Again, we obtain a decision procedure, and this can be proved as before.

THEOREM 12. — *The bottom-up algorithm for particles decides satisfiability of* **K** *formulae.*

As mentioned before, we can represent a particle or a type as a bit vector, and we can encode a set of bit vectors in a BDD. It is easy to see that bit vectors for particles may be longer than bit vectors for types because, for example, our input may involve subformulae $\Box p$ and $\Diamond\neg p$. This, in turn, means that encoding particle sets using BDDs may require more BDD variables than their encoding of types. The size of the BDD may, however, be smaller for particles since particles impose fewer constraints than types.[2] Beside a possible reduction in the size required to encode a bit-vector representation of particle sets, the particle-based approaches also can improve run time of our algorithms. From the definition of bad and supp, we can see that, in the type-based approaches, for each fixpoint iteration and each type, we have to check all box formulae—even though the "real" test is only required on the negated ones that are present in the type considered. In contrast, in the particle-based approaches, we only have to check all diamond formulae that are subformulae of the input.

4.2. *Lean approaches*

Even though the particle approach imposes less constraints than type approach, it still involves redundant information: like types, particles may contain both a conjunction and the corresponding conjuncts. Next, to further reduce the size of the corresponding BDDs, we propose a representation where we only keep track of the "non-redundant" subformulae. We call this variation the lean approach, and we present it for both the type and the particle approach and, for both top-down and bottom-up.

First, we define a set of "non-redundant" subformulae $\mathsf{atom}(\psi)$ as the set of those formulae in $\mathsf{cl}(\psi)$ that are neither conjunctions nor disjunctions, i.e., each $\varphi \in \mathsf{atom}(\psi)$ is of the form $\Box\varphi'$, q, $\neg\Box\varphi'$, or $\neg q$. By definition of types, each ψ-type $a \subseteq \mathsf{cl}(\psi)$, corresponds one-to-one to a *lean type* $\mathsf{lean}(a) := a \cap \mathsf{atom}(\psi)$. To specify our algorithms for lean types, we define inductively a relation $\dot{\in}$ between (non-atomic) formulae and lean types as follows: $\varphi \mathrel{\dot{\in}} a$ if

- $\varphi \in \mathsf{atom}(\psi)$ and $\varphi \in a$,
- $\varphi = \neg\varphi'$ and not $\varphi' \mathrel{\dot{\in}} a$,
- $\varphi = \varphi' \wedge \varphi''$, $\varphi' \mathrel{\dot{\in}} a$, and $\varphi'' \mathrel{\dot{\in}} a$, or
- $\varphi = \varphi' \vee \varphi''$ and $\varphi' \mathrel{\dot{\in}} a$ or $\varphi'' \mathrel{\dot{\in}} a$.

The top-down and bottom-up approach for types can be easily modified to work for lean types: it suffices to modify the definition of the functions bad and supp as follows:

$$\begin{aligned}
\mathsf{bad}(A) &:= \{a \in A \mid \text{there exists } \neg\Box\varphi \in a \text{ and, for all } b \in A \text{ with } \Delta(a,b), \\
&\qquad\qquad \text{we have } \varphi \mathrel{\dot{\in}} b\}. \\
\mathsf{supp}(A) &:= \{a \subseteq \mathsf{cl}(\psi) \mid a \text{ is a type and, for all } \neg\Box\varphi \in a, \text{ there exists } b \in A \\
&\qquad\qquad \text{with } \neg\varphi \mathrel{\dot{\in}} b \text{ and } \Delta(a,b)\}.
\end{aligned}$$

---

[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.

The following theorem is then a direct consequence of the correctness of our algorithms for types: given the one-to-one relationship between types and lean types, we can easily see that, for a type a, its lean version $a' = a \cap \mathsf{atom}(\psi)$, and all φ, we have $\varphi \mathrel{\dot{\in}} a'$ iff $\varphi \in a$.

THEOREM 13. — *The top-down and the bottom-up algorithm for lean types decide satisfiability for* **K**.

Analogously, we can define a lean representation for particles. First, we define the relevant subformulae $\mathsf{part}(\psi)$ 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 particle $p \subseteq \mathsf{sub}(\psi)$, we define the corresponding *lean particle* $\mathsf{lean}(p)$ as follows: $\mathsf{lean}(p) = p \cap \mathsf{part}(\psi)$. Because the constraints on particles are more relaxed than those of types, more than one particle may lead to the same lean particle. Secondly, we define the relation $\tilde{\in}$ between formulae and particles as follows: $\varphi \mathrel{\tilde{\in}} a$ if

- $\varphi \in \mathsf{part}(\psi)$ and $\varphi \in a$,
- $\varphi = \varphi' \wedge \varphi''$, $\varphi' \mathrel{\tilde{\in}} a$, and $\varphi'' \mathrel{\tilde{\in}} a$, or
- $\varphi = \varphi' \vee \varphi''$ and $\varphi' \mathrel{\tilde{\in}} a$ or $\varphi'' \mathrel{\tilde{\in}} a$.

Thirdly, we define the relations supp and bad for lean particles as follows:

$$\begin{array}{lll} \mathsf{bad}(P) & := & \{p \in P \mid \text{ there exists } \Diamond\varphi \in p \text{ such that, for all } q \in P \\ & & \qquad \text{with } \Delta(p,q), \text{ we have not } \varphi \mathrel{\tilde{\in}} q\}. \\ \mathsf{supp}(P) & := & \{p \subseteq \mathsf{sub}(\psi) \mid p \text{ is a } \psi\text{-particle and, for all } \Diamond\varphi \in p, \\ & & \qquad \text{there exists } q \in P \text{ with } \varphi \mathrel{\tilde{\in}} q \text{ and } \Delta(p,q)\}. \end{array}$$

Again, correctness of the lean approach for particles follows from the correctness of the particle algorithms.

THEOREM 14. — *The top-down and the bottom-up algorithm for lean particles decide satisfiability for* **K**.

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*

In this last variation of our basic algorithms, we exploit the fact that **K** enjoys the finite-tree-model property, i.e., each satisfiable formula ψ of **K** has a finite tree model of depth bounded by the depth of nested modal operators $\mathsf{depth}(\psi)$ of ψ. We can think of such a model as being partitioned into *layers*, where all worlds that are at distance i from the root are said to be in layer i. Instead of representing a complete model using a set of particles or types, we represent each layer in the model using a separate set. For a level-based approach in the context of the first-order approach to **K**, see [ARE 00]. Since only a subset of all subformulae appears in one layer, the representation can be more compact. We only present this optimization for the approach using (full) types—the particle approach and the lean approach can be constructed analogously. For $0 \leq i \leq \mathsf{depth}(\psi)$, 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 maximal 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

$$K_A = \langle AP(\psi), A_0 \uplus \ldots \uplus A_d, R, L\rangle$$

as follows, where $\uplus$ denotes the disjoint union:

– For a world $a \in A_i$ and $q \in AP(\psi)$, we define $a \in L(q)$ iff $q \in a$.

– 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 $\Delta_i(a, a')$.

We define a bottom-up algorithm for level-based evaluation as follows:

```
$d \Leftarrow \mathsf{depth}(\psi)$
$X_d \Leftarrow \mathtt{Init}_d(\psi)$
**for** $i = d - 1$ downto 0 **do**
    $X_i \Leftarrow \mathtt{Update}(X_{i+1}, i)$
**end for**
**if** exists $x \in X_0$ where $\psi \in x$ **then**
    $\psi$ is satisfiable.
**else**
    $\psi$ is not satisfiable.
**end if**
```

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 moves up the tree model toward the root, adding nodes that are "witnessed". In contrast, the bottom-up approach presented earlier starts with *all* leaves of a tree model.

For the level-based algorithm and types as data structure, the auxiliary functions can be defined as follows:

– $\mathtt{Init}_i(\psi) = \{a \subseteq \mathsf{cl}_i(\psi) \mid a \text{ is a type}\}$.

– $\mathtt{Update}(A, i) = \{a \in \mathtt{Init}_i(\psi) \mid \text{for all } \neg\Box\varphi \in a \text{ there exists } b \in A \text{ with } \neg\varphi \in b \text{ and } \Delta_i(a, b)\}$.

For a set A of types of formulae at level $i + 1$, $\mathtt{Update}(A, i)$ represents all types of formulae at level i that are witnessed in A.

THEOREM 15. — *The level-based algorithm for types is sound and complete.*

PROOF. — We write the sequence of assignment sets constructed by the level based algorithm as $A = \langle A_0, A_1, \ldots, A_d \rangle$ where $d = \mathsf{depth}(\psi)$. Termination after d steps is trivial.

LEMMA 16. — ***(Soundness)*** *For all $\varphi \in \mathsf{cl}_i(\psi)$, and $a \in A_i$, if $\varphi \in a$, then $K_A, a \models \varphi$.*

Soundness can be proved as for the bottom-up approach, with the additional observation that R only relates worlds in A_i with worlds in A_{i+1}.

