RICE UNIVERSITY

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

by

Guoqiang Pan

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Master of Science

Approved, Thesis Committee:

Dr. Moshe Y. Vardi (Chair),
Karen Ostrum George Professor,
Computer Science

Dr. Devika S. Subramanian,
Associate Professor,
Computer Science

Dr. Dan Osherson,
Professor,
Psychology

Houston, Texas

March, 2003

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

Guoqiang Pan

Abstract

We describe BDD-based decision procedures for $\mathcal{K}$. Our approach is inspired by the automata-theoretic approach, but we avoid explicit automata construction. Our algorithms compute the fixpoint of a set of types, which are sets of formulas satisfying some consistency conditions. We use BDDs to represent and manipulate such sets. By viewing the sets of types as symbolic encoding of all possible models of a formula, we developed particle-based and lean-vector-based representation techniques which gives more compact representations. By taking advantage of the finite-tree-model property of $\mathcal{K}$, we introduced a level-based evaluation scheme to speed up construction and reduce memory consumption. We also studied the effect of formula simplification on the decision procedures. As part of the benching procedure, we compared the BDD-based approach with a representative selection of current approaches, as well as developing an algorithm to translate $\mathcal{K}$ to QBF based on our decision procedure. Experimental results show that the BDD-based approach dominates for modally heavy formulas, while search-based approaches dominate for propositionally-heavy formulas.

Acknowledgments

I wish to thank a number of people who have contributed to the making of this thesis. First and foremost, I would like to thank my adviser Dr. Moshe Y. Vardi for his visionary guidance throughout the project, and the freedom and flexibility he allowed me at every stage. The driving force can be rightly attributed to his adamant insistence, especially during the formative days, that this project can actually be pulled off. Dr. Ulrike Sattler provided helpful suggestions in the development of the ldea and I would like to thank her for the discussions on the theoretical background. I would also like to thank Dr. Armando Tacchella for his feedback and intuitions on the problem and for providing his solver (*SAT) as a reference, which greatly supported my understanding of the problem. Dr. Roberto Sebstiani provided important suggestions on the intuition of the QBF translation. Also, both Andrew Ladd and Ben McMahan provided useful comments that allows me to see in new ways on optimizing the algorithm.

Contents

Illustrations

Chapter 1

Introduction

Modal logic, the logic of necessity and possibility, of "must be" and "may be", was discussed by several authors in ancient times. Like most work before the modern period, it was non-symbolic and not particularly systematic in approach. The first symbolic and systematic approach to the subject appears to be the work of Lewis, beginning in 1912 and culminating in the book *Symbolic Logic* with Langford [LL59]. Propositional modal logic is obtained from propositional logic by adding a modal connective $\Box$, i.e., if ϕ is a formula, then $\Box\phi$ is also a formula. Intuitively, $\Box\phi$ asserts that ϕ is *necessarily* true. Dually, $\neg\Box\neg\phi$, abbreviated as $\Diamond\phi$, asserts that ϕ is *possibly* true. Modal logic has many applications, due to the fact that the notions of necessity and possibility can be given many concrete interpretations. For example, "necessarily" can mean "according to the laws of physics", or "according to my knowledge", or even "after the program terminates". In the last 20 years modal logic has been applied to numerous areas of computer science, including artificial intelligence [BLMS94, MH69], program verification [CES86, Pra76, Pnu77], hardware verification [Boc82, RS83], database theory [CCF82, Lip77], and distributed computing [BAN88, HM90].

In this thesis, we restrict our attention to the smallest normal modal logic $\mathcal{K}$, and describe a new approach to decide the satisfiability of formulas in this logic. Since modal logic extends propositional logic, the study in modal satisfiability is deeply connected with that of propositional satisfiability. In the past, a variety of approaches to propositional satisfiability have been successfully combined with various approaches to handle modal connectives. For example, tableau based decision procedures for $\mathcal{K}$ are presented in [Lad77, HM92, PSH99]. Such methods are built on top of the propo-

sitional 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 as the propositional engine by treating all modal sub-formulas as propositions and, when a satisfying assignment is found, checking modal sub-formulas for the legality of this assignment [GS00, Tac99].

Recently, we see efforts to unifying the optimizations used in tableau and DPLL based approaches. Introduction of semantical methods like semantic branching and Boolean constraint propagation into tableau allowed DLP to become one of the fastest solvers for $\mathcal{K}$.

Another approach to modal satisfiability, the inverse calculus for $\mathcal{K}$ [Vor01], can be seen as a modalized version of propositional resolution. Non-propositional methods take a different approach to the problem. It has been shown recently that, by embedding $\mathcal{K}$ into first order logic, a first-order theorem prover can be used for deciding modal satisfiability [AGHd00]. 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 [HS00]. 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 [BdV01, HM92, Var97].

The basic algorithms presented here are inspired by the automata-theoretic approach for logics with the tree-model property [Var97]. 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 automata is tested for non-emptiness, i.e., does it accept some tree. In our approach here we, in essence, combine the two steps and we apply the non-emptiness test without explicitly constructing the automaton. As pointed out in [BT01], the inverse method described in [Vor01] can also be viewed as an application of the automata-theoretic approach that avoids an explicit automata construction.

The logic $\mathcal{K}$ is simple enough that the automaton 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 automata that correspond to formulas each state is a *type*, i.e., a set of formulas satisfying some consistency conditions. The algorithms presented here all start from some set of types, and then repeatedly apply a monotone operator until a fixpoint is reached: either they start with the set of *all* types and remove those types with "possibilities" $\Diamond\varphi$ for which no "witness" can be found, or they start with the set of types having no possibilities $\Diamond\varphi$, and add those types whose possibilities are witnessed by a type in the set. The two approaches, top-down and bottom-up, corresponds to the two ways in which non-emptiness can be tested for automata for $\mathcal{K}$: via a greatest fixpoint computation for automata on infinite trees or via a least fixpoint computation for automata on finite trees. The bottom-up approach is closely related to the inverse method described in [Vor01], while the top-down approach is reminiscent of the "type-elimination" method developed for Propositional Dynamic Logic in [Pra80].

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) [Bry86]: BDDs are a compact representation of propositional formulas, 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 sets of types with BDDs, we are able to symbolically construct fixpoint type sets efficiently.

We then study optimization issues for BDD-based $\mathcal{K}$ solvers. First we focus on different representations that can be used for the constructed state set. Types exert a strict consistency requirement on the assignment to related subformulas, which is a major factor in the size of the BDD used to represent the type sets. The normal form used for the type-based approach also makes no distinction between box and

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

diamond operators, increasing the number of necessary witness checks. By using a relaxed consistency representation called particles, we are able to reduce the number of witness checks and simplify the consistency requirement. We also investigate the lean vector approach, in which the constraint is further simplified by removing the variables whose values are implied by the constraints. These approaches reduce the memory consumption of the BDDs and improve performance.

Next, we take advantage of the properties of $\mathcal{K}$, namely the finite-tree-model property. The sets of types/particle vectors implicitly encodes a model for the formula. By considering a layered model instead of a general model, we can modify the bottom-up procedure so each step only checks witness for diamond operators occurring at a specific depth. This approach yields further performance improvements.

Finally, we turn to a preprocessing 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 [DLL62]. Preprocessing has also been shown to be useful for linear-time formulas [SB00, EH00], but has not been explored for $\mathcal{K}$. Our preprocessing is based on a modal pure-literal simplification, which takes advantage of the tree-model property of $\mathcal{K}$. We show that adding preprocessing yield a fairly significant performance improvements, enabling us to handle the hard formulas of TANCS 2000.

This thesis consists of a viability study for our approach. As a measure of competitiveness between different optimizations on BDD-based approaches, we use existing benchmarks of modal formulas, TANCS 98 [HS96] and TANCS 2000 [MD00], and we used *SAT [Tac99] as a reference. A straightforward implementation of our approach did not yield a competitive algorithm, but an optimized implementation did yield a competitive algorithm indicating the viability of our approach.

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 [BCL91, GB94, RAB+95, CCGR00], 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 [Bry86]. While finding an optimal variable order is known to be intractable [THY93], heuristics often work quite well in practice [Rud93]. We focus here on finding a good initial variable order (for large problem instances we have no choice but to invoke dynamic variable ordering, provided by the BDD package), tailored to the application at hand. Our finding is that choosing a good initial variable order does improve performance, but the improvement is rather modest.

To assess the competitiveness of our optimized solver, called $\mathcal{KBDD}$, we benchmark it against both native solver and translation-based solvers.Besides comparing with the standard first-order translation approach, we also developed a translation from $\mathcal{K}$ to QBF (which is of independent interest), and apply QuBE, which is a highly optimized QBF solver [GNT01]. Our results indicate that the BDD-based approach dominates for modally heavy formulas while search-based approaches dominate for propositionally-heavy formulas.

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

Chapter 2

Preliminaries

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

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

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

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

The abbreviated operators can be defined as follows:

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

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

The most important property of $\mathcal{K}$ 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 2.1. *$\mathcal{K}$ has the finite-tree-model property, where for any formula φ, if there is some M, w such that $M, w \models \varphi$, then there exists a finite M' such that exists $w' \in M'$ where $M', w' \models \varphi$.*

Proof. See [BdV01].

In fact, a formula ψ in $\mathcal{K}$ have a finite tree model that is only as deep as its *modal depth*, defined as:

Definition 2.1. *Given a formula ψ, call its set of subformulas $\mathsf{sub}(\psi)$. Take any $\varphi \in \mathsf{sub}(\psi)$, we define $dist(\psi, \varphi)$ as follows:*

- *If $\psi = \varphi$, then $dist(\psi, \varphi) = 0$;*
- *If $\varphi = \varphi' \wedge \varphi''$, $\varphi' \vee \varphi''$, or $\neg\varphi'$, then $dist(\psi, \varphi') = dist(\psi, \varphi'') = dist(\psi, \varphi)$;*
- *If $\varphi = \Box\varphi'$ or $\Diamond\varphi'$, then $dist(\psi, \varphi') = dist(\psi, \varphi) + 1$.*

The modal depth $md(\psi)$ is defined as $max_{\varphi \in \mathsf{sub}(\psi)}(dist(\psi, \varphi))$.

We will use this property for certain optimizations in our algorithm.

We restrict our attention to formulas in a certain normal form to simplify considerations on the form of a formula. A formula ψ of $\mathcal{K}$ is said to be in *box normal form* (BNF) if all its sub-formulas are of the form $\varphi \wedge \varphi'$, $\varphi \vee \varphi'$, $\Box\varphi$, $\neg\Box\varphi$, q, or $\neg q$ where $q \in AP(\psi)$. All $\mathcal{K}$ formulas can be obviously converted into BNF without blow up by pushing negation inwards and, if not stated otherwise, we assume all formulas to

be in BNF. The *closure* of a formula $\mathsf{cl}(\psi)$ is defined as the smallest set such that, for all sub-formula φ of ψ, if φ is not $\neg\varphi'$, then $\{\varphi, \neg\varphi\} \subseteq \mathsf{cl}(\psi)$.

Our algorithms will work on *types*, i.e., sets of (sub)formulas that are consistent w.r.t. the Boolean operators, and where (negated) box formulas are treated as atoms. A set of formulas $a \subseteq \mathsf{cl}(\psi)$ 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 the relation $\Delta \subseteq A \times A$ as follows:

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

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

We should compare the structure we built against the *canonical model* of $\mathcal{K}$.

Definition 2.2. *[BdV01]The canonical model for $\mathcal{K}$ is defined as $K = \langle AP, W, R, L\rangle$ where:*

- *W is the set of all maximal consistent set of $\mathcal{K}$ formulas.*
- *R is defined as $R(x, y)$ iff for all $\varphi \in y$, $\Diamond\varphi \in x$. (Note together with consistency, this says for all $\Box\varphi \in x$, $\varphi \in y$.*
- *L is the normal interpretation, where for $p \in AP$, $L(W)(p)$ iff $p \in W$.*

We know that we can filter K against $\mathsf{cl}(\psi)$ to produce a model K_ψ for the purpose of checking whether there are any state w in K_ψ where $\psi \in w$. We would show the K_A we build will have the same property as K_ψ.[1]

Thus we would like to prove that, for all $\varphi \in \mathsf{cl}(\psi)$:

[1] In fact, their structure is very similar.

Claim 2.1. $K_A, a \models \varphi$ *iff* $\varphi \in a$.

We can see this is clearly true for propositional φ by requirement on types and true for $\varphi = \Box\varphi'$ by construction of Δ. The only case that needs special consideration is the case $\neg\Box\varphi \in a$: it might be the case that $\varphi \in b$ for all b with $\Delta(a, b)$. In the following chapter, we will show that our construction satisfies claim 2.1.

Chapter 3

Our Algorithms

The two algorithms presented here take a certain "initial" set of types and apply repeatedly a monotone operator to it. If this application reaches a fixpoint, we can show that it yields a set of types where the above construction yields indeed a Kripke structure that satisfies the claim 2.1, i.e., all negated box formulas are indeed "witnessed" by some $b \in A$. This Kripke structure is then a model of ψ iff $\psi \in a$ for some $a \in A$.