LEMMA 17. — ***(Completeness)*** *For $\varphi \in \mathsf{cl}_i(\psi)$, if φ is satisfiable, then there is a type $a \in A_i$ with $\varphi \in a$.*

PROOF. — Let $\varphi \in \mathsf{cl}_i(\psi)$ be satisfiable. We know from [HAL 92] that φ has a finite tree model $K_\varphi = \langle AP(\psi), W, R, L \rangle$ of depth $d_\varphi = \mathsf{depth}(\varphi)$ such that $K_\varphi, w_0 \models \varphi$ for the root w_0 of K_φ. We also know from the definition of depth and cl_i that $i + d_\varphi \leq d_\psi = \mathsf{depth}(\psi)$, and thus we have that $d_\varphi \leq d_\psi - i$. Since K_φ is a tree model, we can partition its set of worlds W into $\{W_0, W_1, \ldots, W_{d_\varphi}\}$ such that each $w \in W_j$ occurs at distance j from the root. Similar to our completeness proofs before, from a world $w \in W_j$, we define a type $a(w)$ as follows: $a(w) = \{\varrho \in \mathsf{cl}\, i + j\psi) \mid K_\varphi, w \models \varrho\}$. We define $A(W_j) = \{a(w) \mid w \in W_j\}$ and now show that $A(W_j) \subseteq A_{i+j}$ by induction on depth j:

– if $j = d_\varphi$, then, for each world $w \in W_{d_\varphi}$, there is no world w' that is R-accessible from w. It follows that, for all $\varrho = \neg\Box\varrho'$, we have $K_\varphi, w \not\models \varrho$, and thus $\varrho \notin a(w)$. Since $a(w)$ is a type, we thus have $a(w) \in A_{i+j}$.

– let $j < d$ and let $\varrho = \neg\Box\varrho' \in a(w)$. Hence there exists some $w' \in W_{j+1}$ with $\neg\varrho' \in a(w')$. By induction, $a(w') \in A_{i+j+1}$. Since this is true for each negated box formula in $a(w)$, we have that $a(w) \in A_{i+j}$ by definition of `Update`.

■

■

Analogously, a level-based algorithm can be defined for particles: let $\mathsf{sub}_i(\psi)$ denote the set of ψ's subformulae occurring at depth i in ψ, and define the auxiliary functions as follows:

– $\mathtt{Init}_i(\psi) = \{p \subseteq \mathsf{sub}_i(\psi) \mid p \text{ is a particle}\}$.

– $\mathtt{Update}(P, i) = \{p \in \mathtt{Init}_i(\psi) \mid \text{for all } \Diamond\varphi \in p \text{ there exists } q \in P \text{ with } \varphi \in q \text{ and } \Delta_i(p, q)\}$.

The following theorem can be proved like the one for the type approach.

THEOREM 18. — *The level-based algorithm for particle assignments is sound and complete.*

4.4. *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 [DAV 62], 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 formulae [SOM 00, ETE 00] and for description logic reasoner [HOR 00, HAA 01]. Our preprocessing is based on a modal pure-literal simplification, which takes advantage of the layered-model property of $\mathbf{K}$.

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

– *Equivalence preserving transformations*, when applied to some φ, yield a formula φ' which is logically equivalent to φ. Unit propagation is an example of an equivalence preserving transformation which is used in model checking [SOM 00, ETE 00], where the semantics of the formula needs to be preserved. Clearly, applying an equivalence preserving transformation to a subformula yields an equivalent formula, and thus these transformations can be applied to subformulae.

– *Satisfiability preserving transformations*, when applied to some φ, yield a formula φ' which is satisfiable if and only if φ is satisfiable. Pure-literal simplification [DAV 60] is an example of a satisfiability-preserving transformation. Such transformations allow for more aggressive simplifications, but cannot be applied to subformulae, and they cannot be used 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 the following aspects:

1) The primary goal is to minimize the size of the formula. 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.

2) We also aim at minimizing the number of modal operators in the formula. This leads to a smaller transition relation, where we have a constraint for each $\Box$ subformula, as well as a smaller number of BDD operations involved in witnessing $\Diamond$ subformulae.

We found that our preprocessing was beneficial for DLP, a tableau-based modal solver, as well as *SAT, a DPLL-based solver, but not for MSPASS, a resolution-based solver.

4.4.1. *Rewrite rules*

Our preprocessing includes rewriting according to the rewrite rules given in Table 1. It is easy to see that the rules are equivalence or satisfiability preserving. These rules by themselves are only modestly effective for $\mathbf{K}$ formulae; 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.

Table 1. *Simplification rewriting rules for* **K**

Propositional rules		
Equivalence	$f \wedge \mathsf{true} \leadsto f$	$f \wedge \mathsf{false} \leadsto \mathsf{false}$
	$f \vee \mathsf{true} \leadsto \mathsf{true}$	$f \vee \mathsf{false} \leadsto f$
	$f \wedge f \leadsto f$	$f \vee f \leadsto f$
	$f \wedge \neg f \leadsto \mathsf{false}$	$f \vee \neg f \leadsto \mathsf{true}$
Modal rules		
Equivalence	$\Diamond\, \mathsf{false} \leadsto \mathsf{false}$	$\Box\, \mathsf{true} \leadsto \mathsf{true}$
	$\Diamond f \vee \Diamond g \leadsto \Diamond(f \vee g)$	$\Box f \wedge \Box g \leadsto \Box(f \wedge g)$
Satisfiability preserving	$\Diamond f \wedge \Box g \wedge h \leadsto \Diamond(f \wedge g) \wedge h$ where $\mathsf{depth}(h) = 0$.	$\Diamond f \leadsto f$

4.4.2. *Pure-literal simplification*

To apply pure-literal simplification to **K** satisfiability solving, we first need to extend it to the modal setting.

DEFINITION 19. — *Given a set S of (propositional or modal) formulae in NNF, we define* $\mathsf{lit}(S) = \{\ell \in S \mid \ell = q$ *or* $\ell = \neg q,$ *for some* $q \in \Phi\}$ *as the set of literals of* S*. The set* $\mathsf{pure}(S)$ *is defined as the set of literals that have a pure-polarity occurrence in* S*, i.e.,* $\mathsf{pure}(S) := \{\ell \in \mathsf{lit}(S) \mid \dot{\neg}\ell \notin \mathsf{lit}(S)\}$ *for* $\dot{\neg}\ell$ *the negation normal form of* $\neg\ell$*.*

It is well known that pure-literal simplification preserves propositional satisfiability; that is, given a propositional formula φ, for any literal $\ell \in \mathsf{pure}(\varphi)$, φ is satisfiable iff $\varphi[\ell/\,\mathsf{true}]$ is satisfiable. There are a number of ways to extend the definition of pure literals to modal logics. We first present a naive definition, and then explain how to extend it.

DEFINITION 20. — *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*. Applying pure literal simplification to* ψ *yields a formula* ψ'_G *that is obtained from* ψ *by replacing each occurrence of each literal in* $\mathsf{pure}(\psi)$ *with* true*.*

Pure literal simplification can be made more efficient because **K**'s tree model property implies that assignments to literals at different modal depths are in different worlds, and thus independent of each other. Hence we define the following, stronger version of pure literals simplification.

DEFINITION 21. — *First, we define the level* $\mathsf{level}(\psi, \varphi)$ *of the occurrence of a subformula* φ *in a formula* ϕ *as follows:*[3]

– *If* $\psi = \varphi$*, then* $\mathsf{level}(\psi, \varphi) = 0$*;*

– *If* $\varphi = \varphi' \wedge \varphi''$*,* $\varphi' \vee \varphi''$*, or* $\neg\varphi'$*, then* $\mathsf{level}(\psi, \varphi') = \mathsf{level}(\psi, \varphi'') = \mathsf{level}(\psi, \varphi)$*;*

– *If* $\varphi = \Box\varphi'$ *or* $\Diamond\varphi'$*, then* $\mathsf{level}(\psi, \varphi') = \mathsf{level}(\psi, \varphi) + 1$.

For ψ *in NNF, we define* level-pure literals *by* $\mathsf{pure}_i(\psi) = \mathsf{pure}(\mathsf{sub}_i(\psi))$*, for* $0 \le i \le \mathsf{depth}(\psi)$*, and we define* $\psi[\mathsf{pure}_i(\psi)/\,\mathsf{true}]_i$ *to be the result of substituting each occurrence at level* i *of a literal in* $\mathsf{pure}_i(\psi)$ *with* true*. Applying level-wise pure literal simplification to* ψ *yields a formula* $\psi'_L = \psi[\mathsf{pure}_0(\psi)/\,\mathsf{true}]_0 \ldots [\mathsf{pure}_{\mathsf{depth}(\psi)}(\psi)/\,\mathsf{true}]_{\mathsf{depth}(\psi)}$.

It is possible to push this idea of "separation" further. Because each world in the model may satisfy a different subset of formula, if a literal occurs both positively and negatively at level i inside a diamond subformula, then we still might replace it with true whilst preserving satisfiability. However, checking whether a subformula may be substituted with true then involves such a huge overhead that we do not believe that it justifies its implementation.

We now prove that pure-literal simplification preserves satisfiability.

THEOREM 22. — *Let* ψ *be in NNF. Then* ψ *is satisfiable iff* ψ'_G *is satisfiable iff* ψ'_L *is satisfiable.*

PROOF. — We write ψ' instead of ψ'_G or ψ'_L, when the formula used is clear from the context. First, we show that substituting a single pure literal ℓ preserves satisfiability. Theorem 22 follows then by induction on the number of pure literals.

The only-if direction is due to the fact that the $\Box$ and $\Diamond$ operators are *monotone* [BLA 01]. More precisely, let ψ be a formula in NNF, α a subformula occurrence of ψ, and β a formula that is logically implied by α, then $\psi[\alpha/\beta]$ is logically implied by ψ. Since every ℓ implies true, satisfiability of ψ implies satisfiability of ψ'.

For the if direction, let $K' = \langle \Phi, W, R, L' \rangle$ be a finite tree Kripke structures of depth $\mathsf{depth}(\psi)$ with $w_0 \in W$ the root of the tree and $K', w_0 \models \psi'$.

– *Globally pure literals,* i.e., $\psi' = \psi'_G$. Since ℓ does not occur in ψ'_G, we can assume that L' does not define a truth value for ℓ. We construct a model $K = (W, R, L)$ from K' by taking L to be the following extension of L': if $\ell \in AP$, then $L(\ell) = W$, otherwise $\ell = \neg q$ for some $q \in AP$ and we set $L(\ell) = \emptyset$. We claim that, for every world $w \in W$ and every formula $\varphi \in \mathsf{sub}(\psi)$, $K', w \models \varphi[\ell/\,\mathsf{true}]$ implies $K, w \models \varphi$.

This claim is an immediate consequence of the fact that, for all $w \in W$, $K, w \models \ell$.

– *Level-pure literals*: Assume $K', w_0 \models \psi'$, and consider the occurrence of ℓ in ψ at level i. For $0 \le i \le \mathsf{depth}(\psi)$, let $W_i = \{w \mid \text{distance between } w \text{ and } w_0 = i\}$. We construct K from K' by defining L as follows: (1) $L(q) = L'(q)$ for each $q \in$

---

[3] Please note that a subfomula can occur at more than one level in a formula.

$\Phi \setminus AP(l)$. (2) $L(q) \cap W_j = L'(q) \cap W_j$ for each $j \neq i$, (3) if $\ell \in AP$, then set $L(\ell) \cap W_i = W_i$, otherwise set $L(\ell) \cap W_i = \emptyset$.

For each $\varphi \in \mathsf{sub}_i(\psi)$ and $w \in W_i$, we have that $K', w \models \varphi[\ell/\,\mathsf{true}]_{d-i}$ implies $K, w \models \varphi$. This is an immediate consequence of the fact that, for all $w \in W_i$, $K, w \models \ell$. Since K and K' coincide on the interpretation of all propositional variables in worlds in $W \setminus W_i$, and on the interpretation of all propositional variables different from $AP(\ell)$ in all worlds, it follows that $K, w_0 \models \psi[\ell/\,\mathsf{true}]_d$.

■

5. Implementation

In this section, we describe how to implement our algorithms and their variations using Binary Decision Diagrams (BDDs).

5.1. *Base algorithms*

We use Binary Decision Diagrams (BDDs) [BRY 86, AND 98] 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 representation for Boolean functions. Experience has shown that BDDs often provide a very compact representation for very large Boolean functions, and that various operations on Boolean functions can be carried out efficiently on their BDD representation. Consequently, over the last decade, BDDs have had a dramatic impact in the areas of synthesis, testing, and verification of digital systems [BEE 94, BUR 92].

In this section, we describe how our two basic algorithms, top-down and bottom up with types, 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: first, we split $\mathsf{cl}(\psi)$ into positive and negative formulae, i.e.,

$$\begin{array}{lcl} \mathsf{cl}_+(\psi) & := & \{\varphi_i \in \mathsf{cl}(\psi) \mid \varphi_i \text{ is not of the form } \neg\varphi'\} \text{ and} \\ \mathsf{cl}_-(\psi) & := & \{\neg\varphi \mid \varphi \in \mathsf{cl}_+(\psi)\}, \end{array}$$

and we use m for $|\,\mathsf{cl}_+(\psi)| = |\,\mathsf{cl}(\psi)|/2$. Then, for $\mathsf{cl}_+(\psi) = \{\varphi_1, \ldots, \varphi_m\}$, a vector $\vec{a} = \langle a_1, \ldots, a_m \rangle \in \{0,1\}^m$ represents the set[4] $a \subseteq \mathsf{cl}(\psi)$ with $\varphi_i \in a$ iff $a_i = 1$.

---

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

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 $\mathsf{consistent}_\psi$ as the characteristic predicate for types: $\mathsf{consistent}_\psi(\vec{a}) = \bigwedge_{1 \leq i \leq m} \mathsf{consistent}^i_\psi(\vec{a})$, where $\mathsf{consistent}^i_\psi(\vec{a})$ is defined as follows:

– if φ_i is neither of the form $\varphi' \wedge \varphi''$ nor $\varphi' \vee \varphi''$, then $\mathsf{consistent}^i_\psi(\vec{a}) = 1$,

– if $\varphi_i = \varphi' \wedge \varphi''$, then $\mathsf{consistent}^i_\psi(\vec{a}) = (a_i \wedge a' \wedge a'') \vee (\neg a_i \wedge (\neg a' \vee \neg a''))$,

– if $\varphi_i = \varphi' \vee \varphi''$, then $\mathsf{consistent}^i_\psi(\vec{a}) = (a_i \wedge (a' \vee a'') \vee (\neg a_i \wedge \neg a' \wedge \neg a''))$,

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 $a'' = a_k$ if $\varphi'' = \varphi_k \in \mathsf{cl}_+(\psi)$, and $a'' = \neg a_k$ if $\varphi'' = \neg\varphi_k$ for $\varphi_k \in \mathsf{cl}_+(\psi)$.

From this, the implementation of `Init` is fairly straightforward: For the top-down algorithm,

$$\texttt{Init}(\psi) := \{\vec{a} \in \{0,1\}^m \mid \mathsf{consistent}_\psi(\vec{a})\},$$

and for the bottom-up algorithm,

$$\texttt{Init}(\psi) := \{\vec{a} \in \{0,1\}^m \mid \mathsf{consistent}_\psi(\vec{a}) \wedge \bigwedge_{\varphi_i = \Box\varphi'} a_i = 1\}.$$

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

– $\Diamond_{1,i}(\vec{x})$ is read as "$\vec{x}$ needs a witness for a diamond operator at position i" and is true iff $x_i = 0$ and $\varphi_i = \Box\varphi'$.

– $\Diamond_{2,i}(\vec{y})$ is read as "$\vec{y}$ 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 $y_j = 1$.

– $\Box_{1,i}(\vec{x})$ is read as "$\vec{x}$ requires support for a box operator at position i" and is true iff $x_i = 1$ and $\varphi_i = \Box\varphi'$.

– $\Box_{2,i}(\vec{y})$ is read as "$\vec{y}$ 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 $(\vec{x}, \vec{y})$ such that $\vec{y}$ supports all of $\vec{x}$'s box formulae. For types $\vec{x}, \vec{y} \in \{0,1\}^m$, we define

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

Given a set A of types, we write the corresponding characteristic function as χ_A, and we use $\chi_{\overline{A}}$ for the characteristic function of the complement of A. Next, we show how to implement the top-down and the bottom-up algorithm using the predicates χ_A, Δ, $\Diamond_{j,i}$, and $\Box_{j,i}$.

For the top-down approach, the predicate bad is true on those types that contain a negated box formula which is not witnessed in the current set of types. Thus, for a negated box formula $\varphi_i = \neg\Box\varphi_j$, we define the predicate bad_i as follows:

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

and thus $\mathsf{bad}(X)$ can be written as

$$\chi_{\mathsf{bad}(X)}(\vec{x}) = \bigvee_{1\leq i\leq m} \chi_{\mathsf{bad}_i(X)}(\vec{x}).$$

In our implementation, we compute each $\chi_{\overline{\mathsf{bad}_i(X)}}$ 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

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

For the top-down algorithm, the `Update` function can be written as:

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

For the bottom-up algorithm, we must take care to only add bit vectors representing types, and so the `Update` function can be implemented as:

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

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

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

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

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

Finally, the bottom-up algorithms can be implemented as iterations over the sets $\chi_{X\cup\mathsf{supp}(X)}$, and the top-down algorithms can be implemented as iterations over the sets $\chi_{X\setminus\mathsf{bad}(X)}$ until 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 [BUR 92] and it has been the subject of extensive research (cf. [BUR 91, GEI 94, RAN 95, CIM 00]) 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, namely one for each box subformula. Thus, to compute the pre-image preim_Δ, 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_is are Boolean formulae. Suppose, however, that the variable y_j does not occur in the clauses $c_{i+1}, \ldots, c_m$. Then the formula above 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_is. One way we used is the methodology developed in [RAN 95], called the "IWLS 95" methodology, to compute pre-images. We have also tried other clustering mechanisms, namely the "bucket-elimination" approach described in [San 01]. Given a set of conjunctive components $c_1, \ldots, c_n$, we first compute the variable support set for each component as $Y_1, \ldots, Y_n$. Then, a graph of interference of variables is constructed: every vertex represents a variable, and there is an edge between variables y_i and y_j if y_i and y_j occur together in some Y_k. We conduct a "maximum cardinality ordering" of the variables, after which y_1 is the variable that occurs with the maximal number of edges, and y_i has the maximum number of edges into $y_1, \ldots, y_{i-1}$. Given such a variable order, we can order the conjunctive components in the order of the first occurrence of the highest (or lowest) ordered variables (either forward or backward). We have implemented all four combinations in this case, but it will turn out that the performance improvements are minimal.

5.2. *Optimizations*

5.2.1. *Particles*

The encoding of the particle-based approach with BDDs is analogous to the encoding of the type-based approach. Since the consistency requirement for particles is more relaxed than that of types, each subformula in $\mathsf{sub}(\psi)$ (also the negated ones) is represented by a variable. Given $\mathsf{sub}(\psi) = \{\varphi_1, \ldots\varphi_n\}$, a vector $\vec{p} = \langle p_1, \ldots p_m \rangle \in \{0,1\}^n$ represents a set $p \subseteq \mathsf{sub}(\psi)$ with $\varphi_i \in p$ iff $p_i = 1$.

Then, as for types, we define a characteristic predicate for particle vectors $\mathsf{consistent}_\psi(\vec{p}) := \wedge_{1 \leq i \leq n} \mathsf{consistent}^i_\psi(\vec{p})$, where $\mathsf{consistent}^i_\psi(\vec{p})$ is defined as follows:

- if φ_i is neither of the form $\varphi_j \wedge \varphi_k$ nor $\varphi_j \vee \varphi_k$, then $\mathsf{consistent}^i_\psi(\vec{p}) = 1$,
- if $\varphi_i = \varphi_j \wedge \varphi_k$, then $\mathsf{consistent}^i_\psi(\vec{p}) = (p_i \rightarrow (p_j \wedge p_k))$,
- if $\varphi_i = \varphi_j \vee \varphi_k$, then $\mathsf{consistent}^i_\psi(\vec{p}) = (p_i \rightarrow (p_j \vee p_k))$, and
- if $\varphi_i = \neg\varphi_j$, then $\mathsf{consistent}^i_\psi(\vec{p}) = \neg(p_i \wedge p_j)$.