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

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:

$X \Leftarrow Initial(\psi)$
repeat
  $X' \Leftarrow X$
  $X \Leftarrow Iterate(X')$
until $X = X'$
if exists $x \in X$ such that $\psi \in x$ **then**
  return "ψ is satisfiable"

else

return "ψ is not satisfiable"

end if

Since this algorithm will work on a fixed set of types and use a monotone $Iterate(\cdot)$ operator, it obviously terminates. In fact, we can show that it will terminate in $md(\psi)+1$ iterations. It remains to define $Initial(\psi)$ and $Iterate(\cdot)$.

3.1 Top-Down Approach

The top-down approach is closely related to the type elimination approach which is, in general, used for more complex modal logics, see, e.g., Section 6 of [HM92]. For the algorithm pursuing the top-down approach, the functions $Initial(\psi)$ and $Iterate(\cdot)$ are defined as follows:

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

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

Theorem 3.1. *The top-down algorithm decides satisfiability of $\mathcal{K}$ formulas.*

Proof. Let A be the set of types that is the fixpoint of the top-down algorithm, i.e., $Iterate(A) = A$. We use A^0 for $Initial(\psi)$ and A^i for the set of types after i iterations.

Lemma 3.1. ***(Soundness)*** *For each type $a \in A$ and formula $\varphi \in \mathsf{cl}(\psi)$, we have $K_A, a \models \varphi$ iff $\varphi \in a$.*

Proof. By induction on the structure of formulas.

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

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

- If $\varphi = \Box\varphi' \in a$, the definition of Δ implies that $\varphi' \in a'$ for all a' with $\Delta(a, a')$, and by induction, $K_A, a' \models \varphi'$. Hence $K_A, a \models \Box\varphi'$.

- If $\varphi = \neg\Box\varphi' \in a$, then $a \notin \mathsf{bad}(A)$ because $Iterate(A) = A$, and thus there exists $b \in A$ with $\Delta(a, b)$ and $\varphi' \notin b$. By induction, $K_A, b \models \neg\varphi'$, and thus $K_A, a \models \neg\Box\varphi'$. □

Lemma 3.2. ***(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 every world $w \in W$, we define a type $a(w) = \{\varrho \in \mathsf{cl}(\psi) : K, w \models \varrho\}$. Next, 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 it can be shown by induction on the number of iterations that $A(W) \subseteq A$. 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 algorithm pursuing the top-down approachstarts with all valid types, and repeatly removes unwitnessed types. In contrast, the algorithm persuing the bottom-up approach starts with a small set of types (i.e., those without

negated box formulas), and repeatedly adds those types whose negated box formulas are witnessed in the current set. More precisely, for the bottom-up approach, the functions $\mathit{Initial}(\psi)$ and $\mathit{Iterate}(\cdot)$ are defined as follows:

- $\mathit{Initial}(\psi)$ 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 formulas in $\mathsf{cl}(\psi)$. More precisely,

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

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

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

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

Theorem 3.2. *The bottom-up algorithm decides satisfiability of $\mathcal{K}$ formulas.*

Proof. As in the proof of Theorem 3.1, we use A for the fixpoint of the bottom-up algorithm, A^0 for $\mathit{Initial}(\psi)$, and A^i for the set of types after i iterations.

Lemma 3.3. ***(Soundness)*** *For each type $a \in A$ and formula $\varphi \in \mathsf{cl}(\psi)$, we have $K_A, a \models \varphi$ iff $\varphi \in a$.*

Proof. Again by induction on the structure of formulas.

- If $\varphi = q$, then $K_A, a \models \varphi$ iff $\varphi \in a$ by construction of L.
- If $\varphi = \neg q$, $\varphi' \wedge \varphi''$, or $\varphi' \vee \varphi''$, the claim follows immediately by induction and the definition of types.
- If $\varphi = \Box\varphi' \in a$, then by definition of Δ and induction, $K_A, a \models \varphi$.

- If $\varphi = \neg\Box\varphi' \in a$, then there exists by construction of A some $b \in A$ with $\neg\varphi' \in b$ and $\Delta(a,b)$. Thus, by induction, $K_A, a \models \varphi$. □

Lemma 3.4. ***(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 $\mathcal{K}$ has the finite-tree-model property (see, e.g. [HM92]), i.e., each satisfiable $\mathcal{K}$ 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 3.2. 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 $a(w) \in \mathsf{supp}(A^{i-1}) \subseteq A^i$. □

Chapter 4

Optimizations

The decision procedures described above handles a formula in three steps. First, the formula is converted into box normal form. Then, a set of bit vectors representing types is generated. 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

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

A $\mathcal{K}$ formula ψ is said to be in *negation normal form* (NNF) if all its sub-formulas are of the form $\varphi \wedge \varphi'$, $\varphi \vee \varphi'$, $\Box\varphi$, $\Diamond\varphi$, q, or $\neg q$ where $q \in AP(\psi)$. When needed, we assume the formula ψ is already in NNF. We use $\mathsf{sub}(\psi)$ to represent the set of sub-formulas of $NNF(\psi)$. All $\mathcal{K}$ formulas can be converted into negation normal form without blow up by pushing negation inwards.

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

- If $\varphi = \neg\varphi'$, then $\varphi \in p$ implies $\varphi' \notin p$.

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

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

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

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

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

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

Theorem 4.1. *The top-down algorithm for particles decides satisfiability of $\mathcal{K}$ formulas.*

Proof.

Lemma 4.1. ***(Soundness)*** *For each type $p \in P$ and formula $\varphi \in \mathsf{sub}(\psi)$, if $\varphi \in p$, then $K_P, p \models \varphi$.*

Proof. The same proof as lemma 3.1 applies, except for the $\neg\Box\varphi'$ part, (which does not exist for particles), and for the $\Diamond$ operator:

- If $\varphi = \Diamond\varphi' \in p$, then $p \notin \mathsf{bad}(P)$ because $Iterate(P) = P$, and thus 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 4.2. ***(Completeness)*** *For all $\varphi \in \mathsf{sub}(\psi)$, if φ is satisfiable, then there exists some $p \in P$ where $\varphi \in p$.*

Proof. See lemma 3.2. A analogous proof can be constructed by taking a model K for φ and generate a particle set $P(W)$ from the states of K. To show $P(W) \subseteq P$, we can follow the same proof by contradiction scheme:

- $P(W) \subset P^0$ since P^0 contains *all* particles $p \subseteq \mathsf{sub}(\psi)$.
- Let $P(W) \subseteq P^i$ and assume that $P(W) \not\subseteq P^{i+1}$. 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 $K, w \models \Diamond\varphi$.

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

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

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

Theorem 4.2. *The bottom-up algorithm for particles decides satisfiability of $\mathcal{K}$ formulas.*

Proof.

Lemma 4.3. ***(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. Analogous to 3.3 and lemma 4.1.

Lemma 4.4. ***(Completeness)*** *For all $\varphi \in \mathsf{sub}(\psi)$, if φ is satisfiable, then there exists some $p \in P$ with $\varphi \in p$.*

Proof. Analogous to 3.4 and lemma 4.2.

While encoding particle sets by BDDs may require more BDD variables, we still might see a reduction in BDD size, because particles requires fewer constraints than types.[1] Beside a possible reduction in the size required to encode a bit-vector representation of particle sets, the particle-based approaches also can improve running time. From the definition of bad and supp, we can see that, in the type-based approaches, for each fixpoint iteration, the number of constraints that needs to be applied to the state set in each iteration is equal to the number of $\Box$ operators, which is equal to the total number of all modal operators in the original formula. On the other hand, in particle-based approaches, the number of constraints only have to be equal to the number of $\Diamond$ operators in the NNF form of the formula, which is smaller.

4.2 Lean Approaches

This optimization is also motivated by the idea to compress the size of the bit vector representing a type by omitting redundant information. To this purpose, we first define a set of "non-redundant" sub-formulas $\mathsf{atom}(\psi)$ as the set of those formulas 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 the definition of types, each type $a \subseteq \mathsf{cl}(\psi)$, corresponds one-to-one to a *lean type* $\mathsf{lean}(a) := a \cap \mathsf{atom}(\psi)$. So storing types in lean form is equivalent to storing them in full form. Thus the following theorem is trivial.

Theorem 4.3. *The top-down/bottom-up algorithms for lean atoms decide satisfiability for $\mathcal{K}$.*

Proof. Take any atom set A during the algorithm and its lean version A'. Define $\mathsf{full}(A') = \{a \mid a \cap \mathsf{atom}(\psi) \in A' \text{ and } a \text{ is an atom}\}$, we can see $\mathsf{full}(A') = A$. So given the full atom algorithms are sound and complete, so are the lean atom algorithms.

[1] 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.

Analogously, we can define a lean representation for particles. First, we define the relevant sub-formulas $\mathsf{part}(\psi)$ as follows: For $\varphi \in \mathsf{sub}(\psi)$, if φ is $\Diamond\varphi'$, $\Box\varphi'$, q, or $\neg q$, then φ is in $\mathsf{part}(\psi)$. For a full particle $p \subseteq \mathsf{sub}(\psi)$, we define the corresponding *lean particle* $\mathsf{lean}(p)$ as follows: $\mathsf{lean}(p) = p \cap \mathsf{part}(\psi)$. Because the (first) condition on particles is more relaxed than that of atoms, a lean particle does not correspond to a single full particle, but can represent several full particles.

Theorem 4.4. *The top-down/bottom-up algorithms for lean particles decide satisfiability for $\mathcal{K}$.*

Proof. Given a particle set P used in the full version of the algorithm and the its lean correspondent P', there does not exist a bijection from P to P'.[2] But the particle set P we construct in our algorithms have additional properties. In particular, for if $p \in P$, for all consistent $q \subseteq \mathsf{cl}(\psi)$ such that $p' = q'$, $q \in P$. So P' fully characterizes P. This is by definition of $Initial(\psi)$ and $Iterate(P)$ since these functions only apply constraints on subformulas that are member of $\mathsf{part}(\psi)$. So at any step in the algorithm, take P' and build $\mathsf{full}(P') = \{p \subseteq \mathsf{sub}(\psi) \mid \mathsf{lean}(p) \in P' \wedge p \text{ is consistent}\}$, $\mathsf{full}(P') = P$. So given the full particle algorithms are sound and complete, so are the lean particle algorithms.

Although lean approaches can possibly reduce the size required for representing worlds, we have to pay for these savings since computing bad and supp using lean types and particles can be more complicated.

4.3 Level-based evaluation

As already mentioned, $\mathcal{K}$ has the finite-tree-model property, i.e., each satisfiable formula ψ of $\mathcal{K}$ has a finite tree model of depth bounded by the depth $md(\psi)$ of nested modal operators in ψ. Here, we take advantage of this property and, instead of representing a complete model using a set of particles or types, we represent each layer

[2] In fact, $\mid P \mid \neq \mid P' \mid$

(i.e., all worlds being at the same distance from the root node) in the model using a separate set (For a level-based approach in the context of the first-order approach to $\mathcal{K}$, see [AGHd00]). Since only a subset of all sub-formulas appears in one layer, the representation can be more compact. We only present the optimization for the approach using (full) types. The particle approach and the lean approaches can be constructed analogously. For $0 \leq i \leq md(\psi)$, we write

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

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

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

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

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

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

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

The algorithm for level-based evaluation works as follows:

$d \Leftarrow md(\psi)$
$X_d \Leftarrow Initial_d(\psi)$
for $i = d - 1$ downto 0 **do**
  $X_i \Leftarrow Iterate(X_{i+1}, i)$
end for
if exists $x \in X_0$ where $\psi \in x$ **then**

ψ is satisfiable.

else

ψ 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 can be said to start 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:

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

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

Definition 4.1. *Since in the level-based evaluation algorithm, we use each assignment set to represent only assignments to* $\mathsf{cl}_i(\psi)$*, the assignments are not valid types, but they are consistent enough for what formulas occuring at level* i*. We call an assignment a consistent for level* i *if it meets the requirement on types for formulas in* $\mathsf{cl}_i(\psi)$*. The same definition can be made for assignment sets, and if the set* A_i *is labeled with a level, we take a shortcut and use it as if it is a set of types if it is consistent for level* i*.*

Theorem 4.5. *The level based algorithm for atom assignments 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 $d = md(\psi)$.

Lemma 4.5. ***(Soundness)*** *For all $\varphi \in \mathsf{cl}_i(\psi)$, and $a \in A_i$, we have $K_A, w_{a,i} \models \varphi$ iff $\varphi \in a$.*

Proof. Induction on the structure of the formula.