Finally, we update the auxiliary predicates for particles:

- $\Diamond_{1,i}(\vec{x})$ is true iff $x_i = 1$ and $\varphi_i = \Diamond\varphi'$,
- $\Diamond_{2,i}(\vec{y})$ is true iff $\varphi_i = \Diamond\varphi_j$ and $y_j = 1$,
- $\Box_{1,i}(\vec{x})$ is true iff $x_i = 1$ and $\varphi_i = \Box\varphi'$ (the same as for types), and
- $\Box_{2,i}(\vec{y})$ is true iff $\varphi_i = \Box\varphi_j$ and $y_j = 1$.

All other predicates such as preim and bad do not change.

5.2.2. *Lean vector approaches*

Lean approaches have much more relaxed consistency predicates at the cost of bigger witness/support predicates. For lean approaches, we first need to define

Only the $\mathsf{consistent}_i(\vec{x})$ that is related to those φ_i in $\mathsf{atom}(\psi)$ (or $\mathsf{part}(\psi)$) are used.

In contrast, the auxiliary (witness/support) predicate for the lean approach is significantly more complex. We now define the corresponding auxiliary functions for lean assignments.

For lean types and lean particles, $\Diamond_{1,i}$ and $\Box_{1,i}$ are the same as for full types and particles. However, since the subformula occurring inside a modal operator may be a Boolean combination, we need to redefine the functions $\Diamond_{2,i}$, $\Box_{2,i}$ with the same intuition as for full type and particle vectors. To do this, we first define the auxiliary function strip_i as follows:

$$\mathsf{strip}_i(\vec{y}) = \begin{cases} \mathsf{strip}_j(\vec{y}) \wedge \mathsf{strip}_k(\vec{y}) & \text{if } \varphi_i = \varphi_j \wedge \varphi_k \\ \mathsf{strip}_j(\vec{y}) \vee \mathsf{strip}_k(\vec{y}) & \text{if } \varphi_i = \varphi_j \vee \varphi_k \\ \neg\,\mathsf{strip}_j(\vec{y}) & \text{if } \varphi_i = \neg\varphi_j \\ y_i & \text{if } \varphi_i \in \mathsf{atom}(\psi) \text{ for types or } \mathsf{part}(\psi) \text{ for particles} \end{cases}$$

Obviously, for both lean types and lean particles , strip_i can be computed when parsing the input formula, and be kept in a table.

Next, $\Diamond_{2,i}$ and $\Box_{2,i}$ can be defined as follows:

$$\begin{aligned} \Diamond_{2,i}(\vec{y}) &= \begin{cases} \mathsf{strip}_j(\vec{y}) & \text{for particles, if } \varphi_i = \Diamond\varphi_j \\ \neg\,\mathsf{strip}_j(\vec{y}) & \text{for types, if } \varphi_i = \Box\varphi_j \\ \mathsf{strip}_j(\vec{y}) & \text{for types, if } \varphi_i = \Box\neg\varphi_j \end{cases} \\ \Box_{2,i}(\vec{y}) &= \mathsf{strip}_j(\vec{y}) \text{ for } \varphi_i = \Box\varphi_j \end{aligned}$$

Again, all other predicates such as preim and bad do not change.

5.2.3. *Level-based evaluation*

The level-based evaluation approaches is computed in a similar way. However, since all levels are treated separately, at each level, we only need to consider those unwitnessed box formulae of that level—before, we had to consider all possibly unwitnessed negated box formulae—which leads to the relativized predicate $\overline{\mathsf{bad}_j(X)}$. Similarly, to test whether a given vector indeed represents a type, the constraint predicate $\mathsf{consistent}^i_\psi$ for the level-based approach only needs to consider subformulae of the same level i. So, we can relativize both the lean and the full variant of the type approach by defining $\chi_{\mathsf{level}_i(X)}$ as follows:

$$\chi_{\mathsf{level}_i(X)}(\vec{x}) = \chi_{\mathsf{consistent}^i_\psi}(\vec{x}) \wedge \bigwedge_{\{j \mid \varphi_j \in \mathsf{cl}_i(\psi)\}} (\chi_{\overline{\mathsf{bad}_j(X)}}(\vec{x})).$$

Then we can specify the level-based variants by setting $\chi_{\mathtt{Init}_i}(\vec{a}) = \mathsf{consistent}_i(\vec{a})$ and $\chi_{\mathtt{Update}(A,i)}(\vec{a}) = \chi_{\mathsf{consistent}_i}(\vec{a}) \wedge \chi_{\mathsf{level}_i(A)}(\vec{a})$.

The level-based evaluation for particles can be implemented in the same way by replacing cl_i with sub_i and relativizing the corresponding predicates for particles.

5.3. *Variable ordering*

It is well-known that the performance of BDD-based algorithms is very sensitive to BDD variable order since it is a primary factor influencing BDD size [BRY 86]. In our experiments, a major factor in performance degradation is space blow-ups of BDDs, including the intermediate BDDs computed during pre-image operation. In all our algorithms, however, every step in the REPEAT loop uses BDDs with variables from different modal depth, and thus dynamic variable ordering is of limited benefit for $\mathcal{KBDD}$ (though it is necessary when dealing with intermediate BDDs blowups) because there may not be sufficient reuse to make it worthwhile. Thus, we focused here on heuristics to construct a good initial variable order, i.e., one that is appropriate for $\mathcal{KBDD}$. In this, we follow the work of Kamhi and Fix [KAM 98a] who argued in favor of application-dependent variable order. As we show in Section 7.1.5, choosing a good initial variable order does improve performance, but the improvement is rather modest.

A naive method for assigning an initial variable order to a set of subformulae would be to traverse the syntax DAG of the input formula[5] 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 "close" variables. We found that related variables correspond to subformulae that are related via the "sibling" or "niece" relationships. More precisely, we say that v_x is a *child* of v_y if, for the corresponding subformulae, we have that $\varphi_x \in \mathsf{sub}_i(\psi)$, $\varphi_y \in \mathsf{sub}_{i+1}(\psi)$, and φ_y is a subformula of φ_x, for some $0 \leq i < \mathsf{depth}(\psi)$.[6] 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*

---

[5] The syntax DAG is obtained from the syntax tree of a formula by identifying nodes labeled with the same subformula.

[6] For the type approach, sub_i has to be replaced with cl_i accordingly.

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.

Our heuristics builds a variable order from the root of the formula DAG down. We start with left-to-right traversal order of top variables in the parse tree of ψ as the order for variables corresponding to subformulae 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, and choose a location for v that minimizes the cumulative distance from other dependent variables. We refer to this approach as the *greedy* approach, as opposed to the *naive* approach of depth-first pre-order.

6. Reducing K to QBF

Both **K** and QBF have PSPACE-complete satisfiability problems [LAD 77, STO 77], and thus these two problems are polynomially reducible to each other. A natural reduction from QBF to **K** is described in [HAL 92]. In the last few years, extensive effort was carried out into the development of highly-optimized QBF solvers [GIU 01, CAD 99, LET 02]. One motivation for this effort is the hope of using QBF solvers as generic search engines [RIN 99], much is the same way that SAT solvers are being used as generic search engines, cf. [BIE 99]. This suggests that we can realistically hope to decide **K** satisfiability by using a natural reduction of **K** to QBF, and then applying one of the highly optimized QBF solver. Such an approach is suggested in [CAD 99] without providing either details or results. Next, we describe such a reduction, and evaluate it empirically in the next section, together with our $\mathcal{KBDD}$ algorithms.

QBF is an extension of propositional logic with quantifiers. The set of QBF formulae 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 formulae, QBF formulae are interpreted over truth assignments $\tau : \Phi \longrightarrow \{1, 0\}$. The semantics of quantifiers is defined as usual for the Boolean part, and as follows for the quantifiers: $\tau \models \forall p.\varphi$ iff $\tau[p/0] \models \varphi$ and $\tau[p/1] \models \varphi$, where $\tau[p/i]$ is obtained from τ by setting $\tau(p) := i$.

By Theorem 18, a $\mathbf{K}$ formula ψ of modal depth d is satisfiable iff there exists a sequence $\mathbf{P} = \langle P_0, P_1, \ldots, P_d\rangle$ of particle sets such that $\psi \in p$ for some $p \in P_0$. We construct QBF formulae $f_0, f_1, \ldots f_d$ so that each f_i encodes the particle set P_i. 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. Then, for each $p \subseteq \mathsf{sub}_i(\psi)$, we 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 $P_i = \{p \subseteq \mathsf{sub}_i(\psi) | \tau_p^i \models f_i\}$. We then say that f_i *characterizes* P_i.

To define f_i, we need some notation: we use $\mathsf{particle}_i(\psi)$ for the set of all consistent particle vectors of $\mathsf{sub}_i(\psi)$. We start by constructing a propositional formula lc_i such that, for each $p \subseteq \mathsf{sub}_i(\psi)$ we have that $p \in \mathsf{particle}_i(\psi)$ iff $\tau_p^i \models lc_i$. The formula lc_i is a conjunction of clauses as follows:

– For $\varphi = \neg\varphi' \in \mathsf{sub}_i(\psi)$, we have the clause $x_{\varphi,i} \rightarrow \neg x_{\varphi',i}$.

– For $\varphi = \varphi' \wedge \varphi'' \in \mathsf{sub}_i(\psi)$, we have the clauses $x_{\varphi,i} \rightarrow x_{\varphi',i}$ and $x_{\varphi,i} \rightarrow x_{\varphi'',i}$.