- $\varphi = q$: Then $K_A, w_{a,i} \models \varphi$ by construction of L.
- Otherwise, assume inductively the claim holds for all subformulas of φ.
  - $\varphi = \varphi' \wedge \varphi''$, $\varphi' \vee \varphi''$, $\neg\varphi'$: We here prove for the $\wedge$ case. By definition of cl_i, $\varphi' \in \mathsf{cl}_i(\psi)$, $\varphi'' \in \mathsf{cl}_i(\psi)$. By consistency of A_i, for all $a \in A_i$, if $\varphi \in a$, we have $\varphi' \in a$, $\varphi'' \in a$. By inductive hypothesis, $K_A, w_{a,i} \models \varphi'$, $K_A, w_{a,i} \models \varphi''$. So $K_A, w_{a,i} \models \varphi$ by semantics of $\mathcal{K}$. The same argument can be made of $\vee$ and $\neg$.
  - $\varphi = \Box\varphi'$: If $\varphi \in a$, then by construction of R, we have: (1) For all $a' \in A_{i+1}$ that $R(w_{a,i}, w_{a',i+1})$, we have $\varphi' \in a'$. (2) There are no $j \neq i+1$ where $R(w_{a,i}, w_{b,j})$ for any assignment b. By definition of cl_i, $\varphi' \in \mathsf{cl}_{i+1}(\psi)$. So by the inductive hypothesis, $K_A, w_{a',i+1} \models \varphi'$ for all such a'. So by semantics of $\mathcal{K}$, $K_A, w_{a,i} \models \varphi$. If $\varphi \notin a$, By definition of *Iterate*, every assignment $a \in A_i$ is consistent for $\mathsf{cl}_i(\psi)$. So $\neg\varphi \in a$. By definition of *Iterate*, exists $a' \in A_{i+1}$ such that $\neg\varphi' \in a'$ and for all $\Box\varrho \in a$, we know that $\Box\varrho \in \mathsf{cl}_i(\psi)$ so $\varrho \in a'$. So by definition of R, $R(w_{a,i}, w_{a',i+1})$ holds. By induction hypothesis, $K_A, w_{a',i+1} \models \neg\varphi'$. So $K_A, w_{a,i} \models \neg\varphi$.

Corollary 4.1. *If exists $a \in A_0$ such that $\psi \in a$, then $K_A, w_{a,0} \models \psi$.*

Lemma 4.6. ***(Completeness)*** *For $\varphi \in \mathsf{cl}_i(\psi)$, if φ is satisfiable, then there is an atom assignment $a \in A_i$ where $\varphi \in a$.*

Proof. We know from [HM92] that if φ is satisfiable, it have a finite tree model of depth $d_\varphi = md(\varphi)$. We also know from definition of md and cl_i that $i + d_\varphi \leq d_\psi = md(\psi)$. Given that φ is satisfiable, take a tree model K_φ with depth d where

$K_\varphi, w_\varphi \models \varphi$ and $d \leq d_\psi - i$. Since $i + d_\varphi \leq d_\psi$, such a model exists. Since K_φ is a tree model, each state $w \in W$ only occurs at a unique distance from the root w_φ. We partition the state set W of K_φ into $\{W_0, W_1, \ldots, W_d\}$, in which a state $w \in W_j$ occurs at distance j from the root. For each $w \in W_j$ we define an atom assignment $a(w) = \{\varrho \in \mathsf{cl}\, i + j\psi) \mid K_\varphi, w \models \varrho\}$. We define $A(W_j) = \{a(w) \mid w \in W_j\}$ as a set of assignments to formulas in $\mathsf{cl}_{i+j}(\psi)$. We now show that $A(W_j) \subseteq A_{i+j}$.

We prove by induction on depth j:

- $j = d$: We know every state $w \in W_d$ is a terminal state in K_φ. It follows that for all $\varrho = \neg\Box\varrho'$, we have $K_\varphi, w \not\models \varrho$. So $\varrho \notin a(w)$. And by semantics of $\mathcal{K}$, $a(w)$ is consistent. Since A_{i+j} contains all witnessed assignments to $\mathsf{cl}_{i+j}(\psi)$ by definition, we have $a(w) \in A_{i+j}$.

- $j < d$: Assume inductively that the claim holds for $j' = j + 1$. For a state $w \in W_j$, $a(w)$ is consistent by the semantics of K. For all $\varrho = \neg\Box\varrho' \in \mathsf{cl}_{i+j}(\psi)$, we have $\neg\varrho' \in \mathsf{cl}_{i+j'}(\psi)$. If $\varrho \in a(w)$, we have $K_\varphi, w \models \neg\Box\varrho'$. So exists $w' \in W_{j'}$ where $R(w, w')$ and $K_\varphi, w' \models \neg\varrho'$. So $\neg\varrho' \in a(w')$. By inductive hypothesis, $a(w') \in A_{i+j'}$. By the semantics of $\mathcal{K}$, for all $\Box\varrho'' \in \mathsf{cl}_{i+j}(\psi)$, where $\Box\varrho'' \in a(w)$, we have $\varrho'' \in a(w')$. So $a(w) \in A_{i+j}$ by definition of $Iterate$.

Since $w_\varphi \in W_0$, we have $a(w_\varphi) \in A_i$, and because $\varphi \in \mathsf{cl}_i(\psi)$, we have $\varphi \in a(w_\varphi)$ by construction of a.

Corollary 4.2. *If ψ is satisfiable, then exists $a \in A_0$ such that $\psi \in a$.*

The same algorithms can be defined for particle based approaches, by defining an analogous $\mathsf{sub}_i(\psi)$, and using

- $Initial_i(\psi) = \{p \subseteq \mathsf{sub}_i(\psi) \mid p \text{ is a particle}\}$.

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

Theorem 4.6. *The level based algorithm for particle assignments is sound and complete.*

Proof. We write the sequence of assignment sets constructed by the level based algorithm as $P = \langle P_0, P_1, \ldots, P_d \rangle$ where $d = md(\psi)$.

Lemma 4.7. ***(Soundness)*** *For all $\varphi \in \mathsf{sub}_i(\psi)$, and $p \in P_i$, if $\varphi \in p$, then $K_P, w_{p,i} \models \varphi$.*

Proof. Induction on the structure of the formula.

- $\varphi = q, \neg q$: Then $K_P, w_{p,i} \models \varphi$ by construction of L.
- Otherwise, assume inductively the claim holds for all subformulas of φ.
  - $\varphi = \varphi' \wedge \varphi''$, $\varphi' \vee \varphi''$: We here prove for the $\wedge$ case. By definition of sub_i, $\varphi' \in \mathsf{sub}_i(\psi)$, $\varphi'' \in \mathsf{sub}_i(\psi)$. By consistency of P_i, for all $p \in P_i$, if $\varphi \in p$, we have $\varphi' \in p$, $\varphi'' \in p$. By inductive hypothesis, $K_P, w_{p,i} \models \varphi'$, $K_P, w_{p,i} \models \varphi''$. So $K_P, w_{p,i} \models \varphi$ by semantics of $\mathcal{K}$. The same argument can be made for $\vee$.
  - $\varphi = \Box\varphi'$: By construction of R, for every $w_{p',j}$ where $R(w_{p,i}, w_{p',j})$, we know that $j = i + 1$ and $\varphi' \in p'$. So by semantics of $\Box$, we get $K_P, w_{p,i} \models \varphi$.
  - $\varphi = \Diamond\varphi'$: Assume $K_P, w_{p,i} \not\models \varphi$. Then $K_P, w_{p,i} \models \Box\neg\varphi'$. So for all $w_{p',i+1}$ where $R(w_{p,i}, w_{p',i+1})$, $K_P, w_{p',i+1} \models \neg\varphi'$ by semantics of $\mathcal{K}$. By inductive hypothesis, $\varphi' \notin p'$. So $p \notin \mathit{Iterate}(P_{i+1}, i)$ by definition of $\mathit{Iterate}$ which is a contradiction. So $K_P, w_{p,i} \models \varphi$.

Lemma 4.8. ***(Completeness)*** *For $\varphi \in \mathsf{sub}_i(\psi)$, if φ is satisfiable, then there is a particle assignment $p \in P_i$ where $\varphi \in p$.*

Proof. Given that φ is satisfiable, take a tree model K_φ rooted at w_φ with depth d where $K_\varphi, w_\varphi \models \varphi$ and $d \leq d_\psi - i$. Since $i + d_\varphi \leq d_\psi$, such a model exists. Since K_φ is a tree model, each state $w \in W$ only occurs at a unique distance from the root w_φ.

We partition the state set W of K_φ into $\{W_0, W_1, \ldots, W_d\}$, in which a state $w \in W_j$ occurs at distance j from the root. For each $w \in W_j$ we define a particle assignment $p(w) = \{\varrho \in \mathsf{sub}_{i+j}\,\psi) \mid K_\varphi, w \models \varrho\}$. We define $P(W_j) = \{p(w) \mid w \in W_j\}$ as a set of assignments to formulas in $\mathsf{sub}_{i+j}(\psi)$. We now show that $P(W_j) \subseteq P_{i+j}$.

We prove by induction on depth j:

- $j = d$: We know every state $w \in W_d$ is a terminal state in K_φ. It follows that for all $\varrho = \Diamond\varrho'$, we have $K_\varphi, w \not\models \varrho$. So $\varrho \notin p(w)$. And by semantics of $\mathcal{K}$, $p(w)$ is consistent. Since P_{i+j} contains all witnessed assignments to $\mathsf{sub}_{i+j}(\psi)$ by definition, we have $p(w) \in P_{i+j}$.

- $j < d$: Assume inductively that the claim holds for $j' = j + 1$. For a state $w \in W_j$, $p(w)$ is consistent by the semantics of K. For all $\varrho = \Diamond\varrho' \in \mathsf{sub}_{i+j}(\psi)$, we have $\varrho' \in \mathsf{sub}_{i+j'}(\psi)$. If $\varrho \in p(w)$, we have $K_\varphi, w \models \Diamond\varrho'$. So exists $w' \in W_{j'}$ where $R(w, w')$ and $K_\varphi, w' \models \varrho'$. So $\varrho' \in p(w')$. By inductive hypothesis, $p(w') \in P_{i+j'}$. By the semantics of $\mathcal{K}$, for all $\Box\varrho'' \in \mathsf{sub}_{i+j}(\psi)$, where $\Box\varrho'' \in p(w)$, we have $\varrho'' \in p(w')$. So $p(w) \in P_{i+j}$ by definition of $Iterate$.

Since $w_\varphi \in W_0$, we have $p(w_\varphi) \in P_i$, and because $\varphi \in \mathsf{sub}_i(\psi)$, we have $\varphi \in p(w_\varphi)$ by construction of p.

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 [DLL62], which can be applied both in a preprocessing step as well as dynamically, following the unit-propagation step. Preprocessing has also been shown to be useful for linear-time formulas [SB00, EH00], but has not been systematically

explored for $\mathcal{K}$. Our preprocessing is based on a modal pure-literal simplification, which takes advantage of the layered-model property of $\mathcal{K}$.

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

1. Equivalence preserving: This requires that the simplified formulas φ' is logically equivalent to the input formula φ. Unit propagation is an example of an equivalence-preserving transformation. Such a transformation is used in model checking [SB00, EH00], where the semantics of the formula needs to be preserved. An equivalence-preserving rule can be applied to subformulas.

2. Satisfiability preserving: This requires only that φ' is satisfiable iff φ is satisfiable. Pure-literal simplification is an example of a satisfiability-preserving transformation. Such transformations allow for more aggressive simplification, but cannot be applied to subformulas. Note that such a transformation can be used for satisfiability solving but not for model checking.

Our preprocessing was designed to reduce the number of BDD operations called by $\mathcal{KBDD}$, though its correctness is algorithm independent. (We found that such preprocessing was beneficial for DLP, a tableau-based modal solver, as well as QuBE, a DPLL-based solver but not for MSPASS, a resolution-based solver.) 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 $\Diamond$ subformulas.

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

Table 4.1 : Simplification rewriting rules for $\mathcal{K}$

4.4.0.1 Rewrite rules

Our preprocessing includes rewriting according to a collection of rewrite rules (see Table 4.1). Although the rules can be applied in both directions, we apply only the direction that reduces the size of the formula. It is easy to see that the rules are equivalence or satisfiability preserving. These rules by themselves are only modestly effective for $\mathcal{K}$ formulas; they do become quite effective, however, when implemented in combination with pure-literal simplification, described below. These rules allows us to propagate the effects of pure-literal simplification by removing redundant portions of the formula after pure-literal simplification. This usually allows more pure literals to be found and can greatly reduce the size of the formula.

4.4.0.2 Pure-literal simplification

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

Definition 4.2. *Given a set S of (propositional or modal) formulas in NNF, we define* $\mathsf{lit}(S) = \{l \mid l \in S \text{ and } l \text{ is } q \text{ or } \neg q, \text{ where } q \in \Phi\}$ *as the set of literals of S. The set* $\mathsf{pure}(S)$ *of defined as the set of literals that have a pure-polarity occurrence in S, i.e., $l \in \mathsf{pure}(S)$ iff $l \in \mathsf{lit}(S)$ and $\neg l \notin \mathsf{lit}(S)$.*

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

Definition 4.3. *For a formula ψ in NNF, we define* $\mathsf{pure}(\psi) = \mathsf{pure}(\mathsf{sub}(\psi))$ *as the set of* globally pure literals *of ψ, and define the corresponding formula after pure literal simplification as $\psi'_G = \psi[\mathsf{pure}(\psi)/\mathsf{true}]$.*

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

Definition 4.4. *For ψ in NNF, we define* level-pure literals *by* $\mathsf{pure}_i(\psi) = \mathsf{pure}(\mathsf{sub}_i(\psi))$*, for $0 \leq i \leq md(\psi)$. The substitution used for level-pure literals needs to take into consideration that $l \in \mathsf{pure}_i(\psi)$ is only pure at modal depth i, so we let $\psi[\mathsf{pure}_i(\psi)/\mathsf{true}]_i$ be the substitution with* true *of all level-pure literals l that occur at distance i from ψ. The result of the pure-literal simplification is $\psi'_L = \psi[\mathsf{pure}_0(\psi)/\mathsf{true}]_0 \ldots [\mathsf{pure}_{md(\psi)}(\psi)/\mathsf{true}]_{md(\psi)}$.*

Remark 4.1. *It is possible to push this idea of "separation" further. Because each world in the model only needs to satisfy a subset of* $\mathsf{sub}(\psi)$*, the possible subsets can be constructed to determine which of the literals can be pre-assigned* true*. For example, it is possible to construct sets of subformulas that can occur together in a tableau and define pure literals based on such sets. We did not find that the performance benefit justified the implementation overhead for this extension.*

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

Theorem 4.7. *Both global and level pure-literal simplifications are satisfiability preserving. That is, for a formula ψ, we have that ψ is satisfiable iff ψ'_G(or ψ'_L) is satisfiable.*

Proof. We write ψ' instead of ψ'_G or ψ'_L, when the formula used is clear from the context. Without loss of generality, we assume that only one literal l is substituted. Since other pure literals for ψ are still pure with respect to ψ' under both definitions, the general case can be shown by induction on the number of literals.

The completeness part of the claim is easy. It is known that the $\Box$ and $\Diamond$ operators are *monotone* [BdV01]. More formally, if ψ is a formula in NNF, α is a subformula occurrence of ψ and β is another formula that is logically implied by α, then $\psi[\alpha/\beta]$ is logically implied by ψ. It follows that ψ' is logically implied by ψ. In particular, if ψ is satisfiable, then ψ' is satisfiable.

In the following, we take $K = \langle \Phi, W, R, L \rangle$ and $K' = \langle \Phi, W, R, L' \rangle$ to be finite tree Kripke structures of depth $md(\psi)$ with the same underlying frame, and $w_0 \in W$ to be the root of the tree, where we want ψ and ψ' to hold.

The soundness proof for pure-literal simplification depends whether we use globally pure or level-pure literals.

- *Globally pure literals*: Assume $K', w_0 \models \psi'$. Note that l does not occur in ψ'_G, so we can assume that L does not define a truth value for l. We construct K from K' by taking L to be an extension of L' such that $L(w)(l) = \mathsf{true}$ for every $w \in W$. We claim that for every state $w \in W$ and every formula $\varphi \in \mathsf{sub}(\psi)$, we have that $K', w \models \varphi[l/\mathsf{true}]$ implies $K, w \models \varphi$. We prove the claim by induction on the structure of the formula. If φ is a propositional literal, the property holds because either $\varphi = l$, in which case $K, w \models l$ by construction, or

φ is a literal l' such that $AP(l') \neq AP(l)$, in which case $L(w)$ and $L'(w)$ agree on l', so $K', w \models l'$ implies $K, w \models l'$. For the induction, we show only the case when $\varphi = \Box\varphi'$. Given $K', w \models \varphi[l/\mathsf{true}]$, we have that $K', w' \models \varphi'[l/\mathsf{true}]$ holds for all w' such that $R(w, w')$. By the inductive hypothesis, $K, w' \models \varphi'$ for all such w' as well. So $K, w \models \varphi$ holds. Thus $K', w_0 \models \psi'$ implies $K, w_0 \models \psi$.

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

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

Chapter 5

Implementation

5.1 Base Algorithm

We use Binary Decision Diagrams (BDDs) [Bry86, And98] to represent sets of types. BDDs, or more precisely, Reduced Ordered Binary Decision Diagrams (ROBDDs), are obtained from binary decision trees by following a fixed variable splitting order and by merging nodes that have identical child-diagrams. BDDs provide a canonical form of representation for Boolean functions. Experience has shown that BDDs often provide a very compact representation for very large Boolean functions. Consequently, over the last decade, BDDs have had a dramatic impact in the areas of synthesis, testing, and verification of digital systems [BBG$^+$94, BCM$^+$92].

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

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

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

and use m for $|\mathsf{cl}_+(\psi)| = |\mathsf{cl}(\psi)|/2$. 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 a set[1] $a \subseteq \mathsf{cl}(\psi)$ where $\varphi_i \in a$ iff $a_i = 1$.

[1] 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 $Consistent_\psi$ as the characteristic predicate for types: $Consistent_\psi(\vec{a}) = \bigwedge_{1\leq i\leq m} Cons_i(\vec{a})$, where $Cons_i(\vec{a})$ is defined as follows:

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

where $a' = a_\ell$ if $\varphi' = \varphi_\ell \in \mathsf{cl}_+(\psi)$, and $a' = \neg a_\ell$ if $\varphi' = \neg\varphi_\ell$ for $\varphi_\ell \in \mathsf{cl}_+(\psi)$.

From this, the implementation of $Initial$ is fairly straight forward: For the top-down algorithm,

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

and for the bottom-up algorithm,

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

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 formulas. 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. Both the top-down and the bottom-up algorithm can be defined using the predicates χ_A, Δ, $\Diamond_{j,i}$, and $\Box_{j,i}$.

The predicate bad is true on those types that contain a negated box formula that is not witnessed in the current set of types. We can define a predicate bad_i for each negated box formula $\varphi_i = \neg\Box\varphi_j$ that can be used to remove unwitnessed bit vectors 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 *Iterate* function can be written as:

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

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

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

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:

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

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 $\chi_{X \setminus \mathsf{bad}(X)}$ 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 [BCM+92] and it has been the subject of extensive research (cf. [BCL91, GB94, RAB+95, CCGR00]), since it can be a quite time and space consuming operation. Various optimizations can be applied to the pre-image computation to reduce the time and space requirements. A method of choice is that of *conjunctive partitioning* combined with *early quantification*. The idea is to avoid building a monolithic BDD for the relation Δ, since this BDD can be quite large. Rather, we take advantage of the fact that Δ is defined as a conjunction of simple conditions. Thus, to compute the pre-image we have to evaluate a quantified Boolean formula of the form $(\exists y_1) \ldots (\exists y_n)(c_1 \wedge \ldots \wedge c_m)$, where the c_i's are Boolean formulas. Suppose, however, that a variable y_j does not occur in the clauses $c_{i+1}, \ldots, c_m$. Then the formula can be rewritten as

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

This enables us to apply existential quantification to smaller BDDs.

Of course, there are many ways in which one can cluster and re-order the c_i's. One of which we used is the methodology developed in [RAB+95], called the "IWLS 95" methodology, to compute pre-images. We also have tried other clustering mechanisms,

namely the "bucket-elimination" approach used in [SV01]. Given a set of conjunctive components $c_1 \ldots c_n$, we get the variable support set for each component as $Y_1 \ldots Y_n$. Then, a graph of interference of variables is constructed so every vertex represents a variable, and there is an edge between variables y_i and y_j if y_i and y_j occurs together in some Y_k. We conduct an "maximum cardinality ordering" of the variables, so y_1 is the variable that occurs with the maximal number of edges, and y_i have the maximum number of edges into previously chosen variables. Given such an 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, although the performance improvements are minimal.

5.2 Optimizations

5.2.1 Particles

Encoding of the particle based approach with BDDs is analogous to the encoding of the atom based approach. Since the consistency requirement for particles is more relaxed then that of atoms, each subformula in $\mathsf{sub}(\psi)$ needs to be assigned to a variable. So given $\mathsf{sub}(\psi) = \{\varphi_1, ...\varphi_n\}$, a vector $\vec{p} = \langle p_1, ...p_m\rangle \in \{0,1\}^n$ represents a set $p \subseteq \mathsf{sub}(\psi)$ with $\varphi_i \in p$ iff $p_i = 1$.

Then, for particles, $Consistent_\psi(\vec{p}) = \wedge_{1\leq i\leq n} Cons_i(\vec{p})$, where $Cons_i(\vec{p})$ is defined as follows:

- If φ_i is neither of the form $\varphi_j \wedge \varphi_k$ nor $\varphi_j \vee \varphi_k$, then $Cons_i(\vec{p}) = 1$,
- If $\varphi_i = \varphi_j \wedge \varphi_k$, then $Cons_i(\vec{p}) = (p_i \rightarrow (p_j \wedge p_k))$,
- If $\varphi_i = \varphi_j \vee \varphi_k$, then $Cons_i(\vec{p}) = (p_i \rightarrow (p_j \vee p_k))$,
- If $\varphi_i = \neg\varphi_j$, then $Cons_i(\vec{p}) = \neg(p_i \wedge p_j)$.

We also need to 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 $\varphi_i = \Box\varphi'$.
- $\Box_{2,i}(\vec{y})$ is true iff $\varphi_i = \Box\varphi_j$ and $y_j = 1$.

5.2.2 Lean vectors

Lean approaches have much more relaxed consistency predicates at the cost of bigger witness/support predicates. For lean approaches, Only the $Cons_i(\vec{x})$ that is related to those φ_i in $\mathsf{atom}(\psi)$ (or $\mathsf{part}(\psi)$) are used.

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

Definition 5.1. *For a formula ψ, we define $\mathsf{bcl}(\psi) = \mathsf{cl}(\psi) - \mathsf{atom}(\psi)$, representing the Boolean (non-modal) subformulas in the BNF of ψ. The same can be defined for the NNF of ψ as $\mathsf{bsub}(\psi) = \mathsf{sub}(\psi) - \mathsf{part}(\psi)$.*

Definition 5.2. *For lean particle/atom assignments, $\Diamond_{1,i}$ and $\Box_{1,i}$ is the same as full particle/atom assignments. But since the subformula with the modal operator stripped may not be in $\mathsf{atom}(\psi)$ or $\mathsf{part}(\psi)$, we need to redefine the functions $\Diamond_{2,i}$, $\Box_{2,i}$ with the same intuition as for full particle/atom vectors.*

We do so by defining the helper function strip_i inductively as:

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

Obviously, for both lean particle or atom assignments, 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:*

$$\begin{aligned} \Diamond_{2,i}(\vec{y}) &= \begin{cases} \mathsf{strip}_i(\vec{y}) & particles \\ \neg\, \mathsf{strip}_i(\vec{y}) & atoms \end{cases} \\ \Box_{2,i}(\vec{y}) &= \mathsf{strip}_i(\vec{y}) \end{aligned}$$

5.2.3 Level based evaluation

The level-based evaluation approaches is computed in a similar way. Since in the level-based algorithm, we keep an assignment set for each modal level, so going through all the $\overline{\mathsf{bad}_i(X)}$ is no longer necessary. Also since the level based algorithm only requires the assignment set to be consistent w.r.t. to subformulas in a single modal level, we can split the constraint predicate $Consistent$ to $d+1$ sets $Consistent_0$ to $Consistent_d$ where each only consists of constraints related to $\varphi_i \in \mathsf{cl}_i(\psi)$. So for a full/lean atom/particle approach alg, We define $\chi_{\mathsf{level}_i(X)}$ as:

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

Then $\chi_{Initial_i}(\vec{a}) = Consistent_i(\vec{a})$, and $\chi_{Iterate(A,i)}(\vec{a}) = \chi_(Consistent_i)(\vec{a}) \wedge \chi_{\mathsf{level}_i(A)}(\vec{a})$.

The level based evaluation for particles can be implemented in the same way.

5.2.4 Variable Ordering

Performance of BDD-based algorithms is very sensitive to BDD variable order, since it is a primary factor influencing BDD size [Bry86]. Space blowups of of BDDs for the state sets P_i, as well as intermediate BDDs during pre-image operation, is observed in our experiments to be a major factor in performance degradation. Since every step in

the iteration process uses BDDs with variables from different modal depth, dynamic variable ordering is of limited benefit 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 constructing heuristically a good initial variable order. Our heuristic attempts to find a variable order that is appropriate for $\mathcal{KBDD}$. In this we follows the work of Kamhi and Fix, who argued in favor of application-dependent variable order [KF98]. 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 initial variable order to a set of subformulas would be to traverse the DAG for the formula in some order. We used a depth-first, pre-order traversal. This order, however, does not meet the basic principle of BDD variable ordering, which is to keep related variables in close proximity. Our heuristic is aimed at identifying such variables. Note that in our lean representation variables correspond to modal subformulas or atomic subformulas. We found that related variables correspond to subformulas that are related via the sibling or niece relationships. We say that v_x is the *child* of v_y if for the corresponding subformulas we have that $\varphi_x \in \mathsf{sub}_i(\psi)$, $\varphi_y \in \mathsf{sub}_{i+1}(\psi)$, and φ_y is a subformula of φ_x, for some $0 \leq i < md(\psi)$. We say that v_x and v_y are *siblings* if either both φ_x and φ_y are in $\mathsf{sub}_i(\psi)$ or they are both children of another variable v_z. We say that v_y is a *niece* of v_x if there is a variable v_z such that v_z is a sibling of v_x and v_y is a child of v_x. We say that v_x and v_y are *dependent* if they are related via the sibling or the niece relationship. The rationale is that we want to optimize state-set representation for pre-image operations. Keeping siblings close helps in keeping state-set representation compact. Keeping nieces close to their "aunts", helps in keeping intermediate BDDs compact.