– For $\varphi = \varphi' \vee \varphi'' \in \mathsf{sub}_i(\psi)$, we have the clause $x_{\varphi,i} \rightarrow (x_{\varphi'',i} \vee x_{\varphi'',i})$.

For $i = d$ we simply set $f_d := lc_d$. Indeed, we have $P_d = \mathsf{particle}_d(\psi) = \{p \subseteq \mathsf{sub}_d(\psi) \mid \tau_p^d \models f_d\}$. Thus, f_d characterizes $\mathtt{Init}_d(\psi)$.

For $i < d$, suppose we have 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 set $f'_d = f_d$ and

$$f'_i := lc_i \wedge \bigwedge_{\Diamond\varphi \in \mathsf{sub}_i(\psi)} mc_{\Diamond\varphi},$$

where $mc_{\Diamond\varphi}$ ensures 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, for $\mathsf{sub}_{i+1}(\psi) = \{\theta_1, \ldots, \theta_{k_{i+1}}\}$, we set

$$\begin{array}{rcl} mc_{\Diamond\varphi} & := & x_{\Diamond\varphi,i} \rightarrow \exists x_{\theta_1,i+1} \ldots \exists x_{\theta_{k_i},i+1}(f_{i+1} \wedge x_{\varphi,i+1} \wedge tr_i), \text{ where} \\ tr_i & := & \bigwedge_{\Box\eta \in \mathsf{sub}_i(\psi)} [x_{\Box\eta,i} \rightarrow x_{\eta,i+1}]. \end{array}$$

LEMMA 23. — *If f'_{i+1} characterizes P_{i+1}, then f'_i characterizes $P_i = \mathtt{Update}(P_{i+1}, i)$.*

PROOF. — By construction, lc_i characterizes $\mathsf{part}_i(\psi)$. For the witnessing requirement, we can see that, if $\tau_p^i \models mc_{\Diamond\varphi}$ and $x_{\Diamond\varphi,i}$, then there is an assignment $\tau_{p'}^{i+1}$ where $\tau_p^i \cup \tau_{p'}^{i+1} \models f'_{i+1} \wedge x_{\varphi,i+1} \wedge tr_i$. This is equivalent to asserting that $p' \in P_{i+1}$, $\varphi \in p'$ and $R_i(p, p')$. ■

COROLLARY 24. — *ψ is satisfiable iff $\exists x_{\theta_1,0} \ldots \exists x_{\theta_{k_0},0} x_{\psi,0} \wedge f'_0$ is satisfiable.*

PROOF. — The claim follows from the soundness and completeness of $\mathcal{KBDD}$. ■

This reduction of $\mathbf{K}$ 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. To construct f_i which uses only a single copy of f_{i+1}, we replace the conjunction over all $\Diamond\varphi$ formulae in $\mathsf{sub}_i(\psi)$ by a universal quantification. Let k be an upper bound on the number of $\Diamond\varphi$ formulae in $\mathsf{sub}_i(\psi)$, for $0 \leq i \leq \mathsf{depth}(\psi)$. 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 $m = \lceil \log(k) \rceil$. We introduce m new Boolean variables $y_1, \ldots, y_m$. Each truth assignment to the variables y_i represents a number between 0 and k in binary coding, and we refer to this number by $val(\mathbf{y})$ and use it to refer to $\Diamond$ subformulae. Let $witness_i$ be the formula $\bigvee_{j=0}^{k-1} x_{\xi^i_j}$, which asserts that some witnesses are required.

Using this notation, we can now write f_i in a compact way:

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

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

The formula f_i first asserts the local consistency constraint lc_i. The quantification on $y_1, \ldots, y_m$ simulates the conjunction on all k $\Diamond$ subformulae 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 $\Box$ subformulae. Finally, we let $val(\mathbf{y})$ point to the $\Diamond$ subformulae that needs to be witnesses. Note that f_i contains only one copy of f_{i+1}.

LEMMA 25. — *f_i and f'_i are logically equivalent.*

As an immediate consequence of the above lemma and Corollary 24, we obtain the following result.

COROLLARY 26. — *ψ is satisfiable iff $\exists x_{\theta_1,0} \ldots \exists x_{\theta_{k_0},0} x_{\psi,0} \wedge f_0$ is satisfiable.*

Analogously to our other approaches, this approach can be optimized further by reducing redundancy, i.e., by restricting variables to those representing non-Boolean subformulae. We implemented this optimization and report on our experiments in the next section.

7. Experimental Results

In this section, we report on our empirical evaluation of the algorithms described throughout this paper and their optimizations. We first report on comparisons between the various $\mathcal{KBDD}$ algorithms and analyze the effects of the different optimizations and the influence of variable ordering. This allows us to determine the "best" configuration for our $\mathcal{KBDD}$ approach. Secondly, we compare this best $\mathcal{KBDD}$ algorithm with other **K** solvers and with an implementation of the translation into QBF method described in the previous section.

We implemented the BDD-based decision procedure and its variants in C++ using the CUDD 2.3.1 [SOM 98] package for BDDs, and we implemented formula simplification preprocessor in OCaml. The parser for the languages used in the benchmark suites are taken with permission from *SAT [TAC 99].[7]

---

[7] All tests were 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 with parts in OCaml 3.04.

7.1. *Comparing the $\mathcal{KBDD}$ variants*

To analyze the usefulness of each optimization techniques used, we run the algorithm with different optimization configurations on the **K** part of TANCS 98 [HEU 96] and the MODAL PSPACE division of TANCS 2000 [MAS 00]. Both are suites of scalable benchmarks which contain both provable and non-provable formulae. In TANCS 98, simple formulae have their difficulty increased by re-encoding them with superfluous subformulae. In TANCS 2000, formulae are constructed by translating QBF formulae into **K** using three translation schemes, namely Schmidt-Schauss-Smolka translation, which gives easy formulae, Ladner translation, which gives medium difficulty formulae, and Halpern translation, which gives hard formulae. TANCS 98 provides more "easy" formulae, and we used it to provide a clearer picture in case that the unoptimized algorithms took too long on TANCS 2000. To get a clearer picture and a guideline, each test was also run with *SAT, and we limited the memory available for BDDs to 384MB and the time to 1000s.

7.1.1. *The basic algorithms*

To compare our basic algorithms, top-down and bottom-up using full types, we run them both on TANCS 98. The results are presented in Fig. 1. We can see that *SAT clearly outperforms our two basic algorithms. A reason for this "weak" behavior of our approaches is that the intermediate results of the pre-image operation are so large that the we ran out of memory. The difference between top-down and bottom-up approaches is minor. Top-down slightly outperforms bottom-up since top-down removes types, which only requires the consistency requirement to be asserted once before iteration, while bottom-up adds types, which requires an extra conjunction to ensure only consistent types are added.

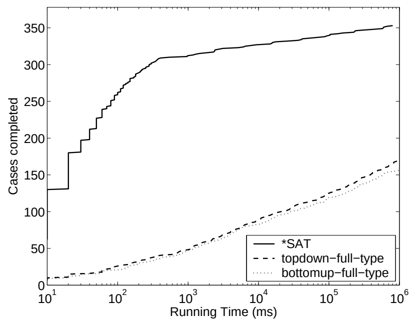


Figure 1. *Top-down versus bottom-up on TANCS 98*

7.1.2. *Particle approaches*

Next, we compare the variants using types with their full particle-based variants. The results are presented in Fig. 2. We can see that, on 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 a reduction of the number of operations to compute pre-images.

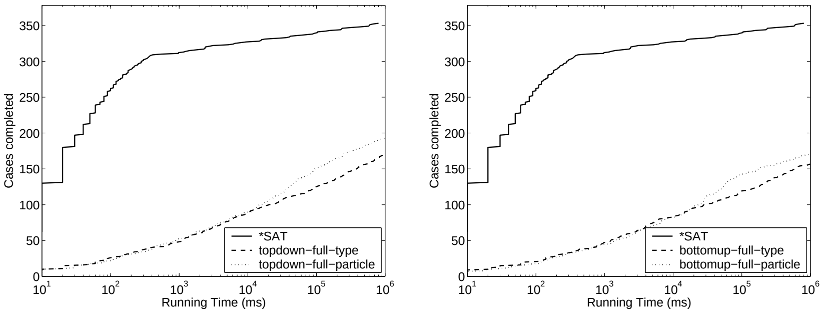


Figure 2. *Particles vs. types on TANCS 98*

7.1.3. *Lean approaches*

Since, so far, all variants behave quite similar, we compare the “full” approaches with their lean variants for types and particles, bottom-up and top-down. The results can be found in Fig. 3 and 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, the lean approaches outperform their full variants in each combination. This indicates that, as a general guideline, we should 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 [KAM 98b].