We build variable order from the top of the formula down. We start with left-to-right traversal order of top variables in the parse tree of ψ as the order for variables corresponding to subformulas in $\mathsf{sub}_0(\psi)$. Given an order of the variables of modal

depth $< i$, a greedy approach is used to determine the placement of variables at modal depth i. When we insert a new variable v we measure the cumulative distance of v from all variables already in the order that are dependent on v. We find a location for v that minimizes the cumulative distance from other dependent variables. We refer to this approach as the *greedy* approach, as opposed to the *naive* approach of depth-first pre-order.

Chapter 6

Embedding $\mathcal{K}$ with QBF

Both $\mathcal{K}$ and QBF have PSPACE-complete decision problems [Lad77, Sto77]. This implies that the two problems are polynomially reducible to each other. A natural reduction from QBF to $\mathcal{K}$ is described in [HM92]. In the last few years extensive effort was carried out into the development of highly-optimized QBF solvers [GNT01, CSGG99]. One motivation for this effort is the hope of using QBF solvers as generic search engines [Rin99], much is the same way that SAT solvers are being used as generic search engines, cf. [BCCZ99]. This suggests that another approach to $\mathcal{K}$ satisfiability is to find a natural reduction of $\mathcal{K}$ to QBF, and then apply a highly optimized QBF solver. We describe now such a reduction. (A similar approach is suggested in [CSGG99] without providing either details or results.)

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

By Theorem 4.6, A $\mathcal{K}$ formula ψ of modal depth d is satisfiable iff there exists a proper 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 formulas $f_0, f_1, \ldots f_d$ so 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. The intuition is that f_i describes the set P_i. That is, for each $p \subseteq \mathsf{sub}_i(\psi)$, define the truth assignment τ_p^i as follows:

$\tau_p^i(x_{\varphi,i}) = 1$ iff $\varphi \in p$. The intention is to have $P_i = \{p \subseteq \mathsf{sub}_i(\psi) | \tau_p^i \models f_i\}$. We then say that f_i *characterizes* P_i.

In the following, we define $\mathsf{particle}_i(\psi)$ as 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 take f_d to be lc_d. Indeed, we have $P_d = \mathsf{particle}_d(\psi) = \{p \subseteq \mathsf{sub}_d(\psi) | \tau_p^d \models f_d\}$. Thus, f_d characterizes $Initial_d(\psi)$.

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

Lemma 6.1. *If f_{i+1}' characterizes P_{i+1}, then f_i' characterizes $P_i = Iterate(P_{i+1}, i)$.*

Proof. By construction, lc_i characterizes $\mathsf{part}_i(\psi)$. 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 $R_i(p, p')$. So the lemma holds.

Corollary 6.1. *ψ is satisfiable iff $\exists x_{\theta,0:\{\theta \in \mathsf{sub}_0(\psi)\}} x_{\psi,0} \wedge f_0'$ is satisfiable.*

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

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

We can now write f_i in a compact fashion:

$$lc_i \wedge \forall y_1, \ldots, \forall y_m : \exists x_{\theta,i+1:\{\theta \in \mathsf{sub}_{i+1}(\psi)\}} : witness_i \rightarrow$$
$$\left(f_{i+1} \wedge tr_i \wedge \bigwedge_{j=0}^{k-1} ((val(\mathbf{y}) = j \wedge x_{\xi_j^i,i}) \rightarrow x_{\mathsf{strip}(\xi_j^i),i+1}) \right).$$

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 $\Diamond$ subformulas in $\mathsf{sub}_i(\psi)$. We then check if $witness_i$ holds, in which case we assert the existence of the witnessing particle. We use f_{i+1} to ensure that this particle is in P_{i+1} and tr_i to ensure satisfaction of $\Box$ subformulas. Finally, we let $val(\mathbf{y})$ point to the $\Diamond$ subformulas that needs to be witnesses. Note that f_i contains only one copy of f_{i+1}.

Lemma 6.2. *If f_{i+1} characterizes P_{i+1}, then f_i characterizes $P_i = Iterate(P_{i+1}, i)$.*

Corollary 6.2. *ψ is satisfiable iff $\exists x_{\theta,0:\{\theta \in \mathsf{sub}_0(\psi)\}} x_{\psi,0} \wedge f_0$ is satisfiable.*

Proof. The claim follows from the fact that f_i is logically equivalent to f_i'.

We implemented this approach by optimizing the translation further. As in the BDD-based implementation, we represent only Boolean literals, $\Box$ subformulas and

$\diamond$ subformulas with Boolean variables. The other subformulas are not represented explicitly, but are logically implied.

Chapter 7

Results

We implemented the BDD-based decision procedure in C++ using the CUDD 2.3.1 [Som98] 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 [Tac99]. In the following, we describe and compare the performance of the different algorithms.[1]

As benchmarks, we use both the $\mathcal{K}$ part of TANCS 98 [HS96] and the MODAL PSPACE division of TANCS 2000 [MD00], as well as random formulas generated with [PSS01].

We present the result in two parts. First, using TANCS 98 and TANCS 2000, we study the influence of each optimization technique and the influence of variable ordering to determine the best configuration for $\mathcal{KBDD}$. Then, we provide a comparison of $\mathcal{KBDD}$ with other solvers across a spectrum of different benchmarks .

For most comparison, we set the time out at 1000s and the space limit for BDDs at 384MB. To avoid getting into overwhelming details in the comparison of solvers and to present a global view of performance, we use the presentation technique suggested in [SS01], where we plot the number of cases solved against the running time used. The chart is scaled so the full scale is the total number of cases in the benchmark. Thus, the solver with a higher curve is faster than one with a lower curve.

[1] All the tests run on a Pentium 4 1.7GHz with 512MB of RAM, running linux kernel version 2.4.2. The solver is compiled with gcc 2.96 with parts in OCaml 3.04.

7.1 Comparison in depth

To analyze the usefulness of each optimization techniques used, we run the algorithm with different optimization configurations on the $\mathcal{K}$ part of TANCS 98 and TANCS 2000 benchmark suites[2] , both scalable benchmarks which contains both provable and non-provable formulas. In TANCS 98, simple formulas have their complexity increased by re-encoding them with superfluous sub-formulas. In TANCS 2000, formulas are constructed by translating QBF formulas into $\mathcal{K}$ using three translation schemes, namely Schmidt-Schauss-Smolka translation, which gives easy formulas, Ladner translation, which gives medium difficulty formulas, and Halpern translation, which gives hard formulas.

7.1.1 The basic algorithms

To compare our approaches, we benchmark the basic algorithms on TANCS 98. The results are presented in Fig. 7.1. We can see that *SAT clearly outperforms our two basic algorithms. An explanation of this "weak" behavior of our approaches is that the intermediate results of the pre-image operation are so large that the BDDs space constraint is usually reached. The difference between top-down and bottom-up approaches is minor. Top-down slightly outperforms bottom-up since in a BDD-based implementation, 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.

7.1.2 Particle approaches

Now we compare the variants using types with their full particle-based variants. The results are presented in Fig. 7.2. We can see that, for TANCS 98, the particle approach slightly outperforms the type approach. Most of the improvements come from

[2]We used TANCS 98 in cases where too few cases in TANCS 2000 complete under an unoptimized scheme, allowing better comparison.

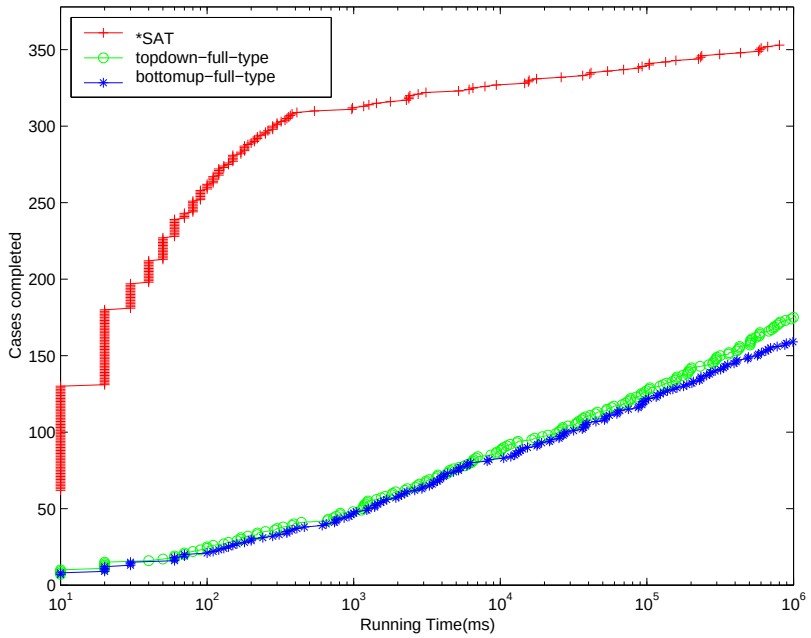


Figure 7.1 : Performance on TANCS 98 (basic approaches)

the use of negation normal form, which allows us to distinguish between diamonds and boxes, resulting in the reduction of the image operations needed.

7.1.3 Lean vector approaches

Next, for types and particles, bottom-up and top-down, we compare the "full" approaches with their lean variants (see Fig. 7.3 and Fig. 7.4). Intuitively, the full variants trade a larger number of BDD variables in the representation of the transition relation for simpler consistency constraints. On TANCS 98, we can see that the lean approaches outperform in each combination their full variants. This shows that, as a general guideline, we should always attempt to reduce the number of BDD variables, since this results in smaller BDDs. Indeed, experience in symbolic model checking suggests that BDD size is typically the dominant factor when evaluating the performance of BDD-based algorithms [KFB98].

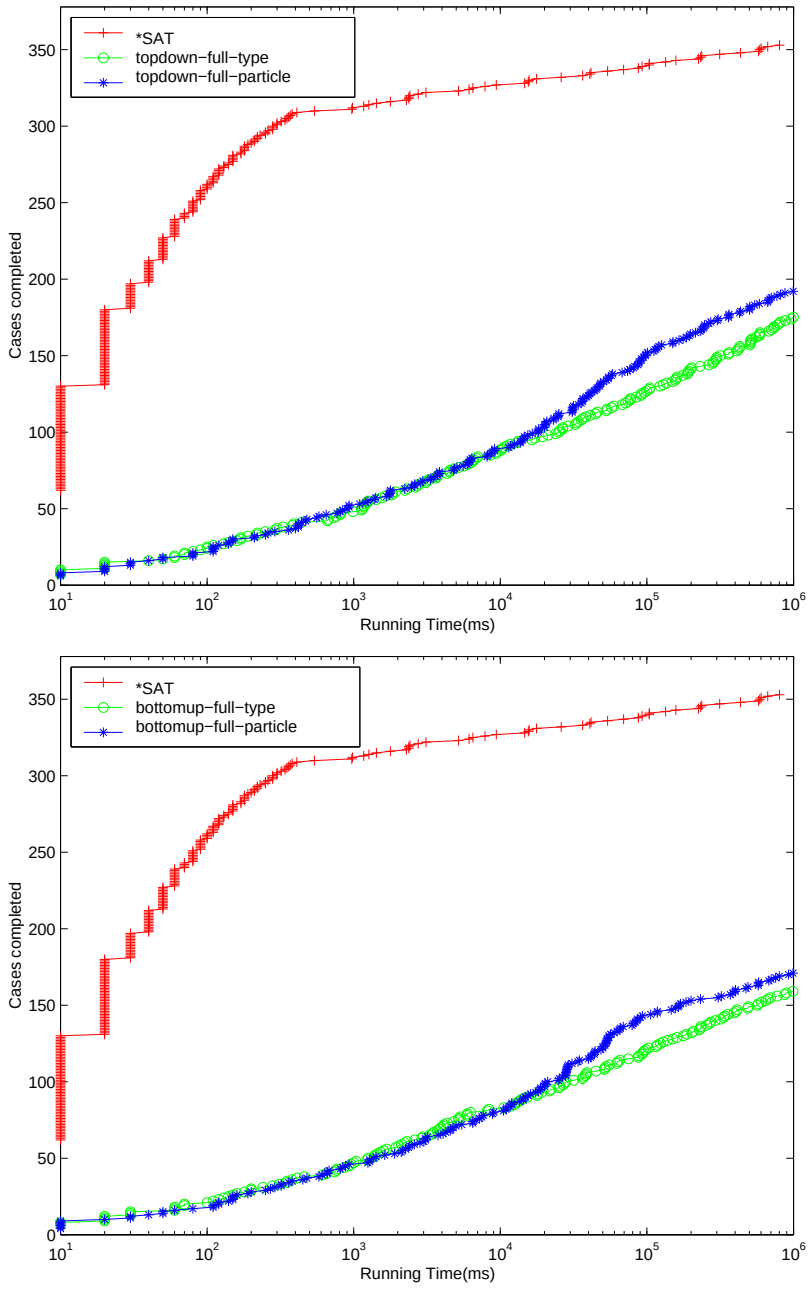


Figure 7.2 : Performance on TANCS 98 (particles vs. types)

7.1.4 Level based evaluation

Next, we compared the level-based approach with the top-down and the bottom-up approach. It turns out that the level-based approach outperforms both, and that,

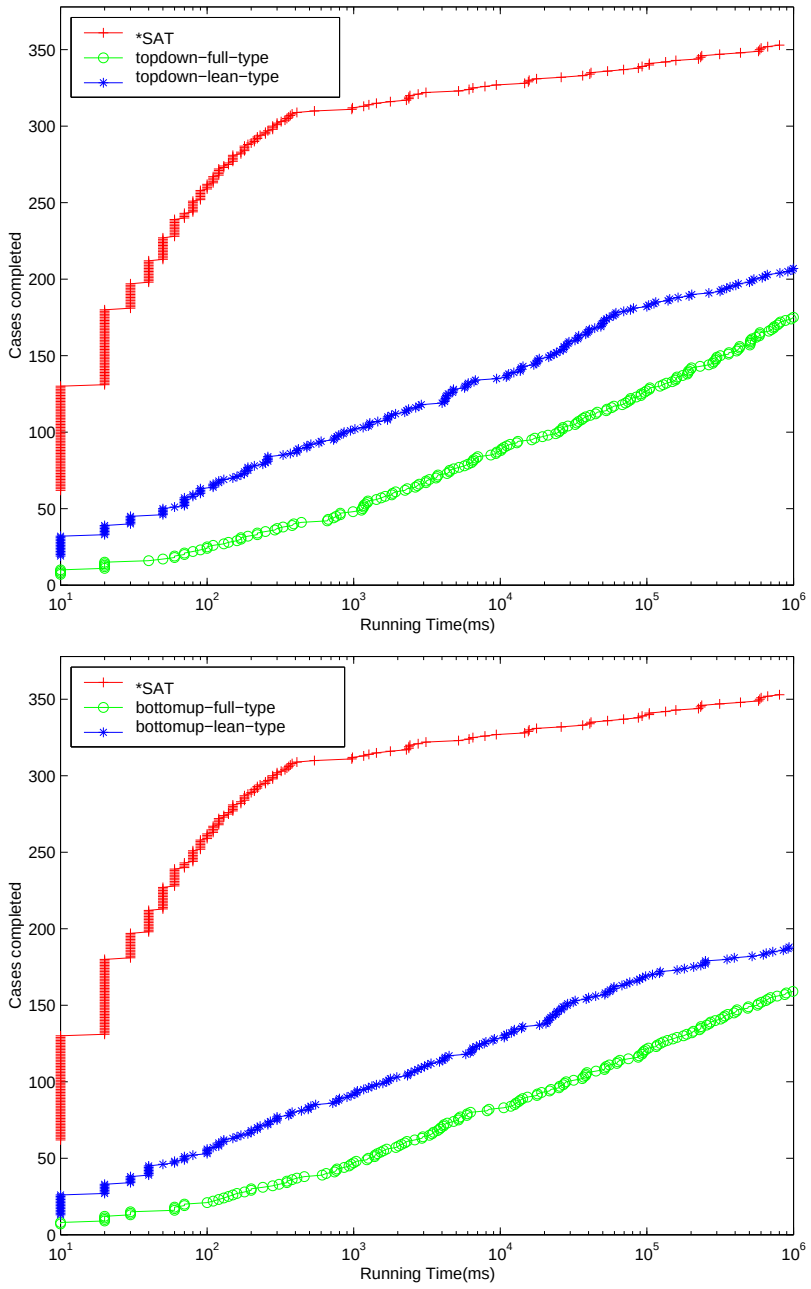


Figure 7.3 : Performance on TANCS 98 lean vs. full types

both for types and particles, the lean approach again outperforms the full one, see Fig. 7.5. By taking advantage of $\mathcal{K}$'s layered model property, we can split various space-consuming BDDs into smaller ones based on the modal depth of the corresponding

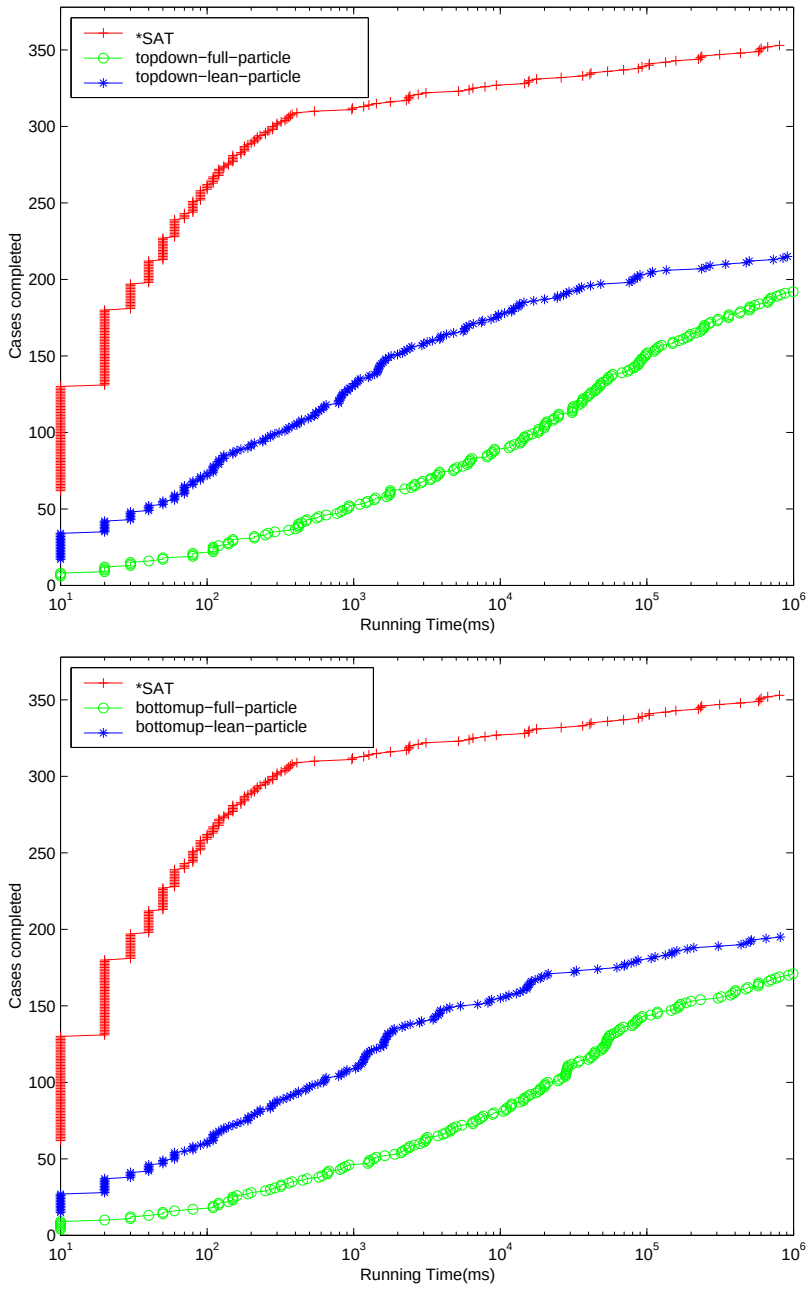


Figure 7.4 : Performance on TANCS 98 lean vs. full particles

sub-formulas. This minimizes space-outs and improves running time. The associated reduction in number of pre-image operations is also substantial for most formulas. In the following, $\mathcal{KBDD}$ would refer to the level-based lean particle version of the algorithm.

7.1.5 Variable ordering and formula simplification

To demonstrate the effects of variable ordering and formula simplification, we tested $\mathcal{KBDD}$ with both naive and greedy variable ordering, and with and without formula simplification, using TANCS 2000 easy and medium formulas [MD00][3] ($\mathcal{KBDD}$ without formula simplification cannot handle the hard formulas of TANCS 2000). The results are in Figure 7.6.

We see in Figure 7.6 that formula simplification yields a significant performance improvement. This improvements was observed for different types of formulas and different variable-ordering algorithms. In particular, $\mathcal{KBDD}$ was able to avoid space outs in many cases. We can also see that greedy variable ordering is useful in conjunction with formulas simplification, improving the number of completed cases and sometimes running time as well. Without formula simplification, the results for greedy variable ordering are not consistent, as overhead of finding the variable order may offset any advantages of applying it. The combination of formula simplification and greedy variable ordering clearly improves the performance of $\mathcal{KBDD}$ in a significant way. In the next section, we compare the performance of optimized $\mathcal{KBDD}$ against three other solvers.

7.2 Comparison between solvers

To assess the effectiveness of BDD-based decision procedures for $\mathcal{K}$, we compared the optimized $\mathcal{KBDD}$ against three solvers: (1) DLP is a tableau-based solver [PSH99],

[3] See `http://www.dis.uniroma1.it/~tancs/`.

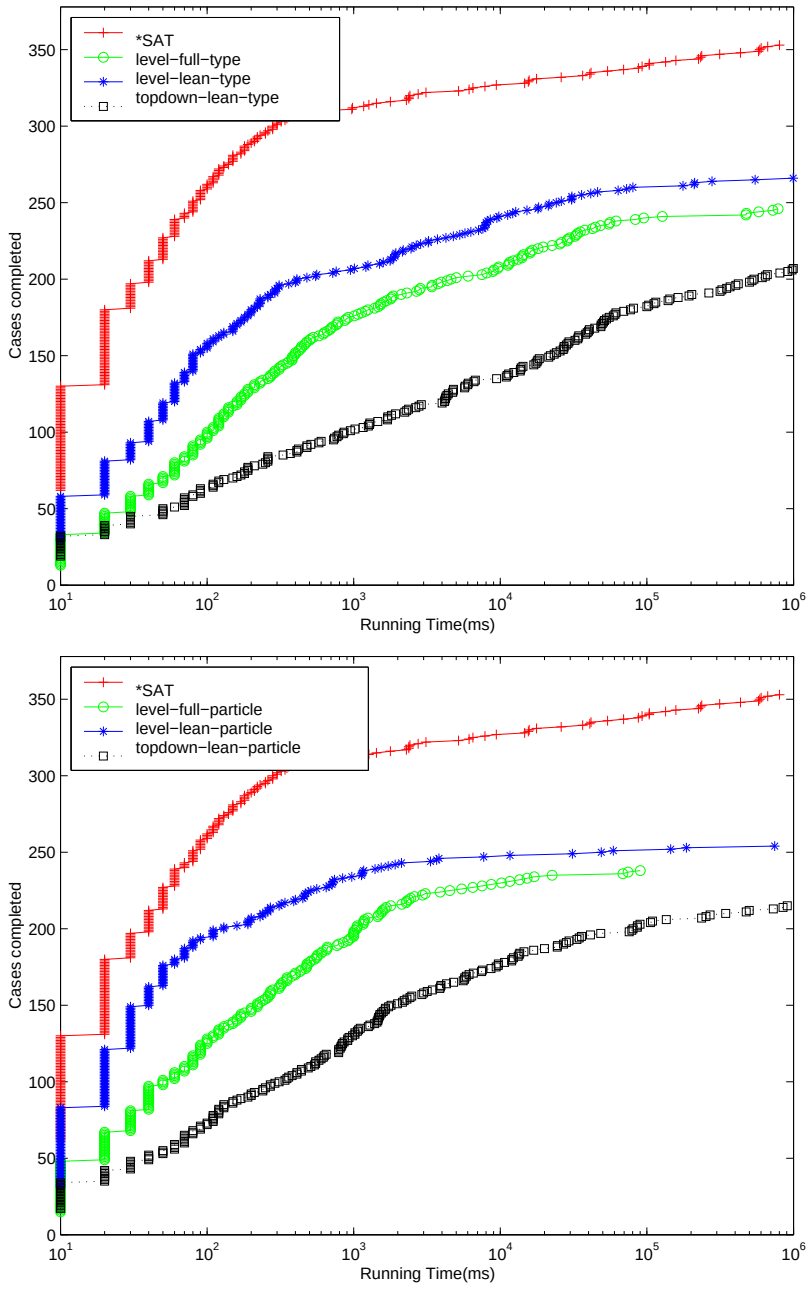
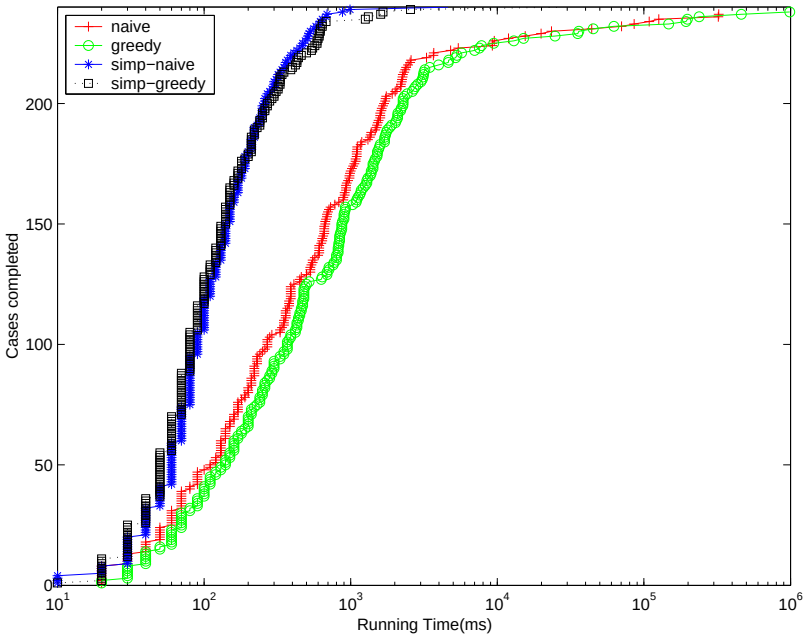
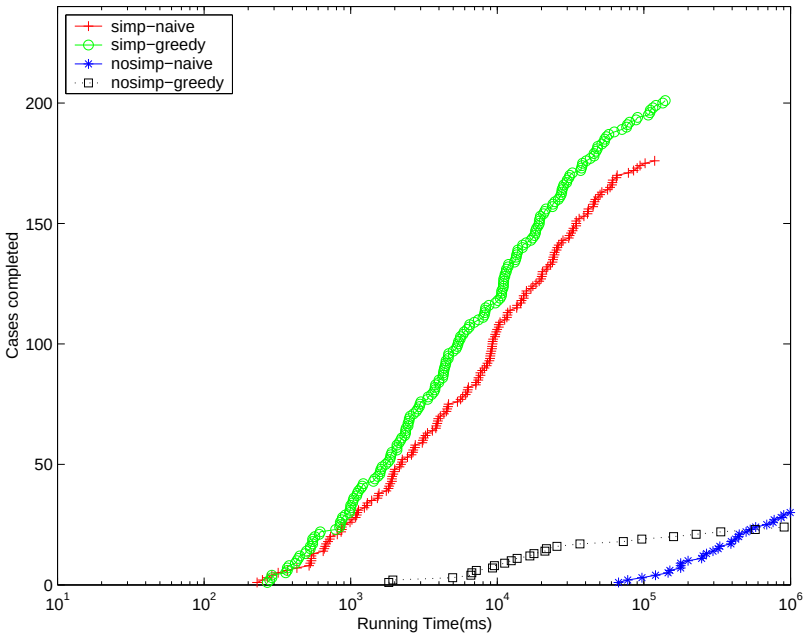