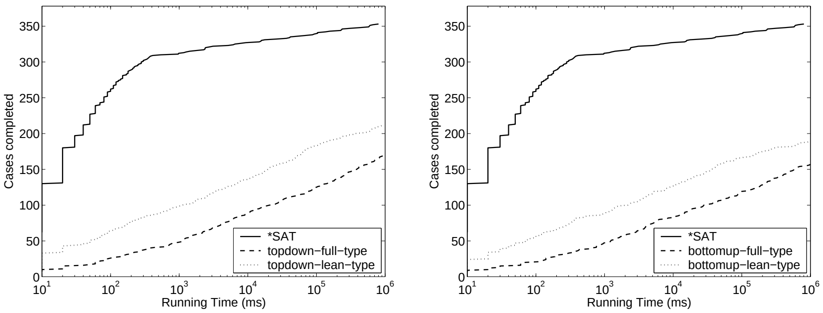


Figure 3. *Lean vs. full types on TANCS 98*

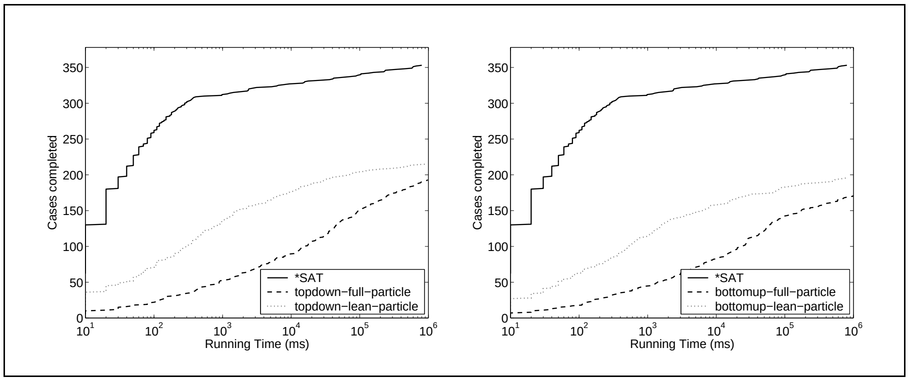


Figure 4. *Lean vs. full particles on TANCS 98*

7.1.4. *Level-based evaluation*

Next, we compare various level-based approaches with the top-down lean type approach, see Fig. 5. It turns out that each level-based approach outperforms the top-down approach, and that, both for types and particles, the lean approach again outperforms the full one. This superior performance of the level-based approaches is, again, due to a smaller BDD size: recall that, in the level-based approach, all BDDs are split into smaller ones according to the level at which the corresponding formulae occur in the input. Moreover, the level-based approach involves a smaller number of operations to compute the pre-image, and this turns out to be substantial for most formulae.

Summing up, our experimental comparison of the different BDD-based algorithms indicate that level-based lean particle version performs best, and we thus use, in the remainder of this paper, $\mathcal{KBDD}$ to refer to this version of our algorithm.

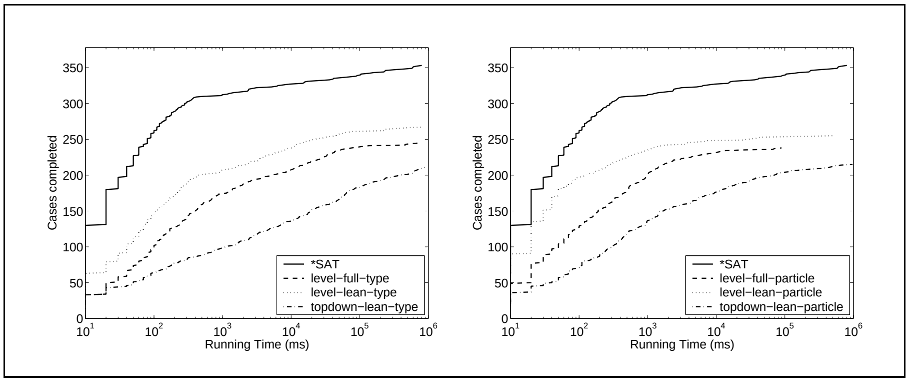
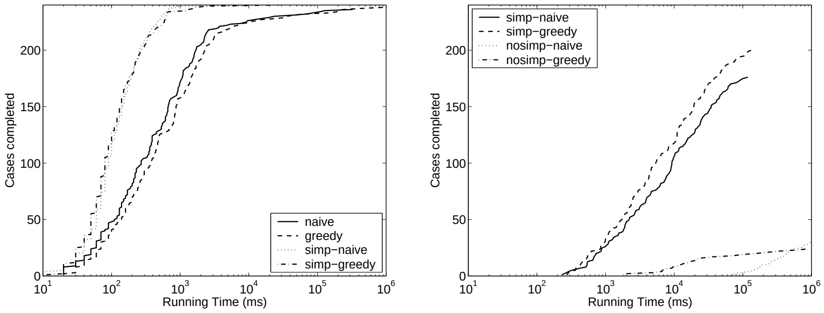


Figure 5. *Level-based evaluation vs. top-down lean approaches on TANCS 98*

7.1.5. *Variable ordering and formula simplification*

To gain inside into the effects of variable ordering and formula simplification, we tested $\mathcal{KBDD}$ with both naive and greedy variable ordering described in Section 5.3, and with and without the formula simplification described in Section 4.4. We compared the influence of these optimizations using TANCS 2000 easy and medium formulae [MAS 00] ($\mathcal{KBDD}$ without formula simplification cannot handle the hard formulae of TANCS 2000). The results are presented in Figure 6.



TANCS 2000 Easy (cnfSSS)

TANCS 2000 Medium (cnfLadn)

Figure 6. *Different optimizations for $\mathcal{KBDD}$ on TANCS 2000*

We see in Figure 6 that formula simplification yields a significant performance improvement. This improvements was observed for different types of formulae and different variable-ordering algorithms. In particular, $\mathcal{KBDD}$ was able to avoid running out of memory in many cases. We can also see that greedy variable ordering is useful in conjunction with simplification, improving the number of completed cases and sometimes run time as well. Without simplification, the results for greedy variable ordering are not consistent: the overhead of finding the variable order seems to sometimes offset any advantages of applying it.

Summing up, our experiments indicate that the combination of simplification and greedy variable ordering significantly improves the performance of $\mathcal{KBDD}$. In the following, we will use "optimized $\mathcal{KBDD}$" to refer to this variant, and we will compare its performance with that of three other solvers.

7.2. *Comparing $\mathcal{KBDD}$ with other solvers*

To assess the effectiveness of BDD-based decision procedures for **K**, we compare the optimized $\mathcal{KBDD}$ against three solvers: (1) DLP, which is a tableau-based solver [PAT 99], (2) MSPASS, which is a combination of an optimized translation of modal formulae to first-order formulae and a resolution-based theorem prover [HUS 00][8], (3) K-QBF, which is a combination of our reduction of **K** to QBF from Section 24 and the highly optimized QBF solver semprop [LET 02]. For a fair comparison, we first checked for which our simplification optimization is useful, and then used it in these two cases, namely for DLP and K-QBF.

In addition to the formulae from TANCS 98 and TANCS 2000, we also use randomly generated formulae, as suggested in [PAT 01]. This scheme generates random modal-CNF formulae parameterized with the number N of propositions, the number K of literals in each clause, the fraction α of modal literals in each clause, the modal-depth bound d, and the number L of top level clauses. L clauses are generated with K literals each, where αK literals are modal and the rest are propositional (the polarity of the literals is chosen uniformly). Each modal literal is expanded into a clause in the same fashion. The modal depth of the formula is bounded by d. We used $d = [1, 2]$, $K = 3$ and $\alpha = 0.5$ in our experiments. In each experiment, N is fixed and the propositional complexity of the formula was varied by increasing the *density* L/N.

7.2.1. *Results on TANCS suites*

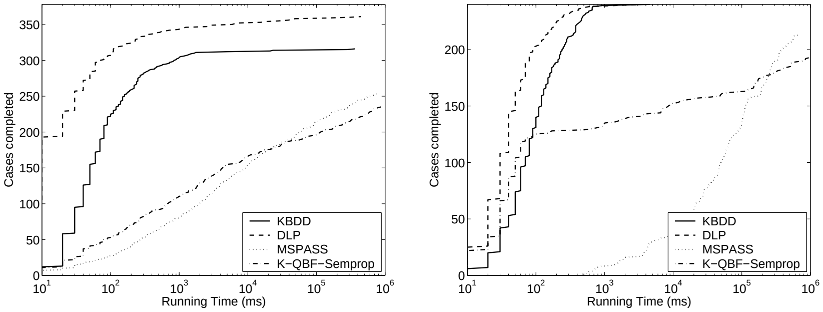
In Figure 7 and Figure 8 we see that, on the TANCS 98 benchmarks, DLP outperforms all other solvers whereas, on the more challenging TANCS 2000 benchmarks, $\mathcal{KBDD}$ outperforms the other solvers. The difference between $\mathcal{KBDD}$ and the other solvers is most noticeable on the harder portion of the suite, where $\mathcal{KBDD}$ had to use dynamic variable reordering. We take this as an indicator that, indeed, BDD-based approaches may be useful in practice for **K** satisfiability. MSPASS's performance could have been enhanced by a different choice for its numerous parameters. However, we have chosen to stick to the setting that worked well for TANCS98 since (a) the current investigation is merely a feasibility study, and (b) finding optimal parameter settings for MSPASS for each experiment would go beyond the scope of this paper. Finally, it turns out that reducing **K** satisfiability to a search-based QBF solver such as semprop is not a viable approach: this approach was dominated by all other approaches and was only able to solve a small fraction of the benchmark formulae in TANCS 98. For TANCS 2000 this approach was so inefficient that we did not report the results.

7.2.2. *Results on random modal CNF formulae*

More insight into the behavior of $\mathcal{KBDD}$ can be gained by analyzing its behavior on random modal-CNF formulae. The generation of the formulae follows the suggestions in [PAT 01]. This scheme generates random modal-CNF formulae parametrized with the number N of propositions, the number K of literals in each clause, the fraction α of modal literals in each clause, the modal-depth bound d, and the number L of top level clauses. L clauses are generated with K literals each, where αK literals are

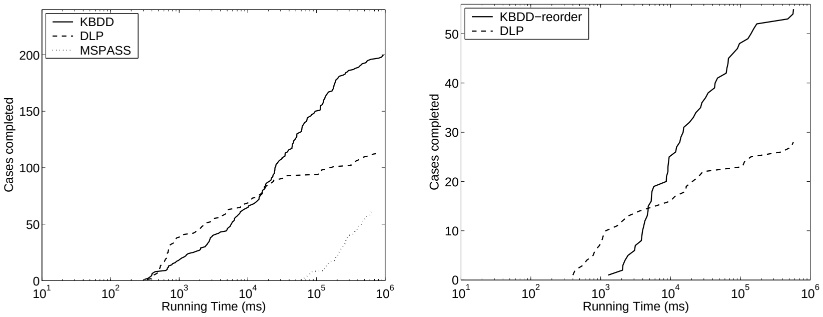
---

[8] We used MSPASS 1.0.0t1.3 with options -EMLTranslations=2 -EMLFuncNary=1 -Select=2 -PProblem=0 -PGiven=0 -Sorts=0 -CNFOptSkolem=0 -CNFStrSkolem=0 -CNFRenOps=1 -Split=-1 -Ordering=0 -CNFRenMatch=0 -TimeLimit=1000. Compiler used is gcc-3.1.1 because gcc-2.96 have a serious bug that crashes the resulting executable.