Figure 7.5 : Performance on TANCS 98 (level-based evaluation)

(2) MSPASS is a resolution-based solver, apply to a translation of modal formulas to



TANCS 2000-easy (cnfSSS)



TANCS 2000-medium (cnfLadn)

Figure 7.6 : Optimizations on TANCS 2000

first-order formulas [HS00][4], (3) We developed also a reduction of $\mathcal{K}$ to QBF (which is of independent interest), and applied QuBE, which is a highly optimized QBF solver [GNT01]. For a fair comparison, we checked first whether our formula-simplification optimization is useful for these solvers, and used it when it was (DLP and QuBE).

In addition to TANCS 98 and TANCS 2000, we also use randomly generated formulas, as suggested in [PSS01]. This scheme generates random modal-CNF formulas 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 7.7 and Figure 7.8 we see that on the TANCS 98 benchmarks, DLP has the best performance, but on the more challenging TANCS 2000 benchmarks, $\mathcal{KBDD}$ outperformed the other solvers, especially on the harder portions of the suite (the hard formulas of TANCS 2000 required dynamic variable reordering). MSPASS was a distant third, especially on the harder formulas, and is omitted on the hard formulas of TANCS 2000[5]. It is also clear that reducing $\mathcal{K}$ satisfiability to a search-based QBF solver is not a viable approach; it was dominated all other approaches and solved

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

[5]Better results for MSPASS is possible if different parameters is used for different cases. We did not take this approach because it is outside the scopt of this thesis.

only a small fraction of the benchmark formulas in TANCS 98. (For TANCS 2000 this approach was so ineffective that we did not report the results.) It would be interesting to try the reduction-to-QBF approach with another type of QBF solver, e.g., a resolution-based QBF solver [BKF95].

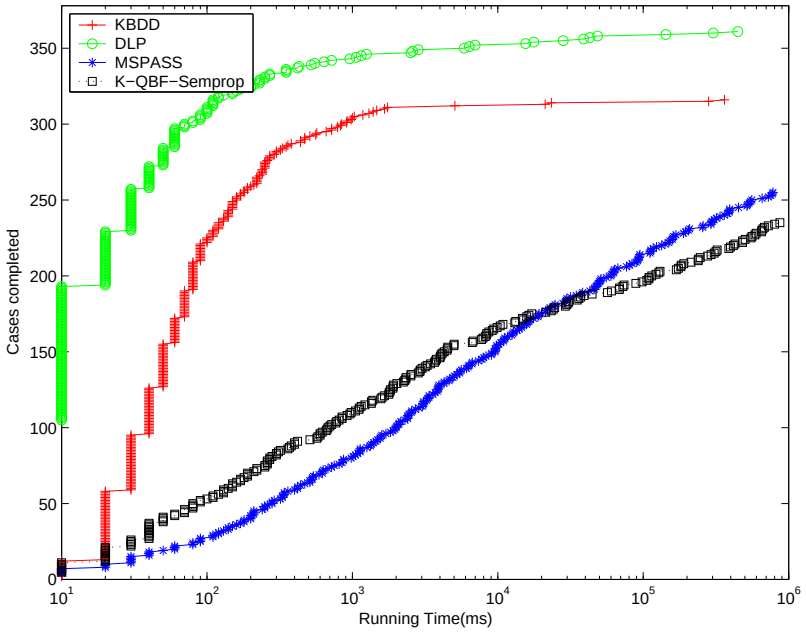
7.2.2 Results on random modal CNF formulas

A different perspective on the comparison between DLP, a search-based solver, and $\mathcal{KBDD}$, a symbolic solver, is demonstrated on random modal-CNF formulas. The generation of the formulas are as suggested in [PSS01]. This scheme generates random modal-CNF formulas 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 was fixed and the propositional complexity of the formula was varied by increasing the *density* L/N.

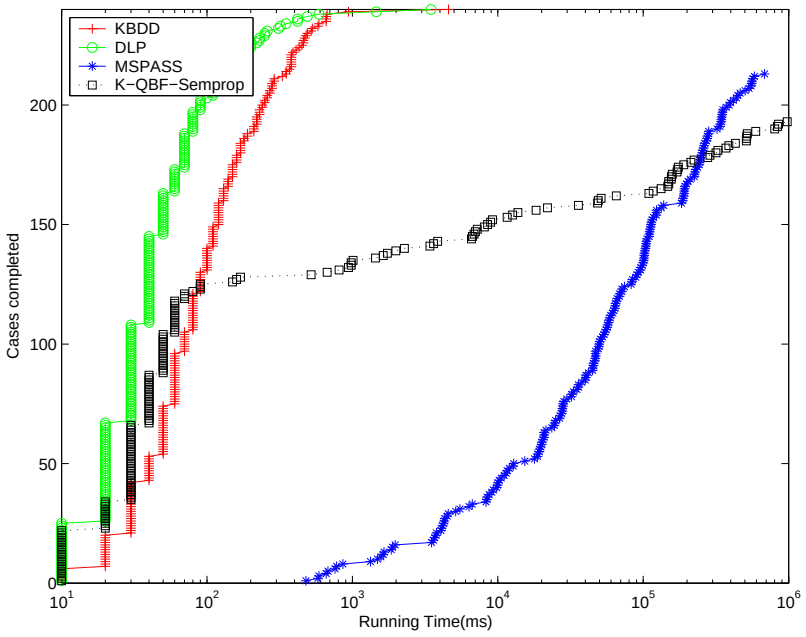
We plot here median running time (16 samples per data point) as a function of density (L/N) to demonstrate the difference between the behavior of the two solvers.

As we can see in figure 7.9, for $d = 1$, DLP demonstrates the bell-shaped "easy-hard-easy" pattern that is familiar from random propositional CNF formulas [SML96] and random QBF formulas [GW99]. In contrast, for $\mathcal{KBDD}$ we see an increase in running time as a function of the density; that is, the higher the density the harder the problem for $\mathcal{KBDD}$. This is consistent with known results on the performance of BDD-based algorithm for random propositional CNF formulas [CDS$^+$00]. For each modal level, $\mathcal{KBDD}$ builds a BDD for the appropriate particle set. With increased density, the construction of these BDDs gets quite challenging, often resulting in space outs or requiring extensive variable reordering. (In the propositional case, one

can develop algorithms that avoid the construction of a monolithic BDD, cf. [SV01]. It would be interesting to try to apply such ideas for $\mathcal{KBDD}$.) This explains why DLP performs much better than $\mathcal{KBDD}$ on random modal-CNF formulas. Unlike the benchmark formulas of TANCS 98 and TANCS 2000, the random modal-CNF formulas have a very high propositional complexity (low modal depth). In contrast, the formulas in TANCS 98 and TANCS 2000 have high modal complexity (high modal depth). Our conclusion is that DLP is better suited for formulas with high propositional complexity, while $\mathcal{KBDD}$ is better suited for formulas with high modal complexity.

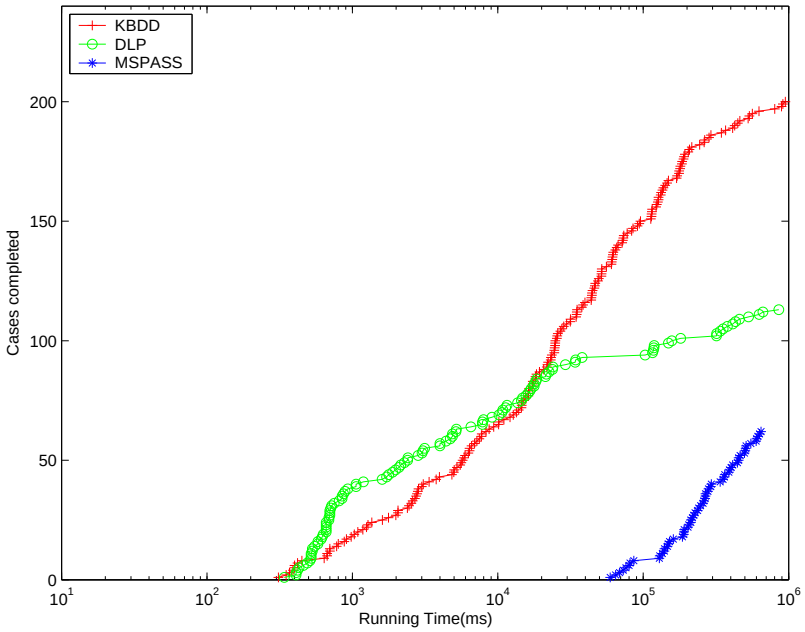


TANCS 98

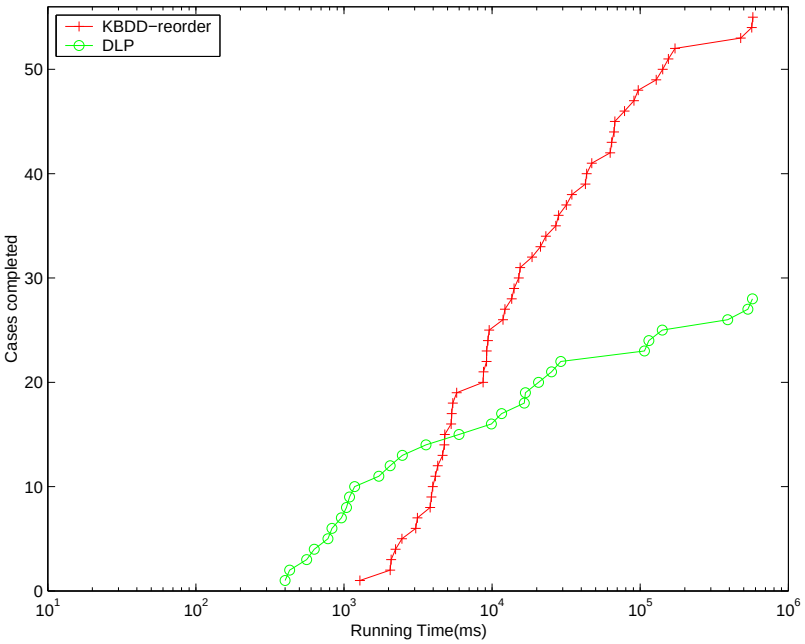


TANCS 2000 Easy (cnfSSS)

Figure 7.7 : Comparison of $\mathcal{KBDD}$, DLP, QuBE/QBF and MSPASS on $\mathcal{K}$ formulas (part 1)

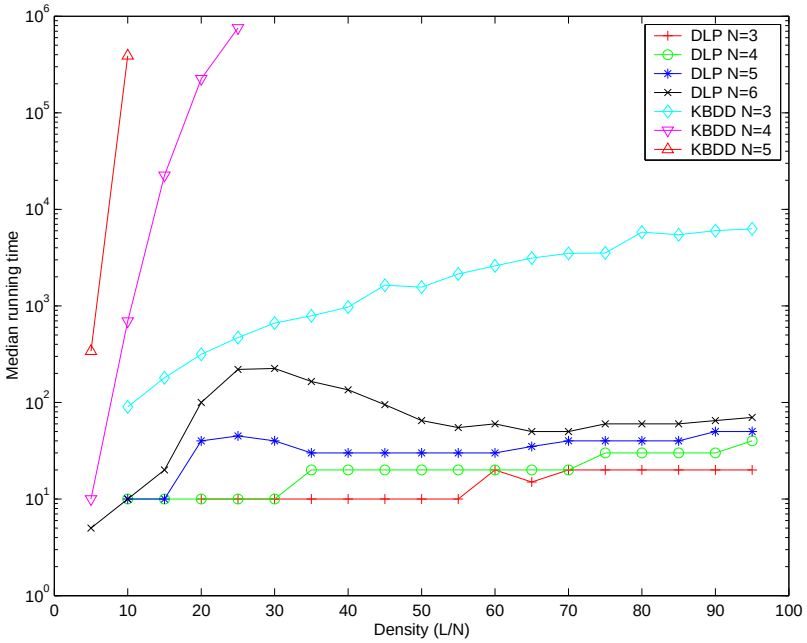


TANCS 2000 Medium (cnfLadn)

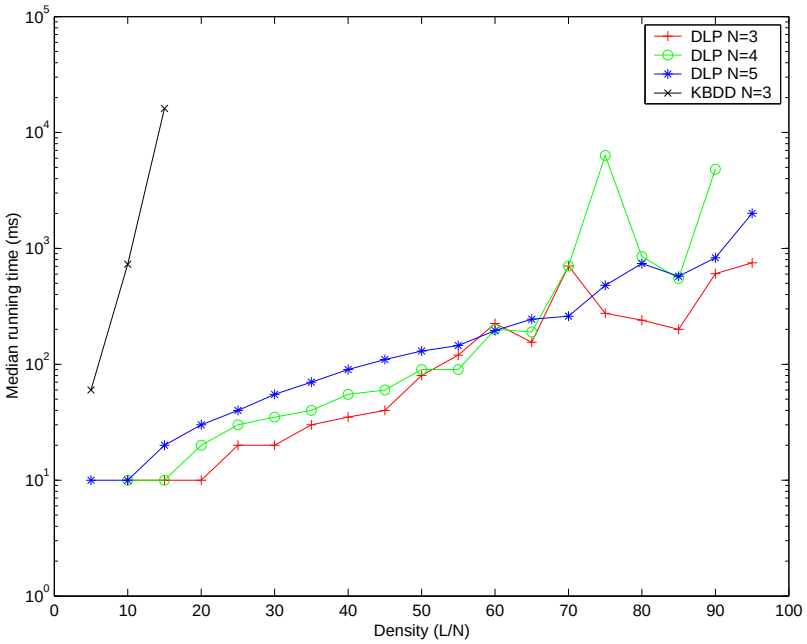


TANCS 2000 Hard (cnf)

Figure 7.8 : Comparison of $\mathcal{KBDD}$, DLP, QuBE/QBF and MSPASS on $\mathcal{K}$ formulas (part 2)



$d = 1$



$d = 2$

Figure 7.9 : Comparison of DLP and $\mathcal{KBDD}$ on Random formulas

Chapter 8

Conclusions

We described here BDD-based decision procedures for $\mathcal{K}$. Our approach is inspired by the automata-theoretic approach, but we avoid explicit automata construction. We explored a variety of optimization techniques and concluded that, in general, it is preferred to work with looser constraints; in general, we got the best performance with lean particles. We also showed that it is necessary to use a level-based approach to obtain a competitive implementation. Formula preprocessing by removing pure literals and propogating the effects by syntactical simplification, though not specialized to our method in particular, is also important for improving performance. We also attempted to optimize the implementation by applying BDD-centric techniques like clustering with early quantification and initial variable ordering.

Our results show that the payoff of the variable-ordering optimization is rather modest, while the payoff of the pure-literal optimization is quite significant. We benchmarked $\mathcal{KBDD}$, our optimized solver, against both native solvers (DLP) and translation-based solvers (MSPASS and QuBE). Our results indicate that the BDD-based approach dominates for modally heavy formulas, while search-based approaches dominate for propositionally heavy formulas.

One way to look at the results is that the $\mathcal{KBDD}$ approach, by using a more powerful underlying solver (BDDs vs. satisfiability) allows the use of a simpler decision procedure. Instead of requiring exponential number of calls to a propositional satisfiability procedure, we only required a polynomial number of calls to BDD operations. The question would of course be, is such an trade off reasonable. We know that the complexity of BDD operations are 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 would be acceptable.

Another explanation would be we traded modal complexity for propositional complexity. This way, we managed to solve a large amount of problems which have "big" models, which cause problems with SAT based solvers. We suggest that a comparison of BDD and SAT based $\mathcal{K}$ solver would be like in table 8.1.seem in the following table:

	Propositionally sparse	Propositionally dense
Big model	BDD better	neither work good
Small Model	Both work good SAT slightly faster	neither work good SAT could be given more time

Table 8.1 : A hypothetical comparison of BDD vs. SAT based solvers

Although our goal is not to develop the "fastest $\mathcal{K}$ solver", the $\mathcal{KBDD}$ approach is very competitive for most benchmarks. With all the optimization schemes, we obtained very good results with current structured benchmark suites. Further research is required to quantify the distinction between propositionally heavy and modally heavy formulas. This might enable the development of a combined solver, which invokes the appropriate engine for the formula under test. Another approach would be to develop a a hybrid solver, combining BDD-based and search-based techniques (cf. [GYA$^+$01] for a hybrid approach in model checking), which would perform well on both modally heavy and propositionally heavy formulas. We leave this for future research.

Bibliography

[AGHd00] C. Areces, R. Gennari, J. Heguiabehere, and Maarten de Rijke. Tree-based heuristics in modal theorem proving. In *Proc. of the ECAI'2000*, 2000.

[And98] H.R. Andersen. An introduction to binary decision diagrams. Technical report, Department of Information Technology, Technical University of Denmark, 1998.

[BAN88] M. Burrows, M. Abadi, and R. Needham. Authetication: a practical study in belief and action. In *Proc. 2nd Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 325–342, 1988.

[BBG+94] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In *Proc. 6th Conf. on CAV*, volume 818 of *LNCS*, pages 182–193, Stanford, June 1994.

[BCCZ99] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Analysis and Construction of Systems*, volume 1579 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[BCL91] J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. In *Int. Conf. on VLSI*, pages 49–58, 1991.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[BdV01] P. Blackburn, M. de Rijke, and Y. Venema. *Modal logic*. Camb. Univ. Press, 2001.

[BKF95] H.K. Buning, M. Karpinski, and A. Flogel. Resolution for quantified Boolean formulas. *Information and Computation*, 117(1):12–18, 1995.

[BLMS94] R. Brafman, J.-C. Latombe, Y. Moses, and Y. Shoham. Knowledge as a tool in motion planning under uncertainty. In R. Fagin, editor, *Theoretical Aspects of Reasoning about Knowledge: Proc. Fifth Conference*, pages 208–224. Morgan Kaufmann, San Francisco, Calif., 1994.

[Boc82] G. V. Bochmann. Hardware specification with temporal logic: an example. *IEEE Transactions on Computers*, C-31:223–231, 1982.

[Bry86] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, Vol. C-35(8):677–691, August 1986.

[BT01] Franz Baader and Stephan Tobies. The inverse method implements the automata approach for modal satisfiability. Technical Report LTCS-Report 01-03, Research group for theoretical computer science, Aachen university of Technology, 2001.

[CCF82] J. M. V. Castilho, M. A. Casanova, and A. L. Furtado. A temporal framework for database specification. In *Proc. 8th Int. Conf. on Very Large Data Bases*, pages 280–291, 1982.

[CCGR00] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: A new symbolic model checker. *Int. J. on Software Tools for Tech. Transfer*, 2(4):410–425, 2000.

[CDS+00] C. Coarfa, D.D. Demopoulos, A. San Miguel Aguirre, D. Subramanian, and M.Y. Vardi. Random 3-SAT: The plot thickens. In *Proc. of the International Conference on Constraint Programming*, 2000.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[CSGG99] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified Boolean formulae and its experimental evaluation. Technical report, Dipartmento di Imformatica e Sistemistica, Universita de Roma, 1999.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5:394–397, 1962.

[EH00] K. Etessami and G.J. Holzmann. Optimizing Büchi automata. In *CONCUR 2000*, pages 153–167, 2000.

[GB94] D. Geist and H. Beer. Efficient model checking by automated ordering of transition relation partitions. In *Proc. of the sixth Int. Conf. on CAV*, pages 299–310, 1994.

[GNT01] E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE, a system for deciding quantified Boolean formulae satisfiability. In *IJCAR'01*, 2001.

[GS00] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedure - the case study of modal K(m). *Infomation and Computation*, 162:158–178, 2000.

[GW99] E. Graedel and I. Walukiewicz. Guarded fixed point logic. In *Proc. 14th Symp. on Logic in Computer Science*, July 1999.

[GYA+01] A. Gupta, Z. Yang, P. Ashar, L. Zhang, and S. Malik. Partition-based decision heuristics for image computation using SAT and BDDs. In *ICCAD 2001*, pages 286–292, 2001.

[HM90] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990. A preliminary version appeared in *Proc. 3rd ACM Symposium on Principles of Distributed Computing*, 1984.

[HM92] J.Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

[HS96] A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. Technical report, Universität Bern, Switzerland, 1996.

[HS00] U. Hustadt and R. Schmidt. MSPASS: modal reasoning by translation and first order resolution. In *Proc. of TABLEAUX 2000*, pages 67–71, 2000.

[KF98] G. Kamhi and L. Fix. Adaptive variable reordering for symbolic model checking. In *ICCAD 1998*, pages 359–365, 1998.

[KFB98] G. Kamhi, L. Fix, and Z. Binyamini. Symbolic model checking visualization. In *Formal Methods in Computer-Aided Design, Second International Conference FMCAD'98*, volume 1522 of *LNCS*, pages 290–303. Springer-Verlag, November 1998.

[Lad77] R.E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput.*, 6(3):467–480, 1977.

[Lip77] W. Lipski. On the logic of incomplete information. In *Proc. 6th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol. 53, pages 374–381. Springer-Verlag, Berlin/New York, 1977.

[LL59] C. Lewis and C. Langford. *Symbolic Logic*. Dover Publications, New York, 1959.

[MD00] F. Massacci and F.M. Donini. Design and results of TANCS-2000. In *Proc. of TABLEAUX 2000*, pages 52–56, 2000.

[MH69] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In D. Michie, editor, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. on Foundation of Computer Science*, pages 46–57, 1977.

[Pra76] V. R. Pratt. Semantical considerations on Floyd-Hoare logic. In *Proc. 17th IEEE Symp. on Foundations of Computer Science*, pages 109–121, 1976.

[Pra80] V.R. Pratt. A near-optimal method for reasoning about action. *Journal of Computer and System Sciences*, 20(2):231–254, 1980.

[PSH99] P.F. Patel-Schneider and I. Horrocks. DLP and FaCT. In *Analytic Tableaux and Related Methods*, pages 19–23, 1999.

[PSS01] P.F. Patel-Schneider and R. Sebastiani. A new system and methodology for generating random modal formulae. In *IJCAR 2001*, pages 464–468, 2001.

[RAB+95] R. Ranjan, A. Aziz, R. Brayton, B. Plessier, and C. Pixley. Efficient BDD algorithms for FSM synthesis and verification. In *Proc. of IEEE/ACM International Workshop on Logic Synthesis*, 1995.

[Rin99] J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[RS83] J. H. Reif and A. P. Sistla. A multiprocessor network logic with temporal and spatial modalities. In *Proc. 12th International Colloq. on Automata, Languages, and Programming*, Lecture Notes in Computer Science, Vol. 104. Springer-Verlag, Berlin/New York, 1983.

[Rud93] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *ICCAD'93*, pages 42–47, 1993.

[SB00] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *CAV 2000*, pages 247–263, 2000.

[SML96] B. Selman, D.G. Mitchell, and H.J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.

[Som98] F. Somenzi. CUDD: CU decision diagram package, 1998.

[SS01] G. Sutcliffe and C. Suttner. Evaluating general purpose automated theorem proving systems. *Artificial intelligence*, 131:39–54, 2001.

[Sto77] L.J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[SV01] A. San Miguel Aguirre and M.Y. Vardi. Random 3-SAT and BDDs: The plot thickens further. In *CP01*, 2001.

[Tac99] A. Tacchella. *SAT system description. In *Collected Papers from the International Description Logics Workshop (DL'99). CEUR*, 1999.

[THY93] S. Tani, K. Hamaguchi, and S. Yajima. The complexity of the optimal variable ordering problems of shared binary decision diagrams. In *ISAAC: 4th International Symposium on Algorithms and Computation*, 1993.

[Var97] M.Y. Vardi. What makes modal logic so robustly decidable? In N. Immerman and Ph.G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, pages 149–183. American Mathematical Society, 1997.

[Vor01] Andrei Voronkov. How to optimize proof-search in modal logics: new methods of proving redundancy criteria for sequent calculi. *Computational Logic*, 2(2):182–215, 2001.