TANCS 98 TANCS 2000 Easy (cnfSSS)

Figure 7. *$\mathcal{KBDD}$ vs. DLP, K-QBF and MSPASS on "easy"* **K** *formulae*



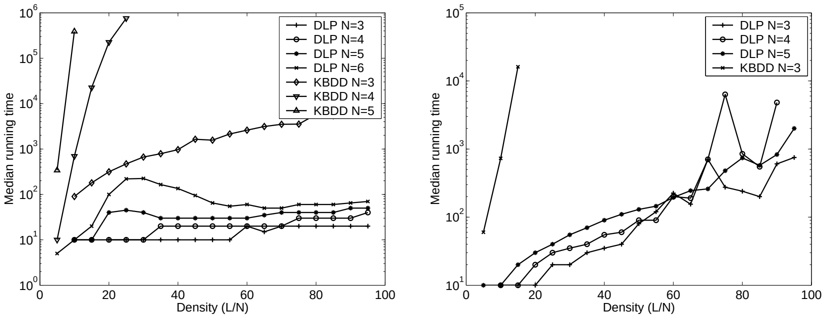
TANCS 2000 Medium (cnfLadn) TANCS 2000 Hard (cnf)

Figure 8. *$\mathcal{KBDD}$ vs. DLP, and MSPASS on more "difficult"* **K** *formulae*

modal and the rest are propositional (the polarity of the literals is chosen uniformly). Each modal literal is expanded into a clause in the same fashion. The modal depth of the formula is bounded by d. We used $d = 1, 2$, $K = 3$ and $\alpha = 0.5$ in our experiments. In each experiment N was fixed and the propositional complexity of the formula was varied by increasing the *density* L/N.

In Figure 9, we show the median run time (16 samples per data point) as a function of the density (L/N) to demonstrate the difference between the behavior of $\mathcal{KBDD}$ and DLP.

For $d = 1$, DLP demonstrates the bell-shaped "easy-hard-easy" pattern that is familiar from random propositional CNF formulae [SEL 96] and random QBF formulae [GEN 99]. In contrast, $\mathcal{KBDD}$'s run time is proportional to the density; that is, the higher the density, the harder the problem for $\mathcal{KBDD}$. This behavior is consistent with



$d = 1$ $d = 2$

Figure 9. *Comparison of DLP and $\mathcal{KBDD}$ on Random formulae*

known results on the performance of BDD-based algorithms for random propositional CNF formulae [COA 00]. For each modal level, $\mathcal{KBDD}$ builds a BDD for the corresponding particle set. The higher the density, the more challenging the construction of these BDDs becomes, often resulting in running out of memory or requiring extensive variable reordering. This explains why DLP outperforms $\mathcal{KBDD}$ on random modal-CNF formulae.

Comparing these findings with the ones on TANCS 98 and TANCS 2000, we conclude that DLP is better suited for formulae with high propositional complexity such as the randomly generated ones, whereas $\mathcal{KBDD}$ is better suited for formulae with high modal complexity such as the ones in TANCS 98 and TANCS 2000.

8. Conclusions and outlook

We described here BDD-based decision procedures for **K**. Our approach is inspired by the automata-based approach, but it avoids explicit automata construction. We explored a variety of representation options and concluded that, in general, it is beneficial to work with representations that involve fewer constraints, i.e., with particles. In general, the best performance was obtained with lean particles. It also turned out that only a level-based approach yields a competitive implementation. Furthermore, formula preprocessing such as pure literal simplification and syntactical simplification proved to have a significant influence on the performance, even though it is not specialized to our method. Finally, we tested various BDD-centric optimizations such as clustering with early quantification and initial variable ordering but the effect proved to be rather modest.

We benchmarked $\mathcal{KBDD}$, our optimized BDD-based solver, against both a native modal solver, DLP, and two translation-based solver, MSPASS and K-QBF. Our results indicate that the BDD-based approach dominates for modally heavy formulae, while search-based approaches dominate for propositionally heavy formulae.

One way to look at the results is that $\mathcal{KBDD}$, by using a more powerful underlying data structure, BDDs, allows the use of a simpler decision procedure. Instead of requiring, in the worst case, an exponential number of calls to a propositional satisfiability procedure, $\mathcal{KBDD}$ only requires a polynomial number of calls to BDD operations. The natural question arising is, of course, whether such a dependence on the data structure is reasonable. We know that the complexity of BDD operations is highly dependent to the size of the BDDs. So, if we are able to control the size of the BDDs, the performance of our decision procedure is acceptable.

Another explanation is that we traded modal complexity for propositional complexity. This way, we managed to perform quite well on formulae that only have "big" models. Such formulae are known to cause problems SAT based solvers. We observe that, on formulae that are satisfiable in "small" models, SAT solvers outperform $\mathcal{KBDD}$. In contrast, on formulae that are only satisfiable in "big" models, $\mathcal{KBDD}$ outperform SAT solver. In both cases, the performance of both solvers degrades with increasing propositional density.

Although our goal was a feasibility study for a BDD-based approach to modal solvers, and not to develop the "fastest **K** solver", the $\mathcal{KBDD}$ approach turned out to behave quite well on a considerable number of benchmarks. This is mostly due to the fact that our approach allowed us to compare and explore the effects of numerous optimizations. One obvious optimization technique we did not explore is to avoid the construction of a monolithic BDD, such as the technique developed for the purely propositional case [San 01]. Further research is also required to quantify the distinction between "propositionally heavy" and "modally heavy" formulae. This would enable the development of a combined solver which invokes the appropriate engine for the formula under test. Another approach would be to develop a hybrid solver, combining BDD-based and search-based techniques (cf. [GUP 01] for a hybrid approach in model checking), which would perform well on both modally heavy and propositionally heavy formulae. Finally, we hope the connection between **K** and QBF would be part of a call to a more comprehensive range of decision procedures for QBF, for example, the quantifier-elimination based solvers Quantor [BIE 04] and QMRES [PAN 04] that used Q-resolution [BUN 95]. We leave all this for future research.

Acknowledgements

The authors are 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.

9. References

[AND 98] ANDERSEN H., “An Introduction to Binary Decision Diagrams”, report , 1998, Department of Information Technology, Technical University of Denmark.

[ARE 00] ARECES C., GENNARI R., HEGUIABEHERE J., DE RIJKE M., “Tree-Based Heuristics in Modal Theorem Proving”, *Proc. of the 14th Eur. Conf. on Artif. Int.*, 2000, p. 199-203.

[BAA 01] BAADER F., TOBIES S., “The inverse method implements the automata approach for modal satisfiability”, report num. LTCS-Report 01-03, 2001, Research group for theoretical computer science, Aachen university of Technology.

[BEE 94] BEER I., BEN-DAVID S., GEIST D., GEWIRTZMAN R., YOELI M., “Methodology and system for practical formal verification of reactive hardware”, *Proc. 6th Conf. on CAV*, vol. 818 of *LNCS*, Stanford, June 1994, p. 182–193.

[BIE 99] BIERE A., CIMATTI A., CLARKE E., ZHU Y., “Symbolic Model Checking without BDDs”, *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS ’99*, vol. 1579 of *LNCS*, Springer-Verlag, 1999, p. 193-207.

[BIE 04] BIERE A., “Resolve and Expand”, *Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2004)*, 2004, p. 238-246.

[BLA 01] BLACKBURN P., DE RIJKE M., VENEMA Y., *Modal logic*, Camb. Univ. Press, 2001.

[BOC 82] BOCHMANN G. V., “Hardware specification with temporal logic: an example”, *IEEE Transactions on Computers*, vol. C-31, 1982, p. 223–231.

[BRA 94] BRAFMAN R., LATOMBE J.-C., MOSES Y., SHOHAM Y., “Knowledge as a tool in motion planning under uncertainty”, FAGIN R., Ed., *Theoretical Aspects of Reasoning about Knowledge: Proc. Fifth Conference*, p. 208–224, Morgan Kaufmann, San Francisco, Calif., 1994.

[BRY 86] BRYANT R., “Graph-based Algorithms for Boolean Function Manipulation”, *IEEE Trans. on Comp.*, vol. Vol. C-35, num. 8, 1986, p. 677-691.

[BUN 95] BUNING H., KARPINSKI M., FLOGEL A., “Resolution for quantified Boolean formulas”, *Inf. and Comp.*, vol. 117(1), 1995, p. 12-18.

[BUR 88] BURROWS M., ABADI M., NEEDHAM R., “Authetication: a practical study in belief and action”, *Proc. 2nd Conference on Theoretical Aspects of Reasoning about Knowledge*, 1988, p. 325–342.

[BUR 91] BURCH J. R., CLARKE E. M., LONG D. E., “Symbolic Model Checking with Partitioned Transition Relations”, *Int. Conf. on VLSI*, 1991, p. 49–58.

[BUR 92] BURCH J., CLARKE E., MCMILLAN K., DILL D., HWANG L., “Symbolic model checking: 10^{20} states and beyond”, *Information and Computation*, vol. 98, num. 2, 1992, p. 142–170.

[CAD 99] CADOLI M., SCHAERF M., GIOVANARDI A., GIOVANARDI M., “An algorithm to evaluate quantified Boolean formulae and its experimental evaluation”, report , 1999, Dipartmento di Imformatica e Sistemistica, Universita de Roma.

[CAS 82] CASTILHO J. M. V., CASANOVA M. A., FURTADO A. L., "A temporal framework for database specification", *Proc. 8th Int. Conf. on Very Large Data Bases*, 1982, p. 280-291.

[CIM 00] CIMATTI A., CLARKE E., GIUNCHIGLIA F., ROVERI M., "NUSMV: A New Symbolic Model Checker", *Int. J. on Software Tools for Tech. Transfer*, vol. 2, num. 4, 2000, p. 410-425.

[CLA 86] CLARKE E., EMERSON E., SISTLA A., "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications", *ACM Transactions on Programming Languages and Systems*, vol. 8, num. 2, 1986, p. 244-263.

[COA 00] COARFA C., DEMOPOULOS D., SAN MIGUEL AGUIRRE A., SUBRAMANIAN D., VARDI M., "Random 3-SAT: The Plot Thickens", *Proc. of the Int. Conf. on Constraint Prog. (CP 2000)*, 2000, p. 143-159.

[DAV 60] DAVIS S., PUTNAM M., "A computing procedure for quantification theory", *J. ACM*, vol. 7, 1960, p. 201-215.

[DAV 62] DAVIS M., LOGEMANN G., LOVELAND D., "A machine program for theorem proving", *Journal of the ACM*, vol. 5, 1962, p. 394-397.

[ETE 00] ETESSAMI K., HOLZMANN G., "Optimizing Büchi Automata", *CONCUR 2000 - Concurrency Theory, 11th Int. Conf.*, 2000, p. 153-167.

[GEI 94] GEIST D., BEER H., "Efficient Model Checking by Automated Ordering of Transition Relation Partitions", *Proc. of the sixth Int. Conf. on CAV*, 1994, p. 299–310.

[GEN 99] GENT I., WALSH T., "Beyond NP: The QSAT Phase Transition", *AAAI: 16th National Conference on Artificial Intelligence*, AAAI / MIT Press, 1999, p. 648-653.

[GIU 00] GIUNCHIGLIA F., SEBASTIANI R., "Building Decision Procedures for Modal Logics from Propositional Decision Procedure - The Case Study of Modal K(m)", *Inf. and Comp.*, vol. 162, 2000, p. 158-178.

[GIU 01] GIUNCHIGLIA E., NARIZZANO M., TACCHELLA A., "QuBE, a system for deciding quantified Boolean formulae satisfiability", *Automated Reasoning, First Int. Joint Conf., IJCAR 2001*, 2001, p. 364-369.

[GUP 01] GUPTA A., YANG Z., ASHAR P., ZHANG L., MALIK S., "Partition-Based Decision Heuristics for Image Computation Using SAT and BDDs", *International Conference on Computer-Aided Design (ICCAD 2001)*, 2001, p. 286-292.

[HAA 01] HAARSLEV V., MÖLLER R., "High Performance Reasoning with Very Large Knowledge Bases: A Practical Case Study", NEBEL B., Ed., *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, vol. 1847, Morgan Kaufmann, Los Altos, 2001.

[HAL 90] HALPERN J. Y., MOSES Y., "Knowledge and Common Knowledge in a Distributed Environment", *Journal of the ACM*, vol. 37, num. 3, 1990, p. 549–587, A preliminary version appeared in *Proc. 3rd ACM Symposium on Principles of Distributed Computing*, 1984.

[HAL 92] HALPERN J., MOSES Y., "A guide to completeness and complexity for modal logics of knowledge and belief", *Artificial Intelligence*, vol. 54, 1992, p. 319-379.

[HEU 96] HEUERDING A., SCHWENDIMANN S., "A benchmark method for the propositional modal logics K, KT, S4", report , 1996, Universität Bern, Switzerland.

[HOR 00] HORROCKS I., SATTLER U., TOBIES S., "Practical Reasoning for Very Expressive Description Logics", *Logic Journal of the IGPL*, vol. 8, num. 3, 2000, p. 239–264.

[HUS 00] HUSTADT U., SCHMIDT R., "MSPASS: modal reasoning by translation and first order resolution", *Automated Reasoning with Analytic Tableaux and Related Methods, Int. Conf., TABLEAUX 2000*, 2000, p. 67-71.

[KAM 98a] KAMHI G., FIX L., "Adaptive variable reordering for symbolic model checking", *International Conference on Computer-Aided Design (ICCAD 1998)*, 1998, p. 359-365.

[KAM 98b] KAMHI G., FIX L., BINYAMINI Z., "Symbolic Model Checking Visualization", *Formal Methods in Computer-Aided Design, Second International Conference FMCAD'98*, vol. 1522 of *LNCS*, Springer-Verlag, November 1998, p. 290-303.

[LAD 77] LADNER R., "The Computational Complexity of Provability in Systems of Modal Propositional Logic", *SIAM J. Comput.*, vol. 6, num. 3, 1977, p. 467-480.

[LET 02] LETZ R., "Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas", *TABLEAUX 2002*, 2002, p. 160-175.

[LIP 77] LIPSKI W., "On the logic of incomplete information", *Proc. 6th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol. 53, p. 374–381, Springer-Verlag, Berlin/New York, 1977.

[MAS 00] MASSACCI F., DONINI F., "Design and results of TANCS-2000", *Automated Reasoning with Analytic Tableaux and Related Methods, Int. Conf., TABLEAUX 2000*, 2000, p. 52-56.

[MCC 69] MCCARTHY J., HAYES P. J., "Some Philosophical Problems From the Standpoint of Artificial Intelligence", MICHIE D., Ed., *Machine Intelligence 4*, p. 463–502, Edinburgh University Press, Edinburgh, 1969.

[PAN 04] PAN G., VARDI M. Y., "Symbolic Decision Procedures for QBF", *Proceedings of 10th Int. Conf. on Principles and Practice of Constraint Programming (CP 2004)*, 2004, p. 453-467.

[PAT 99] PATEL-SCHNEIDER P., HORROCKS I., "DLP and FaCT", *Automated Reasoning with Analytic Tableaux and Related Methods, Int. Conf., TABLEAUX '99*, 1999, p. 19-23.

[PAT 01] PATEL-SCHNEIDER P., SEBASTIANI R., "A new system and methodology for generating random modal formulae", *Automated Reasoning, First Int. Joint Conf., IJCAR 2001*, 2001, p. 464-468.

[PNU 77] PNUELI A., "The temporal logic of programs", *Proc. 18th IEEE Symp. on Foundation of Computer Science*, 1977, p. 46–57.

[PRA 76] PRATT V. R., "Semantical considerations on Floyd-Hoare logic", *Proc. 17th IEEE Symp. on Foundations of Computer Science*, 1976, p. 109–121.

[PRA 80] PRATT V., "A near-optimal method for reasoning about action", *Journal of Computer and System Sciences*, vol. 20, num. 2, 1980, p. 231–254.

[RAN 95] RANJAN R., AZIZ A., BRAYTON R., PLESSIER B., PIXLEY C., "Efficient BDD algorithms for FSM synthesis and verification", *Proc. of IEEE/ACM International Workshop on Logic Synthesis*, 1995.

[REI 83] REIF J. H., SISTLA A. P., "A multiprocessor network logic with temporal and spatial modalities", *Proc. 12th International Colloq. on Automata, Languages, and Programming*, Lecture Notes in Computer Science, Vol. 104, p. 629-639, Springer-Verlag, Berlin/New York, 1983.

[RIN 99] RINTANEN J., "Constructing conditional plans by a theorem-prover", *J. of A. I. Res.*, vol. 10, 1999, p. 323-352.

[RUD 93] RUDELL R., "Dynamic Variable Ordering for Ordered Binary Decision Diagrams", *International Conference on Computer-Aided Design (ICCAD 1993)*, 1993, p. 42-47.

[San 01] SAN MIGUEL AGUIRRE A., VARDI M., "Random 3-SAT and BDDs: The Plot Thickens Further", *Principles and Practice of Constraint Programming - CP 2001, 7th Int. Conf.*, 2001, p. 121-136.

[SEL 96] SELMAN B., MITCHELL D., LEVESQUE H., "Generating Hard Satisfiability Problems", *Artificial Intelligence*, vol. 81, num. 1-2, 1996, p. 17–29.

[SOM 98] SOMENZI F., "CUDD: CU Decision Diagram package", http://vlsi.colorado.edu/~fabio/CUDD/, 1998.

[SOM 00] SOMENZI F., BLOEM R., "Efficient Büchi automata from LTL formulae", *Computer Aided Verification, 12th Int. Conf., CAV 2000*, 2000, p. 247-263.

[STO 77] STOCKMEYER L., "The polynomial-time hierarchy", *Theo. Comp. Sci.*, vol. 3, 1977, p. 1–22.

[TAC 99] TACCHELLA A., "*SAT system description", *Collected Papers from (DL'99). CEUR*, 1999.

[TAN 93] TANI S., HAMAGUCHI K., YAJIMA S., "The Complexity of the Optimal Variable Ordering Problems of Shared Binary Decision Diagrams", *Algorithms and Computation, 4th International Symposium, ISAAC '93*, 1993, p. 389-398.

[VAR 97] VARDI M., "What makes modal logic so robustly decidable?", IMMERMAN N., KOLAITIS P., Eds., *Descriptive Complexity and Finite Models*, p. 149-183, AMS, 1997.

[VOR 01] VORONKOV A., "How to optimize proof-search in modal logics: new methods of proving redundancy criteria for sequent calculi", *Comp. Logic*, vol. 2, num. 2, 2001, p. 182-215.